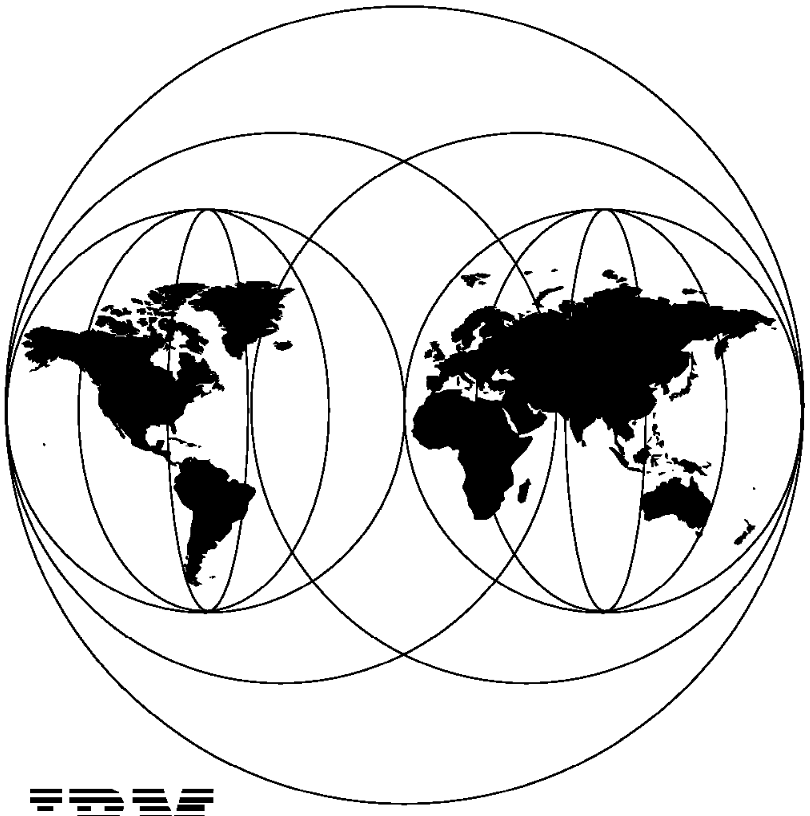
International Technical Support Organization SG24-4381-00

Learning Practical TCP/IP for AIX V3.2/V4.1 Users: Hints and Tips for Debugging and Tuning

May 1996



International Technical Support Organization
Raleigh Center

International Technical Support Organization

SG24-4381-00

Learning Practical TCP/IP for AIX V3.2/V4.1 Users: Hints and Tips for Debugging and Tuning

May 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page xv.

First Edition (May 1996)

This edition applies to the RISC System/6000 for use with the AIX Operating System Version 3.2 and 4.1.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader′s feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

Abstract

This redbook is intended to provide practical knowledge and useful information on the AIX V3.2 and V4.1 System TCP/IP networking function.

In a UNIX environment, Transmission Control Protocol/Internet Protocol (TCP/IP) networking is almost a mandatory feature. Many functions and operations are common between UNIX environments, but still there are unique, machine-dependent considerations. This document helps the reader to recognize and understand AIX- and RISC System/6000-unique features and pitfalls.

This book was written for customers, IBM technical professionals, and third-party professionals concerned with TCP/IP operation of a RISC System/6000. Some knowledge of TCP/IP, AIX, and RISC System/6000 is assumed.

(363 pages)

Contents

Figures

Tables

Special Notices

ADD	Applications Design and Development, Incorporated
Apollo	Apollo Computer, Incorporated
BSD UNIX	AT&T Bell Laboratories Incorporated
Cisco	Cisco Systems, Incorporated
DCE	The Open Software Foundation
DEC	Digital Equipment Corporation
HP	Hewlett-Packard Company
NCS	Apollo Computer, Incorporated
Network Computing System	Apollo Computer, Incorporated
Network File System	Sun Microsystems, Incorporated
NFS	Sun Microsystems Incorporated
ODM	Optical Disk Mastering, Incorporated
ONC	Sun Microsystems, Incorporated
OSF	Open Software Foundation, Incorporated
Portmapper	Sun Microsystems, Incorporated
POSIX	Institute of Electrical and Electronic Engineers
SCSI	Security Control Systems, Incorporated
Sun	Sun Microsystems, Incorporated
Sun Microsystems	Sun Microsystems, Incorporated
X-Windows	Massachusetts Institute of Technology
Xerox	Xerox Corporation
3Com	3Com Corporation
486	Intel Corporation

Preface

This redbook is intended to provide hints and tips for TCP/IP troubleshooting and performance tuning of the AIX Version 3.2/4.1 and RISC System/6000. This book outlines our experiences of working with the AIX TCP/IP. This book covers various subjects regarding TCP/IP and AIX and supplemental information which may not be covered by AIX manuals. Current commercial books are available for the same purpose, but they are written about UNIX and TCP/IP, and of course, are also useful for AIX users. This book is more focused on AIX and RISC System/6000 implementation and doesn′t cover UNIX and TCP/IP basics.

This book is intended for system/network administrators, or planners, who use TCP/IP, which helps AIX run successfully.

Versions of AIX in this book

Although this book is written mainly for AIX V4.1, many portions of this book are valid for AIX V3.2. We added remarks where the difference between V3.2 and V4.1 is important.

Purpose of This Book

This book is for “bare hands” debugging and tuning. Today many sophisticated devices are commercially available; not only hardware but also software have been developed to contribute to solving network problems. We can troubleshoot and debug many problems more easily and efficiently with these devices, but our concern is that not all people or all sites have such tools or environments.

In this book, we intentionally avoid describing any tools or utilities not available on the regular AIX operating system. We only refer to the AIX commands and tools available on AIX V4.1, and you will soon learn what we can do with those basic facilities. In order to use those tools efficiently, we explain not only “how” but also “why.” We show you how you can do troubleshooting and performance tuning with “bare hands.”

How This Document Is Organized

The document is organized as follows:

- Chapter 1, “TCP/IP Configuration for AIX V4.1”

  This chapter provides TCP/IP configuration procedure and configuration hints and tips. Where and how the configuration parameters are stored and their relation to boot process are also explained. The objective of this chapter is to give the readers knowledge and understanding in order to debug configuration problems. ODM, an AIX-unique feature, is also explained. After reading this chapter, readers will be able to fix configuration problems.

- Chapter 2, “Debugging TCP/IP Troubles”

  This chapter provides practical procedures to debug TCP/IP connectivity problems. Debugging procedures are described for each network layer, which is based on the OSI seven-layer network model. The objective of this chapter is to give you the knowledge and understanding on how to isolate a problem. Some tools (commands) useful for debugging are explained in detail, using trace examples. AIX′s unique feature, the System Resource Controller (SRC), is also explained. After reading this chapter, you will be able to identify what layer has a problem.

- Chapter 3, “Getting Information for Performance Tuning”

  This chapter describes how to get necessary data for performance tuning. When your system has a performance problem, the first step you should take is to gather data and figure out the bottleneck or cause. Here we explain a netstat command. This command has flags for each protocol and gives you many statistic counters. We show you which counters are important and what symptoms you have to check. Also, some original commands are introduced. They provide supplemental statistic data that the netstat command cannot give you. After reading this chapter, you will be able to gather data and analyze it.

- Chapter 4, “System Parameter Tuning”

  This chapter describes how to tune system parameters. In this chapter we treat AIX system parameters that impact TCP/IP network performance. After reading this chapter, you will be able to adjust some system parameters and improve TCP/IP performance.

- Chapter 5, “TCP/IP Related Parameter Tuning”

  This chapter explains how to tune TCP/IP parameters. In this chapter we describe TCP/IP parameters in detail. Many IP trace examples are used. One of the crucial TCP mechanisms, timeouts, are explained using IP trace and socket-level trace examples. You will learn which parameters are for which mechanism. After reading this chapter, you will be able to adjust almost any TCP/IP parameter and improve TCP/IP performance.

- Chapter 6, “Performance Tuning Tools”

  This chapter briefly describes some tools (commands) useful for performance tuning.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

- *AIX Version 4.1 System Management Guide: Communications and Networks*, SC23-2526
- *AIX Version 4.1 System Management Guide: Operating System and Devices*, SC23-2525
- *AIX Version 4.1 System User′s Guide: Communications and Networks*, SC23-2545
- *AIX Version 4.1 Communications Programming Concepts*, SC23-2610
- *AIX Version 3.2 and 4.1 Performance Tuning Guide*, SC23-2365
- *AIX Version 4.1 Files Reference*, SC23-2512
- *Token-Ring Network Architecture Reference*, SC30-3374

International Technical Support Organization Publications

- *TCP/IP Tutorial and Technical Overview*, GG24-3376

A complete list of International Technical Support Organization publications known as redbooks, with a brief description of each, may be found in *International Technical Support Organization Bibliography of Redbooks*, GG24-3070.

How Customers Can Get Redbooks and Other ITSO Deliverables

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **IBMLINK**

  Registered customers have access to PUBORDER to order hardcopy, to REDPRINT to obtain BookManager BOOKs

- **IBM Bookshop** — send orders to:

  usib6fpl@ibmmail.com (United States)
  bookshop@dk.ibm.com (Outside United States)

- **Telephone orders**

1-800-879-2755	Toll free, United States only
(45) 4810-1500	Long-distance charge to Denmark, answered in English
(45) 4810-1200	Long-distance charge to Denmark, answered in French
(45) 4810-1000	Long-distance charge to Denmark, answered in German
(45) 4810-1600	Long-distance charge to Denmark, answered in Italian
(45) 4810-1100	Long-distance charge to Denmark, answered in Spanish

- **Mail Orders** — send orders to:

IBM Publications	IBM Direct Services
P.O. Box 9046	Sortemosevej 21
Boulder, CO 80301-9191	DK-3450 Allerød
USA	Denmark

- **Fax** — send orders to:

1-800-445-9269	Toll-free, United States only
45-4814-2207	Long distance to Denmark

- **1-800-IBM-4FAX (United States only)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services**

  Send note to `softwareshop@vnet.ibm.com`

- **Redbooks Home Page on the World Wide Web**

  `http://www.redbooks.ibm.com/redbooks`

- **IBM Direct Publications Catalog on the World Wide Web**

  `http://www.elink.ibmlink.ibm.com/pbl/pbl`

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How IBM Employees Can Get Redbooks and Other ITSO Deliverables

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet**

  Type `GOPHER.WTSCPOK.ITSO.IBM.COM`
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET GG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET GG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```
- **Redbooks Home Page on the World Wide Web**

  `http://w3.itso.ibm.com/redbooks/redbooks.html`

  IBM employees may obtain LIST3820s of redbooks from this page.
- **IBM Direct Publications Catalog on the World Wide Web**

  `http://www.elink.ibmlink.ibm.com/pbl/pbl`
- **ITSO4USA category on INEWS**
- **IBM Bookshop** — send orders to:

  USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Acknowledgments

The author of this document is:

Akihisa Matoba
IBM Japan

Thanks to the following people for their invaluable advice and guidance provided in the production of this document:

Ichiroh Takiguchi
IBM Japan

Masahiro Furutera
IBM Japan

Special thanks to the following people who continuously encouraged me to complete this book.

Yoshimichi Kosuge
IBM Japan

Hisashi Shirai
IBM Japan

Hitoshi Denda
IBM Japan

Chapter 1. TCP/IP Configuration for AIX V4.1

This chapter provides TCP/IP configuration procedures for AIX V4.1 and presents basic configuration methods and related knowledge. The purpose of this chapter is not only to provide the basic configuration procedure knowledge ,but also the necessary knowledge and understanding to debug or correct any configuration anomalies.

It is assumed that you already have a basic knowledge and understanding of the TCP/IP protocol suite and LAN (Ethernet and token-ring). You can refer to *System Management Guide: Communications and Networks*, SC23-2526 to get detailed information. If you know the usual UNIX network configuration procedures, they will help you in understanding this chapter more thoroughly.

Differences Between AIX Version and Level

We have prepared this book using AIX Version 4.1.2. All the panel examples, command outputs, and configuration files are taken using this version and level. You may see different results on your system, but we believe there are no serious differences and that almost all portions of this book are valid.

1.1 Configuration Overview

Basically, the following three configuration methods are provided with AIX V4.1:

- System Management Interface Tool (SMIT)
- High-level commands
- Standard UNIX commands

Remember, SMIT uses high-level commands internally, so they both have the same effect (or just consider that SMIT provides an ease-of-use front end to the high-level commands). Both SMIT and high-level command interact with the object data manager (ODM). The ODM is a system configuration database that keeps the configuration data in binary form; you cannot read or modify it without SMIT or high-level commands. The SMIT, high-level commands, and ODM are unique AIX features and advantages.

Note: From the end users′ point of view, there are three methods (as listed above). In this book we treat both SMIT and high-level commands as equivalents.

The other method is to use standard UNIX commands and ASCII configuration files. This configuration option provides a compatible configuration procedure with other UNIX systems and may help ease your administration tasks in multi-vendor environments.

Traditional UNIX uses ASCII configuration files and startup scripts (usually called rc files). The configuration parameters stored in these ASCII files are easy to review, modify or update.

From the users′ point of view, both ODM and ASCII files have the same effects and can run TCP/IP successfully without any noticeable differences. However, from the network or system administrator′s point of view, they are not the same;

this is especially true when you have to debug configuration parameters. If you don′t understand both procedures well enough, you may not be able to isolate the cause of trouble.

A brief comparison of both of the methods for the minimum configuration is listed below:

SMIT and High-Level Command with ODM

- The following configuration parameters are stored in ODM:

  Network device inet0, and its attributes

  Interface devices such as tr0 and en0, and their attributes

  Adapter devices such as tok0 and ent0, and their attributes

- The following parameters are stored in ASCII files:

  IP address and interface name in /etc/hosts

  Domain Name System (DNS) server and domain name in /etc/resolv.conf

  Note: The use of DNS and this file are optional.

- The parameters are loaded from ODM when /etc/rc.net is executed.

  Note: This file is usually executed during the boot procedure.

Standard UNIX Command with ASCII Files

- The following configuration parameters are written in the startup script /etc/rc.bsdnet:

  Network interface configuration with ifconfig command

  Routing information with route command

  Host name with hostname command

- The following parameters are stored in ASCII files:

  IP address and interface name in /etc/hosts

  DNS server and domain name in /etc/resolv.conf

  Note: The use of DNS and this file are optional.

- The parameters are set when /etc/rc.bsdnet is executed.

  Note: This file is usually executed during the boot procedure.

It is highly recommended that you *do not* mix both methods. Using both methods concurrently may introduce confusion into your system.

1.2 SMIT and High-Level Command with ODM

Usually the configuration procedures are explained for each network function, such as the interface configuration or static route configuration. This is what commercial books usually do, but we have organized this chapter based on the logical device structure in ODM. We believe that this organization helps you to get a better understanding of the relationship between each parameter and the corresponding data in the ODM. For the TCP/IP configuration there are three logical devices involved. They are as follows:

- Network device for internet protocol (TCP/IP)
- Network interface device
- Adapter device

We explain the configuration procedures of each logical device in this order.

Although you can configure TCP/IP by configuring each logical device one by one, AIX offers a more convenient approach called *minimum configuration*. With this option you need to run only one SMIT panel or only one high-level command. We recommend that you use this option if you are going to configure TCP/IP for the first time on your system. Then, if you get additional requirements to update or expand the minimum configuration, customize each logical device as you see fit. We explain the minimum configuration procedure first.

1.2.1 Minimum Configuration

If your system has more than two network adapter cards and you need to configure interfaces for all adapter cards, you have to repeat this procedure for every interface you need to configure.

In this procedure, you are allowed to configure minimum parameters to run TCP/IP. When you need to configure parameters that are not covered in this procedure, you have to go to the corresponding logical device configuration procedure later.

1.2.1.1 Using SMIT

1. Issue the following command to invoke SMIT:

   ```
   # smitty mktcpip
   ```

2. Select the appropriate interface that you need to configure, as shown in the following figure:

```
                      Available Network Interfaces

 Move cursor to desired item and press Enter.

   en0     Standard Ethernet Network Interface
   et0     IEEE 802.3 Ethernet Network Interface
   tr0     Token Ring Network Interface

 F1=Help                F2=Refresh             F3=Cancel
 F8=Image               F10=Exit               Enter=Do
 /=Find                 n=Find Next
```

Figure 1. SMIT Available Network Interface Selection Panel

You should see the appropriate interfaces for each adapter card installed. You have two options for an Ethernet adapter card: en0 and et0.

Note: It is possible to configure both en0 to et0 on the same Ethernet adapter. In that configuration you have two distinct Ethernets (Version 2 and IEEE802.3) running on the same physical network (cable). You may have to configure your system as a router or gateway to make it possible to communicate between each interface.

If you don′t see the necessary interface for the adapter card you installed, reboot the system or issue the following command:

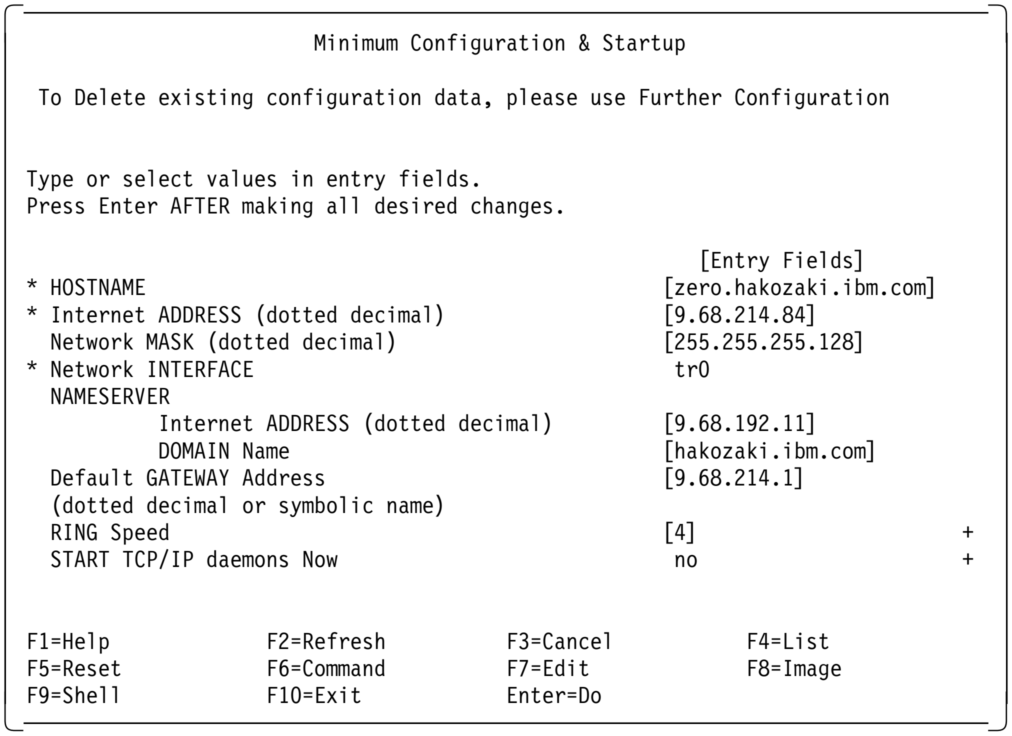
```
# cfgmgr
```

If the command doesn′t fix the problem, you have a system configuration problem not a TCP/IP or networking problem. The configuration manager cannot recognize the adapter. You may need to call an IBM CE to fix this problem.

Important Notice for ISA Bus Adapter Users

If you are using an ISA bus adapter card, the cfgmgr cannot recognize your card. You need to follow an explicit card configuration procedure through SMIT or a high-level command after the card installation. Refer to 1.5.9, "ISA Bus Adapter Consideration" on page 52.

3. Enter the appropriate parameters in the following panel. Some sample parameters have been placed in the fields in this example.



```
                          Minimum Configuration & Startup

 To Delete existing configuration data, please use Further Configuration


Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                       [Entry Fields]
* HOSTNAME                                          [zero.hakozaki.ibm.com]
* Internet ADDRESS (dotted decimal)                 [9.68.214.84]
  Network MASK (dotted decimal)                     [255.255.255.128]
* Network INTERFACE                                  tr0
  NAMESERVER
           Internet ADDRESS (dotted decimal)        [9.68.192.11]
           DOMAIN Name                              [hakozaki.ibm.com]
  Default GATEWAY Address                           [9.68.214.1]
  (dotted decimal or symbolic name)
  RING Speed                                        [4]                      +
  START TCP/IP daemons Now                           no                      +


F1=Help            F2=Refresh         F3=Cancel          F4=List
F5=Reset           F6=Command         F7=Edit            F8=Image
F9=Shell           F10=Exit           Enter=Do
```

Figure 2. SMIT TCP/IP Minimum Configuration Startup Panel (Token-Ring)

HOSTNAME
Fill this entry with the interface name that you configure. If you are using ONS name resolution, this should be the fully qualified host name.

Internet ADDRESS
Fill this entry with the IP address of this interface.

Network MASK 50
If your network is using subnet mask, you have to fill this entry with the mask value or you cannot communicate with other systems.

NAMESERVER
If you are using DNS in your network, you may need to fill in these entries. These are completely optional and you can configure DNS later.

Default GATEWAY Address
If your network has a default gateway machine, you may want to specify the default gateway. If you specify it, the static route is created for the default gateway.

Note: This example shows the use of an IP address. An IP address should be used. Using a host name may cause problems if name resolution is incorrect or not functioning. But if you have not written this host name in the /etc/hosts file, you have to specify the IP address.

RING Speed
You can choose the token-ring speed. Of course, this value must match the speed of the other stations.

START TCP/IP daemons Now
If you specify Yes, the startup script /etc/rc.tcpip is executed after the interface configuration. This script starts several TCP/IP application daemons.

Note: AIX V3.2 and V4.1 display only a START Now message here.

During this procedure, necessary information (devices and their attributes) is stored in the ODM. Although this is only one SMIT panel, the adapter card device, the network interface device, and the network device are configured at once. Be aware that this configuration procedure only sets minimum attributes. For other attributes of each device in the ODM, default attribute values are applied automatically. You can check all the attributes stored in the ODM with the high-level command.

In this method, not only the ODM but also some ASCII files are updated or created.

1.2.1.2 Using High-Level Command

The following high-level commands provide the same functionality as does the SMIT minimum configuration procedure.

1. If you want to review the available interface devices on your system, you can use the lsdev command, as follows:

```
# lsdev -C -c if
lo0 Available  Loopback Network Interface
et0 Defined    IEEE 802.3 Ethernet Network Interface
tr0 Available  Token Ring Network Interface
en0 Defined    Standard Ethernet Network Interface
#
```

   With this command, you can also see the device status. The following is a brief explanation:

   Available The device (interface) is recognized and defined in the ODM. Also, it has already been configured.

   Defined The device (interface) is recognized and defined in the ODM. In other words, the customized object for this device was created in the ODM, but the device (interface) has not yet been configured.

This means that the corresponding device driver has not yet been loaded.

Stopped The device (interface) is recognized and defined in the ODM. It has already been configured but has currently stopped running.

2. Issue the following command to configure an interface. This example does the same thing as the SMIT example, and you can easily understand which flag means what. Of course, you should refer to the InfoExplorer or a manual.

```
# mktcpip -h zero.hakozaki.ibm.com -a 9.68.214.84 -m 255.255.255.128
-i tr0 \ >n 9.68.192.11 -d hakozaki.ibm.com -g 9.68.214.1 -r 4
```

3. Now the command is completed as shown below:

```
# mktcpip -h zero.hakozaki.ibm.com
-a 9.68.214.84 -m 255.255.255.128 -i tr0 \
> -n 9.68.192.11 -d hakozaki.ibm.com -g 9.68.214.1 -r 4
tr0
zero
inet0 changed
tok0 changed
Warning: Token Ring device busy. The ring speed change
         (if any) will take effect on the next reboot.
tr0 changed
inet0 changed
#
```

During the SMIT configuration procedure, if you press the PF6 key for F6=Command, you can see the corresponding high-level command. After the SMIT procedure is complete, you can also check the smit.script file and see the same command.

One advantage of using the high-level command may be that you can execute the configuration from a shell script and make the process automatic.

1.2.2 Where the Configuration Information Is Stored

During the minimum configuration procedure, the configuration information is stored in several places in the system. Some information is stored in the ODM and some is stored in the ASCII files.

1.2.2.1 Network Interface Device Information in ODM

In the following, you can see what attributes have been configured for your network interface. This is the case for the token-ring interface.

```
# lsattr -E -l tr0
mtu       1492            Maximum IP Packet Size for This Device     True
mtu_4     1492            Maximum IP Packet Size for This Device     True
mtu_16    1492            Maximum IP Packet Size for This Device     True
remmtu    576             Maximum IP Packet Size for REMOTE Networks True
netaddr   9.68.214.84     Internet Address                           True
state     up              Current Interface Status                   True
arp       on              Address Resolution Protocol (ARP)          True
allcast   on              Confine Broadcast to Local Token-Ring      True
hwloop    off             Enable Hardware Loopback Mode              True
netmask   255.255.255.128 Subnet Mask                                True
security  none            Security Level                             True
authority                 Authorized Users                           True
broadcast                 Broadcast Address                          True
#
```

During minimum configuration, only the following attributes are set and stored. Other attributes are set with the predefined template that provides default attributes.

netaddr This is the network address (IP Address) for this interface.

netmask This is the subnet mask used for this interface.

state This is the status of the interface.

The right-most column shows whether or not the attribute is being used. The term True means that the attribute is effective and applied. But for the token-ring, the following attributes are not used in the current release although they are displayed as true:

- mtu_4
- security
- authority
- mtu_16

1.2.2.2 Adapter Device Information in ODM

You can see (below) what attributes have been configured for your adapter card. This is the case for the token-ring adapter.

```
# lsattr -E -l tok0
intr_level      9              Bus interrupt level                 True
intr_priority   3              Interrupt priority                  False
bios_addr       0xcc000        Address of bus memory used for BIOS True
xmt_que_size    96             TRANSMIT queue size                 True
bus_io_addr     0xa20          Bus I/O address                     True
shared_mem_addr 0xd0000        Bus memory address                  True
shared_mem_leng 0x4000         Width of shared bus memory          True
ring_speed      4              RING speed                          True
attn_mac        no             Receive ATTENTION MAC frame         True
beacon_mac      no             Receive BEACON MAC frame            True
use_alt_addr    no             Enable ALTERNATE TOKEN RING address True
alt_addr        0x40007e086670 ALTERNATE TOKEN RING address        True
#
```

Note: This example was taken with the ISA bus token-ring adapter.

During minimum configuration, only the following attribute is set and stored. Other attributes are set with the predefined template that provides the default attributes.

ring_speed This is the token-ring′s speed.

1.2.2.3 Network Device Information in ODM

You can see (below) what attributes have been configured for your network. Other devices, such as the interface or the adapter, can have multiple instances. For example, if you configure the two Ethernet interfaces, en0 and en1, you will get both en0 and en1 in ODM. You can configure *only one* network device called inet0. There are no network devices called inet1.

```
# lsattr -E -l inet0
hostname      zero.hakozaki.ibm.com Host Name                      True
gateway                             Gateway                         True
route         net,,0,9.68.214.1     Route                           True
bootup_option no                    Serial Optical Network Interface True
#
```

During minimum configuration, only the following attributes are set and stored. Other attributes are set with the predefined template that provides the default attributes.

hostname The host name (system name) should be stored in this attribute. But if you run smitty mktcpip, you will see that the interface name is stored here. If you run smitty mktcpip more than once, only the latest interface name is stored. This may be a problem for a multi-homed host.

route This is the default route for this system.

Note: The attribute route is used to store routing information, and it is not restricted to store only the default route. In this minimum configuration procedure, only the default route (gateway) can be entered. Later, if you add other routing information, you will see that the route has multiple entries in inet0.

1.2.2.4 The Host Table /etc/hosts

This file must exist before the configuration procedure begins. AIX provides a default template file for /etc/hosts. After the procedure described in the previous section, the file should contain the following lines:

```
# cat /etc/hosts
127.0.0.1              loopback localhost      # loopback (lo0) name/address
9.68.214.84     zero.hakozaki.ibm.com
#
```

Note: The default file provided for the system contains many comments and sample entries. In this example, we omitted those lines.

1.2.2.5 The DNS Resolver Configuration File /etc/resolv.conf

This file is only created if you specify the following entry fields of the SMIT panel:

```
  NAMESERVER
           Internet ADDRESS (dotted decimal)         [9.68.214.84]
           DOMAIN Name                               [hakozaki.ibm.com]
```

The existence of this file automatically invokes the DNS resolver routine when an IP address of a host is needed. DNS configuration is completely optional, so don′t specify NAMESERVER if you don′t use DNS. This file should look as follows:

```
# cat /etc/resolv.conf
nameserver      9.68.192.11
domain  hakozaki.ibm.com
#
```

Note: When you don′t need DNS, do not create /etc/resolv.conf even if the file is left blank. If this file exists, the DNS resolver is automatically used and tries to access the DNS server. If the file is blank, the resolver considers that the server (named) is running on the same machine and tries to access it.

1.2.3 The Startup Script /etc/rc.net

The configuration you made with the smitty mktcpip or mktcpip commands are immediately available and are saved for the system reboot. When the system boots, during the second boot phase, the configuration manager calls the startup script /etc/rc.net. In this script, several high-level commands are executed and network-related configuration parameters are loaded from the ODM to the AIX kernel (memory). Then, the AIX kernel is set with those parameters and they become effective. Refer to A.1, "/etc/rc.net" on page 337 for the complete list of this script.

1.2.3.1 Where Network Interfaces Are Set at Boot

In this section, network interfaces are defined and configured in the script /etc/rc.net as follows:

```
##################################################################
# Part I - Configuration using the data in the ODM database:
# Enable network interface(s):
##################################################################
# This should be done before routes are defined.
# For each network adapter that has already been configured, the
# following commands will define, load and configure a corresponding
# interface.
# NOTE: If you are using a diskless/dataless machine, you may want to
#       disable the logging of messages to the LOGFILE by the cfgif
#       routine. On some diskless/dataless machines, the message
#       logging causes the client to hang on LED 581 when booting.

/usr/lib/methods/defif                  >>$LOGFILE 2>&1
/usr/lib/methods/cfgif  $*                      >>$LOGFILE 2>&1
# If a diskless or dataless machine uses this configuration method, you
# may want to replace the previous line with the following.
#
# /usr/lib/methods/cfgif  $*
```

Notice that the configuration methods defif and cfgif are used. A brief explanation of these commands is shown as follows:

defif	This command creates the interface devices in the ODM for all the recognized adapter devices and sets the default attributes for each interface device with predefined attributes. Finally, it sets the status flag to Defined.
cfgif	This command retrieves the customized attributes from the ODM, sets them for the created interface devices, invokes the ifconfig command internally, and loads the interface instances in the kernel. Finally, it sets the status flag to Available.

1.2.3.2 Where Static Routes Are Set at Boot

In this section, static routes are defined and configured in the script /etc/rc.net.

```
##################################################################
# Configure the Internet protocol kernel extension (netinet):
##################################################################
# The following commands will also set hostname, default gateway,
# and static routes as found in the ODM database for the network.
/usr/lib/methods/definet                       >>$LOGFILE 2>&1
/usr/lib/methods/cfginet                       >>$LOGFILE 2>&1
```

Notice that the configuration methods definet and cfginet are used. A brief explanation of these commands is shown below:

definet This command creates the network device inet0 in ODM and sets the customized attributes for the network device with the attributes in the ODM. Finally, it sets the status flag to Defined.

cfginet This command loads the necessary protocol module for inet0 (TCP/IP module) and initializes it. Then, it sets the status flag to Available. Next, it invokes the hostname command internally, sets the host name from the attribute host name of inet0, invokes the route command internally, and configures static routes from the attribute route of inet0.

These configuration methods are intended to run through the use of high-level commands such as mkdev. It is not recommended that you execute these commands from the command line.

If you have done the minimum configuration via the smitty mktcpip or mktcpip command, the interface name is stored as an attribute host name of the device inet0. Then, that attribute is loaded by the cfginet method.

What is the Configuration Method?

A method is a procedure to access a logical device. The logical device is a system component which is recognized by the configuration manager; its configuration information is stored in the ODM.

Methods need not be a command. Rather, they are intended to be used internally by the configuration manager or other system functions. High-level commands such as chdev and mkdev provide the front end to the methods. It is highly recommended that you use a high-level command instead of invoking a method directly at the command line.

1.2.3.3 Where hostid and uname Are Set at Boot

Also, there are the following commands in this script to set the hostid and uname based on the host name loaded by cfginet:

```
##################################################################
# Part III - Miscellaneous Commands.
##################################################################
# Set the hostid and uname to hostname, where hostname has been
# set via ODM in Part I, or directly in Part II.
# (Note it is not required that hostname, hostid and uname all be
# the same).
/usr/sbin/hostid hostname             >>$LOGFILE 2>&1
/bin/uname -S hostname|sed 's/\..*$//' >>$LOGFILE 2>&1
```

If you have any shell scripts or programs which use the hostid or uname command or gethostid(), there may be a problem. They may get an interface name instead of a hostname if your system has more than one interface.

1.2.3.4 Where Standard UNIX Commands Should Be Written

In this script, standard UNIX configuration commands ifconfig and route can be executed, although they are totally commented out. If you prefer to use standard UNIX configuration procedures during boot up and don′t want to use the ODM, you can comment out AIX high-level commands and make these UNIX commands effective.

```
##################################################################
# Part II - Traditional Configuration.
##################################################################
# An alternative method for bringing up all the default interfaces
# is to specify explicitly which interfaces to configure using the
# ifconfig command.  Ifconfig requires the configuration information
# be specified on the command line.  Ifconfig will not update the
# information kept in the ODM configuration database.
#
# Valid network interfaces are:
# lo=local loopback, en=standard ethernet, et=802.3 ethernet
# sl=serial line IP, tr=802.5 token ring, xs=X.25
#
# For example, en0 denotes standard Ethernet network interface, unit zero.
#
# Below are examples of how you could bring up each interface using
# ifconfig.  Since you can specify either a host name or a dotted
# decimal address to set the interface address, it is convenient to
# set the host name at this point and use it for the address of
# an interface, as shown below:
#
#/bin/hostname robo.austin.ibm.com      >>$LOGFILE 2>&1
#
# (Remember that if you have more than one interface,
# you′ll want to have a different IP address for each one.
# Below, xx.xx.xx.xx stands for the internet address for the
# given interface).
#
#/usr/sbin/ifconfig lo0 inet loopback    up >>$LOGFILE 2>&1
#/usr/sbin/ifconfig en0 inet hostname  up >>$LOGFILE 2>&1
#/usr/sbin/ifconfig et0 inet xx.xx.xx.xx  up >>$LOGFILE 2>&1
#/usr/sbin/ifconfig tr0 inet xx.xx.xx.xx  up >>$LOGFILE 2>&1
#/usr/sbin/ifconfig sl0 inet xx.xx.xx.xx  up >>$LOGFILE 2>&1
#/usr/sbin/ifconfig xs0 inet xx.xx.xx.xx  up >>$LOGFILE 2>&1
#
#
# Now we set any static routes.
#
# /usr/sbin/route add 0 gateway                 >>$LOGFILE 2>&1
# /usr/sbin/route add 192.9.201.0 gateway       >>$LOGFILE 2>&1
```

1.2.4 Network Device Configuration

Whenever you want to update or modify the minimum configuration, you can use the procedure described here. You can choose an appropriate SMIT panel for the device′s attributes that you need to change. Also, the AIX high-level command is available.

You have a few things to do with the network device inet0; it may be that the only attribute you need to add or delete is a static route. So, we only explain how to add and delete a static route.

1.2.4.1 Using SMIT to Add a Static Route

1. Issue the following command to invoke SMIT:

   ```
   # smitty mkroute
   ```

2. Enter the appropriate parameters in the panel shown in Figure 3.

```
                               Add a Static Route

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
  Destination TYPE                                    net                   +
* DESTINATION Address                                []
  (dotted decimal or symbolic name)
* GATEWAY Address                                    []
  (dotted decimal or symbolic name)
* METRIC (number of hops to destination gateway)     [1]                     #
  Network MASK (hexadecimal or dotted decimal)       []

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 3. SMIT Add Static Route Panel

Destination TYPE
: This specifies whether the destination of the static route is a host machine or an IP network.

DESTINATION Address
: This specifies the IP address of the destination. If the destination is a network, only the network portion of the IP address is needed. Although you can use a symbolic name if it is defined in /etc/hosts for a host system and in /etc/networks for a network, it is not recommended.

GATEWAY Address
: This specifies the gateway machine′s IP address. As previously described, you can use a symbolic name if it is defined in /etc/hosts, but it is not recommended.

METRIC
: Metric is a unit of distance from your machine to the destination. It usually means the number of gateways you have to go through. But, for a static route, the metric doesn′t have much meaning. The only thing you should care about is that the metric is greater than or equal to 0 or 1.

Network MASK

From AIX V4.1, you can specify the subnet mask for each route. But with the current routing scheme, all routes attached to the same network interface must share the same network mask. The only exception is the default route, which doesn′t have a network mask. You don′t need to specify anything here.

Note: Metric is also referred to as the hop count. We use both terms in this book.

METRIC Field Mystery

Actually, some AIX Versions don′t have the metric field at this panel, but this should not be a problem. AIX Version 3.1 had an entry field for METRIC in the SMIT panel. When AIX Version 3.2 was introduced, this entry field for METRIC had been removed. AIX Version 3.2.5 contains the entry field used for METRIC. It must have been reinstalled in a release between Version 3.2.0 and 3.2.5. We are not sure of the reason for this stray METRIC field, but again, this has never been a problem.

1.2.4.2 Using SMIT to Remove a Static Route

1. Issue the following command to invoke SMIT:

```
# smitty rmroute
```

2. Enter the appropriate parameters in the following panel:

```
                             Remove Static Route

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                       [Entry Fields]
  Destination TYPE                                   net                   +
* DESTINATION Address                               []
  (dotted decimal or symbolic name)
* GATEWAY Address                                   []
  (dotted decimal or symbolic name)
  Network MASK (dotted decimal)                     []

F1=Help            F2=Refresh         F3=Cancel          F4=List
F5=Reset           F6=Command         F7=Edit            F8=Image
F9=Shell           F10=Exit           Enter=Do
```

Figure 4. SMIT Remove Static Route Panel

1.2.4.3 Using High-Level Command to Add a Static Route

You can do the same thing with a high-level command as you do with SMIT. You can change any device attribute with the chdev command. This command, as well as SMIT, updates the parameter in both the kernel (memory) and the ODM.

If you want to add or remove a static route, you have to work with the attribute route of the interface device inet0.

In this example, the destination is the network and the parameters are assumed to be as shown below:

Destination is the network with the IP address 9.170.7.

Gateway′s IP address is 9.68.214.82.
The hop count is 2.

Note: There are no differences for the metric greater than or equal to 1. Although we use 2, 1 is enough.

Execute the chdev command with the previous attribute. See the following example:

```
# chdev -l inet0 -a route=net,-hopcount,2,9.170.7,9.68.214.82
inet0 changed
#
```

You can see the ODM with the following command. Although the chdev command updates both the ODM and the routing table in the kernel (memory), the lsattr command shows only the data in the ODM. You may want to view the kernel routing table using the netstat -r command.

```
# lsattr -E -l inet0
hostname      zero                                 Host Name                        True
gateway                                            Gateway                          True
route         net,,0,9.68.214.1                    Route                            True
route         net,-hopcount,2,9.170.7,9.68.214.82
bootup_option no                                   Serial Optical Network Interface True
#
```

In this example, the route net,,0,9.170.2.45 is the default route configured when smitty mktcpip was executed. Notice the hop count is 0 and -hopcount is not stored with the default route.

An interesting pitfall is that route is a multivalued attribute and you cannot display routing information only. If you try to do this, you get the following error message:

```
# lsattr -E -l inet0 -a route
lsattr: 0514-530 Cannot display information about ″route″ because
        this type of attribute cannot be displayed.
#
```

1.2.4.4 Using High-Level Command to Remove a Static Route

Enter the following command to remove a static route:

```
#chdev -l inet0 -a delroute=net,-hopcount,2,9.170.7,9.68.214.82
inet0 changed
#
```

You can see the ODM with the following command. Of course, the kernel routing table should have been updated, although the lsattr command doesn′t confirm this.

```
# lsattr -E -l inet0
hostname      zero              Host Name                        True
gateway                         Gateway                          True
route         net,,0,9.68.214.1 Route                            True
bootup_option no                Serial Optical Network Interface True
#
```

1.2.4.5 How to Know Whether or Not Routes Are Effective

As you already know, the lsattr command involves only the parameters in the ODM. You can also update parameters in the kernel with the chdev command. You should use the netstat -r command to check the kernel routing table.

You can see current routings in the kernel as shown in the following example. In this example, the default route is set. Pay attention to the flags for a real gateway host and network interface. The gateway 9.68.214.1 newton has the flag UG which means that the static route through 9.68.214.1 is currently active and 9.68.214.1 is really a remote host. The gateway zero.haozaki.ibm.com is the name of the network interface and has the flag U. This means that the destination 9.68.214 is the local network to which this interface is directly attached.

```
# netstat -r | grep U
Destination      Gateway            Flags     Refs     Use  Interface
default          9.68.214.1         UG          0      256  tr0
9.68.214         zero.hakozaki.ibm. U           2      327  tr0
localhost        localhost          UH          1        0  lo0
#
```

If you add a static route with the following command:

```
# chdev -l inet0 -a route=net,-hopcount,2,9.170.7,9.68.214.82
inet0 changed
#
```

then you will see the following with the netstat -r command. In this example, the gateway host 9.68.214.82 is stored in /etc/hosts, and the IP address is automatically translated to the host name mat.

```
# netstat -r | grep U
Destination      Gateway            Flags     Refs     Use  Interface
default          9.68.214.1         UG          1      281  tr0
9.68.214         zero.hakozaki.ibm. U           2      327  tr0
9.170.7          mat.hakozaki.ibm.c UG          0        0  tr0
localhost        localhost          UH          1        0  lo0
#
```

1.2.5 Network Interface Device Configuration

Whenever you want to update or modify the minimum configuration, you can use the procedure described here. You can choose an appropriate SMIT panel for the device′s attributes that you need to change. Also, the AIX high-level command is available.

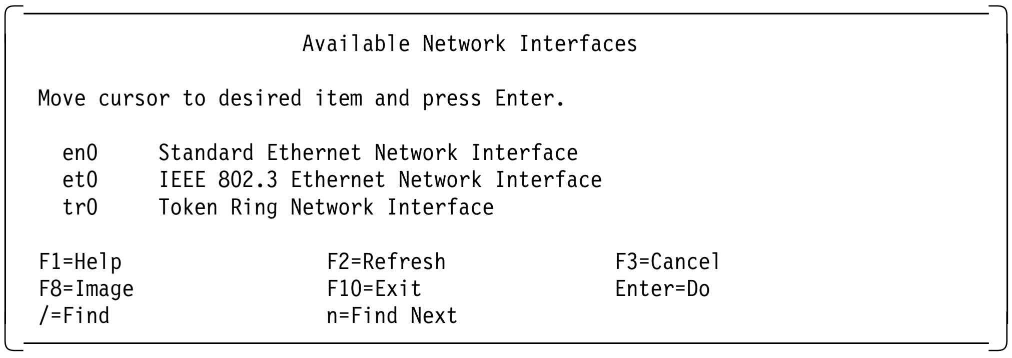
1.2.5.1 Using SMIT

Although several parameters or attributes of an interface device can be changed, once you get TCP/IP running successfully, you may not need to change any of it. In our experience, the interface′s status is the only parameter that needed to be changed from up to down or vice versa. This is necessary when you change an adapter configuration.

1. Issue the following command to invoke SMIT:

   ```
   # smitty chinet
   ```

2. Select a necessary interface that you want to change in the following panel:



```
                         Available Network Interfaces

  Move cursor to desired item and press Enter.

    en0     Standard Ethernet Network Interface
    et0     IEEE 802.3 Ethernet Network Interface
    tr0     Token Ring Network Interface

  F1=Help                 F2=Refresh              F3=Cancel
  F8=Image                F10=Exit                Enter=Do
  /=Find                  n=Find Next
```

Figure 5. SMIT Available Network Interfaces Selection Panel

You should see the appropriate interfaces for each adapter card installed. You have two options for an Ethernet adapter card (en0 and et0).

Note: It is possible to configure both en0 and et0 on the same Ethernet adapter. In that configuration, you have two distinct Ethernets (Version 2 and IEEE802.3) running on the same physical network (cable). You may have to configure your system as a router or gateway to make it possible to communicate between each interface.

If you don′t see the necessary interface for the adapter card you installed, issue the following command or reboot the system:

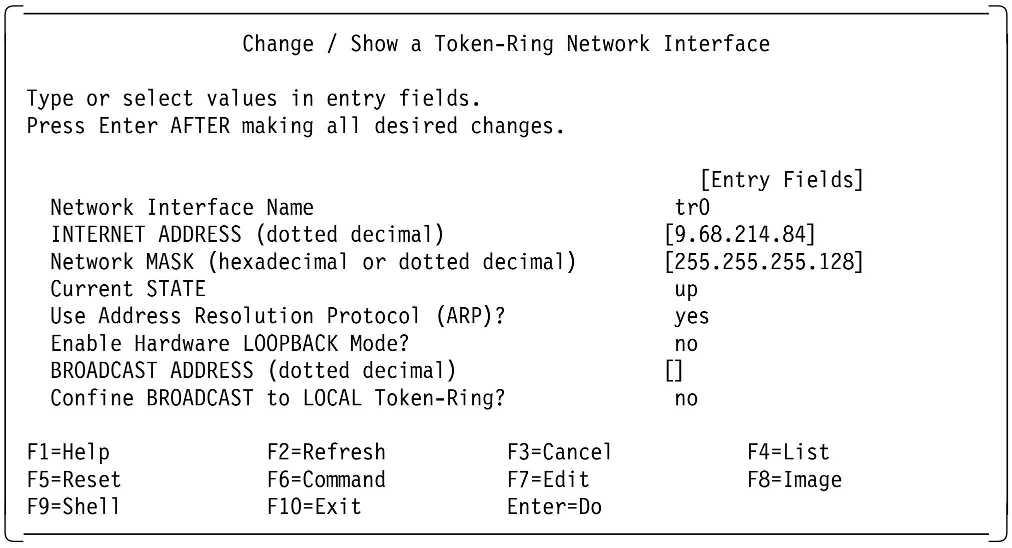
```
# cfgmgr
```

If the command doesn′t fix the problem, you have a system configuration problem (not a TCP/IP or networking problem). The configuration manager cannot recognize the adapter. You may need to call an IBM CE to fix this problem.

Important Notice for ISA Bus Adapter Users

If you are using an ISA bus adapter card, cfgmgr cannot recognize your card. You need to follow an explicit card configuration procedure through SMIT or high-level command after the card installation. Refer to 1.5.9, “ISA Bus Adapter Consideration” on page 52.

3. Enter the necessary attributes in the following panel:



```
                  Change / Show a Token-Ring Network Interface

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
  Network Interface Name                               tr0
  INTERNET ADDRESS (dotted decimal)                   [9.68.214.84]
  Network MASK (hexadecimal or dotted decimal)        [255.255.255.128]
  Current STATE                                        up
  Use Address Resolution Protocol (ARP)?               yes
  Enable Hardware LOOPBACK Mode?                       no
  BROADCAST ADDRESS (dotted decimal)                  []
  Confine BROADCAST to LOCAL Token-Ring?               no

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 6. SMIT Network Interface Configuration Panel (Token-Ring)

This panel completely corresponds to the data in the ODM, which is displayed with the following command:

```
# lsattr -E -l tr0
```

The following are brief explanations for each field given:

INTERNET ADDRESS
Specify whether you want to change the IP address of this interface.

Note: This panel only updates the IP address in the ODM and in the kernel. The host table /etc/hosts is not updated. You should change the host table manually to maintain the consistency.

Network MASK
Specify this if you want to change the subnet mask of this interface.

Current STATE
This is the interface′s current state that is retrieved from the ODM. There are three states: up, down and detach. Although it is displayed as current STATE, it may not be accurate because the state can be changed with the `ifconfig` command without changing the ODM. This should be the STATE just after the system boot.

Note: AIX Version 3.1 doesn′t have an option to make it to the detach state.

Use Address Resolution Protocol (ARP)
Today, almost all systems use Address Resolution Protocol (ARP) to get the destination system′s Media Access Control (MAC) address. If you set this field to no, you must manually provide the IP address to MAC address mapping information to the kernel. Usually it is done with the arp command. We don′t see any reasons to set this field to no except for debug or test purposes.

Enable Hardware LOOPBACK Mode
This parameter is used to enable or disable the hardware loopback mode. It is usually set to no (the default).

BROADCAST ADDRESS
The default sets all bits of the host portion of the destination IP address to 1s. The 4.2 BSD has the broadcast address of all 0s. To

communicate with 4.2 BSD or an equivalent system, you may need to set this field. Changing the broadcast address is a very bad practice. All systems must agree with the broadcast address, and the best way is to abandon such an old system.

Confine BROADCAST to LOCAL Token-Ring
If you set this parameter to Yes, no broadcast packets can get out from the local ring through a bridge. This will reduce the entire network traffic but also limit your communication capability. Notice that this limitation is applied to a data link layer (MAC Layer) broadcast (and indirectly limits IP broadcast). If you set this to Yes, the first 3 bits (called broadcast bits) of the token-ring routing control field of a packet header are set to 0. Possibly the most affected function is ARP, and as a result, the IP function may also be limited.

1.2.5.2 Using High-Level Command

If you want to change the IP address of the interface, use the following steps:

1. If you want to review the list of interfaces on your system, enter the following command:

   ```
   # lsdev -C -c if
   lo0 Available  Loopback Network Interface
   et0 Defined    IEEE 802.3 Ethernet Network Interface
   tr0 Available  Token Ring Network Interface
   en0 Defined    Standard Ethernet Network Interface
   #
   ```

   With this command you can also see the device status.

2. Issue the chdev command to change an attribute, as follows:

   ```
   # chdev -l tr0 -a netaddr=9.170.10.21
   tr0 changed
   #
   ```

   Note: This option only updates the kernel parameter and ODM. If you change the IP address as previously shown, you have to update the host table /etc/hosts manually.

3. You can use the lsattr command to review the updated value. Do not forget that you are looking for the data in the ODM. The kernel may be currently recognizing other values. Use ifconfig to review the values in the kernel.

   ```
   # lsattr -E -l tr0 -a netaddr
   netaddr 9.170.10.21 Internet Address True
   #
   ```

It′s quicker than SMIT, but you have to know the exact name of the device and the attribute that you want to change.

1.2.6 Adapter Device Configuration

Some adapter device attributes have influences on the network performance. Occasionally you may have to change some of them. There is one thing you should know before you change the adapter configuration: first detach the interface or reboot the system.

1.2.6.1 Detaching the Interface or Rebooting the System

Sometimes you may end up with the following error messages when you try to change an adapter configuration. This is the result of a smitty chgtok execution.

```
                                 COMMAND STATUS

Command: failed       stdout: no          stderr: yes

Before command completion, additional instructions may appear below.

Method error (/usr/lib/methods/chgtok):
        0514-062 Cannot perform the requested function because the
                 specified device is busy.

F1=Help             F2=Refresh          F3=Cancel           F6=Command
F8=Image            F9=Shell            F10=Exit            /=Find
n=Find Next
```

Figure 7. Configuration Error Message Example (Micro Channel Bus)

```
                                 COMMAND STATUS

Command: failed       stdout: no          stderr: yes

Before command completion, additional instructions may appear below.

Method error (/usr/lib/methods/chgisatok):
        0514-018 The values specified for the following attributes
                 are not valid:
     xmt_que_size     TRANSMIT queue size

F1=Help             F2=Refresh          F3=Cancel           F6=Command
F8=Image            F9=Shell            F10=Exit
```

Figure 8. Configuration Error Message Example (ISA Bus)

Because a network interface device is built on the top of the adapter device, whenever the network interface is active, you cannot change or update any attribute of the adapter device. In this case, first you have to detach the interface, and after updating the adapter configuration, you have to up the interface again. An alternative is to update the adapter configuration in the ODM only and reboot the system. This is the easiest procedure to do to avoid the error. You should understand that there are three statuses for an interface device, as follows:

up The network interface is currently active. The interface can receive and send packets, and you cannot change the attribute.

down The network interface is currently stopped. The interface cannot receive and send packets. In this status, the device driver is recognizing this interface and keeping attribute values. In order to prevent inconsistency, you cannot change the attribute.

detach The network interface is currently stopped. The interface cannot receive and send packets. In this status, the device driver no longer recognizes this interface, and you can change the attribute.

We show three ways to detach and up a network interface. These procedures aren′t substantially different from each other. But it′s a good idea to know more than one alternative so that you have a better understanding of adapter configuration.

1.2.6.2 Using SMIT to Detach and Up an Interface

1. Issue the following command to get the SMIT panel:

   ```
   # smitty chinet
   ```

2. Choose the appropriate interface in the following panel:

```
                      Available Network Interfaces

 Move cursor to desired item and press Enter.

   en0     Standard Ethernet Network Interface
   et0     IEEE 802.3 Ethernet Network Interface
   tr0     Token Ring Network Interface

 F1=Help                 F2=Refresh              F3=Cancel
 F8=Image                F10=Exit                Enter=Do
 /=Find                  n=Find Next
```

Figure 9. SMIT Available Network Interface Selection Panel

3. You will get the following panel.

```
                  Change / Show a Token-Ring Network Interface

 Type or select values in entry fields.
 Press Enter AFTER making all desired changes.

                                                          [Entry Fields]
   Network Interface Name                               tr0
   INTERNET ADDRESS (dotted decimal)                   [9.170.3.21]
   Network MASK (hexadecimal or dotted decimal)        [255.255.255.0]
   Current STATE                                        up
   Use Address Resolution Protocol (ARP)?               yes
   Enable Hardware LOOPBACK Mode?                       no
   BROADCAST ADDRESS (dotted decimal)                  []
   Confine BROADCAST to LOCAL Token-Ring?               no

 F1=Help             F2=Refresh          F3=Cancel           F4=List
 F5=Reset            F6=Command          F7=Edit             F8=Image
 F9=Shell            F10=Exit            Enter=Do
```

Figure 10. SMIT Network Interface Configuration Panel (Token-Ring)

4. Specify the following and press Enter:

   ```
   Current STATE                                         detach
   ```

   Note: AIX V3.1 doesn′t have the selection option detach in this panel, and so you have to use the ifconfig or chdev command.

5. Then, you can change the adapter device′s attribute. Refer to 1.2.6.7, "Using SMIT to Change the Adapter Device" on page 24 or 1.2.6.8, "Using High-Level Command to Change Adapter Device" on page 26 for details.

6. After you change the adapter device, you have to come back to the previous panel and activate the interface by specifying the following:

```
  Current STATE                                      up
```

7. Then, you have to reconfigure all routing information that was attached to the detached interface. Refer to 1.2.6.6, "A Big Pitfall: Routes Were Lost" on page 23 for details.

During this procedure the configuration data in the ODM and kernel are always consistent with each other because SMIT changes both of them.

1.2.6.3 Using High-Level Command to Detach and Up an Interface

1. If you want to review the list of interfaces on your system, do the following:

   ```
   # lsdev -C -c if
   lo0 Available  Loopback Network Interface
   et0 Defined    IEEE 802.3 Ethernet Network Interface
   tr0 Available  Token-Ring Network Interface
   en0 Defined    Standard Ethernet Network Interface
   #
   ```

   With this command you can also see the device status. You already know the meanings of available and defined.

2. Change the interface state to detach with the chdev command.

   ```
   # chdev -l tr0 -a state=detach
   tr0 changed
   #
   ```

3. You can review it to see whether the change was made to the ODM. If you want to confirm the interface state in the kernel, use the ifconfig or netstat command.

   ```
   # lsattr -E -l tr0 -a state
   state detach Current Interface Status True
   #
   ```

4. You can change the adapter device′s attribute if needed. Refer to 1.2.6.7, "Using SMIT to Change the Adapter Device" on page 24 or 1.2.6.8, "Using High-Level Command to Change Adapter Device" on page 26 for details.

5. After you change the adapter device, you have to issue the chdev command again to activate the interface.

   ```
   # chdev -l tr0 -a state=up
   tr0 changed
   #
   ```

6. You can review it to see if the change was made to the ODM. If you want to confirm the interface state in the kernel, use the ifconfig or netstat command.

   ```
   # lsattr -E -l tr0 -a state
   state up Current Interface Status True
   #
   ```

7. Then, you have to reconfigure all routing information that was attached to the detached interface. Refer to 1.2.6.6, "A Big Pitfall: Routes Were Lost" on page 23 for details.

1.2.6.4 Using the ifconfig Command to Detach and Up an Interface

The ifconfig is a standard UNIX command. In this procedure, you will use both the standard UNIX command and the AIX high-level command simultaneously. We don′t generally recommend this way, but you can avoid problems by *careful* operation.

1. Issue the following ifconfig command to detach the interface:

   ```
   # ifconfig tr0 detach
   ```

   With this method, be aware that you are causing an inconsistency between the data in the ODM and the kernel. After you complete the previous command, the interface is actually in the detach status. But if you check to see the same information in the ODM with the lsattr command, you will get a different answer.

   ```
   # lsattr -E -l tr0 -a state
   state up Current Interface Status True
   #
   ```

   If you want to confirm the interface state in the kernel, use either the ifconfig or netstat command. You don′t need to worry about the inconsistency that exists during the updating procedure. Also, if you reboot the system, the configuration manager will see the data in the ODM and automatically up the interface.

2. You can change the adapter device′s attribute if needed. Refer to 1.2.6.7, "Using SMIT to Change the Adapter Device" on page 24 or 1.2.6.8, "Using High-Level Command to Change Adapter Device" on page 26 for details.

3. After you change the adapter device, you have to activate the interface with the following command:

   ```
   # ifconfig tr0 up
   ```

4. Then, you have to reconfigure all the routing information that was attached to the detached interface. Refer to 1.2.6.6, "A Big Pitfall: Routes Were Lost" on page 23 for details.

1.2.6.5 How to Know Whether an Interface Is Really Up

You can change or see the status in the ODM with chdev and SMIT. The lsattr command only shows the data in the ODM, and you cannot see the status kept in the kernel (memory). Be aware that the value in the kernel is currently effective.

An alternative is the `ifconfig` command. For the interface in the up state, it is displayed as follows:

```
# ifconfig tr0
tr0: flags=80a0043<UP,BROADCAST,RUNNING,ALLCAST,MULTICAST>
        inet 9.68.214.84 netmask 0xffffff80 broadcast 9.68.214.127
#
```

If the interface is in the down state, it is not explicitly displayed as in the following example. Of course, the flag UP is not shown. The flag RUNNING means that the resource, such as a buffer, is assigned to the interface, but it *doesn′t* mean it is usable.

```
# ifconfig tr0
tr0: flags=80a0042<BROADCAST,RUNNING,ALLCAST,MULTICAST>
        inet 9.68.214.84 netmask 0xffffff80 broadcast 9.68.214.127
#
```

Use the following UNIX commands to see the kernel value. With the netstat command, an interface that is in the down state is shown with an *. If an interface is in the detach state, it is not displayed.

```
# netstat -i
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                           186     0      186     0     0
lo0   16896 127         loopback             186     0      186     0     0
tr0*  1492  <Link>40.0.7e.8.66.70              0     0        0     0     0
tr0*  1492  9.68.214    zero                   0     0        0     0     0
#
```

If the interface is in the detach state, it is displayed as follows. The flag UP and RUNNING are not shown.

```
# ifconfig tr0
tr0: flags=80a0002<BROADCAST,ALLCAST,MULTICAST>
        inet 9.68.214.84 netmask 0xffffff80 broadcast 9.68.214.127
#
```

Also, the detached interface cannot be displayed with the netstat -i command:

```
# netstat -i
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                           186     0      186     0     0
lo0   16896 127         loopback             186     0      186     0     0
#
```

1.2.6.6 A Big Pitfall: Routes Were Lost

Be careful about playing with interfaces. If you detach an interface, all the routes attached to the interface will be lost. Even if you up the interface again, the lost routes are not recovered. If you have a problem such as lost communication capability, check the routing information as follows:

```
# netstat -r | grep U
Destination      Gateway            Flags     Refs     Use  Interface
loopback         loopback           UH          1        0  lo0
#
```

If you lost routing information, one of the easiest recovery procedures is to issue the following:

```
# /usr/lib/methods/cfgif
# /usr/lib/methods/cfginet
zero
9.68.214.1 net 0: gateway 9.68.214.1
9.68.214.82 net 9.170.7: gateway 9.68.214.82
#
```

Of course, running SMIT or a high-level command (chdev) is a better alternative to adding routes. You can confirm that the routes are added safely.

```
# netstat -r | grep U
Destination      Gateway            Flags     Refs     Use  Interface
default          9.68.214.1         UG          0        0  tr0
9.68.214         zero.hakozaki.ibm. U           1        0  tr0
9.170.7          mat.hakozaki.ibm.c UG          0        0  tr0
localhost        localhost          UH          1        0  lo0
#
```

1.2.6.7 Using SMIT to Change the Adapter Device

1. Get the interface detach state. Use any of the procedures already explained.
2. Issue the following command to invoke SMIT. This is for the token-ring adapter.

   ```
   # smitty chgtok
   ```

   In the case of an Ethernet adapter, issue the following command:

   ```
   # smitty chgenet
   ```

   If you don′t know the SMIT fast path name, issue the following command and choose your adapter type from the selection menu:

   ```
   # smitty commodev
   ```

   Then, choose the appropriate adapter.

```
                             Token Ring Adapter

 Move cursor to desired item and press Enter.

   tok0 Available 00-01 Token-Ring High-Performance Adapter (8fc8)

 F1=Help                 F2=Refresh              F3=Cancel
 F8=Image                F10=Exit                Enter=Do
 /=Find                  n=Find Next
```

Figure 11. SMIT Network Adapter Selection Panel

3. Enter the necessary parameters that you want to change.

```
              Change / Show Characteristics of a Token Ring Adapter


Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                     [Entry Fields]
  Token Ring Adapter                                 tok0
  Description                                        Token-Ring High-Perfor>
  Status                                             Available
  Location                                           00-01
  TRANSMIT queue size                               [30]                +#
  RING speed                                         4                      +
  Receive ATTENTION MAC frame                        no                 +
  Receive BEACON MAC frame                           no                 +
  Enable ALTERNATE TOKEN RING address                no                 +
  ALTERNATE TOKEN RING address                      [0x]                +
  RECEIVE queue size                                [30]                +#
  STATUS queue size                                 [10]                +#
  Apply change to DATABASE only                      no                 +

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 12. SMIT Network Adapter Configuration Panel

   This panel corresponds to the data in the ODM, which is displayed with the following command:

   ```
   # lsattr -E -l tok0
   ```

The following are brief explanations for each field:

TRANSMIT queue size
This parameter affects network performance. This parameter defines the queue size between the adapter card and the network layer. This queue is used for transmitting data.

RECEIVE queue size
This parameter affects network performance. This parameter defines the queue size between the adapter card and the network layer. This queue is used for receiving data.

STATUS queue size
This parameter may affect network performance. This parameter defines the queue size to store the status of the network protocol and adapter.

RING speed
You can choose 4 Mbps or 16 Mbps. If you set incorrect values, the ring will be filled with beacon frames and it will impact all the other stations. You will also see terrible performance degradation.

Receive ATTENTION MAC frame
Setting this attribute to the yes value places the attention MAC frames, received by the adapter, in the receive ring queue for the application to read. If you specify the no value, the attention MAC frames are ignored. The default value is no.

Receive BEACON MAC frame
Setting this attribute to the yes value places the beacon MAC frames, received by the adapter, in the receive ring queue for the application to read. If you specify the no value, the beacon MAC frames are ignored. The default value is no.

Enable ALTERNATE TOKEN-RING address
You must specify this if you use the alternate MAC address for this adapter. If yes, the hard-coded, burned-in MAC address is not used.

ALTERNATE TOKEN-RING address
You must specify the alternate MAC address. This MAC address must be managed locally, and you must confirm that there are no duplicated addresses. If you are running SNA on the same adapter, you may need to use this function.

Apply change to DATABASE only
Specify whether the input parameters on this panel should be applied to only the ODM.

If you press the Enter key and get the following error, the corresponding interface is not really in the detach state:

```
Method error (/etc/methods/chgtok):
        0514-062 Cannot perform the requested function because the
                 specified device is busy.
```

```
Method error (/usr/lib/methods/chgisatok):
        0514-018 The values specified for the following attributes
                 are not valid:
     xmt_que_size     TRANSMIT queue size
```

The easiest way to avoid this error is to set the needed attribute values in the previous panel and to set the following attribute:

```
  Apply change to DATABASE only                        yes                 +
```

This procedure updates the data in the ODM and doesn′t touch any parameter in the kernel. Therefore, the change cannot be effective immediately, and you need to reboot the system to apply the update. During the shutdown and reboot process, once the interface is put in the detach status, the updated attributes are loaded from the ODM before the interface activates again. In this procedure, by using reboot, you don′t need to detach the interface first.

4. Get the interface up again. Now the changes are effective and the network function can be used.

1.2.6.8 Using High-Level Command to Change Adapter Device

The commands that you have to use may be the following if you change the token-ring speed from 4 MB to 16 MB. With this procedure, you don′t have to reboot the system.

1. If you need to review the available adapters on your system, issue the following command. This example is for token-ring adapters.

```
# lsdev -C -c adapter -t tokenring
tok0 Available 00-03 Token-Ring High-Performance Adapter
tok1 Available 00-04 Token-Ring High-Performance Adapter
#
```

For Ethernet adapters, specify -t ethernet. If you are using an integrated Ethernet adapter, you need to use the following `lsparent` command to list all the Ethernet adapters.

```
# lsparent -C -k ent
ent0 Available 00-00-0E Integrated Ethernet Adapter
#
```

2. Change the interface state to detach. Use any of the procedures already explained.

3. Change the adapter attribute value ring_speed as follows:

```
# chdev -l tok0 -a ring_speed=16
tok0 changed
#
```

You can review to see whether the change was made to the ODM by issuing the following:

```
# lsattr -E -l tok0 -a ring_speed
ring_speed 16 RING speed True
#
```

4. Change the interface state to be up again. Use any of the procedures already explained.

In the above example, although the chdev command can change both the ODM and the kernel, the lsattr command only shows the data in the ODM. If you want to confirm the interface state in the kernel, use the ifconfig command.

Here is an alternative procedure which needs system reboot. The prior procedure doesn′t need to reboot, but it needs the interface detached; this causes communication disruptions during the procedure. Some applications may become a failure state. A reboot confirms that all the applications restart gracefully and you can avoid potential problems.

1. Change the adapter attribute as follows. With the -P flag, this command only updates the data in the ODM and leaves the data in the kernel intact.

   ```
   # chdev -l tok0 -a xmt_que_size=40 -P
   tok0 changed
   #
   ```

2. Shut down the system and reboot it, as follows. After the reboot, the previous change is effective:

   ```
   # shutdown -Fr
   ```

1.3 Standard UNIX Commands with ASCII Configuration Files

With AIX you can configure and run TCP/IP with standard UNIX configuration commands only. You can live in the TCP/IP world without SMIT, the high-level command, or the ODM.

Although this is possible, we don′t see any need to do so except where you need to share the configuration information or procedure with other vendors′ systems. Even in this case, you need to customize the script for each system. This may not be an easy task.

Whenever you need to change the configuration, you need to run the UNIX configuration command to change the kernel parameter. If you need to have a permanent change, you also have to edit the startup script manually. SMIT or a high-level command does this for you.

Note: You cannot update or change an adapter card configuration with a UNIX command. You have to use SMIT or the high-level commands.

1.3.1 BSD Style Startup Configuration

Since the default boot and configuration procedure use the high-level command and the ODM, you have to tell the system you will not use them. We believe that this option is really for the BSD users.

1. To set the parameter in the ODM that you will use UNIX commands to configure the host name, network interfaces, and routings during the boot process, issue the following command:

   ```
   # smitty setbootup_option
   ```

2. Specify yes in the following panel.

```
                   Select BSD style rc Configuration

          Please answer yes if you want BSD style rc configuration.
                          The default is no.

            AIX configuration uses the data in the ODM database and
                     uses the file /etc/rc.net to define,
                 load and configure a corresponding interface.

   BSD style configuration uses the traditional ifconfig command and it uses
       the file /etc/rc.bsdnet to configure the corresponding interface.

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                          [Entry Fields]
  Use BSD Style rc Configuration                        yes

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 13. BSD Style Startup Configuration Panel

3. You can confirm that the change is stored in the ODM by issuing the following command:

   ```
   # lsattr -E -l inet0 -a bootup_option
   bootup_option yes Serial Optical Network Interface True
   #
   ```

   Although the description seems erroneous (Serial Optical Network Interface is not correct), bootup_option shows yes for the next boot, and the configuration manager executes /etc/rc.bsdnet instead of /etc/rc.net script.

4. Edit the script /etc/rc.bsdnet and add the necessary commands to configure your TCP/IP.

In this case, the configuration manager only recognizes /etc/rc.bsdnet, and the data in the ODM is not loaded at the boot time. This doesn′t necessarily mean that the ODM is never used. If you happen to use the chdev and lsattr commands (or SMIT) after the boot, the parameters in the kernel are updated as well as the data in the ODM.

1.3.2 Customizing the Startup Script /etc/rc.bsdnet

You can edit the startup script /etc/rc.bsdnet with your favorite editor. You can write the host name, ifconfig, route and any other command that you want in the script. Refer to A.2, “Startup Script /etc/rc.bsdnet” on page 339 for the complete list of this script.

This script is the only place where the network interfaces and routing information are kept. During the boot process, the configuration is loaded into the kernel.

1.3.2.1 Where the Host Name Is Set

The following section of the script /etc/rc.bsdnet is where the host name is set with the hostname command:

```
#
LOGFILE=/tmp/rc.net.out     # LOGFILE is where all stdout goes.
>$LOGFILE                   # truncate LOGFILE.

no -d lowclust    # set buffer low water mark

/bin/hostname aoot.austin.ibm.com              >>$LOGFILE 2>&1
```

1.3.2.2 Where Interfaces and Static Routes Are Set

The following section of the script /etc/rc.bsdnet is where the interfaces and routings are set with ifconfig and route commands.

```
##################################################################
# Valid network interfaces are:
# lo=local loopback, en=standard ethernet, et=802.3 ethernet
# sl=serial line IP, tr=802.5 token-ring, xt=X.25
##################################################################

/usr/sbin/ifconfig lo0 inet 127.0.0.1 up  >>$LOGFILE 2>&1
/usr/sbin/ifconfig en0 inet hostname up  >>$LOGFILE 2>&1

#/usr/sbin/route add 0 gateway    >>$LOGFILE 2>&1
#/usr/sbin/route add 192.9.201.0 gateway  >>$LOGFILE 2>&1
```

1.3.2.3 Where Other Parameters Are Set

The following section of the script /etc/rc.bsdnet is where the hostid and uname are set. Notice that these parameters are taken from the result of the hostname command:

```
/usr/sbin/hostid hostname   >>$LOGFILE 2>&1
/bin/uname -Shostname|sed 's/\..*$//'  >>$LOGFILE 2>&1
```

1.3.3 Interface Configuration with ifconfig Command

You can interactively configure or change an interface configuration with the ifconfig command; although, the configuration or change is lost with the system reboot.

1.3.3.1 Adding an Interface

An interface configuration process (following) is shown step by step. The interface tr0 is not configured yet.

1. The command ifconfig shows only some flags.

   ```
   # ifconfig tr0
   tr0: flags=80a0002<BROADCAST,ALLCAST,MULTICAST>
   #
   ```

   Also, with `netstat -i` the interface does not exist.

   ```
   # netstat -i
   Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
   lo0   16896 <Link>                           209     0      209     0     0
   lo0   16896 127         loopback             209     0      209     0     0
   #
   ```

2. Issue the ifconfig command. In the following example, the interface address is given as the interface name zero because this name was already written in the host table /etc/hosts with the corresponding IP address:

```
# ifconfig tr0 inet zero netmask 255.255.255.128up
#
```

3. Check to see whether the interface was successfully configured. The flag UP and RUNNING should be shown, as in the following example:

```
# ifconfig tr0
tr0: flags=80a0043<UP,BROADCAST,RUNNING,ALLCAST,MULTICAST>
        inet 9.68.214.84 netmask 0xffffff80 broadcast 9.68.214.127
#
```

You can confirm the result with the netstat command. The interface en0 now exists in the following example:

```
# netstat -i
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                           209     0      209     0     0
lo0   16896 127         loopback             209     0      209     0     0
tr0   1492  <Link>8.0.5a.ab.23.19             10     0        0     0     0
tr0   1492  9.68.214    zero                  10     0        0     0     0
#
```

1.3.3.2 Removing an Interface

You can use the ifconfig command to remove the interfaces, as follows:

1. At first, the interface is set to the down state with the ifconfig command.

```
# ifconfig tr0 down
#
```

2. Check to see whether the interface tr0 is in the down state. The flag UP should not be shown (as in the following example). Notice that the flag RUNNING doesn′t mean the interface is usable. Remember that the interface is not usable, but the resources such as buffer are still allocated.

```
# ifconfig tr0
tr0: flags=80a0042<BROADCAST,RUNNING,ALLCAST,MULTICAST>
        inet 9.68.214.84 netmask 0xffffff80 broadcast 9.68.214.127
#
```

3. Then, the interface is set to the detach state with the ifconfig command, as follows:

```
# ifconfig tr0 detach
#
```

   Check to see whether the interface is in the detach state. The flag UP and RUNNING should not be shown as in the following example:

```
# ifconfig tr0
tr0: flags=80a0002<BROADCAST,ALLCAST,MULTICAST>
        inet 9.68.214.84 netmask 0xffffff80 broadcast 9.68.214.127
#
```

   Note: Do not forget that any routes attached to the detached interface were lost already. You need explicit route configuration for recovery.

You can also confirm the result with the netstat command, because the detached interface was not shown with this command, as follows:

```
# netstat -i
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                           209     0      209     0     0
lo0   16896 127         loopback             209     0      209     0     0
#
```

1.3.4 Routing Configuration with Route Command

You can interactively configure or change the static route with the route command, although the configuration or change is lost with a system boot.

1.3.4.1 Adding a Static Route

Before adding a route, we show what routes are currently configured in this example. Just after the configuration of an interface, no routes are configured except for a route for loopback, as follows:

```
# netstat -r | grep U
Destination      Gateway            Flags     Refs     Use  Interface
loopback         loopback           UH          6      209  lo0
#
```

Then, issue the route command with the necessary parameters. First we have to configure the route to the local network to which the interface is attached. In this example, verbose option (-v) is specified.

Note: We specify the hop count of 0 this time. This is an exceptional operation for the route to the local network.

```
# route -v add -net 9.68.214.0 zero 0
old usage of trailing 0, assuming route to if
so_dst: inet 9.68.214.0; so_gate: inet 9.68.214.84; RTM_ADD: Add Route
pid: 0, len 112, seq 1, errno 0, flags:<UP>
locks:  inits:
sockaddrs: <DST,GATEWAY,NETMASK>
 9.68.214.0 zero (0) 0 ffff ff00 0
0 net 9.68.214.0: gateway zero
#
```

You can confirm that the route was configure successfully, as follows:

```
# netstat -r | grep U
Destination      Gateway            Flags     Refs     Use  Interface
9.68.214         zero               U          0        0  tr0
loopback         loopback           UH          6      209  lo0
#
```

Basically with the AIX V4.1.2 route command syntax we don′t need to specify hop count or metric. As you already know for the static route, the difference between zero and one is important. Because hop count 0 is reserved for local network interfaces, all the network interfaces are configured with `ifconfig` or `smitty mkinet`.

Note: Although the above is true, AIX V3.1 had the metric option with the route command.

Any static route through the gateway has a one or greater hop count and their difference does not matter. Therefore, you don′t need a metric specification with the route command. This is what we are doing now. See the following example for configuring a default route. The flag -v stands for the verbose mode.

```
# route -v add default 9.68.214.1
so_dst: inet 0.0.0.0; so_gate: inet 9.68.214.1; RTM_ADD: Add Route
pid: 0, len 108, seq 1, errno 0, flags:<UP,GATEWAY>
locks:  inits:
sockaddrs: <DST,GATEWAY,NETMASK>
 default 9.68.214.1 (0)
9.68.214.1 net default: gateway 9.68.214.1
#
```

Now the route command is successfully completed and you can check to see whether the route was really added.

```
# netstat -r | grep U
Destination      Gateway            Flags     Refs     Use  Interface
default          9.68.214.1         UG          0        1  tr0
9.68.214         zero.hakozaki.ibm. U           0        0  tr0
localhost        localhost          UH          6      209  lo0
#
```

As shown above, netstat -r confirms that the route is really created and set in the kernel routing table. If you need this change to remain permanent, write the route command sequence in the script /etc/rc.bsdnet.

1.3.4.2 Removing a Static Route

You can use the route command to remove static routes. See the following example:

```
# route -v delete default
so_dst: inet 0.0.0.0; RTM_DELETE: Delete Route
pid: 0, len 92, seq 1, errno 0, flags:<UP,GATEWAY>
locks:  inits:
sockaddrs: <DST,NETMASK>
 default (0)
default net default: gateway
#
```

Now that the route command is successfully completed. You can check to see whether the route was really deleted.

```
# netstat -r |grep U
Destination      Gateway            Flags  Refcnt Use       Interface
9.170.3          newton             U           4    15823  tr0
127              loopback           U           1        0  lo0
#
```

As shown, netstat -r confirms that the route is really deleted and doesn′t exist in the kernel routing table.

1.4 AIX Network-Related Boot Process

The AIX and RISC System/6000 boot processes are briefly summarized in this section. You can refer to the manual *System Management Guide: Operating System and Devices*, SC23-2525 for details.

1.4.1 Boot Process Overview

1. Read-only storage (ROS) Kernel Init Phase

   a. The on-chip sequencer (OCS) checks to see whether there are any problems with the system motherboard. Control is passed to ROS, which performs a power-on self-test (POST).

   b. ROS checks the user boot list. The first valid boot device found in the boot list is used for the system startup.

   c. When a valid boot device is found, the first record or program sector number (PSN) is checked.

   d. The boot image is read from the boot device into memory. The boot image consists of the kernel, a RAM file system, and the base customized device information.

   e. Control is passed to the kernel, which begins system initialization.

   f. Process 1 executes init, which executes phase 1 of the rc.boot script. This script is in the RAM file system now.

   Note: The init process invoked here is not the same one on which you can see your login session after the boot. This init is called a simple shell boot init and is only used during the boot process.

2. Base Device Configuration Phase (Phase 1)

   a. The boot script rc.boot calls the restbase program to build the base customized ODM database in the RAM file system from the compressed base customized data.

   b. The boot script starts the configuration manager, which accesses phase 1 configuration rules to configure the base devices.

   c. The configuration manager invokes the sys, bus, disk, small computer system interface (SCSI) and the logical volume manager (LVM) and root volume group (VG) configuration methods.

   Adapter Cards are Defined and Configured Here

   Here is the place where any adapter cards attached to the system bus are recognized by the configuration manager. If you have network adapters, such as tok0 and ent0, they are detected at this stage of the boot process.

   d. The configuration methods load the device drivers, create special files, and update the customized data in the ODM.

3. System Boot Phase (Phase 2 and 3)

   a. The init process starts the phase 2 execution of the rc.boot script.

      1) Call the ipl_varyon program to vary on the root VG.

      2) Run swapon to start paging.

      3) Mount the hard disk file systems onto the RAM file system.

      4) Merge the ODM database in the RAM file system with the ODM database in the hard disk file system by calling the mergebase program.

      5) Unmount hard disk file systems.

6) Exit the rc.boot script. At this moment, RAM init exits and the kernel is accessed.

b. After phase 2 of rc.boot, the newroot program switches from the RAM file system to the hard disk root file system. At this moment, the kernel forks the process 1 init.

Note: This init is what you see in your login session.

c. Then, the init process executes the process defined by records in the /etc/inittab file. One of the instructions in the /etc/inittab file executes phase 3 of the rc.boot script as follows:

```
brc::sysinit:/sbin/rc.boot 3 >/dev/console 2>&1 # Phase 3 of system boot
```

This includes the following steps:

1) Mount hard disk file system.

2) Invoke the configuration manager phase 2 to configure all the remaining devices.

/etc/rc.net Is Executed Here

Here is the place where the network interface devices, such as tok0 and ent0, are configured with a configuration script.

The script /etc/rc.net or /etc/rc.bsdnet is invoked. The script invoked is defined in the configuration rules stored in the ODM.

3) Use the savebase command to save the base customized data to the boot logical volume.

4) Run the diagnostics if the key switch is set to the service position.

5) Exit the rc.boot.

d. init executes other instructions in the /etc/inittab.

SRC (/usr/sbin/srcmstr) Is Started Here

The system resource controller (SRC) is started. This daemon controls subsystems defined in the ODM. For example, inetd, routed, and portmap are the subsystems or the SRC. Almost all daemons started in the script /etc/rc.tcpip and /etc/rc.nfs are subsystems of the SRC.

/etc/rc.tcpip Is Executed Here

Here is the place where the TCP/IP application servers (daemons), such as inetd, sendmail, and portmap, are started. If you are using DNS and your system is the name server, named is also started.

/etc/rc.nfs Is Executed Here

Here is the place where NFS and NIS daemons are started. They are ypsrv, ypbind, biod, nfsd, and rpc.mountd. Be aware that they need the SUN/RPC portmap, but it′s not started here.

e. Now the system is up and ready for use.

1.4.2 How Is the Script /etc/rc.net Executed?

The previous section explained that the startup script /etc/rc.net or /etc/rc.bsdnet is executed (phase 3 of the boot script /sbin/rc.boot). But if you check through the script /sbin/rc.boot, you can never find the /etc/rc.net or /etc/rc.bsdnet scripts.

Then how can AIX know what script (/etc/rc.net or /etc/rc.bsdnet) to execute, and who invokes the script. The answer is the configuration manager. In the script /sbin/rc.boot, the configuration manager cfgmgr is executed. It reads the configuration parameters from the ODM, and in the objectinet0, the value of bootup_option is examined. If it is YES, /etc/rc.bsdnet is run. If it is NO, /etc/rc.net is run.

1.4.2.1 The Configuration Manager

The cfgmgr configures devices into the system. The devices to be configured are specified in the configuration rules object class, which is part of the device configuration database, ODM.

During system boot, the cfgmgr configures all the devices that are necessary in order to use the system. From the configuration manager′s point of view, the system boot is a two-step process. The first step is called phase 1, and it begins when the kernel is brought into the system, and the boot file system is initialized. During this phase, the cfgmgr command is invoked, specifying this is phase 1 by using the -f flag. The cfgmgr command executes all of the phase 1 configuration rules; this results in the base devices being configured. After this, the phase 2 execution begins, and the cfgmgr command is called with the -s flag.

The following are the three phases of configuration rules recognized by the cfgmgr command:

Phase 1 (phase = 1)
This is the phase to configure the basic devices. In this phase, any adapter cards attached to the system bus, such as tok0 and ent0, are recognized and the corresponding entries are created in the ODM.

Phase 2 (phase = 2)
This is the phase to configure the remaining devices, which may be children of the devices configured at phase 1. For example, tr0 is a child device of the device tok0.

Phase 2 Service (phase = 3)
This is the service mode, and if the key position on the system unit is service, this phase is used.

Note: The above phases are the configuration manager′s configuration phases and are not to be confused with the boot phases.

The script /etc/rc.net or /etc/rc.bsdnet is executed during phase 2. As a result, the network interfaces are configured during phase 2.

1.4.2.2 Looking at the Config_Rules in the ODM

We can review the config_rules that the configuration manager uses for the configuration tasks. The following is the example of the rule where the interface configuration script is selected:

```
# odmget -q "phase='2' AND seq='18'" Config_Rules

Config_Rules:
        phase = 2
        seq = 18
        boot_mask = 0
        rule = "/usr/lib/methods/definet > \
                /dev/null 2>&1;opt=lsattr -E -l inet0 \
                -a bootup_option -F value\n\
        if [ $opt = \"no\" ];then\n\
                nf=/etc/rc.net\n\
        else \n\
                nf=/etc/rc.bsdnet\n\
        fi;$nf -2;x=$?;test $x -ne 0&&echo $nf failed. \
        Check for invalid commands;exit $x"
#
```

If you want to review the entire config_rules, issue the following command:

```
# odmget Config_Rules

Config_Rules:
        phase = 2
        seq = 15
        boot_mask = 0
        rule = "/etc/methods/ptynode"

Config_Rules:
        phase = 3
        seq = 15
        boot_mask = 0
        rule = "/etc/methods/ptynode"

Config_Rules:
        phase = 1
        seq = 2
        boot_mask = 0
        rule = "/usr/lib/methods/deflvm"
...
#
```

1.5 Configuration Hints and Tips

In this section, some configuration hints and tips are explained.

1.5.1 Do Not Run mktcpip or smitty mktcpip Twice

You can use the high-level command, mktcpip or the SMIT fast path smitty mktcpip command to configure a network interface. These are very easy procedures and they are advantages of the AIX, but there is a trick to using them.

Do *not* use them to change the IP address of an interface using the mktcpip command or the SMIT fast path smitty mktcpip. They are designed for the initial configuration. One pitfall is that they simply add an entry in the /etc/hosts file when they are executed. They do not replace the existing entry with a new entry. We changed the IP address of the host guru using the SMIT fast path, smitty mktcpip. As a result, we received duplicated entries for the guru, as follows:

```
# grep guru /etc/hosts
9.68.192.142    guru
9.68.214.75     guru
#
```

On the contrary, changing the hostname is done by adding an alias. In the following example, we changed the hostname from zero to zero_new using the SMIT fast path, smitty mktcpip:

```
# grep zero /etc/hosts
9.68.214.84     zero  zero_new
#
```

From the ODM point of view, the previous procedure works fine. In other words, the entry in the ODM is replaced as intended. There is only enough room for each attribute, as follows:

```
# lsattr -E -l inet0 -a hostname
hostname zero_new Host Name True
# lsattr -E -l tr0 -a netaddr
netaddr 9.68.214.84 Internet Address True
#
```

The correct procedure to change an IP address is to use the chdev command or the SMIT fast path smitty chinet command. This can also be done by running the mktcpip, which manually updates the /etc/hosts file.

Our Experience

Sometimes this pitfall is difficult to notice. If you are using DNS, all IP address retrievals go to the DNS name server and /etc/hosts is not referred to. When you change your system′s IP address, you ask the zone file update to your DNS administrator. Thus far, the DNS is running correctly, and everything seems fine.

In our environment, everything seemed fine except for one application program (sendmail). After the IP address was updated, our sendmail had a malfunction. It didn′t forward the incoming mail correctly and updated mail header address incorrectly. Only applications that refer to /etc/hosts are impacted.

1.5.2 IP Address Retrieval Priority

In the TCP/IP environment, users use host names instead of IP addresses for convenience. So, there must be a mechanism to convert a host name to the IP address. The simplest one is the conversion file called /etc/hosts. DNS and NIS are more sophisticated alternatives.

1.5.2.1 For V3.2 Environment

When you are using DNS and/or NIS, a host name to the IP address translation follows certain orders, depending on the system configuration, as follows:

NIS Client Without DNS Resolver

If the system running NIS client, ypbind, doesn′t have the /etc/resolv.conf file, the system always refers to the NIS map in the NIS server.

Note: If the NIS server has the /etc/resolv.conf file and the translation using the NIS map fails, the NIS server can refer to the DNS server next. You need to modify the /var/yp/Makefile of the NIS server in order to have the NIS server machine look up the DNS servers.

NIS Client With DNS Resolver
If the system running NIS client, ypbind, has the /etc/resolv.conf file, the system first refers to the DNS server. If the look up fails, then the system refers to the NIS server.

Note: If the NIS server has the /etc/resolv.conf file and the translation using the NIS map fails, again the DNS server is looked up by the NIS server. You need to modify the /var/yp/Makefile of the NIS server in order to have the NIS server machine look up the DNS servers.

NIS Server Without DNS Resolver
If the system, running both the NIS client and server and ypbind and ypserv, doesn′t have the /etc/resolv.conf file, the NIS map is always referred to. No other source (the DNS server and /etc/hosts) is referred to.

NIS Server With DNS Resolver
If the system, running both the NIS client and server and ypbind and ypserv, has the /etc/resolv.conf file, the DNS server is referred to first. If the DNS look up fails, then the NIS map is referred to.

With AIX V3.2, the DNS look up has higher priority than the NIS look up. If your system is only running the DNS resolver without the NIS client, the DNS server is always referred to. If the DNS look up fails, the /etc/hosts file is finally referred to. For the DNS, /etc/hosts is the final resort.

On the contrary, NIS doesn′t usually consult /etc/hosts even when the NIS map cannot satisfy the query. Therefore, a correctly configured NIS map is crucial. As mentioned above, if you need to forward the query to the DNS server after the NIS map failure, you have to modify the Makefile for the NIS maps. Add -b flags after $(MAKEDBM) in the hosts.time stanza in the /var/yp/Makefile. Refer to *AIX Version 3.2 System Management Guide: Communications and Networks*, GC23-2487 for details.

1.5.2.2 For V4.1 Environment

With AIX V4.1, you can specify the priority of look up. A new environmental variable NSORDER and a new configuration file /etc/netsvc.conf are introduced. You can use NSORDER or /etc/netsvc.conf to put the following sources in an arbitrary order:

- DNS
- NIS
- /etc/hosts

Refer to the manuals or InfoExplorer for details.

1.5.3 Host Name for Multi-Homed Host

One mysterious thing to understand is the host name for a multi-homed host. The term multi-homed host means that the system has *more than one* network interface. For example, any gateway (router) system is a multi-homed host. We also have to define the term host name clearly. Usually the term host name means processor name or system name, and there should be only one host name for each system. But in the TCP/IP world, the term host name means *interface name*. This is a very bad convention. The system name and interface name should be clearly distinguished from each other, but unfortunately both of them are referred to as the host name. But it′s not important because the system has only one interface. In this case, we can assign the same name for both the system name and the network interface name, and there are no potential problems or confusion.

1.5.3.1 Host Name

Any AIX system has its own host name. This is a system name or processor name, and not an interface name. This name is kept in the AIX kernel and also in the ODM. You can see the current host name by issuing the hostname command, as follows:

```
# hostname
grover
#
```

Also, you can use the following command to see the ODM. An odd thing is that the host name is stored as an attribute of the network device inet0.

```
# lsattr -E -l inet0 -a hostname
hostname grover Host Name True
#
```

You can change the host name with the hostname command. This command only changes the kernel and leaves the ODM unchanged, but the chdev command changes both the kernel and the ODM.

```
# hostname newton
newton
# lsattr -E -l inet0 -a hostname
hostname grover Host Name True
#
```

1.5.3.2 Interface Name

In the TCP/IP world, each network interface usually has its own IP address and interface name. The interface name is mainly for the user′s convenience.

Note: Of course there are some applications which need to refer to the host table /etc/hosts and must get an interface name. NFS is a good example along with the .rhosts authentication.

Although it′s not good convention, this interface name is usually called the host name. This name is kept both in the AIX kernel and non-volatile storage (ODM and ASCII file). You can see the interface name in the ASCII file /etc/hosts, as follows:

```
# cat /etc/hosts
127.0.0.1               loopback localhost
9.170.3.21      grover
#
```

Unfortunately, if you use the smitty mktcpip or mktcpip command for a minimum configuration, the interface name is stored in the ODM as the attribute host name of inet0. You can check the interface name in the ODM. Although a system can have more than two interfaces, ODM can only store one interface name. When you have configured more than two interfaces, the latest interface name is stored as follows:

```
# lsattr -E -l inet0 -a hostname
hostname grover Host Name True
#
```

Interface names, however, are not kept in the AIX kernel (remember the host name is kept in the AIX kernel). Instead, IP addresses are kept in the kernel and if they are referred, the kernel refers to /etc/hosts or equivalent (DNS or NIS server) and translates them to the corresponding interface names. Here is an example:

```
# netstat -i
Name  Mtu   Network     Address         Ipkts    Ierrs Opkts    Oerrs Coll
lo0   1536  <Link>                          0     0        0     0     0
lo0   1536  127         loopback            0     0        0     0     0
tr0   1492  <Link>                      31200     0     1570     0     0
tr0   1492  9.170.3     grover          31200     0     1570     0     0
#
```

In this example, we can see the host name grover, but actually the kernel kept the IP address 9.170.3.45. Use netstat -in in order to refer to the IP address.

1.5.3.3 What Happens During the Boot

During the boot process, the host name is set automatically. In the default setting, the host name is loaded into the kernel from the ODM. The host name, which is really an interface name and an attribute of the device inet0, is loaded as the system name. This is done with the cfginet command in the /etc/rc.net script. After this command is executed, you can see the system name using the hostname command, and this name is really an interface name. If the system is a multi-homed host, you will see the latest configured interface name.

If you have some shell scripts that use the hostname command to get the system name, you may have a problem.

One way to work around it is *not* to use smitty mktcpip or mktcpip. Instead, you can use smitty commodev to set an adapter card parameter. Internally, the chdev command is used. Then, use smitty mkinet to configure a network interface. Internally, the mkdev command is used. Although we don′t go into details, you can review the correct command arguments if you press F6=Command at the appropriate SMIT panel.

1.5.4 If You Mess Up the ODM

Up to now some of the high-level commands have been explained. Three more important commands, cfgmgr, mkdev and rmdev, are explained here. These commands also have the capability to update the kernel and the ODM.

When you messed up the configuration in the ODM and the kernel, in certain situations it is easier to delete all related configuration and start again from the beginning. The three new commands are useful for such a task. Be careful and consider these methods as the last resort when you get into difficult situations.

1.5.4.1 Removing a Logical Device Using the rmdev Command

Removing a logical device using the rmdev command shows how to remove the network device inet0 with the rmdev command. You can see that it really has been removed by issuing the lsattr command.

Note: If you run rmdev, your network function gets hung up immediately and will not recover until the device is created again. Be careful when you run this command and consider the risk that the applications may not be recovered.

Issue the rmdev command with the -d flag and the customized device information (more accurately, the object instance in the CuDv and CuAt object classes) is deleted from the ODM, as follows:

```
# rmdev -l inet0 -d
inet0  deleted

#lsattr -E -l inet0
lsattr: 0514-519 The following device was not found in the customized
        device configuration database:
        inet0
#
```

You can also remove the network interfaces (such as tr0) and adapter cards (such as tok0) in this order. Be aware that you cannot remove the adapter card first.

```
# rmdev -l tr0 -d
tr0  deleted

# rmdev -l tok0 -d
tok0  deleted

#
```

If you don′t use the -d flag, this command will put the specified logical device in the defined status as shown below. In the defined status, the logical device is still in the ODM, but the corresponding device driver is unloaded. Therefore, the device cannot be used until the device driver is loaded again (in other words, until it gets in the available state).

```
# rmdev -l tok0
tr0 Defined

#
```

1.5.4.2 Creating a Logical Device Using cfgmgr

The easiest way to create a logical device is to use the cfgmgr. It creates the device inet0 with all the default values. After that, you can change the device attributes again with either the SMIT or high-level command. Run the configuration manager as follows:

```
# cfgmgr -s
#
```

You can enjoy the three LEDs on the front panel flicks displaying numbers which you see during the system boot.

Note: The previous command needs a few minutes or so to complete. Do not try to interrupt this command because this command updates the ODM; an interruption may leave the ODM in an inconsistent state.

You can check whether the device was created successfully by issuing the lsattr command as follows:

```
# lsattr -E -l inet0
hostname        Host Name                        True
gateway         Gateway                          True
route           Route                            True
bootup_option no Serial Optical Network Interface True
#
```

You can invoke the configuration manager via SMIT with the first path smitty dev and choose the Install/Configure Devices Added After IPL menu.

During this procedure, the configuration manager does a lot of configuration tasks. If you specify -v, you see that events are executed as follows:

```
# cfgmgr -s -v
phase = 2
----------------------------------------------------------------------
invoking top level program -- "/etc/methods/defsys"
return code = 0
****************** stdout ***********
sys0

****************** no stderr ***********
----------------------------------------------------------------------
attempting to configure device 'sys0'
invoking /usr/lib/methods/cfgsys -2 -l sys0
return code = 0
****************** stdout ***********
bus0 bbl0
****************** no stderr ***********
----------------------------------------------------------------------
attempting to configure device 'bus0'
invoking /usr/lib/methods/cfgbus -2 -l bus0
return code = 0
****************** stdout ***********
fda0,sioka0,sa0,sa1,scsi0,siota0,sioma0,ppa0,ent0,tok0,rby0

****************** no stderr ***********
...
#
```

Important Notice for ISA Bus Adapter Users

If you are using an ISA bus adapter card, cfgmgr cannot recognize your card. You need to follow the explicit card configuration procedure through SMIT or high-level command after the card installation. Refer to 1.5.9, “ISA Bus Adapter Consideration” on page 52.

1.5.4.3 Creating a Logical Device Using the mkdev Command

Invoking the configuration manager is more than enough to create one logical device. The other alternative is to issue the mkdev command. If the device is only put in the defined state, the -l flag is enough.

```
# mkdev -l inet
ient0 Available
#
```

If the device is deleted from the ODM, you need to specify the -t flag showing which type of device you want to make. See the following example:

```
# rmdev -l inet0 -d
inet0  deleted

# mkdev -l inet0
mkdev: 0514-519 The following device was not found in the customized
        device configuration database:
        name = 'inet0'
# mkdev -t inet -l inet0
inet0 Available
#
```

You can do the same thing to other devices like tr0 and en0. In the previous example, the created device has all the default attributes. You may need to customize it with the SMIT or chdev command. You can do those customizations using the mkdev command. Refer to the manual or InfoExplorer for details. Of course, after the device removal, you can shut down and reboot the system because cfgmgr is executed during reboot.

1.5.4.4 How to Find the Valid Range and Default Values

When you set or configure the logical device attribute with the chdev command, you may need to review which value is in the valid range. In this case, use the lsattr -R command, as follows:

```
# lsattr -R -l tok0 -a ring_speed
4
16
# lsattr -R -l tok0 -a rec_que_size
20...150 (+1)
#
```

The first example shows that the token-ring adapter's ring speed can be either 4 or 16. The second example shows that the token-ring adapter's RECEIVE queue size can be 20 to 150, with an increment of 1.

If you need to know the default values, use the lsattr -D command. The following are the default values for the token-ring interface tr0:

```
# lsattr -D -l tr0
mtu       1492 Maximum IP Packet Size for This Device     True
mtu_4     1492 Maximum IP Packet Size for This Device     True
mtu_16    1492 Maximum IP Packet Size for This Device     True
remmtu    576  Maximum IP Packet Size for REMOTE Networks True
netaddr        Internet Address                           True
state     down Current Interface Status                   True
arp       on   Address Resolution Protocol (ARP)          True
allcast   on   Confine Broadcast to Local Token-Ring      True
hwloop    off  Enable Hardware Loopback Mode              True
netmask        Subnet Mask                                True
security  none Security Level                             True
authority      Authorized Users                           True
broadcast      Broadcast Address                          True
#
```

1.5.5 Two Tips for netstat

Two tips for the netstat command are given in this section.

1.5.5.1 Getting Consistent Output with Other UNIX

Since AIX Version 3.2 has been issued, we got an enhanced version of that netstat command. When you issue it with the -r flag, you will get more than enough information, as follows:

```
# netstat -r
Routing tables
Destination      Gateway            Flags     Refs     Use  Interface
Netmasks:
255.255.255.128

Route Tree for Protocol Family 2:
default          9.68.214.1         UG          0       85  tr0
9.68.214         zero.hakozaki.ibm. U           2      307  tr0
localhost        localhost          UH          1        0  lo0
#
```

Run this command with grep -U and you will a get consistent output with the other UNIX system, as follows:

```
# netstat -r | grep U
Destination      Gateway            Flags     Refs     Use  Interface
default          9.68.214.1         UG          0      107  tr0
9.68.214         zero.hakozaki.ibm. U           2      307  tr0
localhost        localhost          UH          1        0  lo0
```

Difference between V4.1 and V3.2

With V3.2, you got more than enough information. If you are not working with the Xerox Network System (XNS), you may want to cut off the output.

```
# netstat -r
Routing tables
Destination      Gateway            Flags  Refcnt Use       Interface
Netmasks:
(root node)
(0)0 ff00 0
(0)0 ffff ff00 0
(root node)

Route Tree for Protocol Family 2:
(root node)
default          grover             UG          1    46349  tr0
9.170.3          newton             U           6     2608  tr0
127              loopback           U           0     1870  lo0
(root node)

Route Tree for Protocol Family 6:
(root node)
(root node)
#
```

Run this command with grep -U and you will get the following:

```
# netstat -r | grep U
Destination      Gateway            Flags  Refcnt Use       Interface
default          grover             UG          1    46733  tr0
9.170.3          newton             U          28     7405  tr0
127              loopback           U           1     1937  lo0
#
```

1.5.5.2 Suppressing IP Address and Name Translation

One more tip is to use this command with the -n flag. This is valuable especially if you are working with the name service (DNS or NIS) debugging. If you have problems with name service and, as a result, the translation from a host name to the IP address doesn't work, this netstat command will never return. As you see in the following, this flag suppresses the IP address to the host name mapping function and shows the result in numerical form. It doesn't rely on the name service.

```
# netstat -rn | grep U
Destination      Gateway            Flags     Refs     Use  Interface
default          9.68.214.1         UG          0      121  tr0
9.68.214         9.68.214.84        U           2      307  tr0
127.0.0.1        127.0.0.1          UH          1        0  lo0
#
```

Of course, the -n flag can be used with any other flags.

```
# netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                           209     0      209     0     0
lo0   16896 127         127.0.0.1            209     0      209     0     0
tr0   1492  <Link>8.0.5a.ab.23.19           1249     0      610     0     0
tr0   1492  9.68.214    9.68.214.84         1249     0      610     0     0
#
```

1.5.6 SMIT Hints

These are small pieces of knowledge, but they may improve your working ability with AIX and TCP/IP.

1.5.6.1 How to Find the SMIT Fast Path

First, invoke SMIT and go through the menus until you get to the destination panel. This is completely a trial-and-error approach. At the SMIT destination panel, press the F8=Image key. Then, a small subpanel will open and you will find the fast path name.

```
                    PRINT SCREEN

Press Enter to save the panel image
    in the log file.
Press Cancel to return to the application.

  Current fast path:
      "chinet"

F1=Help          F2=Refresh       F3=Cancel
F8=Image         F10=Exit         Enter=Do
```

Basically the option F8=Image is for the people who need the SMIT panel hardcopy image for material such as this book.

1.5.7 Ethernet Configuration

Up to now we explained the TCP/IP configuration procedure with the token-ring because token-ring has many more parameters than Ethernet, but there are also some unique Ethernet parameters.

1.5.7.1 Minimum Configuration

Figure 14 on page 47 is the panel that you see when you run smitty mktcpip. The entry fields are almost identical to the token-ring panel. The only difference is that there is a Your CABLE Type field in place of the ring speed field.

```
                         Minimum Configuration & Startup

 To Delete existing configuration data, please use Further Configuration menus


Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                     [Entry Fields]
* HOSTNAME                                          [inoki]
* Internet ADDRESS (dotted decimal)                 []
  Network MASK (dotted decimal)                     []
* Network INTERFACE                                  en0
  NAMESERVER
           Internet ADDRESS (dotted decimal)        [9.170.1.9]
           DOMAIN Name                              [fscjapan.ibm.com]
  Default GATEWAY Address                           []
         (dotted decimal or symbolic name)
  Your CABLE Type                                    N/A                      +
  START TCP/IP daemons Now                           no                       +


F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Undo             F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 14. SMIT TCP/IP Configuration Panel (Ethernet)

Your CABLE Type

There are four choices for this field: dix, tp bnc and N/A. If your Ethernet adapter has more than one port (for example, nine pin D-shell and bnc connectors), your adapter has an internal software switch that makes the specified port available. (In other words, only one port is usable at any moment.) In this case, you have to choose an appropriate cable (port) type to use. The bnc cable type configures a coax port for 10Base2. The dix cable type configures a nine pin D-shell connector port (to attach an external transceiver) for 10base5. The tp cable type configures a twisted-pair port for 10BaseT.

Note: More accurately, dix is attached to an external transceiver via nine pin D-shell connector. It doesn′t necessarily mean that you have to use 10Base5. With an appropriate external transceiver, you can also use 10BaseT or 10Base2.

If your Ethernet adapter doesn′t have a software switch, that is, your adapter is an integrated Ethernet adapter that has only one port (for example, nine pin D-shell connector port), N/A must be chosen. The external transceiver defines the actual cable type.

Note: If the adapter is an integrated adapter, but has two ports (bnc and nine pin D-shell connectors), it has a hardware switch (jumper switch) on the card. You should choose dix or bnc and also change the jumper to an appropriate position.

All of the configuration files updated during the configuration are the same as the token-ring. If you fill the NAMESERVER fields, then the file /etc/resolv.conf is created.

1.5.7.2 Your CABLE Type Pitfalls

Traditionally, Ethernet uses an external transceiver. In the early stages of the RS/6000, this was only true for 10Base5 Ethernet. At that time, we had only the F/C 2980 Ethernet High-Performance LAN Adapter and configuration was simple because it had two ports (one for 10Base5 and the other for 10Base2). You can directly attach a T-connecter to the 10Base2 port without a transceiver.

A lot of CPU models have been introduced since that time and some of them use integrated adapters. They support 10Base5, 10Base2 and 10BaseT. Since their integrated adapter has only one 10Base5 port, you need an appropriate external transceiver for any cable type. (You need a transceiver to convert 10Base5 port to 10Base2 or 10BaseT.) Please refer to the manual *RISC/System 6000 System Overview*, GC24-2406 for details.

Now RS/6000 supports the following cable types, and some configuration procedures need adapter- and cable type-specific operation:

- 10Base5 (thick)
- 10Base2 (thin)
- 10BaseT (twisted-pair)

When you configure the Your CABLE Type field via smitty mktcpip, you can choose bnc or dix for only the F/C 2980 Ethernet High-Performance LAN Adapter. You must *always* choose N/A for any integrated Ethernet adapter. See the following table:

Table 1. Valid Selection for Your CABLE Type Field

Adapter	**10Base5**	**10Base2**	**10BaseT**	**Jumper Switch**
Integrated Ethernet Adapter for Model M20/M2A/220/230	N/A *1	N/A *1	N/A *1	No jumper
Integrated Ethernet Adapter for Model 250/41T/41W	N/A *1	N/A *1	N/A *2	No jumper
Integrated Ethernet Adapter for Model 3AT/3BT/350/360/370 (FC 4221)	dix *1	bnc	dix *1	N/ARequired (the default is bnc)
Integrated Ethernet Adapter for Model 3AT/3BT/350/360/370 (FC 4222)	Not Supported	Not Supported	N/A	No jumper
Ethernet High-Performance LAN Adapter (FC 2980)	dix *1	bnc	dix *1	No jumper
Ethernet/FDX AUI MC Adapter (FC 2992)	dix *1	dix *1	tp	No jumper

*1 An appropriate external transceiver is needed.

*2 An appropriate external or 10baseT connector (shipped with the system) is needed.

Another pitfall with the Ethernet cable type is the jumper switch on the Ethernet board (adapter). Any integrated Ethernet adapter for model 3XX (except 32X) has a jumper switch to specify the cable type. You should choose N/A at the SMIT panel, and you also have to set the jumper switch to the appropriate position. When the machine is shipped, the jumper is configured to the bnc. You have to call an IBM CE. If you try to change the cable type via SMIT only, you will get an error message.

1.5.7.3 Network Interface Device Configuration

The following is an Ethernet interface configuration panel. We don′t have much to explain since all the entry fields have already been covered by the token-ring interface explanation. This is only for your information.

```
                   Change / Show a Standard Ethernet Interface

 Type or select values in entry fields.
 Press Enter AFTER making all desired changes.

                                                           [Entry Fields]
   Network Interface Name                                en0
   INTERNET ADDRESS (dotted decimal)                    [9.170.2.9]
   Network MASK (hexadecimal or dotted decimal)         [255.255.255.0]
   Current STATE                                         up                      +
   Use Address Resolution Protocol (ARP)?                yes                     +
   BROADCAST ADDRESS (dotted decimal)                   []


 F1=Help             F2=Refresh          F3=Cancel           F4=List
 F5=Reset            F6=Command          F7=Edit             F8=Image
 F9=Shell            F10=Exit            Enter=Do
```

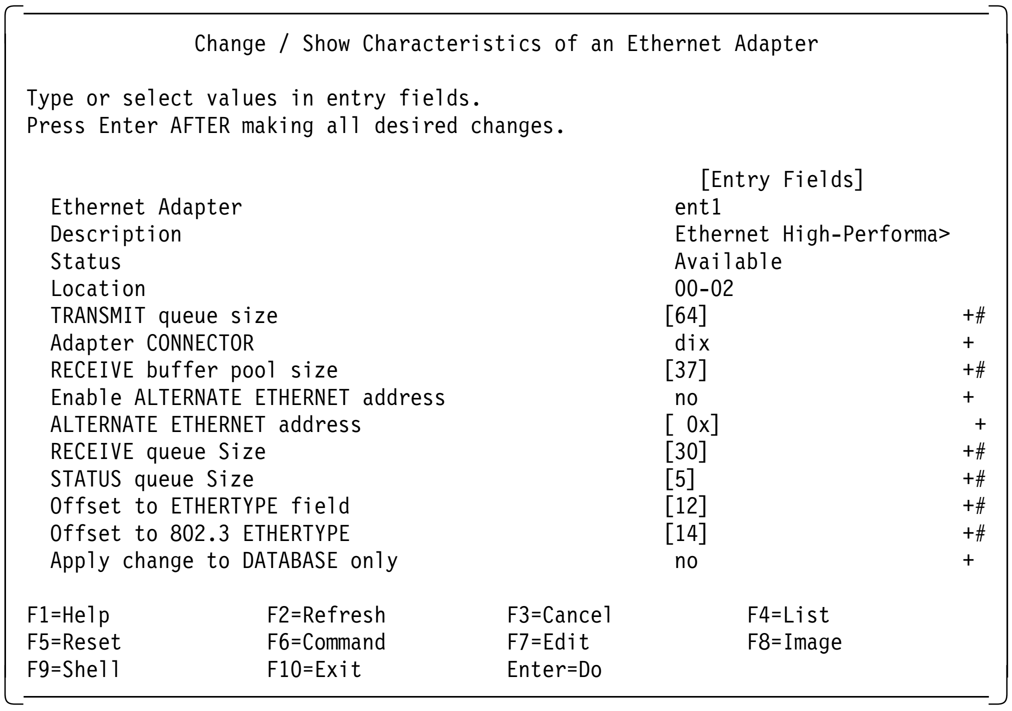
Figure 15. SMIT Network Interface Configuration Panel (Ethernet)

The ODM information about an Ethernet interface also has nothing special. You can change any of them by issuing the high-level command chdev as we have already explained.

```
# lsattr -E -l en0
mtu       1500            Maximum IP Packet Size for This Device     True
remmtu    576             Maximum IP Packet Size for REMOTE Networks True
netaddr   192.168.1.16    Internet Address                           True
state     down            Current Interface Status                   True
arp       on              Address Resolution Protocol (ARP)          True
netmask   255.255.255.128 Subnet Mask                                True
security  none            Security Level                             True
authority                 Authorized Users                           True
broadcast                 Broadcast Address                          True
#
```

1.5.7.4 Adapter Device Configuration

The Ethernet adapter configuration panel may need some explanation. It has a few new entry fields that are Ethernet-unique.



```
              Change / Show Characteristics of an Ethernet Adapter

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
  Ethernet Adapter                                  ent1
  Description                                       Ethernet High-Performa>
  Status                                            Available
  Location                                          00-02
  TRANSMIT queue size                              [64]                   +#
  Adapter CONNECTOR                                 dix                   +
  RECEIVE buffer pool size                         [37]                   +#
  Enable ALTERNATE ETHERNET address                 no                    +
  ALTERNATE ETHERNET address                       [ 0x]                   +
  RECEIVE queue Size                               [30]                   +#
  STATUS queue Size                                [5]                    +#
  Offset to ETHERTYPE field                        [12]                   +#
  Offset to 802.3 ETHERTYPE                        [14]                   +#
  Apply change to DATABASE only                     no                    +

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 16. SMIT Ethernet Adapter Configuration Panel

We will only explain those fields that are unique.

Adapter CONNECTOR
This indicates which one of the external connectors on the adapter is being used. There are two connectors: a 15-pin DIX connector selected with the value of dix and a BNC connector selected with the value of bnc. The default value is bnc.

Note: If you are configuring an integrated Ethernet adapter, you don′t see this field.

Enable ALTERNATE ETHERNET address
This specifies whether you use the alternate MAC address for this adapter. If yes, the hard-coded, burned-in MAC address is not used.

ALTERNATE ETHERNET address
This specifies the alternate MAC address. This MAC address must be managed locally, and you must confirm that there are no duplicated addresses. If you are running SNA on the same adapter, you may need to use this function.

Offset to ETHERTYPE field
For Ethernet Version 2, the data link frame header format has 6 bytes of destination address, 6 bytes of source address, and 2 bytes of type field from the beginning of the header. Therefore, the ETHERTYPE should be located at the 12th byte. *Do not* change this field.

Offset to 802.3 ETHERTYPE
For the IEEE 802.3 Ethernet, the MAC frame header format has 6 bytes of destination address, 6 bytes of source address, and 2 bytes of length field from the beginning of the header. Then the LLC header follows. Therefore, the LLC header (not exactly the ETHERTYPE field) should be located at the 14th byte. *Do not* change this field.

Note: The IEEE has another MAC address format specification that had destination and source addresses of 2 bytes in length. This 2-byte length version is almost obsolete now.

The ODM information about an Ethernet adapter can be seen (following) with the lsattr command. You can change any of them by issuing the high-level command chdev as we have already explained.

```
# lsattr -E -l ent1
bus_intr_lvl   12       Bus interrupt level               False
intr_priority  3        Interrupt priority                False
xmt_que_size   64       TRANSMIT queue size               True
bus_mem_addr   0xd0000  Bus memory address                False
dma_bus_mem    0x544000 Address of bus memory used for DMA False
bus_io_addr    0x7280   Bus I/O address                   False
dma_lvl        5        DMA arbitration level             False
bnc_select     dix      Adapter CONNECTOR                 True
use_alt_addr   no       Enable ALTERNATE ETHERNET address  True
alt_addr       0x       ALTERNATE ETHERNET address        True
rec_pool_size  37       RECEIVE buffer pool size          True
rec_que_size   30       RECEIVE queue size                True
sta_que_size   5        STATUS BLOCK queue size           True
type_field_off 12       Offset to ETHERTYPE field         True
net_id_offset  14       Offset to 802.3 ETHERTYPE         True
#
```

1.5.8 Ethernet Transceiver Configuration

When you attach the cables and the transceiver, there are some pitfalls. You should read the instructions (shipped with the transceiver) carefully.

1.5.8.1 Heart Beat (SQE)

This is for both 10Base5 and 10Base2. Our transceiver (P/N 02G7435) has a switch called SQE test on the side of the box. This switch is sometimes called the heart beat, and you can choose on or off.

This switch enables the testing function of the collision detection circuit on the adapter card. If you set this switch to the on position, the circuit is tested every time a packet is sent out. This function never affects the connectivity. You can use the adapter with the switch setting to any position. It doesn′t need to match the destination station.

Note: Our 10BaseT Transceiver (P/N 02G7429) has an SQE switch, but it is not located on the surface. Open the cover and you can see the jumper switch for the SQE. Refer to the user′s manual, which is shipped with the transceiver, for details.

1.5.8.2 Link Test

This is for 10BaseT. The transceiver has a switch called Link Test on the side of the box. This switch provides 10BaseT-unique functions and confirms the electrical connectivity between the transceiver and HUB. You can choose on or off. The HUB must also have this switch, and both switch positions must match each other. If you set the transceiver to on, you must also set the HUB to on. If there is a mismatch, you cannot communicate.

In the case of an integrated Ethernet adapter card for 10BaseT, you don′t have the transceiver (you can connect a twisted-pair cable directly to the adapter).

The link test switch is located on the adapter card as a jumper switch. When the machine is shipped, the jumper is configured to the on position.

Note: Now the Link Test function is a new part of the formal 10BaseT specification. This function was introduced into the specification later; some 10BaseT HUBs designed and manufactured by early specification *may* not have this function. In this case, you should choose the off position on your transceiver.

Our Experience

We received a question from a customer about the position to which the Link Test should be set. This customer had a 10BaseT HUB with two ports. There was a connectivity problem, and the customer had to set the RS/6000′s transceiver to on for one port and off for the other port.

The HUB had a Link Test switch inside, and all that was needed was to open the cover to fix that situation. We see this kind of problem often.

1.5.9 ISA Bus Adapter Consideration

We now have the new RS/6000 models that have ISA bus. The model 40P is an example. In this book, we are mainly concerned with the Micro Channel bus machine. There are some pitfalls for ISA bus. Maybe the biggest one is that the ISA bus adapter cards are not automatically configured.

The configuration manager recognizes the adapter cards installed in the Micro Channel slot because the Micro Channel hardware provides card detection capability by hardware. This is clearly an AIX and RS/6000 advantage provided by the configuration manager and the ODM. The ISA bus *does not* have this feature. Therefore, you have to explicitly configure any ISA bus adapter card when it is installed through SMIT or the high-level command. Rebooting your system doesn′t help.

Read the Instruction Pamphlet Shipped with the System

An instruction pamphlet, *Configuring ISA Ethernet and Token-Ring Adapters*, is shipped with the system. You must read these instructions in order to configure your adapter correctly.

Although we don′t describe the detailed configuration procedure in this book, the following is a list of hints and tips:

You Have to Run the System Management Service

In order to install your card, you have to run the system management service first and set the parameters on the card. You need the diskette and system reboot to invoke the system management service.

You Have to Run SMIT or High-Level Command Also

The system management service doesn′t update the ODM. Whenever you update or set the parameters on the card, you have to reflect the updated parameters into the ODM manually. You can use SMIT or the high-level command.

Note: For Micro Channel adapters, the configuration manager automatically takes care of consistency with the ODM.

You May Need to Set a Jumper on the Card
For the ISA token-ring adapter, you have to set a jumper switch on the card to change the ring speed (4 or 16 MB).

When You Install more than Two Adapters
You have to choose the parameters correctly to avoid any conflict among the cards. For example, if you install both ISA Ethernet and the token-ring adapter, you have to configure the following parameters explicitly to avoid any conflicts:

- Bus interrupt level
- Address of bus memory used for BIOS
- Bus I/O address
- Bus memory address
- Width of shared bus memory

We encourage you to consult the instructions *Configuring ISA Ethernet and Token-Ring Adapters* if you have a question or problem.

1.6 Network Application Configuration

We briefly explain the TCP/IP network application configuration procedures in this section. As you know, many application servers are started or stopped by the daemon inetd. Sometimes, this inetd is called the super server because it controls other TCP/IP daemons. In this book we call the daemons controlled by the inetd, the *inetd subserver*. The major reason to put other daemons under the inetd is to reduce overhead. They are started by the inetd as required and stopped by the inetd when no longer needed. They do not always need to be kept activated and we can avoid any system resource consumed by idle subservers.

You should know that not all applications are suitable for this scheme. Some applications don′t fit the inetd approach. An application that has a very heavy startup procedure, such as loading initial configuration from a file, may not be a good candidate.

AIX provides a unique process control mechanism: a system resource controller or srcmstr. The srcmstr is designed to control all background application processes running on an AIX operating system. Of course, you can choose not to use the srcmstr. With the default system configuration, the inetd is controlled by the srcmstr. Therefore any inetd subserver is indirectly under the control of srcmstr.

1.6.1 The Internet Super Server inetd

With the default system configuration, the inetd is invoked automatically during the boot process. It is started by the script /etc/rc.tcpip. Read the script carefully and you will find that the inetd is invoked by the high-level command startsrc if the srcmstr is running. With the default configuration, all the inetd configuration information is stored in the /etc/rc.tcpip, and the ODM is not used. We are not talking about the subserver configuration file /etc/inetd.conf.

Note: The script /etc/rc.tcpip uses a start function and the command startsrc is used in this function.

When the srcmstr is not available, the inetd is invoked in the same way that it is invoked at the command line.

There are four ways to configure the inetd, as summarized in the following:

Editing the Startup Script /etc/rc.tcpip
: This is the default. Inside this script, the high-level command startsrc is used, but the ODM is not involved. All information to start the inetd itself (and not the subservers) comes from this script.

Using SMIT and the Startup Script /etc/rc.tcpip
: Usually SMIT updates the ODM, but this is the exception. Any configuration update made by SMIT is written in this script.

Using High-Level Commands and the ODM
: You have to use the chssys command. This is a possible alternative, but we don′t recommend it.

Using Command Line Execution
: Exactly speaking, this is not a configuration. It only gives parameters at the run time. The parameters don′t persist during a system reboot.

We explain the previous four procedures, one by one, in this section.

1.6.1.1 Using the Startup Script /etc/rc.tcpip

Refer to A.3, "Startup Script /etc/rc.tcpip" on page 341 for the complete list of this script. This is where the inetd is invoked. If the following lines are commented out, your inetd cannot start during the system boot:

```
# Start up socket-based daemons
start /usr/sbin/inetd ″$src_running″
```

As you see in the previous, the start is a function of ksh. If you could see the beginning of this script, you would also find the definition of this function as follows:

```
# start -
#       starts daemons using either src or command-line method
# args:
#       $1: pathname of daemon
#       $2: non-null if we should use src to start the daemon
#       $3: any arguments to pass it
#
start()
{
     ....
}
```

This script first checks to see whether the srcmstr is running. If it is running, set the variable src_running to 1. Again it checks to see if the srcmstr can accept the high-level command, lssrc -s inetd, to ensure the functionality of the srcmstr. Notice that it does not matter if the inetd is running at this moment. The script checks the exit status of the lssrc command. If the check fails, the script /etc/rc.tcpip exits. In such a case, no daemons are invoked. If the srcmstr passed the above check, the daemons are started by the function start(). Since start() finds that the variable src_running is set, it uses the startsrc command internally.

Note: Although the startsrc command is used, all the customized information is stored in this script unless you update the ODM intentionally. You need to use the chssys or odmchange command to update the ODM for inetd.

If the srcmstr is not running, the variable src_running is not set. Therefore, the start just starts all the daemons as they are invoked at the command line and doesn′t use the startsrc command.

If you need to add some option flags to the inetd, update the corresponding lines as follows. This puts the inetd in the debug mode and the inetd sends debug messages to syslogd. Refer to the manual or InfoExplorer for the available flags.

```
# Start up socket-based daemons
start /usr/sbin/inetd ″$src_running″ ″ -d ″
```

We recommend this approach if you need to do something with the inetd. This is because it is easy to do and easy to check or fix.

1.6.1.2 Using SMIT and the Startup Script /etc/rc.tcpip

You can configure the inetd via SMIT. Please note that this procedure provides the SMIT front end panels to the previous procedure using the startup script /etc/rc.tcpip. In other words, you will use SMIT instead of another editor. This is the only advantage of this procedure.

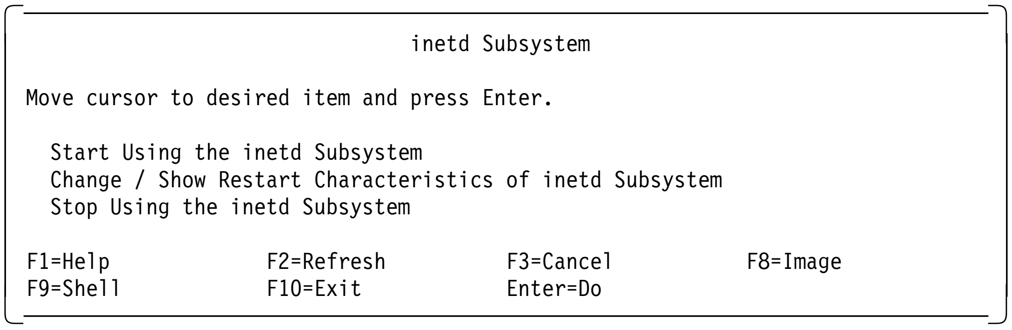
Again, with SMIT, nothing is updated in the ODM. Since the inetd is controlled by the srcmstr and the srcmstr uses the ODM, the inetd has the corresponding entry in the ODM. Changes *are not* written in the ODM. Any operations described in this section are reflected in the startup script /etc/rc.tcpip instead of the ODM. Therefore, the changes are made permanent.

Follow the steps below:

1. Issue the following command:

   ```
   # smitty inetdsubsys
   ```

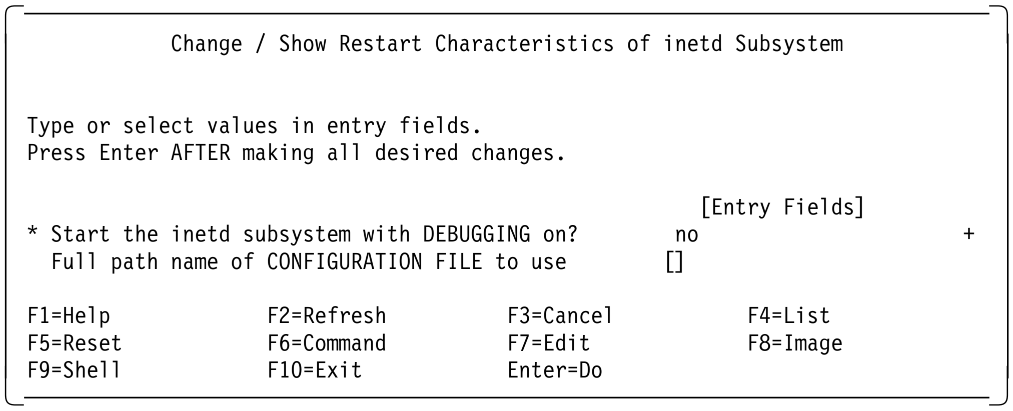
   You will get Figure 17. Each submenu provides very simple and obvious instructions.



```
                                inetd Subsystem

Move cursor to desired item and press Enter.

  Start Using the inetd Subsystem
  Change / Show Restart Characteristics of inetd Subsystem
  Stop Using the inetd Subsystem

F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 17. SMIT inetd Subsystem Configuration Menu Panel

2. If you choose the option **Change / Show Restart Characteristics** from the inetd Subsystem panel, you will see the following:



```
           Change / Show Restart Characteristics of inetd Subsystem


Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Start the inetd subsystem with DEBUGGING on?       no                    +
  Full path name of CONFIGURATION FILE to use       []

F1=Help            F2=Refresh         F3=Cancel          F4=List
F5=Reset           F6=Command         F7=Edit            F8=Image
F9=Shell           F10=Exit           Enter=Do
```

Figure 18. SMIT inetd Subsystem Configuration Panel

The changes made in this panel will be available immediately. If you press the Enter key, inetd is stopped and restarted.

Start the inetd subsystem with DEBUGGING on
If you set this field to yes, the inetd generates debugging messages for syslog. Therefore, you have to configure syslogd before enabling this option.

Full path name of CONFIGURATION FILE to use
With the default, the inetd refers to and loads the configuration information from the file /etc/inetd.conf. You can specify an alternative configuration file here.

If you specify yes and /etc/new_inetd.conf for the above fields, the script /etc/rc.tcpip is updated as follows:

```
# Start up socket-based daemons
start /usr/sbin/inetd ″$src_running″ ″-d /etc/new_inetd.conf″
```

1.6.1.3 Using High-level Commands and the ODM

You need high-level commands when the inetd is controlled by the srcmstr. Use the following startsrc command to start:

```
# startsrc -s inetd
0513-059 The inetd Subsystem has been started. Subsystem PID is 18392.
#
```

Use the stopsrc command to stop it as follows:

```
# stopsrc -s inetd
0513-044 The stop of the /usr/sbin/inetd Subsystem was completed successfully.
#
```

Even when you use SMIT or the editor to update /etc/rc.tcpip, you should use the startsrc and stopsrc commands for interactive operation. You can check the current status with the lssrc command. If you need detailed information, such as what subservers are available, use -ls instead of the -s flag:

```
# lssrc -s inetd
Subsystem         Group            PID     Status
 inetd            tcpip            18392   active
#
```

The most important difference is to update the configuration. You can use the chssys command for this purpose. If you need the debug mode for the inetd and new subserver configuration file /etc/new_inetd.conf, you can issue the following command:

```
# chssys -s inetd -a ″-d /etc/new_inetd.conf″
0513-077 Subsystem has been changed.
#
```

With this command, the change is written in the ODM only. Unfortunately there are no simple ways to review the ODM contents. To see if the change was successful, use the odmget command. You should get something similar to the following:

```
# odmget -q ′subsysname=inetd′ SRCsubsys | grep cmdargs
        cmdargs = ″-d /etc/new_inetd.conf″
#
```

One pitfall that you should expect is that the customized ODM data has a higher priority than that of the /etc/rc.tcpip and the command line arguments. Also, If you make a change to the /etc/rc.tcpip as follows, what happens?

```
# Start up socket-based daemons
start /usr/sbin/inetd ″$src_running″ ″/etc/another_new_inetd.conf″
```

The srcmstr first loads the configuration from the ODM and then, if it finds more configuration information in the /etc/rc.tcpip, it just appends the new information to the ODM information. As a result, the inetd is invoked as follows:

```
# inetd -d /etc/new_inetd.conf /etc/another_new_inetd.conf
```

Since the inetd gets incorrect arguments, the inetd may be put in an inconsistent state and your system may have problems. Thus, we don′t recommend this approach because it is difficult to debug.

1.6.1.4 Using Command-Line Execution

If you are not using the srcmstr, you don′t need to use any high-level commands. Just use the command-line interface to start and stop. Start the inetd with the necessary arguments:

```
# inetd -d /etc/new_inetd.conf
#
```

For example, you can check the status with the ps command, as follows:

```
# inetd
# ps -ef | grep inetd | grep -v grep
    root 15302     1   0 17:16:57      -  0:00 inetd -d /etc/new_inetd.conf
#
```

To stop the inetd, use the kill command:

```
# kill 18398
#
```

It is not recommended that you use high-level commands and the command line interface simultaneously. The srcmstr has some control mechanisms on its subsystems. For example, some subsystems (including the inetd) cannot be started twice. It is defined in the ODM that many instances are invoked simultaneously. See the following example:

```
# startsrc -s inetd
0513-059 The inetd Subsystem has been started. Subsystem PID is 15310.
# startsrc -s inetd
0513-029 The inetd Subsystem is already active.
Multiple instances are not supported.
#
```

For the command line invocation, there are no such restrictions (protections). Refer to the following example:

```
# inetd
# inetd
# inetd
# ps -ef | grep inetd | grep -v grep
    root  5585     1   0 17:21:23      -  0:00 inetd
    root 15310  2684   0 17:21:03      -  0:00 /usr/sbin/inetd /etc/new_inetd.conf
    root 15571     1   0 17:21:25      -  0:00 inetd
    root 16341     1   0 17:21:27      -  0:00 inetd
#
```

This can be used as a debug method. If your daemon, such as inetd, doesn′t start, invoke it from command line. If it successful, the problem is due to the srcmstr. If it doesn′t start, the problem is not the srcmstr.

1.6.2 The inetd Subservers

In this section the inetd subservers are explained. The inetd subservers are daemons controlled by the inetd. These subservers are designed to be invoked and stopped by the inetd, and the inetd needs subserver configuration information to start them. There are two places where the subserver configuration is stored: one is the ASCII file /etc/inetd.conf. and the other is the ODM. Thus, the configuration or customization task of inetd subservers is to update these files.

Difference between V4.1 and V3.2

With AIX V4.1, the inetd no longer uses the ODM. Some inetd- and ODM-related commands, such as inetimp and inetserv, are obsolete. Only the ASCII file /etc/inetd.conf is necessary. For compatibility purposes, you can configure V4.1 inetd as well as the V3.2 inetd. For such a case, you must install the fileset bos.compat.net. This section is written for V3.2 and V4.1 in which this fileset is installed.

Usually the srcmstr starts the inetd. Therefore, the inetd is given the necessary information for its subservers (by the srcmstr). This is why the ODM is involved. The srcmstr retrieves the inetd subserver information from the ODM and passes it to the inetd. In other words, if you don′t use the srcmstr, the ODM is not referred to (although this is unusual for AIX). If the inetd is invoked at the command line, it reads the ASCII file.

It is a good practice to update the ASCII file first, and then reflect the update to the ODM. This is because the ASCII file /etc/inetd.conf is easy to review and update. There is an excellent explanation (comment) in the following file /etc/inetd.conf:

```
########################################################################
##
##                 Internet server configuration database
##
##      Services can be added and deleted by deleting or inserting a
##      comment character (ie. #) at the beginning of a line  If inetd
##      is running under SRC control then the "refresh -s inetd" command
##      needs to be executed for inetd to re-read the inetd.conf file.
##
##      NOTE: The TCP/IP servers do not require SRC and may be started
##      by invoking the service directly (i.e. /etc/inetd). If inetd
##      has been invoked directly, after modifying this file, send a
##      hangup signal, SIGHUP to inetd (ie. kill -1 "pid_of_inetd").
##
##      NOTE: The services with socket type of "sunrpc_tcp" and "sunrpc_udp"
##      require that the portmap daemon be running.
##      Also please use ## to designate comments in this file so that
##      the smit commands can edit this file correctly.
##
```

For the complete list of the /etc/inetd.conf, refer to A.6, "Configuration File /etc/inetd.conf" on page 347.

Of course, if you are sure that the srcmstr is always available (active), you can update or just maintain the ODM. You can update the ODM using the inetserv command without changing the ASCII file. However, we *don't* recommend this approach. Again we emphasize that the ODM is difficult to fix if it is corrupted. Refer to the manual or InfoExplorer for details about this command. This inetserv command is a low-level command. A low-level command is intended to be used internally by a high-level command.

Note: The inetd subserver information is stored at InetServ Object Class in the ODM.

Also, it is possible to update the ODM first and then propagate the change to the ASCII file using the inetserv command. However, we never recommend these approaches because there are no reasons to do so.

1.6.2.1 Integrated inetd Subservers

Basically, each inetd subserver is a daemon or a program. Some inetd subservers are not completely independent daemons or programs, and they are integrated with the inetd. The purpose of the inetd is to save the system resource when the inetd subservers are not used. If an inetd subserver is small enough, the forking and executing costs of the subserver are not negligible. To avoid the overhead, those subservers are contained within the inetd. When you see the /etc/inetd.conf, those subservers have the term internal in the path name field.

```
...
echo    stream  tcp     nowait  root    internal
discard stream  tcp     nowait  root    internal
chargen stream  tcp     nowait  root    internal
daytime stream  tcp     nowait  root    internal
time    stream  tcp     nowait  root    internal
echo    dgram   udp     wait    root    internal
discard dgram   udp     wait    root    internal
chargen dgram   udp     wait    root    internal
daytime dgram   udp     wait    root    internal
...
```

The configuration procedure of those integrated subservers is the same as the other subservers.

1.6.2.2 Using ASCII File /etc/inetd and /etc/services

You can update the ASCII file /etc/inetd.conf first and then propagate the update to the ODM with this procedure.

1. Edit the file /etc/inetd.conf with your favorite editor. To enable a subserver (daemon), remove the comment mark (#). If necessary, specify the flag(s) in the right most column. The following is an example:

   ```
   ...
   ftp     stream  tcp     nowait  root    /usr/sbin/ftpd          ftpd
   telnet  stream  tcp     nowait  root    /usr/sbin/telnetd       telnetd -s
   shell   stream  tcp     nowait  root    /usr/sbin/rshd          rshd
   ...
   ```

2. Next, reflect the /etc/inetd.conf into the ODM. This can be done by issuing the inetimp command as follows:

   ```
   # inetimp
   #
   ```

   Note: At this moment, both /etc/inetd.conf and /etc/services are imported to the InetServ object class.

3. Finally, you need to restart or refresh the inetd in order to have it read the updated information. The srcmstr supports the refresh command.

   ```
   # refresh -s inetd
   0513-095 The request for subsystem refresh was completed successfully.
   #
   ```

   If your inetd is managed by the command line execution, you need the following operation to refresh the inetd:

   ```
   # ps -ef | grep inetd | grep -v grep
       root  5585     1   0 17:21:23      -  0:02 inetd
   # kill -1 5585
   #
   ```

If you are doubtful that the ODM is correctly updated, you can use the inetserv command as below. The following flags make this command display the ODM information with the compatible format of /etc/inetd.conf:

```
# inetserv -s -I -X
Service Socket  Protocol Wait/  User    Server Program  Server Program
Name    Type            Nowait                          Arguments
------------------------------------------------------------------------------
echo    stream  tcp     nowait  root    internal
echo    dgram   udp     wait    root    internal
discard stream  tcp     nowait  root    internal
discard dgram   udp     wait    root    internal
daytime stream  tcp     nowait  root    internal
daytime dgram   udp     wait    root    internal
chargen stream  tcp     nowait  root    internal
chargen dgram   udp     wait    root    internal
ftp     stream  tcp     nowait  root    /usr/sbin/ftpd ftpd
telnet  stream  tcp     nowait  root    /usr/sbin/telnetd telnetd -s
time    stream  tcp     nowait  root    internal
time    dgram   udp     wait    root    internal
ttdbserverd     sunrpc_tcp tcp  wait    root    /usr/dt/bin/rpc.ttdbserverd rpc.ttdbserverd 100083 1
rstatd  sunrpc_udp udp  wait    root    /usr/sbin/rpc.rstatd rstatd 100001 1-3
rusersd sunrpc_udp udp  wait    root    /usr/lib/netsvc/rusers/rpc.rusersd rusersd 100002 1-2
rwalld  sunrpc_udp udp  wait    root    /usr/lib/netsvc/rwall/rpc.rwalld rwalld 100008 1
sprayd  sunrpc_udp udp  wait    root    /usr/lib/netsvc/spray/rpc.sprayd sprayd 100012 1
pcnfsd  sunrpc_udp udp  wait    root    /usr/sbin/rpc.pcnfsd pcnfsd 150001 1-2
cmsd    sunrpc_udp udp  wait    root    /usr/dt/bin/rpc.cmsd cmsd 100068 2-4
exec    stream  tcp     nowait  root    /usr/sbin/rexecd rexecd
login   stream  tcp     nowait  root    /usr/sbin/rlogind rlogind
shell   stream  tcp     nowait  root    /usr/sbin/rshd rshd
ntalk   dgram   udp     wait    root    /usr/sbin/talkd talkd
dtspc   stream  tcp     nowait  root    /usr/dt/bin/dtspcd /usr/dt/bin/dtspcd
#
```

In the previous procedure, you don't need to update the /etc/services file. But, this doesn't mean that /etc/services is not referred to. The /etc/inetd.conf file doesn't include the port number information. When the inetd opens the ports for its subservers, it has to map the subservers to the appropriate socket ports. Then, when you issue the inetimp command, /etc/services is also referred to, and the port number information is stored in the InetServ object class. You can retrieve the equivalent information with the /etc/services file from the ODM using the inetserv command, as follows:

```
# inetserv -s -S -X
Service         Port/Protocol   Aliases
------------------------------------------------------------------------------
tcpmux          1/tcp            # TCP Port Service Multiplexer
tcpmux          1/udp            # TCP Port Service Multiplexer
compressnet     2/tcp            # Management Utility
compressnet     2/udp            # Management Utility
compressnet     3/tcp            # Compression Process
compressnet     3/udp            # Compression Process
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
...
afs3-rmtsys     7009/udp                # Remote Cache Manager Service
graPHIGS        8000/tcp                # graPHIGS Remote Nucleus
x_st_mgrd       9000/tcp                # IBM X Terminal
man             9535/tcp
man             9535/udp
isode-dua       17007/tcp
isode-dua       17007/udp
dtspc           6112/tcp
cppbrowse       4242/tcp
#
```

1.6.2.3 Using SMIT

You can use SMIT for the inetd subserver configuration. Again, SMIT only provides front end panels from the previous procedure.

1. Issue the following command:

   ```
   # smitty inetdconf
   ```

   Then, you will see the following panel. Each submenu provides very simple and obvious instructions. Then, we only need to explain the Add an inetd Subserver option.

```
                               inetd Subservers

Move cursor to desired item and press Enter.

  List All inetd Subservers
  Add an inetd Subserver
  Change / Show Characteristics of an inetd Subserver
  Remove an inetd Subserver

F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 19. SMIT inetd Subserver Configuration Menu Panel

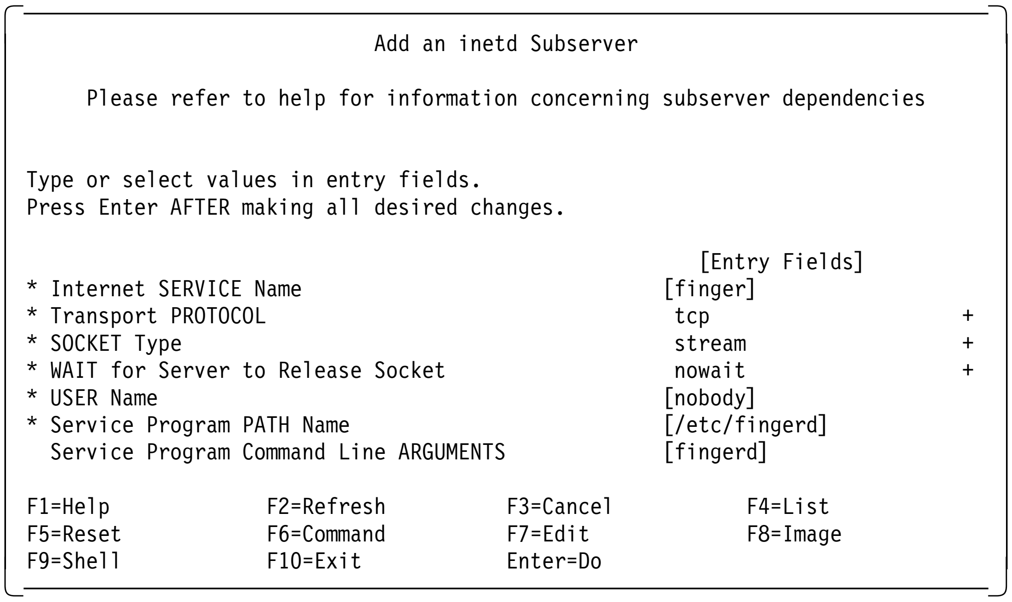
2. You will see the following panel. With this option you can make a subserver available for use. You can specify any subserver, currently commented out, that is already written in the configuration file /etc/inetd.conf.

```
                            Add an inetd Subserver

     Please refer to help for information concerning subserver dependencies


Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* Available Subservers                                [] 

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 20. SMIT inetd Subserver Selection Panel

3. Then, you will see the following panel.

   If you compare this panel with the /etc/inetd.conf file, the meaning of each entry field is obvious. You can change it if you need to change some of the fields.



```
                         Add an inetd Subserver

     Please refer to help for information concerning subserver dependencies


Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                     [Entry Fields]
* Internet SERVICE Name                             [finger]
* Transport PROTOCOL                                 tcp               +
* SOCKET Type                                        stream            +
* WAIT for Server to Release Socket                  nowait            +
* USER Name                                         [nobody]
* Service Program PATH Name                         [/etc/fingerd]
  Service Program Command Line ARGUMENTS            [fingerd]

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 21. SMIT inetd Subserver Configuration Panel

If you press the Enter key on the previous panel, the comment mark (#) is removed from the corresponding line of the /etc/inetd.conf file and the other fields are updated as you enter information. Then the changes are propagated to the ODM, and the inetd is refreshed.

Note: If the inetd is not invoked by the srcmstr, this panel will fail.

An inetd subserver is invoked by the inetd when a request comes in. This panel doesn′t start the subservers, but it makes it possible for the inetd to start it.

1.6.3 Other Network Subsystems (Servers)

As we mentioned, some servers (daemons) are not controlled by the inetd. Their configuration or customization procedures and starting or stopping procedures are completely identical to the inetd. For your convenience, the summary is shown as follows. For details, refer to 1.6.1, “The Internet Super Server inetd” on page 53.

Editing the Startup Script /etc/rc.tcpip
This is the default. Inside this script, the high-level command startsrc is used. The ODM is not involved. All the information to start the subsystem (server daemon) comes from this script.

Using SMIT and the Startup Script /etc/rc.tcpip
Usually SMIT updates the ODM, but this is an exception. Any configuration updates made by SMIT are written in this script.

Using High-level Commands and the ODM
You have to use the chssys command. This is a possible alternative, but we *don′t* recommend it.

Using Command Line Execution
Exactly speaking, this is not a configuration. It only gives parameters at the run time. The parameters don′t persist during a system reboot.

In this section, we only describe the procedure using SMIT and the /etc/rc.tcpip file.

1.6.3.1 Using SMIT and the Startup Script /etc/rc.tcpip

Do not forget that nothing is updated in the ODM with SMIT. Any operations described in this section are reflected in the startup script /etc/rc.tcpip. The following are used to make permanent changes:

1. Issue the following command and invoke SMIT:

   ```
   # smitty otherserv
   ```

   Then, you will see the following panel. Other subsystems are available through this selection menu.

```
                              Other Available Services

Move cursor to desired item and press Enter.

  Super Daemon (inetd)
  syslogd Subsystem
  routed Subsystem
  gated Subsystem
  named Subsystem
  rwhod Subsystem
  timed Subsystem
  portmap Subsystem (information only)

F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

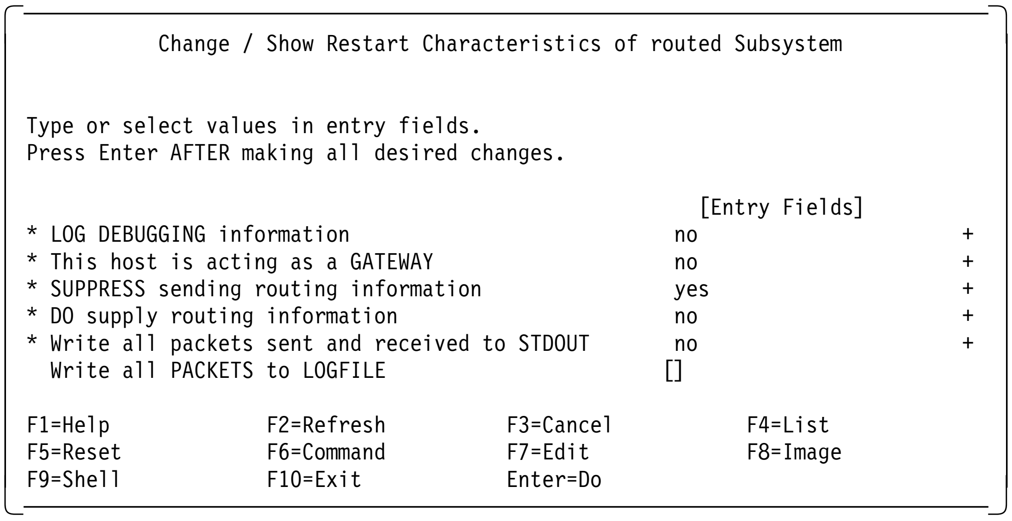
Figure 22. SMIT TCP/IP Subsystem (Application) Selection Menu

2. After you select a subsystem (daemon), you will see the following panel. In this example we selected routed (each subsystem has this same submenu).

```
                                  routed Subsystem

Move cursor to desired item and press Enter.

  Start Using the routed Subsystem
  Change / Show Restart Characteristics of routed Subsystem
  Stop Using the routed Subsystem

F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 23. SMIT TCP/IP Subsystem Menu (Routed)

3. Select **Change / Show Restart Characteristics of routed Subsystem**, and you will see the following panel. Each entry field corresponds to the option or flag that is given at the startup time. These entry fields correspond to the command line arguments when you invoke the daemons from the command line. Of course, you can directly edit the /etc/rc.tcpip file.



```
           Change / Show Restart Characteristics of routed Subsystem


Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* LOG DEBUGGING information                           no                     +
* This host is acting as a GATEWAY                    no                     +
* SUPPRESS sending routing information                yes                    +
* DO supply routing information                       no                     +
* Write all packets sent and received to STDOUT       no                     +
  Write all PACKETS to LOGFILE                       []

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 24. SMIT TCP/IP Subsystem (Routed) Configuration Panel

Remember that the change or configuration made from the SMIT panel is only reflected in the script /etc/rc.tcpip and is not introduced in the ODM (the object class SRCsubsys). Before you update the configuration, /etc/rc.tcpip has the following description for the routed Subsystem:

```
# Start up routing daemon (only start ONE)
#start /usr/sbin/routed "$src_running" -q
#start /usr/sbin/gated "$src_running"
```

The flag -q corresponds to the SUPPRESS sending routing information entry field.

1.7 The ODM

The object data manager (ODM) stores the data in the ODM database. It maintains system configuration, devices and vital product data. It also provides a reliable database facility for system management, which includes commands, C-language subroutines, and an editor (odme) to create and manipulate the ODM database. The system data that is managed by the ODM includes the following:

- Device Configuration Information
- Display Information for SMIT
- Vital Product Data for Install/Update of LPPs
- Communications Configuration Data
- System Resource Information

The previous data is organized in a structure called the object class.

1.7.1 The ODM Database

The ODM database is built on the AIX JFS. It is a collection of files located under the /etc/objrepos directory. This directory is set to the shell variable ODMDIR. You can confirm it as follows:

```
# echo $ODMDIR
/etc/objrepos
#
```

See the following ODM example:

```
# cd /etc/objrepos
# ls
CDiagDev        DSMenu          PdCn            errnotify       sm_cmd_hdr.vc
Config_Rules    FRUB            PdDv            history         sm_cmd_opt
CuAt            FRUs            PdDv.vc         history.vc      sm_cmd_opt.vc
CuAt.vc         InetServ        SRCnotify       inventory       sm_menu_opt
CuDep           MenuGoal        SRCodmlock      inventory.vc    sm_menu_opt.vc
CuDv            PDiagAtt        SRCsubsvr       lpp             sm_name_hdr
CuDvDr          PDiagAtt.vc     SRCsubsys       lpp.vc          sm_name_hdr.vc
CuVPD           PDiagDev        SWservAt        lvm_lock        sna
DAVars          PDiagDev.vc     SWservAt.vc     product
DSMOptions      PdAt            TMInput         product.vc
DSMOptions.vc   PdAt.vc         config_lock     sm_cmd_hdr
#
```

Each file stands for an object class and includes all the objects that belong to the object class. For example, the object class Config_Rules is the collection of the object's configuration rules.

/etc/objrepos
Customized object classes and software VPD (LPP root part) are located here. Also, the symbolic links to predefined object classes and SMIT object classes are placed here.

/etc/lib/objrepos
Predefined object classes and software VPD (LPP /usr part) are located here. Also, the SMIT menus and controls are located here.

/usr/share/lib/objrepos
Software VPD (LPP /usr/share part) is located here.

1.7.2 Object Class, Object and Object Descriptor

The object class is a collection of objects with the same semantic definition. You can review the object class definition as follows:

```
# odmshow Config_Rules
class Config_Rules {
        short phase;                                /* offset: 0xc ( 12) */
        short seq;                                  /* offset: 0xe ( 14) */
        long boot_mask;                             /* offset: 0x10 ( 16) */
        char rule[256];                             /* offset: 0x14 ( 20) */
        };
/*
        columns:        4
        structsize:     0x114 (276) bytes
        data offset:    0x210
        population:     19 objects (19 active, 0 deleted)
*/


#
```

As you can see, the definition is written by the object descriptor. The object descriptor is made up of elements of the object that describe the object. In this example, phase, seq, boot_mask, and rule are the object descriptors (short, long, and char are descriptor types). The following are the seven descriptor types:

short
long

- char
- vchar
- binary
- link
- method

Using these descriptor types mean that any object that belongs to the Config_Rules object class must have the above data structure. You can see that the object conforms to the object descriptors of this object class. See the following example of the object in this object class:

```
# odmget -q ″seq=1″ Config_Rules

Config_Rules:
        phase = 1
        seq = 1
        boot_mask = 0
        rule = ″/etc/methods/defsys″
#
```

This ODM data scheme is quite flexible and provides a reliable, object-oriented database facility. This provides a means for users to create their own database. The configuration manager (cfgmgr) uses the ODM as the database to store configuration rules that the cfgmgr must execute. The object class Config_Rules is designed for this purpose. Therefore, the meaning of each object descriptor such as phase or sequence are user (cfgmgr) dependent.

1.7.3 Device and the ODM

The ODM is used to store the device configuration information. The device configuration information is divided and stored in several object classes. The following acronyms are defined:

PdDv	Predefined device object class
PdAt	Predefined device attribute object class
PdCn	Predefined connection object class
CuDv	Customized device object class
CuAt	Customized device attribute object class
CuDep	Customized device dependency object class
CuVPD	Customized vital product data object class
CuDvDr	Customized device driver object class

Note: Any device that will be configured must be defined in the object class PdDv, PdAt and PdCn. In other words, you cannot configure any device that is not defined in the predefined object class.

In this section, we explain these object classes using a network interface device tr0. Up to now we referred to the command lsattr as the tool to review the ODM. We show this command usage below, and you will see how this data is stored in the ODM:

```
# lsattr -E -l tr0
mtu       1492            Maximum IP Packet Size for This Device     True
mtu_4     1492            Maximum IP Packet Size for This Device     True
mtu_16    1492            Maximum IP Packet Size for This Device     True
remmtu    576             Maximum IP Packet Size for REMOTE Networks True
netaddr   9.68.214.82     Internet Address                           True
state     up              Current Interface Status                   True
arp       on              Address Resolution Protocol (ARP)          True
allcast   on              Confine Broadcast to Local Token-Ring      True
hwloop    off             Enable Hardware Loopback Mode              True
netmask   255.255.255.128 Subnet Mask                                True
security  none            Security Level                             True
authority                 Authorized Users                           True
broadcast                 Broadcast Address                          True
#
```

If you need to know the details and exact definition of each object class and descriptor, refer to the manual or InfoExplorer. You can find a complete explanation, but it may not be easy to read.

1.7.4 Predefined Device Information

Any device that is configured must have been in predefined device object classes in the ODM. You cannot configure a device that is not predefined in the ODM. If your predefined device object class information is corrupted, you will have serious problems. These object classes are supplied with AIX, and you may need to restore or reinstall AIX to fix the corrupted data. You can review all the predefined devices with the high-level command lsdev -P, as follows:

```
class          type          subclass   description

logical_volume vgtype        vgsubclass N/A
logical_volume lvtype        lvsubclass N/A
lvm            lvdd          lvm        LVM Device Driver
aio            aio           node       Asynchronous I/O
pty            pty           pty        Asynchronous Pseudo-Terminal
adapter        ppa           sio        Standard I/O Parallel Port Adapter
adapter        ppa_iod       sio        Standard I/O Parallel Port Adapter
printer        opp           parallel   Other parallel printer
...
```

Refer to the manual or InfoExplorer for details.

1.7.4.1 Predefined Device PdDv

The predefined device is a kind of template. It defines all the supported devices. In other words, any devices that will be configured must be defined in this object class. See the following example. This is the example of a token-ring interface device, which we continuously refer to through this section. Sometimes we refer to token-ring adapter instead.

```
# odmget -q ″type=′tr′″ PdDv

PdDv:
        type = ″tr″
        class = ″if″
        subclass = ″TR″
        prefix = ″tr″
        devid = ″″
        base = 0
        has_vpd = 0
        detectable = 0
        chgstatus = 1
        bus_ext = 0
        fru = 0
        led = 1409
        setno = 110
        msgno = 21
        catalog = ″devices.cat″
        DvDr = ″if_tr″
        Define = ″/usr/lib/methods/defif″
        Configure = ″/usr/lib/methods/cfgif″
        Change = ″/usr/lib/methods/chgif″
        Unconfigure = ″/usr/lib/methods/ucfgif″
        Undefine = ″/usr/lib/methods/udefif″
        Start = ″″
        Stop = ″″
        inventory_only = 0
        uniquetype = ″if/TR/tr″
#
```

As you know, each token-ring interface device has a number in this name (for example tr0 or tr1). Notice that this object class PdDv only defines the common object descriptors among any token-ring interface device and doesn′t hold each interface-specific (customized) information. Therefore, in this example, the object is identified by the object descriptor type (tr).

Of course all the devices we explained, such as network device, network interface device and adapter device, have corresponding templates in this object class.

1.7.4.2 Predefined Device Attribute PdA

The device may have attributes that can be configured. This object class defines such customizable device attributes. Default parameters are provided in this object class (remember as an output of the command lsattr). Now you are seeing where those attributes, listed by the lsattr command, come from. You can see how many attributes are in the predefined, as follows. Notice that the devices are identified by the descriptor uniquetype:

```
# odmget -q ″uniquetype=′if/TR/tr′″ PdAt | grep attribute
        attribute = ″mtu″
        attribute = ″mtu_4″
        attribute = ″mtu_16″
        attribute = ″remmtu″
        attribute = ″netaddr″
        attribute = ″state″
        attribute = ″arp″
        attribute = ″allcast″
        attribute = ″hwloop″
        attribute = ″netmask″
        attribute = ″if_keyword″
        attribute = ″security″
        attribute = ″authority″
        attribute = ″broadcast″
#
```

The above list matches the output of the lsattr command. You can see the details of each attribute, as follows:

```
# odmget -q ″uniquetype=′if/TR/tr′ AND attribute=′mtu′″ PdAt

PdAt:
        uniquetype = ″if/TR/tr″
        attribute = ″mtu″
        deflt = ″1492″
        values = ″60-8500,1″
        width = ″″
        type = ″R″
        generic = ″DU″
        rep = ″nr″
        nls_index = 2
#
```

Now you know where the mtu default value (1492) is stored and where the range 60 - 17792 is defined. This is what you can refer to with the lsattr command, as follows:

```
# lsattr -R -l tr0 -a mtu
60...8500 (+1)
#
```

1.7.4.3 Predefined Connection PdCn

This object class contains information that is necessary in order to connect terminal devices to their intermediate devices. A terminal device is a device that doesn′t have a child. Examples are disks, printers, display terminals and keyboards. An intermediate device is a device that has both a parent and a child. An example is adapter cards. A pseudo device can be an intermediate or a terminal device. Only intermediate devices can have instances in this object class. Network interface is both a pseudo device and an intermediate device because the TCP/IP network is connected to it. The descriptor connkey represents a subclass of devices to which it can be connected. Since the network interface doesn′t have any physical port or some place where the TCP/IP network is connected, the descriptor connwhere is blank, as follows:

```
# odmget -q ″uniquetype=′if/TR/tr′″ PdCn

PdCn:
        uniquetype = ″if/TR/tr″
        connkey = ″TCPIP″
        connwhere = ″″
#
```

1.7.5 Customized Device Information

If any specific device is configured, that device gets an instance in the customized device object classes in the ODM. Although predefined device object classes provide templates to hold common attributes between each device instance, customized device object classes hold device instance-unique information.

When the configuration manager (cfgmgr) runs during system boot, the cfgmgr reads the config_rules object class and invokes a series of configuration procedures defined in the object class. Based on the config_rules, the cfgmgr detects a device. Then, the the cfgmgr reads predefined device object classes (PdDv, PdAt and PdCn) and creates customized instances in the customized device object classes (CuDv, CuAt, CuDep, CuDvDr and CuVPD) for detected devices.

The cfgmgr uses the method defined in the descriptor definition in the PdDv object class in order to place the device in the defined state (in order to create information in the customized device object classes). Also, the cfgmgr uses the method defined in the descriptor configured in the PdDv object class in order to place the device in the available state (in order to load the device driver). As in the following example, the token-ring interface device has the following methods:

```
# odmget -q ″type=′tr′″ PdDv | grep Define
        Define = ″/usr/lib/methods/defif″
# odmget -q ″type=′tr′″ PdDv | grep Configure
        Configure = ″/usr/lib/methods/cfgif″
#
```

1.7.5.1 Customized Device CuDv

Whenever you configure a device, the corresponding object instance is created in the object class CuDv. Remember that the predefined object class PdDv only stores common object descriptors in the same type. Configuring a specific device means building each device instance, such as tr0. Notice that any device that has the status of defined has the instance in this object class. See the following example:

```
# odmget -q ″name=′tr0′″ CuDv

CuDv:
        name = ″tr0″
        status = 1
        chgstatus = 1
        ddins = ″if_tr″
        location = ″″
        parent = ″inet0″
        connwhere = ″″
        PdDvLn = ″if/TR/tr″
#
```

1.7.5.2 Customized Attribute CuAt

As you know, any attribute defined in the object class PdAt can be changed if you set your preferred value (in other words, updated the default value). The customized attribute is created in the object class CuAt. See the following example for the details. We configured the interface tr0 and set the IP address and the subnet mask. As a result, the interface got the up state. Then, three attributes have been stored in the CuAt. For any other attributes, PdAt is referred to for getting the default values:

```
# odmget -q ″name=′tr0′″ CuAt

CuAt:
        name = ″tr0″
        attribute = ″netaddr″
        value = ″9.68.214.82″
        type = ″R″
        generic = ″DU″
        rep = ″s″
        nls_index = 4

CuAt:
        name = ″tr0″
        attribute = ″netmask″
        value = ″255.255.255.128″
        type = ″R″
        generic = "DU"
        rep = "s"
        nls_index = 8

CuAt:
        name = "tr0"
        attribute = "state"
        value = "up"
        type = "R"
        generic = "DU"
        rep = "sl"
        nls_index = 5
#
```

1.7.5.3 Customized Dependency CuDep

The customized dependency (CuDep) object class describes device instances that depend on other device instances. Dependency does not imply a physical connection. This object class describes the dependence links between logical devices and physical devices as well as dependence links between logical devices, exclusively. Physical dependencies of one device on another device are recorded in the customized device (CuDv) object class. This example shows that the logical device inet0 has a dependency to another logical device tr0. This dependency is obvious because the TCP/IP network instance inet0 requires the network interface or it cannot be configured, as follows:

```
# odmget -q "name='tr0'" CuDep

CuDep:
        name = "tr0"
        dependency = "inet0"
#
```

1.7.5.4 Customized Vital Product Data CuVPD

The customized vital product data (CuVPD) object class contains the vital product data (VPD) for the customized devices. The following example shows that the device tok0 doesn't have VPD in the ODM:

```
# odmget -q "name='tok0'" CuVPD

CuVPD:
        name = "tok0"
        vpd_type = 0
        vpd = "*NA"
#
```

1.7.5.5 Customized Device Driver CuDvDr

The customized device driver (CuDvDr) object class stores information about critical resources that need concurrency management through the use of the device configuration library routines. These routines are used by the cfgmgr and high-level commands. When the descriptor resource is devno, the descriptors value1 and value2 represent the device major number and minor number, respectively. Concurrency control to a specific device can be made using the following values:

```
# odmget -q ″value3=′tok0′″ CuDvDr

CuDvDr:
        resource = ″inst″
        value1 = ″tokdd″
        value2 = ″0″
        value3 = ″tok0″

CuDvDr:
        resource = ″devno″
        value1 = ″36″
        value2 = ″0″
        value3 = ″tok0″
#
```

This is true if we compare the previous information with the major and minor number:

```
# ls -al /dev/tok0
c---------   1 root     system    36,  0 Aug  1 18:12 /dev/tok0
#
```

1.7.6 Updating the ODM Information

The ODM information is kept in a consistent state if everything is working well. Since you are using only SMIT and high-level commands, logically you should not get any inconsistency in the ODM. However, the real world is filled with exceptions. For example, serious damage can occur in the ODM data, which cannot be fixed by SMIT (high-level commands and configuration manager). However, you would still have two final resorts. They are the odmchange command and the ODM editor.

We recommend that you never use these tools for daily maintenance because they don′t take into consideration the relation or dependency between objects or instances in the object classes. This is why these tools are powerful and can achieve more than SMIT and high-level commands. Careless operation of these tools can easily create major problems.

1.7.6.1 Create Your Working Directory

Because the ODM has very critical data for the system, when you update the ODM by hand, the first step is to get the backup. One convenient alternative is to create a working directory for the ODM using the following steps:

1. Create a working directory for the ODM update, as follows:

   ```
   # mkdir /home/objrepos
   #
   ```

2. Copy the entire ODM to the working directory, as follows:

   ```
   # cp -R /etc/objrepos/* /home/objrepos
   #
   ```

3. Set the environment variable ODMDIR to the working directory. After doing this, the ODM-related commands and tools will refer to the ODM in the working directory. The original version will be intact.

   ```
   # export ODMDIR=/home/objrepos
   #
   ```

4. If you issue the odmget command as follows, it reads the data in the working directory:

```
# odmget -q ″seq=1″ Config_Rules

Config_Rules:
        phase = 1
        seq = 1
        boot_mask = 0
        rule = ″/etc/methods/defsys″
#
```

After you complete the update task, check to see whether there are any problems. Then, copy the modified ODM onto the original version.

1.7.6.2 The odmchange and odmdelete Commands

Typical usage of the odmchange and odmdelete commands is shown with an example. It is strongly recommended that you read the manual or InfoExplorer before you continue. Although we don′t explain it, there is another command, odmadd.

1. The interface device tr0 is customized to have the updated mtu. We used the chdev command:

   ```
   # chdev -l tr0 -a mtu=800
   tr0 changed
   #
   ```

2. Next, retrieve the customized information about the attribute mtu using the odmget command. For those ODM-related commands, the usage of the qualifier is important. Character string constants must be enclosed in *single* quotation marks. In this example, the output is sent to a file for edit:

   ```
   # odmget -q ″name=′tr0′ AND attribute=′mtu′″ CuAt > tr0_CuAt_mtu
   # cat tr0_CuAt_mtu

   CuAt:
           name = ″tr0″
           attribute = ″mtu″
           value = ″800″
           type = ″R″
           generic = ″DU″
           rep = ″nr″
           nls_index = 2
   #
   ```

3. Use your preferred editor and edit the file. To disable the trailer protocol, you should change the file, as follows:

   ```
   # cat tr0_CuAt_mtu

   CuAt:
           name = ″tr0″
           attribute = ″mtu″
           value = ″1200″
           type = ″R″
           generic = ″DU″
           rep = ″nr″
           nls_index = 2
   #
   ```

4. In this step, the contents of the updated file are introduced into the ODM. Again, pay attention to how the qualifier is written. If you are using the wrong qualifier, you will destroy the data in the ODM.

```
# odmchange -o CuAt -q ″name=′tr0′ AND attribute=′mtu′″ \
> tr0_CuAt_mtu
#
```

5. Check the ODM with the odmget command to see whether the update was made successfully.

```
# odmget -q ″name=′tr0′ AND attribute=′mtu′″ CuAt

CuAt:
        name = ″tr0″
        attribute = ″mtu″
        value = ″1200″
        type = ″R″
        generic = ″DU″
        rep = ″nr″
        nls_index = 2
#
```

Note: These ODM commands update the ODM and do not touch the corresponding information in the kernel. Therefore, changes are usually effective after the next reboot. Notice that if you made a mistake during the previous procedure, it may not be found until you reboot the system.

If you need to delete the attribute (to make it as if it had not been customized), use the odmdelete command.

1. Again, care must be taken to specify the qualifier. The wrong qualifier may destroy the object instance completely. However, it can still be recovered if the destroyed data belonged to the customized object class.

```
# odmdelete -o CuAt -q ″name=′tr0′ AND attribute=′mtu′″
1 objects deleted
#
```

2. Now, you can no longer see the attribute in the CuAt. It was successfully deleted.

```
# odmget -q ″name=′tr0′ AND attribute=′′mtu′″ CuAt
odmget: Could not retrieve object for CuAt, odm errno 5904
#
```

3. Although the customized attribute was deleted, it doesn′t mean that the attribute mtu of tr0 no longer exists. Remember that this attribute is also in the PdAt object class. If it is not in the CuAt, the PdAt is referred to instead. Then, you can see the current value (default), as follows:

```
# lsattr -E -l tr0 -a mtu
mtu 1492 Maximum IP Packet Size for This Device True
#
```

Difference between V4.1 and V3.2

With V3.2 you have the ODM editor tool. The ODM editor provides the interactive editing capability of the ODM. This tool provides interactive front-end panels to those ODM commands explained in the previous section. Explaining the details of the ODM editor is out of the scope of this book. You can refer to the manual and InfoExplorer for details. You can invoke the ODM editor with the odme command, specifying the object class being edited, as follows:

```
# odme CuAt
```

The odme command or the ODM editor is removed for V4.1. Only the odmdelete, odmchange and odmadd commands are available to update the ODM contents directly.

Chapter 2. Debugging TCP/IP Troubles

In this chapter we explain troubleshooting procedures for TCP/IP problems. Generally, there is no established, well-organized debugging method. Thus, this book is an experiment. We believe that detailed knowledge and understanding has greatly helped us in debugging problems. Therefore, we don′t simply provide procedures. To facilitate this understanding, the backgrounds and mechanisms are described as well.

2.1 When Something Is Wrong with the Network

If you have trouble connecting your system to a network, many steps may be required to fix this. In this chapter, we discuss cases where we have a connectivity problem and not a performance problem. In such cases, you may have a lot of options to change. We show the approaches that worked well for us.

2.1.1 Keep In Mind

Here are some precepts we utilized to make debugging easier. You may want to build your own list of precepts.

2.1.1.1 Knowledge and Understanding of TCP/IP

There are a lot of excellent books on TCP/IP and they are available at bookstores. Magazine articles also help. TCP/IP is a protocol for an open, multi-vendor environment. Thus, we can learn a lot from public sources.

Of course, manuals or InfoExplorer is also a crucial source for mandatory information. Regardless of whether your concern is an AIX unique function or not, you should refer to InfoExplorer first because AIX may have a different approach or implementation from another UNIX system.

You can find a few explanations in the InfoExplorer about TCP/IP or Ethernet, or you can buy commercial books at bookstores.

Our Experience

If you are running or planning to run DNS or sendmail, do not hesitate to buy books from O′Reilly, *DNS and BIND in a Nutshell* or *sendmail*. If you are planning to change some kernel configuration parameters, you should carefully read the articles related to the no command in the InfoExplorer.

2.1.1.2 You Have to Know Your Environment

Many customers do not understand their systems or networks in detail. A few sites are managed and administrated very well, and a lot of remaining sites run with very little organized administrative effort. Thus, you have to know your network environment and system environment very well when you debug them. This is easy to say but difficult to do.

Our Experience

We know a person who happened to run secure TCP/IP for a test and left the system without reconfiguring it. Several months later this person could not configure the .rhosts authentication. It was a long time before it was remembered what had been done on the system.

2.1.1.3 Any Information Would Be a Clue

Any symptom can be a clue. Error messages and the system′s behavior are important information necessary to find the root of the problem. Of course, the event or operation history is crucial information, especially when a working system is not working well. We do not overlook anything unusual. (We know it is difficult to figure out what is *unusual*.) The problem is that it is almost impossible to know exactly who did what on the system before the problem arose.

Our Experience

On a sophisticated, distributed software product, problems arose in AIX/DCE, which the security server could not start. We were lucky enough because someone happened to type the date command and noticed that the clock had been reset to 1970. We adjusted the system clock and the DCE began running. Later, we were informed that another person had reset the NVRAM by removing the battery to resolve another problem.

2.1.1.4 Do Not Believe What People Say

We are not saying that you should not believe people. We are saying that you should not believe what people say without verifying what they say.

Some people say that they didn′t do anything but the system went bad, or that they did what the manual said, but the system still went bad. But consider whether or not they are correct and if they did and what they said. No matter what people say they have done, it is not the same as what they have really done. This philosophy helps you to determine problems. You may have to start from a very basic point in the system to solve the problem.

Our Experience

A customer asked our help via telephone. His problem was that he could not unmount his NFS-mounted file system. We asked him if somebody was working in that directory or if some program was running in that directory. We asked this question a hundred times and his answer was always the same: “No, I′m quite sure nobody or nothing is there!” Finally we said to him, “Would you please issue the command, fuser -u on that directory?” Then his answer was, “Someone is there!”

2.1.1.5 Your Skill Is Limited by the Level of Who Is Now On-Site

This is absolutely true, especially when you are providing problem determination support via telephone. Even if you are a top-class technical specialist, whenever you support people through a telephone line, it is difficult to use your skills and abilities due to the other person′s limited skills and abilities.

If possible, you should visit the site where the problem is and check on the problem yourself. Of course, this may not be easy.

Our Experience

A customer asked us to resolve his problem through a phone call. He claimed that ping worked successfully but he could not use Telnet or any other network applications while using ping. We asked him a lot of questions to isolate the problem and noticed something was very strange. Finally, we found the problem was as he explained to us, which was that the ping worked successfully and that Telnet and other applications could not be run with ping. See the following example:

```
# ping gozira
PING 9.170.4.8: (9.170.4.8): 56 data bytes
0821-069 ping: sendto: A route to the remote host is not available.
ping: wrote 9.170.4.8 64 chars, ret=-1
0821-069 ping: sendto: A route to the remote host is not available.
ping: wrote 9.170.4.8 64 chars, ret=-1
0821-069 ping: sendto: A route to the remote host is not available.
ping: wrote 9.170.4.8 64 chars, ret=-1
^C
----9.170.4.8 PING Statistics----
3 packets transmitted, 0 packets received, 100% packet loss
#
```

2.1.2 Bottom-Up Approach

In our experience, the bottom-up approach seems to work best. This approach checks and confirms the connectivity of each protocol layer from the bottom (layer 1, physical layer) to the top (layer 7, application layer). Since any layer′s connectivity is dependent upon the layer just beneath it, it′s quite reasonable to check from the bottom to the top.

Although TCP/IP doesn′t conform to the OSI seven-layered network model completely, we use this model for our convenience. In this book, we define and use the following definitions. Each layer is explained from the bottom to the top.

Physical Layer (Layer 1)

This layer represents physical media such as coax cable and fiber optic cable. This layer is only responsible for the transmission of a bit stream across a physical circuit. WANs, such as the public telephone network, provide this layer.

At first, it seems that this layer is simple and straightforward, but if you look at the FDDI physical layer, you will change your mind. It is full of strange words such as 4B/5B Code, Symbol Encoding, and PMD Sublayer.

Sometimes, a device called a repeater is used to expand a cable length.

Data Link Layer (Layer 2)

This layer represents a communication link between two nodes (systems). The unit of data processed in this layer is often called the frame. LAN (Ethernet, token-ring, FDDI, etc.) is the technology that covers up until this layer. If you plan to use WAN other than as a packet-switched network, you have to prepare this layer by yourself.

Note: For the TCP/IP protocol stack, PPP is getting popular for WAN, but our RS/6000 doesn′t provide this protocol. You can use SLIP, although it only runs on the async port.

In LAN, the 48-bit MAC address is the key to identify a node (or adapter). In the TCP/IP protocol suite, ARP and RARP are data link layer protocols.

Sometimes, a device called a bridge is used to expand a communication link. A lot of bridges nowadays have learning and filtering functions and most can filter packets except for broadcast packets. Notice that this filtering is based on the MAC address. A portion of a network separated by a bridge is called a segment. In other words, a bridge connects two segments, and a network can be built with more than one segment. Even when more than one bridge is involved on the route to the destination node, it is still one communication link.

Network Layer (Layer 3)

This network layer consists of multiple nodes. The unit of data processed in this layer is often called a packet.

Note: The IP packet is also often called an IP datagram.

The IP is the protocol used to provide this layer. In IP, a 32-bit IP address is the key in identifying a node and in routing packets to a destination. In the TCP/IP protocol suite, IP and ICMP are network layer protocols. IP is the *only* protocol that can carry user data.

This layer allows a connection between any two nodes, and the connection can pass through other nodes. This function is called routing. Complex path controls, which involve more than one communication link (or data link layer), can be made in this layer. Any host system or delegated device that can route an IP datagram is called a router. In the TCP/IP tradition, the router is also called a gateway, although in the OSI world, the term gateway has a different definition.

Note: In this book we use the terms router and gateway interchangeably.

Transport Layer (Layer 4)

This layer represents a connection between two processes. Each node (system) can have multiple processes on it. The TCP and UDP are the protocols used to achieve this layer. The unit of data processed by TCP is often called a segment (and is often called a datagram by UDP). In TCP and UDP, a 16-bit port number is the key to identifying a process. This is because any process that uses TCP or UDP has the delegated port.

This layer also provides miscellaneous functions such as flow control and reliable data communications including retransmission. TCP provides this functionality, but UDP doesn′t.

Session Layer (Layer 5)

This layer provides dialog sessions, such as half-duplex and full-duplex and also synchronization points in the dialog. In the TCP/IP protocol stack, there are no clear definitions for the session layer. Some functionality of this layer is included in TCP.

Presentation (Layer 6)

This layer provides common data presentation between applications. In the TCP/IP protocol stack, there are no clear definitions for the presentation layer, and some applications include this function by itself. For example, Sun Microsystem′s ONC/RPC provides External Data Representation (XDR).

Application (Layer 7)

This layer represents the application entity. Usually, an application program or user is an application entity.

Of course, you don′t need to check through all the layers when something is wrong. You can start with the layer where you are sure about connectivity.

2.2 Debugging the Physical Layer

This section may be the most difficult section to explain since we have few relative topics to discuss.

2.2.1 How to Recognize a Physical Layer Problem

It would be almost impossible to define the symptoms or conditions that indicate hardware trouble. The following list may help you if you are concerned with your hardware′s performance, but do not consider it to be *complete*:

- See whether the problem is intermittent and whether there seems to be no clear indication of what triggers the problem. Of course, some subtle problems could exist (for example, a traffic pattern or load).
- See whether the problem seems to depend on a geographical condition (for example, you may see that the problem is related to a specific workstation, although all conditions are the same with other workstations).
- See whether the problem is caused by a condition that is not directly related to the problem. For example, if someone is using workstation A and workstation B experiences problems, then there is no relation between the two workstations. In other words, there are no TCP/IP communications between A and B.

Unfortunately, there are no clear rules of thumb. If you cannot explain your problem from a software or configuration point of view, the problem may be your hardware, although this is not easy determine.

Our Experience

We had a problem when we loaded Ethernet and all communications (NFS, Telnet etc.) of the destination system were hung. All symptoms implied that there was some faulty hardware (a terminator). We checked all the coax cables and found a terrible misconfiguration where someone had inserted a small cable between the Ethernet adapter port and the T-connector. We saw that there was very little room around an Ethernet adapter card to attach a T-connector directly.

We found that the cable misconfiguration was not the cause. The real cause was the low EC level of the Ethernet adapter. This time the misconfigured hardware was not the problem, but it might have become a problem at any moment.

You may be able to find something in the error log. If your system has some hardware trouble and you are lucky enough, you may be able to see error messages by issuing the errpt command.

2.2.2 How to Identify a Failed Unit

If you want to be certain of the functionality of a physical layer component, you may need an appropriate device. For example, you may need a tester to check to see whether a cable has not disconnected internally. In this section, we explain some inexpensive alternatives. These alternatives are for small environments. If you have more than 100 workstations and PCs connected together, these procedures may not be directly applicable.

2.2.2.1 Replacing the Failed (or Problem) Unit

One inexpensive alternative to using sophisticated devices is the replacement of parts. If you have spare parts stocked, you can just replace the problem cable or terminator with the stocked one and see whether this fixes the problem. Even when you don′t have any stocked parts, you can borrow a well-proven part from somewhere else in your network and test it.

2.2.2.2 Splitting and Building the Network

If you have failed parts, but still are not sure of the location of the failed part, you can split the network into two smaller networks. You may need additional hardware, such as terminators, to split your network. Check to see what network still has the problem. Repeat this procedure until you end up with only one machine. Eventually this method leads you to the failed unit. Some people call this procedure a *binary search*.

Another approach is to build your network from the beginning. Start from a system that you are absolutely certain is working. Then, connect cables and other systems one by one, and see when the problem arises.

You may not be able to perform these procedures in a production environment. Also, this requires a lot of physical work.

2.2.3 Check the Error Log with errpt Command

If you are lucky enough, you can find something in the system error log. It′s worth checking. Be aware that not all hardware problems are recorded in the error log. The following is an example of a connectivity problem. The symptom was clearly that the ARP had failed. Many adapter (tok0) errors appeared on the panel.

```
# errpt
IDENTIFIER TIMESTAMP  T C RESOURCE_NAME  DESCRIPTION
C14C511C   0816110095 T H scsi0          ADAPTER ERROR
2BFA76F6   0815221095 T S SYSPROC        SYSTEM SHUTDOWN BY USER
9DBCFDEE   0816103695 T O errdemon       ERROR LOGGING TURNED ON
...
C14C511C   0712142995 T H scsi0          ADAPTER ERROR
018DF788   0710155395 N U tok0
0734DA1D   0706163595 P H fd0            DISKETTE MEDIA ERROR
0734DA1D   0706163595 P H fd0            DISKETTE MEDIA ERROR
00000020   0508104195 N U tok0
00000020   0508104195 N U tok0
00000020   0508104195 N U tok0
00000020   0508104195 N U tok0
00000020   0508104195 N U tok0
00000020   0508104095 N U tok0
00000020   0508104095 N U tok0
00000020   0508104095 N U tok0
00000020   0508103995 N U tok0
```

```
00000020   0508103995 N U tok0
00000020   0508103995 N U tok0
00000020   0508103895 N U tok0
#
```

You can refer to the details of each error with an -a flag, as shown below:

```
# errpr -a
...
---------------------------------------------------------------------------
LABEL:          NONE
IDENTIFIER:     00000020

Date/Time:       Fri Mar 17 12:24:59
Sequence Number: 4
Machine Id:      00FFFFFF4D00
Node Id:         zero
Class:           U
Type:            NONE
Resource Name:   tok0
Resource Class:  adapter
Resource Type:   tokenring
Location:        00-1X
VPD:
        Displayable Message.........ISA Token Ring

Detail Data

0000 AAAA 0000 0000 0000 0000 0000 0A20 0800 5AAB 2319 0800 5AAB 2319 0000 0000

0000 0000 0000 0000 0000 0000
```

The solution was fairly easy. We pulled and pushed the token-ring data connector and the problem was fixed.

Our Experience

For hands-on troubleshooting, it′s quite useful to have the following two items:

VICTORINOX Swiss Army knife
MINI-MAGLITE flashlight

We had lots of opportunities to crawl under desks and pry panels open. The above items are available at fairly reasonable prices if you compare their price to that of a protocol analyzer.

2.2.4 Ethernet Adapter Problem (EC Level)

There is a problem with the Ethernet High-Performance LAN Adapter, F/C 2980. This problem is clearly documented in the AIX release note which is shipped with AIX, so this should not be a problem.

When you upgrade from an existing AIX 3.1 system with a separate Ethernet adapter, you may need to apply a hardware fix to upgrade your Ethernet adapter. Check your Ethernet adapter card EC Level with the lscfg -v -l command as follows:

```
# lscfg -v -l ent1
  DEVICE            LOCATION          DESCRIPTION

  ent1              00-02             Ethernet High-Performance LAN
                                      Adapter (8ef5)

        Network Address.............02608C2ED435
        ROS Level and ID............0009
        Displayable Message.........802.3/ETHERNET
        Part Number.................071F1182
        EC Level....................C26428
        Device Driver Level.........00
        Diagnostic Level............00
        FRU Number..................081F7913
        Serial Number...............00026236
        Manufacturer................204491

#
```

Note: If you are using an integrated Ethernet adapter, you have to use sio0 instead of ent0. If you don′t, you cannot get the ROS Level and ID information.

If the level (ROS Level and ID) is 008, 009 or 0010, then your Ethernet adapter needs to be upgraded. Call your hardware service representative and request the ECA 015 fix. Do not use your system on the network until the ECA 015 fix has been applied. If you run on the network without the ECA 015 fix, you may experience Ethernet network problems, or the adapter may hang in your system.

2.2.4.1 The Symptom

The symptom arising from using a low-EC level Ethernet adapter is quite difficult to isolate. The following are the typical symptoms:

- When you put a heavy load on the adapter, the adapter hangs.
- When the adapter hangs, ARP fails.

The following errors are logged in the system error log:

ENT_ERR2 COMMUNICATION PROTOCOL ERROR
ENT_ERR6 CSMA/CD LAN COMMUNICATION LOST

Rebooting the system fixes this problem.

Pay attention to these symptoms. The important thing is that these symptoms are almost the same as hardware troubles (such as a faulty cable). The only difference is that the problem can be fixed by rebooting the system. Thus the symptoms can be quite confusing. The failed ARP and the error log suggest that the problem is hardware-related. But, the rebooting fix suggests that the problem is software-related. If you haven′t experienced this problem, which is that a low-EC level adapter hangs the system, you cannot make the problem determination. If someone complains about Ethernet connectivity problems fixed by reboot, consider the possibility of a low-EC level.

2.2.4.2 Why Is this a Problem

A low-EC level Ethernet adapter works fine if the AIX is V3.1. When the AIX V3.2 was introduced, the TCP/IP networking module was updated and the performance was substantially improved. This is why the low-EC level adapter hangs the system.

Even now, we sometimes find that AIX V3.2/V4.1 is running with the low-EC level Ethernet adapter during troubleshooting. Even with AIX V3.2/V4.1, the low-EC level Ethernet adapter can work fine until a user puts a heavy load on the adapter. When they upgraded from AIX V3.1 to V3.2/V4.1, they introduced a potential problem. Later, when the traffic pattern or network usage is changed, the problem arises and the card hangs.

When the upgrade is made, people can read the release notes and find the description of this problem. Eventually they notice that their system still works without the EC upgrade. Then, it becomes easy to forget this potential problem.

2.3 Debugging Data Link Layer with ARP

In this section, we discuss Address Resolution Protocol (ARP) and related topics. ARP provides a dynamic address translation function from the IP address to the MAC address. Now, almost all workstations and other systems that run TCP/IP depend upon this protocol. Although an end user need not to be conscious about the ARP, it is useful and even crucial in debugging a network problem.

Why ARP is Important

We do emphasize the importance of ARP in the following ways:

- When the ARP fails, you can not communicate by any means.
- When the ARP succeeds, you don′t have any hardware problems.
- When the ARP succeeds and still you fail to communicate, there is a software problem, including misconfiguration.

The ARP is a data link layer protocol, and this cannot be used to check higher layers. You cannot check IP routing functions. You can check to see whether or not your system can send and receive a frame to and from the system in the same IP network.

2.3.1 ARP Mechanism Basics

In this section, we explain the ARP mechanism and operation briefly. This will give you the knowledge needed to use ARP as a debugging tool.

2.3.1.1 Why ARP Is Necessary

As you know, the data link layer uses a 48-bit MAC address, and the network layer (IP layer) uses a 32-bit IP address. In the TCP/IP world, any network interface must have an IP address. Also, if LAN is used for the data link layer, the adapter card must have a MAC address.

Whenever you send a packet, you must know the destination system′s IP address and MAC address before you send the packet. You can get the IP address by looking at /etc/hosts or from a name service database such as DNS or NIS. Usually UNIX doesn′t have such a table or database for MAC addresses.

The major reason for this is to reduce administrative nuisance. The IP address is assigned by the network administrator of each site. By contrast, the MAC address is assigned by the adapter card manufacturer and the address is usually burned into the chip so that the user cannot change it.

Note: Our RS/6000 can have an alternate, user-defined address. For SNA, a user-defined address is often assigned to an adapter.

Building a translation table between an IP and a MAC address is tough work. This is because when a system is connected to or disconnected from your network, you should know the MAC address and should update the translation table. Even when an adapter card is exchanged, you need to update the table. Remember that each system has the translation table, and if you have 100 workstations, you need to update 100 tables. If your network is very large, it is almost impossible.

The ARP provides dynamic translation, so you need not gather or control MAC addresses. The destination MAC address is received just before you send a packet. This mechanism allows you to exchange an adapter at any moment without any administrative tasks.

2.3.1.2 ARP Mechanism Brief Review

One solution is to ask the destination machine directly about its MAC address. This is the basic concept of ARP. Of course, the destination MAC address is unknown, and ARP has to use a broadcast. The overview of the ARP procedure is as follows:

1. Have your system search the ARP cache for the destination MAC address. The search key gives you the IP address. If the MAC address is found, use it. If the destination IP address is not in the same IP network, your system searches the routing table and finds the appropriate gateway. Then, search the ARP cache for the gateway MAC address since the packet should be sent to the gateway.

   Note: Many implementations actually use ARP *back pocket* (for example, a cache of ARP cache is used for performance improvement). This will be explained later.

2. If the search fails, the system sends an ARP request message. This is called a broadcast packet.

   The following is a sample ARP request packet on token-ring by iptrace. In this example, the system mat is asking what the MAC address is of a system kashima.

```
Packet Number 42
TOK: ====( 52 bytes transmitted on interface tr0 )==== 13:51:27.606365696
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 90:00:5a:a8:b5:c1, dst = ff:ff:ff:ff:ff:ff]
TOK: routing control field = 8240,  0 routing segments
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 806 (ARP)
ARP: hardware address format = 6 (802.3, 802.5)
ARP: protocol address format = 800 (IP)
ARP: address lengths ; hardware = 6, protocol = 4
ARP: arp operation = 1 (request)
ARP: source addresses: hw [10:00:5a:a8:b5:c1]
ARP:            protocol [9.68.214.82] (mat.hakozaki.ibm.com)
ARP: target addresses: hw [00:00:00:00:00:00]
ARP:            protocol [9.68.214.76] (kashima.hakozaki.ibm.com)
```

Note: Notice that the source MAC address in the token-ring header is 90:00:5a:a8:b5:c1 and that it doesn′t match the source MAC address in the ARP information field. Due to the token-ring architecture, the first bit of the source address has to be used as a routing information indicator. This rule changed the first byte from 10: to 90:.

The Content of the ARP Request Packet

This packet has the target IP/MAC address fields, and the MAC field is filled with all 0s. The source IP/MAC address fields, are filled with the target field′s addresses. Note that these fields are in the information field and *not* in the packet header.

3. All systems in the same network (LAN) receive the ARP request packet.

ARP Cache Update

Even if the system is not the destination and the system has the entry of the requesting system in its ARP cache, the system updates the entry with the packet′s source IP/MAC address fields. But, if the system doesn′t have the entry, nothing happens.

4. Only the destination system responds to an ARP request with an ARP reply. The destination system swaps the target and the source IP/MAC address fields. Then, it fills the source IP/MAC address fields with its addresses. It was originally filled with all 0s.

ARP Cache Update or Creation

If the destination system doesn′t have the entry for the requesting system in the ARP cache, then it creates this entry. If it already has the entry, then it updates it. For this creation/update, the source IP/MAC address fields of the ARP request packet are used.

5. The requesting system receives the ARP reply and gets the destination′s MAC address. Then, it stores the IP/MAC address in the ARP cache.

   The following is a sample ARP response packet on token-ring by iptrace. In this example a system kashima is answering by sending its MAC address to the system mat.

```
Packet Number 44
TOK: ====( 52 bytes received on interface tr0 )==== 13:51:27.608872704
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = c0:00:7e:06:44:40, dst = 10:00:5a:a8:b5:c1]
TOK: routing control field = 02c0,  0 routing segments
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 806 (ARP)
ARP: hardware address format = 6 (802.3, 802.5)
ARP: protocol address format = 800 (IP)
ARP: address lengths ; hardware = 6, protocol = 4
ARP: arp operation = 2 (reply)
ARP: source addresses: hw [40:00:7e:06:44:40]
ARP:             protocol [9.68.214.76] (kashima.hakozaki.ibm.com)
ARP: target addresses: hw [10:00:5a:a8:b5:c1]
ARP:             protocol [9.68.214.82] (mat.hakozaki.ibm.com)
```

The destination system′s MAC address is kept in the ARP cache (memory). This reduces the future ARP traffic. An ARP cache entry expires after 20 minutes. Thus, even for the same destination machine, the ARP broadcast may be needed once every 20 minutes. If you reboot the system, all entries in the cache will be lost.

Note: Twenty minutes is the default value and you can change it with the no command option arpt_killc. We don′t see any reason to change it. To review the current value, issue the no command as follows:

```
# no -o arpt_killc
arpt_killc = 20
#
```

2.3.2 ARP Cache Operation

Now we will explain some concrete procedures on how to use ARP for TCP/IP troubleshooting. We can use the arp command to work with the ARP cache. The arp command has the following options:

```
# arp -h
Usage: arp hostname
        arp -a[n] [/dev/kmem]
        arp -d hostname
        arp -s ether hostname ether_addr [temp] [pub]
        arp -s 802.3 hostname ether_addr [temp] [pub]
        arp -s fddi hostname fddi_addr [fddi_route] [temp] [pub]
        arp -s 802.5 hostname token_addr [token_route] [temp] [pub]
        arp -f filename [type]
#
```

Difference between V4.1 and V3.2

With AIX V4.1, the -n flag is newly added. You can see arp table with suppressing IP address and host name resolution. This flag is valuable for troubleshooting.

2.3.2.1 Looking at ARP Cache

If you want to review the current ARP cache content, issue the `arp -a option`. This is the translation table to convert an IP address to the MAC address.

```
# arp -a
  ? (9.170.3.71) at 40:0:70:9b:1d:c0 [token ring]
  ? (9.170.3.75) at 40:0:70:9b:1d:bb [token ring]
  os2sat.fscjapan.ibm.com (9.170.3.180) at 40:0:70:9b:1f:55 [token ring] rt=ca0:a961:1a01:1001:1b01:b960
  ? (9.170.3.93) at 10:0:5a:a9:26:9d [token ring]
  ? (9.170.3.94) at 10:0:5a:25:6b:a4 [token ring]
  cedar.fscjapan.ibm.com (9.170.1.11) at 10:0:5a:4f:18:7 [token ring]
  osi6000.fscjapan.ibm.com (9.170.3.100) at 40:0:12:17:61:55 [token ring] rt=cc0:a961:1a01:1001:1b01:b960
  ishii.fscjapan.ibm.com (9.170.1.13) at 10:0:5a:4f:50:2b [token ring]
  noname02.fscjapan.ibm.com (9.170.3.102) at 10:0:5a:a8:1f:f6 [token ring]
  sugahara.fscjapan.ibm.com (9.170.3.3) at 10:0:5a:c9:17:fa [token ring]
  coral.fscjapan.ibm.com (9.170.3.5) at 10:0:5a:c9:0:64 [token ring]
  baba.fscjapan.ibm.com (9.170.3.107) at 10:0:5a:a8:8f:76 [token ring]
  aries.fscjapan.ibm.com (9.170.3.9) at 10:0:5a:a8:4e:75 [token ring]
  noname04.fscjapan.ibm.com (9.170.3.109) at 40:0:70:9b:1c:dd [token ring] rt=c40:b961:1b01:1001:1a01:a960
#
```

You may see host names that you have never tried to access. When a system accesses your system, even by sending a ping packet, that system appears in the ARP cache automatically. This mechanism has the effect of reducing future ARP traffic. The ARP implies that someone is trying to send something to your system and that there must be a very high possibility that your system will receive the response. Therefore, it′s quite reasonable to create the entry of the accessing machine by just looking at the ARP request packet information field.

Note: If the access is broadcast, the entry is not created in the ARP cache. The broadcast includes both the MAC layer broadcast and the IP broadcast.

Some products, such as NetView, keep sending a ping packet periodically for monitoring purposes. You may see the never disappearing entries in your ARP cache.

In LAN, every system can hear all the ARP request packets. This is because ARP request packets are broadcasted. It′s possible to collect all the IP and MAC address pairs from those ARP request packets and to build the ARP cache, but this is not good approach. Whenever a system sends a packet, the ARP cache should be looked up. A bigger ARP cache reduces system performance, and the above approach may build an unnecessarily big ARP cache. For this reason, consider why the cache entry has an expiration mechanism.

AIX V4.1 provides options to configure the ARP cache (table) size with the no command as follows:

```
# no -a | grep arptab
              arptab_bsiz = 7
                arptab_nb = 25
#
```

For the V4.1, the default cache size for a router configured system is 25 and for a non-router configured system it is 20.

Difference between V4.1 and V3.2

V3.2 can have up to 20 entries in the ARP cache. If the RS/6000 is a router, it can have up to 100 entries in the ARP cache. These maximums should be enough. You can refer to the header file /usr/include/net/if_arp.h for details.

2.3.2.2 ARP Connectivity Test Procedure

In order to make matters clear, you can do the following ARP operation. This procedure checks the connectivity.

1. First, review the ARP cache as follows:

   ```
   # arp -a
     ? (9.170.3.107) at 10:0:5a:a8:8f:76 [token ring]
     inoki (9.170.3.240) at 40:0:70:9b:1d:e4 [token ring]
     newton (9.170.3.45) at 10:0:5a:a8:46:2d [token ring]
   #
   ```

2. Then you can delete any entry in the cache as follows:

   ```
   # arp -d newton
   Entry newton (9.170.3.45) was deleted from local arp table.
   #
   ```

3. Confirm the entry has been removed as follows:

   ```
   # arp -a
     ? (9.170.3.107) at 10:0:5a:a8:8f:76 [token ring]
     inoki (9.170.3.240) at 40:0:70:9b:1d:e4 [token ring]
   #
   ```

4. Then you can try the connectivity test. Try to access the destination system in your favorite way. Ping may be the appropriate command. Of course, you can use Telnet instead. In the following example, we sent only one ping packet:

   ```
   # ping newton -c 1
   PING newton: (9.170.3.45): 0 data bytes
   8 bytes from 9.170.3.45: icmp_seq=0 ttl=255

   ----newton PING Statistics----
   1 packets transmitted, 1 packets received, 0% packet loss
   #
   ```

5. Then, check the ARP cache again, as follows, and you can find the appropriate entry if the command succeeded:

   ```
   # arp -a
     ? (9.170.3.107) at 10:0:5a:a8:8f:76 [token ring]
     inoki (9.170.3.240) at 40:0:70:9b:1d:e4 [token ring]
     newton (9.170.3.45) at 10:0:5a:a8:46:2d [token ring]
   #
   ```

2.3.2.3 How to Enable/Disable the ARP Function

You can disable the ARP functionality and see what happens. If you disable ARP, you have to load the IP and MAC addresses to your ARP cache manually or you will not be able to communicate after the current ARP cache entries are expired. You can load an entry with the arp -s command. The ARP disable procedure is as follows:

```
# chdev -l tr0 -a arp=off
tr0 changed
#
```

You can enable ARP as follows:

```
# chdev -l tr0 -a arp=on
tr0 changed
#
```

2.3.3 ARP Successful Examples

After the connectivity test, you will see the following entries in your ARP cache:

2.3.3.1 Ethernet

This is a self-explanatory process.

```
miami (9.116.158.5) at 2:60:8c:2f:6:d6 [ethernet]
```

2.3.3.2 Token-Ring

The following is the case when the destination system is in the same physical ring:

```
admi (9.116.128.38) at 10:0:5a:4f:35:b8 [token ring]
```

2.3.3.3 Token-Ring with Bridge

We can connect more than one ring with bridges and in that case, the result will be as follows:

```
paris (9.116.158.249) at 10:0:5a:a8:b5:c1 [token ring] rt=8a8:241:ffe1:90
```

If the destination system is located on a different physical ring, the ARP request/response packet has to go through a bridge or bridges. In token-ring, the source routing is usually used, and the routing information is set in the packet header (MAC header). In the previous example, the routing information is also displayed.

Note: Be aware that this is not IP routing. This is only for data link-level routing, and from the IP layer′s point of view, there is still only one IP network. Bridges can pass broadcast packets which is important. This is a prerequisite for the ARP function to work.

The following lists each field of the routing information. Unfortunately, each field doesn′t fit into the 8-bit boundary, and it makes the reading and understanding of these fields difficult.

```
rt=8a8:241:ffe1:90
```

- The first two bytes, 8a8, are for the routing control field. This field includes the following information:

  Broadcast indicator (3 bits)
  Length bit (5 bits)
  Direction bit (1 bit)
  Largest frame bits (3 bits)
  Reserved bit (4 bits)

- The second and the following two byte fields are separated by a colon. 241, ffe1, and 90 all make up a route designator field. This field consists of a 12-bit ring number and a 4-bit bridge number.

You can refer to *Token-Ring Network Architecture Reference*, SC30-3374 for a more complete explanation.

2.3.3.4 X-Station

If your system is serving an X-Station, in other words, if your system is running an X-Station Manager program, you may find an entry with the term permanent as follows:

```
  hal (9.116.158.5) at 8:0:5a:3a:1:4c [ethernet] permanent
```

This entry is not the subject to be expired or deleted. It remains until the system reboot or until you can remove it with the explicit command arp -d.

During the X-Station boot process, the server system has to send packets to the X-Station, but the X-Station cannot respond to the ARP request. This is why it needs a permanent entry.

2.3.3.5 The Destination Host Is Not in Your /etc/hosts File

If your system cannot map the IP address to the host name, the host name portion is as follows:

```
  ? (9.110.158.212) at 2:60:8c:2f:6:d9 [ethernet]
```

If the destination system is not registered in /etc/hosts or the name service database that you are using (usually DNS or NIS), then your system uses a question mark (?) for the host name. This is never a connectivity problem of the data link layer. Your system will not have a problem with hardware such as the cable, adapter, and terminator.

This may be a problem because some applications need the IP address to perform host name mapping. One example is NFS which will not work if it doesn′t have the host name. We recommend that you check your name service or /etc/hosts file.

2.3.4 ARP Failed Example

If your system failed with the ARP request/response operation, you get the following result:

```
  rio (9.110.181.3) at (incomplete)
```

A failed entry is removed from the ARP cache after three minutes. In this case, you have a connectivity problem at the data link layer or physical layer. In our experience, in almost all cases the causes were faulty cables, terminators or adapter cards.

In this situation, you can not communicate. No communication applications are available.

2.3.5 Proxy ARP and RARP

In this section, we explain two special situations where the ARP mechanism doesn′t work or is not applicable. In this case, Proxy ARP and RARP have been devised.

2.3.5.1 Proxy ARP

The usual ARP procedure stores a set of MAC addresses and an IP address after an ARP broadcast request or response is received. There are some devices that cannot support ARP or respond to the ARP request For example, machines attached through SLIP or PPP lines can not support ARP because these serial lines can not provide broadcast capability. For such a case, terminal servers such as the 7318-s20, provides Proxy ARP). In this case, there must be someone in the same network who can respond to the ARP request as the proxy.

Another situation is when you have to use a device that cannot understand subnet or subnet mask. This situation is quite rare nowadays. A router between subnets responds to the ARP request, which is searching for a MAC address in the other subnet, as the proxy ARP server. In this case, the router informs the ARP of its MAC address instead of informing the destination system.

It′s rare for you to face the previous situations. You have an option to add an entry in the ARP cache manually. For this purpose, use the -s and pub flags as follows:

```
# arp -s ether newton 10:0:5a:a8:46:2d pub
# arp -a
  ? (9.170.3.107) at 10:0:5a:a8:8f:76 [token ring]
  newton (9.170.3.45) at 10:0:5a:a8:46:2d [token ring] permanent published
#
```

A manually added entry is a permanent entry if you don′t specify the temp option. In this example, the system will respond if it gets an ARP request asking for the MAC address of the system newton (9.170.3.45).

2.3.5.2 RARP

The Reverse ARP (RARP) is for the system that knows its MAC address but doesn′t know its IP address. RARP is a protocol that allows a system to get its IP address from the RARP server. A diskless/dataless machine and X-Station are good candidates for the RARP. They don′t know their IP addresses when they boot and RARP may be used. There must be more than one RARP server in the same network if you use RARP.

Our RS/6000 and X-Station don′t use RARP. They use BOOTP instead. BOOTP has the following advantages over RARP:

- RARP as well as ARP cannot pass through a router. BOOTP can pass through a router since BOOTP is built on top of the UDP and IP. It can use the IP routing function to do this.
- If you need RARP, your operating system must support and provide RARP functionality. On the other hand, BOOTP is usually implemented as an application module. You can add the BOOTP capability to your system.
- BOOTP can carry more information than RARP.

An interesting point is that we can make our RS/6000 an RARP server. Although it never uses RARP by itself, your system will respond to the system that sends the RARP broadcast request. Of course, we have to set the IP and MAC address in the ARP cache manually.

Add the entry in the cache with the -s flag as follows:

```
# arp -s ether newton 10:0:5a:a8:46:2d
# arp -a
  ? (9.170.3.107) at 10:0:5a:a8:8f:76 [token ring]
  newton (9.170.3.45) at 10:0:5a:a8:46:2d [token ring] permanent
#
```

Note: You cannot disable the RARP functionality if the entry is in the ARP cache. Other vendor′s implementations, such as Sun Microsystems, need the daemon rarpd running. RS/6000 doesn′t use rarpd.

2.3.6 Duplicated IP Address

One problem that occurs is a duplicated IP address. This is due to a careless IP address assignment or administrative confusion.

Note: We name the system, which has the identical IP address with your intended destination system, a false system or false destination.

2.3.6.1 The Mechanism of Trouble

Consider the situation when there are two systems that have an identical IP address. If they are located in the same IP network, when your system sends an ARP request with that IP address, your system will get two ARP responses including different destinations for the MAC addresses. The ARP is a very simple protocol and doesn′t have any duplication detection or avoidance mechanism. As a result, the latest ARP response overwrites the prior ARP response. Then, the slow-responding system will get its entry put into your system′s ARP cache.

An important point is that it′s impossible to know which system will be the slow-responding and have its entry put into your ARP cache. Once a system wins, its MAC address remains for the next 20 minutes in your ARP cache. Then, this process begins again, which invites a very unstable situation. Sometimes you can communicate successfully and sometimes you cannot. If one system is located beyond a bridge and the other is located in the same LAN segment, usually the farthest one wins. When the nearest one is heavily loaded, the nearest one wins. You can figure out a lot of scenarios, but almost all cases should have intermittent characteristics.

If you check the ARP cache, you find the cause much easier. It′s not a bad idea to list all of your destination′s MAC addresses and compare them with the ARP cache.

Note: Although the MAC addresses are subject to being updated by chance and ARP is designed to compensate this concern, it would still be worth writing down the MAC address list.

For example, when everything is fine, you see the following:

```
# arp -a
  grover.fscjapan.ibm.com (9.170.5.21) at 10:0:5a:b1:5b:23 [token ring]
#
```

When you lose the communication to grover, as follows, you are almost sure to have found the reason:

```
# arp -a
  grover.fscjapan.ibm.com (9.170.5.21) at10:0:5a:b1:6e:fd [token ring]
#
```

One problem is how to find the system that has the MAC address of 10:0:5a:b1:6e:fd. It′s not so easy if you don′t have a complete list of the MAC addresses in your site. If your site uses several vendors, you have to learn the appropriate commands to display the MAC address to all the vendor products.

Our Experience

We had a situation where we could not communicate with a certain system at all. The problem was that we telneted to another system first and then issued telnet again to the destination from that system and it worked. Finally, we found that there was a system that belonged to another department that had the duplicate IP address. The theory is that the slow-responding system wins or that the farthest one wins. This explains the symptoms clearly. When we logged on to the other system, which was located electrically closer to the false system, the correct destination system has its entry put into your ARP cache.

Nowadays, restructuring (not the network but the whole company) is a trend, and this kind of administrative confusion happens easily.

If you have a protocol analyzer, this kind of problem should be discovered immediately. A protocol analyzer triggers the alarm to inform us that there is a duplicated IP address system when it finds a duplicated IP address packet. In our experience, it took only 10 or 20 seconds to reveal the existence of an IP address duplication. This is the value of the protocol analyzer.

2.3.6.2 What Happens on the Destination System

This depends on the method (application) you are using. When your packets are sent to a false destination system, they are received as follows:

1. The data link module checks the destination MAC address in the data link header, and the packets pass the check.
2. The IP module checks the destination IP address in the IP header, and the packets pass the check.
3. The transport module (TCP or UDP) checks the port number in the TCP/UDP header. This is the place where the problem may be detected.
    - If your packets are for the port of the application that is not available or not running on the false system, the transport module cannot deliver your packets. In the case of UDP, the communication ends up with the ICMP error, Destination unreachable - Port unreachable. In the TCP case, the TCP module sends the ACK packet with the RST flag on and forces the TCP session to terminate.
    - If the false system happens to have the destination port in your packet, the transport module delivers your packets to that port. Then, the result is totally application dependent. For example, the TELNET server daemon telnetd accepts your packets without any problems and shows you the login prompt.

Since most workstations have telnetd running on them, as mentioned above, you may be able to log in to the false system, because the telnetd is always listening to the fixed number port 20. If the system has a customized login prompt or a different login screen, you can recognize the problem.

If the false system is a PC or is some system which is not running telnetd, your TELNET session request is terminated and you will see the following message:

```
# tn joker
Trying...
telnet: connect: Connection refused
#
```

This symptom is exactly the same as if the telnetd is not started on the correct destination system. Then you may not notice that you are connecting to the false destination.

2.3.6.3 When Your Communication Suddenly Hangs

Consider the situation that someone in your company happens to have assigned the identical IP address as your system to someone′s PC. When someone, who we call Mr. X, turns his PC on, then that PC responds to your destination system′s ARP request. Based on the theory that the farthest system wins, Mr. X′s PC updates the ARP cache of your destination system. We assume that Mr. X′s desk is far from your office, and that he is working in another division of your company. It′s a reasonable assumption because IP address duplication usually happens with more than two administration entities.

Then, your communication suddenly hangs. This is the moment when Mr.X′s PC updates the ARP cache of your destination system. Now, your destination system no longer recognizes your system because your system′s MAC address was replaced with the MAC address of Mr. X′s PC. Since all responses of the destination system are sent to Mr.X′s PC, there must be many Destination unreachable error counts on the destination system.

When Mr.X turn his PC off, you will gain the access again. From your point of view, the problem is intermittent and can not find a clear condition or criteria.

2.3.7 ARP Pitfalls

In this section we address some ARP-related hints and tips. You should know these in order to be able to complete a debug.

2.3.7.1 When You Have a Router

Be aware that ARP is a data link layer protocol. If your destination system is located in another IP network, The IP routing mechanism is invoked first. This is because all packets going to their destination have to pass through a gateway (also called a router). The IP routing finds the appropriate gateway by searching the routing table. Your system sends an ARP request packet that asks for the MAC address of the gateway system. As a result, you get the entry of the gateway system in your ARP cache. See the following example:

1. Check the ARP cache where you now have only one entry, as follows:

   ```
   # arp -a
     grover.fscjapan.ibm.com (9.170.5.21) at 10:0:5a:b1:5b:23 [token ring]
   #
   ```

2. Issue ping as follows to the system lazy in the IP network 9.170.1:

```
# ping lazy -c 1
PING lazy.fscjapan.ibm.com: (9.170.1.9): 0 data bytes
8 bytes from 9.170.1.9: icmp_seq=0 ttl=254

----lazy.fscjapan.ibm.com PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
#
```

Note: Pay attention to ttl=254. The ping request and reply initially have the time-to-live (TTL) of 255 in the IP header. This ping reply passed through a router, and the TTL in the IP header was decremented by one.

3. Check the ARP cache again and see what happened. The entry of the gateway host inoki5 is as follows:

```
# arp -a
  grover.fscjapan.ibm.com (9.170.5.21) at 10:0:5a:b1:5b:23 [token ring]
  inoki5.fscjapan.ibm.com (9.170.5.240) at 40:0:70:9b:1d:e4 [token ring]
#
```

Notice that the gateway or router system inoki5.fscjapan.ibm.com appeared in the ARP ache. Refer to the following routing table. The destination lazy is beyond inoki5, which is the default gateway. Any packet that has the destination address other than 9.170.5 is sent to inoki5.

```
# netstat -r | grep U
Destination      Gateway            Flags     Refs     Use  Interface
default          inoki5.fscjapan.ib UG           0     988  tr0
9.170.5          newton.fscjapan.ib U            7   45628  tr0
127              loopback           U            1       4  lo0
#
```

2.3.7.2 ARP Cache Back Pocket (V3.2 Only)

Many ARP implementations, including RS/6000, have a mechanism to improve ARP cache look-up performance. In this book we call it back pocket. You will see a mysterious situation that you cannot understand or explain without the knowledge of the back pocket. We explain this with the following example:

1. Check the ARP cache and you will see that nothing is there, as follows:

```
# arp -a
#
```

2. Then ping to a system and see the ARP cache. Of course, the entry is created, and the following is what we expected:

```
# ping jodie -c 1
PING jodie: (9.170.5.44): 0 data bytes
8 bytes from 9.170.5.44: icmp_seq=0 ttl=255

----jodie PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
# arp -a
  jodie (9.170.5.44) at 10:0:5a:b1:6e:ca [token ring]
#
```

3. Delete the entry and confirm that the entry is really removed, as follows:

```
# arp -d jodie
Entry jodie (9.170.5.44) was deleted from local arp table.
# arp -a
#
```

4. Then, ping again to the same system as follows, and see the ARP cache again:

   ```
   # ping jodie -c 1
   PING jodie: (9.170.5.44): 0 data bytes
   8 bytes from 9.170.5.44: icmp_seq=0 ttl=255

   ----jodie PING Statistics----
   1 packets transmitted, 1 packets received, 0% packet loss
   # arp -a
   #
   ```

   Nothing is in the ARP cache, but you can ping to the system. This is the effect of ARP cache back pocket.

This mechanism is a part of the network interface. A network interface holds only one destination MAC and IP address pair. This is for the latest system that you accessed. If the destination IP address is the same as the address in back pocket, the ARP cache is never used. The network interface just refers to the back pocket. This means that if the destination system exchanged its adapter card and the MAC address was updated, it could not be known to your system.

Note: If the destination system sends a packet (for example, ping) to your system, it will update your ARP cache automatically. Even the ARP reply broadcast of the destination system updates your ARP cache. But since the ARP cache is not referred to, there is no way to know about it.

If your system communicates with only one destination system, the ARP cache is never looked up. It is possible to use only the information in the back pocket while the ARP cache was updated one hundred times. The only way to force the system to look up the ARP cache is to access any other system with your favorite methods (ping is enough). This will update the content of back pocket and force the network interface to search for the ARP cache the next time.

Although a lot of vendor implementations have this feature, this is not a part of RFC or any standard. As you have seen previously, this may invite unexpected situations. It is good to know that if you use some software-dependent products on correct ARP behavior, it may be impacted. HACMP/6000 is one candidate.

Difference between V4.1 and V3.2

V4.1 doesn't have this sophisticated functionality.

2.3.8 MAC Address Basics

We briefly explain the MAC address. As we have seen in the ARP explanation, the MAC address has a crucial role in any communication.

As in the convention, we call a 48-bit data link layer address a MAC address. Exactly speaking, the Ethernet Version 2 uses the term hardware address. IEEE 802.3 Ethernet and other IEEE LAN, such as IEEE 802.5 token-ring uses the term MAC address. The hardware address and the MAC address are almost the same but not completely identical by definition.

2.3.8.1 MAC Address Assignment

The MAC address is a 48-bit address for a data link layer. It has some rules for address assignment. As in the convention, we write this address as a sequence of hexadecimal characters separated by a colon.

- The first bit is used as an individual/group identifier.

  If this bit is 1, then this address is considered to be a group address. This means that this frame should be received by a group of stations. If this bit is 0, then this address is considered to be an individual address. This means that this frame should be received by only one station.

  Note: This definition is effective when the address is a destination address.

- The second bit is used as a universally/locally administered identifier.

  If this bit is 1, then this address is considered to be a locally administered address. This means that this address was assigned and controlled by your site′s network administrator. It′s your network administrator′s responsibility to guarantee the uniqueness of the address. If this bit is 0, then this address is considered to be a universally administered address. This means that this address was assigned and controlled by the IEEE and the adapter card manufacturer. The first 24 bits are assigned by the IEEE to each manufacturer. The last 24 bits are assigned by the adapter card manufacturer.

  Note: In the Ethernet Version 2 specification, there is no definition about a universally/locally administered identifier.

Now we examine the following example:

```
10:0:5a:b1:5b:23

   1    0    0    0    5    a    b    1    5    b    2    3
|----+----|----+----|----+----|----+----|----+----|----+----|
 0001 0000 0000 0000 0101 1010 1011 0001 0101 1011 0010 0011
```

The first byte - second bit means that the least significant bit (LSB) and the second least significant bit of the first byte, respectively in this example, the LSB is the farthest right bit of each byte. When a frame is transmitted in the header address fields, each byte must be transmitted in the order of the LSB first. This is called *canonical order*. In the previous example, bits are transmitted in the following order on a cable (from left to right). Notice that the order of the bytes is not changed.

```
10:0:5a:b1:5b:23

   0    8    0    0    5    a    8    d    d    a    c    4
|----+----|----+----|----+----|----+----|----+----|----+----|
 0000 1000 0000 0000 0101 1010 1000 1101 1101 1010 1100 0100
```

Note: Unfortunately, token-ring doesn′t follow the canonical order. When you connect an Ethernet and a token-ring, the bridge must convert the bit ordering in the header address fields of each frame (usually bridges do this).

Since the first bit and the second bit are both 0, this is an individual and universally administered address. In this example, the first 24 bits 10:0:5a were assigned by the IEEE to IBM Corp. When you are looking for a duplicated IP address system and if you know the MAC address of a system from ARP cache, it may help you identify the system. Of course, this could tell you who the adapter card manufacturer is but not who the workstation manufacturer is. If you get the universally administered address of 3Com Corp., it could be anything.

Note: Recently 3Com used up one block of 24-bit addresses, 02:60:8c, and they got another new block, 00:60:8c, from the IEEE.

Our Experience

We heard an interesting story. When 3Com changed their MAC address block, they got a lot of problem reports from their customers. As in the definition, we can assign the MAC address locally. But many network management tools or software assume that the MAC address will never be changed. Also they used the first 24 bits as a fixed manufacturer identifier. In almost all situations, this approach worked. This could happen at any moment.

If you need to know the list of universally administered MAC addresses, refer to the *RFC 1700 ASSIGNED NUMBERS*.

2.3.8.2 How to Know Your MAC Address

For debugging, you may need to know your system′s MAC address. Log on to a system to which you sent a packet and check the system′s ARP cache. This gives you the correct answer, but is not a smart solution. Use the netstat -v command on your system. You can see the line named Hardware Address which is the MAC address as follows:

```
# netstat -v
-------------------------------------------------------------
TOKEN-RING STATISTICS (tok0) :
Device Type: Token-Ring IBM ISA Adapter
Hardware Address: 08:00:5a:ab:23:19
Elapsed Time: 0 days 4 hours 23 minutes 14 seconds
...
```

Another alternative is to use the netstat -i command. Before issuing this command, you have to configure the network interface. You will see the following MAC address:

```
# netstat -i
Name  Mtu   Network     Address           Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                          209     0      209     0     0
lo0   16896 127         localhost           209     0      209     0     0
tr0   1492  <Link>8.0.5a.ab.23.19           129     0       53     0     0
tr0   1492  9.68.214    zero.hakozaki.i     129     0       53     0     0
#
```

A MAC address is burned into a chip on the adapter card and you cannot change it. RS/6000 allows you to assign an alternative MAC address. The netstat -v command always tells you the original burned-in address. On the contrary, the netstat -i command tells you the current effective MAC address. If you configured an alternative one, it is shown as follows:

```
# netstat -v | grep Hardware
Hardware Address: 08:00:5a:ab:23:19
# netstat -i
Name  Mtu   Network     Address           Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                          176     0      176     0     0
lo0   16896 127         localhost           176     0      176     0     0
tr0   1492  <Link>40.0.7e.8.66.70           204     0       58     0     0
tr0   1492  9.68.214    zero.hakozaki.i     204     0       58     0     0
#
```

Another way to get your MAC address is to issue the lscfg -v -l command. This command always shows you the original burned-in address. Issue this command as follows:

```
# lscfg -v -l tok0
  DEVICE            LOCATION          DESCRIPTION

  tok0              00-1X             IBM 16/4 PowerPC Token-Ring Adapter
                                      (isa)

        Network Address.............08005AAB2319
        Displayable Message.........ISA Token Ring

#
```

Difference between V4.1 and V3.2

With V3.2, the following is one tip for using the lscfg command: if you are looking for an integrated Ethernet adapter's MAC address, you cannot refer to it with a usual convention, such as ent0. Nothing is displayed if you search adapters. Instead, you have to look at the standard I/O planner as follows:

```
# lscfg -v -l sio0
  DEVICE            LOCATION          DESCRIPTION

  sio0              00-00             Standard I/O Planar

        Machine Type and Model......70110230
        Part Number.................051G8271
        EC Level....................D18224
        Processor Identification....000281
...
        Displayable Message.........ETHERNET
        Network Address.............08005AC7AF8C
        Device Driver Level.........00
...
```

2.3.9 A MAC Address Pitfall

Although the MAC addresses are now strictly administered (this is especially true for universally administered addresses), there are still some pitfalls. We introduce only one example in this section. Of course, there may be other possibilities, and this section gives you an understanding that anything can happen at any moment.

2.3.9.1 Universally Administered Address Violation

As you already know, any universally administered address must require the second bit to be 0. This is a strict rule, but there are exceptions to every rule. Due to the historical background, there are some MAC address blocks that were assigned before the rule was fixed. In *RFC 1700 ASSIGNED NUMBERS*, there are some remarks on this issue.

RFC 1700 ASSIGNED NUMBERS

...If there is a global algorithm for which addresses are designated to be physical (in a chipset) versus logical (assigned in software), or globally-assigned versus locally-assigned addresses, some of the known addresses do not follow the scheme (for example, AA0003; 02xxxx).

In this RFC, you find that the following 24-bit MAC address blocks are violating the rule. They were formally assigned universal addresses by the IEEE (or formerly by Xerox Corp.), but their second bits are 1.

```
02:07:01   Racal InterLan
02:04:06   BBN   BBN internal usage (not registered)
02:60:86   Satelcom MegaPac (UK)
02:60:8C   3Com  IBM PC; Imagen; Valid; Cisco
02:CF:1F   CMC   Masscomp; Silicon Graphics; Prime EXL
AA:00:00   DEC   obsolete
AA:00:01   DEC   obsolete
AA:00:02   DEC   obsolete
AA:00:03   DEC   Global physical address for some DEC machines
AA:00:04   DEC   Local logical address for system running DECNET
```

Notice that the standards IEEE 802.3 and IEEE 802.5 define what bit is what. But they don′t define what we should do for one or the other. If the second bit is 1, it is a universal address. But it′s not clearly defined how the packets that have this address should be treated. This ambiguous situation can allow vendor-dependent hardware and software behaviors.

Our Experience

One of our customers had the following problem. They had an FDDI backbone network and several Ethernet branch networks. Our RS/6000s were attached to one of these Ethernets. The problem was that these RS/6000s could not communicate with PCs in the other Ethernet. An engineer of the FDDI vendor got a trace and told us that their Ethernet-FDDI connecting module (a kind of bridge) could not pass packets that had a locally administered address. Our RS/6000s had a MAC address of 02:60:8c from 3Com. Finally, the FDDI vendor agreed to modify their firmware.

2.4 Debugging IP Layer with ICMP (ping)

The ping command may be the best command and the first command users issue when they finish the network configuration. Many of them use this command only to find out whether or not the connectivity is fine. By just reading the result, the user can get a lot of information from this command when trying to isolate the problem even when ping fails.

Why PING is Important

Ping can reveal the following options which may help in the problem determination:

- When ping fails, you can not communicate by using the IP protocol.

  Note: Even when ping fails, the ARP may work. You have to check the ARP when ping fails.

- When ping succeeds, you don′t have any hardware and network configuration problems under the IP layer.
- When ping succeeds and you still fail to communicate, there is an application problem (including misconfiguration).

The ping command uses the network layer protocol ICMP. You can confirm the IP layer functionality with other tools too, but ping has the following advantages:

- You don′t need any additional configuration or setup. When you have finished network configuration with smitty mktcpip, you can use ping.
- Almost all systems that support TCP/IP support this command.
- All TCP/IP systems must support ICMP echo. This is documented in *RFC 1122 Requirements for Internet Hosts*. Then, any host that supports TCP/IP must respond to your ping command. You don′t need any additional requirements to be able to communicate with the destination system.

2.4.1 ping Basics

It′s good to know the details about the ping command operation. If you know these details, it will help you to isolate a problem.

2.4.1.1 ping Uses ICMP ECHO/REPLY

The ping command uses the ICMP protocol. This protocol is for IP layer management and control. It is a supplemental protocol for IP operation and doesn′t carry user data. Technically, an ICMP packet is encapsulated in an IP packet. Therefore, ICMP packets, used by the ping command, can confirm the IP layer′s functionality.

The ICMP has several functions and each function has its own type code as in the following list. This is not the complete list. Refer to *RFC 1700 ASSIGN NUMBERS* for the complete list of ICMP types and codes:

ICMP Type	***Message***
0	Echo Reply
3	Destination Unreachable

Code	***Meaning***
0	Network Unreachable
1	Host Unreachable
2	Protocol Unreachable
3	Port Unreachable
4	Fragmentation needed and do not fragment the bit set
5	Source Route Failed
6	Destination Network Unknown
7	Destination Host Unknown
8	Source Host Isolated (Obsolete)
9	Destination Network Administratively Prohibited

	10	Destination Host Administratively Prohibited
	11	Network Unreachable for TOS
	12	Host Unreachable for TOS
	13	Communication Administratively Prohibited by Filtering
	14	Host Precedence Violation
	15	Precedence Cutoff in Effect
4	Source Quench	
5	Route Change Request	
	Code	***Meaning***
	0	Redirect datagrams to go to that network
	1	Redirect datagrams to reach that host
	2	Redirect datagrams for that network with that TOS.
	3	Redirect datagrams for that host with that TOS.
8	Echo Request	
11	Time Exceeded for Datagram	
	Code	***Meaning***
	0	Time-to-live Equals 0 During Transit
	1	Time-to-live Equals 0 During Reassembly
12	Parameter Problem Message	
	Code	***Meaning***
	0	IP Header Bad
	1	Required Option Missing
13	Time Stamp Request	
14	Time Stamp Reply	
15	Information Request	
16	Information Reply	
17	Address Mask Request	
18	Address Mask Reply	

The ping command is built on the ICMP Echo Request (type 8) and ICMP Echo Reply (type 0). Refer to *RFC 1122 Requirements for Internet Hosts* and *RFC 792 INTERNET CONTROL MESSAGE PROTOCOL* for ICMP implementation details.

2.4.1.2 ping Mechanism Overview

The mechanism of ping is described as follows:

1. First, you issue the ping command, as follows. In this example, only one packet is sent to the destination system zero.

   ```
   # ping -c 1 zero
   ```

2. If you are using a name service such as DNS or NIS, the first step is to translate the destination host name to the IP address. In the case of DNS, the following packets may be sent and received. In this example, ping is issued on the system mat, and the destination is the system zero. The DNS server is the system hzname1.

   a. The mat asks the IP address for the DNS server hzname1. If your system doesn't have the MAC address of hzname1, the ARP is invoked before the name service look-up. If you have hardware connectivity problems with the name server, the ARP fails and the ping hangs here.

      Note: In this case, the hzname1 is located in a different IP network (subnet 9.68.192). Then a router is involved in this DNS look-up and the MAC address `00:00:fa:37:91:64` is the router's MAC address.

```
Packet Number 10
TOK: ====( 89 bytes transmitted on interface tr0 )==== 11:10:19.238628608
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 00:00:fa:37:91:64]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.192.11 >  (hzname1.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=67, ip_id=338, ip_off=0
IP:     ip_ttl=30, ip_sum=f272, ip_p = 17 (UDP)
UDP:    <source port=1052, <destination port=53(domain) >
UDP:    [ udp length = 47 | udp checksum = e41f ]
DNS Packet breakdown:
    QUESTIONS:
        zero.hakozaki.ibm.com, type = A, class = IN
```

b. The DNS server hzname1 responded with the IP address of zero.

```
Packet Number 11
TOK: ====( 105 bytes received on interface tr0 )==== 11:10:19.244147200
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 00:00:fa:37:91:64, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.192.11 >  (hzname1.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=83, ip_id=11023, ip_off=0
IP:     ip_ttl=28, ip_sum=caa5, ip_p = 17 (UDP)
UDP:    <source port=53(domain), <destination port=1052 >
UDP:    [ udp length = 63 | udp checksum = 328d ]
DNS Packet breakdown:
    QUESTIONS:
        zero.hakozaki.ibm.com, type = A, class = IN
    ANSWERS:
    ->  zero.hakozaki.ibm.com   internet address = 9.68.214.84
```

Note: If you specify the IP address instead of the host name, you can bypass this name service look-up process. Then you can isolate the name service-related problem.

You May Have a Problem Here

If the name service has a problem, the ping command hangs. In the DNS case, you get timeout and the look-up fails in the local /etc/hosts file. It takes 75 seconds if you have one DNS name server. If you are using NIS, no timeout exists and you have to wait a long time.

If you are not using any name service, your system refers to /etc/hosts. If there are no corresponding entries, you will see the error message `host name NOT FOUND`.

3. IP routing function is invoked based on the destination system′s IP address. Your system refers to the routing table to find the appropriate route.

   If it is in the same IP network, then your system looks up the ARP cache to get the destination′s MAC address. If it cannot be found in the cache, the system uses ARP to get it.

   If it is in the different IP network, then your system looks up the routing table and finds the appropriate gateway system. Then your system looks up the ARP cache to get the gateway′s MAC address. If it cannot be found in the cache, the system uses ARP to get it.

You May Have a Problem Here

If you don′t have the route to the destination IP address, your system cannot send a packet. You will see the error message `A route to the remote host is not available`.

4. Send an ICMP echo request packet. The ping sets the IP header time to live (TTL) field to 255. The value 255 is the maximum. Usually a TTL of 30 is used for UDP datagram and 60 for TCP segment. This TTL value is decremented at least by one when the packet passes a router, and when it reaches 0, the packet is discarded by the router.

```
Packet Number 15
TOK: ====( 106 bytes transmitted on interface tr0 )==== 11:10:19.254320128
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=339, ip_off=0
IP:     ip_ttl=255, ip_sum=fb26, ip_p = 1 (ICMP)
ICMP:   icmp_type=8 (ECHO_REQUEST)  icmp_id=10278  icmp_seq=0
```

5. The destination system receives the ICMP echo request packet. Then it sends back the corresponding ICMP echo reply packet. The destination system also has to have a routing to the source system or the reply packet cannot be sent. The echo reply packet also has the TTL field of 255.

```
Packet Number 16
TOK: ====( 106 bytes received on interface tr0 )==== 11:10:19.256564992
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=496, ip_off=0
IP:     ip_ttl=255, ip_sum=fa89, ip_p = 1 (ICMP)
ICMP:   icmp_type=0 (ECHO_REPLY)  icmp_id=10278  icmp_seq=0
```

Each ICMP packet has an ICMP ID (icmp_id). Corresponding requests and replies have the same ICMP ID.

You May Have a Problem Here

In the destination system, a route to return the ICMP packet must be set. Of course, the subnet masks must match each other. If the route includes more than one gateway, these conditions must be satisfied by all the gateways. Do not forget to configure the return route. You may have to add a route at the destination system and gateways.

6. The ICMP echo reply packet is received by the source system. Then the ping command displays statistics as follows:

```
# ping -c 1 zero
PING zero.hakozaki.ibm.com: (9.68.214.84): 56 data bytes
64 bytes from 9.68.214.84: icmp_seq=0 ttl=255 time=2 ms

----zero.hakozaki.ibm.com PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 2/2/2 ms
#
```

2.4.1.3 ping with Route Information

The ping command has a lot of options. One of the useful options for debugging is the -R option. With this option, this command shows you the path through which the echo request and reply packets have passed. Although you can also check the destination route with the traceroute command, ping -R also tells you the returning route. If there is more than one available route, you will know which route is actually used. See the example below. In this example, we issued the ping command on the newton system. There is the inoki gateway on the way to the destination lazy.

```
# ping -R -c 1 lazy
PING lazy.fscjapan.ibm.com: (9.170.1.9): 56 data bytes
64 bytes from 9.170.1.9: icmp_seq=0 ttl=254 time=6 ms
RR:     inoki.fscjapan.ibm.com (9.170.1.240)
        lazy.fscjapan.ibm.com (9.170.1.9)
        inoki5.fscjapan.ibm.com (9.170.5.240)
        newton.fscjapan.ibm.com (9.170.5.45)

----lazy.fscjapan.ibm.com PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 6/6/6 ms
#
```

In this output, the displayed addresses are for all the interfaces where the packet was going out (and not where the packet was coming in) except the last line. In the previous example, the packet was sent from newton (9.170.5.45) and through one gateway, which has two interfaces: inoki (9.170.1.240) and inoki5 (9.170.5.240).

Note: This function uses the record route option of IP (not ICMP). In this option, up to nine hosts (IP addresses) can be recorded in the IP header option field. This may not be enough for a long path. Also, all the routers on the way must support the record route option. This may not work perfectly or correctly in some environments.

2.4.2 When ping Doesn′t Work

For your reference, a successful example is shown below:

```
# ping jodie
PING jodie.fscjapan.ibm.com: (9.170.3.44): 56 data bytes
64 bytes from 9.170.3.44: icmp_seq=0 ttl=255 time=2 ms
64 bytes from 9.170.3.44: icmp_seq=1 ttl=255 time=2 ms
64 bytes from 9.170.3.44: icmp_seq=2 ttl=255 time=2 ms
64 bytes from 9.170.3.44: icmp_seq=3 ttl=255 time=2 ms
64 bytes from 9.170.3.44: icmp_seq=4 ttl=255 time=2 ms
^C
----jodie.fscjapan.ibm.com PING Statistics----
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 2/2/2 ms
#
```

In the above example, we are using DNS and the host name (exactly the interface name) jodie is translated to jodie.fscjapan.ibm.com automatically. Even for a ping, a rather complicated mechanism is involved. In this case, before sending the first ping packet, a look-up packet to the DNS server was sent and the response packet was received. There are a thousand reasons for the ping command to fail. The reason of a failure is invisible to end users.

2.4.2.1 Try with the IP Address

If the ping doesn′t work, it′s a good practice to try with the IP address, not with the host name. Ping should work find with the IP address if there are no problems. Ping can accept both the IP address and host name as the argument. If the cause is because of the name service you are using, this alternative bypasses the faulty name service. Even when you are referring to /etc/hosts, you can avoid the corrupted entry in the file. If the situation is changed by using the IP address, you will get closer to the cause.

This approach is so powerful that you can use it for other network tools applications. A lot of tools and applications allow you to use both the IP address and host name as their arguments, but this is not always true. Some network application servers need to map.the IP address in a packet header to the host name (this is sometimes called reverse mapping) for verification. For such an application, even when you use an IP address, you cannot bypass name service-related problems.

2.4.2.2 Check to See Whether the ARP Is Working

This is also important. If the ARP doesn′t work, the ping will also not work. If the ARP fails, you have a problem under the data link layer (most likely it′s related to the hardware). You have to fix the ARP first or you cannot check and validate the network layer (IP) configuration.

After you have fixed the ARP problem, if you still have the ping problem, the reason must be in the IP layer configuration.

2.4.3 Error Message: host name NOT FOUND

You may get the following error:

```
# ping gozira
0821-062 ping: host name gozira NOT FOUND
#
```

2.4.3.1 Name Service Problems

In this case, the host name to the IP address translation failed. If you are using a name service such as DNS or NIS, the name service is working correctly. If the name server could not find the corresponding IP address. This implies a registration problem or that the host name is wrong. When you ping with the IP address and get good results, it is a name-to-IP address translation problem. Check your name server database or /etc/hosts.

Note: You may have to wait until you get the previous error. DNS needs at least 75 seconds to get timeout and returns from a name server search.

Our Experience

A customer complained that an update to /etc/hosts was not referred to. As a result, a ping to newly added host system ended up with the error host name NOT FOUND. According to the information, /etc/hosts seemed perfect. We asked whether NIS was running. The answer was yes. The customer didn′t know that the NIS maps need to be rebuilt whenever the ASCII files are changed.

2.4.3.2 Deleted Routing Information

Another possibility of this error is, lost routing information. If the route to the local network for the interface is removed, the system cannot send any packets and so gives you an error message. Checking routing information is very worthwhile. When you get following error, check the routing information:

```
# ping mat
0821-062 ping: host name mat NOT FOUND
#
```

If you get following result, the problem is in the routing information:

```
# netstat -r
Routing tables
Destination      Gateway           Flags     Refs     Use  Interface
Netmasks:
255.255.255.128

Route Tree for Protocol Family 2:
loopback         loopback          UH         4      172  lo0
#
```

Configure the necessary route. Usually this (lost route) should not happen. But if you detach and up an interface manually with the ifconfig command, the route is lost.

2.4.4 Error Message: A Route to the Remote Host Is Not Available

You may get the following error:

```
# ping king
PING king: (9.170.8.25): 56 data bytes
0821-069 ping: sendto: A route to the remote host is not available.
ping: wrote king 64 chars, ret=-1
0821-069 ping: sendto: A route to the remote host is not available.
ping: wrote king 64 chars, ret=-1
0821-069 ping: sendto: A route to the remote host is not available.
ping: wrote king 64 chars, ret=-1
0821-069 ping: sendto: A route to the remote host is not available.
ping: wrote king 64 chars, ret=-1
^C
----king PING Statistics----
4 packets transmitted, 0 packets received, 100% packet loss
#
```

In this case, no packets are sent out. In addition, this ARP procedure is not proceeded and no ARP packets are sent out. The problem is in your system routing table. Your system is saying that it can not find a route to the specified destination and can not send any packets. The possible causes are explained below.

2.4.4.1 No Routes Are Set

In this case, there are no routes to the destination. This may be the easiest case for a ping-related problem if you check to see the routing table and cannot find the route to the destination. In this example below, there are no routes to the IP network 9.170.8:

```
# netstat -r |grep U
Destination      Gateway            Flags  Refcnt Use       Interface
9.170.1          inoki5             UG          0      227  tr0
9.170.5          newton             U           7    41261  tr0
127              loopback           U           0        9  lo0
#
```

Note: If you have a default route, the symptom will be completely different (the ping hangs). In such a case, any packet that has an invalid network address will be sent to the default gateway. Then, you also have to check the routing table of the gateway. Actually, your system gets many ICMP error notification packets (carrying the code for Destination unreachable) from the gateway. This notification does not appear on your display. You should use the netstat command to see the appropriate counter.

2.4.4.2 Corrupted Kernel Routing Table

If you often add and delete routes in your routing table, an inconsistency might be caused in the kernel. As a rule of thumb, if you cannot fix the routing problem, rebooting the system or running cfgmgr may resolve the issue. Remember that routing information is stored in the ODM with the logical device inet0. The reboot procedure loads the ODM information.

2.4.5 No Response, But ping with the IP Address Is OK

This is also one of the easiest cases. The symptom is as follows:

```
# ping gozira
^C#
```

Try ping again with the IP address in order to bypass the host name to the IP address translation function. This time ping should work, as follows:

```
# ping 9.68.210.140
PING 9.68.210.140: (9.68.210.140): 56 data bytes
64 bytes from 9.68.210.140: icmp_seq=0 ttl=255 time=18 ms
64 bytes from 9.68.210.140: icmp_seq=1 ttl=255 time=2 ms
64 bytes from 9.68.210.140: icmp_seq=2 ttl=255 time=2 ms
64 bytes from 9.68.210.140: icmp_seq=3 ttl=255 time=2 ms
^C
----9.68.210.140 PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 2/6/18 ms
#
```

The possible causes are explained below.

2.4.5.1 Name to IP Address Translation Problem

You will notice that no messages are displayed. This is important. This means that the host name to the IP address translation was not working, and this suspends the ping command. The possible causes are given below:

- The name server is not working (NIS or DNS). In the case of the DNS, if the server is not running, the resolver searches the /etc/hosts file after the timeout (75 seconds if you have one DNS server). In the case of the NIS, the ping hangs until you interrupt.
- It is searching through several name servers (DNSs). In this case, your ping will work after the search is completed.

2.4.6 No Response, and ping Fails Even with the IP Address

This case is the toughest for several reasons. The symptom is as follows. In this case, ping hangs until you interrupt by pressing Ctrl-C:

```
# ping grover
PING grover.fscjapan.ibm.com: (9.170.3.21): 56 data bytes
^C
----grover.fscjapan.ibm.com PING Statistics----
23 packets transmitted, 0 packets received, 100% packet loss
#
```

The important symptom is that the ping hangs without responses, but you can see the following line:

```
PING grover.fscjapan.ibm.com: (9.170.3.21): 56 data bytes
```

In this example, the host name grover is replaced with grover.fscjapan.ibm.com and also the corresponding IP address 9.170.3.21 is displayed. These imply that the name service DNS is working correctly and the problem is somewhere else.

2.4.6.1 A Route Is Not Set at the Destination System/Gateway

If the destination system doesn′t have the route to your system, even when the packet can reach the destination, it cannot respond. Routes must be set on both side; the source and the destination. If there is more than one gateway or router between your system and destination, try TELNET to the gateway first. If it works, then try TELNET from the gateway to the destination system. If this works again, there must be the lack of routing at the destination system.

The other possibility is that the gateway system doesn′t know the route to the destination. This could happen easily when you set the default route and default gateway on your routing table. Even if the destination network address is terribly wrong, your system forwards the packet to the default gateway. Then the gateway cannot deliver the packet, and as the result, the ping suspends.

Note: If the gateway also has a default route, again it fowards the packet. This can be repeated forever and the packet will be lost somewhere in the Internet.

In this case, although you may not notice it, the gateway system sends back the ICMP packet with the type code 3, Destination Unreachable. The following is an example received by IP trace:

1. A ping packet is sent to the system king from the system newton. This packet is sent to the default gateway inoki5 first. Notice that the destination MAC address 40:00:70:9b:1d:e4 is the gateway′s MAC address.

```
Packet Number 1
TOK: =====( packet transmitted on interface tr0 )=====Mon Jul 11 20:14:38 1994
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:46:2d, dst = 40:00:70:9b:1d:e4]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =      9.170.5.45 >  (newton)
IP:     < DST =      9.170.8.25 >  (king)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=60758, ip_off=0
IP:     ip_ttl=255, ip_sum=adb8, ip_p = 1 (ICMP)
ICMP:   icmp_type=8 (ECHO_REQUEST)  icmp_id=13216  icmp_seq=0
ICMP: 00000000     2e21299e 00077e13 08090a0b 0c0d0e0f     |.!)...........|
ICMP: 00000010     10111213 14151617 18191a1b 1c1d1e1f     |................|
ICMP: 00000020     20212223 24252627 28292a2b 2c2d2e2f     | !″#$%&′()*+,-./|
ICMP: 00000030     30313233 34353637                       |01234567        |
```

2. The gateway inoki5 could not find the route to the king, discarded the ping packet, and then sent the ICMP DEST UNREACH packet back to the source system newton.

```
Packet Number 2
TOK: =====( packet received on interface tr0 )=====Mon Jul 11 20:14:38 1994
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 18, frame control field = 40
TOK: [ src = 40:00:70:9b:1d:e4, dst = 10:00:5a:a8:46:2d]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.170.5.240 >  (inoki5)
IP:     < DST =      9.170.5.45 >  (newton)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=56, ip_id=59076, ip_off=0
IP:     ip_ttl=255, ip_sum=b68f, ip_p = 1 (ICMP)
ICMP:   icmp_type=3 (DEST UNREACH)  icmp_code=1(BAD_HOST)
```

Note: You can check the count of ICMP packets with the netstat -p icmp command.

If you check the routing table on the gateway system inoki5, you will find the following routes. The inoki5 doesn′t have a route to the destination network 9.170.8 where the destination system king (9.170.8.25) should be located.

```
inoki# netstat -r | grep U
Destination      Gateway            Flags  Refcnt Use      Interface
9.170.1          inoki.fscjapan.ibm U           7   150878  tr1
9.170.2          lazy.fscjapan.ibm. UG          0      206  tr1
9.170.5          inoki5.fscjapan.ib U           6   176794  tr0
127              localhost          U           7     7678  lo0
inoki#
```

Note: If the gateway inoki5 had a default route, the ping echo request would be forwarded automatically to the default gateway. Then the default route may hide this problem. Setting the default route makes administration easier, but it makes the debug more difficult.

If you check the ICMP counter in the kernel of the gateway inoki5, you find that the counter destination unreachable of Output histogram has been incremented by one. In a router or gateway system, the sending of many ICMP destination unreachable packets is not a good sign. This suggests that there may be bad routes on some systems.

```
inoki# netstat -p icmp
icmp:
        210 calls to icmp_error
        0 errors not generated ′cuz old message was icmp
        Output histogram:
                echo reply: 828
                destination unreachable: 207
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo: 828
        828 message responses generated
inoki#
```

On the source system newton, the destination unreachable of Input histogram has been incremented by one. On a system, receiving a lot of ICMP destination unreachable packets is also a bad sign. Check the routes on that system.

```
newton# netstat -p icmp
icmp:
        440 calls to icmp_error
        0 errors not generated ′cuz old message was icmp
        Output histogram:
                echo reply: 15635
                destination unreachable: 440
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo reply: 654
                destination unreachable: 150
                routing redirect: 2
                echo: 15647
                time exceeded: 63
                address mask request: 26
        15635 message responses generated
newton#
```

2.4.6.2 Incorrect Subnet Mask

If there is an IP subnet mask mismatch between your system and the destination system, a ping (ICMP) packet will not reach the destination. Of course, any gateway or router involved also must have a consistent subnet mask. If the mask doesn′t match, a system cannot find the correct route or cannot identify the correct network address.

Currently, a system can have a subnet mask for each network interface and cannot have a subnet mask for each route.

Note: Some of the latest routing techniques, such as OSPF, allow us to have a subnet mask for each route.

If a system has an interface tr0, any route attached to that interface should have the same subnet mask. See the following example. This system has two routes for networks attached to the interface tr0 and both have the same subnet mask. Please note that the Destination field shows network addresses after having the subnet mask applied.

```
# netstat -r | grep U
Destination      Gateway            Flags  Refcnt Use       Interface
9.170.1          lazy.fscjapan.ibm. U           4   236646  tr1
9.170.2          leo.fscjapan.ibm.c U           3      167  en0
9.170.3          aries.fscjapan.ibm U           2    24998  tr0
9.170.5          inoki.fscjapan.ibm UG          2    94447  tr1
9.170.10         jbridge.fscjapan.i UG          0    10553  tr0
127              127.0.0.1          U           2      496  lo0
#
```

The following is an example of a bad configuration. Due to some unknown reason, there is an inconsistency. The mask is 255.255.0.0 for the network 9.170 and 255.255.255.0 for the network 9.170.3. Both networks (routes) are attached to the same interface tr0 and therefore we can say that this is the problem.

```
# netstat -r | grep U
Destination      Gateway            Flags  Refcnt Use       Interface
9.170.3          aries.fscjapan.ibm U           2    24998  tr0
9.170            jbridge.fscjapan.i UG          0    10553  tr0
127              127.0.0.1          U           2      496  lo0
#
```

2.4.6.3 Hardware Problem

In this situation, it is clear that ARP doesn′t work. In such a case, you literally could have hundreds of potential reasons. Our recommended procedures are as follows:

- Confirm that the ARP fails.
- Check the hardware (cables, terminators, transceiver and adapter card). Somebody might have changed something.
- Look at the system error log. You may find some error records regarding the problem.
- Compare the connectivity with another system. Determine whether your system has lost total communication capability or only lost the communication to the specific destination. In such a case, the destination system may have a hardware problem. There is even a possibility that the destination is not powered on.

2.4.6.4 Corrupted Kernel Data

We can not explain clearly which situation would be the case. Just remember that if you cannot isolate a problem and there are no clear symptoms suggesting other causes, rebooting your system may fix the problem.

2.4.7 ping for a While

Although ping doesn′t show any trouble symptoms, you may still be certain that a problem exists. If the problem is an intermittent characteristic or if it happens only a few times in a day, the ping may not catch the symptom. Sometimes in such case (for example, when the response time degrades dramatically or the application hangs for a while) run the ping command until you find something. This may take quite some time. See the following example:

```
# ping inoki5
PING inoki5: (9.170.5.240): 56 data bytes
64 bytes from 9.170.5.240: icmp_seq=0 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=1 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=2 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=3 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=4 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=5 ttl=255 time=2 ms
64 bytes from 9.170.5.240: icmp_seq=6 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=7 ttl=255 time=2 ms
64 bytes from 9.170.5.240: icmp_seq=8 ttl=255 time=2 ms
64 bytes from 9.170.5.240: icmp_seq=9 ttl=255 time=16396 ms
64 bytes from 9.170.5.240: icmp_seq=10 ttl=255 time=15403 ms
64 bytes from 9.170.5.240: icmp_seq=11 ttl=255 time=14404 ms
64 bytes from 9.170.5.240: icmp_seq=12 ttl=255 time=13404 ms
64 bytes from 9.170.5.240: icmp_seq=13 ttl=255 time=12405 ms
64 bytes from 9.170.5.240: icmp_seq=14 ttl=255 time=11405 ms
64 bytes from 9.170.5.240: icmp_seq=15 ttl=255 time=10405 ms
64 bytes from 9.170.5.240: icmp_seq=16 ttl=255 time=9406 ms
64 bytes from 9.170.5.240: icmp_seq=17 ttl=255 time=8406 ms
64 bytes from 9.170.5.240: icmp_seq=18 ttl=255 time=7407 ms
64 bytes from 9.170.5.240: icmp_seq=19 ttl=255 time=6408 ms
64 bytes from 9.170.5.240: icmp_seq=20 ttl=255 time=6007 ms
64 bytes from 9.170.5.240: icmp_seq=21 ttl=255 time=7198 ms
64 bytes from 9.170.5.240: icmp_seq=22 ttl=255 time=6199 ms
64 bytes from 9.170.5.240: icmp_seq=23 ttl=255 time=5200 ms
64 bytes from 9.170.5.240: icmp_seq=24 ttl=255 time=4200 ms
64 bytes from 9.170.5.240: icmp_seq=25 ttl=255 time=3200 ms
64 bytes from 9.170.5.240: icmp_seq=26 ttl=255 time=2201 ms
64 bytes from 9.170.5.240: icmp_seq=27 ttl=255 time=1201 ms
64 bytes from 9.170.5.240: icmp_seq=28 ttl=255 time=201 ms
64 bytes from 9.170.5.240: icmp_seq=29 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=30 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=31 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=32 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=33 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=34 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=35 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=36 ttl=255 time=2 ms
64 bytes from 9.170.5.240: icmp_seq=37 ttl=255 time=16243 ms
64 bytes from 9.170.5.240: icmp_seq=38 ttl=255 time=15251 ms
64 bytes from 9.170.5.240: icmp_seq=39 ttl=255 time=14258 ms
64 bytes from 9.170.5.240: icmp_seq=40 ttl=255 time=13258 ms
64 bytes from 9.170.5.240: icmp_seq=41 ttl=255 time=12259 ms
64 bytes from 9.170.5.240: icmp_seq=42 ttl=255 time=11259 ms
```

```
64 bytes from 9.170.5.240: icmp_seq=43 ttl=255 time=10259 ms
64 bytes from 9.170.5.240: icmp_seq=44 ttl=255 time=9260 ms
64 bytes from 9.170.5.240: icmp_seq=45 ttl=255 time=8260 ms
64 bytes from 9.170.5.240: icmp_seq=46 ttl=255 time=7260 ms
64 bytes from 9.170.5.240: icmp_seq=47 ttl=255 time=6261 ms
64 bytes from 9.170.5.240: icmp_seq=48 ttl=255 time=5261 ms
64 bytes from 9.170.5.240: icmp_seq=49 ttl=255 time=4262 ms
64 bytes from 9.170.5.240: icmp_seq=50 ttl=255 time=3292 ms
64 bytes from 9.170.5.240: icmp_seq=51 ttl=255 time=2293 ms
64 bytes from 9.170.5.240: icmp_seq=52 ttl=255 time=1293 ms
64 bytes from 9.170.5.240: icmp_seq=53 ttl=255 time=293 ms
64 bytes from 9.170.5.240: icmp_seq=54 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=55 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=56 ttl=255 time=2 ms
64 bytes from 9.170.5.240: icmp_seq=57 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=58 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=59 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=60 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=61 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=62 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=63 ttl=255 time=15724 ms
64 bytes from 9.170.5.240: icmp_seq=64 ttl=255 time=14725 ms
64 bytes from 9.170.5.240: icmp_seq=65 ttl=255 time=13727 ms
64 bytes from 9.170.5.240: icmp_seq=66 ttl=255 time=12731 ms
64 bytes from 9.170.5.240: icmp_seq=67 ttl=255 time=11742 ms
64 bytes from 9.170.5.240: icmp_seq=68 ttl=255 time=10746 ms
64 bytes from 9.170.5.240: icmp_seq=69 ttl=255 time=9746 ms
64 bytes from 9.170.5.240: icmp_seq=70 ttl=255 time=8746 ms
64 bytes from 9.170.5.240: icmp_seq=71 ttl=255 time=7747 ms
64 bytes from 9.170.5.240: icmp_seq=72 ttl=255 time=6747 ms
64 bytes from 9.170.5.240: icmp_seq=73 ttl=255 time=5747 ms
64 bytes from 9.170.5.240: icmp_seq=74 ttl=255 time=4747 ms
64 bytes from 9.170.5.240: icmp_seq=75 ttl=255 time=3748 ms
64 bytes from 9.170.5.240: icmp_seq=76 ttl=255 time=2748 ms
64 bytes from 9.170.5.240: icmp_seq=77 ttl=255 time=1749 ms
64 bytes from 9.170.5.240: icmp_seq=78 ttl=255 time=749 ms
64 bytes from 9.170.5.240: icmp_seq=79 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=80 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=81 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=82 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=83 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=84 ttl=255 time=2 ms
64 bytes from 9.170.5.240: icmp_seq=85 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=86 ttl=255 time=3 ms
64 bytes from 9.170.5.240: icmp_seq=87 ttl=255 time=2 ms
^C
----inoki5 PING Statistics----
88 packets transmitted, 88 packets received, 0% packet loss
round-trip min/avg/max = 2/4926/16396 ms
#
```

The above data was obtained from a token-ring network. Someone set the ring speed of the system to the wrong speed. As a result, the network was filled with beacon frames and all the systems attached to the network suffered slow response times. An interesting attribute in this example is that no packets were lost. The longest delay reached 16 seconds, but the source system received all the ICMP echo replies.

2.4.8 ping to the Broadcast Address

An interesting experiment is to ping to the broadcast address. We made this experiment to give you an example of an implementation-dependent issue and to show that standards don′t cover everything. It is not a good practice to ping to the broadcast address because it gives unnecessary traffic to your network. In this example, we are using the subnet mask 255.255.255.128, and therefore, the broadcast address is 9.68.214.127.

```
# ping -c 2 9.68.214.127
PING 9.68.214.127: (9.68.214.127): 56 data bytes
64 bytes from 9.68.214.1: icmp_seq=0 ttl=60 time=6 ms
64 bytes from 9.68.214.69: icmp_seq=0 ttl=255 time=10 ms (DUP!)
64 bytes from 9.68.214.20: icmp_seq=0 ttl=255 time=15 ms (DUP!)
64 bytes from 9.68.214.74: icmp_seq=0 ttl=255 time=29 ms (DUP!)
64 bytes from 9.68.214.70: icmp_seq=0 ttl=255 time=30 ms (DUP!)
64 bytes from 9.68.214.19: icmp_seq=0 ttl=255 time=32 ms (DUP!)
64 bytes from 9.68.214.61: icmp_seq=0 ttl=255 time=33 ms (DUP!)
64 bytes from 9.68.214.29: icmp_seq=0 ttl=255 time=35 ms (DUP!)
64 bytes from 9.68.214.21: icmp_seq=0 ttl=32 time=36 ms (DUP!)
64 bytes from 9.68.214.76: icmp_seq=0 ttl=255 time=38 ms (DUP!)
64 bytes from 9.68.214.17: icmp_seq=0 ttl=255 time=43 ms (DUP!)
64 bytes from 9.68.214.15: icmp_seq=0 ttl=255 time=44 ms (DUP!)
64 bytes from 9.68.214.69: icmp_seq=1 ttl=255 time=2 ms

----9.68.214.127 PING Statistics----
2 packets transmitted, 2 packets received, +11 duplicates, 0% packet loss
round-trip min/avg/max = 2/27/44 ms
#
```

Note: Notice that we are sending two echo request packets with the -c 2 flag. If we only send one echo request, the first echo reply terminates the ping command and we cannot watch what happens.

The message (DUP!) means that we got a duplicated echo reply for the same Echo request. We have numerous RS/6000s and PCs on the network 9.68.214 and some systems have responded. Which should be better, to respond or not to respond? It′s a very difficult argument and we don′t have an answer. This question has been left for each vendor′s implementation. Refer to *RFC 1122 Requirements for Internet Hosts -- Communication Layers* for details.

RFC 1122 Requirements for Internet Hosts, page 42-43

...An ICMP Echo Request destined to an IP broadcast or IP multicast address MAY be silently discarded.

DISCUSSION:

This neutral provision results from a passionate debate between those who feel that ICMP Echo to a broadcast address provides a valuable diagnostic capability and those who feel that misuse of this feature can too easily create packet storms.

See the previous example again. For example, the system of 9.68.214.70 was OS/2 Warp running TCP/IP software and it responded. Almost all other systems are OS/2 machines. We had a bunch of RS/6000s and these kept silent. The behavior is not only vendor or product-dependent, but even version or release-dependent. Even you can find the wrong implementation.

You can configure your RS/6000 running AIX V4.1 whether it responds to a broadcast ping or not. V4.1 has a new option for the no command, bcastping. With this option, you can adjust the behavior of your V4.1 for broadcast ping.

```
# no -o bcastping
bcastping = 0
#
```

If this option is set to 1, V4.1 will respond to a broadcast ping. The default is 0. Issue the following commands and your system will be updated:

```
# no -o bcastping=1
# no -o bcastping
bcastping = 1
#
```

Difference between V4.1 and V3.2

The AIX V3.2.4 and any prior version/release respond to a broadcast ping automatically. AIX V3.2.5 never responds to a broadcast ping. There are no configuration options with V3.2.

2.4.9 ping to the Multicast Address

With AIX V4.1, a new capability, multicast is supported. A group of interfaces shares a multicast address in addition to its own interface address. A packet sent to a multicast address, is received all the interfaces (systems) that have the multicast address. This scheme is suitable for 1 to N relation, such as TV conference application. You can refer to the multicast address of your interface with netstat -i -a command, as follows:

```
# netstat -i -a
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                          1019     0     1019     0     0
lo0   16896 127         localhost           1019     0     1019     0     0
                        224.0.0.1
tr0   1492  <Link>10.0.5a.a8.b5.c1         23820     0     9811     0     0
tr0   1492  9.68.214    mat.hakozaki.ib    23820     0     9811     0     0
                        224.0.0.1
                        c0:00:00:04:00:00
#
```

In this example, interface tr0 has a multicast of 224.0.0.1, in addition to its interface address mat.hakozaki.ibm.com (9.68.214.82). Also, you should notice that a multicast address can have a corresponding MAC address. For token-ring, 224.0.0.1 can be mapped to c0:00:00:04:00:00. In the case of Ethernet, 224.0.0.1 can be mapped to 01:00:5e:00:00:01.

The multicast address 224.0.0.1 has special meaning where all interfaces that support multicast must share this address. If you need to assign other multicast addresses, you have to write a program to do so. You can see a list of current available multicast addresses in *RFC 1700 ASSIGNED NUMERS*. If you need to join the SGI-Dogfight game, you have to assign 244.0.1.2 to your interface.

The following is an experiment to ping the multicast address 224.0.0.1. 224.0.0.1 is to all systems on this subnet; all systems that support multicast in the same subnet should respond. In this example, four systems, including the 224.0.0.1 system responded:

```
# ping -c 2 224.0.0.1
PING 224.0.0.1 (224.0.0.1): 56 data bytes
64 bytes from 9.68.214.82: icmp_seq=0 ttl=255 time=0 ms
64 bytes from 9.68.214.82: icmp_seq=0 ttl=255 time=2 ms (DUP!)
64 bytes from 9.68.214.84: icmp_seq=0 ttl=255 time=3 ms (DUP!)
64 bytes from 9.68.214.77: icmp_seq=0 ttl=255 time=5 ms (DUP!)
64 bytes from 9.68.214.14: icmp_seq=0 ttl=255 time=6 ms (DUP!)
64 bytes from 9.68.214.82: icmp_seq=1 ttl=255 time=0 ms

--- 224.0.0.1 ping statistics ---
2 packets transmitted, 2 packets received, +4 duplicates, 0% packet loss
round-trip min/avg/max = 0/2/6 ms
#
```

1. This is a ping echo request packet. Notice that the destination MAC address is c0:00:00:20:00:00 and the destination IP address is 224.0.0.1.

```
Packet Number 65
TOK: ====( 108 bytes transmitted on interface tr0 )==== 19:00:45.651938304
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 90:00:5a:a8:b5:c1, dst = c0:00:00:20:00:00]
TOK: routing control field = 8220,  0 routing segments
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =       224.0.0.1 >
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=6060, ip_off=0
IP:     ip_ttl=1, ip_sum=e265, ip_p = 1 (ICMP)
ICMP:   icmp_type=8 (ECHO_REQUEST)  icmp_id=14018  icmp_seq=0
```

2. This is one of the response echo reply packets.

```
Packet Number 77
TOK: ====( 106 bytes received on interface tr0 )==== 19:00:45.655625728
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=12415, ip_off=0
IP:     ip_ttl=255, ip_sum=cbfa, ip_p = 1 (ICMP)
ICMP:   icmp_type=0 (ECHO_REPLY)  icmp_id=14018  icmp_seq=0
```

2.5 Debugging IP Routing with traceroute

We only briefly discuss the traceroute command. This command has been introduced into AIX since Version 3.2.5 was released. A lot of vendors also provide this command, and the source code of this command is available through public archives. On the contrary, to the ping command, some systems that have limited resources, such as PCs, and may not have this useful feature.

This command uses UDP as the communication vehicle, and it also uses the ICMP mechanism to identify reachability. For details, refer to the *InfoExplorer* or the *Command Reference* manual.

2.5.1 traceroute Basics

Knowing the internal traceroute gives you a good understanding about the TCP/IP protocol mechanism. This command is an excellent example of ICMP usage, and you can learn how the ICMP works.

2.5.1.1 When traceroute Is Needed

We already have ping for the IP network reachability confirmation. Why do we need traceroute? Although ping is a very convenient tool, we cannot improve some isolated problems effectively. Consider the following situation:

- When you have a lot of hops, say gateways or routes between your system and the destination, there seems to be a problem somewhere on the way. Of course, the destination system may have a problem. You need to know where a packet is lost.
- As we have seen in the previous section, the ping command hangs up and doesn′t tell you the reasons for a lost packet. You need to know why a packet is lost.

The traceroute command can tell you where and why the route is lost. If your packets must pass through routers and links, which belong to and are managed by other organizations or companies, it would be difficult to check the related routers via TELNET. The traceroute command provides a supplemental role to the ping command.

2.5.1.2 traceroute Mechanism Overview

The mechanism of traceroute is described below. In this example, we address the destination ts and, router 9.68.214.1 and s4 are on the way to the ts. The system where we run traceroute is mat:

1. Issue following command on the system mat:

   ```
   # traceroute ts
   ```

   Of course, if you are using some name service, before sending a traceroute packet, the host name to the IP address translation is invoked. If the name server′s MAC address is not in the ARP cache, the ARP is also involved.

2. The traceroute sends a UDP packet to the destination and in this case it is sent to the first router 9.68.214.1. Notice the IP header has the final destination ts, but the token-ring header has MAC address of the router 9.68.214.1, 00:00:fa:37:91:64.

   The most important thing to know is that the time-to-live (ttl) in the IP header is set to 1.

   Note: This packet is sent three times (as default) although we only show one packet. You can change the number of query packets by using the -q option.

```
Packet Number 6
TOK: ====( 62 bytes transmitted on interface tr0 )==== 13:37:43.733352960
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 00:00:fa:37:91:64]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=43597, ip_off=0
IP:     ip_ttl=1, ip_sum=5411, ip_p = 17 (UDP)
UDP:    <source port=43596, <destination port=33435 >
UDP:    [ udp length = 20 | udp checksum = 0 ]
UDP: 00000000     01010000 3032c797 000b2f56             |....02..../V    |
```

3. The router system 9.68.214.1 sends back an ICMP error packet to the source mat. This is ICMP Type 11 Time Exceeded for Datagram. The TTL value must be decremented by one when it passes a router. When the TTL reaches 0, the packet must be discarded and the ICMP error is sent back to the source system. In this case, the 9.68.214.1 received the TTL of 1, it is decremented to 0 before the 9.68.214.1 forwards the packet. Then the UDP packet that caused this error is discarded by the 9.68.214.1.

   Note: Since the packet is sent three times, the ICMP response packet (as follows) is sent back three times.

```
Packet Number 7
TOK: ====( 78 bytes received on interface tr0 )==== 13:37:43.735151488
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 00:00:fa:37:91:64, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =      9.68.214.1 >
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=56, ip_id=36325, ip_off=0
IP:     ip_ttl=60, ip_sum=3204, ip_p = 1 (ICMP)
ICMP:   icmp_type=11 (TIME_EXCEEDED)  icmp_code=0(IN_TRANSIT)
ICMP:   Referenced IP header:
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=43597, ip_off=0
IP:     ip_ttl=1, ip_sum=5411, ip_p = 17 (UDP)
IP:  00000000     aa4c829b 00140000                      |.L......        |
```

   This packet seems to have two IP headers. ICMP error message packets carry a part of the original error-causing packet to help later analyze at the source system. The first 64 bytes are sent back on the ICMP error message packet. In this example, lines after “Referenced IP header” are the returned 64 bytes (the IP header is, of course, included in the first 64 bytes).

4. At this moment, the following intermediate result is displayed:

```
# traceroute ts
traceroute to ts.hakozaki.ibm.com (9.68.210.140), 30 hops max, 40 byte packets
 1  9.68.214.1 (9.68.214.1)  2 ms  2 ms  2 ms
```

5. traceroute sends a UDP packet again, but this time, it increments the TTL in the IP header by one and it becomes 2. Now this packet can pass the router

9.68.214.1 and get one hop ahead. This procedure is repeated until a packet reaches the destination system; Here, the next router s4 is challenged.

Note: This packet is also sent three times, but we only show one.

```
Packet Number 14
TOK: ====( 62 bytes transmitted on interface tr0 )==== 13:37:43.766337152
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 00:00:fa:37:91:64]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=43600, ip_off=0
IP:     ip_ttl=2, ip_sum=530e, ip_p = 17 (UDP)
UDP:    <source port=43596, <destination port=33438 >
UDP:    [ udp length = 20 | udp checksum = 0 ]
UDP: 00000000     04020000 3032c797 000bb048             |....02.....H    |
```

6. This time the UDP packet arrived at the next router s4. When the packet arrived at s4, the TTL decremented to 1 because it passed through the fisrt router 9.68.214.1. Therefore, the s4 cannot forward this packet for the same reason as the 9.68.214.1 did. The same error, ICMP Type 11 Time Exceeded for Datagram is returned to the mat.

```
Packet Number 15
TOK: ====( 78 bytes received on interface tr0 )==== 13:37:43.768731008
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 00:00:fa:37:91:64, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =    9.68.226.123 >  (s4.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=56, ip_id=14500, ip_off=0
IP:     ip_ttl=59, ip_sum=7bcb, ip_p = 1 (ICMP)
ICMP:   icmp_type=11 (TIME_EXCEEDED)  icmp_code=0(IN_TRANSIT)
ICMP:   Referenced IP header:
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=43600, ip_off=0
IP:     ip_ttl=1, ip_sum=540e, ip_p = 17 (UDP)
IP:  00000000     aa4c829e 00140000                      |.L......        |
```

7. Then, the following result is displayed:

```
# traceroute ts
traceroute to ts.hakozaki.ibm.com (9.68.210.140), 30 hops max, 40 byte packets
 1  9.68.214.1 (9.68.214.1)  2 ms  2 ms  2 ms
 2  s4.hakozaki.ibm.com (9.68.226.123)  3 ms  3 ms  3 ms
```

8. The traceroute sends a UDP packet again, but this time, it increments the TTL in the IP header by one and it becomes 3. This time this packet can pass the router 9.68.214.1 and s4 and get one more hop ahead. This procedure is repeated until a packet reaches the destination system; here, it reaches the final destination ts.

Note: This packet is also sent three times, but we only show one.

```
Packet Number 22
TOK: ====( 62 bytes transmitted on interface tr0 )==== 13:37:43.795548544
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 00:00:fa:37:91:64]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=43603, ip_off=0
IP:     ip_ttl=3, ip_sum=520b, ip_p = 17 (UDP)
UDP:    <source port=43596, <destination port=33441 >
UDP:    [ udp length = 20 | udp checksum = 0 ]
UDP: 00000000     07030000 3032c797 000c2287             |....02....″.
```

9. This time the UDP packet arrived at the ts destination system safely, but it could not be delivered to the port 33441; this is because no applications (server daemon, etc) are listening to this port on the ts. Therefore, the ts sent back the ICMP error Destination Unreachable - port unreachable.

```
Packet Number 23
TOK: ====( 78 bytes received on interface tr0 )==== 13:37:43.799451520
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 00:00:fa:37:91:64, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=56, ip_id=7288, ip_off=0
IP:     ip_ttl=253, ip_sum=e5e5, ip_p = 1 (ICMP)
ICMP:   icmp_type=3 (DEST UNREACH)
ICMP:   icmp_code=3 (9.68.210.140: UDP PORT 33441 unreachable, src=43596)
```

Note: Of course, three responses are sent back.

10. Then, the following result is displayed:

```
# traceroute ts
traceroute to ts.hakozaki.ibm.com (9.68.210.140), 30 hops max, 40 byte packets
 1  9.68.214.1 (9.68.214.1)  2 ms  2 ms  2 ms
 2  s4.hakozaki.ibm.com (9.68.226.123)  3 ms  3 ms  3 ms
 3  ts.hakozaki.ibm.com (9.68.210.140)  4 ms  4 ms  4 ms
#
```

Since traceroute uses ICMP, its activity is recorded by the kernel and you can review it with the command netstat -p icmp. Commonly, the counter `destination is unreachable` and `time is exceeded` messages mean that some problems have occurred. You should not forget that debugging tools, such as traceroute, impact system statistics. This is one example of the Heisenberg principle.

```
# netstat -p icmp
icmp:
        28 calls to icmp_error
        0 errors not generated 'cuz old message was icmp
        Output histogram:
                echo reply: 133
                destination unreachable: 28
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo reply: 5
                destination unreachable: 24
                echo: 137
                time exceeded: 36
                address mask request: 3
        133 message responses generated
#
```

The previous example was taken at the system mat. After this experiment, the counters were incremented by 3 (destination unreachable) and 6 (time exceeded), respectively.

2.5.1.3 The Destination Port 33434

With the default setting, it is said that traceroute uses the port 33434. This port is usually not used. Since this command ultimately ends with the Port Unreachable message, the destination port should not be used by any application.

If the port is used by someone, what happens? The UDP module on the destination system can deliver the packet from the traceroute, and the result is unpredictable. The application on the port may silently discard the packet, or the application may print an error message on the console. But we are sure that the ICMP Port Unreachable is not returned. Then, the traceroute cannot know what happened at the destination system. From this command's point of view, it's only a no response and it is possible that the destination system is not even turned on.

In order to avoid the previous problem, traceroute slightly changes the destination port number of each packet. Although this is not clearly documented, our experiment revealed that traceroute increments the destination port number by one whenever it sends a packet. The number 33434 is used to provide the base, and the first packet has the destination port 33435. If your destination can be reached through one router, the router is queried by the ports 3345, 3346, and 3347. Then, the destination system is queried by port 3348, 3349 and 3350.

You can specify an arbitrary base port with the -p flag. Notice that this is the base port and base +1 is used first. See the following example. In this case, the destination hitoh route is in the same IP network and we didn't need a router. The hitoh is queried by ports 111, 112 and 113.

```
# traceroute -p 110 hitoh
traceroute to hitoh.fscjapan.ibm.com (9.170.5.32), 30 hops max, 40 byte packets
 1  * hitoh.fscjapan.ibm.com (9.170.5.32)  5 ms  3 ms
#
```

The port 111 is used by the ONC/RPC portmapper daemon. The portmapper ignored the packet from traceroute and we got the first timed out packet. It was displayed with an asterisk (*).

2.5.2 Successful Example

traceroute uses UDP packets and utilizes the ICMP error reporting function. It sends a UDP packet three times to each gateway or router on the way. It starts with the nearest gateway and expands the search by one hop. Finally, the search gets to the destination system.

In the output, you see the gateway name, the gateway′s IP address and the three round trip times for the gateway. See the following example:

```
# traceroute lazy
traceroute to lazy.fscjapan.ibm.com (9.170.1.9), 30 hops max, 40 byte packets
 1  inoki5.fscjapan.ibm.com (9.170.5.240)  5 ms  4 ms  4 ms
 2  lazy.fscjapan.ibm.com (9.170.1.9)  11 ms  6 ms  6 ms
#
```

Here is another example. We repeated the same command after a while in order to have the ARP entry expired. You should note that the first packet to each gateway or destination took a longer round trip time. This is due to the overhead caused by the ARP. If the public switched network (WAN) is involved in the route, the first packet uses up a lot of memory due to a connection establishment and may cause a timeout. The default timeout for each packet is 3 seconds. You can change it with the -w option:

```
# traceroute lazy
traceroute to lazy.fscjapan.ibm.com (9.170.1.9), 30 hops max, 40 byte packets
 1  inoki5.fscjapan.ibm.com (9.170.5.240)  17 ms  8 ms  4 ms
 2  lazy.fscjapan.ibm.com (9.170.1.9)  20 ms  6 ms  6 ms
#
```

The first 17 ms is due to the ARP between the source system (newton) and the gateway inoki5. The second 20 ms is due to the ARP between the inoki5 and the final destination (lazy). In this case, we are using DNS and we should not forget the DNS lookup overhead. Every time before traceroute sends a packet, the DNS server is searched.

2.5.3 Failed Examples

Now we briefly explain typically failed cases. For a long path to your destination or complex network routes, you may see a lot of troublesome results with the traceroute command. As you already know, there are many things that are left implementation-dependent. Also, we cannot avoid bugs. Thus, searching for the problem may only waste your time. If all routers or systems involved are under your control, you may be be able to investigate the problem completely, but it′s rare.

2.5.3.1 Gateway (Router) Problem

In this example, packets were sent from the system newton (9.170.5.45), and there are two router systems on the way to the bridge. We intentionally removed the routing capability from the second router system (RS/6000) by setting the option ipforwarding of the no command to 0. See the following example:

```
# traceroute jbridge
traceroute to jbridge.fscjapan.ibm.com (9.170.3.253), 30 hops max, 40 byte packets
 1  inoki5.fscjapan.ibm.com (9.170.5.240)  4 ms  4 ms  3 ms
 2  inoki5.fscjapan.ibm.com (9.170.5.240)  3 ms !H  3 ms !H  3 ms !H
#
```

If an ICMP error message, excluding Time Exceeded and Port Unreachable, is received, it is displayed as follows:

!H	Host Unreachable
!N	Network Unreachable
!P	Protocol Unreachable
!S	Source route failed
!F	Fragmentation needed.

Due to product-dependent implementations and bugs, you may see some odd results. Please refer to the sections on traceroute in the Command Reference or InfoExplorer for the odd results.

2.5.3.2 Destination System Problem

When the destination system doesn′t respond, all queries are timed out and the results are displayed with an asterisk. Refer to the following example:

```
# traceroute grover
traceroute to grover.fscjapan.ibm.com (9.170.5.21), 30 hops max, 40 byte packets
 1  inoki.fscjapan.ibm.com (9.170.1.240)  3 ms  3 ms  3 ms
 2  * * *
 3  * * *
^C#
#
```

The most simple reason is that the system is turned off. If you doubt a communication link, you can try with a longer timeout period using the -w flag. Although rare, all the ports queried might have been used. Of course, you can change the ports and try again.

2.6 Debugging TCP/IP Applications

In this section we discuss concrete debugging procedures for the TCP/IP application. There are many applications and you have to choose the appropriate debugging method for each application.

2.6.1 Checking the TCP Connection with TELNET

People use TELNET to log in to other systems via the TCP/IP network. Of course, this is the primary objective of TELNET. We can use TELNET as a common TCP debug tool because you can specify any TCP port number when you invoke the `telnet` or `tn` command. You can force it to connect the TELNET client (`telnet` command) to any TCP application port and can watch the reaction of the application.

2.6.1.1 Successful Example

The following is an example of invoking the TELNET client to the daytime server on the system zero. The default server is the telnetd daemon listening to the well-known port 23.

```
# tn zero daytime
Trying...
Connected to zero.hakozaki.ibm.com.
Escape character is ′[T′.
Thu Aug 17 14:58:03 1995
Connection closed.
#
```

The application daytime server is listening to the well-known port 13. In the previous example, we use the server name daytime to specify the destination port. This is possible because the port is registered in the /etc/services file as follows. Of course, you could use the port number 13 instead.

```
# grep daytime /etc/services
daytime         13/tcp
daytime         13/udp
#
```

Note: Although the daytime server listens to both TCP port 13 and UDP port 13, you can only check the TCP connectivity with the telnet command.

The function of the daytime is fairly simple. It displays the current time and date, then exits. In the previous example, you may notice that the connection was closed by the daytime server. Since the daytime server is invoked by the inetd, the previous procedure checked both the inetd and daytime server.

Note: Precisely speaking, there is no entity of the daytime server daemon. The daytime server is completely integrated with the inetd. So, the previous procedure was used to check a part of the inetd functionality. Some other daemons are also integrated with the inetd. Refer to the 1.6.2, “The inetd Subservers” on page 58 for details.

The following is another example. We plugged in the TELNET client to the FTP server daemon ftpd. Notice that the TELNET client doesn′t have the FTP client functionality; we had to use the USER subcommand to enter a user name, and type the PASS subcommand to enter a password. These subcommands are usually issued by the ftp client program and are not issued by a human user. If you transfer a file, you issue the get or put command to the ftp client. Then the ftp client internally uses the RETR or STOR subcommand. Although we don′t explain in detail, compare this operation sequence with the usual FTP session; this could provide you with the knowledge of the inside of the FTP server.

```
# tn zero ftp
Trying...
Connected to zero.hakozaki.ibm.com.
Escape character is ′[T′.
220 zero FTP server (Version 4.1 Sat Aug 27 17:18:21 CDT 1994) ready.
user matoba
331 Password required for matoba.
pass bmw320i
230 User matoba logged in.
help
214- The following commands are recognized (* =>′s unimplemented).
   USER    PORT    STOR    MSAM*   RNTO    NLST    MKD     CDUP
   PASS    PASV    APPE    MRSQ*   ABOR    SITE    XMKD    XCUP
   ACCT*   TYPE    MLFL*   MRCP*   DELE    SYST    RMD     STOU
   SMNT*   STRU    MAIL*   ALLO    CWD     STAT    XRMD    SIZE
   REIN    MODE    MSND*   REST    XCWD    HELP    PWD     MDTM
   QUIT    RETR    MSOM*   RNFR    LIST    NOOP    XPWD
214 Direct comments to ftp-bugs@zero.
pwd
257 "/home/matoba" is current directory.
stat
211- zero FTP server status:
     Version 4.1 Sat Aug 27 17:18:21 CDT 1994
     Connected to mat.hakozaki.ibm.com
     Logged in as matoba
     No data connection
211 End of status
quit
221 Goodbye.
Connection closed.
#
```

2.6.1.2 Failed Example: System Is Not Up

When the destination system is not turned on, the TELNET looks hung as follows (remember that the port 13 is the daytime server):

```
# tn mat 13
Trying...
```

If you are patient enough and don′t interrupt the telnet command, you will eventually see the error message, as follows:

```
# tn mat 13
Trying...
telnet: connect: Connection timed out
#
```

You have to wait only 75 seconds to get this error. This is because the TCP session initiation timeout is 75 seconds. Up to now these timeout values had been hard-coded and you could not change them. Now the new option tcp_keepinit of the no command has been introduced and you can set your favorite value. Until you get this message, the telnet command looks hung. Of course, the ARP also fails and the ARP cache must be as follows:

```
# arp -a
  ? (9.68.214.1) at 0:0:fa:37:91:64 [token ring]
  mat.hakozaki.ibm.com (9.68.214.82) at (incomplete)
#
```

Until we got the final timeout, a total of six SYN segments were retransmitted. This is due to the implementation of RS/6000, and this behavior complies with the RFC. Notice that finally the error was returned from the TCP module to the application (TELNET client) and this is why the error message was displayed by the telnet command.

RFC 1122 Requirement for Internet Hosts, Page 101

An attempt to open a TCP connection could fail with excessive retransmissions of the SYN segment or by receipt of an RST segment or an ICMP Port Unreachable. SYN retransmissions *must* be handled in the general way described for data retransmissions, including notification of the application layer.

2.6.1.3 Failed Example: Server Is Not Running

One problem may be that an application server cannot start or cannot work correctly. In the case of TELNET, the typical symptom is that the TELNET client ends with the following error message:

```
# tn zero
Trying...
telnet: connect: Connection refused
tn> q
#
```

Note: This message is returned immediately.

The previous message indicates that your system has just received the TCP ACK segment with the RST (Reset) flag. The reset means that the destination system TCP module was forced to have the session initiation request of your system terminated. We show the IP trace example in the following. We prepared this trace by killing the inetd:

1. When you invoke

   ```
   # tn zero
   ```

   the following TCP segment is sent. This is the TCP session initiation request from the system mat. Notice that the destination TCP port is 23 (telnet) and the SYN flag is set.

   ```
   Packet Number 16
   TOK: ====( 66 bytes transmitted on interface tr0 )==== 16:44:23.596971648
   TOK: 802.5 packet
   TOK: 802.5 MAC header:
   TOK: access control field = 0, frame control field = 40
   TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
   TOK: 802.2 LLC header:
   TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
   IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
   IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
   IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=3225, ip_off=0
   IP:     ip_ttl=60, ip_sum=b304, ip_p = 6 (TCP)
   TCP:    <source port=1054, destination port=23(telnet) >
   TCP:    th_seq=b10d3201, th_ack=0
   TCP:    th_off=6, flags<SYN>
   TCP:    th_win=16384, th_sum=b1bb, th_urp=0
   TCP: 00000000     020405ac                                 |....            |
   ```

2. This is the response from the system zero. Notice that the flag is set to both RST and ACK. The ACK number is set to b10d3202 and this was determined by the SEQ number of the request segment, b10d3201. (ACK number is the SEQ number of the next incoming segment. If the connection request segment doesn′t have user data, then SEQ +1 is set to ACK. This mechanism shows that the TCP module is working correctly.) This segment is immediately returned and will close the TCP port at the system mat.

```
Packet Number 17
TOK: ====( 62 bytes received on interface tr0 )==== 16:44:23.598930048
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=8499, ip_off=0
IP:     ip_ttl=60, ip_sum=9e6e, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1054 >
TCP:    th_seq=0, th_ack=b10d3202
TCP:    th_off=5, flags<RST | ACK>
TCP:    th_win=0, th_sum=95d, th_urp=0
```

Note: Although the destination system knows that the application server is not available and cannot deliver TCP segments, no ICMP messages are generated. In case of UDP, you get the ICMP message Destination Unreachable - Port Unreachable.

Do not forget that we are discussing the implementation of the RS/6000 and AIX V4.1.2 as an example. The previous behavior of our RS/6000s (both mat and zero) comply with the *RFC 973 TRANSMISSION CONTROL PROTOCOL, PROTOCOL SPECIFIATION* completely. That is, the SYN segment sent to an application that is not running is rejected by RST segment.

RFC 793 TRANSMISSION CONTROL PROTOCOL, Page 36

...If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means.

If the incoming segment has ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the CLOSED state.

When the RST segment was sent back and the connection was forced to be closed, the following counters shown with the netstat -p tcp command were incremented. These counters are of the system mat. All are incremented by one, as follows:

```
# netstat -p tcp
tcp:
        34166 packets sent
                18782 data packets (309217 bytes)
...
        21 connection requests
        25 connection accepts
        43 connections established (including accepts)
        67 connection closed (including 1 drops)
        1 embryonic connection dropped
        18761 segments updated rtt (of 18765 attempts)
        6 retransmit timeouts
                0 connections dropped by rexmit timeout
        0 persist timeouts
        3 keepalive timeouts
                0 keepalive probes sent
                1 connection dropped by keepalive
#
```

2.6.2 Watching ICMP for UDP Application

If the application uses UDP, the behavior is totally different. Unfortunately, there are no simple tools to directly test a UDP application status. (As you know, traceroute can be used to send a UDP packet.) In the example below, we update the DNS resolver configuration file /etc/resolv.conf and make a DNS query packet to the non-DNS server system. Notice that the DNS query and reply use UDP.

1. A DNS lookup request is sent to the ayano system, which is not the DNS server system. The packet has the destination port of 53 for the named.

```
Packet Number 375
TOK: ====( 87 bytes transmitted on interface tr0 )==== 17:00:55.469524096
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 08:00:5a:ab:2a:e6]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.83 >  (ayano.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=65, ip_id=3706, ip_off=0
IP:     ip_ttl=30, ip_sum=cf04, ip_p = 17 (UDP)
UDP:    <source port=1197, <destination port=53(domain) >
UDP:     udp length = 45 | udp checksum = 30c3
DNS Packet breakdown:
    QUESTIONS:
        ts.hakozaki.ibm.com, type = A, class = IN
```

2. Since the named is not running and the UDP module could not deliver the query packet, it sent back the ICMP message Destination Unreachable - Port Unreachable.

```
    Packet Number 376
    TOK: ====( 78 bytes received on interface tr0 )==== 17:00:55.471707136
    TOK: 802.5 packet
    TOK: 802.5 MAC header:
    TOK: access control field = 18, frame control field = 40
    TOK: [ src = 08:00:5a:ab:2a:e6, dst = 10:00:5a:a8:b5:c1]
    TOK: 802.2 LLC header:
    TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
    IP:     < SRC =     9.68.214.83 >  (ayano.hakozaki.ibm.com)
    IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
    IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=56, ip_id=6778, ip_off=0
    IP:     ip_ttl=255, ip_sum=e21c, ip_p = 1 (ICMP)
    ICMP:   icmp_type=3 (DEST UNREACH)
    ICMP:   icmp_code=3 (9.68.214.83: UDP PORT 53 unreachable, src=1197)
```

As you can see, the result is the same with the traceroute command. Be aware that the previous ICMP is received by the IP module and recorded in the kernel statistics counter. The application (the DNS resolver in the previous example) never gets a response. Therefore, any UDP application that needs a response must have timeout and retransmission or some other mechanism to detect (and recover if necessary) from the problem.

The previous ICMP packet is recorded as follows. You can briefly review the UDP status with netstat -p icmp. Unfortunately, it′s only the sum of destination unreachable counts and doesn′t tell you the details. Although each destination unreachable has the code, it is not recorded.

```
# netstat -p icmp
icmp:
        28 calls to icmp_error
        0 errors not generated ′cuz old message was icmp
        Output histogram:
                echo reply: 1186
                destination unreachable: 27
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo reply: 59
                destination unreachable: 75
                echo: 1186
                time exceeded: 21
                address mask request: 3
        1186 message responses generated
#
```

2.6.3 Monitoring TCP and UDP Connection with netstat -a

You can check the current communication status via the netstat command. This command can show the status of each socket port. This is useful when you connect to the destination application, but you still find something wrong. For example, the connection suddenly hangs or the connection is abruptly closed. The flag -a is used to display all the active TCP connections and all the TCP/UDP server daemons listening on their ports. The flag -f inet is used to display the ports used for networking. If you don′t use this flag, you will also see the UNIX domain socket status.

```
# netstat -a -f inet
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  zero.hakozaki.ib.1094  hzname1.hakozaki.domai TIME_WAIT
tcp        0      0  zero.hakozaki.ib.1093  hzname1.hakozaki.domai TIME_WAIT
tcp        0      0  zero.hakozaki.ib.1092  hzname1.hakozaki.domai TIME_WAIT
tcp        0      0  zero.hakozaki.ib.ftp   mat.hakozaki.ibm.1061  ESTABLISHED
tcp        0      0  zero.hakozaki.ib.1034  mat.hakozaki.ibm.telne ESTABLISHED
tcp        0      0  *.25941                *.*                    LISTEN
tcp        0      0  *.blackjac             *.*                    LISTEN
tcp        0      0  *.writesrv             *.*                    LISTEN
tcp        0      0  *.www                  *.*                    LISTEN
tcp        0      0  *.971                  *.*                    LISTEN
tcp        0      0  *.968                  *.*                    LISTEN
tcp        0      0  *.*                    *.*                    CLOSED
tcp        0      0  *.961                  *.*                    LISTEN
tcp        0      0  *.956                  *.*                    LISTEN
tcp        0      0  *.929                  *.*                    LISTEN
tcp        0      0  *.901                  *.*                    LISTEN
tcp        0      0  *.dtspc                *.*                    LISTEN
tcp        0      0  *.time                 *.*                    LISTEN
tcp        0      0  *.daytime              *.*                    LISTEN
tcp        0      0  *.chargen              *.*                    LISTEN
tcp        0      0  *.discard              *.*                    LISTEN
tcp        0      0  *.echo                 *.*                    LISTEN
tcp        0      0  *.exec                 *.*                    LISTEN
tcp        0      0  *.shell                *.*                    LISTEN
tcp        0      0  *.ftp                  *.*                    LISTEN
tcp        0      0  *.sunrpc               *.*                    LISTEN
tcp        0      0  *.smtp                 *.*                    LISTEN
tcp        0      0  *.6000                 *.*                    LISTEN
udp        0      0  *.1204                 *.*
udp        0      0  *.1203                 *.*
udp        0      0  *.1202                 *.*
udp        0      0  *.1201                 *.*
udp        0      0  *.1200                 *.*
udp        0      0  *.1199                 *.*
udp        0      0  *.1198                 *.*
udp        0      0  *.1197                 *.*
udp        0      0  *.1196                 *.*
udp        0      0  *.1195                 *.*
udp        0      0  *.1194                 *.*
udp        0      0  *.1193                 *.*
udp        0      0  *.1191                 *.*
udp        0      0  *.1189                 *.*
udp        0      0  *.1187                 *.*
udp        0      0  *.1185                 *.*
udp        0      0  *.1183                 *.*
udp        0      0  *.1181                 *.*
udp        0      0  *.1018                 *.*
udp        0      0  *.1019                 *.*
udp        0      0  *.1021                 *.*
udp        0      0  *.1022                 *.*
udp        0      0  *.*                    *.*
udp        0      0  *.*                    *.*
udp        0      0  *.973                  *.*
udp        0      0  *.1023                 *.*
udp        0      0  *.1012                 *.*
udp        0      0  *.966                  *.*
udp        0      0  *.963                  *.*
udp        0      0  *.958                  *.*
udp        0      0  *.927                  *.*
udp        0      0  *.899                  *.*
udp        0      0  *.shilp                *.*
udp        0      0  *.1046                 *.*
udp        0      0  *.time                 *.*
udp        0      0  *.daytime              *.*
udp        0      0  *.chargen              *.*
udp        0      0  *.discard              *.*
udp        0      0  *.echo                 *.*
udp        0      0  *.1044                 *.*
udp        0      0  *.1042                 *.*
udp        0      0  *.1040                 *.*
udp        0      0  *.1038                 *.*
udp        0      0  *.*                    *.*
udp        0      0  *.1036                 *.*
udp        0      0  *.ntalk                *.*
udp        0      0  *.sunrpc               *.*
udp        0      0  *.1035                 *.*
udp        0      0  *.syslog               *.*
udp        0      0  *.src                  *.*
udp        0      0  *.xdmcp                *.*
#
```

The columns Local Address and Foreign Address show the server or client running on your system and the destination system. It is displayed as <host name>*<port number>. If the port number is registered in /etc/services, then the application name is displayed instead of the port number. The asterisk (*) is used to represent a wildcard. This means that any number is acceptable to make a connection. See the following example:

```
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
...
tcp        0      0  *.700                  *.*                    LISTEN
...
```

The TCP server listening to the port 700 can accept a connection from any system (IP address) and any port because Foreign Address is *.*. If your system has more than one network interface (IP addresses), the server can accept a connection from any interface because Local Address is *.700.

A concrete example to explain an established TCP connection is as follows:

```
# netstat -f inet
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  zero.hakozaki.ib.ftp   mat.hakozaki.ibm.1061  ESTABLISHED
tcp        0      0  zero.hakozaki.ib.1034  mat.hakozaki.ibm.telne ESTABLISHE

#
```

In this example, we have two TELNET sessions. The first session is from the mat to the zero. (The zero is the FTP server.) The second session is from the zero to the mat. (The mat is the TELNET server.) This is because ftpd and telnetd use fixed, well-known ports in /etc/services, and the name ftp and telnet are displayed instead of the port number 21 and 23, respectively. For the FTP/TELNET client, the operating system dynamically assigns an available port when they request to open the sockets, and the port numbers 1061 and 1034 are displayed. Usually the lowest available port number of a socket is assigned. If the applications are not registered in /etc/services (this is common if the application is a client program), there is no easy way to know what port the application received (if the application cannot tell you the port explicitly).

Note: Even for a server there are some schemes to assign a port dynamically when the server starts. All the RPC-based applications use port mapper or a similar mechanism for dynamic port assignment.

For UDP ports, state is not displayed. Since the UDP is a connectionless protocol, there are no concepts of state.

Note: Since a UDP application doesn′t have a connection, it would be extremely difficult to know the owner of a port.

The columns Recv-Q and Send-Q are used to show the socket buffer status. If there is some data queued in the socket buffer, the number of bytes are displayed. Usually these columns are 0 because the queues are flushed quicker than you expect. If you want to know when the value is displayed, establish a TELNET session and pull out the Ethernet or token-ring cable or shut the interface down. Then when you type some characters, you can see that the Send-Q gets some number. Of course, during this experiment, the TELNET session screen hangs, but the typed characters are queued.

The most important point for TCP is the right-most column state. See the following for the definitions:

CLOSED	Connection is closed.
LISTEN	Listening for a connection. The port is passively opened and waiting for a connect request.
SYN_SENT	The port is actively opened and has already sent a connection request (a TCP segment with SYN flag). This represents waiting for a matching connection request (a TCP segment with SYN and ACK flags).
SYN_RCVD	Represents waiting for a confirming connection request acknowledgement (a TCP segment with ACK flag). It has received the connection request (a TCP segment with SYN flag) and has also sent a connection request (a TCP segment with SYN and ACK flags). Now it is waiting to receive ACK.
ESTABLISHED	Connection established.
CLOSE_WAIT	It has received a close request (a TCP segment with FIN flag) and has sent back the acknowledgment (a TCP segment with ACK flag). Notice that it can send data, but it can no longer receive data.
FIN_WAIT_1	It has already sent a close request (a TCP segment with FIN flag) and is waiting for an acknowledgment (a TCP segment with ACK flag) or a close request (a TCP segment with FIN flag). Notice that it cannot send data, but it can receive data.
CLOSING	It has sent a close request (a TCP segment with FIN flag). Also, it has received a close request (a TCP segment with FIN flag) and sent back the acknowledgment (a TCP segment with ACK flag).

LAST_ACK	It has received a close request (a TCP segment with FIN flag) and has sent back the acknowledgment (a TCP segment with ACK flag). It has also sent a close request (a TCP segment with FIN flag). If the FIN is sent out from CLOSE_WAIT, it becomes LAST_ACK. It is now waiting for FIN ACK.
FIN_WAIT_2	It has already sent a close request (a TCP segment with FIN flag) and has also received the acknowledgment (a TCP segment with ACK flag). It is waiting for a close request (a TCP segment with FIN flag) from the destination. Since TCP is bidirectional, close (FIN) should be issued at both sides. If FIN_WAIT_1 receives ACK, it becomes FIN_WAIT_2.
TIME_WAIT	The connection was closed completely. For 2MSL (twice the maximum segment lifetime) quiet wait period is needed after close.

Although there are many TCP states, almost all states are only transit states, and it′s very rare to see them whenever you issue the netstat command. Only the following states are stable states:

LISTEN
ESTABLISHED
CLOSED

If any state, except those listed above, remains for a long period of time, this implies that a some problem has occurred. Maybe the most observed state is TIME_WAIT. Due to the internet protocol specification described in the *RFC 1122 Requirements for Internet Hosts*, any closed socket port should not be available or reused immediately. If you can reuse the port just after the close completion, your new connection may receive a stray segment belonging to the prior connection. Then your new connection may be abnormally closed or may get in trouble. To prevent this situation, it is recommended that you wait for a period of twice the maximum segment lifetime (MSL). With our RS/6000, the MSL is 30 seconds. Then you cannot reuse the same port within 60 seconds. It′s not a problem for the TIME_WAIT to be 60 seconds. This value is hard coded and cannot be changed. You can see that this value is defined in the header file /usr/include/netinet/tcp_timer.h. For details, refer to the *RFC 1122*.

Note: If you are absolutely sure that the destination system will open and use a brand new port, you can override the TIME_WAIT restriction using SO_REUSEADDR option of setsockopt().

If you have a program that opens and closes a lot of sockets in a very short period, it would dramatically consume the system resources. If the system is a PC or some resource limited system, it may easily use up all of the memory. Of course, in an extremely rare situation, UNIX is not an exception. In such a case, you would get an error when the application tries to open a socket port.

There is some software that uses a very direct method called hard close. In order to avoid the TIME_WAIT state, those programs use the RST (Rest) flag to close a connection. They don′t follow a normal, graceful closing procedure and only send a TCP segment with the RST flag. This is the same way in which we explained the failed TCP connection establishment. You may find this procedure when you monitor software for PCs.

Our Experience

A customer migrated a TCP application program from a PC to a UNIX workstation. That program didn′t work well in the UNIX environment. The program abnormally terminated with the error message Can′t create socket, address already in use. We helped debug the problem. We found that the PC′s TCP/IP module didn′t have a 2MSL quiet period. It followed the TCP graceful closing procedure but did allow the use of the same port immediately (and the program did so). The program had been built on this inappropriate implementation nature. We spent a long time explaining why a non-workable TCP implementation was correct and a workable TCP implementation was bad.

2.6.4 Socket Port Number Basics

In the TCP/IP world, the IP address is used to identify a system (host) and the port number is used to identify a process (application) running on the system. Both transport protocols, TCP and UDP, have their own port. They use independent port numbers. The port 1000 in TCP is nothing to do with the port 1000 in UDP. The port number is represented in 16 bits, so we can have only 65535 ports. The port number is considered to be a limited system resource.

Usually, a server program uses its own fixed port number. Any server must listen to incoming requests from the clients. Why can the clients send their requests to the correct server port? Because the server is always sitting on the fixed port, the clients can have knowledge about it beforehand. (If the server changes its listening port, we need some mechanism to inform the clients of the new server port.) A server port may be hard-coded in the client program; usually it is described in the file /etc/services. Then the client refers to this file to get the server port number. This means that the server system and the client systems must share the same /etc/services file. Refer to B.2, “Well-Known Ports in /etc/services” on page 350 for a complete example of this file. You can use NIS for this purpose.

Note: For a server program, there are some dynamic port assignment schemes. RPC is one of them.

On the contrary, a client can get its port number at the run time. Usually the operating system assigns the port when the client opens the socket. This is because a client can imbed its port number in a request packet to the server and the server is informed of the client port automatically. Therefore, a client port is dynamically mapped as required.

Note: It is described in many documents that an operating system assigns a random port number to a client program. Accurately speaking, this is not correct. It is not random but non-predictable. For AIX, the kernel assigns a free port. The kernel searches ports from 1024 through 5000 for an unused port, and the first unused port is allocated. If the search reaches the port 5000, then the search starts again from port 1024. So the logic search is not random, but we cannot say which port is assigned beforehand.

We are not saying that a client program cannot choose a specific port explicitly. If you need, you can allocate a specific port using bind(). Of course, if the port you specified has been used by someone, you cannot get it.

2.6.4.1 Well-Known Ports (0-1023)

The ports between 0 and 1023 are controlled by the Internet Assign Number Authority (IANA). These ports are registered in the *RFC 1700 ASSIGN NUMBERS*. On most systems they can only be used by system processes or by programs executed by privileged users (root user or equivalent). These ports are called well-known ports.

This port can be coded in a program, but it is usually written in the /etc/services file. Refer to B.2, “Well-Known Ports in /etc/services” on page 350 for a complete example of this file.

2.6.4.2 Registered Ports (1024-65535)

The Registered Ports are not controlled by the IANA and on most systems can be used by ordinary user processes or programs executed by ordinary users. Although they are not formally controlled, for your reference they are listed in the *RFC 1700 ASSIGN NUMBERS*.

This port can be coded in a program, but it is usually written in the /etc/services file. Refer to B.2, “Well-Known Ports in /etc/services” on page 350 for a complete example of this file.

2.6.4.3 Port Conflict

One potential problem is port conflict. As described above, the registered ports are not controlled formally. If a server program and another server program happen to use the same port, a problem occurs. The problem symptom depends on how the servers are programmed. The most usual symptom may be that the server invoked later terminates or exits with an error message. Then, a user (or a server program) must guarantee the unique port usage among servers. The most common way is to share the /etc/services file to register port numbers. If you have a duplicated port program, you should notice it when you install your program. Of course, it is only possible when all systems and applications are under your control. Whenever you decide to use or introduce some vendor packages, there is some possibility of having a port conflict problem.

You can follow the procedure below to check the unique port usage when you introduce or develop a server program. Keep in mind that there are no procedures to absolutely guarantee the uniqueness.

1. Check the *RFC 1700 ASSIGN NUMBERS*. Do not forget that this document provides only reference information and we may have millions of package software that are not registered.
2. Check the /etc/services file on your system. Do not forget that not all server programs use this file. The most famous example may be X Server of AIXwindow (port 6000). X-Windows, including AIXwindow, don′t use /etc/services.
3. Check the header files and other potential places, such as configuration file or startup script where a port number may be written. For AIXwindow, you can find that the port number is hard-coded in the header file, as follows:

   ```
   # cat -n /usr/include/X11/Xproto.h | grep 6000
      249  #define X_TCP_PORT 6000     /* add display number */
   #
   ```

4. Still, you have a program which has an unknown port; you have to ask the vendor who provided the program for the port.

2.6.4.4 Dynamic Port Assignment (RPC)

Since the socket ports are a limited resource, we have a potential risk of using a conflicting port. Now some dynamic server port assignment mechanisms are commercially available. They are called Remote Procedure Calls (RPCs). With the programs written with RPC, you don′t have port conflict problems and you don′t need to worry about the port assignment administration.

Note: The RPC provides a programming paradigm, and its primary value or purpose is not dynamic server port assignment.

The RPC uses the port mapping service or daemon. When an application server starts, the port mapping service dynamically assigns an available port. The client has to contact the port mapping service first, and after getting the server port it can communicate to the server. In an RPC environment, the only server that uses the fixed port is the port mapping service. Now we have the following RPCs on RS/6000 and AIX:

ONC/RPC Open Network Computing (ONC) developed by Sun Microsystems. It uses the port mapper or portmap daemon for the dynamic server port assignment. The portmap uses the port 111.

NCS/RPC Network Computing System (NCS) developed by HP/Apollo. It uses the local location broker or llbd daemon for the dynamic server port assignment. The llbd uses the port 135.

DCE/RPC Distributed Computing Environment (DCE) developed by OSF. It uses endpoint mapper or rpcd daemon for dynamic server port assignment. The rpcd uses the port 135. If you run both NCS and DCE on the same system, llb and rpcd have a port conflict problem. You cannot run both or only the rpcd because the rpcd includes the functionality of the llb (this is the solution developed by the OSF).

They are different products and basically aren′t compatible with each other.

2.7 Getting TCP Socket-Level Trace

If you know the TCP mechanism very well, you can use the socket-level debug function. This is a trace function provided by the kernel. We can expect most of the UNIX workstations to provide this capability. Although it is not easy to read and understand the trace log, it shows you the contents of Protocol Control Block (PCB), which is quite difficult to review by other methods.

2.7.1 Enabling TCP Socket Trace Function

In order to use the socket-level trace function, the application for which you want to get the trace must have debug mode. The debug mode can be set with the system call setsockopt() specifying SO_DEBUG option. A well-designed application should have a command line option to enable the debug mode. For example, both the TELNET client /usr/bin/telnet and the TELNET server /usr/sbin/telnetd have this option flag. The client uses a -d flag and the server uses an -s flag. Check the manual or InfoExplorer for the applications provided with the system.

2.7.1.1 telnetd Enabling Example

For the TELNET client trace, invoke the client as shown below. This is only one thing you must do:

```
# tn -d inoki5
```

For the TELNET server trace, there are some alternatives to enable the debug mode. The easiest way is to edit the /etc/inetd.conf file and add the flag as follows. Be aware that the telnetd is a subserver of the inetd.

```
##
## service  socket  protocol  wait/  user    server    server program
##  name     type             nowait         program     arguments
##
ftp     stream  tcp     nowait  root    /usr/sbin/ftpd         ftpd
telnet  stream  tcp     nowait  root    /usr/sbin/telnetd      telnetd -s
shell   stream  tcp     nowait  root    /usr/sbin/rshd         rshd
...
```

With the previous procedure, the debug log is written in the ring buffer.

2.7.2 Trace Result Example (TCP Closing Operation)

Now we explain how to get the trace log. You have to use the trpt command to dump the ring buffer. In the ring buffer, the socket debug logs of all the active PCBs are logged together. Without specifying the PCB, you will get all the logs of sockets opened on the system together, and you may not want this. So, first you have to find the address of the necessary PCB. Use netstat -A for this purpose. See the following example:

```
# netstat -f inet -A
Active Internet connections
PCB/ADDR Proto Recv-Q Send-Q  Local Address      Foreign Address    (state)
 5a4bb00 tcp       0      0  mat.hakozaki.telne zero.hakozak.1099  ESTABLISHED
 5a57300 tcp       0    187  mat.hakozaki.telne zero.hakozak.1098  ESTABLISHED
 5a4b800 tcp       0      0  mat.hakozaki.cppbr mat.hakozaki.1026  ESTABLISHED
 5a4bd00 tcp       0      0  mat.hakozaki.1026  mat.hakozaki.cppbr ESTABLISHED
 5a48900 tcp       0      0  mat.hakozaki.cppbr *.*                LISTEN

#
```

Notice that netstat -A is executed on the system mat and the TELNET server telnetd is running on this system. Now you can get the TELNET server′s debug log. If you need the TELNET client′s debug log, you have to do this procedure on the system zero.

Since you got the PCB address, 554af14, issue the trpt command with this address and necessary options. The following example shows the TELNET connection closing procedure. For the meanings of output, you should refer to the manual, InfoExplorer or the following header files:

/usr/include/sys/protosw.h

This header file defines what activities (routines) are logged by a socket-level trace. For example, in this trace, pr_input() is shown as an input. An argument of pr_usrreq(), PRU_SEND, is shown as SEND. For the complete list of this header file, refer to C.4, “/usr/include/sys/protosw.h” on page 374.

/usr/include/netinet/tcp_var.h

In this header file, the window-related parameters, such as snd_una and snd_nxt, are defined. For the complete list of this header file, refer to C.3, "/usr/include/netinet/tcp_var.h" on page 369.

/usr/include/netinet/tcp_timer.h

In this header file, timer-related parameters are defined. In the trace, TCPT_REXMT is shown as REXMT. You can find explanations about each timer in this header file. For the complete list of this header file, refer to C.2, "/usr/include/netinet/tcp_timer.h" on page 367.

The numbers printed in the left-most column are the elapsed time in milliseconds.

```
# trpt -p 5a4bb00 -a

5a4bb00:
936 ESTABLISHED:user RCVD -> ESTABLISHED
937 ESTABLISHED:output (src=9.68.214.82,23, dst=9.68.214.84,1099)
    [c936c2fa..c936c2fb)@c1f93a3b(win=3e64)<ACK,PUSH> -> ESTABLISHED
937 ESTABLISHED:user SEND -> ESTABLISHE
....
104 ESTABLISHED:user RCVD -> ESTABLISHED
104 ESTABLISHED:output (src=9.68.214.82,23, dst=9.68.214.84,1099)
    [c936c2fe..c936c300)@c1f93a40(win=3e64)<ACK,PUSH> -> ESTABLISHED
104 ESTABLISHED:user SEND -> ESTABLISHED
105 FIN_WAIT_1:output (src=9.68.214.82,23, dst=9.68.214.84,1099)
    c936c300@c1f93a40(win=3e64)<ACK,FIN> -> FIN_WAIT_1
105 ESTABLISHED:user DISCONNECT -> FIN_WAIT_1
105 FIN_WAIT_1:user DETACH -> FIN_WAIT_1
105 FIN_WAIT_1:input (src=9.68.214.84,1099, dst=9.68.214.82,23)
    c1f93a40@c936c301(win=3e64)<ACK> -> FIN_WAIT_2
105 FIN_WAIT_2:input (src=9.68.214.84,1099, dst=9.68.214.82,23)
    c1f93a40@c936c301(win=3e64)<ACK,FIN> -> TIME_WAIT
105 TIME_WAIT:output (src=9.68.214.82,23, dst=9.68.214.84,1099)
    c936c301@c1f93a41(win=3e64)<ACK> -> TIME_WAIT
```

The previous example is of a TELNET server (telnetd) when the client terminated the connection by pressing Ctrl+D. You can clearly understand the status transition from ESTABLISHED to FIN_WAIT_1 by the user level DISCONNECT operation. FIN_WAIT_1 was changed to FIN_WAIT_2 by receiving ACK, and FIN_WAIT_2 was changed to TIME_WAIT by receiving ACK/FIN.

2.8 Debugging ONC/RPC Applications

Open Network Computing (ONC) RPC may currently be the most widely used RPC scheme. It provides a programming paradigm for the distributed environment, and you can write your client-server application using RPC. RPC expands the concept of system call or subroutine call onto the network. It hides the underlying complexity of network protocol from the programmers. Many UNIX workstations, including AIX, are shipped with some PRC applications. The most famous and widely used ONC/RPC applications are Network File System (NFS) and Network Information System (NIS).

If you don′t have the intention of writing the ONC/RPC application, just knowing about the RPC mechanism helps you a great deal when you debug the NFS or

NIS problems. You cannot completely understand NFS and NIS without the knowledge of ONC/RPC.

2.8.1 ONC/RPC Basics

A well-designed application would have its own test or debug facility. The ONC/RPC has a common utility rpcinfo. This command can be used to see the status on any ONC/RPC application server. Also, it can test the functionality of the RPC portmap daemon and application servers. Our AIX trace tool iptrace can understand and interpret the RPC packet. Explaining the whole RPC picture is out of the scope of this book. We only cover the minimum knowledge needed.

2.8.1.1 RPC Mechanism Overview

The RPC server and client communicate using the following procedure. This may be the simplest explanation. Since RPC is a way of programming, there can be *many* variations.

1. When the server starts, it registers the port to the portmap daemon running on the same system. This is made by the RPC library routine registerrpc() or svc_register(). The portmap is usually started at the system boot. Any application server must be started after the portmap or it cannot register the port.

   Note: In AIX, the portmap is invoked from the startup script /etc/rc.tcpip.

2. The client contacts the portmap on the destination system and looks up the server port number. This is made by the RPC library routine clnt_create(). The client gets the client handle, which is necessary in order to make contact with the application server. The server port number is imbedded in the client handle.

3. The client sends a request to the application server. This is made by the RPC library routine clnt_call() using the client handle.

4. The server processes the client request and returns the result.

5. The client receives the response.

The client can use the RPC library routine callrpc() instead of both clnt_create() and clnt_call(). This makes the client program simpler but it may introduce excessive operations. Consider the situation where a client needs to send more than one RPC to the server. If a client uses clnt_create(), it can reuse the client handle for each clnt_call(), and it only needs to contact the portmap once. But if the client uses callrpc(), whenever the client sends an RPC request to the server, the portmap is looked up. For performance reasons this is not a good practice. RPC hides the complexity of the network from the programmers. It is an advantage because programmers can concentrate on application logics and do not need to touch the network function directly. But it is very easy to write non-optimized code from a performance point of view.

Note: The callrpc() is currently only available for UDP.

A good example is the RPC tool SPRAY. The spray command looks up the portmap every time before it contacts the sprayd server. It′s a totally unnecessary operation. But if you know the concrete programming methods (as we mentioned above), you can guess that the design objective of the spray command (client) had higher priority on the simplicity.

2.8.1.2 RPC Considerations

With our experience, an RPC program, such as NFS, has a lot of exceptional behaviors that are not clearly documented. For example, NFS read and write operations use UDP as the transport protocol. A lot of commercial books mention that NFS uses UDP; therefore, NFS is not good for unreliable networks such as a WAN. But RPC itself has no limitations to the transport layer protocols. You can use TCP and UDP, and if a program only uses TCP or UDP, it′s totally implementation matter. Unfortunately our AIX and almost all other NFS implementations currently use only UDP for read and write. In AIX, for some operations of NFS, such as querying an export list from the server, TCP is used.

Note: If you get an IP trace of showmount -e command, this command (client) contacts the mountd on the server system. During this query operation, TCP is used.

If you get traces of several implementations, they may show you different behaviors. Current ONC/RPC specification is documented in the *RFC 1050 RPC: Remote Procedure Call Protocol Specification Version 2*, but it allows some implementation flexibilities.

2.8.2 Checking Server Port Registration Status with rpcinfo

See the following example. In this example, the port status of the system mat is displayed. The system rpcinfo just contacted the portmap daemon running on the mat and got the dump list of all available RPC server ports with other crucial information.

```
# rpcinfo -p mat
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100003    2   udp   2049  nfs
    100005    1   udp    768  mountd
    100005    1   tcp    778  mountd
    100024    1   udp    789  status
    100024    1   tcp    791  status
    100021    1   tcp    814  nlockmgr
    100021    1   udp    816  nlockmgr
    100021    3   tcp    819  nlockmgr
    100021    3   udp    821  nlockmgr
    100020    1   udp    824  llockmgr
    100020    1   tcp    826  llockmgr
    100021    2   tcp    829  nlockmgr
    100001    1   udp   1172  rstatd
    100001    2   udp   1172  rstatd
    100001    3   udp   1172  rstatd
    100002    1   udp   1174  rusersd
    100002    2   udp   1174  rusersd
    100008    1   udp   1176  walld
    100012    1   udp   1178  sprayd
    150001    1   udp   1180  pcnfsd
    150001    2   udp   1180  pcnfsd
    100083    1   tcp   1052  ttdbserverd
    100068    2   udp   1182  cmsd
    100068    3   udp   1182  cmsd
    100068    4   udp   1182  cmsd
#
```

As you can see above, an RPC client needs the following information to access a server program.

Server Program Number
Server Program Version Number
Transport Protocol supported by Server Program
Port Number

Among the information listed above, the port number is the most important. In the RPC scheme, any server port is registered dynamically by the portmap at the server start time; you would see a different port for the same server in other systems or after reboot. The only way to get the server port number is to send a request to the portmap.

Note: Other information, such as the program number and version number, are given to the client beforehand; these may be hard-coded. They are displayed only for confirmation. Information that is available only by rpcinfo is the server port number.

The program names listed in the right-most column are quoted from the /etc/rpc file using the program number as the key.

The program number must follow this rule:

0x00000000 - 0x1FFFFFFF	Defined by Sun.
0x20000000 - 0x3FFFFFFF	User-Defined.
0x40000000 - 0x5FFFFFFF	Transient.
0x60000000 - 0xFFFFFFFF	Reserved.

2.8.3 Finding an RPC Server with rpcinfo

The system rpcinfo can send an RPC broadcast packet. With this function you can search the system running the service (the RPC server) that you are interested in. In the example below, we are sending RPC broadcast packets to ask the sprayd Version 1. All systems that are running sprayd Version 1 respond and display.

Note: This RPC broadcast is mapped to the IP broadcast to the local network. Then the RPC broadcast is limited within the same LAN and it cannot pass through a router.

```
# rpcinfo -b spray 1
9.68.214.82 mat.hakozaki.ibm.com
9.68.214.75 guru.hakozaki.ibm.com
9.68.214.88 takesue.hakozaki.ibm.com
9.68.214.81 kishi.hakozaki.ibm.com
9.68.214.37 saka.hakozaki.ibm.com
9.68.214.83 ayano.hakozaki.ibm.com
9.68.214.77 kashima1.hakozaki.ibm.com
9.68.214.84 zero.hakozaki.ibm.com
9.68.214.14 kewpie.hakozaki.ibm.com
^C#
#
```

The system rpcinfo displays the responses in the order of receipt. The order of the display shows the electronical distance. The broadcast packets are sent periodically until you interrupt.

2.8.3.1 RPC Broadcast Mechanism (rpcinfo)

We briefly explain above RPC broadcast using IP trace.

1. This is the RPC Request (CALL) packet. In order to check the server sprayd availability, rpcinfo sends a broadcast to the address 9.68.214.127 (notice we are using subnet mask 255.255.255.128) and the port 111 for the portmap daemon. This procedure only talks to the portmap and doesn't talk to the server itself. The parameters passed to the portmap are the program number of the sprayd 100012 and Version 1.

```
Packet Number 12
TOK: ====( 156 bytes transmitted on interface tr0 )==== 18:35:38.095935488
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 90:00:5a:a8:b5:c1, dst = ff:ff:ff:ff:ff:ff]
TOK: routing control field = 8220,  0 routing segments
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =    9.68.214.127 >
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=132, ip_id=5480, ip_off=0
IP:     ip_ttl=30, ip_sum=c7a7, ip_p = 17 (UDP)
UDP:    <source port=1216, <destination port=111(sunrpc) >
UDP:    [ udp length = 112 | udp checksum = 9122 ]
RPC: **CALL**   XID=808602752
RPC: Program=100000 (PMAPPROG) Version=2 Procedure=5 (PMAPPROC_CALLIT)
RPC: AUTH_UNIX
RPC: Cred:
RPC:    Time=0x30330d6a (Thu Aug 17 18:35:38 1995)
RPC:    Machine=mat Uid=0 Gid=0 Group List Length=6
RPC:    Groups= ( 0 2 3 7 8 10 )
PMP: Prog=100012 Vers=1 Proc=0
PMP: Parms:
```

2. This is the corresponding RPC Response (REPLY) packet. This is a reply from the takesue system and usually you may receive more than one RPC reply. Please note that the corresponding REPLY must have the same transaction ID (XID) with the CALL. The portmap on the hitoh also responded with the port number of sprayd, 1038.

   Note: Before sending the REPLY, the system may have to invoke the ARP procedure. Just after the RPC broadcast CALL, your network may be filled with ARP broadcasts. Then it is not a good practice to run the rpcinfo broadcast function so often.

```
Packet Number 29
TOK: ====( 82 bytes received on interface tr0 )==== 18:35:38.139991680
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 18, frame control field = 40
TOK: [ src = 10:00:5a:b1:87:58, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.88 >  (takesue.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=60, ip_id=38381, ip_off=0
IP:     ip_ttl=30, ip_sum=4791, ip_p = 17 (UDP)
UDP:    <source port=111(sunrpc), <destination port=1216 >
UDP:    [ udp length = 40 | udp checksum = ba7a ]
```

```
    RPC: **REPLY**  XID=808602752
    RPC: 100000(PMAPPROG) 5(PMAPPROC_CALLIT)
    RPC: Reply Stat: MSG_ACCEPTED
    RPC: Accepted Reply Stat:  SUCCESS
    PMP: Port=1038
    PMP: Parms:
```

As you can see, this procedure, rpcinfo -b, only contacts the portmap. If you see the application server status displayed, it doesn′t really mean the server is currently up and running. Well-designed RPC servers unregister their port and other information from the portmap using the RPC library call svc_unregister(). But this doesn′t cover abnormal server termination.

2.8.4 Checking Server Status with rpcinfo

You can check an RPC server status directly. With this procedure, the rpcinfo directly contacts the RPC server daemon and you can find out if the server is really working. The following example shows that the NFS server daemon nfsd is working on the inoki5 system. The option -u means that UDP is asked. Currently for the NFS, only Version 2 is available and this response is fine.

```
rpcinfo -u mat nfs
program 100003 version 2 ready and waiting
#
```

This is another example. The RPC server status monitor (or rpc.statd) has three versions and all of them are available, as follows:

```
# rpcinfo -u ayano rstat
program 100001 version 1 ready and waiting
program 100001 version 2 ready and waiting
program 100001 version 3 ready and waiting
#
```

Be aware that these operations follows a complete RPC operation. This means that the rpcinfo first contacts the portmap and gets the port number, then accesses the target server. Another interesting example is mount daemon or rpc.mountd. As we already mentioned, this daemon supports both TCP and UDP; it responds to both -u and -t, as follows:

```
# rpcinfo -t mat mount
program 100005 version 1 ready and waiting
# rpcinfo -u mat mount
program 100005 version 1 ready and waiting
#
```

On the contrary, NFS daemon or nfsd supports only UDP. It returns an error when you specify -t. This result shows that our current NFS implementation can not use TCP for a read and write operation.

```
# rpcinfo -u mat nfs
program 100003 version 2 ready and waiting
# rpcinfo -t mat nfs
rpcinfo: RPC: Program not registered
program 100003 is not available
#
```

Notice that this error was returned by the portmap and not by the nfsd. Because the nfsd doesn′t register any port for TCP, the portmap cannot return the port number for TCP to the rpcinfo.

2.8.5 RPC Mechanism and Pitfalls

We explain the mechanism involved by using the following procedure:

```
# rpcinfo -u mat nfs
program 100003 version 2 ready and waiting
#
```

Although the operation and response are fairly simple, the activity invoked in the background could be more than you expected.

1. rpcinfo sends the port lookup request (PMAPPROC_GETPORT) to the portmap (PMAPPROG). The search parameters are Prog=100003, Vers=0, Prot=17 and Port=0. Notice that this time the Version 0 is specified. There are five procedures defined to the portmap, as follows (any RPC server has a set of its unique procedures):

   PMAPPROC_NULL(0)
   Takes nothing, returns nothing.

   PMAPPROC_SET(1)
   Registers the tuple [prog, vers, prot, port].

   PMAPPROC_UNSET(2)
   Un-registers pair [prog, vers]. prot and port are ignored.

   PMAPPROC_GETPORT(3)
   0 is failure. Otherwise it returns the port number where the pair [prog, vers] is registered. It may be incorrect.

   PMAPPROC_DUMP(4)

   PMAPPROC_CALLIT(5)
   Calls the procedure on the local machine. If it is not registered, this procedure is quiet; in other words, it does not return error information. This procedure is only supported on RPC/UDP and calls via RPC/UDP. This routine only passes the null authentication parameters.

   You can refer to the header file /usr/include/rpc/pmap_prot.h.

   ```
   Packet Number 52
   TOK: ====( 106 bytes received on interface tr0 )==== 19:20:27.892176384
   TOK: 802.5 packet
   TOK: 802.5 MAC header:
   TOK: access control field = 10, frame control field = 40
   TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
   TOK: 802.2 LLC header:
   TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
   IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
   IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
   IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=14109, ip_off=0
   IP:     ip_ttl=30, ip_sum=a64d, ip_p = 17 (UDP)
   UDP:    <source port=1285, <destination port=111(sunrpc) >
   UDP:    [ udp length = 64 | udp checksum = 5b5 ]
   RPC: **CALL**   XID=809236348
   RPC: Program=100000 (PMAPPROG) Version=2 Procedure=3 (PMAPPROC_GETPORT)
   RPC: AUTH_NULL Opaque Authorization Base 0 Opaque Authorization Length 0
   PMP: Prog=100003 Vers=0 Prot=17 Port=0
   ```

2. The portmap responds the port number of NFS daemon (Prog=100003), 2049. There are two reply statuses defined in ONC/RPC. They are as follows:

   MSG_ACCEPTED(0)
   RPC was accepted by server.

SUCCESS(0)
RPC executed successfully.

PROG_UNAVAIL(1)
Remote hasn′t exported the program.

PROG_MISMATCH(2)
Remote hasn′t supported a version number.

PROC_UNAVAIL(3)
Remote hasn′t supported procedure.

GARBAGE_ARGS(4)
Procedure can′t decode parameters.

SYSTEM_ERR(5)
???

MSG_DENIED(1)
RPC was not accepted by server.

RPC_MISMATCH(0)
RP version number is not 2.

AUTH_ERROR(1)
Remote can′t authenticate caller.

You can refer to the header file /usr/include/rpc/rpc_msg.h.:

```
Packet Number 53
TOK: ====( 78 bytes transmitted on interface tr0 )==== 19:20:27.892853376
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=56, ip_id=7195, ip_off=0
IP:     ip_ttl=30, ip_sum=c16b, ip_p = 17 (UDP)
UDP:    <source port=111(sunrpc), <destination port=1285 >
UDP:    [ udp length = 36 | udp checksum = b49 ]
RPC: **REPLY**  XID=809236348
RPC: 100000(PMAPPROG) 3(PMAPPROC_GETPORT)
RPC: Reply Stat: MSG_ACCEPTED
RPC: Accepted Reply Stat:  SUCCESS
PMP: Returning 2049
```

3. rpcinfo sends the RPC call (RFS_NULL) to the NFS daemon (port 2049) in order to check the server status. This time, the Version 0 is specified and asked. In ONC/RPC convention, the procedure=0 means NULL procedure for any server and the servers should respond NULL REPLY (XDR_VOID). This procedure is for testing or debugging purposes.

```
Packet Number 54
TOK: ====( 90 bytes received on interface tr0 )==== 19:20:27.896235136
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
```

```
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=68, ip_id=14110, ip_off=0
IP:     ip_ttl=30, ip_sum=a65c, ip_p = 17 (UDP)
UDP:    <source port=1286, <destination port=2049(shilp) >
UDP:    [ udp length = 48 | udp checksum = 86d3 ]
RPC: **CALL**   XID=809235874
RPC: Program=100003 (NFS_PROGRAM) Version=0 Procedure=0 (RFS_NULL)
RPC: AUTH_NULL Opaque Authorization Base 0 Opaque Authorization Length 0
RPC: NULL PROC
```

4. The server NFS daemon returns PROG_MISMATCH since nfsd doesn't support Version 0. Although the version number is also registered in the portmap, the version number mismatch never causes an error at the portmap lookup. (Remember the protocol mismatch causes an error at the portmap lookup.) This is the ONC/RPC specification.

```
Packet Number 55
TOK: ====( 82 bytes transmitted on interface tr0 )==== 19:20:27.896596864
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=60, ip_id=7196, ip_off=0
IP:     ip_ttl=255, ip_sum=e065, ip_p = 17 (UDP)
UDP:    <source port=2049(shilp), <destination port=1286 >
UDP:    [ udp length = 40 | udp checksum = d83 ]
RPC: **REPLY**  XID=809235874
RPC: 100003(NFS_PROGRAM) 0(RFS_NULL)
RPC: Reply Stat: MSG_ACCEPTED
RPC: Accepted Reply Stat:  PROG_MISMATCH
RPC: XDR_VOID
```

5. Again, this time the rpcinfo sends the port lookup request (PMAPPROC_GETPORT) to the portmap (PMAPPROG). Notice that this time the Version 2 is specified. We don't know why 2 is specified after 0, skipping the Version 1.

```
Packet Number 56
TOK: ====( 106 bytes received on interface tr0 )==== 19:20:27.899750144
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=14111, ip_off=0
IP:     ip_ttl=30, ip_sum=a64b, ip_p = 17 (UDP)
UDP:    <source port=1287, <destination port=111(sunrpc) >
UDP:    [ udp length = 64 | udp checksum = 6d11 ]
RPC: **CALL**   XID=809209884
RPC: Program=100000 (PMAPPROG) Version=2 Procedure=3 (PMAPPROC_GETPORT)
RPC: AUTH_NULL Opaque Authorization Base 0 Opaque Authorization Length 0
PMP: Prog=100003 Vers=2 Prot=17 Port=0
```

6. The portmap responds to the port number of the NFS daemon (Prog=100003), 2049. Remember that the portmap doesn′t check the version number, and if the rpcinfo (client) caches the port number, this procedure is completely useless. We don′t know the coding of the rpcinfo; it is not optimized for performance.

```
Packet Number 57
TOK: ====( 78 bytes transmitted on interface tr0 )==== 19:20:27.900630144
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=56, ip_id=7197, ip_off=0
IP:     ip_ttl=30, ip_sum=c169, ip_p = 17 (UDP)
UDP:    <source port=111(sunrpc), <destination port=1287 >
UDP:    [ udp length = 36 | udp checksum = 72a7 ]
RPC: **REPLY**  XID=809209884
RPC: 100000(PMAPPROG) 3(PMAPPROC_GETPORT)
RPC: Reply Stat: MSG_ACCEPTED
RPC: Accepted Reply Stat:  SUCCESS
PMP: Returning 2049
```

7. rpcinfo sends the RPC call (RFS_NULL) to the NFS daemon (port 2049) in order to check the server status. This time, the Version 2 is specified and asked. In the ONC/RPC convention, the procedure=0 means that the NULL procedure and servers should respond NULL REPLY (XDR_VOID) as we already explained.

```
Packet Number 58
TOK: ====( 90 bytes received on interface tr0 )==== 19:20:27.903449344
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=68, ip_id=14112, ip_off=0
IP:     ip_ttl=30, ip_sum=a65a, ip_p = 17 (UDP)
UDP:    <source port=1286, <destination port=2049(shilp) >
UDP:    [ udp length = 48 | udp checksum = ec52 ]
RPC: **CALL**   XID=809209889
RPC: Program=100003 (NFS_PROGRAM) Version=2 Procedure=0 (RFS_NULL)
RPC: AUTH_NULL Opaque Authorization Base 0 Opaque Authorization Length 0
RPC: NULL PROC
```

8. This time the reply status is SUCCESS and the rpcinfo knows that NFS_PROGRAM Version 2 is up and running.

```
Packet Number 59
TOK: ====( 74 bytes transmitted on interface tr0 )==== 19:20:27.903854720
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=52, ip_id=7198, ip_off=0
IP:     ip_ttl=255, ip_sum=e06b, ip_p = 17 (UDP)
UDP:    <source port=2049(shilp), <destination port=1286 >
UDP:    [ udp length = 32 | udp checksum = 731a ]
RPC: **REPLY**  XID=809209889
RPC: 100003(NFS_PROGRAM) 0(RFS_NULL)
RPC: Reply Stat: MSG_ACCEPTED
RPC: Accepted Reply Stat:  SUCCESS
RPC: XDR_VOID
```

As you know, RPC is quite a heavy mechanism. Just checking a server status took four RPCs (and they were mapped to eight IP datagrams). Of course, this doesn′t include ARP and DNS lookup packets. Get the IP trace log by yourself and read it. You will realize how many activities are involved behind a simple (from a user′s point of view) operation.

We saw that an anomaly, which was the port search of the Version 1, was skipped. We show you another example. We tried the following command and got the IP trace log. The trace revealed that the port search began from the Version 0 through the Version 3, and no versions were skipped. Since Version 2 is not a supported version, the following error displayed is obvious. (You can confirm that Version 2 is not registered on the portmap by issuing rpcinfo.)

```
# rpcinfo -u mat nlockmgr
program 100021 version 1 ready and waiting
rpcinfo: RPC: Procedure unavailable
program 100021 version 2 is not available
program 100021 version 3 ready and waiting
#
```

We cannot explain the anomaly we found at the NFS daemon. Although we don′t mention more now, we found some other similar behaviors.

Our Experience

When we got an IP trace log of the mount command during the NFS mount operation, we found the mount command didn′t use the portmap to get the port of NFS daemon nfsd 2049. It is directly sent RPC to the nfsd and we had to conclude that the port number 2049 was hard-coded in the mount command.

This exception was due to the development history of NFS and RPC. Sun Microsystems developed both NFS and RPC simultaneously. NFS is built on top of the RPC, but NFS had to be coded before the RPC specification was fixed. As a result, nfsd got the fixed port of 2049. This is the only server that has a xed port (except the portmap), and this may be the most known anomaly of NFS. Of course (the nfsd registers the port to the portmap), you can still use the portmap for consistency.

2.9 Using syslog

Usually any well-designed application should have its own debug mode or trace function. In the UNIX world, the daemon syslogd provides general logger function if an application is coded to use it. You can configure a syslog server in your network and gather all the other system′s log information in one place through the network.

If you would like to get a log of an application, you have to turn the application debug mode on. Of course, the application must be coded with syslog API such as syslog(), openlog(), closelog(), or setlogmask(). The concrete procedure to turn the mode on should be the specific application, and you should refer to the application′s document or manual. If the application is implemented as a daemon, the daemon may have a -d or -D flag. Since it is impossible to explain all the cases, we only introduce the procedures for inetd, telnetd and ftpd in this section. They are the most used applications and have crucial roles for daily operation.

Due to the unique features of AIX, you need some knowledge about the ODM and the System Resource Controller (SRC) when you configure syslogd. You can refer to *AIX Version 4.1 System Management Guide: Operating System and Devices*, SC23-2525 and *AIX Version 4.1 General Programming Concepts*, SC23-2205.

2.9.1 Configuration File /etc/syslog.conf

The syslogd uses the configuration file /etc/syslog.conf. You have to write your logging requests with the format that the syslogd can understand. You have to specify both the facility and level for logging. The facility represents an instance which generates a log message. The level represents how the log message should be processed or treated.

2.9.1.1 Facility

The facility is a group to which the programs belong. It allows you to manage a group of messages by specifying the facility name. When you write a program, you can use the call openlog() to specify the facility for your program. The supported facilities are shown below. Refer to the header file /usr/include/sys/syslog.h for details.

Facility	***Description***
kern	Kernel-generated messages are grouped in this facility.
user	User Process-generated messages are grouped in this facility. When an end user needs to use the syslogd, this facility is usually used.
mail	Mail subsystem-generated messages are grouped in this facility.
daemon	System daemons such as ftpd- and routed-generated messages are grouped in this facility.
auth	Security- or authorization-related messages are grouped in this facility. For example, messages generated by login and su belong to this facility.
syslog	syslogd daemon-generated messages are grouped in this facility.

lpr	Line-printer subsystem-generated messages are grouped in this facility. For example, messages generated by lpd and lpr belong to this facility.
news	News subsystem-generated messages are grouped in this facility.
uucp	uucp subsystem-generated messages are grouped in this facility.
local0 - local7	Locally used facility by each system. Some applications use this facility. For example, AIX/DCE security server daemon secd uses the facility local6 to log the audit messages.
*****	All facilities (wildcard)

2.9.1.2 Level

The level represents the priority of the log message and you can define how the message is treated. When you write a program, you can use the call syslog() to generate a message, and at the moment you also have to specify the level. Refer to the header /usr/include/sys/syslog.h. file. The supported levels are shown below. They are listed in order of highest priority. When you specify a level, all the higher levels are considered to be specified and they will be treated as you defined them.

Level	***Description***
emerg	Specifies emergency messages (LOG_EMERG), such as fixed-disk errors. LOG_EMERG priority messages can be logged into a separate file for reviewing.
alert	Specifies important messages (LOG_ALERT), such as a serious hardware error.
crit	Specifies critical messages not classified as errors (LOG_CRIT), such as improper login attempts. LOG_CRIT and higher-priority messages can be sent to the system console.
err	Specifies messages that represent error conditions (LOG_ERR), such as an unsuccessful disk write. Warning specifies messages for abnormal, but recoverable conditions (LOG_WARNING).
notice	Specifies important informational messages (LOG_NOTICE). Messages without a priority designation should be mapped into this priority message.
info	Specifies informational messages (LOG_INFO). These messages can be discarded, but are useful in analyzing the system.
debug	Specifies debugging messages (LOG_DEBUG). These messages may be discarded.
none	Excludes the selected facility. This priority level is useful only if preceded by an entry with an asterisk (*) in the same selector field.

2.9.1.3 /etc/syslog.conf Format

You have to specify the selector and action in this file. The selector is a combination of facility and level separated with a colon. For example, user.err is a selector. The action is to define where the messages specified by the selector are to be sent. Usually a file such as /var/adm/syslog/user_err.log is used for action. See the following example:

```
mail.debug          /usr/spool/mqueue/syslog
*.debug             /dev/console
*.crit              *
user.notice      @inoki5
```

Mail messages, at debug or higher, go to the log file /usrspool/mqueue/syslog. The file must exist before the syslogd starts. All facilities at debug and higher go to console. All facilities at crit or higher go to all user′s terminals. User messages at notice or higher go to the host inoki5. The syslogd at the system inoki5 treats the forwarded messages based on its configuration file.

There is no default treatment for a selector. If you specify a selector without the action as follows, it is ignored.

```
auth.debug
```

If you write a selector more than once, they are all granted and executed. If you specified it as follows, the messages are displayed at the console and also logged in the file.

```
auth.err; /dev/console
auth.err; /var/adm/syslog/auth.error.log
```

2.9.2 Configuration Procedure

A detailed configuration procedure is described here. In the RS/6000, the syslogd is under the control of the SRC. The configuration procedure looks different from the other vendor′s syslogd. Although you can run the syslogd without the SRC, it′s not a common AIX method.

1. Edit the syslogd configuration file /etc/syslog.conf. You could use other files, and in such a case, you have to specify the configuration file with -f when you start the syslogd. You have to specify facility, level, and action.

   If you don′t know what facility and level you should choose, use the asterisk (*) for facility. Use debug or the asterisk (*) for level. You may get a lot of messages in the file, but we believe that it is still a manageable amount.

   Note: We don′t recommend this approach for system trace or other trace facilities. They usually build up so quickly that it is like looking for a particular pebble on the beach.

   You can specify arbitrary file in the action field as follows:

   ```
   *.debug                 /var/adm/syslog/all_debug.log
   ```

2. Create the file explicitly for logging. This is important because the syslogd doesn′t create a new file automatically.

   ```
   # touch /var/adm/syslog/all_debug.log
   ```

3. Restart the syslogd daemon. In AIX the syslogd is under the control of the System Resource Controller srcmstr; it is recommended that you use the commands stopsrc and startsrc, or refresh. If the syslogd has it already running, just refresh it as follows:

   ```
   # refresh -s syslogd
   0513-095 The request for subsystem refresh was completed successfully.
   #
   ```

   If the syslogd is not running, just start it as follows:

```
# startsrc -s syslogd
0513-059 The syslogd Subsystem has been started. Subsystem PID is 5219.
#
```

Note: You can start the syslogd from the startup script /etc/rc.tcpip when the system boots if you need to start the syslogd automatic. Refer to 1.6.3, "Other Network Subsystems (Servers)" on page 63 for details.

4. Confirm whether the syslogd has read the configuration file as follows:

```
# lssrc -ls syslogd
Subsystem         Group            PID     Status
 syslogd          ras              5219    active

Syslogd Config   *.debug           /var/adm/syslog/all_debug.log


#
```

5. Check to see whether the logging function is working. At this stage, at least the restart of the syslogd should be recorded if you have specified it to do so. Notice in this example we used *.debug and this covered the facility syslog.

```
# cat /var/adm/syslog/all_debug.log
Jul 12 16:44:14 newton syslogd: restart
#
```

2.9.2.1 Interactive Logging with logger Command

The syslogd daemon provides a command line interface to make a log. You can use the command logger to add an arbitrary character string in the file specified in /etc/syslogd.conf. See the following example:

```
# logger -p debug 'I will debug inetd!'
#
```

Then the message was logged as follows:

```
# tail all_debug.log
...
Aug 17 20:21:12 mat inetd[5446]: ADD : time proto=udp, wait=0, user=root builtin=20000d20 server=internal
Aug 17 20:21:12 mat inetd[5446]: ADD : ttdbserverd proto=tcp, wait=1,
user=root builtin=0 server=/usr/dt/bin/rpc.ttdbserverd
Aug 17 20:21:12 mat inetd[5446]: ADD : dtspc proto=tcp, wait=0, user=root builtin=0 server=/usr/dt/bin/dtspcd
Aug 17 20:21:12 mat inetd[5446]: ADD : cmsd proto=udp, wait=1, user=root builtin=0 server=/usr/dt/bin/rpc.cmsd
Aug 17 20:22:41 mat root: I will debug inetd!
#
```

2.9.3 inetd Example

The daemon inetd is the most used application and has crucial roles for TCP/IP communication. Many TCP/IP server daemons, such as ftpd and telnetd, are invoked by the inetd. This inetd daemon is controlled by the SRC and uses the ODM when necessary.

2.9.3.1 Invoking the inetd in Debug Mode

Since the inetd is controlled by srcmstr, it is recommended that you use the commands stopsrc and startsrc. This procedure makes the debug mode effective until you reboot the system or stop the inetd daemon.

1. Stop the inetd.

```
# stopsrc -s inetd
0513-044 The stop of the /usr/sbin/inetd Subsystem was completed successfully.
#
```

2. Start the inetd with the -d flag.

```
# startsrc -s inetd -a ″-d″
0513-059 The inetd Subsystem has been started. Subsystem PID is 5446.
#
```

3. Confirm whether the debug mode was really set.

```
# ps -ef | grep inetd | grep -v grep
    root  5446  2748   0 20:21:12      -  0:00 /usr/sbin/inetd -d
#
```

An alternative is shown below:

```
# lssrc -ls inetd | grep Debug
Debug         Active
#
```

Note: You can do the same thing with the SMIT. Invoke the SMIT with the fast path smitty tracessyson and choose the inetd by pressing the PF4 key. The high-level command traceson is issued internally.

If you want to permanently set the debug mode (in other words, whenever you start the inetd you want to have the debug mode turn on) you have two options. One is to update the ODM. The other is to update the startup script /etc/rc.tcpip. The procedure is as follows.

2.9.3.2 Updating the ODM

Notice that this procedure only updates the ODM data and doesn′t affect the currently running inetd. Also, if you need the debug mode immediately, you have to restart the inetd as previously described.

1. Use the chssys command to update the ODM data:

```
# chssys -s inetd -a ″-d″
0513-077 Subsystem has been changed.
#
```

2. Confirm that the ODM was correctly updated. The object descriptor cmdargs is the command argument passed to the inetd whenever the inetd is invoked:

```
# odmget -q ′subsysname=inetd′ SRCsubsys | grep cmdargs
        cmdargs = ″-d″
#
```

3. If you would check the currently running inetd, you would find that the debug mode is not set. You need to restart the inetd if you want to reflect the ODM update:

```
# lssrc -ls inetd | grep Debug
Debug         Not active
#
```

Note: You feel this is the way the SRC and the ODM are designed. There is a drawback: when the SRC (srcmstr) cannot start for some reason, the debug mode is not activated.

2.9.3.3 Updating the Script /etc/rc.tcpip

This is the alternative procedure and may be the easiest way. You only need to update a startup script and the debug mode is automatically set when the system boots. Update the startup script /etc/rc.tcpip as follows:

```
...
# Start up socket-based daemons
start /usr/sbin/inetd ″$src_running″ ″ -d ″
...
```

This is what happens when you invoke the SMIT byissuing smitty chinetd and specify Yes to the entry Start the inetd subsystem with DEBUGGING on? field. Notice that the SMIT updates the ASCII file and doesn′t update the ODM.

Note: Updating the script /etc/rc.tcpip doesn′t activate the debug mode immediately. You have to restart the inetd using /etc/rc.tcpip. On the contrary, smitty chinetd does both, updating the /etc/rc.tcpip file and setting the debug mode immediately.

2.9.3.4 Log Example

The following is the example of the inetd syslog output. As you can see, even with the level debug, you will get little information. In this example, we telneted to the system mat and issued ls commands, then exited from the newton by pressing Ctrl+D.

```
...
Aug 17 20:22:41 mat root: I will debug inetd!
Aug 17 20:29:59 mat inetd[5446]: someone wants telnet
Aug 17 20:29:59 mat inetd[5446]: accept, ctrl 37
Aug 17 20:31:26 mat inetd[5446]: 10092 reaped
```

Difference between V4.1 and V3.2

With V3.2, you could see a different message. In the example below, we did the same thing:

```
Jul 13 18:56:09 newton inetd[5379]: A connection requires telnet service.
Jul 13 18:56:09 newton inetd[5379]: The accept system call returned file descriptor 31.
Jul 13 18:56:09 newton inetd[16402]: Process number 16402. Server /etc/telnetd.
Jul 13 18:57:20 newton inetd[5379]: Child process 16402 has ended.
```

You can determine whether the inetd successfully forked the subserver daemon telnetd. Again, you cannot know what the user did after the login.

2.9.4 ftpd Example

The daemon ftpd provides the server function of the File Transfer Protocol (FTP). This daemon is started and terminated by the Internet super server daemon, inetd. As a result, it is controlled by the srcmstr indirectly. This daemon is invoked when it is needed (accessed); this mechanism reduces the system overhead.

2.9.4.1 Configuration Procedure

The FTP daemon ftpd has two options to use syslog. One is debug and the other is log. The setup procedure is as follows:

1. Edit the inetd configuration file /etc/inetd.conf and add the necessary flag (-d for debug and -l for log). Of course, you can specify both simultaneously. See the following example:

   ```
   ftp     stream  tcp     nowait  root    /usr/sbin/ftpd         ftpd -d
   ```

2. Since the ftpd is invoked by the inetd, you have to refresh the inetd in order to load the updated /etc/inetd.conf file. The interesting thing to remember is that the inetd doesn′t read the /etc/inetd.conf file directly when it is invoked

by the SRC. The ODM keeps the copy of the /etc/inetd.conf file and the SRC refers to the ODM data and passes it to the inetd. Then you have to update the ODM. There is the inetimp command which imports the /etc/inetd.conf into the ODM. Issue this command without any arguments, as follows:

```
# inetimp
#
```

You can confirm whether the update was really imported to the ODM. In the ODM, the information in the /etc/inetd.conf file is stored in the InetServ object class. Each column of the /etc/inetd.conf file has the corresponding object descriptor and you can easily understand which object descriptor means what. You can review the ODM using the odmget command, as follows:

```
# odmget -q 'servname = ftp' InetServ

InetServ:
        state = 3
        servname = "ftp"
        socktype = "stream"
        protocol = "tcp"
        waitstate = "nowait"
        user = "root"
        path = "/usr/sbin/ftpd"
        command = "ftpd -d"
        portnum = 21
        alias = ""
        description = ""
        statusmethod = "stinet"
#
```

Difference between V4.1 and V3.2

With AIX V4.1, the inetd no longer uses the ODM. Some inetd- and ODM-related commands, such as inetimp and inetserv, are obsolete. Only the ASCII file /etc/inetd.conf is necessary. The object class InetServ was deleted from V4.1. For compatibility purposes, you can configure V4.1 inetd as well as the V3.2 inetd. For such a case, you must install the fileset bos.compat.net. This fileset provides the InteServ object class. This section is written for V3.2 and V4.1, in which this fileset is installed.

3. Now you have to refresh the inetd to load the ODM data as follows:

```
# refresh -s inetd
0513-095 The request for subsystem refresh was completed successfully.
#
```

4. You can see whether the debug mode is really effective by issuing the following:

```
#  lssrc -t ftp
Service       Command                  Arguments                Status







 ftp         /usr/sbin/ftpd           ftpd -d                  active






#
```

Note: The previous example may be the wrong example and could be fixed in the future. With V3.2, we could get following output with the same command:

```
# lssrc -t ftp
Service      Command                  Description              Status
 ftp         ftpd -d                                           active

Debug        active
Logging      inactive
Tracing      inactive
#
```

Remember that the configuration information of the daemon ftpd is stored in both the ODM and the ASCII file. Since the ftpd is a SRC subserver, in order to give the information to the SRC, the ODM is needed. Also, the ftpd is an inetd subserver, and in the case of an SRC failure, the inetd has to read the /etc/inetd.conf file directly. Keep both databases consistent to prevent unnecessary confusion.

2.9.4.2 Log Example: Log Mode

We run ftp from the system zero to the system mat. This log was received from mat. After establishing the FTP session, a small file .profile was transferred. The information you can get with the log mode is as follows:

```
Aug 17 20:54:29 mat inetd[5446]: someone wants ftp
Aug 17 20:54:29 mat inetd[5446]: accept, ctrl 37
Aug 17 20:54:29 mat ftpd[6090]: connection from zero.hakozaki.ibm.com at Thu Aug 17 20:54:29 1995
Aug 17 20:54:35 mat ftpd[6090]: FTP LOGIN FROM zero.hakozaki.ibm.com, matoba
Aug 17 20:54:48 mat ftpd[6090]: FTPD: EXPORT file local , remote .profile
Aug 17 20:54:50 mat inetd[5446]: 6090 reaped
```

2.9.4.3 Log Example: Debug Mode

We run ftp from the system zero to the system mat. This log was received from mat. After establishing the FTP session, a small file .profile was transferred. The information you can get with the debug mode is as follows:

```
Aug 17 20:45:31 mat inetd[5446]: someone wants ftp
Aug 17 20:45:31 mat inetd[5446]: accept, ctrl 37
Aug 17 20:45:31 mat ftpd[10914]: <--- 220
Aug 17 20:45:31 mat ftpd[10914]: mat FTP server (Version 4.1 Sat Aug 27 17:18:21 CDT 1994) ready.
Aug 17 20:45:33 mat ftpd[10914]: command: USER matoba[M
Aug 17 20:45:33 mat ftpd[10914]: <--- 331
Aug 17 20:45:33 mat ftpd[10914]: Password required for matoba.
Aug 17 20:45:36 mat ftpd[10914]: command: PASS takotako[M
Aug 17 20:45:36 mat ftpd[10914]: <--- 230
Aug 17 20:45:36 mat ftpd[10914]: User matoba logged in.
Aug 17 20:45:38 mat ftpd[10914]: command: TYPE I[M
Aug 17 20:45:38 mat ftpd[10914]: <--- 200
Aug 17 20:45:38 mat ftpd[10914]: Type set to I.
Aug 17 20:45:45 mat ftpd[10914]: command: PORT 9,68,214,84,4,94[M
Aug 17 20:45:45 mat ftpd[10914]: <--- 200
Aug 17 20:45:45 mat ftpd[10914]: PORT command successful.
Aug 17 20:45:45 mat ftpd[10914]: command: RETR .profile[M
Aug 17 20:45:45 mat ftpd[10914]: <--- 150
Aug 17 20:45:45 mat ftpd[10914]: Opening data connection for .profile (254 bytes).
Aug 17 20:45:45 mat ftpd[10914]: <--- 226
Aug 17 20:45:45 mat ftpd[10914]: Transfer complete.
Aug 17 20:45:46 mat ftpd[10914]: command: QUIT[M
Aug 17 20:45:46 mat ftpd[10914]: <--- 221
Aug 17 20:45:46 mat ftpd[10914]: Goodbye.
Aug 17 20:45:46 mat inetd[5446]: 10914 reaped
```

2.10 The SRC Basics

Both the system resource controller (SRC) and the object data manager (ODM) are unique AIX features. It is a bit difficult and tough when a problem is related to the SRC or the ODM (or both). If you don′t have enough knowledge about them, the problem remains a mystery even when it happens to be resolved. In this section we reveal a part of the mystery. ODM is briefly explained in 1.7, “The ODM” on page 65.

2.10.1 The SRC Overview

The SRC provides a set of common commands and subroutines to make it easier for the system manager and programmer to create and control subsystems (daemons). The SRC is useful if you want a common way to start, stop, and collect status information on processes. For example, you can use following set of commands on any subsystem:

startsrc
stopsrc
lssrc
chssys
refresh
traceson
tracesoff

The concept of the SRC may be an expansion of the concept of the inetd. The Internet super server inetd only controls the TCP/IP daemons. The SRC adds one more hierarchy on the top of the inetd and supporting other subsystems (daemons) as well as the inetd. The SRC has a hierarchical structure. The top of the hierarchy begins with the SRC; the srcmstr process is followed by a subsystem group (such as tcpip). A subsystem group may contain more than one subsystem (such as the inetd daemon). Each subsystem can have several subservers (such as the ftp daemon and the finger command).

A subsystem group is a group of any specified subsystems. Grouping subsystems together allows control of several subsystems at one time. A few subsystem group examples are TCP/IP (tcpip), NIS (yp) and NFS (nfs). As you see below, groups provide convenient mechanisms to manage all the related subsystems. You can start, stop and review the status of all subsystems at the same time.

```
# startsrc -g nfs
0513-059 The biod Subsystem has been started. Subsystem PID is 7837.
0513-059 The nfsd Subsystem has been started. Subsystem PID is 8606.
0513-059 The rpc.statd Subsystem has been started. Subsystem PID is 9375.
0513-059 The rpc.lockd Subsystem has been started. Subsystem PID is 8096.
0513-059 The rpc.mountd Subsystem has been started. Subsystem PID is 8353.
#
```

This hierarchy is stored in the ODM with following object classes. In other words, the SRC recognizes and controls each subsystem and subservers based on the ODM information.

SRCsubsys
SRCsubsvr
SRCnotify

The SRC is invoked in /etc/inittab during system boot. You can see the line where /etc/srcmstr is invoked, as follows:

```
...
srcmstr:2:respawn:/usr/sbin/srcmstr # System Resource Controller
...
```

2.10.2 The SRC Subsystems

A subsystem is any program or process or set of programs or processes that is usually capable of operating independently. A subsystem is designed as a unit to provide a designated function.

2.10.2.1 TCP/IP Subsystems

TCP/IP daemons (Subsystems) controlled by the SRC are listed below. These daemons belong to mainly TCP/IP groups, but some belong to their own groups, such as mail.

/etc/syslogd (ras)
/usr/lpd/lpd (spooler)
/etc/routed (tcpip)
/etc/gated (tcpip)
/usr/lib/sendmail (mail)
/usr/etc/portmap (portmap)
/etc/inetd (tcpip)
/etc/named (tcpip)
/etc/timed (tcpip)
/etc/rwhod (tcpip)
/usr/sbin/snmpd (tcpip)

These daemons are started from the startup script /etc/rc.tcpip. Remember this script is executed from the /etc/inittab. If the SRC or srcmstr is not running when the script is executed, they were invoked without the srcmstr just as if they are invoked directly at the command line.

Note: Of course, it depends on the configuration as to whether a subsystem is started. Check the script /etc/rc.tcpip and confirm the comment mark (#) has been removed.

2.10.2.2 NFS and NIS Subsystems

These daemons belong to mainly yp and nfs groups, but some belong to their own group, such as keyserv. NFS and NIS daemons (subsystems) controlled by the SRC are listed as follows:

/usr/etc/ypserv (yp)
/usr/etc/ypbind (yp)
/usr/etc/keyserv (keyserv)
/usr/etc/rpc.ypupdated (yp)
/usr/etc/biod (nfs)
/usr/etc/nfsd (nfs)

/usr/etc/rpc.mountd (nfs)
/usr/etc/rpc.statd (nfs)
/usr/etc/rpc.lockd (nfs)
/usr/etc/rpc.yppasswdd (yp)

These daemons are started from the startup script /etc/rc.nfs. Remember this script is executed from the /etc/inittab. This script doesn′t provide an alternative when the SRC is not running.

Note: Of course it depends on the configuration as to whether a subsystem is started. Check the script /etc/rc.nfs and confirm that the comment mark (#) has been removed.

2.10.2.3 Other Subsystems

You can review all the subsystems supporting the SRC on your system. With the command lssrc -a, you will see following list:

```
# lssrc -a
Subsystem         Group            PID     Status
 sna              sna              4720    active
 gpenamesvc       sna              5498    active
 syslogd          ras              7326    active
 sendmail         mail             12740   active
 portmap          portmap          13526   active
 inetd            tcpip            13792   active
 httpd            tcpip            14732   active
 qdaemon          spooler          13258   active
 writesrv         spooler          14548   active
 infod            infod            10496   active
 hcon             system           15108   active
 biod             nfs              20326   active
 nfsd             nfs              19816   active
 rpc.statd        nfs              19562   active
 rpc.lockd        nfs              20844   active
 rpc.mountd       nfs              23668   active
 lpd              spooler                  inoperative
 gated            tcpip                    inoperative
 named            tcpip                    inoperative
 routed           tcpip                    inoperative
 rwhod            tcpip                    inoperative
 iptrace          tcpip                    inoperative
 timed            tcpip                    inoperative
 snmpd            tcpip                    inoperative
 dtsrc                                     inoperative
 keyserv          keyserv                  inoperative
 ypbind           yp                       inoperative
 ypserv           yp                       inoperative
 ypupdated        yp                       inoperative
 yppasswdd        yp                       inoperative
 llbd             ncs                      inoperative
 glbd             ncs                      inoperative
 nrglbd           ncs                      inoperative
 APPNss_cp_cap    sna                      inoperative
 APPNds_rsc_reg   sna                      inoperative
 APPNds_search    sna                      inoperative
 APPNtrs_db_upd   sna                      inoperative
 APPNcnos_rcv     sna                      inoperative
 MSmsu_handler    sna                      inoperative
 MSsend_tp        sna                      inoperative
 MSreceive_tp     sna                      inoperative
 MSsess_outage    sna                      inoperative
 cycle1           sna                      inoperative
 cycled           sna                      inoperative
 RMTMAND          sna                      inoperative
 APINGD           sna                      inoperative
 AREXECD          sna                      inoperative
 ATELLD           sna                      inoperative
 ANAMED           sna                      inoperative
 AFTPD            sna                      inoperative
 netlsd           netlsd                   inoperative
#
```

Some of them are directly invoked from /etc/inittab. The subsystem qdaemon and writesrv are from this example. Some of them are invoked from the startup script which is executed from /etc/inittab.

2.10.3 The SRC Subservers

A subsystem is a daemon, or server, that is controlled by the SRC. A subserver is a daemon that is controlled by a subsystem. The only TCP/IP subsystem that controls other daemons (subservers) is the inetd daemon. Thus, all TCP/IP subservers are also inetd subservers. See the file /etc/inetd.conf for all the TCP/IP subservers. The categories of subsystem and subserver are mutually exclusive. That is, daemons are not listed as both a subsystem and a subserver. The TCP/IP daemons controlled by the inetd subsystem are as follows:

/etc/uucpd
/etc/ftpd
/etc/telnetd
/etc/rshd
/etc/rlogind
/etc/rexecd
/etc/bootpd
/etc/fingerd
/etc/tftpd
/etc/comsat
/etc/talkd
/usr/etc/rpc.rexd
/usr/etc/rpc.rstatd
/usr/etc/rpc.rusersd
/usr/etc/rpc.rwalld
/usr/etc/rpc.sprayd
/etc/rpc.pcnfsd
/u/netinst/bin/instsrv
echo (internal)
discard (internal)
chargen (internal)
daytime (internal)
time (internal)

Note: The daemons marked internal don′t have a separate entity (program). They are integrated in the inetd in order to avoid the overhead of forking and executing them. Since their tasks are so tiny that we cannot ignore the startup cost.

You can review all the subservers supported on your system with the command lssrc -T, and you can see subservers other than the inetd subserver:

```
# lssrc -T
#sub_type:subsysname:sub_code:
ftp:inetd:21:
uucp:inetd:540:
telnet:inetd:23:
shell:inetd:514:
login:inetd:513:
exec:inetd:512:
finger:inetd:79:
tftp:inetd:69:
ntalk:inetd:517:
echo:inetd:7:
discard:inetd:9:
chargen:inetd:19:
daytime:inetd:13:
time:inetd:37:
comsat:inetd:1512:
bootps:inetd:67:
systat:inetd:11:
netstat:inetd:15:
link_station:sna:2:
gateway:sna:40:
#
```

Notice one daemon is not the inetd subserver, butis the SNA subserver. See the second column of the previous list, separated by a colon. Also, ONC/RPC daemons are not the subservers. /usr/etc/rpc.rexd and other RPC daemons are not listed.

The previous list is only a dump list of the SRCsubsvr object class as follows:

```
# odmget SRCsubsvr | grep type
        sub_type = "ftp"
        sub_type = "uucp"
        sub_type = "telnet"
        sub_type = "shell"
        sub_type = "login"
        sub_type = "exec"
        sub_type = "finger"
        sub_type = "tftp"
        sub_type = "ntalk"
        sub_type = "echo"
        sub_type = "discard"
        sub_type = "chargen"
        sub_type = "daytime"
        sub_type = "time"
        sub_type = "comsat"
        sub_type = "bootps"
        sub_type = "systat"
        sub_type = "netstat"
        sub_type = "link_station"
        sub_type = "gateway
#
```

You can retrieve the status of each subserver such as telnetd with the lssrc command, as follows:

```
# lssrc -t telnet
Service      Command                Arguments              Status







 telnet      /usr/sbin/telnetd      telnetd -s             active







#
```

Note: The above result may be wrong and would be fixed in the future. With AIX V3.2, we could get the following result:

```
# lssrc -t telnet
Service     Command                Description           Status
 telnet     telnetd -s                                   active

Pid         Inet Address           Hostname
 12561      9.170.5.45             newton
 terminal   type = TERM=aixterm
#
```

2.10.4 The SRC-Related Object Class in the ODM

The SRC defines and manages three object classes: SRC subsystem object class, SRC subserver type object class and SRC notify object class. Together, these object classes represent the domain in which the SRC performs its functions. A predefined set of object-class descriptors comprise the possible set of subsystem configurations supported by the SRC.

Note: Only the SRC subsystem object class is mandatory. Use of the subserver type and notify object classes is subsystem-dependent.

The SRC uses or relates to the following object classes:

SRCsubsys The subsystem object class contains the descriptors for all SRC subsystems. A subsystem must be configured in this class before it can be recognized by the SRC.

SRCsubsvr An object must be configured in this class if a subsystem has subservers and the subsystem expects to receive subserver-related commands from the srcmstr daemon.

SRCnotify This class provides a mechanism for the srcmstr daemon to invoke subsystem-provided routines when the failure of a subsystem is detected. When the SRC daemon receives a SIGCHLD signal indicating the termination of a subsystem process, it checks the status of the subsystem (maintained by the srcmstr daemon) to determine whether the termination was caused by a stopsrc command. If no stopsrc command was issued, the termination is interpreted as an abnormal termination. If the restart action in the definition does not specify respawn (or if respawn attempts fail), the srcmstr daemon attempts to read an object associated with the subsystem name from the notify object class. If such an object is found, the method associated with the subsystem is run. If no subsystem object is found in the notify object class, the srcmstr daemon determines whether the subsystem belongs to a group. If so, the srcmstr daemon attempts to read an object of that group name from the notify object class. If such an object is found, the method associated with it is invoked. In this way, groups of subsystems can share a common method.

2.10.4.1 SRCsubsys Object Class Example

We show an example of the SRCsubsys object class. When the SRC, srcmstr, invokes a subsystem (daemon), it refers to the information stored in this object class. Be aware that you still can specify some of the parameters at the time of a subsystem invocation and can override the parameters in the ODM. The following is the example of the syslogd subsystem (daemon) data stored in the ODM SRCsubsys object class:

```
# odmget -q "subsysname='syslogd'" SRCsubsys

SRCsubsys:
        subsysname = "syslogd"
        synonym = ""
        cmdargs = ""
        path = "/usr/sbin/syslogd"
        uid = 0
        auditid = 0
        standin = "/dev/console"
        standout = "/dev/console"
        standerr = "/dev/console"
        action = 2
        multi = 0
        contact = 3
        svrkey = 0
        svrmtype = 0
        priority = 20
        signorm = 0
        sigforce = 0
        display = 1
        waittime = 20
        grpname = "ras"
#
```

For example, if the option flag -d is not specified in the object descriptor cmdargs, you can add this flag when the startsrc command is executed, as follows:

```
# startsrc -s syslogd -a "-d"
0513-059 The syslogd Subsystem has been started. Subsystem PID is 10412
#
```

We briefly explain each object descriptor. You can get the information about data type and maximum length (including the null terminator) of each object descriptor by issuing the odmshow command, as follows:

```
# odmshow SRCsubsys
class SRCsubsys {
        char subsysname[30];                        /* offset: 0xc ( 12) */
        char synonym[30];                           /* offset: 0x2a ( 42) */
        char cmdargs[200];                          /* offset: 0x48 ( 72) */
        char path[200];                             /* offset: 0x110 ( 272) */
        long uid;                                   /* offset: 0x1d8 ( 472) */
        long auditid;                               /* offset: 0x1dc ( 476) */
        char standin[200];                          /* offset: 0x1e0 ( 480) */
        char standout[200];                         /* offset: 0x2a8 ( 680) */
        char standerr[200];                         /* offset: 0x370 ( 880) */
        short action;                               /* offset: 0x438 ( 1080) */
        short multi;                                /* offset: 0x43a ( 1082) */
        short contact;                              /* offset: 0x43c ( 1084) */
        long svrkey;                                /* offset: 0x440 ( 1088) */
        long svrmtype;                              /* offset: 0x444 ( 1092) */
        short priority;                             /* offset: 0x448 ( 1096) */
        short signorm;                              /* offset: 0x44a ( 1098) */
        short sigforce;                             /* offset: 0x44c ( 1100) */
        short display;                              /* offset: 0x44e ( 1102) */
        short waittime;                             /* offset: 0x450 ( 1104) */
        char grpname[30];                           /* offset: 0x452 ( 1106) */
        };
/*
        columns:        20
        structsize:     0x470 (1136) bytes
        data offset:    0x514
        population:     51 objects (51 active, 0 deleted)
*/


#
```

You can also refer to the header file /usr/include/spc.h for the symbolic constant such as SRCYES.

subsysname Specifies the name of the subsystem object. This descriptor must be POSIX-compliant. This field is required.

synonym Specifies a character string to be used as an alternate name for the subsystem. This field is optional.

cmdargs Specifies any arguments that must be passed to the command that starts the subsystem. The arguments are parsed by the srcmstr daemon according to the same rules used by shells. For example, quoted strings are passed as a single argument and blanks outside quoted strings delimit arguments.

path Specifies the full path name for the program executed by the subsystem start command. The path name must be POSIX-compliant. This field is required.

uid Specifies the user ID (numeric) under which the subsystem is run. A value of 0 indicates the root user.

auditid Specifies the subsystem audit id. Created automatically by the srcmstr daemon when a subsystem is defined, this field is used by the security system, if configured. This field cannot be set or changed by a program.

standin	Specifies the file or device from which the subsystem receives its input. The default is /dev/console. This field is ignored if the communication type is sockets.
standout	Specifies the file or device to which the subsystem sends its output. The default is /dev/console.
standerr	Specifies the file or device to which the subsystem writes its error messages. Failures are handled as part of the notify method. The default is /dev/console. **Note:** Catastrophic errors are sent to the error log.
action	Specifies whether the srcmstr daemon should restart the subsystem after an abnormal end. A value of RESPAWN(1) specifies that the srcmstr daemon should restart the subsystem. A value of ONCE(2) specifies the srcmstr daemon should not attempt to restart the failed system. There is a respawn limit of two restarts within a specified wait time. If the failed subsystem cannot be successfully restarted, the notification method option is consulted. The default value is ONCE.
multi	Specifies the number of instances of a subsystem that can run at one time. A value of SRCNO(0) specifies that only one instance of the subsystem can run at one time. Attempts to start this subsystem if it is already running will fail (as will attempts to start a subsystem on the same IPC message queue key). A value of SRCYES(1) specifies that multiple subsystems may use the same IPC message queue and that there can be multiple instances of the same subsystem. The default value is SRCNO.
contact	Contact type. The value of this field indicates either a signal SRCSIGNAL(2), a message queue SRCIPC(1), or a socket SRCSOCKET(3).
srvkey	Specifies a decimal value that corresponds to the IPC message queue key that the srcmstr daemon uses to communicate to the subsystem. This field is required for subsystems that communicate using IPC message queues. Use the ftok. subroutine with a fully qualified path name and an ID parameter to ensure that this key is unique. The srcmstr daemon creates the message queue prior to starting the subsystem.
svrmtype	Specifies the mtype of the message that is placed on the subsystem′s message queue. The subsystem uses this value to retrieve messages by using the msgrcv or msgxrcv subroutine. This field is required if you are using message queues.
priority	Nice value, a number from 1 to 40. Specifies the process priority of the subsystem to be run. Subsystems started by the srcmstr daemon run with this priority. The default value is 20.
signorm	Specifies the value to be sent to the subsystem when a stop normal request is sent. This field is required of subsystems using the signals communication type (contact).
sigforce	Specifies the value to be sent to the subsystem when a stop force request is sent. This field is required of subsystems using the signals communication type (contact).

display Indicates whether the status of an inoperative subsystem can be displayed on lssrc -a or lssrc -g output. The default is to display. The value of this field can be either SRCYES(1) or SRCNO(0).

waittime Specifies the time in seconds that a subsystem has to complete a restart or stop request before alternate action is taken. The default is 20 seconds. Stop cancel time to wait before sending a SIGKILL signal to the subsystem restart time period. (A subsystem can be restarted only twice in this time period if it does not terminate normally.)

grpname Designates the subsystem as a member of a group. This field is optional.

Although we don′t explain in this book, you can add your own subsystem in this object class using the mkssys command, and you can delete a subsystem using the rmssys command.

2.10.4.2 SRCsubsvr Object Class Example

We explain the SRCsubsvr object class shortly. See the following example. Since this SRC subserver telnetd is the inetd subserver, it is also registered in the InetServ object class.

```
# odmget -q ″sub_type=′telnet′″ SRCsubsvr

SRCsubsvr:
        sub_type = ″telnet″
        subsysname = ″inetd″
        sub_code = 23
#
```

The definitions of each object descriptor, data type and maximum length (including the null terminator), are listed with the odmshow command as follows:

```
# odmshow SRCsubsvr
class SRCsubsvr {
        char sub_type[30];                          /* offset: 0xc ( 12) */
        char subsysname[30];                        /* offset: 0x2a ( 42) */
        short sub_code;                             /* offset: 0x48 ( 72) */
        };
/*
        columns:        3
        structsize:     0x4c (76) bytes
        data offset:    0x1e8
        population:     20 objects (20 active, 0 deleted)
*/


#
```

Each object descriptor is explained as follows:

sub_type Specifies the name of the subserver type object identifier. The set of subserver type names defines the allowable values for the -t flag of the subserver commands.

subsysname Specifies the name of the subsystem that owns the subserver object. This field is defined as a link to the SRC subsystem object class.

sub_code Specifies a decimal number that identifies the subserver. The code point is passed to the subsystem controlling the subserver in the object field of the subreq structure of the SRC request structure. If a subserver object name is also provided in the command, the srcmstr daemon forwards the code point to the subsystem in the objname field of the subreq structure.

2.10.4.3 SRCnotify Object Class Example

We explain the SRCnotify object class shortly. Currently the httpd is the only one object in this class (if it is installed).

Note: With AIX V3.2, inetd was the only object in this class.

The following is the only example of this object class:

```
# odmget SRCnotify

SRCnotify:
        notifyname = "httpd"
        notifymethod = "/etc/rc.httpd"
#
```

The definitions of each object descriptor, data type and maximum length (including the null terminator), are listed with the odmshow command as follows:

```
# odmshow SRCnotify
class SRCnotify {
        char notifyname[30];                          /* offset: 0xc ( 12) */
        method notifymethod[256];                     /* offset: 0x2a ( 42) */
        };
/*
        columns:        2
        structsize:     0x12c (300) bytes
        data offset:    0x1bc
        population:     1 objects (1 active, 0 deleted)
*/


#
```

Each object descriptor is explained below:

notifyname This specifies the name of the subsystem or group for which a notify method is defined.

notifymethod This specifies the full path name to the routine that is executed when the srcmstr daemon detects abnormal termination of the subsystem or group.

2.10.5 The inetd-Related Object Class InetServ

As we mentioned, the inetd uses its own object class InetServ. When the inetd is invoked by the SRC srcmstr, this object class is referred to. When the inetd is invoked by the command line or in the case of SRC failure, the configuration file /etc/inetd.conf is is referred to instead. In order to avoid any problem caused by an inconsistency between the ODM and the configuration file, you should issue the inetimp command whenever you update /etc/inetd.conf. This command updates the InetServ object class using the /etc/inetd.conf file.

Note: There is a command inetserv and it allows you to update or modify the InetServ object class directly. With this command, you can export the content of the InetServ object class into the /etc/inetd.conf file. We know few people actually do this.

With AIX V4.1, the inetd no longer use the ODM.

Some inetd- and ODM-related commands, such as inetimp and inetserv, are obsolete. Only the ASCII file /etc/inetd.conf is necessary. The object class InetServ was deleted from V4.1. For compatibility purposes, you can configure V4.1 inetd as well as the V3.2 inetd. For such a case, you must install the fileset bos.compat.net. This fileset provides the InteServ object class. This section is written for V3.2 and V4.1, in which this fileset is installed.

```
# odmget -q ″servname=′telnet′″ InetServ

InetServ:
        state = 1
        servname = ″telnet″
        socktype = ″stream″
        protocol = ″tcp″
        waitstate = ″nowait″
        user = ″root″
        path = ″/usr/sbin/telnetd″
        command = ″telnetd″
        portnum = 23
        alias = ″″
        description = ″″
        statusmethod = ″stinet″
#
```

Although we don′t explain each object descriptor, if you compare it with /etc/inetd.conf, you can understand their meanings easily.

```
# grep telnet /etc/inetd.conf
#telnet  stream  tcp     nowait  root    /usr/sbin/telnetd      telnetd
#
```

The definitions of each object descriptor, data type and maximum length (including the null terminator), are listed with the odmshow command as follows:

```
# odmshow InetServ
class InetServ {
        long state;                               /* offset: 0xc ( 12) */
        char servname[20];                        /* offset: 0x10 ( 16) */
        char socktype[20];                        /* offset: 0x24 ( 36) */
        char protocol[10];                        /* offset: 0x38 ( 56) */
        char waitstate[10];                       /* offset: 0x42 ( 66) */
        char user[20];                            /* offset: 0x4c ( 76) */
        char path[50];                            /* offset: 0x60 ( 96) */
        char command[50];                         /* offset: 0x92 ( 146) */
        long portnum;                             /* offset: 0xc4 ( 196) */
        char alias[50];                           /* offset: 0xc8 ( 200) */
        char description[40];                     /* offset: 0xfa ( 250) */
        method statusmethod[256];                 /* offset: 0x122 ( 290) */
        };
/*
        columns:        12
        structsize:     0x224 (548) bytes
        data offset:    0x398
        population:     664 objects (664 active, 0 deleted)
*/


#
```

2.10.6 Debugging inetd Example

In the case that you have made a mistake during the inetd customization, you might have left an inconsistency in the ODM and you will have a tough problem. If the inconsistency is in the kernel (memory), rebooting the system will fix it. But the inconsistency in the ODM remains until you correct the ODM. The ODM is not a flat ASCII file and it has its unique interfaces (commands). It stores data in binary form and has its own data structure. Therefore, those who only know about standard UNIX tools or commands cannot fix the problem. We explain a problem which we actually faced during the development of this book.

2.10.6.1 The Symptom

One day, you find that nobody could log in to the system zero via the TCP/IP network.

```
# tn zero
Trying...
telnet: connect: Connection refused
#
```

2.10.6.2 The Problem Determination Procedure

The problem was not only the TELNET but also FTP and other network services. Even the daytime server was not available. As you already know, this error message is caused by the TCP ACK segment with RST flag, and means that the application is not started. Refer to the following result (remember port 13 is daytime server):

```
ftp zero
ftp: connect: Connection refused
ftp> quit
# tn zero 13
Trying...
telnet: connect: Connection refused
#
```

1. Since all of these applications are invoked by the inetd, these symptoms suggest that the inetd has a problem. Then you check the status on the system zero. When you check the status with lssrc -s, it seems to be working:

   ```
   # lssrc -s inetd
   Subsystem         Group            PID     Status
    inetd            tcpip            27118   active
   #
   ```

2. Then you try to see the status of each daemons (subservers) controlled by the inetd, with lssrc -ls. You have a problem, as follows:

   ```
   # lssrc -ls inetd
   0513-056 Timeout waiting for command response.
   #
   ```

Note: The srcmstr responds to the command lssrc -s. But for the command lssrc -ls, the query request is passed to the subservers (daemon). Then successful -s and failed -ls implies that the inetd couldn′t communicate to the subservers.

3. Next you try to restart the inetd. The first step was to stop the inetd, but the stopsrc command didn′t work:

   ```
   # stopsrc -s inetd
   0513-056 Timeout waiting for command response.
   #
   ```

   Then you use the ps command to get the process id (PID) and kill command to stop the inetd. This time it works. You find it curious that an option (?), d was displayed with /etc/inetd.

   ```
   # ps -ef |grep inet |grep -v grep
       root 27118  3684   0 11:13:38      -  0:00 /usr/sbin/inetd d
   # kill 27118
   # ps -ef |grep inet |grep -v grep
   #
   ```

4. You start the inetd at the command line. In this procedure, you can invoke the inetd without the SRC. Again, it works and you see that all the services controlled by the inetd also work this time.

   Note: Do not forget that this time the inetd read /etc/inetd.conf and doesn′t use the InetServ object class in the ODM.

   ```
   # /etc/inetd
   # ps -ef |grep inet |grep -v grep
       root 26886     1   7 11:20:52      -  0:00 /etc/inetd
   #
   ```

5. The successful result of this procedure implies that the problem was SRC-related. After rebooting the system, the inetd got into the completely same situation. This time, again, the mysterious option d was also back. Since the problem came back after a reboot, this suggests the cause is stored in non-volatile storage such as the ODM. Now you know that the command argument passed to the daemon can be written in the cmdargs object descriptor. Then you check it with the odmget command:

   ```
   # odmget -q ″subsysname=′inetd′″ SRCsubsys | grep cmdargs
           cmdargs = ″d″
   #
   ```

Now you reveal the cause that the data in the ODM is wrong.

2.10.6.3 The Repairing Procedure

Now you find that incorrect argument d is stored instead of -d.

1. This can be corrected with the chssys command:

   ```
   # chssys -s inetd -a ″-d″
   0513-077 Subsystem has been changed.
   #
   ```

   Note: In this example, we were lucky because the chssys command worked well. We cannot always expect this. In the case that the chssys fails, you need to use the odmchange command or the ODM Editor.

2. You should check to see whether the change was correctly made to the ODM:

```
# odmget -q "subsysname='inetd'" SRCsubsys | grep cmdargs
        cmdargs = "-d"
#
```

3. Be aware that the chssys command only updates the ODM and the current running daemon doesn't know the update. You have to stop and restart the daemon. In this case, the inetd was started with the wrong argument and we couldn't stop it with the stopsrc (as you saw). Then, we used the kill command:

```
# ps -ef |grep inetd |grep -v inetd
    root  5412  2684   0 21:59:15      -  0:00 /etc/inetd d
# kill 5412
# startsrc -s inetd
0513-059 The inetd Subsystem has been started. Subsystem PID is 26924.
#
```

4. Now you can get detailed information about the inetd with the lssrc -ls command. This time the inetd responded to the command, as follows:

```
# lssrc -ls inetd
Subsystem        Group            PID     Status
 inetd           tcpip            26924   active

Debug         Active

Signal        Purpose
 SIGALRM      Establishes socket connections for failed services
 SIGHUP       Rereads configuration database and reconfigures services

 SIGCHLD      Restarts service in case the service dies abnormally

Service       Command                Arguments                Status
 login        /usr/sbin/rlogind      rlogind                  active
 telnet       /usr/sbin/telnetd      telnetd                  active
 cmsd         /usr/dt/bin/rpc.cmsd   cmsd 100068 2-4          active
 dtspc        /usr/dt/bin/dtspcd     /usr/dt/bin/dtspcd       active
 time         internal                                        active
 daytime      internal                                        active
 discard      internal                                        active
 echo         internal                                        active
 time         internal                                        active
 daytime      internal                                        active
 chargen      internal                                        active
 discard      internal                                        active
 pcnfsd       /usr/sbin/rpc.pcnfsd   pcnfsd 150001 1-2        active
 sprayd       /usr/lib/netsvc/spray/rpc.sprayd sprayd 100012 1        active
 rwalld       /usr/lib/netsvc/rwall/rpc.rwalld rwalld 100008 1        active
 rusersd      /usr/lib/netsvc/rusers/rpc.rusersd rusersd 100002 1-2      active
 rstatd       /usr/sbin/rpc.rstatd   rstatd 100001 1-3        active
 ntalk        /usr/sbin/talkd        talkd                    active
 shell        /usr/sbin/rshd         rshd                     active
 ftp          /usr/sbin/ftpd         ftpd                     active
#
```

Notice that the previous example is due to the srcmstr capability. If you invoked the inetd from the command line without using the srcmstr, the lssrc command responds with the wrong answer. This does not necessarily mean something is wrong. For example, you can confirm whether the inetd invoked at the command line is running with the ps command:

```
# inetd
# ps -ef | grep inetd | grep -v grep
    root 25158     1   0 11:30:08      -  0:00 inetd
#
```

The lssrc command gives you a totally different result. Although the inetd and its subservers are running perfectly, you wouldn't know it with the lssrc command:

```
# lssrc -s inetd
Subsystem         Group            PID     Status
 inetd            tcpip                    inoperative
# lssrc -ls inetd
0513-036 The request could not be passed to the inetd subsystem.
Start the subsystem and try your command again.
#
```

2.10.7 Other SRC Pitfalls

The SRC (srcmstr) adds some administrative benefits. For example, you can manage the related subsystems (daemons) as a group, and the srcmstr can prevent a daemon carelessly invoked twice. On the contrary, the SRC adds some complexity to the system. The SRC-related problems may be difficult to debug because this is totally an AIX unique feature.

2.10.7.1 When the SRC (srcmstr) Is Killed

The srcmstr is invoked from the /etc/inittab, as follows:

```
srcmstr:2:respawn:/usr/sbin/srcmstr # System Resource Controller
```

It is specified as respawn and this means when the /usr/sbin/srcmstr is killed for some reason, the init automatically restarts the srcmstr. This is quite a convenient and reasonable function, but it has a pitfall. Due to the restart, the srcmstr loses consistency about the subserver's status. See the following example:

1. Check the status of the srcmstr subservers. They are working correctly.

   ```
   # lssrc -s inetd
   Subsystem         Group            PID     Status
    inetd            tcpip            13800   active
   # lssrc -s portmap
   Subsystem         Group            PID     Status
    portmap          portmap          13534   active
   # lssrc -s sendmail
   Subsystem         Group            PID     Status
    sendmail         mail             12748   active
   #
   ```

2. Kill /usr/sbin/srcmstr with kill -9, as follows. Then the /usr/sbin/srcmstr file will be immediately respawned by the init. Then you can know that the PID was updated.

   ```
   # ps -ef | grep srcmstr | grep -v grep
       root  3690     1   0 12:03:19      -  0:00 /usr/sbin/srcmstr
   # kill -9 3690
   # ps -ef | grep srcmstr | grep -v grep
       root  3736     1   1 12:08:54      -  0:00 /usr/sbin/srcmstr
   #
   ```

3. Check the status of the srcmstr subservers again. The lssrc -s commnd says that they are not working.

```
# lssrc -s inetd
Subsystem         Group            PID     Status
 inetd            tcpip                    inoperative
# lssrc -s portmap
Subsystem         Group            PID     Status
 portmap          portmap                  inoperative
# lssrc -s sendmail
Subsystem         Group            PID     Status
 sendmail         mail                     inoperative
#
```

4. Check the status of the srcmstr subservers once more with the ps -ef command. This time those subserver daemons are working. Notice the PIDs are the same as the previous PIDs before the srcmstr was killed (and restarted). This implies that those subserver daemons are intact and have not been stopped.

```
# ps -ef |grep inetd | grep -v grep
    root 13800     1   0 12:03:36      -  0:00 /usr/sbin/inetd -d
# ps -ef |grep portmap | grep -v grep
    root 13534     1   0 12:03:33      -  0:00 /usr/sbin/portmap
# ps -ef |grep sendmail | grep -v grep
    root 12748     1   0 12:03:29      -  0:00 /usr/lib/sendmail -bd -q30m
#
```

5. If you are confused with the lssrc -s command output, you may invoke the subserver daemons with the startsrc command, as follows:

```
# startsrc -s inetd
0513-059 The inetd Subsystem has been started. Subsystem PID is 22722.
#
```

   It works fine, but now you have two instances of the daemon running. The ps -ef command tells the truth.

```
# ps -ef |grep inetd | grep -v grep
    root 13800     1   0 12:03:36      -  0:00 /usr/sbin/inetd -d
    root 22722  3736   0 12:10:56      -  0:00 /usr/sbin/inetd -d
#
```

   In our experience, the easiest and most certain recovery procedure of the srcmstr inconsistency is to reboot the system. Since the subservers are running well, you can use the communication functions without any problems.

   Another visible symptom may be the system error log. If you noticed something wrong about the srcmstr, reviewing the system error log (following) is a good idea:

```
# errpt
IDENTIFIER TIMESTAMP  T C RESOURCE_NAME  DESCRIPTION
E18E984F   0818121295 P S SRC            SOFTWARE PROGRAM ERROR
2BFA76F6   0818120095 T S SYSPROC        SYSTEM SHUTDOWN BY USER
9DBCFDEE   0818120395 T O errdemon       ERROR LOGGING TURNED ON
192AC071   0818115895 T O errdemon       ERROR LOGGING TURNED OFF
E18E984F   0818115895 P S SRC            SOFTWARE PROGRAM ERROR
E18E984F   0818115895 P S SRC            SOFTWARE PROGRAM ERROR
E18E984F   0818115895 P S SRC            SOFTWARE PROGRAM ERROR
E18E984F   0818115895 P S SRC            SOFTWARE PROGRAM ERROR
E18E984F   0818115895 P S SRC            SOFTWARE PROGRAM ERROR
E18E984F   0818115895 P S SRC            SOFTWARE PROGRAM ERROR
...
```

Difference between V4.1 and V3.2

With V3.2 there was a severe pitfall. The srcmstr restarts automatically when it gets killed, but all the subservers (daemons) controlled by the srcmstr are also killed and they aren′t restarted automatically. They die when the srcmstr dies. As a result, you would get the following problems:

```
# tn newton
Trying...
telnet: connect: A remote host refused an attempted connect operation.
tn> quit
#
```

We performed several experiments and can conclude that the most certain recovery procedure is to reboot the system. So far, even when the srcmstr looks fine, if it has been killed once, you will have some tough problems.

As you can see, there must be some inconsistencies in the system. We made several experiments and can conclude that the most certain recovery procedure is to *reboot* the system. So far, even if the srcmstr looks fine, if it has been killed once, you will have some tough problems.

2.10.7.2 SRC Uses UNIX Domain Socket

As you can see in the following example, the srcmstr uses some UNIX domain sockets. Those sockets are placed in the /dev directory. Anything bad for a socket impacts the srcmstr functionality. Filling up the root directory or removing these domain sockets damages the srcmstr. Of course, such problems should not happen on a well-administered system.

```
# netstat -a -f unix
Active UNIX domain sockets
SADR/PCB  Type   Recv-Q Send-Q Inode    Conn     Refs     Nextref  Addr
 5a20100 dgram       0      0  7a0de34        0        0        0 /dev/SRC
 5a41a80
 5a20900 stream      0      0  7a00c5c        0        0        0 /tmp/.X11-unix/X0
 5a41b00
 5a14c00 dgram       0      0  7a12d34        0        0        0 /dev/.SRC-unix/SRCaSdCEa
 5a41a00
 5a14d00 dgram       0      0  7a13224        0  5a41400        0 /dev/log
 5a41a40
 5a14a00 dgram       0      0  7a14fc4        0        0        0 /dev/.SRC-unix/SRCgtdCEb
 5a41980
 5a14500 dgram       0      0        0  5a41a40        0        0
 5a418c0
 5a11e00 dgram       0      0  7a16384        0        0        0 /dev/.SRC-unix/SRCrwdCEc
 5a41840
 5a14300 dgram       0      0        0  5a41a40        0  5a418c0
 5a41880
 5a4fe00 dgram       0      0  7a18b04        0        0        0 /dev/.SRC-unix/SRCRAdCEd
 5a417c0
 5a4fc00 stream      0      0  7a1f52c        0        0        0 /tmp/.info-help
 5a41800
 5a20300 dgram       0      0        0  5a41a40        0  5a41880
 5a41ac0
 5a4f600 stream      0      0  7a2b970        0        0        0 /tmp/.X11-unix/XIM
 5a41780
 5a56200 dgram       0      0  7a48854        0        0        0 /dev/.SRC-unix/SRCbudF7a
 5a413c0
 5a56000 dgram       0      0        0  5a41a40        0  5a41ac0
 5a41400
#
```

2.10.7.3 SRC-Related System Error Log

As you know, the SRC-related errors may be logged by the system. Unfortunately, there are no detailed documentations that describe the meanings of each field in the error log report. An example of an error log is as follows:

```
# errpt -a -j E18E984F
---------------------------------------------------------------------------
LABEL:          SRC
IDENTIFIER:     E18E984F

Date/Time:       Fri Aug 18 11:58:15
Sequence Number: 1712
Machine Id:      000970044D00
Node Id:         zero
Class:           S
Type:            PERM
Resource Name:   SRC

Description
SOFTWARE PROGRAM ERROR

Probable Causes
APPLICATION PROGRAM

Failure Causes
SOFTWARE PROGRAM

        Recommended Actions
        PERFORM PROBLEM RECOVERY PROCEDURES

Detail Data
SYMPTOM CODE
           0
SOFTWARE ERROR CODE
       -9020
ERROR CODE
         256
DETECTING MODULE
'srchevn.c'@line:'265'
FAILING MODULE
writesrv
```

The only thing you can refer to is the header file /usr/include/srcerrno.h. In this file you can find a short explanation about SOFTWARE ERROR CODE.

```
# cat /usr/include/srcerrno.h
...
#define SRC_SVKO -9017  /* Subsystem ended */
#define SRC_INVALID_USER_ROOT -9018 /* not root */
#define SRC_INVALID_USER -9019  /* not root or system */
#define SRC_TRYX -9020  /* Retry limit exceeded restarting subsystem */
#define SRC_RES  -9021  /* Resource not found */
#define SRC_CURR -9022  /* Resource currently under command processing */
#define SRC_PROC -9023  /* Command already processed */
...
```

In conclusion, the error log example can be understood that a SRC subsystem portmap has a problem. The portmap subsystem was restarted twice, but since multiple instances of the portmap is not allowed, the system got the error. You can confirm how many instances are allowed for the portmap as below, looking at the configuration in the ODM.

```
# odmget -q ″subsysname=′writesrv′″ SRCsubsys | grep multi
        multi = 0
#
```

The “0” means only one instance is allowed, but we still don′t know why the portmap was started twice.

2.10.7.4 Meaning of Debug Mode (traceson, tracesoff)

One advantage of using the SRC is that you can use the common commands to manage the subsystems (daemons). Examples are the lssrc, startsrc, stopsrc, refresh and chssys commands. About these commands, you can expect that they have the same effects to any of their subsystems. But when you use traceson and tracesoff commands, you have to pay attention. For some daemons, these commands enable and disable socket-level trace. For other daemons, these commands enable and disable the SYSLOG trace. It′s totally subsystem-dependent. When you use these commands, you have to refer to the manual or InfoExplorer first and understand which trace is invoked. The following is a summary of some daemons:

inetd	SYSLOG trace
telnet	Socketr-level trace
telnetd	Socket-level trace
ftp	SYSLOG trace
ftpd	SYSLOG trace

You can invoke this trace function with SMIT. It′s not only the trace function, but other SRC functions also have SMIT support. If you use the trace function via SMIT, follow these steps:

1. Invoke the SMIT with the following fast path:

   ```
   # smitty src
   ```

2. You will see the following menu:

```
                          Processes & Subsystems

Move cursor to desired item and press Enter.

  Processes
  Subsystems
  Subservers

F1=Help            F2=Refresh         F3=Cancel          F8=Image
F9=Shell           F10=Exit           Enter=Do
```

Figure 25. SMIT Processes & Subsystems Selection Screen

3. If you choose Subsystems at the previous screen, you will see the following menu for the subsystems:

```
                                Subsystems

Move cursor to desired item and press Enter.

  List All Subsystems
  Query a Subsystem
  Start a Subsystem
  Stop Subsystem
  Refresh a Subsystem
  Trace Subsystem

F1=Help            F2=Refresh         F3=Cancel          F8=Image
F9=Shell           F10=Exit           Enter=Do
```

Figure 26. SMIT Subsystems Menu Screen

4. If you use the trace function, choose Trace Subsystem at the previous menu. You can start or stop the trace function in this screen.

```
                             Trace Subsystem

Move cursor to desired item and press Enter.

  Start Trace
  Stop Trace

F1=Help            F2=Refresh         F3=Cancel          F8=Image
F9=Shell           F10=Exit           Enter=Do
```

Figure 27. SMIT Trace Subsystem Screen

2.11 IP Trace

Now it′s a good time to explain the iptrace and ipreport functions. In this book, we included many IP trace sample outputs. It′s a very powerful and convenient tool to analyze what packets are actually sent and received. In order to understand (interpret) a trace result, you must have complete knowledge of TCP/IP. Explaining TCP/IP in detail is out of the scope of this book. It would take several volumes of books and there are a lot of good books available at book stores. Of course, RFCs are very good references. So far in this section, we mainly focus on how to get IP traces.

Difference between V4.1 and V3.2

With AIX V4.1, you need explicit installation of the fileset bos.net.tcp.server to use IP trace. Not only is the IP trace included, but also some other useful commands are included in this fileset. trpt and tcdump are the examples.

2.11.1 IP Trace Basics

The following contains some points you should know about the IP trace.

The most difficult part is interpretation of the result.
Honestly speaking, getting a trace (not only regarding IP trace) is not the goal. Usually a trace gives you almost everything and you have to pick clues from a huge trace log. Of course, you must have a good knowledge of TCP/IP or you will not be able to detect what is wrong. The IP trace only shows you a dump list of data and protocol headers. The investigation task is left for you. If you don′t know the meaning of each field in a protocol header, the IP trace is useless.

You will get a huge data file quickly.
Since the IP trace captures entire packets and stores it on your disk, if you monitor a file transfer session of 10 MB, 10 MB of data and protocol headers are stored on your disk. The IP trace provides some options to hook only necessary data (packets). You must know what packets are necessary to your problem determination. In our experience, whenever we had to get the IP trace, the situation was pretty tough and we hardly had any idea of the hook condition. If you are sure what packets you are looking for, it means you already know the cause of your problem.

It′s very easy to use IP trace.
Getting an IP trace log or capturing packets is very simple. It is only a matter of issuing a few commands.

It cannot capture packets which are not sent to your system.(V3.2 and some V4.1).
The IP trace doesn′t provide network monitoring capability, which is supported by a protocol analyzer. You can capture all the packets sent from your system, and received by your system. You cannot capture any packets that are sent and received between other systems. (Broadcast packets are received by your AIX.) So far, you cannot monitor the entire network traffic with an IP trace.

It can resolve ARP, IP, ICMP, UDP TCP and more.
The IP trace log can be formatted to a human readable style. Each protocol header is clearly formatted and you can easily read the value of each field, but it doesn′t resolve the application layer information. Then, if you are reading an IP trace report of TELNET, you must read the raw data dumped in hex to understand the TELNET protocol. It′s not easy. Currently only ONC/RPC, NFS, DNS and RIP are exceptions. Giving the appropriate flag -r to the ipreport command, these application headers are formatted.

ISA Adapter
If your system has an ISA adapter, the IP trace captures packets that are received. Any packets that are sent out from your system are not captured in the IP trace.

As mentioned previously, the IP trace is not a perfect tool. But it is still extremely useful and this entire book is the real evidence to prove the capability of the IP trace.

Difference between V4.1 and V3.2

With AIX V4.1, a new capability, promiscuous mode, is supported. With the the new flag, -e, you can capture any packets run through the cable. Those packets need not be destined for your system. The promiscuous mode also depends on the adapter card you use. You can not use the promiscuous mode with the old token-ring adapter (even in V4.1).

2.11.2 Getting the IP Trace with the iptrace and ipreport Command

The basic procedure of getting IP trace is shown below. In this example, log files are stored in the /tmp directory. If you need to trace heavy traffic, log can be bigger than several megabytes. The formatted report file is usually several times greater than the log. It is recommended to prepare an appropriate directory or file system.

1. Invoke the iptrace command (daemon). The command in this example has option -b and -d lazy. This means that only the packets between this system and the lazy are captured. The flag -a means that no ARP packets are captured:

   ```
   # iptrace -a -b -d kashima /tmp/test_iptrace.log
   #
   ```

2. Do the operation which you need to analyze. In the following example, it only involves sending one ping packet to the destination:

   ```
   # ping -c 1 kashima
   PING kashima.hakozaki.ibm.com: (9.68.214.76): 56 data bytes
   64 bytes from 9.68.214.76: icmp_seq=0 ttl=255 time=3 ms

   ----kashima.hakozaki.ibm.com PING Statistics----
   1 packets transmitted, 1 packets received, 0% packet loss
   round-trip min/avg/max = 3/3/3 ms
   #
   ```

3. If you complete the operation, kill the iptrace daemon. You can use the kill command as follows:

   ```
   # ps -e | grep iptrace | grep -v grep
    13346      -  0:00 iptrace
   # kill 13346
   #
   ```

   If you need to automate this procedure using the shell script, you may prefer to write the previous commands as follows:

   ```
   # kill $(ps -e | grep iptrace | grep -v grep | awk '{print $1}')
   ```

   Now you have an IP trace log file. All the data in this file is binary and you cannot read it.

   ```
   # ls -la /tmp/test_iptrace.log
   -rw-r--r--   1 root     system       305 Aug 18 14:56 /tmp/test_iptrace.log
   #
   ```

4. You have to format the log file to a report file in order to translate it to readable form. Do not forget the redirection mark >, or you just see the formatted report on your display. You can use the ipreport command. The

flag -r means that the ONC/RPC and NFS headers are also formatted. The flag -n means that the sequential packet number is included in the formatted report. The flag -s means to append the protocol name to every line in a packet. The following is an example of the ipreport command usage:

```
# ipreport -rns /tmp/test_iptrace.log > /tmp/test_iptrace.rpt
```

Now you have an IP report file as follows:

```
# ls -la /tmp/test_iptrace.rpt
-rw-r--r--   1 root     system      1386 Aug 18 14:57 /tmp/test_iptrace.rpt
#
```

5. You can read the IP report file with your favorite editor.

```
# cat /tmp/test_iptrace.rpt
# cat /tmp/test_iptrace.rpt |pg
IPTRACE version: 2.0

Packet Number 1
TOK: ====( 106 bytes transmitted on interface tr0 )==== 14:56:03.033315072
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:06:44:40]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.76 >  (kashima.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=978, ip_off=0
IP:     ip_ttl=255, ip_sum=f8af, ip_p = 1 (ICMP)
ICMP:   icmp_type=8 (ECHO_REQUEST)  icmp_id=10532  icmp_seq=0
...
```

2.11.3 Some Hints and Tips

As usual, experience is the best teacher. You should get accustomed to using the IP trace and report. Sometimes you may fail to get the necessary IP trace data by specifying the inappropriate flags.

2.11.3.1 How to Choose iptrace Hooks (Flags)

It is extremely difficult to resist the fascination of using flags to narrow down the packets to be captured. Without any flags, you will see everything. If your system is a heavily used server machine, you have to pick a few packets up from hundreds of others. The following flags are now supported. Refer to the manual or InfoExplorer for details.

-a	Suppresses ARP packets. If you don′t use this flag, a lot of ARP packets may cause you concern. Because the ARP request is broadcast, all the ARP requests in the same network are captured.
-b	Changes the -d or -s flags to the bidirectional mode.
-d <Host>	Records packets headed for the specified destination host. If used with the -b flag, the -d flag records packets both going to and coming from the specified host.
-i <Interface>	Records packets received on the interface specified by the Interface variable.
-P <Protocol>	Records packets that use the specified protocol.

-p <Port No> Records packets that use the specified port number.

-s Host Records packets coming from the specified source host. If used with the -b flag, the -s flag records packets both going to and coming from the host specified by the Host variable.

In our experience, the combination of -a, -b and -d works fine in many situations. You should always keep in mind that you are not looking at everything as long as you use flag(s). There is a risk of eliminating clues. For example, if you are using DNS and you use a -d flag, any DNS lookup activity is not traced. If your problem is in the DNS lookup procedure, you cannot find anything.

Our Experience

We tried to capture an IP trace to analyze the TCP timeout mechanism. In order to observe retransmission due to no response, we sent the TCP segment to a system that was not turned on. We used the -a, -b, and -d flags. The result was unbelievable because no packets were captured. We repeated the experiment dropping each flag one by one, and we realized what happened when we dropped the -a flag.

Since the destination system was turned off, it could not respond to the ARP request. Due to the ARP failure, no TCP segments (packets) were sent. If we had not eliminated the ARP packets, it would have been very obvious.

2.11.3.2 Pseudo Real-Time IP Trace

The IP trace and report are rather a batch procedure. They don′t provide real-time log analyzing capability. After everything is finished, it is easy to know if the IP trace is successful. If you really need strict real-time capability, use a protocol analyzer. We only explain a pseudo-alternative procedure.

1. Issue the following command to create a pipe:

   ```
   # mknod /tmp/iptrace p
   #
   ```

   Difference between V4.1 and V3.2

   With AIX V4.1, you have a new tool, tcpdump. This command provides real-time packet capturing and displaying capability. If you need real-time packet analysis, we recommend that you use tcpdump, although the output looks tough to read.

2. Invoke the ipreport command at background:

   ```
   # ipreport -rns /tmp/iptrace > /tmp/iptrace_realtime.rpt &
   [1]     13384
   #
   ```

3. Invoke the iptrace. Immediately after this command, the formatted report is continuously generated in the specified file, /tmp/iptrace_realtime.rpt:

   ```
   # iptrace -i tr0 /tmp/iptrace
   #
   ```

4. You can review the content of the report file dynamically with the following command:

   ```
   # tail -f /tmp/iptrace_realtime.rpt
   ```

Chapter 3. Getting Information for Performance Tuning

In the previous chapter we mainly discussed connectivity problems. In this chapter we mainly treat performance problems. When you make connections to your destination, and you find that your network application is slower than usual, there may be a performance problem.

Consider that a problem can have two aspects: connectivity and performance. The same problem can have different symptoms in different environments. So be flexible when you make decisions based on a preconceived notion. Always keep in mind that the slow response is an aspect of the problem and not the whole problem.

3.1 When Something Is Wrong...

Generally speaking, it is more difficult to resolve a performance problem than a connectivity problem. Even when we have a performance problem, we can assume that almost all components are working properly (or you would have lost the connectivity), and this makes the problem more complicated.

3.1.1 How to Approach the Performance Problem

Since there is no methodology to performance tuning and problem determination, you can start at any point that you like. In this section we described our approach. We don′t insist that this is the only way or the best way. We believe you can develop your own methodology, and it must be the best way for you.

3.1.1.1 Gather Information

First we have to gather as much information as possible. This is a very important step to reduce possible causes or reasons for the problem. The information you should gather is listed below:

- Are all functions (applications) suffering slow performance, or do only some functions or applications have the performance problem?
- Are all operations suffering slow performance, or do only some operations have the performance problem? For example, when an end user complains about slow response of TELNET, it is important to know if all operations (commands) during the TELNET session are slow or if only the login procedure is slow.
- Are all systems in the network suffering slow performance, or do only some of the systems have the performance problem?
- When did the problem begin? Has the problem existed since the network configuration or an application installation, or has the system developed this problem since then? (It′s not easy to remember the operation log or history to find what event or activity caused the performance problem.)
- Is the problem intermittent or permanent? If it is intermittent, the debugging procedure is usually difficult. (It depends on the frequency of occurrence.) If it happens once a day, it′s not so bad because it is possible to stay on the site and wait until it happens again. However, if it happens once a week or once a month, you need some automated procedures to gather data.
- Are there any error messages on the console or error log?

Answering the above questions may not be so easy. In our experience, 50% or more of the people who asked for our help were not be able to answer all of the questions. If we get less information, we have to use more hunches and guesses.

3.1.1.2 Understand the Specification

You have to know the network environment with which you are dealing. If your environment includes multi-vendor products, you have to know about each product. Although we presently have many standards, the standards don′t define everything.

One important checkpoint is that all products are used within their specifications. Many products can work even if they are outside of their specifications, but they may get into an unstable state. The following are ad hoc examples and are not a complete check list. There are many items that could be checked, as follows:

- Ethernet 10Base2 specification allows up to 30 stations within one segment. The cable cannot exceed 185 meters. In 10Base2 network, adding a station means increasing cable length. It might be longer than 185 meters.
- Supported maximum packet/frame size of your bridges. If one of your bridges has a smaller maximum packet size than the others, almost all packets can be passed or routed. But some extremely large packets will meet a problem inviting retransmission or some recovery action.
- Some PC′s TCP/IP software don′t comply to the full TCP/IP specification. They may not be able to follow TCP flow control completely. Also they may not have enough memory space or buffer area. For such resource restricted systems, communicating with RS/6000 easily invites packets to drop and the need for successive retransmission.

3.1.1.3 Is That Really a Problem?

Sometimes you have to ask this question. Recent innovations have given us many convenient features or functions and we take them for granted. We should not forget that nothing can be designed for all purposes. We can easily get out from the original design scope. It may be a problem of product usage and may not be a problem of the product itself.

For example, NFS provides very useful functions, such as the transparent access to remote files. This is the original NFS design goal, and it has not been designed for file transfer. But we see so often that people easily move a file of hundreds of megabytes through NFS by issuing the cp or mv command. Of course, they don′t care if there is a router (or routers) on the way and they complain of the system being too slow. For a large file transfer, especially in an unreliable network environment, you should use FTP and should not use NFS. Current NFS in AIX only uses UDP, and retransmission is made at application (NFS) level. Once we get a timeout, it impacts the performance dramatically. If you expect any possibility of retransmission, you should use a TCP-based file transfer program, such as FTP.

You should compare the actual result with an estimated value in an ideal environment. You may be struggling to make the “impossible” possible. If you get 300 to 400 KBps with FTP, it equals 2.4 to 3.2 Mbps. If the network media is Ethernet, because the maximum speed in the specification is 10 Mbps, it would be difficult to double the current speed even if you did have an ideal condition. It′s almost impossible to triple if we consider protocol overhead.

Our Experience

A customer asked us to improve the performance of diskless stations. We gave some advice and recommendations and the performance increased greatly (three to ten times). But they requested more improvement. We got data from a stand-alone system and compared it with the data. The customer′s diskless stations were only 1.3 times slower than the stand-alone system.

Of course, we can make more progress, but we should consider the trade off between the effort/cost and the effect/return.

3.1.1.4 Is that Really a Network Function?

Network function is not the only component to affect performance. Usually an application needs I/O activity. Some applications require a huge amount of memory. Before you dig into the network function deeply, you should confirm that the other system resources, such as memory, I/O and CPU, are not the constraints.

You can use iostat, vmstat and ps commands to review those resources quickly. Also our AIX and RS/6000 provide more sophisticated tools such as svmon.

3.1.2 Check Packet Statistics

First you have to start from facts. For a performance problem, we can get objective information by getting statistic data. This is where we usually start.

If you don′t have any ideas, analyzing statistic data would give you valuable information. If the low performance is coming from an error or retransmission, these events should be logged or counted on the counter somewhere in the kernel. An application may have similar capability. After reading this chapter you will know where and how to read those counters. In our experience, almost all performance problems leave some clues in your statistic data.

It′s not easy to figure out a subtle deviation from the usual state. What is usual is another issue. It must depend on each environment. You need to have a good working knowledge of TCP/IP and the network application or you may not be able to find what you are looking for.

3.1.2.1 Use netstat Command for TCP/IP

Usually, UNIX has a common convenient tool called netstat. This is a very important point. In a multi-vendor environment, you may have to get data from other vendor′s platforms. If it is a UNIX workstation, most likely it has the netstat command and this is why we explain it in this book.

This tool has many functions and some of them have already been explained in this book. It is out of the scope of this book to explain all of them. You can issue this command with various flags, and you can pull necessary information out of the AIX kernel. The following flags are available for obtaining packet statistics:

-v	Data link (adapter) level packet statistics
-i	Network (interface) level datagram statistics and status
-r	Network routing (IP) related datagram statistics and status

-rs	Network routing (IP) statistics
-p ip	Network (IP) level datagram statistics
-p icmp	Network (ICMP) level datagram statistics
-p tcp	Transport (TCP) level segment statistics
-p udp	Transport (UDP) level datagram statistics

It′s beneficial to have statistical data from both the source and the destination systems. Comparing information from both sides of communication links helps you greatly. (Sometimes it is impossible to grasp the situation well enough by only seeing one side.) Unfortunately, if your destination is a PC, you may not expect to get detailed statistical data.

For V4.1, a new flag D is introduced to the netstat command. With this flag, you can get summary statistics of all network layers from device driver to application (NFS). See the example below:

```
# netstat -D

Source                          Ipkts                Opkts     Idrops     Odrops
-------------------------------------------------------------------------------
ent_dev0                          493                  254          0          0
ent_dev1                          189                   12          0          0
                ---------------------------------------------------------------
Devices Total                     682                  266          0          0
-------------------------------------------------------------------------------
ent_dd0                           493                  254          0          0
ent_dd1                           189                   12          0          0
                ---------------------------------------------------------------
Drivers Total                     682                  266          0          0
-------------------------------------------------------------------------------
ent_dmx0                          493                  N/A          0        N/A
ent_dmx1                          189                  N/A          0        N/A
                ---------------------------------------------------------------
Demuxer Total                     682                  N/A          0        N/A
-------------------------------------------------------------------------------
IP                               1003                  999          0          0
TCP                               228                  174          0          0
UDP                               767                  353          0          4
                ---------------------------------------------------------------
Protocols Total                  1998                 1526          0          4
-------------------------------------------------------------------------------
lo_if0                            330                  335          5          0
en_if0                            493                  254          0          0
en_if1                            189                   12          0          0
                ---------------------------------------------------------------
Net IF Total                     1012                  601          5          0
-------------------------------------------------------------------------------
NFS/RPC Client                     48                  N/A          0        N/A
NFS/RPC Server                      0                  N/A          0        N/A
NFS Client                         48                  N/A          0        N/A
NFS Server                          0                  N/A          0        N/A
                ---------------------------------------------------------------
NFS/RPC Total                     N/A                   48          0          0
-------------------------------------------------------------------------------
(Note:  N/A -> Not Applicable)
#
```

3.1.2.2 Use Application Unique Tool If Possible

If your application has its own tool or utility to get statistics or trace, use it. Well-developed products should provide such capabilities. For example, NFS provides the nfsstat tool, and you can review the RPC and NFS procedure statistics.

When you have doubtful software and it doesn′t have a handy tool, you may have to run an AIX system trace; this is not a nice alternative to read system trace because the trace is difficult to read and understand. You also have to choose some trace hooks, and it is difficult to find the appropriate trace hooks. Without any trace hooks, you would get a huge trace log file within a few seconds.

3.1.2.3 Use Error Log and Other Utilities

For AIX, the system error logging function has a substantial role in finding a faulty unit or extraordinary situations. Even when a problem is not counted in the kernel counters, it may be logged in the system error log. An example is a collision of the Ethernet. Of course, reading and understanding an error log is a bit tough, but you may find something.

AIX has some advanced utilities. For example, netpmon can provide very useful statistical data that is not available from any other UNIX tools. The netpmon tells you network I/O statistics for each process. If you are interested in each process′s network activity, this tool is for you.

One convenient tool is iptrace, and we used it a lot during development of this book. As you already know, iptrace captures packets that are addressed to and from your RS/6000. (Broadcast packets are the only exception.) You cannot monitor the entire network traffic with iptrace. Also it can interpret and display a packet up to transport layer, UDP and TCP. For any application activity, you should read hex dump yourself (ONC/RPC is the only exception). On some occasions, you should use a protocol analyzer, if possible.

3.1.2.4 You Have to Interpret the Statistics

After you successfully gather the data or statistics, you have to interpret them and figure out what is the cause of your problem. This may be the most difficult step in problem determination.

An inhibitor may be too much information. Usually you have to get considerable amount of data especially if you don′t have a specific unit or component to focus on. Even an iptrace report for a few minutes can be thousands of lines. It′s not difficult to run trace or monitor tools, once you get to know the necessary flags or options. But if you cannot read and understand an output or the resultant data efficiently, then any sophisticated tool has no use.

In this case it′s better to have a few tools with which you have experience than many tools with which you are a novice user. Learn a few favorite tools completely. This is our recommendation. If you are planning to live in a UNIX world for a while, knowledge of the netstat command is mandatory. Of course, you can exploit netpmon and you would be very comfortable with it until you get another vendor′s system.

3.2 Data Link Layer

The data link layer of TCP/IP usually means LAN, such as Ethernet, token-ring, and FDDI. You can review statistics of this layer with the command netstat -v. In this section, we only show Ethernet and token-ring examples. Some of our encounters are common between adapters, and we believe this section will provide useful hints and tips for all adapters. Unfortunately manuals, (or InfoExplorer) don′t tell much about the output of netstat -v. Instead it is recommended that you refer to the header files located in /usr/include/sys. Since this option shows statistics data recorded by an adapter device driver, the output is quite implementation-dependent.

Note: Be aware that the -v flag is RS/6000-unique. Other vendors′ UNIX don′t have this flag.

Although this book only treats TCP/IP, the command netstat -v shows all data transmitted and received on all the available adapters. Therefore, the counted values may include SNA or other protocol packets or frames. If someone uses HCON or SNA Server/6000 on your system, the counters have been affected by the SNA traffic.

Difference between V4.1 and V3.2

In addition to the netstat -v command, V4.1 provides the following adapter device-specific commands:

- tokstat
- entstat
- fddistat

Accurately speaking, the netstat -v command invokes these commands internally. With these commands, you can reset all counters without rebooting a system. This was not possible with V3.2.

3.2.1 Token-Ring

The following is an example of a token-ring adapter statistic. All counters are not crucial.

```
# netstat -v
-------------------------------------------------------------
TOKEN-RING STATISTICS (tok0) :
Device Type: Token-Ring IBM ISA Adapter
Hardware Address: 08:00:5a:ab:23:19
Elapsed Time: 0 days 1 hours 7 minutes 15 seconds

Transmit Statistics:                      Receive Statistics:
--------------------                      -------------------
Packets: 1849                             Packets: 9613
Bytes: 118211                             Bytes: 2510209
Interrupts: 1849                          Interrupts: 9621
Transmit Errors: 0                        Receive Errors: 0
Packets Dropped: 0                        Packets Dropped: 0
Max Packets on S/W Transmit Queue: 2      Bad Packets: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 1

Broadcast Packets: 8                      Broadcast Packets: 7885
Multicast Packets: 0                      Multicast Packets: 0
Timeout Errors: 0                         Receive Congestion Errors: 0
Current SW Transmit Queue Length: 0
Current HW Transmit Queue Length: 1

General Statistics:
-------------------
No mbuf Errors: 0                         Lobe Wire Faults: 0
Abort Errors: 0                           AC Errors: 0
Burst Errors: 0                           Frame Copy Errors: 0
Frequency Errors: 0                       Hard Errors: 0
Internal Errors: 0                        Line Errors: 0
Lost Frame Errors: 0                      Only Station: 0
Token Errors: 0                           Remove Received: 0
Ring Recovered: 0                         Signal Loss Errors: 0
Soft Errors: 0                            Transmit Beacon Errors: 0
Driver Flags: Up Broadcast Running
        AlternateAddress ReceiveFunctionalAddr 4 Mbps
#
```

Another alternative is the tokstat command. Both commands provide almost equivalent capability. You can refer to the manual or InfoExplorer for a detailed explanation of each counter. See the InfoExplorer or manual pages on tokstat, not netstat, to find a detailed explanation.

```
# tokstat tr0
--------------------------------------------------------
TOKEN-RING STATISTICS (tr0) :
Device Type: Token-Ring High-Performance Adapter (8fc8)
Hardware Address: 10:00:5a:a8:b5:c1
Elapsed Time: 0 days 4 hours 47 minutes 59 seconds

Transmit Statistics:                      Receive Statistics:
--------------------                      -------------------
Packets: 1167                             Packets: 39381
Bytes: 295540                             Bytes: 6985319
Interrupts: 1167                          Interrupts: 39381
Transmit Errors: 0                        Receive Errors: 0
Packets Dropped: 0                        Packets Dropped: 0
Max Packets on S/W Transmit Queue: 2      Bad Packets: 0
S/W Transmit Queue Overflow: 0
```

```
Current S/W+H/W Transmit Queue Length: 0
...
```

One big advantage of the tokstat command is the -r option. With this option you can reset all counters.

```
# tokstat -r tr0
-------------------------------------------------------------
TOKEN-RING STATISTICS (tr0) :
Device Type: Token-Ring High-Performance Adapter (8fc8)
Hardware Address: 10:00:5a:a8:b5:c1
Elapsed Time: 0 days 0 hours 0 minutes 5 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 4                                    Packets: 15
Bytes: 759                                    Bytes: 1537
Interrupts: 4                                 Interrupts: 15
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
Max Packets on S/W Transmit Queue: 1          Bad Packets: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0
...
#
```

Difference between V4.1 and V3.2

With V3.2 you got a different output as follows:

```
# netstat -v

TOKEN STATISTICS (tr0) :

Hardware Address: 10:00:5a:a8:46:2d
Transmit Byte Count: 35160141.0 Receive Byte Count: 130336208.0
Transmit Frame Count: 529837.0  Receive Frame Count: 1247858.0
Transmit Error Count: 1         Receive Error Count: 0
Max Netid's in use: 1           Max Transmits queued: 0
Max Receives queued: 0          Max Stat Blks queued: 0
Interrupts lost: 0              WDT Interrupts lost: 0
Timeout Ints lost: 0            Status lost: 0
Receive Packets Lost: 0         No Mbuf Errors: 0
No Mbuf Extension Errors: 0     Receive Int Count: 1356634
Transmit Int Count: 529831              Packets Rejected No NetID: 108768
Packets Accepted Valid NetID: 1247858   Overflow Packets Received: 0
Packets Transmitted and Adapter Errors Detected: 1

#
```

V3.2 didn't have the tokstat command. Also, there was no way to reset the counters except on rebooting. Even with reboot, the counter Transmit Byte Count and Receive Byte Count could not be reset.

3.2.2 Ethernet

The following is an example of an Ethernet adapter statistic:

```
-------------------------------------------------------------
ETHERNET STATISTICS (ent0) :
Device Type: Integrated Ethernet Adapter
Hardware Address: 08:00:5a:cd:03:21
Elapsed Time: 0 days 12 hours 33 minutes 29 seconds

Transmit Statistics:                         Receive Statistics:
--------------------                         -------------------
Packets: 717                                 Packets: 3790
Bytes: 99510                                 Bytes: 360392
Interrupts: 717                              Interrupts: 3790
Transmit Errors: 0                           Receive Errors: 0
Packets Dropped: 0                           Packets Dropped: 0
Max Packets on S/W Transmit Queue: 0         Bad Packets: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 14                        Broadcast Packets: 2754
Multicast Packets: 8                         Multicast Packets: 0
No Carrier Sense: 0                          CRC Errors: 0
DMA Underrun: 0                              DMA Overrun: 0
Lost CTS Errors: 0                           Alignment Errors: 0
Max Collision Errors: 0                      No Resource Errors: 0
Late Collision Errors: 0                     Receive Collision Errors: 0
Deferred: 1                                  Packet Too Short Errors: 0
SQE Test: 0                                  Packet Too Long Errors: 0
Timeout Errors: 0                            Packets Discarded by Adapter: 0
Single Collision Count: 0                    Receiver Start Count: 1
Multiple Collision Count: 0
Current HW Transmit Queue Length: 8

General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Driver Flags: Up Broadcast Running
        Simplex AlternateAddress
```

We believe these counters are easier to understand. Another alternative is the entstat command. Both commands provide almost equivalent capability. You can refer to the manual or InfoExplorer for a detailed explanation of each counter. (See the InfoExplorer or manual pages on entstat, not netstat, to find a detailed explanation.)

```
$ entstat en0
-------------------------------------------------------------
ETHERNET STATISTICS (en0) :
Device Type: Integrated Ethernet Adapter
Hardware Address: 08:00:5a:cd:03:21
Elapsed Time: 0 days 12 hours 49 minutes 57 seconds

Transmit Statistics:                         Receive Statistics:
--------------------                         -------------------
Packets: 1084                                Packets: 4294
Bytes: 202677                                Bytes: 406368
Interrupts: 1084                             Interrupts: 4294
```

```
Transmit Errors: 0                           Receive Errors: 0
Packets Dropped: 0                           Packets Dropped: 0
Max Packets on S/W Transmit Queue: 0         Bad Packets: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0
...
#
```

One advantage of the tokstat command is the -r option. With this option you can reset all counters. You can refer to the manual or InfoExplorer for a detailed explanation of each counter.

```
# entstat -r en0
-------------------------------------------------------------
ETHERNET STATISTICS (en0) :
Device Type: Integrated Ethernet Adapter
Hardware Address: 08:00:5a:8a:c5:92
Elapsed Time: 0 days 0 hours 0 minutes 0 seconds

Transmit Statistics:                         Receive Statistics:
--------------------                         -------------------
Packets: 0                                   Packets: 0
Bytes: 0                                     Bytes: 0
Interrupts: 0                                Interrupts: 0
Transmit Errors: 0                           Receive Errors: 0
Packets Dropped: 0                           Packets Dropped: 0
Max Packets on S/W Transmit Queue: 0         Bad Packets: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0
...
#
```

Difference between V4.1 and V3.2

With V3.2 you got different output as shown below:

```
# netstat -v

ETHERNET STATISTICS (en0) :

Hardware Address: 08:00:5a:19:00:25
Transmit Byte Count: 516032.0   Receive Byte Count: 19676628.0
Transmit Frame Count: 8495.0    Receive Frame Count: 181469.0
Transmit Error Count: 0         Receive Error Count: 0
Max Netids in use: 7            Max Transmits queued: 0
Max Receives queued: 0          Max Stat Blks queued: 0
Interrupts lost: 0              WDT Interrupts lost: 0
Timeout Ints lost: 0            Status lost: 0
Receive Packets Lost: 0         No Mbuf Errors: 0
No Mbuf Extension Errors: 0     Receive Int Count: 181388
Transmit Int Count: 8514        CRC Error Count: 0
Align Error Count: 0            Recv Overrun Count: 0
Packets Too Short: 0            Packets Too Long: 0
No Resources Count: 0           Recv Pkts Discarded: 0
Xmit Max Collisions: 0          Xmit Carrier Lost: 0
Xmit Underrun Count: 0          Xmit CTS Lost Count: 0
Xmit Timeouts: 0                Parity Errors: 0
Diag Overflow Count: 0          Execute Q Overflows: 0
Execute Cmd Errors: 0           Host side End of List Bit: 0
Adpt side End of List Bit: 0    Adapter pkts to be uploaded: 0
Adapter pkts uploaded: 0        Start receptions to adpt: 1
Receive DMA timeouts(lock up): 0

#
```

V3.2 didn′t have the tokstat command. Also there was no way to reset the counters except to reboot. Even with reboot, the counter Transmit Byte Count and Receive Byte Count could not be reset.

3.3 Network Layer (IP)

The IP Layer is crucial for both TCP and UDP. Since IP is a connectionless protocol, it doesn′t have too many counters.

3.3.1 IP Packet Statistics

The following is an example of the IP datagram statistics. You can use netstat -p ip to review it.

```
# netstat -p ip
ip:
        6087 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 with bad options
        0 with incorrect version number
        1560 fragments received
        0 fragments dropped (dup or out of space)
        0 fragments dropped after timeout
        427 packets reassembled ok
        3704 packets for this host
        0 packets for unknown/unsupported protocol
        0 packets forwarded
        4 packets not forwardable
        0 redirects sent
        2797 packets sent from this host
        0 packets sent with fabricated ip header
        0 output packets dropped due to no bufs, etc.
        0 output packets discarded due to no route
        0 output datagrams fragmented
        0 fragments created
        0 datagrams that can't be fragmented
        1250 IP Multicast packets dropped due to no receiver
        0 ipintrq overflows
#
```

Total packets received
Number of total IP datagrams received.

Bad header checksums
Number of IP datagrams that failed the header checksum calculation. Notice that the IP checksum covers only the IP header and data field is left higher layer's validation. This is usually 0 because almost all of the broken frames could be detected and discarded by the CRC of the data link module.

With size smaller than minimum
Number of IP datagrams that had a longer size in the IP header length field than the actual number of bytes received. This implies that these datagrams were broken for some reason. This is usually 0 because almost all broken frames can be detected and discarded by the CRC of the data link module.

With data size < data length
Number of IP datagrams that were shorter than the minimum IP header length (20 bytes). This implies that these datagrams were broken for some reason. This is usually 0 because almost all broken frames can be detected and discarded by the CRC of the data link module.

With header length < data size
Number of IP datagrams that had shorter size in the IP header length field, than the minimum IP header length (20 bytes). This implies that the sender's IP module was implemented incorrectly (or receiver's checking function of IP module was implemented incorrectly, if you believe the sender).

With data length < header length
: Number of IP datagrams that were shorter than the size in the IP header length field. This implies that these datagrams were broken for some reason (or the length field was incorrectly written, if you believe the network is reliable). This is usually 0 because almost all of the broken frames could be detected and discarded by the CRC of the data link module.

  Number of IP datagrams that actual length of was shorter than the header length field. This is usually 0 because almost all of the broken frames could be detected and discarded by the CRC of the data link module.

With bad options
: Number of IP datagrams that had a bad option field. The term bad means incorrect or not support options. The wrong implementation (of destination system) may have this counter incremented.

With incorrect version number
: Number of IP datagrams that had incorrect version number. Currently, only the available and valid version number of IP protocol is 4. This counter should not be incremented. Now a new version (maybe version 6) is under development.

Fragments received
: Number of total fragments received.

Fragments dropped (dup or out of space)
: Number of fragments discarded because a duplicated fragment was detected or the system used up the buffer space.

Fragments dropped after timeout
: Number of fragments discarded because all of fragments needed to reassemble the original IP datagram were not received within the timeout period. The default timeout period is 60 seconds for RS/6000.

Packets reassembled ok
: Number of IP datagrams that were reassembled here without any problems.

Packets for this host
: Number of IP datagrams that the final destination of was this system.

Packets for unknown/unsupported protocol
: Number of IP datagrams that were addressed to the unsupported protocols.

Packets forwarded
: Number of IP datagrams forwarded by the system. This shows the routing activity of this system and only the router system should get this counter incremented.

Packets not forwardable
: Number of IP datagrams that cannot be forwarded. A router system that doesn′t have a necessary route to forward IP datagrams gets this counter incremented. A system that doesn′t have routing capability (non-router system) and receives an IP datagram destined for another system, also gets this counter incremented. This means there is a system (or systems) that has the wrong routing information.

Redirects sent
Number of the ICMP route redirect packets sent out. Only a router system should send this ICMP packet. This counter shows that there was a misconfigured system that didn′t know the correct router. By the ICMP message, the routing table of that system might be dynamically updated.

Packets sent from this host
Number of IP datagrams that were created and sent out from this system. This counter doesn′t include the forwarded datagrams (passthrough traffic).

Packets sent with fabricated IP header
Number of IP datagrams that had a fabricated IP header.

Output packets dropped due to no bufs, etc.
Number of IP datagrams that were lost due to mbuf or mbuf-cluster shortage.

Output packets discarded due to no route
Number of IP datagrams that were discarded because there was no routing information to the destinations.

Output datagrams fragmented
Number of IP datagrams that were fragmented here when they were sent out, without any problems.

Fragments created
Number of fragments created in this system when IP datagrams were sent out.

Datagrams that can′t be fragmented
Number of IP datagrams that had a don′t fragment bit set. The don′t fragment (DF) flag means that this IP datagram cannot be fragmented by a router. If the router has a smaller MTU than the IP datagram to forward, the router has to discard the IP datagram. In this case, the ICMP destination unreachable message (fragmentation needed and the do not fragment bit set) is sent back. Usually the don′t fragment bit in the IP header is set by a boot server of a diskless/dataless machine or X-Station to boot them.

IP Multicast packets dropped due to no receiver
Number of IP multicast datagrams that were received but discarded because no corresponding IP multicast application programs are running. It′s not strange to see a big number in this counter. Somebody (or program) may be sending IP multicast datagrams destined for all-host group (244.0.0.1).

ipintrq overflows
Number of IP receive queue overflowed. If your system received a lot of IP datagrams in a very short period, or the CPU is heavily loaded, IP datagrams can not be processed timely and accumulate in the IP receive queue. This counter shows the number of IP datagrams lost.

Some counters should always remain 0 because we have the CRC mechanism in the most data link layer. But SLIP, for example, doesn′t have the CRC.

Pay attention to the ratio of divided by the fragments received divided by the total packets received. A high fragment ratio implies both sending and receiving systems are heavily loaded by fragmentation and reassembly. If possible, you should adjust the maximum transfer unit (MTU) in order to reduce the fragmentation.

The counts of fragments dropped (dup or out of space) is very symptomatic. Dropping one fragment invites an entire IP datagram retransmission and the cost is significant.

If your system is a router, check the ratio of packets forwarded divided by the total packets received. This shows how much the routing function is used and you can evaluate the usage of routes defined on your system. The counter packets not forwardable and the redirects sent, imply that there is a misconfigured system in your network.

Difference between V4.1 and V3.2

Although AIX V3.2 provided the same command, the output had fewer counters, as shown below:

```
# netstat -p ip
ip:
        319933 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 fragments received
        0 fragments dropped (dup or out of space)
        0 fragments dropped after timeout
        18 packets forwarded
        136944 packets not forwardable
        0 redirects sent
#
```

3.3.2 no Command Options for Router Configuration and ICMP

Our RS/6000 behaves as a router by default (even if it doesn′t have two or more interfaces). You can change this by using the ipforwarding option of the no command. Also, the no command provides an option to suppress the ICMP route redirect messages. The following is an example on how to check both configurations:

```
# no -a
...
                ipfragttl = 60
          ipsendredirects = 1
             ipforwarding = 0
                  udp_ttl = 30
...
           rfc1122addrchk = 0
            nonlocsrcroute = 0
            tcp_keepintvl = 150
...
                ipqmaxlen = 100
       directed_broadcast = 1
#
```

ipsendredirects

If you set this option to 0, the system no longer generates the ICMP route redirect messages. If your system is not a router, this option should be set to 0 and you can avoid unnecessary routing table updates on other systems. The default is 1.

ipforwarding

If you set this option to 0, the system no longer forwards IP datagrams that are destined for other systems. In other words, you can disable the route or gateway capability. The default is 0.

nonlocsrcroute

IP has the option source route. With this option, a source system can explicitly specify the route to the destination. A list of routers, which the IP datagram must pass through, is written in the option field of the IP header. If this system is a router, setting this option to 1 enables a non-local source-routing, which passes IP datagrams (using the source route option) through the router (if the next hop will be through the same physical interface through which the IP datagrams arrived). This is non-local source-routing. This option is due to RFC 1122 (it must be configurable) and the default is 0.

directed_broadcast

If you set this option to 1, your system forwards broadcast datagrams when the network number of the destination address is for other network. If you set 0, your system cannot forward any broadcast datagram. The default is 1. This option is only effective when you set `ipforwarding=1`. (This capability is only for router.)

If your system is not a router, our recommendation is to set both the parameters, ipforwarding and ipsendredirects, to 0. If a system (not intended to be a router) behaves as a router, what happens? If the system has the default route, the wrong IP datagrams assuming the system to be a router, will be forwarded to the default gateway (another router). As a result, it hides the configuration problems and make the debug difficult.

Difference between V4.1 and V3.2

With AIX V3.2, the default value of the ipforwarding is 1. As you can see, the default of ipforwarding=1 violates the RFC 1122. On the contrary, as you have experienced with V3.2, these default values make the TCP/IP configuration task easier for a router system because you don't need any further configuration procedures other than a non-router system.

RFC 1122 Requirements for Internet Hosts, page 28-29

Any host that forwards datagrams generated by another host is acting as a gateway and MUST also meet the specifications laid out in the gateway requirements RFC [INTRO:2]. An Internet host that includes embedded gateway code MUST have a configuration switch to disable the gateway function, and this switch MUST default to non-gateway mode. In this mode, a datagram arriving through one interface will not be forwarded to another host or gateway (unless it is source-routed), regardless of whether the host is single-homed or multi-homed. ...

If you need to change any parameter, issue the no command as follows. This example changes the current value of ipforwarding from 0 to 1:

1. Checking current value.

   ```
   # no -o ipforwarding
   ipforwarding = 0
   #
   ```

2. The change is to 1.

   ```
   # no -o ipforwarding=0
   #
   ```

3. Confirm if the change was made.

   ```
   # no -o ipforwarding
   ipforwarding = 1
   #
   ```

Since the no command updates the parameters in the kernel, rebooting the system clears the update and returns it to the default value. If you want to make the updated configuration permanent, you should write the no command in the startup script /etc/rc.net. The best place for this is at the end of the script as follows:

```
###################################################
# The socket default buffer size (initial advertised TCP window) is being
# set to a default value of 16k (16384). This improves the performance
# for Ethernet and token-ring networks.  Networks with lower bandwidth
# such as SLIP (Serial Line Internet Protocol) and X.25 or higher bandwidth
# such as Serial Optical Link and FDDI would have a different optimum
# buffer size.
# ( OPTIMUM WINDOW = Bandwidth * Round Trip Time )
###################################################
if [ -f /usr/sbin/no ] ; then
        /usr/sbin/no -o tcp_sendspace=16384
        /usr/sbin/no -o tcp_recvspace=16384
        /usr/sbin/no -o ipforwarding=1
        /usr/sbin/no -o ipsendredirects=0
fi
```

Important Notice For AIX V4.1 Users

The new version of our AIX, V4.1 has the default value 0 for the ipforwarding. The V4.1 complies with RFC 1122. This also means that you have to specifically set this option to 1, if your AIX should be a router.

People who are very familiar with the AIX V3.2, may overlook this difference. With the V3.2, if you have configured the two network interfaces, it automatically becomes a router. But, with V4.1, configuring the two interfaces doesn't make your system a router.

3.3.3 Network Interface Statistics

You can get statistics for each network interface with netstat -i. This command can also be used to confirm if an interface is really in an up state.

```
# netstat -i
lo0   16896 <Link>                           575     0      575     0     0
lo0   16896 127        localhost             575     0      575     0     0
tr0   1492  <Link>40.0.7e.8.66.70           7803     0     3163     0     0
tr0   1492  9.68.214    zero.hakozaki.i     7803     0     3163     0     0
#
```

Name Name of each interface.

MTU Maximum Transfer Unit (MTU) of each interface.

Network Network address to which the interface is directly attached. If the network address is registered in the /etc/networks file, the network name in this file is displayed.

Address The host name of each interface. Usually the IP address is internally used and translated to the host name by using /etc/hosts or DNS/NIS. Use netstat -in to show the IP address. This alternative is useful when you get a problem at the name server.

Ipkts Number of IP datagrams received by this interface.

Ierrs Number of errors detected by this interface, for received IP datagrams. The IP datagram was discarded. For a data link that has CRC, this field should be almost 0. Notice that not only the damaged datagrams but also the datagrams which have an unsupported format or option are counted.

Opkts Number of IP datagrams sent out from this interface.

Oerrs Number of errors for IP datagrams sent out from this interface. It is those IP datagrams that were discarded or not sent out. A reason may be that the adapter card can not catch up with the sending speed of the IP module. Use the spray command and place a heavy load on the interface. Then you can get this error.

Coll Collisions detected by this interface. Be aware that currently this counter is only supported by SLIP. Ethernet doesn′t support this counter and it is always 0.

All of these counters are cleared by a system reboot. Maybe the biggest pitfall is that the coll field is not supported by Ethernet. Use netstat -v and errpt as alternatives.

Note: When you ping or send the IP datagram to your system, even with the host name (not with the name, localhost), all those IP datagrams are counted at the interface lo0.

With the specified interface name and time interval, you can get an IP packet traffic log as shown below. The first line shows the total count from the boot, and each line represents a statistic for the specified interval (10 seconds in this case).

```
# netstat -I tr0 10
    input    (tr0)     output           input   (Total)    output
 packets  errs  packets  errs colls  packets  errs  packets  errs colls
    7816     0     3167     0     0     8391     0     3742     0     0
       4     0        0     0     0        4     0        0     0     0
       3     0        0     0     0        3     0        0     0     0
       3     0        0     0     0        3     0        0     0     0
       2     0        1     0     0       18     0       17     0     0
...
```

3.3.4 IP Packet Statistics by Route

You can evaluate the usage of each route configured on your system by using netstat -r. This option is the most used option when you use the routing table.

```
# netstat -r
Routing tables
Destination      Gateway            Flags     Refs     Use  Interface
Netmasks:
255.255.255.128

Route Tree for Protocol Family 2:
default          9.68.214.1         UG         3     1529  tr0
9.68.214         zero.hakozaki.ibm. U          8     2020  tr0
localhost        localhost          UH         2      240  lo0
#
```

Destination Destination IP network address or host address where each route is configured. The address 0.0.0.0 is reserved for the default route.

Gateway The next router, an IP datagram, is sent to between this interface and the destination. Check to see whether the flags are U (and not UG). Gateway doesn′t mean the next router but the network interface itself, and the destination should be in the same local IP network.

Flags Status of each route.

- **U** This route is currently up and available.
- **G** This route is addressed to gateway (router) in the Gateway field. Any route without the G flag is addressed to the local interface on the system.
- **H** The destination of this route is a specific host.
- **D** This route was dynamically added by the ICMP redirect function.
- **M** This was modified by the ICMP redirect function. Modify means that the destination address was updated.

Refcnt Number of current active TCP connections using this route. Since TCP first refers to the routing table to find a route to the destination, this counter is called a reference count. This counter is not cumulative and only shows the current status.

Use Total number of IP datagrams that were sent out from this route. This is not a byte count. Routed (passthrough) IP datagrams are also counted.

Interface Name of the interface where each route is attached.

Note: If you ping to your system by the host name, it is counted to the corresponding route. Counter to the 127 loopback is only incremented when you explicitly use localhost or loopback. Make sure that you compare this with netstat -i.

An interesting thing is we now know the number of current active TCP connections. As you know, UDP is a connectionless protocol and has no concepts of connection. This means that each UDP datagram (which is mapped to an IP datagram) completes by itself and the UDP module doesn′t keep relation among each UDP datagram. The IP is also a connectionless protocol and the matter is the same. So there is no way for us to know if a route is continuously used by any UDP application. Of course, the UDP application should know it. This invites an interesting administrative issue. If your system is a router and you need to shut down the system, you know about TCP if someone is using your system as a router (in other words, if you have passthrough traffic). However, you cannot grasp anything about UDP. As a result, someone′s NFS operation through your system may hang.

You can review the statistic of the routing activity by the ICMP function. Use netstat -rs as shown below:

```
# netstat -rs
routing:
        508 bad routing redirects
        0 dynamically created routes
        0 new gateways due to redirects
        2466 destinations found unreachable
        0 uses of a wildcard route
#
```

bad routing redirects
: Number of received ICMP route redirect messages that had invalid information. For example, the informed gateway was not in the same local IP network.

dynamically created routes
: Number of the routes dynamically created by ICMP route redirect. You should have route with D flag, if this counter got incremented.

new gateways due to redirects
: Number of the routes dynamically created by ICMP route redirect. You should have route with M flag, if this counter got incremented.

destinations found unreachable
: Number of failed routing table lookups due to no routes to the destination. This should have made your system send an ICMP destination unreachable, if this counter got incremented. Your system should be a router when you see this counter is not 0.

uses of a Wildcard route
: Number that default route used.

3.4 Network Layer (ICMP)

The ICMP is the supplemental protocol to manage and control the IP. As we already know, it supports many ICMP types and some types have several codes. The ICMP provides maintenance functions such as adding a new route dynamically or carrying error messages. Watching the ICMP messages tells you very valuable information. These messages tell you whether your network configuration needs adjustment. Also, you may find that your network configuration is not optimized or perfectly correct (but it works without explicit problem symptoms).

If you need to understand the ICMP statistics completely, you have to know the mechanism of ICMP.

3.4.1 Internet Control Message Protocol (ICMP) Basics

Again we show you Type and Code used by ICMP. Although there are many types and codes, not all of them are mandatory. We have many different implementations about ICMP. For example, destination unreachable, port unreachable are defined as *should* in the RFC. Source quench is defined as *may* in the RFC. For details, refer to the *RFC 1122 Requirements for Internet Hosts*.

The ICMP message types are grouped into two classes. One is error and the other is query.

3.4.1.1 ICMP Error Messages

The first group of ICMP message types are for error notification. For a complete list, refer to the *RFC 1700 ASSIGN NUMBERS*. The following types and codes are currently defined:

Type	Message		
3	Destination Unreachable		
		Code	***Meaning***
		0	Network Unreachable
		1	Host Unreachable
		2	Protocol Unreachable
		3	Port Unreachable
		4	Fragmentation needed and the do not fragment bit set.
		5	Source Route Failed
		6	Destination Network Unknown
		7	Destination Host Unknown
		8	Source Host Isolated (Obsolete)
		9	Destination Network Administratively Prohibited
		10	Destination Host Administratively Prohibited
		11	Network Unreachable for TOS
		12	Host Unreachable for TOS
		13	Communication Administratively Prohibited by Filtering
		14	Host Precedence Violation
		15	Precedence Cutoff in Effect
5	Route Change Request		
		Code	***Meaning***
		0	Redirect datagrams to go to that network.
		1	Redirect datagrams to reach that host.
		2	Redirect datagrams for that network with that TOS.
		3	Redirect datagrams for that host with that TOS.

4 Source Quench

Time Exceeded for Datagram

Code	***Meaning***
0	Time-To-Live Equals 0 During Transit
1	Time-To-Live Equals 0 During Reassembly

Parameter Problem Message

Code	***Meaning***
0	IP Header Bad
1	Required Option Missing

3.4.1.2 ICMP Query Messages

The second group of ICMP message types are for information query and response. For a complete list, refer to the *RFC 1700 ASSIGN NUMBERS*. The following types are currently defined:

Type	***Message***
0	Echo Reply
8	Echo Request
13	Time Stamp Request
14	Time Stamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply

The most used types are echo request and echo reply using the ping command. These types are for diagnostic purposes. Any data in echo request message must be returned by echo reply message. By the *RFC 1122*, any system *must* implement echo server function. This means that any system must be able to receive the ICMP echo request message and return the ICMP echo reply message correctly. But the echo client function *should* be implemented for all systems. Then any system, even if it doesn′t have a ping command or equivalent, would respond to your ping command. This is why everyone uses ping.

Information request and information reply are now considered to be obsolete. The RFC defines that they *should not* be implemented. Up to now we have never seen these message types that were counted and displayed with the command netstat -p icmp. This message type was designed to load configuration information for diskless or X-station. But as you know, we currently use RARP or BOOTP.

3.4.2 no Command Option for ICMP (Address Mask)

We have a little complicated story about the address mask request and address mask reply. We use the SMIT or ifconfig command to set a subnet mask for an interface. Due to the RFC 1122, any system *must* provide this or a similar procedure. Also, the address mask *may* be configured by sending the ICMP address mask request(s) and the receiving ICMP address mask reply(s). This means that a system can send address mask requests as a broadcast during the boot and configure its subnet mask based on the address mask reply. In order to make this scheme work, there must be at least one system that responds to the address mask requests in the same IP network. That system is called an authoritative agent. By the RFC 1122, any system that is not an authoritative agent *must not* send an address mask reply and there *should be* a flag to configure a system to be an authoritative agent.

RFC 1122 Requirements for Internet Hosts, Page 46

A system MUST NOT send an address mask reply unless it is an authoritative agent for address masks. An authoritative agent may be a host or gateway, but it MUST be explicitly configured as a address mask agent.

With a statically configured address mask, there should be an additional configuration flag that determines whether the host is to act as an authoritative agent for this mask, i.e., whether it will answer Address Mask Request messages using this mask.

Unfortunately, our AIX, until V3.2.4, had violated this rule. Those AIXs automatically responded to address mask requests. After V3.2.5, we have a new option icmpaddressmask for the no command and we can suppress responding (of course, the default is not to respond).

```
# no -a
...
             tcp_keepidle = 14400
          icmpaddressmask = 0
                  rfc1323 = 0
              tcp_mssdflt = 512
                ipqmaxlen = 50
#
```

icmpaddressmask
If you set this option to 1, your system automatically responds to address mask requests. The default value is 0.

3.4.3 ICMP Message Statistics

We can monitor the ICMP statistic with the netstat -p icmp command. Remember in these statistics, some counters are not shown if the value is 0. This is a slight implementation deviation from other options. You don′t need to worry about it if your output is not the same as the following example:

```
# netstat -p icmp
icmp:
        1555 calls to icmp_error
        0 errors not generated because old message was icmp
        Output histogram:
                echo reply: 5028
                destination unreachable: 1412
                routing redirect: 23648
                time exceeded: 143
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo reply: 110
                destination unreachable: 22
                routing redirect: 7782
                echo: 5028
                time exceeded: 48
        5028 message responses generated
#
```

Calls to icmp_error
Total number of error ICMP message sent out from this system. Errors such as destination unreachable and control message such as echo or routing redirect are not counted.

Errors not generated because old message was icmp
ICMP error message couldn′t be sent because the incoming IP datagram was carrying the ICMP error message. No ICMP error messages are generated to the incoming ICMP error message. If this is allowed, the network can be filled as the storm of ICMP error messages.

Output histogram:
Statistics of ICMP outgoing messages by ICMP type. Although each ICMP error type has the code to identify detailed reasons, those codes are not shown.

Echo reply
Number of echo replies sent out. This means someone pinged to your system and your system responded.

Destination unreachable
Number of ICMP destination unreachable messages sent out. If you got this counter displayed, there is most certainly a misconfigured system that has the wrong routing table. In this case, your system (should be a router) couldn′t find a route to forward the IP datagram(s).

Routing redirect
Number of ICMP route redirect messages sent out. If you have this counter displayed, there seems to be a misconfigured system that has the wrong routing table. In this case, your system (should be a router) was able to find a route (the correct router) to forward IP datagram(s), and informed the correct router to the source system.

Time exceeded
Number of ICMP time exceeded sent out. When an incoming IP datagram should have been forwarded, but it had a TTL of 1 or less (0), then the datagram was discard and this message was sent out. TTL should be decremented by one or more at any router when the datagram passes it, and if the TTL reaches to 0, it must be discarded.

Messages with bad code fields
Number of ICMP messages that had incorrect type.

Messages < minimum length
Number of ICMP messages that had shorter length than the minimum ICMP message (packet). The length of the ICMP message depends on each type. This is usually zero because almost all of the broken frames could be detected and discarded by the CRC of the data link module.

Bad checksums
Number of ICMP messages that had failed to the checksum validation. These ICMP messages were damaged during transmission. This is usually zero because almost all of the broken frames could be detected and discarded by the CRC of the data link module.

Messages with bad length

Number of ICMP messages which had incorrect length. This is usually 0 because almost all of the broken frames could be detected and discarded by the CRC of the data link module.

Input histogram:

Statistics of ICMP incoming messages by ICMP type. Although each ICMP error type has code to identify detailed reason, those codes are not shown.

Echo reply

Number of echo reply received. You pinged some system and the system responded.

Destination unreachable

Number of destination unreachable received. This means that the IP datagram you sent couldn′t be forwarded by a router due to the missing route, or the destination system couldn′t deliver the datagram to the destination ports, because the port was not opened. Of course, we have many other potential reasons and consider that many codes are defined for this type.

Routing redirect

Number of ICMP route redirect message received. This means your system sent an IP datagram to the wrong router. This counter doesn′t necessarily mean that some routes were added or updated dynamically because the ICMP route redirect message might have the wrong information.

Echo

Number of echo request received. This means someone pinged to your system. If there are no problems, this counter should have the same value with the counter echo reply of the Output histogram.

Time exceeded

Number of time exceeded received. This means a packet sent out from your system was discarded due to 0 TTL, after it got decremented.

Message responses generated

Number of ICMP response messages sent out. ICMP response messages were the response to the incoming ICMP messages. This may be the sum of the echo reply, time stamp reply, address mask reply, etc. The ICMP error due to the incoming regular IP datagram was not included.

3.4.4 Modified Route by ICMP

In this section, we show an actual example of the IP routing activity and give you a clear view to each counter. The system mat has the following routing table:

```
mat # netstat -r
Routing tables
Destination      Gateway            Flags     Refs     Use  Interface
Netmasks:
(0) 0 ffff ff00
255.255.255.128

Route Tree for Protocol Family 2:
default          9.68.214.1         UG          2      421  tr0
9.68.214         mat.hakozaki.ibm.c U          10     2019  tr0
9.170.4          zero.hakozaki.ibm. UG          0        0  tr0
localhost        localhost          UH          1        0  lo0
mat #
```

When the mat sends a packet to the network 9.170.4, it should be forwarded to the router system zero. On the system zero, we have the following routing table. You could notice that a packet to the network 9.170.4 is again forwarded to another router system, kashima:

```
zero # netstat -r
Routing tables
Destination      Gateway            Flags     Refs     Use  Interface
Netmasks:
(0) 0 ffff ff00
255.255.255.128
(0) 0 ffff ff00 0 0 0 0
(0) 0 ffff ff80 0 0 0 0

Route Tree for Protocol Family 2:
default          9.68.214.1         UG          3     1704  tr0
9.68.214         zero.hakozaki.ibm. U           8     2702  tr0
9.170.4          kashima.hakozaki.i UG          0        0  tr0
localhost        localhost          UH          2      240  lo0
zero #
```

This is a Bad Practice

As you can see in the previous, the zero doesn′t have two network interfaces and should *not* be configured as a router. Since ipforwarding=1 is set, you can make an RS/6000 a router just by configuring an additional route. From this perspective,it is a bad example.

In our environment, the IP network 9.170.4 is completely imaginary and the system kashima doesn′t have the corresponding route (this may cause an error intentionally at kashima). Now we initiate TELNET from the mat to the imaginary destination system 9.170.4.20 and watch what will happen. This IP trace was obtained at the system mat:

1. We did this on the mat:

   ```
   mat # tn 9.170.4.20
   Trying...
   ```

2. The system mat sends a telnet connection request to the system 9.170.4.20. This packet is sent to the router zero. The destination MAC address 40:00:7e:08:66:70 is of the zero.

   Note: Although we omitted it, an ARP and DNS look up might be involved before sending this packet.

```
Packet Number 66
TOK: ====( 66 bytes transmitted on interface tr0 )==== 17:05:22.652454784
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =      9.170.4.20 >
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=2558, ip_off=0
IP:     ip_ttl=60, ip_sum=877a, ip_p = 6 (TCP)
TCP:    <source port=1039, destination port=23(telnet) >
TCP:    th_seq=b7190c01, th_ack=0
TCP:    th_off=6, flags<SYN>
TCP:    th_win=16384, th_sum=a399, th_urp=0
TCP: 00000000     020405ac                                |....            |
```

3. The route zero notices that the above TCP segment should have been sent to the router kashima, because the zero knows the kashima is the right router to the 9.170.4 network and the kashima is in the same IP network by looking at its routing table. Therefore, it sends back the ICMP route redirect message to the source mat telling it that any packet sent to the 9.170.4.20 should be directly sent to the kashima from now. Back on this ICMP message, the zero forwards the TCP segment to the kashima.

```
Packet Number 67
TOK: ====( 78 bytes received on interface tr0 )==== 17:05:22.655365248
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=56, ip_id=4807, ip_off=0
IP:     ip_ttl=255, ip_sum=e9ce, ip_p = 1 (ICMP)
ICMP:   icmp_type=5 (REDIRECT)
ICMP:   icmp_code=1 (redirect 9.170.4.20 to HOST 9.68.214.76)
```

4. The TCP segment was forwarded to another router kashima and the kashima doesn′t have a route to the final destination 9.170.4.20. In this experiment, the kashima has a default route, and the packet was further forwarded to somewhere. We don′t know where it ended up finally. No responses are returned to the mat. If the kashima doesn′t have a default route, it sends back the ICMP destination unreachable (host unreachable) message to the souce system mat.

5. Then the mat resends the TCP connection request segment (SYN segment) again. This time the request is directly sent to the kashima, because it received the ICMP route redirect message (and routing table has been updated already). Notice that the destination MAC address 40:00:7e:06:44:40 is of the kashima. The mat continues to send TCP SYN segments until the timer expires (75 seconds). In our experiment, six SYN segments were sent (retransmission).

```
Packet Number 70
TOK: ====( 66 bytes transmitted on interface tr0 )==== 17:05:28.261340160
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:06:44:40]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =      9.170.4.20 >
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=2559, ip_off=0
IP:     ip_ttl=60, ip_sum=8779, ip_p = 6 (TCP)
TCP:    <source port=1039, destination port=23(telnet) >
TCP:    th_seq=b7190c01, th_ack=0
TCP:    th_off=6, flags<SYN>
TCP:    th_win=16384, th_sum=a399, th_urp=0
TCP: 00000000     020405ac                                |....            |
```

Finally, the TELNET ended up with the timeout error, as below because only the final SYN segment retransmission timeout was notified as an error to the application, TELNET client. This is a valid implementation as in the RFC 1122.

```
mat # tn 9.170.4.20
Trying...
telnet: connect: Connection timed out
tn> q
mat #
```

RFC 1122 Requirements for Internet Hosts, Pages 103-104

4.2.3.9 ICMP messages

TCP MUST act on an ICMP error message passed up from the IP layer, directing it to the connection that created the error.

- Destination Unreachable -- codes 0, 1, 5

  Since these Unreachable messages indicate soft error conditions TCP MUST NOT abort the connection, and it SHOULD make the information available to the application.

  DISCUSSION:

  TCP could report the soft error condition directly to the application layer with an upcall to the ERROR_REPORT routine, or it could merely note the message and report it to the application only when if the TCP connection times out.

During the above operation, the routing table of the mat was updated as follows. A route to the 9.170.4.20 was dynamically added by the ICMP message. The flag is UGHD and you already know the meaning.

```
mat # netstat -r
Routing tables
Destination      Gateway            Flags     Refs     Use  Interface
Netmasks:
(0) 0 ffff ff00
255.255.255.128

Route Tree for Protocol Family 2:
default          9.68.214.1         UG          1      436  tr0
9.68.214         mat.hakozaki.ibm.c U          10     2230  tr0
9.170.4          zero.hakozaki.ibm. UG          0        1  tr0
9.170.4.20       kashima.hakozaki.i UGHD        0        1  tr0
localhost        localhost          UH          1        0  lo0
mat #
```

If we see the ICMP statistics on the source system mat, we could find that the following counter was incremented by one. This means that the system received an ICMP route redirect message:

```
mat # netstat -p icmp
icmp:
        29 calls to icmp_error
        0 errors not generated because old message was icmp
        Output histogram:
                echo reply: 143
                destination unreachable: 29
                routing redirect: 1
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo reply: 3
                routing redirect: 1
                echo: 145
        143 message responses generated
mat #
```

You can also find out this with the following counter. That counter was incremented by one. This shows that an update was made to the routing table and a route was added.

```
mat # netstat -rs
routing:
        0 bad routing redirects
        1 dynamically created routes
        0 new gateways due to redirects
        8 destinations found unreachable
        0 uses of a wildcard route
mat #
```

If we could see the same counter on the wrong router, zero, we would find that the following counter was incremented by one. You would know that this system sent out an ICMP route redirect message.

```
zero # netstat -p icmp
icmp:
        6 calls to icmp_error
        0 errors not generated because old message was icmp
        Output histogram:
                echo reply: 134
                destination unreachable: 6
                routing redirect: 2
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo reply: 9
                routing redirect: 1
                echo: 134
        134 message responses generated
zero #
```

3.5 Transport Layer

Now we briefly examine transport layer statistics. There are two protocols for this layer, TCP and UDP. These two protocols have completely different mechanisms for each other. Since TCP is a connection oriented protocol, it keeps much information (counters). UDP is a connectionless protocol, and there is little information about it.

3.5.1 TCP Segment Statistics

Since TCP is fairly complicated protocol, there are many counters we should read. If you want to understand the meaning of all the counters, you have to understand TCP completely. We briefly review all of them, as follows:

```
# LANG=C netstat -p tcp
tcp:
        871870 packets sent
                546355 data packets (2336335 bytes)
                44 data packets (2218 bytes) retransmitted
                318230 ack-only packets (317007 delayed)
                0 URG only packets
                0 window probe packets
                6425 window update packets
                836 control packets
     641380 packets received
                546454 acks (for 2337972 bytes)
                409 duplicate acks
                0 acks for unsent data
                597988 packets (47642225 bytes) received in-sequence
                375 completely duplicate packets (160 bytes)
                0 packets with some dup. data (0 bytes duped)
                193 out-of-order packets (5 bytes)
                19 packets (19 bytes) of data after window
                19 window probes
                33 window update packets
                28 packets received after close
                0 discarded for bad checksums
                0 discarded for bad header offset fields
                0 discarded because packet too short
        368 connection requests
        134 connection accepts
        463 connections established (including accepts)
        897 connections closed (including 9 drops)
        17 embryonic connections dropped
        546751 segments updated rtt (of 546839 attempts)
        67 retransmit timeouts
                1 connection dropped by rexmit timeout
        0 persist timeouts
        27301 keepalive timeouts
                88 keepalive probes sent
                2 connections dropped by keepalive
#
```

Note: In this output, the term packet is used. Accurately speaking, the TCP packet is called a segment.

packets sent
Number of segments sent out. They are subdivided into the following counters:

data packets (2336335 bytes)
Number of segments which carried data. The bytes means total bytes carried by these segments.

data packets (2218 bytes) retransmitted
Number of data segments retransmitted due to the ACK timeout of some reasons. The bytes means total bytes carried by these segments.

ack-only packets (317007 delayed)
Number of segments carried by the ACK flag only. TCP is a bidirectional connection. If a system has both data and an ACK to be sent, it can send both in the same segment. If the system doesn′t have data to be sent, just the ACK only segment is sent out. The delayed means that TCP has some built-in sophisticated mechanism that optimizes transmission. In order to reduce traffic, the ACK only segment is not sent out immediately. During the waiting period, some data may be generated and the ACK can be sent with the data. Unfortunately, if the data is not generated within the waiting period, the ACK only segment is sent. This is a delayed ACK only segment.

URG only packets
Number of segments that carried the URG flag only. The URG means urgent and if a sender has some urgent data, this segment is sent.

window probe packets
TCP uses window to control the data flow. A sender that can send data up to the size that the receiver says is an acceptable amount of data, is called window. The receiver system advertises its receiving window size by using the window field of the TCP header. When the receiver′s window is 0, the sender cannot send. If the sender has not received a segment for a certain period telling that the receiver′s window was updated (has some room), the sender sends a probe segment to the receiver to ask for the current window size.

window update packets
Number of segments sent out, which were carrying only window update information.

control packets
Number of segments sent out, which were carrying only a flag, SYN, FIN or RST. SYN is used to establish a connection. FIN is used to close a connection. RST is used to abort a connection.

packets received
Number of segments received. They are subdivided into the following counters:

acks (for 2337972 bytes)
Number of received segments carried ACK only. The bytes are total bytes sent out and confirmed by the receiver by these ACK segments only.

duplicate acks
Number of received segments that had duplicated ACK (the ACK sequence number contains the sequence number which were already acknowledged.) This may be due to retransmission. This doesn′t mean a completely identical segment is retransmitted. TCP always tries to optimize the transmission and more data may be included in the retransmitted segment.

acks for unsent data
Number of received segments that has the ACK sequence number for unsent data. This could happen.

packets (47642225 bytes) received in-sequence
Number of received segments in correct order. Since TCP uses SEQ (sequence) number and ACK (acknowledge) number to control segments, the segments should be sent and received in the correct order (based on SEQ and ACK number), and counted here. The bytes means total bytes received with these segments. With certain network conditions, it is possible to receive segments out of order due to a delay at the routers.

completely duplicate packets (160 bytes)
Number of received segments that were completely duplicated. These segments should be due to the retransmission. This counter implies that there is a flow control problem.

packets with some dup. data (0 bytes duped)
Number of received segments that had some duplicated data. In TCP, duplication doesn′t necessarily mean that the entire segment is duplicated. These segments should be due to the retransmission. This counter implies that there is a flow control problem.

out-of-order packets (5 bytes)
Number of received segments that were arrived at out of order. This is possible but should be rare. Unreliable network media can drop some of segments and invites some retransmissions. This can be the reason.

packets (19 bytes) of data after window
Number of received segments that was a bigger sequence number than the upper limit of a received window. If the TCP window mechanism worked perfectly, this should not happen. This can be caused by a segment on the way when the receive window size was updated to a small size.

window probes
Number of window probe segments received. These segments are to ask if the system window size got bigger than zero. This means this system had zero receive window sometimes. This implies that the system was heavily loaded by TCP data segments, or the receive buffer was too small.

window update packets
Number of window update segments received. These segments are to only inform that the destination system got a new (updated) window size. The typical scenario is that the destination system ran out the receive window and later informs that it is updated.

packets received after close
Number of received segment came from closed connection. Due to network transmission characteristics, IP datagrams are not guaranteed to arrive in sequence. Therefore, we have the possibility of getting segments coming from a closed connection. It should be rare to have this counter. If the port is reused immediately after close, these stray segments may give damage to the new connection on the same port. This is why we have TIME_WAIT for 2MSL after close.

discarded for bad checksums
Number of received segments that were discarded due to checksum error. If the data link layer has some error detection mechanism,such as CRC, you would almost always see a 0 here.

discarded for bad header offset fields
Number of received segments that were discarded due to incorrect offset field in the TCP header. Since the TCP header can have a variable length option field, it is required to have an offset to show the beginning of the user data.

discarded because packet too short
Number of received segments that were discarded due to short length. If a segment has a shorter length than TCP header, it is counted here.

connection requests
Number of SYN segment (connection request segment) sent out to other system. In this case, your system initiates a TCP connection, and this is called active open.

connection accepts
Number of SYN segment received from other system. It also means that your system sent the SYN-ACK segment to the system. In this case, other system that will be a destination, initiates a TCP connection and your system waits it. This is called passive open.

connections established (including accepts)
Number of established connection as the result of sending or receiving SYN segment. In other words, both the active open and passive open are counted here.

connections closed (including 9 drops)
Number of closed connection.

embryonic connections dropped
Number of failure to establish a connection. TCP needs three handshake procedures (SYN, SYN-ACK and ACK) to establish a connection. Any failure during the handshake is counted here.

segments updated rtt (of 546839 attempts)
Number of received segments which updated estimated Round Trip Time (RTT). TCP always estimates RTT for a sending segment and compares it with actual RTT, that is measured by the arrival of corresponding ACK segment. Based on the comparison, TCP module updates estimated RTT. Usually you should see the comparable value here with the packets received.

retransmit timeouts
Number of retransmission timeout. TCP keeps several timers and this is one of them. When TCP sends a data segment, it starts this timer. If it would have not received the ACK when the timer expires, this counter gets incremented. (Then TCP retransmits the unacknowledged data segment and restarts the timer with longer timeout period.) It′s not good symptom to see large counted value here.

connections dropped by rexmit timeout
Number of dropped connection due to retransmission timeout. TCP retransmits the unacknowledged data segment. If it still cannot receive the ACK, TCP repeats retransmission procedure up to 12 times. If all retransmissions fail due to no ACK, TCP module closes the connection. It′s not good symptom to see large counted value here.

persist timeouts
Number of persist timeouts. When the destination system informs 0 receive window, the system suspends sending data segment and starts persist timer. If it cannot receive window update segment until the timer expires (persist timeout), it sends window probe segment. This procedure is repeated if the destination still has 0 receive window.

keepalive timeout
Number of keepalive timeouts. The keepalive timer is reset and is started whenever the system receives data or ACK segment. If keepalive function is enabled, the system sends keepalive probe segment when keepalive timer expires (keepalive timeout). If the system has long period without any traffic, a probe is sent. This mechanism is to confirm if the connection is still established.

keepalive probes sent
Number of keepalive probes sent out.

connection dropped by keepalive
Number of dropped connection due to keepalive. The keepalive probe are retransmitted up to eight times if the system gets no responses. When all probes fail, the system closes the connection.

There is a bug in this option. Sometime you may see the following incorrect output. That should be ack-only packets, not URG only packets as show below:

```
# netstat -p tcp
tcp:
        871870 packets sent
                546355 data packets (2336335 bytes)
                44 data packets (2218 bytes) retransmitted
                318230 URG only packets
                0 URG only packets
...
```

This problem is due to a bug in the message catalogs and appears if the locale is anything other than C. The work around is simple. Issue the command with LANG=C as follows:

```
# LANG=C netstat -p tcp
tcp:
        871870 packets sent
                546355 data packets (2336335 bytes)
                44 data packets (2218 bytes) retransmitted
                318230 ack-only packets (317007 delayed)
                0 URG only packets
...
```

3.5.2 UDP Datagram Statistics

At the TCP explanation we thought it would be too much, but for UDP, the statistics we can get seem too simple. It means almost nothing, because many data link layers already have CRC and as a result you see 0 in almost all occasions.

Note: It may be too risky to completely rely on the lower layer′s reliability and some data link layers don′t have error detection.

```
# netstat -p udp
udp:
        24107 datagrams received
:
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
        10 dropped due to no socket
        3442 broadcast/multicast datagrams dropped due to no socket
        0 socket buffer overflows
        20655 delivered
        671 datagrams output
#
```

As you see above, even you can not know how many UDP datagrams were sent and received.

datagrams received
Number of UDP datagrams received.

incomplete headers
Number of received datagram that was damaged.

bad data length fields
Number of received datagram that had length mismatch between actual datagram and length field in the UDP header.

bad checksums
Number of received datagram that had incorrect checksum. If sending system had incorrect checksum (BSD 4.2 had incorrect checksum algorithm), you would see this counter incremented and the datagram is discarded. Some UNIX implementations have option to disable UDP. (AIX V3.2 didn′t provide such capability.) Disabling the checksum may also improve performance.

Note: For UDP datagrams used for NFS, you can disable UDP checksum. Use the nfso command.

dropped due to no socket
Number of received UDP datagrams of that destination socket ports were not opened. As a result, the ICMP Destination Unreachable - Port Unreachable message must have been sent out. But if the received UDP datagrams were broadcast datagrams, ICMP errors are not generated. See the next counter.

broadcast/multicast datagrams dropped due to no socket
Number of received broadcast/multicast UDP datagrams that didn′t have application daemons listening to the destination port (and they were discarded). In other words, the ports were not opened. Since broadcast have to be received by all systems, usually you see a large countered number here.

socket buffer overflows
Number of socket receive buffer overflows. Since UDP doesn′t have flow control mechanism, this could happen especially if the system is under heavy load.

delivered
Number of received UDP datagrams which had destination application daemon on their ports, and delivered to the applications successfully.

datagrams output
Number of UDP datagrams sent out.

The UDP doesn′t have a concept of connection or acknowledgement, there are no ways to know if datagrams are lost during transmission (this is same with the IP). UDP applications should provide acknowledgement and retransmission. For example, NFS does these.

Difference between V4.1 and V3.2

The netstat -p udp command of V3.2 gave us the following information:

```
# netstat -p udp
udp:
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
        0 socket buffer overflows
#
```

3.6 Application (nfsstat Command for NFS)

Some applications have their own statistics counters. It would be a good practice to implement such counters when you write a program. In this section, we briefly describe the NFS as an example. Of course, you have to know the internal design of the application that you watch if you are to interpret the counters completely.

NFS has a tool, nfsstat, that shows you the counters that are managed by NFS. This command has some flags as listed below:

-c	Only NFS client function statistics are displayed.
-s	Only NFS server function statistics are displayed.
-r	Only RPC statistics are displayed. This option only covers NFS related RPCs. Any other ONC/RPC activities, such as NIS, are not included.
-n	Only NFS procedure statistics are displayed.
-z	Clear all counters. Only used by the privileged user (root).

If you omit all flags, all flags (they are not exclusive) are assumed (-c, -s, -r and -n). Reboot clears all counters. Refer to the manual or InfoExplorer for details. With this command outputs, you can know the statistics of RPC and each NFS procedure. In order to understand these counters, you must have fine working knowledge of ONC/RPC and NFS.

3.6.1 Client Statistics Example

Here is an example of NFS client statistics:

```
# nfsstat -c

Client rpc:
calls      badcalls   retrans    badxid     timeout    wait       newcred
669652     3          11         0          14         0          0

Client nfs:
calls      badcalls   nclget     nclsleep
523296     3          523296     0
null       getattr    setattr    root       lookup     readlink   read
0  0%      72428 13%  0  0%      0  0%      222136 42% 0  0%      10518  2%
wrcache    write      create     remove     rename     link       symlink
0  0%      0  0%      0  0%      0  0%      0  0%      0  0%      0  0%
mkdir      rmdir      readdir    fsstat
0  0%      0  0%      149028 28% 69186 13%
#
```

3.6.2 Server Statistics Example

Here is an example of NFS server statistics:

```
# nfsstat -s

Server rpc:
calls      badcalls   nullrecv   badlen     xdrcall
633        0          0          0          0

Server nfs:
calls      badcalls
457        0
null       getattr    setattr    root       lookup     readlink   read
0  0%      113 24%    30  6%     0  0%      200 43%    0  0%      50 10%
wrcache    write      create     remove     rename     link       symlink
0  0%      0  0%      0  0%      0  0%      0  0%      0  0%      0  0%
mkdir      rmdir      readdir    fsstat
0  0%      0  0%      43  9%     21  4%
#
```

3.7 Error Log

For RS/6000, you should use the system error log as a substantial source of information. Some problems may leave their signs only in the error log. If you can not figure out the possible causes of your problem, looking at the system error log is not a bad idea.

3.7.1 Error Log Example

You can review the error log summary list with the errpt command as shown below. Refer to the manual or InfoExplorer for details of each column.

```
# errpt
ERROR_ID TIMESTAMP  T CL RESOURCE_NAME  ERROR_DESCRIPTION
A386E435 0701214094 P H  ent0           ADAPTER ERROR
ABB81CD5 0701214094 T H  ent0           COMMUNICATION PROTOCOL ERROR
ABB81CD5 0701214094 T H  ent0           COMMUNICATION PROTOCOL ERROR
ABB81CD5 0701214094 T H  ent0           COMMUNICATION PROTOCOL ERROR
A386E435 0701214094 P H  ent0           ADAPTER ERROR
ABB81CD5 0701214094 T H  ent0           COMMUNICATION PROTOCOL ERROR
ABB81CD5 0701214094 T H  ent0           COMMUNICATION PROTOCOL ERROR
ABB81CD5 0701214094 T H  ent0           COMMUNICATION PROTOCOL ERROR
A386E435 0701214094 P H  ent0           ADAPTER ERROR
ABB81CD5 0701214094 T H  ent0           COMMUNICATION PROTOCOL ERROR
...
```

In the above example, you are now sure that your Ethernet adapter (device name is ent0) got a lot of problem. All of the timestamps are concentrated on the same time. For each error details, you can use the errpt -a command as below. Refer to the manual or InfoExplorer for the explanation of each field:

```
# errpt -a -j ABB81CD5
---------------------------------------------------------------------------
ERROR LABEL:    ENT_ERR2
ERROR ID:       ABB81CD5

Date/Time:       Tue Aug  2 18:09:11
Sequence Number: 207483
Machine Id:      000000233400
Node Id:         inoki
Error Class:     H
Error Type:      TEMP
Resource Name:   ent0
Resource Class:  adapter
Resource Type:   ethernet
Location:        00-06
VPD:

Error Description
COMMUNICATION PROTOCOL ERROR

Probable Causes
CSMA/CD ADAPTER
CSMA/CD LAN CABLES
LOCAL CSMA/CD ADAPTER CABLE
CABLE TERMINATOR

Failure Causes
LOCAL CSMA/CD ADAPTER
REMOTE CSMA/CD ADAPTER
CSMA/CD LAN CABLES
LOCAL CSMA/CD ADAPTER CABLE

        Recommended Actions
        PERFORM PROBLEM DETERMINATION PROCEDURES
        CHECK CABLE AND ITS CONNECTIONS

Detail Data
SENSE DATA
0000 0087 0000 0000 0000 0000 0000 0260 8C2C 83BC 0260 8C2C 83BC 3030 3134 0004
0000 0000 0000 0000 0000 0000 0000 0005 000C 000E
...
```

It is obvious that a hardware problem happened on the Ethernet adapter ent0 with this report. This is a hardware error because of the Error Class: H. The error is temporary because of the Error Type: TEMP. The SENSE DATA is usually used by diagnostic program.

Our Experience

The InfoExplorer is a very powerful tool to refer to a meaning of each ERROR LABEL. If you need to retrieve the InfoExplorer, care must be taken for a search word. If you want to specify ENT_ERR2, do not use underscore _. For example, the following search words are fine:

″ENT ERR2″
″ENT″ and ″ERR2″ in Compound Search

But never use ENT_ERR2 or you will get error No matches found. Underscore is not written in the InfoExplorer.

3.7.2 Some Hints and Tips

For almost all occasions the error log just tells the possibility and you needs further problem determination procedure. If the error was due to software, further investigation would be software dependent. If the error was due to hardware, you can proceed with a more direct method.

3.7.2.1 Replace Doubtful Hardware

If the content of the error log suggests that there could be faulty hardware, it is a good approach to replace the suspected units one by one. We believe you can isolate the problem very easily. The following is the error log message of a typical Ethernet error, ENT_ERR2:

```
Failure Causes
LOCAL CSMA/CD ADAPTER
REMOTE CSMA/CD ADAPTER
CSMA/CD LAN CABLES
LOCAL CSMA/CD ADAPTER CABLE

        Recommended Actions
        PERFORM PROBLEM DETERMINATION PROCEDURES
        CHECK CABLE AND ITS CONNECTIONS
```

This message is just saying that all hardware units involved in the communication are potential suspects. Unfortunately the message doesn′t tell the exact cause. (But, it is far better than no information at all.)

Note: Notice a collision of Ethernet can also cause this error. If it is the case, replacement will not resolve the problem. If you doubt the possibility of collision, make a cross check using the netstat -v command.

3.7.2.2 Adding Your Message in Error Log

You can write an arbitrary message in the system error log. It is convenient when you try to recreate a problem, because your own message can be used as a separator of log events. It′s totally up to you how to use this feature. Use the errlogger command as below.

```
# errlogger ″I′ll debug the SRC!″
#
```

Note: Remember that the syslogd has the similar command logger to write your arbitrary message in the syslog′s log.

In the error log summary list, it looks as below:

```
# errpt
IDENTIFIER TIMESTAMP  T C RESOURCE_NAME  DESCRIPTION
AA8AB241   0818181995 T O OPERATOR       OPERATOR NOTIFICATION
0873CF9F   0818152295 T S pts/0          TTYHOG OVER-RUN
0873CF9F   0818152295 T S pts/0          TTYHOG OVER-RUN
...
```

The message is logged as below:

```
# errpt -a -j AA8AB241
---------------------------------------------------------------------------
LABEL:          OPMSG
IDENTIFIER:     AA8AB241

Date/Time:       Fri Aug 18 18:19:26
Sequence Number: 1724
Machine Id:      000970044D00
Node Id:         zero
Class:           O
Type:            TEMP
Resource Name:   OPERATOR

Description
OPERATOR NOTIFICATION

User Causes
ERRLOGGER COMMAND

        Recommended Actions
        REVIEW DETAILED DATA

Detail Data
MESSAGE FROM ERRLOGGER COMMAND
I'll debug the SRC!
#
```

Chapter 4. System Parameter Tuning

In this chapter, we describe some AIX system features that have a crucial role in network performance. Some features are very AIX-unique and some are common among UNIX-unique. The parameters explained here are deeply related to TCP/IP, but they are not a part of TCP/IP. Their adjustment procedures depend on each product and implementation, so you may not find the equivalent parameters on other vendor systems.

For details about topics in this chapter, we strongly recommend that you refer to the manual *Performance Monitoring and Tuning Guide*, SC23-2365.

Difference between V4.1 and V3.2

Substantial enhancements and improvements have been made to V4.1. Some features described in this chapter were removed from V4.1 and they reduce the system administrator′s daily effort. The following features are V3.2 only:

- RDTO (Receive Data Transfer Offset)
- Trailer Protocol

4.1 mbuf Tuning

Currently we have two predominate memory allocation mechanisms for the UNIX networking function. One is the mbuf that has been introduced with 4.2/4.3 BSD. The other is STREAMS that has been introduced with System V R3. AIX V3.2 was completely BSD compliant and followed the mbuf scheme (this is only about TCP/IP). AIX V4.1 uses a STREAMS based memory allocation mechanism, but it also uses the mbuf scheme which is built on the STREAMS. Any commercial books written about the BSD networking function would help you to understand the mbuf of V3.2. We recommend that you read the following book:

The Design and Implementation of the 4.3BSD UNIX Operating System

Written by Samuel J. Leffer, Marshall Kirk McKusiick, Michael J. Karels and John S. Quarterman.
Published by Addison-Wesley Publishing Company, Inc.

The feature mbuf is common among BSD-based UNIXs, and some routers that have BSD based or derivative networking implementation. The mbuf is the memory allocated to the networking function. Although the basic schemes are widely used, the actual adjustment procedures and monitoring commands depend on each implementation. We can use the no command to configure mbuf dynamically (without rebooting). This is one of the AIX advantages.

Difference between V4.1 and V3.2

The AIX V4.1 and V3.2 both support STREAMS. But, the underlaying mechanisms are different between V4.1 and V3.2. V3.2 supports the STREAMS at the API level and it is built on the top of the mbuf structure. On V4.1, STREAMS is implemented natively on the top of the Common Data Link Interface (CDLI). CDLI supports both STREAMS and socket-based protocols.

4.1.1 mbuf Basics

The mbuf or memory buffer is a data structure that holds actual data or offsets it to the memory page, which holds actual data. In the original BSD implementation, the size of a mbuf was 128 bytes and the size of a memory page was 1024 bytes. Thus, one mbuf could represent up to 1024 bytes of data, since more than one mbuf is linked to constructing a chain and the chain can hold an arbitrary length of data. In our AIX V3.2 implementation, the mbuf is 256 bytes and the memory page is 4096 bytes; our one mbuf can represent up to 4096 bytes. (Due to the control information, a mbuf can hold up to 236 bytes of user data.) The mbuf and the memory page referred to by the mbuf are, together, called an mbuf-cluster.

Note: In AIX V3.1, the mbuf was 128 bytes.

If user data is less than 433 bytes, AIX allocates two mbufs to hold the data directly in mbufs. If the data is equal to or greater than 433 bytes, AIX allocates one memory page for the data and puts the offset in a mbuf (in other words allocates a mbuf-cluster). Although we don′t get into the detailed mechanism, mbuf′s flexible memory allocation scheme is used to not only store the data but also to store some control information needed by AIX (UNIX). For example, the socket structure, routing table, PCB (Protocol Control Block) and other important information are stored in mbufs. The mbufs are used to hold and pass data from socket to device driver queue and vice versa. Actually, data is not copied at each layer. In order to avoid overhead, just the pointer of the mbuf chain is passed from one layer to another, and if necessary, protocol header is added by chaining a new mbuf containing the header information.

Mbuf and memory page are dynamically allocated from the mbuf pool as needed. The system holds a memory region called a mbuf pool. Both mbuf and memory pages pointed by mbuf-cluster are allocated from this pool. Allocation is done by the kernel process, the netm, and it runs at a very high priority (37, fixed). Therefore the pool is dynamically expanded or resized from time to time. Unnecessary memory pages used for mbuf-clusers will be released, in order to allow some activities, other than the networking functions, to use them. But mbufs are not released after they are allocated in the pool. The only way to reduce the allocated mbufs is to reboot the system. Because a mbuf is 256 bytes, this should not be a problem.

The summary of the AIX mbuf features are:

- The mbuf and mbuf-cluster are used to pass packets from the adapter device drivers to a network application and vice versa.
- The mbuf and mbuf-cluster are not only used by TCP/IP but also by SNA and other network protocols.
- The mbuf and the mbuf-cluster are allocated in the dedicated memory region called a mbuf pool. This pool is only used for the networking functions and the default size is 2 MB.
- The mbuf pool is not a subject of paging. They are pinned in the memory.

4.1.2 Getting the Current Status (V3.2 Only)

This section is mainly for V3.2 and other BSD derived systems.

You can review the current mbuf usage with the nestat -m command. This output may be common between UNIX which has a mbuf based implementation. If a system is STREAMS based UNIX, although it supports netstat -m, the output is totally different. See the following example:

```
# netstat -m
308 mbufs in use:
        59 mbufs allocated to data
        11 mbufs allocated to packet headers
        90 mbufs allocated to socket structures
        126 mbufs allocated to protocol control blocks
        9 mbufs allocated to routing table entries
        10 mbufs allocated to socket names and addresses
        3 mbufs allocated to interface addresses
55/139 mapped pages in use
633 Kbytes allocated to network  (46% in use)
0 requests for memory denied
0 requests for memory delayed
0 calls to protocol drain routines
#
```

mbufs in use
Number of mbufs which hold actual data. Notice mbuf is used for many purposes. In this example, only 59 of 308 mubfs are used to store data.

Mapped pages in use
Number of memory pages allocated for mbuf-clusters. This shows both allocated (but not used) and currently used memory pages.

KB allocated to network (46% in use)
Amount of total memory size allocated for networking function (mbuf pool). Also, the ratio of currently used memory is shown.

Requests for memory denied
Number of denied requests to allocate mbuf or memory page (mbuf-cluster) due to resource shortage. For a receiving operation, the device driver of the adapter can not wait for the memory allocation if there are no available mbufs or memory pages. Then, the memory allocation returns immediately with an error. As a result, the receiving data (packet) is lost and it is counted here.

Requests for memory delayed
Number of delayed mbuf or memory page allocation. For a sending operation, the device driver of the adapter waits for the memory allocation if there are no available mbufs or memory pages. Notice that mbuf allocation is a subject to be scheduled with certain priority. In this case, if the sending operation is suspended for a while, it is counted here.

Calls to protocol drain routines
Number of protocol drain routines called. When there are no available mbufs or memory pages to allocate, some mbufs or memory pages must be released (drained). The system calls the protocol drain routine ip_drain(), discarding data of a lesser priority (the IP datagram fragments) in the IP reassembly queue. Notice that the drain activity is caused only by the sending operation.

You can confirm the contents of the previous information. In this example, 308 mbufs and 139 mbuf-clusters (memory pages) are allocated.

Total allocated memory size is:

```
308 × 256 + 4096 × 139 = 648,192 bytes
648,192 ÷ 1024 = 633 KB
```

The actually used memory region is:

```
308 × 256 + 4096 × 55 = 304,128 bytes
```

Therefore,

```
304,128 ÷  648,192 = 0.469 ≈  46 %
```

Note: We used 236 bytes for mbuf instead of 256 bytes because 20 bytes are always used for control information.

It gives you a consistent result with the output of 633 KB allocated to the network (46% in use). Notice for mbufs, 59 of 308 are used for user data and other mbufs are for control information. Also, notice that the mbuf pool can be expanded to 2048 KB as necessary (this is default configuration), and we have enough margin from 633 KB.

4.1.3 Getting the Current Status (V4.1)

With AIX V4.1, you will see a quite different output for the netstat -m command. This change reflects the difference of the internal memory allocation scheme between V3.2 and V4.1.

```
# netstat -m
34 mbufs in use:
21 mbuf cluster pages in use
92 Kbytes allocated to mbufs
0 requests for mbufs denied
0 calls to protocol drain routines

Kernel malloc statistics:

By size       inuse     calls failed    free   hiwat   freed
32              677      4239      0      91     640       0
64               51       147      0      13     320       0
128             160     15498      0      32     160       0
256             157     53774      0      19     384       0
512              64     12948      0      24      40       0
1024              2      1386      0       2      20       0
2048              0     10286      0       6      10       0
4096             23       611      0       5     120       0
8192              0        51      0       1      10       0
16384             1        33      0      24      24       7

By type       inuse     calls failed  memuse  memmax  mapb
mbuf             34     49892      0    8704   11520     0
mcluster         21     13578      0   86016   99840     0
socket          169       936      0   22208   22784     0
pcb             101       502      0   15488   16000     0
routetbl          9        35      0    1568    1952     0
fragtbl           0      2821      0       0      32     0
ifaddr            6        13      0     704     704     0
mblk             48     14997      0    7936    9344     0
mblkdata         10     14959      0    5120   17408     0
strhead          18        28      0    4896    4896     0
```

```
strqueue        41         68       0   20992   20992     0
strmodsw        17         17       0    1088    1088     0
strsyncq        46        127       0    5408    5408     0
streams        606        979       0   30144   30144     0
kernel table     2          2       0   40960   40960     0
temp            12         48       0    5632  524288     0
#
```

The meanings of the mbuf statistics are the same with the V3.2 output. The kernel malloc statistics are new for you. With V4.1, the memory allocation scheme uses STREAMS, and it uses data block (mblk). Data block has many sizes ranging from 32 bytes to 16384 bytes. A 256 byte data block is used for mbuf and a 4096 data block is used for mbuf-cluster. The most crucial counter has failed and this means that the allocation of the data block was rejected due to no available data block.

4.1.4 Problem Symptom (V3.2 Only)

As we know, mbufs and mbuf-clusters play a very crucial role in communication. Their shortage impacts network performance dramatically. Some typical symptoms are explained following. It doesn′t necessary mean that all the symptoms are always observed.

Very Slow Network Response

Users suffer very slow network response. It almost looks like a hang state in an extreme situation.

No Mbuf Errors/No Mbuf Extension Errors

You see some values are counted at the counter No Mbuf Errors and/or No Mbuf Extension Errors of the netstat -v command:

```
# netstat -v

TOKEN STATISTICS (tr0) :

Hardware Address: 10:00:5a:a8:46:2d
Transmit Byte Count: 1091048648 Receive Byte Count: 1096451367
Transmit Frame Count: 1084670976       Receive Frame Count: 1089135552
...
Receive Packets Lost: 0         No Mbuf Errors: 0
No Mbuf Extension Errors: 54    Receive Int Count: 56858
Transmit Int Count: 2914               Packets Rejected No NetID: 1790
Packets Accepted Valid NetID: 55070    Overflow Packets Received: 0
Packets Transmitted and Adapter Errors Detected: 0

#
```

No Mbuf Errors mean that the allocation of mbuf has failed. No Mbuf Extension Errors mean that allocation of mbuf-cluster has failed.

Requests for Memory Denied Error

You see some value is counted at the counter, which is the number of requests for memory denied by the netstat -m command.

```
# netstat -m
1126 mbufs in use:
        846 mbufs allocated to data
...
        2 mbufs allocated to interface addresses
426/438 mapped pages in use
2033 KB allocated to network  (97% in use)
108 requests for memory denied
0 requests for memory delayed
0 calls to protocol drain routines
#
```

RESOURCE UNAVAILABLE Error

You see the RESOURCE UNAVAILABLE error in the system error log. Issue the errpt command to review the error log.

4.1.5 Configure mbuf with the no Command

In order to avoid mbuf shortage, you can adjust some parameters by the no command. Be careful about mbuf tuning. Because mbuf pools are dedicated memory region and if considerable amount of memory is used for mbuf, any other system activities may be impacted. Also, the mbuf is managed by the kernel process netm, and the improperly configured mbuf pool can cause thrashing due to conflicting network traffic and mbuf pool thresholds. You should not change any mbuf parameters until you understand what you are trying to do.

```
# no -a
                dog_ticks = 60
                 lowclust = 21
                  lowmbuf = 188
                  thewall = 2048
              mb_cl_hiwat = 42
                compat_43 = 1
                   sb_max = 65536
             detach_route = 1
...
#
```

lowclust

This defines the lower limit of unused memory pages. If the number of unused memory pages gets lower than this value, the netm is scheduled to allocate more memory pages. Since the mbufs and memory pages are not increased immediately, you may need some margin.

lowmbuf

This defines the lower limit of unused mbufs. If a number of unused mbufs gets lower than this value, the netm is scheduled to allocate more mbufs. Since mbufs are not increased immediately, you may need some margin. The arriving packet number per second, or twice that number (especially for UDP) may be a reasonable value. Since a mbuf-cluster must use one mbuf to store pointer information whenever the lowclust is increased, at least the same amount of the lowmbuf should be increased.

thewall

This defines the upper limit of the mbuf pool. Consider the output of netstat -m (KB allocated to network) when you want to expand. If 80 %

of the current thewall value has been used constantly, it is reasonable to expand. The default is 2048 KB.

mb_cl_hiwat
This defines the upper limit of used memory pages. If the number of used memory pages gets higher than this value, the netm is scheduled to release unused memory pages. If you choose a value close to the lowclust, the netm is busily occupied by increasing and decreasing memory pages. As a result, the system suffers performance. It always should be larger than twice the lowclust.

Difference between V4.1 and V3.2

The mbuf adjustment work is made automatically at the AIX V4.1. You have few things to do with V4.1. All mbuf related parameters of the no command, except thewall, are obsolete. In other words, thewall is the *only* tuneable parameter for mbuf.

If you need to change any parameter, issue the no command as follows. This example changes the current value of mb_cl_hiwat from 98 to 60.

1. Checking the current value:

```
# no -o mb_cl_hiwat
mb_cl_hiwat = 98
#
```

2. Changing it to 60:

```
# no -o mb_cl_hiwat=60
#
```

3. Confirming if the change was made:

```
# no -o mb_cl_hiwat
mb_cl_hiwat = 60
#
```

If you need to resume the default value, use the -d flag:

```
# no -d mb_cl_hiwat
#
```

If your system is a router, you should monitor the mbuf usage constantly by using netstat -m, because mbuf is used in the IP datagram routing. Even when you are not using any network applications, passthrough traffic consumes your mbuf pool.

When you configure to expand the mbuf pool, the memory other than the mbuf pool may already be used by other applications or the system. Although the parameters are updated instantaneously, it is not guaranteed that the mbuf pool gets expanded immediately. You may need to reboot the system in order to make room for the mbuf pool, or the mbuf pool expansion should be made just after system boot.

4.1.6 Making the Update Permanent

Since the no command updates parameters in the kernel, rebooting the system clears all your updates, and they return to the default values. If you want to make the updated configuration permanent, you should write the no command in the startup script /etc/rc.net. The best place is the end of the script as follows:

```
##################################################
# The socket default buffer size (initial advertised TCP window) is being
# set to a default value of 16k (16384). This improves the performance
# for Ethernet and token-ring networks.  Networks with lower bandwidth
# such as SLIP (Serial Line Internet Protocol) and X.25 or higher bandwidth
# such as Serial Optical Link and FDDI would have a different optimum
# buffer size.
# ( OPTIMUM WINDOW = Bandwidth * Round Trip Time )
##################################################
if [ -f /usr/sbin/no ] ; then
        /usr/sbin/no -o tcp_sendspace=16384
        /usr/sbin/no -o tcp_recvspace=16384
        /usr/sbin/no -o thewall=3072
        /usr/sbin/no -o mb_cl_hiwat=60
fi
```

Although the parameters updated by the no command, are not saved for reboot, there is one exception. The thewall is also stored in the ODM as the attribute maxmbuf of the device sys0. You can see the current value with the command lsattr -E -l sys0 as follows:

```
# lsattr -E -l sys0
keylock     normal State of system keylock at boot time              False
maxbuf      20     Maximum number of pages in block I/O BUFFER CACHE True
maxmbuf     2048   Maximum KB of real memory allowed for MBUFS   True
maxuproc    100    Maximum number of PROCESSES allowed per user      True
autorestart false  Automatically REBOOT system after a crash         True
iostat      true   Continuously maintain DISK I/O history            True
realmem     65536  Amount of usable physical memory in KB        False
conslogin   enable System Console Login                              False
maxpout     0      HIGH water mark for pending write I/Os per file   True
minpout     0      LOW water mark for pending write I/Os per file    True
fullcore    false  Enable full CORE dump                             True
#
```

You can update the ODM with the chdev command as follows. This example expands the default from 2048 KB to 3072 KB:

1. Check from the current value:

   ```
   # lsattr -E -l sys0 -a maxmbuf
   maxmbuf 2048 Maximum KB of real memory allowed for MBUFS True
   #
   ```

2. Change the value using the chdev command:

   ```
   # chdev -l sys0 -a maxmbuf=3072
   sys0 changed
   #
   ```

3. Check that the value was updated correctly:

   ```
   # lsattr -E -l sys0 -a maxmbuf
   maxmbuf 3072 Maximum KB of real memory allowed for MBUFS True
   #
   ```

Note: What happens if you used both the ODM and the srcipt /etc/rc.net to configure thewall? During the boot process, first the data in the ODM is loaded, then the script is executed. Therefore, /etc/rc.net overrides the ODM.

4.2 RDTO and Trailer Protocol (V3.2 Only)

This section only applies to V3.2. Both RDTO and trailer protocol were removed at V4.1 due to the improved self-tuning capability.

In this section, we mention mechanisms to optimize the packet treatment in the memory. As we have already seen, our AIX V3.2 uses the mbuf to store the data (packet). When we send data, the application program writes the data into a socket send buffer that is built by mbufs and/or mbuf-clusters (it is also called the mbuf chain). Although the data should be passed from the socket layer to the device driver through TCP/UDP and IP layers, the data does not need to be copied among the layers. Just the pointer to the mbuf chain is passed. At each layer, a mbuf that has the corresponding protocol header is linked at the head of the mbuf chain. With this mechanism, we can avoid the overhead caused by copying the entire data or packet.

For the receiving operation, the matter is more complicated than the sending operation. There is more than one data link protocol (for example, Ethernet, token-ring). Also, transport protocol has more than one (for example, UDP and TCP). The length of each protocol header is different. If we place a received frame in memory, we cannot expect that user data will always begins at a fixed location in memory (frame). Usually searching a memory page to locate user data is not acceptable due to performance reason. If we can assume that user data is always located at the beginning (or at fixed place) of memory page, we can avoid this problem. When a frame is passed from the lower layer to upper layer, the protocol header of the lower layer must be removed. At that moment, if the system tries to keep the packet aligned with the boundary of a memory page (or mbuf), the entire packet should be copied from a memory page to another, in order to compensate for the blank space to the removal of the protocol header. This mechanism adds a certain performance penalty.

There are two mechanisms to avoid the above overhead. One is Trailer Protocol and the other is Receive Data Transfer Offset (RDTO). Their purposes are slightly different from each other.

4.2.1 Receive Data Transfer Offset (RDTO) Basics

The RDTO is a quite unique AIX feature to improve the performance regarding the memory operation. When a received packet is copied from an adapter card to mbuf (or memory page), the additional blank space is allocated at the beginning of mbuf, and then the packet (frame) is placed. This blank space is a variable length and the length depends on the data link protocol used by the packet (frame). The length is chosen to make the total of the blank space and the data link header length a fixed length (40 bytes). This blank space is called the Receive Data Transfer Offset (RDTO). The size of RDTO for each data link protocol is listed here:

Ethernet	26
IEEE802.3	18

Token-ring	0
X.25	12

Note: These values are related to TCP/IP. For SNA, 92 is the optimum value for all data links.

The advantage of RDTO is obvious. The first is that the system can always expect that the beginning of the user data (notice it includes the IP header, but excludes data link header) is always the 40th byte in mbuf or memory page.

The second advantage is to improve the routing function of the network protocol (IP). If an incoming frame should be routed to another destination, and if the data link protocols are different between the receiving network and the sending network, the data link header must be replaced. If the receiving network is Ethernet and the sending network is token-ring, there can only be 14 bytes of blank space after removing the Ethernet header. But, the token-ring header needs 40 bytes. Without RDTO, an entire frame must be copied to another mbuf (or memory page) to make room for the token-ring header, and this causes performance penalty. The longest supported data link protocol header is token-ring. This is why RDTO is chosen to make the total length of RDTO and data link header 40 bytes.

There are more points you should know. If you don′t set the RDTO as previously listed, you may suffer some slower performance, but it would never affect the connectivity. Inappropriate RDTO (default is 92 bytes) causes additional copy activity of frames when the system fails to locate the beginning of the IP datagrams. But that′s all you suffer. You should not have any other problems. Also, RDTO doesn′t need to match between source and destination systems (notice that RDTO is a RS/6000 unique feature).

For AIX V4.1 users

The RDTO was removed from V4.1. All the device drivers that run on the V4.1 don′t need or use the RDTO (they are smarter than the prior version).

4.2.2 RDTO Configuration

You can review the current RDTO value with the lsattr command as follows. Since RDTO is not a UNIX feature, only the high-level commands and SMIT are available to review and update:

```
# lsattr -E -l tok0
intr_priority 3       Interrupt priority                False
xmt_que_size  30      TRANSMIT queue size               True
rec_que_size  30      RECEIVE queue size                True
sta_que_size  10      STATUS BLOCK queue size           True
rdto          92       Receive data transfer OFFSET      True
bus_io_addr   0x86a0  Bus I/O address                   False
dma_lvl       0x5     DMA arbitration level             False
...
```

4.2.2.1 Using High-level Command

You can use the chdev command to update the RDTO as follows:

1. First detach the interface. Since the RDTO is an attribute of the adapter device, the interface must be detached before you configure it:

   ```
   # ifconfig tr0 detach
   #
   ```

2. Use the chdev command as follows. You can confirm that the change was correctly made with the lsattr command:

   ```
   # chdev -l tok0 -a rdto=0
   tok0 changed
   # lsattr -E -l tok0 -a rdto
   rdto 0 Receive data transfer OFFSET True
   #
   ```

3. Make the interface up again with the ifconfig command:

   ```
   # ifconfig tr0 up
   #
   ```

If you forget to detach the interface, you will get following error when you issue the chdev command:

```
# chdev -l tok0 -a rdto=0
Method error (/etc/methods/chgtok):
        0514-062 Cannot perform the requested function because the
                 specified device is busy.

#
```

4.2.2.2 Using SMIT

A SMIT screen is also available to configure the RDTO. The RDTO is an attribute of the adapter device, so you can see the RDTO at an adapter configuration screen.

1. First, detach the interface. Since the RDTO is an attribute of an adapter device, the interface must be detached before you configure it. The following is one example. You can use the chdev command instead.

   ```
   # ifconfig tr0 detach
   #
   ```

2. Invoke SMIT. If the adapter is token-ring, the following fast path is available:

   ```
   # smitty chgtok
   ```

3. Through the submenu, you should get the following screen. Then, update the entry field named Receive data transfer OFFSE:

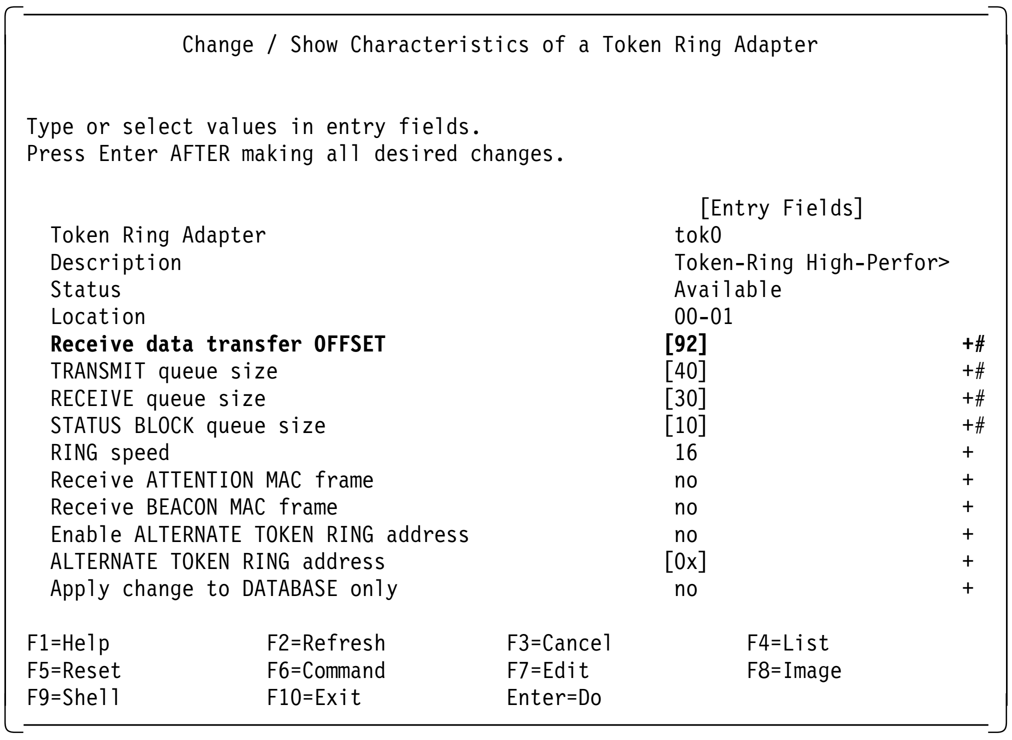

```
                Change / Show Characteristics of a Token Ring Adapter


 Type or select values in entry fields.
 Press Enter AFTER making all desired changes.

                                                          [Entry Fields]
   Token Ring Adapter                                   tok0
   Description                                          Token-Ring High-Perfor>
   Status                                               Available
   Location                                             00-01
   Receive data transfer OFFSET                        [92]                    +#
   TRANSMIT queue size                                 [40]                    +#
   RECEIVE queue size                                  [30]                    +#
   STATUS BLOCK queue size                             [10]                    +#
   RING speed                                           16                     +
   Receive ATTENTION MAC frame                          no                     +
   Receive BEACON MAC frame                             no                     +
   Enable ALTERNATE TOKEN RING address                  no                     +
   ALTERNATE TOKEN RING address                        [0x]                    +
   Apply change to DATABASE only                        no                     +

 F1=Help             F2=Refresh          F3=Cancel           F4=List
 F5=Reset            F6=Command          F7=Edit             F8=Image
 F9=Shell            F10=Exit            Enter=Do
```

Figure 28. SMIT Network Adapter Configuration Screen (Token-Ring)

4. Bring the interface up again with the ifconfig command. The following is one example. You can use the chdev command instead.

   ```
   # ifconfig tr0 up
   #
   ```

4.2.3 Trailer Encapsulation Protocol Basics

The Trailer Protocol was developed and introduced at 4.2BSD. Although this is a very interesting mechanism, it is now considered to have become almost an obsolete function.

In this scheme, the IP header and higher layer header (TCP or UDP) are placed at the end of the packet (frame). With this technique, the user data is located immediately after the data link header. When a packet is received, the system can place user data at the boundary of the memory page or mbuf (the data link header is placed in a separate mbuf and linked). After that, the system just passes the pointer among layers avoiding copying an entire packet from memory to memory because the higher layers protocol headers are located at the end of user data. Removal of a protocol header never impacts the location of the IP datagram.

All systems must support the trailer protocol because it uses a special format of packet; IP and TCP/UDP headers must be at the end of packet. Also, this scheme is only good to those systems that have the same memory management mechanism (such as mbuf). Systems that don′t have virtual memory do not get the benefit from the trailer protocol because they don′t have pages to align user data. Due to these reasons, this method has not been widely used and few people intend to use it now.

4.2.4 Trailer Protocol Configuration

To use the trailer protocol, first both systems (destination and source) must agree to enable this protocol. You can review the current configuration status with the lsattr command. As you know, you are reviewing the ODM.

```
# lsattr -E -l tr0
mtu       1492         Maximum IP Packet Size for This Device     True
mtu_4     1492         Maximum IP Packet Size for This Device     True
remmtu    576          Maximum IP Packet Size for REMOTE Networks True
netaddr   9.170.5.45   Internet Address                           True
state     up           Current Interface Status                   True
trailers  off          TRAILER Link-Level Encapsulation           True
arp       on           Address Resolution Protocol (ARP)          True
allcast   on           Confine Broadcast to Local Token-Ring      True
...
```

To see the current configuration in the kernel, you should use the ifconfig command, as follows:

```
# ifconfig tr0
tr0: flags=8063<UP,BROADCAST,NOTRAILERS,RUNNING,ALLCAST>
        inet 9.170.5.45 netmask 0xffffff00 broadcast 9.170.5.255
#
```

For the configuration, you can only use the chdev command at AIX V3.2.5. Of course, the chdev command updates both the kernel and the ODM, and the updates are saved for the reboot.

Note: In prior versions the ifconfig command supported the parameter trailers. Now these parameters are obsolete.

```
# chdev -l tr0 -a trailers=on
tr0 changed
#
```

When you make the trailer protocol effective, your system negotiates trailer protocol when the system exchanges the ARP packets by using an extra trailer ARP reply packet. Refer to the *RFC 1122 Requirements for Internet Hosts* for details. If the destination doesn′t respond positively, the trailer protocol doesn′t become effective. By this RFC, the default configuration *must* disable the protocol.

4.3 Transmit/Receive Queue

There are queues between the device driver (adapter card) and memory (mbuf). These queues hold mbuf chains for send and receive. If the queue cannot hold all the incoming or outgoing data, some data is lost. As a result, you may get retransmissions and it impacts the system′s performance. These queues are used by all the communication protocols (not only by TCP/IP but also by SNA).

4.3.1 Queue Basics

There is a transmit queue and a receive queue. The transmit queue is used to store mbuf chains which contain data to be sent. This queue defines the number of frames queued between the interface code (adapter) and the network layer module (IP, SNA). There is one transmit queue for each adapter.

The receive queue is used to store mbuf chains which contain data received by the adapter (and has not been passed to the IP module yet). This queue defines the number of frames queued between the interface code (adapter) and the network layer module (IP, SNA). There is one receive queue for each protocol and adapter.

If a queue length is not long enough, the queue gets overrun and the data is lost. It may invite retransmission. Thus, transmit/receive queue tuning is very important. If your system is heavily loaded with file transfers or some network activities, it is easy to get queues that overrun. NFS server and network installation servers are good candidates for tuning.

For ISA Bus Adapter users

ISA Bus Adapters only have a transmit queue. They don′t have receive queue.

4.3.2 Getting the Current Status

A shortage of queues causes a loss of packets and finally invites the packets′ retransmission by the TCP module or applications (UDP). If this happens, you should have several symptoms. Some typical symptoms are described here.

4.3.2.1 Queue Usage Statistics

Here we show you an example which is suffering transmission queue overrun. At first, you can review the current queues status with the netstat -v command.

```
# netstat -v
-------------------------------------------------------------
TOKEN-RING STATISTICS (tok0) :
Device Type: Token-Ring High-Performance Adapter (8fc8)
Hardware Address: 10:00:050:aa0:8b0:5c1
Elapsed Time: 0 days 4 hours 20 minutes 33 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 3169                                 Packets: 26795
Bytes: 562812                                 Bytes: 3024785
Interrupts: 3126                              Interrupts: 26764
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
Max Packets on S/W Transmit Queue: 60         Bad Packets: 0
S/W Transmit Queue Overflow: 84
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 9                          Broadcast Packets: 21558
Multicast Packets: 264                        Multicast Packets: 264
Timeout Errors: 0                             Receive Congestion Errors: 0
Current SW Transmit Queue Length: 0
Current HW Transmit Queue Length: 0
...
```

The counter, Max Packets on S/W Transmit Queue, shows the maximum usage of the queue at the busiest moment since the system boot. Since the queue usage is checked at certain intervals, these counters may not give you the exact maximum values (they can be smaller than the real maximum). The counter, S/W Transmit Queue Overflow tells you about the queue shortage. The previous example is a bad symptom.

Difference between V4.1 and V3.2

```
# netstat -v

ETHERNET STATISTICS (en0) :

Hardware Address: 08:00:5a:0d:0d:a7
Transmit Byte Count: 187446874.0       Receive Byte Count: 42396168.0
Transmit Frame Count: 220279.0  Receive Frame Count: 255640.0
Transmit Error Count: 0         Receive Error Count: 0
Max Netids in use: 7           Max Transmits queued: 33
Max Receives queued: 0          Max Stat Blks queued: 0
Interrupts lost: 0              WDT Interrupts lost: 0
Timeout Ints lost: 0            Status lost: 0
...
```

In the above example, it is clearly understood that the transmit queue has been heavily loaded. If the queue length had the default of 30, we could get tremendous packet losses and retransmissions (it was expanded to 64 in this example, so we were able to avoid the worst situation). In our experience, if the counter reaches 60 % of the queue length, it is reasonable to assume packet drops (losses). Since the administrator of the system has expanded the queue size to 64, the worst situation was prevented.

Our Experience

If you check the counters, Max Transmits queued and Max Receives queued, of each RS/6000 at your site, you will find a fact that a transmit queue is easier to overflow than a receive queue. Usually the counted value of Max Transmits queued is bigger than that of Max Receives queued in the same system. It′s not rare to see a big value at Max Transmits queued while Max Receives queued stays 0.

In our experience when we sent many packets to the NFS server (we made many write operations), curiously the server got the Max Transmits queued counter incremented, but the Max Receives queued counter didn′t get incremented. If an adapter gets many send and receive operations at the same time (since receive operations has higher priority than send operations) the send operations are queued more. As a result, Max Transmits queued gets incremented. In a heavily loaded system, it has a tendency to be a problem for the sending operation even when the receiving operation is busier.

4.3.2.2 Output Error at Network Interface

You can confirm the packet loss with the netstat -i command. This command gives you supplemental information. The counter, Oerrs, only counts errors detected when the device driver was called to send out packets. Be aware that this doesn′t mean transmit queue overflow. But in our experience, the situations that invites transmit queue overflow also tend to cause this error.

```
# netstat -i
Name  Mtu   Network     Address         Ipkts    Ierrs Opkts    Oerrs Coll
lo0   1536  <Link>                         3935     0     3935     0     0
lo0   1536  127         localhost          3935     0     3935     0     0
tr0   1492  <Link>                       112881     0   208813 36722     0
tr0   1492  9.170.5     inoki5.fscjapan  112881     0   208813 36722     0
tr1   1492  <Link>                        36791     0    54701  3648     0
tr1   1492  9.170.1     inoki.fscjapan.   36791     0    54701  3648     0
#
```

4.3.2.3 Application Error

One more bit of information you may be interested in. If the system is configured as an NFS client or a server, than you should check the following NFS related RPC statistics:

```
# nfsstat -r

Server rpc:
calls      badcalls   nullrecv   badlen     xdrcall
6794       0          0          0          0

Client rpc:
calls      badcalls   retrans    badxid     timeout    wait       newcred
4446       37         149        0          183        0          0
#
```

If packets (RPC) are lost somewhere (at interface or transmit/receive queue), it should be detected by NFS, because NFS client doesn′t get the response. Since UDP is connectionless, only NFS knows the transmission status. In this example, this system has 149 retransmission due to 183 detected timeouts. Again, be aware that this doesn′t mean only transmit queue overflow. It could be any other reason, but queue overflow impacts these counters.

4.3.3 Queue Size Configuration

As you already know, the lsattr command and the chdev command are available for review and update. Of course, they are stored in the ODM, as follows:

```
# lsattr -E -l tok0
intr_priority 3       Interrupt priority                  False
xmt_que_size  40      TRANSMIT queue size                 True
rec_que_size  30      RECEIVE queue size                  True
sta_que_size  10      STATUS BLOCK queue size             True
...
#
```

4.3.3.1 Using High-Level Command

You can use the chdev command to update the queue sizes.

1. First detach the interface. Since the queue sizes are attributes of the adapter device, the interface must be detached before you configure it:

   ```
   # ifconfig tr0 detach
   #
   ```

2. Use the chdev command as follows:

   ```
   # chdev -l tok0 -a rec_que_size=90
   tok0 changed
   #
   ```

3. You can confirm that the change was made correctly usibg the lsattr command:

   ```
   # lsattr -E -l tok0 -a rec_que_size
   rec_que_size 90 RECEIVE queue size True
   #
   ```

4. Make the interface up again with the ifconfig command:

   ```
   # ifconfig tr0 up
   #
   ```

5. Do not forget to configure (load) the routing information:

   ```
   # /usr/lib/methods/cfginet
   zero
   9.68.214.1 net 0: gateway 9.68.214.1
   9.68.214.82 net 9.170.5: gateway 9.68.214.82
   ```

If you forget to detach the interface, you will get the following error when you issue the chdev command:

```
# chdev -l tok0 -a rdto=0
Method error (/etc/methods/chgtok):
        0514-062 Cannot perform the requested function because the
                 specified device is busy.

#
```

4.3.3.2 Using SMIT

SMIT screen is also available to configure the queue sizes. The queue sizes are attributes of the adapter device, so you can see the queue sizes at the adapter configuration screen.

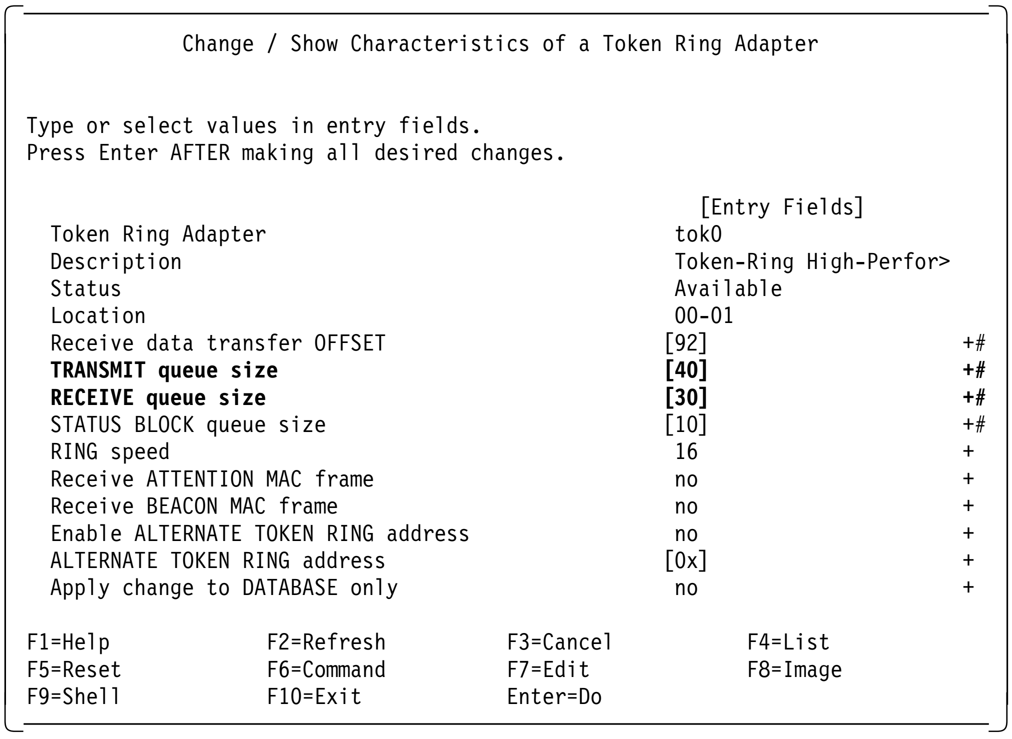
1. First, detach the interface. Since the queue sizes are attributes of the adapter device, the interface must be detached before you configure it. The following is one example. You can use the chdev command instead.

   ```
   # ifconfig tr0 detach
   #
   ```

2. Invoke SMIT. If the adapter is token-ring, the following fast path is available.

   ```
   # smitty chgtok
   ```

3. Through submenu, you should get to the following screen. Then, update the entry field named TRANSMIT queue size, and/or RECEIVE queue size.



```
              Change / Show Characteristics of a Token Ring Adapter


 Type or select values in entry fields.
 Press Enter AFTER making all desired changes.

                                                          [Entry Fields]
   Token Ring Adapter                                   tok0
   Description                                          Token-Ring High-Perfor>
   Status                                               Available
   Location                                             00-01
   Receive data transfer OFFSET                        [92]                   +#
   TRANSMIT queue size                                 [40]                   +#
   RECEIVE queue size                                  [30]                   +#
   STATUS BLOCK queue size                             [10]                   +#
   RING speed                                           16                    +
   Receive ATTENTION MAC frame                          no                    +
   Receive BEACON MAC frame                             no                    +
   Enable ALTERNATE TOKEN RING address                  no                    +
   ALTERNATE TOKEN RING address                        [0x]                   +
   Apply change to DATABASE only                        no                    +

 F1=Help             F2=Refresh          F3=Cancel           F4=List
 F5=Reset            F6=Command          F7=Edit             F8=Image
 F9=Shell            F10=Exit            Enter=Do
```

Figure 29. SMIT Network Adapter Configuration Screen (Token-Ring)

4. Bring the interface up again with the ifconfig command. The following is one example. You can use the chdev command instead.

   ```
   # ifconfig tr0 up
   #
   ```

5. Do not forget to configure (load) the routing information:

   ```
   # /usr/lib/methods/cfginet
   zero
   9.68.214.1 net 0: gateway 9.68.214.1
   9.68.214.82 net 9.170.5: gateway 9.68.214.82
   ```

4.3.3.3 Queue Size Consideration

Remember that the default queue size 30, for both receive and transmission, is usually too small. The queue can be expanded up to 150. You should not have any problems expanding it, as shown:

```
# lsattr -R -l ent0 -a rec_que_size
20...150 (+1)
#
```

One drawback to having a longer queue length may be that a larger portion of memory is needed. But, the queue consists of pointers of mbufs and mbuf-clusters which hold packets. So, the memory occupied by the queue can be negligible small. Queues should have enough capacity to accommodate burst traffic.

Chapter 5. TCP/IP Related Parameter Tuning

In this chapter, we describe the TCP/IP parameter tuning techniques. For TCP/IP tuning, almost all the parameters are hard-coded or automatically adjusted by the TCP/IP modules, and there are a few parameters left for our adjustment. We could have a lengthy discussion to consider if this is advantageous or not. It′s really true that we can configure TCP/IP and have it up and running with only one SMIT panel, smitty mktcpip. Compare this with SNA Service configuration procedure. General flexibility and ease-of-use are in trade-off positions. It′s quite a challenge to achieve both virtues simultaneously.

The TCP/IP mechanism was designed based on the technologies available at the time of the first TCP/IP implementation, during late 1970s through early 1980s. Since then, many things have been changed dramatically. Today, emerging development enables the use of very fast network media such as FDDI. Also TCP/IP usages are expanding from LAN to WAN. Some TCP/IP extension have been introduced to support those requirements. With AIX Version 3.2.5 and later versions/releases, we have functional enhancement described in the *RFC 1323 TCP Extensions for High Performance*.

5.1 MTU and Fragmentation

The Maximum Transfer Unit (MTU) is a very important parameter. Since TCP/IP works completely with the default MTU configuration in almost all situations, few people notice the importance and carefully adjust this parameter before something happens. The fragmentation also impacts the TCP/IP performance. It is deeply related to the MTU. You can minimize the fragmentation and avoid the overhead by adjusting the MTU value. In this section, we mention the mechanism and concrete procedure to adjust the MTU.

5.1.1 MTU Basics

The MTU is a software parameter and it defines the maximum IP datagram length that can be sent out from an interface. Any IP datagram longer than the MTU should be split into several smaller IP datagrams before transmission. This splitting procedure is called fragmentation, and each split IP datagram is referred to as a fragment. All fragments of an IP datagram must be gathered and assembled to build the original IP datagram at the destination system. This is called reassembly. The fragmentation and reassembly procedures both impact TCP/IP performance.

Although the MTU is purely software (network interface) matter, it is deeply related to the underlying network media. For example, Ethernet Version 2 can treat up to 1500 bytes of IP datagram. Therefore, setting the MTU to larger than 1500 bytes for an Ethernet interface is meaningless. It′s more than meaningless because you can not send out any IP datagrams larger than 1500 bytes.

Note: The Ethernet Version 2 frame size can be up to 1514 bytes because the Ethernet has 14 bytes frame header.

Since the MTU is a configuration parameter of interfaces, the supported or available value may be different between systems (products). You or your network administrator has to find the most appropriate MTU value for all the systems connected to your network. All systems attached to the same IP network must share the same MTU value.

5.1.2 Fragmentation Mechanism

Now we explain the mechanism of fragmentation using a concrete example. For this experiment, we set MTU of token-ring network interface tr0 to 100 bytes. In actual environment, this configuration may be idiot example (too small). You can confirm current effective MTU values with the command netstat -i, as below:

```
# netstat -i
Name  Mtu   Network     Address           Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                          219     0      219     0     0
lo0   16896 127         localhost           219     0      219     0     0
tr0   100   <Link>10.0.5a.a8.b5.c1        33074     0     4842     0     0
tr0   100   9.68.214    mat.hakozaki.ib    33074     0     4842     0     0
#
```

Since a ping packet is encapsulated in an IP datagram, we used a ping packet (ICMP echo/reply) to see fragmentation.

1. Send a ping packet. Notice we specified 300 bytes length which exceed the MTU of 100 bytes. This forces the interface to divide the out going packet into four small fragments (do not forget ICMP header, and total data from IP point of view, is more than 300 bytes).

   ```
   # ping -c 1 -s 300 zero
   PING zero.hakozaki.ibm.com: (9.68.214.84): 300 data bytes
   308 bytes from 9.68.214.84: icmp_seq=0 ttl=255 time=5 ms

   ----zero.hakozaki.ibm.com PING Statistics----
   1 packets transmitted, 1 packets received, 0% packet loss
   round-trip min/avg/max = 5/5/5 ms
   #
   ```

   Note: The ping command reported that one packet was sent and received. Because fragmentation/reassembly is transparent to higher protocol than IP.

2. This is the ping packet (ICMP echo message) captured with the IP trace. In order to show that this IP datagram (fragment) has been fragmented and has successive fragments. The More Fragment (MF) bit is set in the IP header. This is the first fragment, the fragment offset is set to 0 as ip_off=0+. The plus (+) sign means a more fragment bit is set in the IP header. Due to the MTU of 100 bytes, the length of this IP datagram is just 100 bytes as ip_len=100.

   ```
   Packet Number 1
   TOK: ====( 122 bytes transmitted on interface tr0 )==== 16:15:14.629324160
   TOK: 802.5 packet
   TOK: 802.5 MAC header:
   TOK: access control field = 0, frame control field = 40
   TOK: ] src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70[
   TOK: 802.2 LLC header:
   TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
   IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
   IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
   IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=100, ip_id=4828, ip_off=0+
   IP:     ip_ttl=255, ip_sum=c98d, ip_p = 1 (ICMP)
   ICMP:   icmp_type=8 (ECHO_REQUEST)  icmp_id=4118  icmp_seq=0
   ```

3. This is the second fragment. Since this fragment is not the final fragment, an important point is that all fragments, derived from the same IP datagram. must share the same Identification Number. In this example, it is ip_id=4828. It still has the more fragments bit (+). The offset 80 means 80th bytes of the original user data is located at the beginning of this fragment (do not forget that the IP header has 20 byte length). Be aware that no transport header (in this case, it would be ICMP header) follows the IP header. During the fragmentation only the IP header is duplicated to each fragment, but it doesn′t care about any encapsulated data (transport layer header and data).

```
Packet Number 2
TOK: ====( 122 bytes transmitted on interface tr0 )==== 16:15:14.629382272
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: ] src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70[
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=100, ip_id=4828, ip_off=80+
IP:     ip_ttl=255, ip_sum=c983, ip_p = 1 (ICMP)
```

4. This is the third fragment. Everything is the same as the second fragment except ip_off=160+.

```
scale=auto width=75 keep=8.
Packet Number 3
TOK: ====( 122 bytes transmitted on interface tr0 )==== 16:15:14.629395840
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: ] src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70[
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=100, ip_id=4828, ip_off=160+
IP:     ip_ttl=255, ip_sum=c979, ip_p = 1 (ICMP)
```

5. This is the fourth and final fragment. Then it has fragment offset ip_off=240, but no longer has MF bit or (more fragments). Also, the length is ip_len=88 and this is the only fragment shorter than the MTU.

```
Packet Number 4
TOK: ====( 110 bytes transmitted on interface tr0 )==== 16:15:14.629409536
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=88, ip_id=4828, ip_off=240
IP:     ip_ttl=255, ip_sum=e97b, ip_p = 1 (ICMP)
```

6. This is ICMP echo reply message returned from the destination system zero. Notice that the system zero has the default MTU of 1492 bytes; it returns the IP datagram that is not fragmented. Another important point you should know is, although our system mat has MTU of 100 bytes, it can receive an IP datagram of 328 bytes. In our RS/6000′s implementation, MTU is applied to

an outgoing datagram, but it may not be applied to an incoming datagram. You can not always take this for granted.

```
Packet Number 5
TOK: ====( 350 bytes received on interface tr0 )==== 16:15:14.633146240
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=328, ip_id=309, ip_off=0
IP:     ip_ttl=255, ip_sum=fa50, ip_p = 1 (ICMP)
ICMP:   icmp_type=0 (ECHO_REPLY)  icmp_id=4118  icmp_seq=0
```

Pay attention to the IP datagram (fragment) length. The first to third fragments have ip_len=100 bytes and the fourth fragment has ip_len=88. The sum is 388 bytes, although we sent 300 byte data, because each fragment is an IP datagram and has an IP header of 20 bytes. Therefore, 80 bytes are used by four IP headers. Notice only the first fragment has an ICMP header of 8 bytes.

Our Experience

We also tried the same experiment with an MTU of 2000 bytes sending 2100 data. The first fragment had ip_len=1996 and the second fragment had ip_len=152. The total length of 2148 bytes, can be explained with two 20 byte IP headers and one 8 byte ICMP header. The curious point was why the first fragment was 1996 bytes although the MTU was 2000 bytes.

In the fragmentation scheme, each fragment must have the offset in the offset field of the IP header. Offset can be specified only by a multiple of 8 bytes or 64 bits. Therefore in this case, a fragment can carry up to 1976 bytes of user data taking 20 bytes of the IP header into account.

In this example, at the destination system, the following counter was incremented by four because it received four fragments:

```
zero # netstat -p ip
ip:
:
        412 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 with bad options
        0 with incorrect version number
        24 fragments received
        0 fragments dropped (dup or out of space)
        0 fragments dropped after timeout
        3 packets reassembled ok
        306 packets for this host
        0 packets for unknown/unsupported protocol
        0 packets forwarded
        4 packets not forwardable
        0 redirects sent
        337 packets sent from this host
        0 packets sent with fabricated ip header
        0 output packets dropped due to no bufs, etc.
        0 output packets discarded due to no route
        0 output datagrams fragmented
        0 fragments created
        0 datagrams that can′t be fragmented
        85 IP Multicast packets dropped due to no receiver
        0 ipintrq overflows
zero #
```

At the source system, the counter (output datagram fragmented) was incremented by one, and the fragments created were incremented by four (the IP datagram was fragmented to four parts).

```
mat # netstat -p ip |pg
ip:
:
        19476 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 with bad options
        0 with incorrect version number
        0 fragments received
        0 fragments dropped (dup or out of space)
        0 fragments dropped after timeout
        0 packets reassembled ok
        8714 packets for this host
        0 packets for unknown/unsupported protocol
        4 packets forwarded
        4 packets not forwardable
        1 redirect sent
        4828 packets sent from this host
        0 packets sent with fabricated ip header
        0 output packets dropped due to no bufs, etc.
        0 output packets discarded due to no route
        1 output datagram fragmented
        4 fragments created
        0 datagrams that can′t be fragmented
        10758 IP Multicast packets dropped due to no receiver
        0 ipintrq overflows
mat #
```

Note: The AIX V3.2 doesn′t provide these counters with the netstat -p ip command.

5.1.3 MTU Configuration

You can configure the MTU for each interface. As usual, SMIT (or high-level command) and a standard UNIX command, ifconfig, are available.

5.1.3.1 Using SMIT

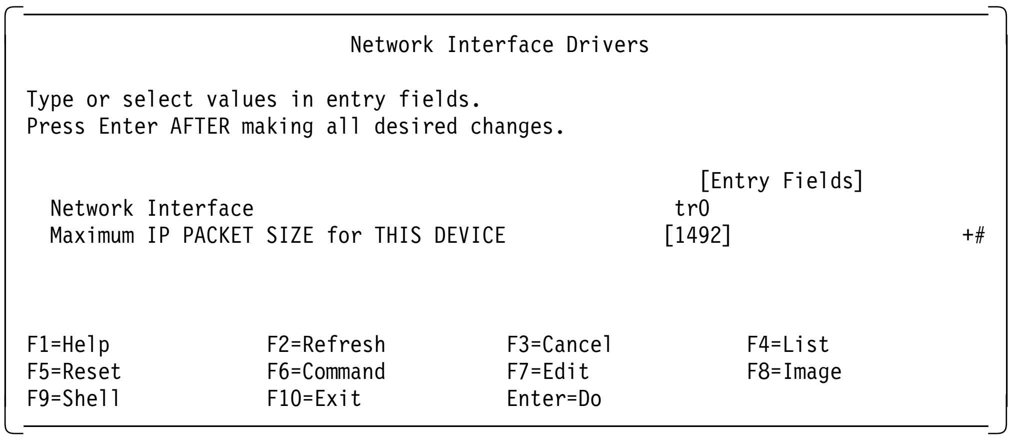
Although MTU is an attribute of a network interface device, the MTU configuration screen has the titled Network Interface Drivers. This is the only attribute of the network interface drivers.

1. Invoke the SMIT with the fast path, as follows:

   ```
   # smitty chif
   ```

   Then, you will be asked to choose the network interface from all the available network interfaces. After the selection, a configuration panel is displayed.

2. There is only one entry field in the configuration panel. This is for the MTU configuration. This operation updates the MTU values both in the ODM and the kernel.



```
                           Network Interface Drivers

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
  Network Interface                                   tr0
  Maximum IP PACKET SIZE for THIS DEVICE             [1492]                +#



F1=Help            F2=Refresh         F3=Cancel          F4=List
F5=Reset           F6=Command         F7=Edit            F8=Image
F9=Shell           F10=Exit           Enter=Do
```

Figure 30. SMIT MTU Configuration Screen (Token-Ring)

5.1.3.2 Using High-Level Command

The high-level command, which has the equivalent effect to SMIT, is the chdev.

1. Issue this command, as follows:

   ```
   # chdev -l tr0 -a mtu=2000
   tr0 changed
   #
   ```

   Note: Token-ring interface device has four MTU related attributes (mtu, remmtu, mtu_4 and mtu_16). Other interfaces devices have two MTU related attribute (mtu and remmtu). They are not used now. The only valid attribute is mtu.

   You can confirm the update in the ODM with the lsattr command as follows:

   ```
   # lsattr -E -l tr0 -a mtu
   mtu 2000 Maximum IP Packet Size for This Device True
   #
   ```

Since SMIT and lsattr confirm only the content of the ODM, use the netstat -i command to confirm if the kernel is really recognizing the updated MTU, as follows:

```
# netstat -i
Name  Mtu   Network     Address           Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                          214     0      214     0     0
lo0   16896 127         localhost           214     0      214     0     0
tr0   2000  <Link>40.0.7e.8.66.70          1523     0      674     0     0
tr0   2000  9.68.214    zero.hakozaki.i    1523     0      674     0     0
#
```

5.1.3.3 Using Standard UNIX Command ifconfig

AIX V3.2 and V4.1 also supports the MTU configuration with the standard UNIX command ifconfig.

1. Issue the ifconfig command, as follows:

   ```
   # ifconfig tr0 mtu 2000
   #
   ```

2. You can confirm if the change was made successfully with the netstat -i command. Notice the that ifconfig only updates the kernel parameter.

   ```
   # netstat -i
   Name  Mtu   Network     Address           Ipkts Ierrs    Opkts Oerrs  Coll
   lo0   16896 <Link>                          214     0      214     0     0
   lo0   16896 127         localhost           214     0      214     0     0
   tr0   2000  <Link>40.0.7e.8.66.70          1448     0      659     0     0
   tr0   2000  9.68.214    zero.hakozaki.i    1448     0      659     0     0
   #
   ```

3. As mentioned previously, the ODM is not updated, as follows:

   ```
   # lsattr -E -l tr0 -a mtu
   mtu 1492 Maximum IP Packet Size for This Device True
   #
   ```

   It stores the default value in this example. The update can not be reserved for the system reboot. When you boot the system next time, the MTU stored in the ODM will be loaded unless you explicitly describe this command in a startup script such as the /etc/rc.net.

The ifconfig command doesn't check the validity of the specified parameter. For example, if the interface is Ethernet Version 2 or en0, the MTU must be between 60 - 1500. But, you can configure the value bigger than 1500 (such as 2000). As a result, any IP datagram bigger than 1500 bytes cannot be sent out. If you do this, any application or operation that sends data bigger than 1500 bytes just hangs.

Note: If it would be SMIT or high-level command, you would get an error message if you tried to configure an invalid MTU.

5.1.3.4 MTU Default and Valid Values

Our AIX V4.1 and RS/6000 have the following default MTU values.

Table 2. Available MTU Values (Bytes)

Interface	Type	Default	Minimum	Maximum
Ethernet Version 2	en0	1500	60	1500
IEEE802.3 Ethernet	et0	1492	60	1492
Token Ring 4M	tr0	1492	60	4096
Token Ring 16M	tr0	1492	60	17960
X.25	xt0	576	60	1024
SLIP	sl0	1006	60	4096
FDDI	fi0	4352	1	4352
Block MPX Channel	ca0	4096	1024	4096
SOCC	so0	61428	1	61428

Note: Be aware that the default MTU of token-ring is not the maximum MTU. This is due to a specific reason (that will be mentioned later), but you had better change the default for many occasions in order to optimize performance.

You can ask your system for the default and possible range of the MTU, by using the high-level lsattr command. This is the case of a token-ring network interface. Use the -R flag to review the supported parameter range as follows:

```
# lsattr -R -l tr0 -a mtu
60...17792 (+1)
```

You can use the -D flag to review the default value, as follows:

```
# lsattr -D -l tr0 -a mtu
mtu 1492 Maximum IP Packet Size for This Device True
#
```

5.1.4 MTU Pitfalls

There are some known pitfalls about MTU. Some are performance issues and some are connectivity issues.

5.1.4.1 A Bigger MTU Is Better to a Point

As we have already seen, MTU optimization is important for performance tuning. Both fragmentation at a sending system and reassembly at a receiving system are considerable overhead. Generally speaking, it would be better to avoid fragmentation. The basic principle for MTU tuning is to make it as large as possible. Of course, the MTU must be smaller than the maximum frame size of the data link.

Note: As you have already seen in AIX V3.2 and V4.1, the default MTU is usually the maximum frame size (maximum MTU). But, token-ring is an exception. It has a considerably smaller default MTU than the maximum MTU.

But, MTU should not be determined by your convenience only because there may be a system that cannot receive frames appropriate for you. RS/6000 can receive frames up to the maximum frame size of the involved data link even if the interface is configured to smaller MTU. This is not true for all products. As in the RFC 1122, there may be a system which can only receive up to 576 bytes of the IP datagram. Nowadays such systems seem exceptional. Therefore, MTU should be agreed by all systems in the same IP network.

Note: A mismatch of MTUs are difficult to notice, because only a large IP datagram that exceeds the destination system′s MTU causes a problem. If TCP is used, it can not be detected because TCP has a mechanism called MSS and it automatically adjusts all the segment sizes to smaller than the effective MTU. Then, only UDP datagrams meet a problem.

RFC 1122 Requirements for Internet Hosts, Page 56

The IP layer MUST implement reassembly of IP datagram.

We designated the largest datagram size that can be reassembled by EMTU_R (″Effective MTU to receive″); this is sometimes called the ″reassembly buffer size″. EMTU_R MUST be greater than or equal to 576, SHOULD be either configurable or indefinite, and SHOULD be greater than or equal to the MTU of the connected network(s).

Another issue is that bigger is not always better. A UDP datagram is mapped to an IP datagram. If you try to avoid fragmentation, UDP datagrams should be smaller than MTU and this means that your UDP application has to issue a system call, sendto() or similar call with data sizes smaller than the MTU (exactly smaller than the MTU minus IP and UDP header). If the MTU is small and you have to send a large amount of data, your application must issue many system calls with small chunks of data. Which is better for performance, having many system calls by an application and suppressing fragmentation, or having a few system calls by an application and accepting fragmentation? We expect that suppressing fragmentation is better in most situations, but if you need the exact answer, you should make a benchmark.

Our Experience

A customer had an FDDI backbone network and some Ethernet branch networks. They used NFS through those networks. NFS uses UDP and UDP datagram size (more accurately the size of RPC) can be configured by the NFS send buffer size. Since Ethernet′s maximum frame size is 1500 bytes and FDDI′s is 4352 bytes, in order to avoid fragmentation at routers between FDDI and Ethernet, the MTU for both FDDI and Ethernet should be 1500 bytes. This rules that each RPC should be smaller than 1500 bytes. As a result, many RPCs are needed.

The customer made several benchmark tests and the results due interesting. In that customer′s environment, accepting fragmentation and reducing the number of RPC enhanced performance. Of course, in this case, the routers performance characteristics were one of the key factors.

5.1.4.2 Fragmentation at Routers

The fragmentation and assembly are completely transparent from any higher layer (than IP), except for performance degradation. It can be overhead for a sender, a receiver and network, including interconnect devices such as a router.

The fragmentation may be done at any router on the way to the destination system. Also, a fragment can be fragmented again into some smaller fragments.

But, the reassembly is done only by the final destination system. All fragments must arrive at the destination system within the timeout period or they are discarded. This invites retransmission of all fragments (entire IP datagram) and not just retransmission of a lost fragment. Then the impact of a lost fragment is not trivial.

Since a source or sending system never knows if the frames (IP datagrams) are fragmented at any router, this means that the sending system cannot control fragmentation if the destination is not located in the same local IP network (or subnet). If an IP datagram must pass through several networks, the network which has the smallest MTU decides the final result. If we could know the smallest MTU on the way *before* sending an IP datagram, we could optimize the performance.

Note: If you have some token-ring networks connected by routers or bridges, running those networks with the default MTU of 1492 bytes doesn′t give you the best result.

Although it is not widely implemented, an interesting mechanism has been proposed. It is described in the *RFC 1191 Path MTU Discovery*. It is a draft standard protocol and the state is elective. This mechanism provides us with a way to determine the smallest MTU value of the path, and allows us to avoid fragmentation on routers. Currently our RS/6000 doesn′t support this protocol.

As for now, the safest way is to choose your MTU for the minimum requirement. It is recommended that you use 576 bytes of MTU if a destination system is located in a remote IP network.

RFC 1122 Requirements for Internet Hosts, Page 58

It is generally desirable to avoid local fragmentation and to choose EMTU_S low enough to avoid fragmentation in any gateway along the path. In the absence of actual knowledge of the minimum MTU along the path, the IP layer SHOULD use EMTU_S <= 576 whenever the destination address is not on a connected network, and otherwise use the connected network′s MTU. ...

DISCUSSION

Picking the correct datagram size to use when sending data is a complex topic ...

Since nearly all networks in the Internet currently support an MTU of 576 or greater, we strongly recommend to use of 576 for datagrams sent to non-local network.

5.1.4.3 Remote MTU (rmmtu) Is Obsolete

The old version of AIX, V3.1, had the option to configure the MTU value for the remote network. This means that AIX V3.1 could have two MTUs for one interface. One is for the locally attached IP network and the other is for the remote IP networks through router(s). This feature is now obsolete and we don′t have this capability with AIX V3.2 and V4.1. There remains some attributes or parameters in AIX V3.2 and V4.1, which look as if they are available and effective, but they are totally obsolete now.

Note: Also, for a token-ring interface, mtu_4 and mtu_16 are not used. Although the lsattr commands show you some MTU related attributes, the only valid attribute is mtu. In the example below, the attributes, remmtu, mtu_4 and mtu_16 are not used:

```
# lsattr -E -l tr0
mtu       1492            Maximum IP Packet Size for This Device     True
mtu_4     1492            Maximum IP Packet Size for This Device     True
mtu_16    1492            Maximum IP Packet Size for This Device     True
remmtu    576             Maximum IP Packet Size for REMOTE Networks True
netaddr   9.68.214.82     Internet Address                           True
state     up              Current Interface Status                   True
arp       on              Address Resolution Protocol (ARP)          True
allcast   on              Confine Broadcast to Local Token-Ring      True
hwloop    off             Enable Hardware Loopback Mode              True
netmask   255.255.255.128 Subnet Mask                                True
security  none            Security Level                             True
authority                 Authorized Users                           True
broadcast                 Broadcast Address                          True
#
```

Although we don′t have a separate MTU configuration procedure for remote IP networks, we can configure separate TCP MSS for remote IP networks. This can be made by the route -mtu command or no -o tcp_mssdflt. Unfortunately this is TCP only.

5.1.4.4 When You Have a Bridge

Care must be taken if you use a bridge to connect two network segments. If you connect an Ethernet Version 2 segment to a token-ring segment, you should not configure the same MTU for both networks. The MTU of token-ring must be eight bytes *less* than that of Ethernet Version 2.

Note: Since token-ring and Ethernet headers are data link layer′s headers, they are not included when MTU is evaluated. Be aware that MTU is a parameter for IP datagram length.

Because the token-ring frame has an eight byte LLC header and Ethernet Version 2 doesn′t have this header, when a token-ring frame passes through a bridge, the bridge must convert the token-ring header to an Ethernet Version 2 header. Almost all token-ring header fields can be mapped to corresponding Ethernet header fields, but Ethernet Version 2 doesn′t have an eight bytes LLC field. Therefore, the bridge merges the LLC field into a data field during the conversion. As a result, a converted frame has additional eight bytes in a data field. If you configure both segments to the same MTU (for instance 1500 bytes) then, when a token-ring frame carrying 1500 bytes of data becomes an Ethernet frame carrying 1508 bytes of data, this frame can not pass through the bridge to the Ethernet segment.

Note: This is why the default MTU of token-ring is 1492 bytes.

It′s not so easy to notice a misconfigured MTU, because as with the previous example, only frames carrying more than 1492 bytes of data in token-ring have this problem. Other frames are safely transmitted.

5.2 TCP Maximum Segment Size

A unit of transmission in a TCP layer is called a *segment*. A UDP datagram has the maximum length of 65,536 bytes. An IP datagram also has the maximum length of 65,536 bytes. How about a TCP segment? A TCP segment also has a maximum length, but it is determined at the moment of connection establishment. The Maximum Segment Size (MSS) is negotiated by both systems and this guarantees connectivity because both systems can avoid sending a segment which the other cannot receive.

Notice that the IP layer has an MTU and any IP datagram larger than the MTU is fragmented. Fragmentation impacts performance and it is recommended that you avoid the fragmentation. The TCP MSS is determined taking the MTU into consideration.

5.2.1 MSS Basics

The MSS negotiation is made by using the TCP option field. The negotiation is as follows:

1. MSS is calculated using the MTU.

   Note: There is a bit of a complex procedure involved in this MSS determination process. Actually, values of route -mtu, no -o tcp_mssdflt and no -o subnetsarelocal are evaluated and used if necessary.

   a. Calculate the following variables A and B. Notice that the IP and TCP headers are 20 bytes each.

      A = MTU value - (TCP header size + IP header size)

      B = Socket Receive Buffer size ÷ 2

   b. Compare A and B; the smaller one is used as the effective MSS.

2. The MSS decided above is advertised with the connection establishment segment (SYN or SYN/ACK segment).

   Note: If the destination system doesn′t support an MSS negotiation option, 536 bytes is used as the destination system′s MSS. This is due to *RFC 1122 Requirements for Internet Hosts*; any system must support at least 576 bytes length of the IP datagram.

3. Get the destination system′s MSS from ACK or SYN/ACK segment received.

4. Compare its MSS and destination′s MSS; the smaller one is decided to be the effective MSS.

Here, we show you a concrete example through a TELNET connection. In this case, both a source system, mat, and a destination system, zero, are AIX V4.1.2.

1. In this example, the mat initiates connection (active open). In this segment, the data 020405ac is for the TCP option. Why can we say it is not application data? This is because the offset field of the TCP header, th_off=6, shows that this segment has one 4 bytes option field. If there are no options, this field should be 5. The format of the TCP option field is as follows.

1st byte	Type of TCP option (MSS is 2.)
2nd byte	Option field length in byte (MSS is 4.)
After 3rd byte	Option data (MSS value if MSS)

In this case, the mat is advertising the MSS value of 05ac in hex. This is 1452 bytes in decimal and if you add to the length of the TCP and IP headers (40 bytes), it becomes 1492 bytes. This is the default MTU value of the token-ring. Also, you can know that the receive buffer size of the mat is 16 KB by referring to the field, th_win=16384:

```
Packet Number 1
TOK: ====( 66 bytes transmitted on interface tr0 )==== 17:21:05.857892352
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=5370, ip_off=0
IP:     ip_ttl=60, ip_sum=aaa3, ip_p = 6 (TCP)
TCP:    <source port=1053, destination port=23(telnet) >
TCP:    th_seq=5180a801, th_ack=0
TCP:    th_off=6, flags<SYN>
TCP:    th_win=16384, th_sum=9b49, th_urp=0
TCP: 00000000     020405ac                                |....            |
```

2. The destination system zero responds with the SYN/ACK segment (passive open). As you saw with the prior segment, this TCP segment also has an MSS option. The advertising MSS is 0fd8 and the reason is that the MSS of zero was updated to 4056 bytes, which is the maximum MTU for 4 MB token-ring. As a result, MSS is decided to be 1452 bytes because the smaller one wins. Although the receive buffer size of the newton is also 16384 bytes, it has already received SYN segment from the mat, the window size is slightly reduced to th_win=15972:

```
Packet Number 2
TOK: ====( 66 bytes received on interface tr0 )==== 17:21:05.860547328
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1],
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=1090, ip_off=0
IP:     ip_ttl=60, ip_sum=bb5b, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1053 >
TCP:    th_seq=22baa001, th_ack=5180a802
TCP:    th_off=6, flags<SYN | ACK>
TCP:    th_win=15972, th_sum=cfec, th_urp=0
TCP: 00000000     02040fd8                                |....            |
```

3. Now that the mat responds, the ACK segment and TCP three-way hand shake is completed:

```
Packet Number 3
TOK: ====( 62 bytes transmitted on interface tr0 )==== 17:21:05.860603008
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
```

```
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=5371, ip_off=0
IP:     ip_ttl=60, ip_sum=aaa6, ip_p = 6 (TCP)
TCP:    <source port=1053, destination port=23(telnet) >
TCP:    th_seq=5180a802, th_ack=22baa002
TCP:    th_off=5, flags<ACK>
TCP:    th_win=15972, th_sum=f1cd, th_urp=0
```

4. This is a TELNET segment. Notice the offset field is `th_off=5` and this segment is not carrying any TCP option. The data `fffd01ff...` is TELNET data and you need to know TELNET protocol if you want to interpret it.

```
Packet Number 4
TOK: ====( 77 bytes transmitted on interface tr0 )==== 17:21:05.883144448
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=55, ip_id=5372, ip_off=0
IP:     ip_ttl=60, ip_sum=aa96, ip_p = 6 (TCP)
TCP:    <source port=1053, destination port=23(telnet) >
TCP:    th_seq=5180a802, th_ack=22baa002
TCP:    th_off=5, flags<PUSH | ACK>
TCP:    th_win=15972, th_sum=bcf5, th_urp=0
TCP: 00000000     fffd01ff fd03fffb 18fffdc8 fffb1f       |............... |
```

In this example, a 4 MB token-ring is used for the data link layer. Also, the system, the mat and the zero are in the same physical ring. The previous MSS of 1452 bytes is a bad example. Since a 4 MB token-ring can support up to 4,096 bytes of MTU, the MTU should be configured to be 4,096 at both sides. Of course, receive buffer sizes should be expanded to larger than 2,048 bytes. Then, you can achieve the maximum efficiency.

5.2.1.1 MSS Customization

As explained before, the MSS is automatically defined by TCP. In most cases, since an interface MTU is smaller than the half of the receive buffer, the interface MTU defines the MSS. But, for some situations, you may need to override this decision logic. Because MTU is defined for each interface, all the TCP segments sent from the same interface share the same MTU. In certain cases, this is not the best choice. Currently two procedures are provided with the AIX V3.2.5 and a later release/version for this issue. One option is to use the -mtu option of the route command. The other is to use the tcp_mssdflt option of the no command.

5.2.2 The Route -mtu Command

Consider the following situation. You have a 16 MB token-ring network, and you configured the MTU for your IP network to 17960 bytes. Later, you have to connect an Ethernet network through a router. In order to avoid fragmentation at the router, you should change the MTU to 1500 bytes. But, many of your destination systems are still in your local token-ring network. Then, changing the MTU to 1500 invites inefficiency within the token-ring.

The -mtu option of the route command allows you to configure the MTU for each route (not just for each interface). With this option, you can give a specific MTU value to only that traffic through the router.

Note: This option is only valid to the TCP segment. In other words, this option is only evaluated at the MSS decision procedure. Other protocols such as UDP, can not have this advantage.

The following is an example of this option.

1. Now our system, mat has a 4MB token-ring interface tr0 with the maximun MTU 4096 bytes.

   ```
   # netstat -i
   Name  Mtu   Network     Address           Ipkts Ierrs    Opkts Oerrs  Coll
   lo0   16896 <Link>                          219     0      219     0     0
   lo0   16896 127         localhost           219     0      219     0     0
   tr0   4096  <Link>10.0.5a.a8.b5.c1        35419     0     5990     0     0
   tr0   4096  9.68.214    mat.hakozaki.ib   35419     0     5990     0     0
   #
   ```

2. On the mat, the TCP receive buffer size is set to 16384 bytes (this is default):

   ```
   # no -a | grep space
               tcp_sendspace = 16384
               tcp_recvspace = 16384
               udp_sendspace = 9216
               udp_recvspace = 41600
   #
   ```

   With this configuration, the mat will advertise its MSS of 4096 bytes during a TCP connection establishment because a half of the receive buffer is 8192 bytes. Still, the MTU is smaller.

3. But this time, we explicitly configured a specific route to the destination system, ts, using the route -mtu, as follows:

   ```
   # route add -host ts 9.68.214.1 -mtu 3072
   3072 host ts: gateway 9.68.214.1
   #
   ```

   Check to see if the route has really been added, as follows:

   ```
   # netstat -r | grep U
   Destination      Gateway            Flags     Refs     Use  Interface
   Route Tree for Protocol Family 2:
   default          9.68.214.1         UG          2      982  tr0
   ts.hakozaki.ibm. 9.68.214.1         UGH         0        0  tr0
   9.68.214         mat.hakozaki.ibm.c U           9     4536  tr0
   localhost        localhost          UH          1        2  lo0
   #
   ```

   Now all routes attached to the tr0 share the MTU of 4096 bytes except the route to the system ts. This route has the MTU of 3072 bytes.

   Note: As you can see, this route specific MTU can only be defined for a remote network or a remote host. You can not specify an explicit route to the host in the same local network.

   There is no way to review if a route has a specific MTU configured. If you are not sure what value was set to a route (or if you want to confirm if a route has the specific MTU) delete and add the MTU again.

4. Then, we invoked the TELNET from the mat to the ts and captured the packet with an IP trace. As you can see, the mat is advertising its MSS as 0bd8 or 3032 bytes in spite of the interface MTU (4096 bytes).

   Note: Do not forget to reduce the IP and TCP header size (40 bytes) from an MTU to compare it with an MSS.

```
Packet Number 4
TOK: ====( 66 bytes transmitted on interface tr0 )==== 17:47:15.562801024
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: src = 10:00:5a:a8:b5:c1, dst = 00:00:fa:37:91:64
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=5550, ip_off=0
IP:     ip_ttl=60, ip_sum=adb7, ip_p = 6 (TCP)
TCP:    <source port=1059, destination port=23(telnet) >
TCP:    th_seq=5d80ec01, th_ack=0
TCP:    th_off=6, flags<SYN>
TCP:    th_win=16384, th_sum=48df, th_urp=0
TCP: 00000000     02040bd8                                |....            |
```

5. The following segment is from the ts destination. The ts is also advertising its MSS as 05ac or 1452 bytes. The ts is also in a 4 MB token-ring network and it is only using the default MTU. Therefore, in spite of our effort, the MSS would be agreed to at 1452 bytes.

   Note: You need to make adjustments on both sides.

```
Packet Number 5
TOK: ====( 66 bytes received on interface tr0 )==== 17:47:15.566755072
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 00:00:fa:37:91:64, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=13225, ip_off=0
IP:     ip_ttl=58, ip_sum=91bc, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1059 >
TCP:    th_seq=a42f6201, th_ack=5d80ec02
TCP:    th_off=6, flags<SYN | ACK>
TCP:    th_win=15972, th_sum=4a65, th_urp=0
TCP: 00000000     020405ac                                |....            |
```

5.2.3 The no -o tcp_mssdflt Command

In the prior section, it is explained how to override the interface wide MTU configuration when you establish a connection to a remote network. If you don′t configure a MTU for the remote network, your system automatically chooses the default MTU for all remote networks, that is 512 bytes. This default value can be changed with the option of the no command, tcp_mssdflt.

Note: This option is only effective for TCP. More accurately, this option is only used by TCP MSS determination process.

The safest approach is to use the minimum MTU of 576 bytes for non-local networks. The default value of tcp_mssdflt is 512 bytes and this is smaller than the value recommended in the *RFC 1122*. The reason is written in the header file /usr/include/netinet/tcp.h with the value itself as follows (refer to C.1, "/usr/include/netinet/tcp.h" on page 365 for the complete list of the file):

```
/*
 * Default maximum segment size for TCP.
 * With an IP MSS of 576, this is 536,
 * but 512 is probably more convenient.
 * This should be defined as MIN(512, IP_MSS - sizeof (struct tcpiphdr)).
 */
#define TCP_MSS 512
```

Notice the option of the no command, subnetsarelocal, has a crucial role at this stage if you are using the subnet mask in your environment. If this option is set to 1, your system considers that any IP networks defined by the subnet mask are all located in the same physical network media. As a result, your system doesn′t apply the tcp_mssdflt to those systems which are located in different IP subnet networks. If the subnetsarelocal is set to 0, your system recognizes each IP subnet network and uses tcp_mssdflt. Be aware that the default of the subnetsarelocal is 1. You *must* explicitly set this option if you need to use anything other than 1.

```
# no -o subnetsarelocal=0
# no -o subnetsarelocal
subnetsarelocal = 0
#
```

Note: Maybe the tcp_mssdflt is the only parameter which is affected by the subnetsarelocal. This is why a lot of people don′t know about the meaning of subnetsarelocal.

tcp_mssdflt
This defines MTU for remote networks. The MTU configured with this option cannot exceed the interface wide MTU. The default is 512. This is quite reasonable. To reach a remote network, any segment first goes through the local network.

subnetsarelocal
This defines if subnets are really different networks (physically). The default is 1; subnets are physically the local network.

The procedure of using this option is outlined below, using a real example.

1. The system checks to see if the route specific MTU is defined to the destination. If the route specific MTU is defined, tcp_mssdflt is never evaluated or used. The value of the route specific MTU always has the higher priority. In this example, our destination is the ts, as follows:

   ```
   # host ts
   ts.hakozaki.ibm.com is 9.68.210.140
   #
   ```

   The source system is mat, and it has following routes. Although it has route to the network, 9.68.210.128, no explicit MTU is defined for that route. As you can see, we used the subnet mask of 255.255.255.128.

```
# netstat -r |grep U
Destination      Gateway            Flags     Refs     Use  Interface
default          9.68.214.1         UG          1     1084  tr0
9.68.210.128     9.68.214.1         UG          0        0  tr0
9.68.214         mat.hakozaki.ibm.c U           9     4653  tr0
localhost        localhost          UH          1        2  lo0
#
```

2. The system checks to see if the destination is located in the same local network or remote network. If the destination is in the local network (in other words in the same IP network and it can be reached without routers), this option is not evaluated. The system just uses the interface MTU for the MSS deduction.

   Note: Do not forget that subnetsarelocal has a meaning here.

   Since we are using subnet mask and the ts is in the remote subnet (actually this remote subnet is physically a different network), then we set subnetsarelocal to 0.

```
# no -o subnetsarelocal
subnetsarelocal = 1
# no -o subnetsarelocal=0
# no -o subnetsarelocal
subnetsarelocal = 0
#
```

3. Next, the system checks to see if the value of the tcp_mssdflt is larger than the interface MTU (exactly speaking, interface MTU minus IP and TCP header, 40 bytes). If tcp_mssdflt is larger, the interface MTU is used for MSS deduction. If the tcp_mssdflt is smaller, this value is used for the MSS negotiation. The interface of the mat has an MTU of 1492 bytes.

```
# netstat -i
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 <Link>                           219     0      219     0     0
lo0   16896 127         localhost            219     0      219     0     0
tr0   1492  <Link>10.0.5a.a8.b5.c1         36713     0     6289     0     0
tr0   1492  9.68.214    mat.hakozaki.ib    36713     0     6289     0     0
#
```

   Our defined tcp_mssdflt is 1024 bytes. Then, this 1024 bytes should be used for the MSS negotiation.

```
# no -o tcp_mssdflt
tcp_mssdflt = 512
# no -o tcp_mssdflt=1024
# no -o tcp_mssdflt
tcp_mssdflt = 1024
#
```

4. We invoked TELNET at the mat to the ts and captured the packets with the IP trace. This is the SYN segment from the mat and the MSS of 0400 or 1024 bytes is advertised.

```
Packet Number 3
TOK: ====( 66 bytes transmitted on interface tr0 )==== 18:46:33.063450112
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: ] src = 10:00:5a:a8:b5:c1, dst = 00:00:fa:37:91:64[
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
```

```
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=5799, ip_off=0
IP:     ip_ttl=60, ip_sum=acbe, ip_p = 6 (TCP)
TCP:    <source port=1070, destination port=23(telnet) >
TCP:    th_seq=78aeee01, th_ack=0
TCP:    th_off=6, flags<SYN>
TCP:    th_win=16384, th_sum=337e, th_urp=0
TCP: 00000000     02040400                                |....            |
```

5. This is the response from the ts and the MSS of 05ac or 1452 bytes is advertised. Since ts had subnetsarelocal=1 and cannot distinguish any subnets, then it used the interface MTU of 1492 bytes (it is the default for token-ring). After the MSS negotiation, a smaller MSS, 1024, would be taken.

```
Packet Number 4
TOK: ====( 66 bytes received on interface tr0 )==== 18:46:33.067565824
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 00:00:fa:37:91:64, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=14885, ip_off=0
IP:     ip_ttl=58, ip_sum=8b40, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1070 >
TCP:    th_seq=bf549a01, th_ack=78aeee02
TCP:    th_off=6, flags<SYN | ACK>
TCP:    th_win=16384, th_sum=d86a, th_urp=0
TCP: 00000000     020405ac                                |....            |
```

In this example, our networks are all token-ring 4 MB. In such an environment, just leaving everything as the default values is not a recommended approach. Of course, choosing an unnecessarily small MSS, such as 1024, is not a good example.

Note: If you are using all the default parameters and using subnet, just changing the subnetsarelocal from 1 to 0 changes the MSS from 1492 to 512 for remote networks.

5.3 IP Queue

The IP module keeps a queue for incoming IP datagrams. The IP queue is only for receiving datagrams, and outgoing datagrams are not queued at the IP layer. The queue length is the only parameter you can do with the IP queue.

Since routing is a function of the IP layer, this parameter is important for the router-configured system. Even if you are not running the TCP/IP applications on your system, someone may be putting a heavy load on your system in order to pass through his or her datagrams.

5.3.1 Getting IP Queue Status

Since the IP queue has a finite length, if a system gets a burst of IP datagrams, the queue can overrun. In that case, some datagrams can be lost.

With V4.1, the netstat -p ip command is enhanced to support the counter for the IP queue overflow. Watch the counter, ipintrq overflows, as follows:

```
# netstat -p ip
...
        0 output datagrams fragmented
        0 fragments created
        0 datagrams that can′t be fragmented
        2531 IP Multicast packets dropped due to no receiver
        0 ipintrq overflows
#
```

Difference between V4.1 and V3.2

Unfortunately with V3.2 there are no commands or utilities which you can use to check the IP queue status or datagram loss. You should use the crash command, and read the overflow counter in the kernel directly. The following is a sample procedure.

1. First, invoke the crash and you will see the following prompt:

   ```
   # crash
   >
   ```

2. Type the subcommand knlist ipintrq. You will see the following address:

   ```
   > knlist ipintrq
           ipintrq: 0x014a5540
   >
   ```

3. Next, use the od subcommand and specify the address 10 (hex) greater than the address you got in the previous step.

   ```
   > knlist ipintrq
           ipintrq: 0x014a5540
   > od 0x014a5550
   014a5550: 00000000
   > q
   #
   ```

4. Now you have the counter. In this example, the overflow counter is 0 and we have not had an IP queue overflow problem. You can exit the crash with the q subcommand.

5.3.2 IP Queue Configuration

The queue length can be configured with the no command. The default value is 50. You can review the current value with the -o flag.

1. Issue the no command with the ipqmaxlen option and review the current value:

   ```
   # no -o ipqmaxlen
   ipqmaxlen = 100
   #
   ```

2. You can configure the queue length, as follows:

```
# no -o ipqmaxlen=200
#
```

3. Confirm that the update was made correctly, as follows:

```
# no -o ipqmaxlen
ipqmaxlen = 200
#
```

If you need this update to be permanent, write it in the startup script /etc/rc.net. The best place is the end of the script, as shown:

```
###################################################
# The socket default buffer size (initial advertised TCP window) is being
# set to a default value of 16k (16384). This improves the performance
# for Ethernet and token-ring networks.  Networks with lower bandwidth
# such as SLIP (Serial Line Internet Protocol) and X.25 or higher bandwidth
# such as Serial Optical Link and FDDI would have a different optimum
# buffer size.
# ( OPTIMUM WINDOW = Bandwidth * Round Trip Time )
###################################################
if [ -f /usr/sbin/no ] ; then
        /usr/sbin/no -o tcp_sendspace=16384
        /usr/sbin/no -o tcp_recvspace=16384
        /usr/sbin/no -o ipqmaxlen=200
fi
```

5.3.3 IP Queue Pitfall

There is a pitfall for expanding the queue size. No additional memory is used by increasing the queue length. However, it may result in more time spent in the off-level interrupt handler since IP will have more packets to process on its input queue. This could adversely affect the processes needing CPU time. The trade-off is reduced packet dropping versus CPU availability for other processing. It is best to increase the ipqmaxlen option by moderate increments if the trade-off is a concern.

If your system is configured as a router, routing activity consumes IP queue and CPU time. Even just receiving many IP broadcast packets may affect the system performance.

5.4 TTL (Time-To-Live)

TTL or Time-To-Live is a rather trivial function from a performance improvement point of view. In a few situations, they may not play a trivial part, but mainly this section intends to give you some TCP/IP knowledge that may help you to understand TCP/IP mechanism.

5.4.1 IP Datagram TTL Basics

The TTL is implemented in the IP layer only. You could see the field ip_ttl in a report generated by the ipreport command, as follows:

```
IP:     < SRC =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=58, ip_id=13232, ip_off=0
IP:     ip_ttl=58, ip_sum=91a7, ip_p = 6 (TCP)
```

The IP datagram TTL has the following purposes:

- It defines the maximum segment life time for TCP.
- It discards any datagrams in an infinite loop due to misconfigured routing table(s).

The unit of TTL is second. The maximum TTL is 255 seconds or 4 minutes and 15 seconds. The TTL is set by the source system. When the IP datagram is set out, it will be decreased at a router by at least one. If the datagram needed more than one second to pass the router, the actual time (second) should be subtracted from the TTL field. Otherwise, one should be subtracted from the TTL field. Up to now, we have not seen a router which decreases the TTL by more than one second. Then, we can consider that the TTL equals the possible hop count or the number of routers on the way to the destination.

When the TTL reaches 0 at a router, the datagram must be discarded by the router. When a datagram is discarded, the ICMP time exceeded message (TTL equals 0 during transit) will be sent back to the source system from the router.

RFC 1122 Requirement for Internet Hosts, Page 34

A host MUST NOT send a datagram with a Time-to-Live (TTL) value of zero.

A host MUST NOT discard a datagram just because it was received with TTL less than 2.

The above description in the RFC 1122 means that any system (including a router) can receive IP datagrams which have the TTL 0 or 1. But, the system cannot forward these datagrams. If the system is a router, the ICMP time exceeded message will be sent back.

Since TTL value defines the maximum hop count, the initial TTL value defines the datagram reachability. A small TTL may not be enough if the destination is located beyond many routers. Then, the current appropriate initial TTL value is defined in the *RFC 1700 ASSIGNED NUMBERS*. This is the current recommendation and is subject to change in the future.

RFC 1700 ASSIGNED NUMBERS, Page 64

The current recommended default time to live (TTL) for the Internet Protocol (IP) [*45,105*] *is 64.*

Usually TTL is not a big concern when you consider performance improvement. Of course, too small a TTL may cause retransmission, or even an IP datagram to never reach the destination system. Mainly, we can use the TTL for debugging. Also, some tools or commands are designed to use the TTL field. You know that the ping command always sets the TTL to 255. Also, the traceroute command sets the TTL to 1 for the first datagram and then increases it for each successive datagram by one.

5.4.2 TTL Configuration

Currently there is only one TTL (the TTL for IP datagrams). Since an IP datagram can carry more than one transport protocol, our AIX V3.2 allows us to configure the IP datagram TTL for each transport protocol.

TTL is configured with the no command. The no command has some options to configure TTL, as follows:

```
# no -a | grep ttl
                   maxttl = 255
                ipfragttl = 60
                  udp_ttl = 30
                  tcp_ttl = 60
#
```

tcp_ttl

Our AIX allows us to configure the TTL value for the TCP segment. Do not be confused that this is not exactly the TTL for the TCP segment. This is the TTL of an IP datagram which carries a TCP segment. As a result, the TTL of the TCP segment is indirectly defined. The default of this TTL is 60 seconds.

udp_ttl

Our AIX allows us to configure the TTL value for the UDP datagram. Do not confuse this is with the TTL for the UDP datagram. This is the TTL of an IP datagram which carries the UDP datagram. As a result, the TTL of the UDP datagram is indirectly defined. The default of this TTL is 30 seconds.

ipfragttl

Our AIX allows us to configure the TTL value for the IP datagram to reassemble. All fragments derived from one IP datagram must arrive at the destination within this period (TTL). If any of them is lost or extremely delayed (and exceeds this TTL), other received fragments are discarded and the entire set of fragments must be retransmitted. If this happens, the amount of ICMP time exceeded (TTL equals 0 during reassembly) is sent to the source system. The default of this TTL is 60 seconds.

maxttl

This is TTL for the Routing Information Protocol (RIP) packet. When you use dynamic routing with RIP by using a gated or a routed, those daemons periodically exchange RIP packets with each other. This is usually done by broadcast. The default of this TTL is 255 seconds (the maximum).

In usual and almost all environments, we don′t see any motivation or reason to change any of TTLs. Setting a shorter TTL than the default may give you hazardous results because it limits the reachability of the IP datagram. Currently no applications use TTL as a criteria of timeout or some other purposes. At least we don′t know of any such applications. Some debug tools and commands set TTL as they need; they are not regular communication applications. All TTLs can be configured with the no command as follows:

1. Issue the no command with the option (TTL name) you need.

   ```
   # no -o tcp_ttl=30
   #
   ```

2. Confirm if the update was successfully made.

   ```
   # no -o tcp_ttl
   tcp_ttl = 30
   #
   ```

As usual (with the no command), if you need to reserve the update through system reboot, write the command in the startup script /etc/rc.net, as follows:

```
##################################################
# The socket default buffer size (initial advertised TCP window) is being
# set to a default value of 16k (16384). This improves the performance
# for Ethernet and token-ring networks.  Networks with lower bandwidth
# such as SLIP (Serial Line Internet Protocol) and X.25 or higher bandwidth
# such as Serial Optical Link and FDDI would have a different optimum
# buffer size.
# ( OPTIMUM WINDOW = Bandwidth * Round Trip Time )
##################################################
if [ -f /usr/sbin/no ] ; then
        /usr/sbin/no -o tcp_sendspace=16384
        /usr/sbin/no -o tcp_recvspace=16384
        /usr/sbin/no -o tcp_ttl=30
fi
```

5.5 Checksum

Checksum confirms consistency and integrity of the data. Performance and checksum are in trade-off positions because checksum generation and validation add a certain load for both sending and receiving systems. If you could turn the checksum function off, the system performance would be improved (sacrificing data accuracy).

Difference between V4.1 and V3.2

An option to disable checksum when the destination is on the same system was removed at V4.1. With V3.2, we had the following option, loop_check_sum of the no command:

```
# no no -o loop_check_sum
loop_check_sum = 1
#
```

5.5.1 TCP Checksum

Checksum is mandatory for TCP. Both the data and TCP header are included for the checksum calculation. There are no options to turn off the checksum calculation.

```
TCP:    <source port=23(telnet), destination port=1059 >
TCP:    th_seq=a42f623a, th_ack=5d80ec2d
TCP:    th_off=5, flags<PUSH | ACK>
TCP:    th_win=15972, th_sum=414d, th_urp=0
TCP: 00000000     0d0a0d0a 0d0a0d0a 0d0a0d0a 0d0a0d0a     |................|
```

RFC 1122 Requirements for Internet Hosts, Page 86

4.2.2.7 TCP Checksum: RFC-793 Section 3.1

Unlike the UDP checksum (see Section 4.1.3.4), the TCP checksum is never optional. The sender MUST generate it and the receiver MUST check it.

An interesting thing is the concept of the pseudo-header. During the checksum calculation, the pseudo-header is added to the TCP header (this is done both at the sending system and the receiving system). The pseudo-header includes both as a source and a destination IP address. Why is this header added? It is because the TCP header doesn′t contain IP addresses and just includes source and destination port numbers. This means if a TCP segment is delivered to the wrong system (wrong destination IP address), the TCP module on that system could not notice it by looking at the TCP header. Including the IP address information in checksum using the pseudo-header prevents this problem.

If a problem is detected after the checksum calculation (validation) in a receiving system, the TCP segment is silently discarded. Nothing informs the sending system. Notice that ICMP is *not* used.

5.5.2 UDP Checksum

Checksum is optional for UDP. But, our old AIX V3.2 doesn′t allow us to turn off the checksum calculation. Some implementations allow it. Since the IP layer doesn′t provide checksum for the data portion, you had better use the UDP checksum even when you have a choice.

Note: If the data link is Ethernet or token-ring, they have checksum. But, if the data link is SLIP, it doesn′t have checksum and you have nothing to confirm the accuracy of the data.

```
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.192.11 >  (hzname1.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=65, ip_id=5549, ip_off=0
IP:     ip_ttl=30, ip_sum=de19, ip_p = 17 (UDP)
UDP:    <source port=1128, <destination port=53(domain) >
UDP:    [ udp length = 45 | udp checksum = 4750 ]
DNS Packet breakdown:
    QUESTIONS:
        ts.hakozaki.ibm.com, type = A, class = IN
```

RFC 1122 Requirements for Internet Hosts, Page 78

4.1.3.4 UDP Checksums

A host MUST implement the facility to generate and validate UDP checksums. An application MAY optionally be able to control whether a UDP checksum will be generated, but it MUST default to checksumming on.

Since the UDP checksum is optional, receiving UDP datagrams which have 0 in their checksum field just means that the sender doesn′t use checksum. In the case of TCP, they are discarded. In UDP they are *not* considered to be an error.

Note: Old BSD 4.2 had a bug in the UDP checksum algorithm and if the destination system would have a corrected algorithm, it could not communicate.

Both data and UDP header are included for the checksum calculation. Also in TCP, the pseudo-header is used for the same purpose. If a problem is detected after the checksum calculation (validation) in a receiving system, the UDP datagram is silently discarded. Nothing is informed to the sending system. Notice that ICMP is not used.

At the AIX V4.1, a new option, udpcksum, is introduced to the no command. This option allows you to enable and disable the UDP checksum.

```
# no -o udpcksum
udpcksum = 1
#
```

The default is 1 (checksum is enabled). The following is an example when this option is set to 0. Compare this trace output with the previous one.

```
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.192.11 >  (hzname1.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=65, ip_id=6071, ip_off=0
IP:     ip_ttl=30, ip_sum=dc0f, ip_p = 17 (UDP)
UDP:    <source port=1144, <destination port=53(domain) >
UDP:    [ udp length = 45 | udp checksum = 0 ]
DNS Packet breakdown:
    QUESTIONS:
        ts.hakozaki.ibm.com, type = A, class = IN
```

Difference between V4.1 and V3.2

AIX V3.2 doesn′t allow us to turn off the checksum calculation. But, for NFS only, it has an option to disable the checksum.

5.5.3 UDP Checksum for NFS

NFS is one of the most used TCP/IP applications, and performance is a crucial factor to run NFS successfully. AIX provides an option to turn off the UDP checksum for NFS only. You can use the nfso command option, udpchecksum. The nfso command is a NFS version of the no command. The default is 1 (checksum enabled). Use this command as follows:

```
# nfso -o udpchecksum
udpchecksum= 1
# nfso -o udpchecksum=0
# nfso -o udpchecksum
udpchecksum= 0
#
```

Note: Set the option udpchecksum to 0, and you have to accept the risk of data corruption gaining the performance.

The nfso command also doesn′t store the current parameter in a file. You have to write down the above command in a start up script if you need a permanent configuration. Maybe the best place is at the end of the /etc/rc.nfs script.

Difference between V4.1 and V3.2

Although AIX V3.2 doesn′t have a capability to disable UDP checksum, it does provide limited capability to disable UDP checksum for NFS operations. You can use nfso command as well as V4.1. But, the option name is different (nfsudpcksum), as below:

```
# nfso -o nfsudpcksum
nfsudpcksum = 1
# nfso -o nfsudpcksum=0
# nfso -o nfsudpcksum
nfsudpcksum = 0
#
```

Note: You may need a PTF in order to get the nfso command for your V3.2.

5.5.4 IP Checksum

Checksum is mandatory for IP. The checksum calculation is only applied to the IP header. Data portion (including TCP or UDP header) is left for the upper layer validation function, because the IP layer would deliver various kinds of data and the priority of data accuracy and speed depends on the data.

```
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =    9.68.210.140 >  (ts.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=43, ip_id=5555, ip_off=0
IP:     ip_ttl=60, ip_sum=adb3, ip_p = 6 (TCP)
```

Due to TTL field and option fields, IP header is modified when it passes through a router (some options such as, timestamp and record route, add information in the IP header option field). This means that the IP checksum must also be updated. If you have a router which has an incorrect checksum algorithm, your IP datagrams are never received by the destination.

If a problem is detected after the checksum calculation in a receiving system, the IP datagram is silently discarded. Nothing is informed to the sending system. Notice that ICMP is not used.

5.5.5 Optional Address Check by RFC 1122

The *RFC 1122 Requirements for Internet Hosts* defines additional IP layer validation. This validation is optional. If you turn this on, the following information in IP header is validated:

IP protocol version
The currently supported IP protocol version is 4 and any other value is invalid. This check is made for receiving IP datagrams.

TCP and broadcast
If the transport protocol is TCP, the IP datagram carrying the TCP segment should not have a broadcast address for the destination or source (or both). This check is made for both the sending IP datagrams and the receiving IP datagrams.

The source address must be a local interface for sending.
For an IP datagram to be sent, the source address must be one of the local network interfaces of the sending system.

You can enable or disable this additional validation using the no command. The default is 0 (disable). Of course, enabling this feature adds a performance penalty.

1. You can enable this with the following command:

   ```
   # no -o rfc1122addrchk=1
   #
   ```

2. You can confirm that the update was completed as follows:

   ```
   # no -o rfc1122addrchk
   rfc1122addrchk = 1
   #
   ```

As usual, you may need to write the command at the end of the startup script /etc/rc.net.

5.6 Socket Buffer

Usually data communication involves some concept of buffer. In TCP/IP, the most widely used API is socket, and the socket provides buffering capability. Therefore, we call the buffer a socket buffer.

We recommend that you refer to *Performance Monitoring and Tuning Guide*, SC23-2365. Detailed mechanism and considerations are explained in it.

5.6.1 Buffer Basics

The buffer is the place where data is stored temporarily between application and device driver (adapter card) when data is sent or received. This buffer consists of mbufs and mbuf-clusters (memory pages). The send buffer and receive buffer are allocated for each socket. These buffers are placed in the mbuf region (mbuf pool) and totally depend on the mbuf allocation mechanism.

An application can use system calls to put or get data from these buffers. For example, read() and write() are used for TCP, and recvfrom() and sendto() are used for UDP.

Do not confuse the difference between the buffer and TCP window. The window is to control data flow between sender and receiver. The buffer is to make system activity efficient. For our AIX V3.2 and V4.1, the available and not used part of the socket buffer space is used as the window. The buffer is a system mechanism, and you can configure the socket buffer as big as the system resource (memory) allows. On the contrary, the window is the TCP mechanism, and the maximum is defined by TCP as 64 KB.

5.6.1.1 TCP

For TCP, a number of bytes which is specified in read() or write() may not coincide with the actual number of bytes successfully sent or received. In TCP, the data is considered as a stream of bytes and the TCP module doesn′t recognize the boundary of chunks processed by read() or write(). Therefore, you have to add certain logic to see how many bytes were already processed and retry it if necessary.

During transmission, the data is divided into a unit of transmission called a segment and the segment is passed to the IP module. You should not forget that each segment doesn′t match each write() system call. Also, a read() call doesn′t read each segment. A segment is mapped to an IP datagram. If a segment is large enough to exceed the MTU, the segment should be fragmented. Actually, fragmentation can not happen because TCP has a Maximum Segment Size (MSS) mechanism. This mechanism guarantees that the segment size never exceeds the MTU.

TCP has flow-control mechanism called window and a sender can not send a TCP segment faster than the receiver′s receiving speed. Thus, if you assign a large send buffer, it doesn′t harm the receiving system; but, it may waste your system mbuf pool. If you assign a large receive buffer, your system can accept many segments without suppressing the sending system by returning a small window size. As a result, larger buffers for both send and receive can improve performance.

5.6.1.2 UDP

UDP uses a more direct scheme than TCP. Each call, such as sendto() or recvfrom() corresponds to an UDP datagram. A UDP datagram is passed to the IP datagram. If the size of the UDP datagram is larger than the MTU, it is fragmented into more than one IP datagram. The maximum size of a UDP datagram is 65,536 bytes (including the UDP header).

UDP doesn′t have any flow control mechanism. It is left to the application program. If you assign a large send buffer, when the buffer is flushed a burst of packets caused and may load the network and receiving system. In order to take advantage of buffering, you should assign enough size to the receive buffer. Since the sending system doesn′t care about the receiving system, a small receive buffer can not bear a burst of traffic. If a receive buffer runs out, the UDP datagrams are lost. Since retransmission is made at the application layer, the cost of retransmission can not be negligible.

5.6.2 Socket Buffer Pitfalls for TCP

As you already know, buffers are made by a chain of mbufs and mbuf-clusters (memory pages). The criteria for using mbuf-cluster is if the data is bigger than 432 bytes. If you issue write() or sendto() with 433 bytes of data, a mbuf-cluster (4096 bytes) is consumed. Assume you have a 16 KB send buffer and issue 433 bytes of write() four times. From the application point of view, only 1.7 KB are put in the 16 KB buffer space. But, from the memory allocation point of view, 4 memory pages (16 KB) have already been used. If this would be UDP, a successive sendto() operation causes the buffer to flush. But, TCP has a strict window based flow-control and certain criteria should be satisfied before sending the data. Therefore, the successive write() of the application just hangs until the criteria is met.

Actually for TCP, a send buffer is flushed when stored data reaches to 1/2 of the receiver′s window size or the effective MSS (this is TCP specification). This means, for all default token-ring environments, the buffer should be flushed when the data reaches to 1.4 KB (the effective MTU is 1492 bytes and the 1/2 of the window, that is the socket receive buffer, is 8 KB). For the small MTU networks, such as Ethernet or token-ring, this problem may not be visible.

When to Send Data

According to *RFC 1122 Requirements for Internet Hosts* (*4.2.3.4 When to Send Data*), TCP has following criteria to send data. These are the recommended rules in this RFC.

1. When more data than the MSS is buffered in send buffer.
2. When there are no unacknowledged data (this means no data are on the transmission), and if the PUSH flag is specified. In this case, sending data must be smaller than the window size of the receiver.
3. When there is no unacknowledged data (this means no data is on the transmission), and if buffered data exceeds the maximum window size of the receiver × Fs. It is recommend that you use 0.5 for the Fs and AIX V3.2 uses 0.5 for this constant.
4. When the PUSH flag is specified, and if the override timeout timer expires, it is recommended that you use 0.1 - 1.0 seconds for the override timeout period. AIX V3.2 uses 0.1 - 1.2 seconds (the timer sets the value between this range).

You should refer to the original RFC.

This issue appears clearly if you are using a high-speed network. TCP sends data when the data in the send buffer reaches half of the receiver′s window size or the Maximum Segment Size (MSS). If the data link layer is FDDI or SOCC, the default MTU is 4 KB to 61 KB, which defines the effective MSS. Also, the size of the receiver′s window is usually the order of a receiving system′s receive buffer size (default is 16 KB for RS/6000), and half of the window size would be 8 KB. As a result, even the send buffer is used up and data is not flushed immediately because 1.7 KB of the buffer data can not satisfy the sending criteria. The send buffer is occupied and the application can not send the next data and has to wait until the receiving system sends an ACK segment to force the sender to send the data segment. It needs a delayed ACK timer expiration at the receiving system.

Note: The delayed ACK timer needs 200 ms to expire.

Remember when the TCP module calculates a window size, it uses actual data bytes that are specified in the read() and write() system calls. But, when the system calculates the buffer size (usage), it uses actual allocated mbuf/mbuf-cluster size. You may need to configure bigger buffer space than you expected, in order to compensate for this implementation pitfall.

5.6.3 Buffer Size Configuration

There are two ways to configure the buffer size. One is to set the system with a wide default buffer size, and the other is to set each (application specific) buffer size.

5.6.3.1 Application Program Specific Configuration (NFS)

An application program can set its own buffer size by using setsockopt() and can retrieve the current value by using getsockopt() system calls.

If possible, this approach is convenient because the optimum buffer size may be different between each application. Also, if a system is multi-homed, the optimum buffer size may be different for each network interface. This approach allows users to fine tune.

Some applications have options to configure the buffer sizes. For example, NFS uses socket internally. NFS is built on the ONC/RPC and the RPC uses socket. AIX V4.1 provides capability to adjust the socket buffer used for NFS operations. You can use the nfso command for this purpose, as follows:

```
# nfso -a
portcheck= 0
udpchecksum= 1
nfs_socketsize= 60000
nfs_setattr_error= 0
nfs_gather_threshold= 4096
nfs_repeat_messages= 0
nfs_duplicate_cache_size= 0
nfs_server_base_priority= 0
nfs_dynamic_retrans= 1
nfs_iopace_pages= 32
#
```

nfs_socketsiz
: This option defines the size of the socket buffer used for NFS (in bytes). Notice that the current AIX implementation only uses UDP for the NFS data transfer (read and write operation); then, this option defines the UDP socket buffer. This value must be 128 bytes smaller than the value of the sb_max (the sb_max is an option of the no command).

To change the option nfs_socketsize, follow this procedure:

1. Issue the nfso command to set the new value. Before issuing the nfso command, make sure that the sb_max is also configured correctly, as follows:

   ```
   # no -o sb_max=131072
   nfso -o nfs_socketsize=100000
   #
   ```

2. Stop the NFS subservers (daemons), as follows:

   ```
   # stopsrc -g nfs
   0513-044 The stop of the biod Subsystem was completed successfully.
   0513-044 The stop of the nfsd Subsystem was completed successfully.
   0513-044 The stop of the rpc.mountd Subsystem was completed successfully.
   0513-044 The stop of the rpc.statd Subsystem was completed successfully.
   0513-044 The stop of the rpc.lockd Subsystem was completed successfully.
   #
   ```

3. Restart the NFS subservers (daemons) in order to reflect the buffer size update, as follows:

   ```
   # startsrc -g nfs
   0513-059 The biod Subsystem has been started. Subsystem PID is 7918.
   0513-059 The nfsd Subsystem has been started. Subsystem PID is 9967.
   0513-059 The rpc.statd Subsystem has been started. Subsystem PID is 8176.
   0513-059 The rpc.lockd Subsystem has been started. Subsystem PID is 8433.
   0513-059 The rpc.mountd Subsystem has been started. Subsystem PID is 8690.
   #
   ```

4. Confirm if the update has been made correctly, if necessary.

```
# nfso -o nfs_socketsize
nfs_socketsize= 100000
#
```

Difference between V4.1 and V3.2

With V3.2, the nfso command supports less options than V4.1, as shown:

```
# nfso -a
              nfs_portmon = 0
              nfsudpcksum = 1
                nfs_chars = 60000
        nfs_setattr_error = 0
     nfs_gather_threshold = 4096
#
```

Also names of options are different. The name of the option to configure NFS socket buffer was nfs_chars.

You can use the rsize and wsize options of the mount command when you mount a filesystem through NFS. Many documents refer these options to configure the buffer sizes of NFS read and write. Accurately speaking, these options define the maximum sizes of each RPC for read and write. Current AIX NFS implementation allows up to 8 KB for an RPC. Thus, the rsize and wsize can be configured up to 8192 (and this is default value). Reducing the rsize and/or wsize may improve the NFS performance in a congested network.

5.6.3.2 System Wide Default Configuration

In this method, the configured values are applied to all applications which don′t explicitly configure the buffer sizes. You can use the no command. To review the current configuration, use the following command:

```
# no -a | grep space
            tcp_sendspace = 16384
            tcp_recvspace = 16384
            udp_sendspace = 9216
            udp_recvspace = 41600
# no -a | grep sb
                   sb_max = 65536
#
```

tcp_sendspace
This parameter defines the TCP send buffer size. The hard-coded default is 4096 bytes.

tcp_recvspace
This parameter defines the TCP receive buffer size. The hard-coded default is 4096 bytes.

udp_sendspace
This parameter defines the UDP send buffer size. The hard-coded default is 9216 bytes.

udp_recvspace
This parameter defines the UDP receive buffer size. The hard-coded default is 41600 bytes.

sb_max

This parameter defines the upper limit of each buffer size. The default is 65536 or 64 KB. If you make any of the buffer bigger than 64 KB, you have to expand this parameter first. Notice that TCP can support only up to a 64 KB window size. You can override this limit after AIX V3.2.5 with the rfc1323 option. This is the rationale for the default.

If you need to change any of them, issue the no command, as follows:

```
# no -o tcp_recvspace=32768
# no -o tcp_recvspace
tcp_recvspace = 32768
#
```

If you need to the restore default value, you can use no -d, as follows:

```
# no -d tcp_sendspace
# no -o tcp_sendspace
tcp_sendspace = 4096
#
```

As usual, with the no command, this update is not saved for reboot. If you make this permanent, write this command somewhere in a start up script. The best place is the end of /etc/rc.net, and there are already two lines to expand the TCP send and receive buffer as follows:

```
##################################################
# The socket default buffer size (initial advertised TCP window) is being
# set to a default value of 16k (16384). This improves the performance
# for Ethernet and token-ring networks.  Networks with lower bandwidth
# such as SLIP (Serial Line Internet Protocol) and X.25 or higher bandwidth
# such as Serial Optical Link and FDDI would have a different optimum
# buffer size.
# ( OPTIMUM WINDOW = Bandwidth * Round Trip Time )
##################################################
if [ -f /usr/sbin/no ] ; then
        /usr/sbin/no -o tcp_sendspace=16384
        /usr/sbin/no -o tcp_recvspace=16384
fi
```

Although the receive window is not equal to the receive buffer size, when TCP establishes a connection, the receive buffer size is advertised as the received window to the destination system. Therefore, if you need to use the large receive window, you should configure a large receive buffer. Also, read the comments in the previous startup script. Large window (large buffer) is especially crucial for long-delay network. Use the ping command to measure the Round Trip Time (RTT). In the following example, the RTT is 33 ms.

```
# ping -c 1 hal.yamato.ibm.com
PING hal.yamato.ibm.com: (9.68.1.14): 56 data bytes
64 bytes from 9.68.1.14: icmp_seq=0 ttl=251 time=33 ms

----hal.yamato.ibm.com PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 33/33/33 ms
#
```

5.6.4 Configuration Pitfalls

If you expand any of the buffer, you must confirm that it has not exceeded the parameter sb_max or you will have problems. For example, if you set tcp_sendspace or tcp_recvspac bigger than sb_max, you cannot initiate the new TCP connection, such as TELNET, as follows:

```
# no -o sb_max
sb_max = 65536
# no -o tcp_sendspace=70000
# no -o tcp_sendspace
tcp_sendspace = 70000
#
```

Then try TELNET to another system.

```
# tn mat
Trying...
telnet: socket: Invalid argument
tn> q
#
```

This problem can be recovered by resetting the option parameter to the original value.

If you happened to configure the sb_max smaller than a memory page size, 4096 bytes, your system will suffer serious problems. See the example below. Even the no command is no longer usable and as a result, you cannot reset the parameter.

```
# no -o sb_max=4095
# no -o sb_max
0821-055 no: Cannot create a socket. The error number is 74.
#
```

Any other newly initiated TCP/IP communication fails. The following is a case of the ping command:

```
# ping ts
ping: host name ts NOT FOUND
#
```

It appears as if you have a name server problem, or missing routing information, but both are fine.

Difference between V4.1 and V3.2

V3.2 issues a more understandable message:

```
# ping grover
0821-067 ping: The socket creation call failed.:
There is not enough buffer space for the requested socket operation.
#
```

Since AIX V4.1 uses the system resource controller or the srcmstr to control daemons (subsystems). The srcmstr uses sockets to communicate to subsystems, and this problem, of course, impacts the srcmstr functionality. For example, the lssrc command doesn′t work well any more:

```
# lssrc -s portmap
0513-053 The System Resource Controller is experiencing problems with its
socket communications.
If you specified a foreign host, try configuring Internet sockets and
try your command again, otherwise contact System Administration.
#
```

The only way to fix this situation may be to reboot the system.

5.7 TCP Window Size

Here we show the TCP Window mechanism briefly. TCP provides a sliding window mechanism in order to control the data flow between sender and receiver. Basically, the concept of a sliding window is not so difficult, but since TCP was implemented, many modifications (enhancements) have been made to TCP; now the TCP flow control mechanism is not simple. Almost all those modifications are to improve performance or reduce network traffic. Improvement efforts are still continuing and in the future we may even have a more complicated implementation.

If you try to analyze the TCP trace data deeply, you will need a good knowledge of this window mechanism.

5.7.1 TCP Window Basics

TCP provides reliable transmission. The term, reliable, means that TCP has the responsibility to confirm that all data segments are delivered to the destination. If any of the data segments is lost, TCP retransmits it until it succeeds or aborts the connection. In order to achieve reliable transmission, TCP uses the ACK (acknowledge) segment. A sender receives an ACK segment from the receiver for all data it sent. This doesn′t mean that an ACK segment is needed for each data segment. An ACK segment can acknowledge more than one data segments. Since the TCP connection is bidirectional, an ACK segment can carry data if the receiver has some data to send to the sender. This is sometimes called piggy-back.

Note: An ACK segment is returned for a data segment or segments. No ACK segment is returned for an ACK only segment. This means an ACK segment is not reliably transmitted.

The window is a amount of data which a sender can send without receiving any ACK segment. This obviously contributes performance improvement. A receiver always advertises its window size in the TCP header of the ACK segments.

Actually, this window size is an available portion of the TCP receive buffer. Therefore, changing the receive buffer size is indirectly related to changing the window size. A pitfall is that you can change the receive buffer size to larger than 64 KB if you change the sb_max option of the no command, but the TCP window size is never to be larger than 64 KB, because the TCP header has only a 16 bit length for the window field. The unit of window is an octet or eight bits. Nowadays we have alot of high-speed network media and memory for a workstation. The maximum of 64 KB window may not be big enough for such an advanced environment. TCP has been enhanced to support such situations by the *RFC 1323 TCP Extensions for High Performance*. AIX V3.2.5 and later releases *do* support this new enhancement.

There are two technical terms about windows that you may sometimes get confused. Basically, a window is a receiver′s matter, telling how much data the receiver can accept now. But, there is a term send window, which is a sender′s matter. They are regarding the same thing, but one some occasions when the congestion avoidance algorithm is working, they represent different values to each other. For details and an exact definition, refer to the *RFC 793 TRANSMISSION CONTROL PROTOCOL PROTOCOL SPECIFICATION*. The definitions in the RFC are as follows:

Receive Window
: This represents the sequence number that the local TCP is expecting to receive.

Send window
: This represents the sequence number which the remote (receiving) TCP is willing to receive. It is the value of the window field specified in the segments from the remote (data receiving) TCP.

If you want to briefly review the window mechanism working, you can use the IP trace. If you need to review the detailed window mechanism in action, you can use the socket-level trace.

5.7.2 Window in Action (IP Trace Example)

The following is an example of a file transfer session by FTP. The system inoki5 is a server and newton is a client. A file was transferred from inoki5 to newton. In this example, we set both send and receive buffer size of both inoki5 and newton to 16384 bytes.

1. This is an ACK segment from the newton to inoki5 advertising that the newton currently has 14000 bytes of receive window. This implies that inoki5 can send up to 14000 bytes of data without receiving the next ACK segment from the newton.

```
Packet Number 34
...
IP:     < SRC =      9.170.5.45 >  (newton.fscjapan.ibm.com)
IP:     < DST =     9.170.5.240 >  (inoki5.fscjapan.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=46162, ip_off=0
IP:     ip_ttl=60, ip_sum=ac0d, ip_p = 6 (TCP)
TCP:    <source port=1029, destination port=20(ftp-data) >
TCP:    th_seq=212a3c02, th_ack=beb2b10e
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=14000, th_sum=89ad, th_urp=0
```

2. Now inoki5 sends the data segment. This segment has the sequence number th_seq=beb2b10e and this coincides with the ACK number th_ack=beb2b10e of the prior segment from the newton. Also the ACK number th_ack=212a3c02 coincides with the sequence number th_seq=212a3c02 of the prior segment from newton. All show that the window mechanism (sequence and ACK numbers) is working correctly.

   Be aware that the sender inoki5 is advertising its receive window size is th_win=15972. If the newton has data to send, it can send up to 15K bytes without ACK from the inoki5.

```
Packet Number 35
...
IP:     < SRC =     9.170.5.240 >  (inoki5.fscjapan.ibm.com)
IP:     < DST =      9.170.5.45 >  (newton.fscjapan.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=1492, ip_id=5305, ip_off=0
IP:     ip_ttl=60, ip_sum=5ac, ip_p = 6 (TCP)
TCP:    <source port=20(ftp-data), destination port=1029 >
TCP:    th_seq=beb2b10e, th_ack=212a3c02
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=15972, th_sum=0, th_urp=0
TCP: 00000000     70726573 656e7461 74696f6e 20626574     |presentation bet|
TCP: 00000010     7765656e 0d0a6170 706c6963 6174696f     |ween..applicatio|
...
```

3. The inoki5 sends data segments continuously without receiving the ACK segment from the newton. This is the advantage of the window mechanism. Now the sequence number is incremented to th_seq=beb2b6ba. The increment is 1452 bytes and this means 1452 bytes of data were transferred. Notice that the prior packet (35) has ip_len=1492, and consider that the IP header and TCP header have 20 bytes each. Although this segment has an ACK flag, no data has come from the newton and the ACK field stays the same th_ack=212a3c02 value.

```
Packet Number 36
...
IP:     < SRC =     9.170.5.240 >  (inoki5.fscjapan.ibm.com)
IP:     < DST =      9.170.5.45 >  (newton.fscjapan.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=1492, ip_id=5306, ip_off=0
IP:     ip_ttl=60, ip_sum=5ac, ip_p = 6 (TCP)
TCP:    <source port=20(ftp-data), destination port=1029 >
TCP:    th_seq=beb2b6ba, th_ack=212a3c02
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=15972, th_sum=0, th_urp=0
TCP: 00000000     6e732061 72652073 616d6520 77697468     |ns are same with|
TCP: 00000010     206f7468 65722077 6f726b73 74617469     | other workstati|
...
```

4. This TCP segment is the same as the previous:

```
Packet Number 37
...
IP:     < SRC =     9.170.5.240 >  (inoki5.fscjapan.ibm.com)
IP:     < DST =      9.170.5.45 >  (newton.fscjapan.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=1492, ip_id=5307, ip_off=0
IP:     ip_ttl=60, ip_sum=5ac, ip_p = 6 (TCP)
TCP:    <source port=20(ftp-data), destination port=1029 >
TCP:    th_seq=beb2bc66, th_ack=212a3c02
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=15972, th_sum=0, th_urp=0
TCP: 00000000     696e6420 6f662062 6f6d622e 0d0a3a65     |ind of bomb...:e|
TCP: 00000010     6c626c62 6f782e0d 0a3a702e 0d0a596f     |lblbox...:p...Yo|
...
```

5. This TCP segment is the same as the previous:

```
Packet Number 38
...
IP:     < SRC =     9.170.5.240 >  (inoki5.fscjapan.ibm.com)
IP:     < DST =      9.170.5.45 >  (newton.fscjapan.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=1492, ip_id=5308, ip_off=0
IP:     ip_ttl=60, ip_sum=5ac, ip_p = 6 (TCP)
TCP:    <source port=20(ftp-data), destination port=1029 >
TCP:    th_seq=beb2c212, th_ack=212a3c02
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=15972, th_sum=0, th_urp=0
TCP: 00000000     69740d0a 796f7572 206e6574 776f726b     |it..your network|
TCP: 00000010     2e0d0a54 68656e20 63686563 6b207768     |...Then check wh|
...
```

6. This TCP segment is the same as the previous:

```
Packet Number 39
...
IP:     < SRC =     9.170.5.240 >  (inoki5.fscjapan.ibm.com)
IP:     < DST =      9.170.5.45 >  (newton.fscjapan.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=1492, ip_id=5309, ip_off=0
IP:     ip_ttl=60, ip_sum=5ac, ip_p = 6 (TCP)
TCP:    <source port=20(ftp-data), destination port=1029 >
TCP:    th_seq=beb2c7be, th_ack=212a3c02
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=15972, th_sum=0, th_urp=0
TCP: 00000000     20202020 20202053 4f465457 41524520     |       SOFTWARE |
TCP: 00000010     50524f47 52414d20 4552524f 520d0a32     |PROGRAM ERROR..2|
...
```

7. Now the receiver newton returns the ACK segment. Since the newton received 1452 bytes of data five times, the window size is reduced to th_win=6740. The receiver sent only one ACK segment for five data segments. This is an advantage of the sliding window.

   Notice the ACK field th_ack=beb2cd6a is 1452 bytes greater than the sequence number of the prior segment (Packet Number 39) form the inoki5, th_seq=beb2c7be. This means all of the data sent from the inoki5 was acknowledged. Be aware that this does not always happen. Some data segments might be on the way and not yet received by the newton. In such a case the ACK field would show different values, but it would not be a problem.

```
Packet Number 40
...
IP:     < SRC =      9.170.5.45 >  (newton.fscjapan.ibm.com)
IP:     < DST =     9.170.5.240 >  (inoki5.fscjapan.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=46163, ip_off=0
IP:     ip_ttl=60, ip_sum=ac0c, ip_p = 6 (TCP)
TCP:    <source port=1029, destination port=20(ftp-data) >
TCP:    th_seq=212a3c02, th_ack=beb2cd6a
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=6740, th_sum=89ad, th_urp=0
```

8. This TCP segment is not of a control connection. Notice the that source port is 1028 and the destination port is 21. In this case, just ignore this packet.

```
Packet Number 41
...
IP:     < SRC =      9.170.5.45 >  (newton.fscjapan.ibm.com)
IP:     < DST =     9.170.5.240 >  (inoki5.fscjapan.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=46164, ip_off=0
IP:     ip_ttl=60, ip_sum=ac0b, ip_p = 6 (TCP)
TCP:    <source port=1028, destination port=21(ftp) >
TCP:    th_seq=210ee46a, th_ack=be972705
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=15972, th_sum=63d1, th_urp=0
```

9. The following is the inoki5 starting data transfer again.

```
Packet Number 42
...
IP:     < SRC =     9.170.5.240 >  (inoki5.fscjapan.ibm.com)
IP:     < DST =      9.170.5.45 >  (newton.fscjapan.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=1492, ip_id=5310, ip_off=0
IP:     ip_ttl=60, ip_sum=5ac, ip_p = 6 (TCP)
TCP:    <source port=20(ftp-data), destination port=1029 >
TCP:    th_seq=beb2cd6a, th_ack=212a3c02
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=15972, th_sum=0, th_urp=0
TCP: 00000000     302c2074 68656e20 796f7572 0d0a4574     |0, then your..Et|
TCP: 00000010     6865726e 65742061 64617074 6572206e     |hernet adapter n|
...
```

Note: In this example, some segments have the TCP checksum field of th_sum=0:. Since TCP checksum is a mandatory feature, this could not happen. So, we conclude that this is a bug of iptrace or ipreport command, not a bug of the TCP module, because any segment, which has the wrong TCP checksum, must be discarded; and, if a 0 was really written in the TCP header, we would have a lot of connectivity problems.

5.7.3 Actual Implementation Complexity

From the previous explanation, you may have thought that the sliding window is so simple. The previous explanation doesn′t cover many irregular situations, for example, if the second or third of five data segments was lost, how should the receiver behave? In the previous example, the receiver responded to the ACK after receiving five data segments. But, why was it after five data segments? Describing these would exceed the scope of this book. Also, up to now, many improvements have been made to the TCP sliding window mechanism. It is much more complicated. Those are following:

Silly Window Syndrome Avoidance Algorithm
The Silly Window Syndrome (SWS) is caused, when a large amount of data is transmitted. If the sender and receiver don′t implement this algorithm, the receiver advertises a small amount of the window each time when the receiver buffer is read by an application. As a result, the sender has to send a lot of small segments and they do not have the advantage of bulk data transfer; that is the purpose of the window mechanism. This algorithm is mandatory (a *must*) by the RFC 1122.

Delayed ACKs
TCP should not send back an ACK segment immediately because there won′t be any data soon which can be sent with the ACK. As a result, the network traffic and protocol module overhead will be reduced. With

this mechanism, an ACK segment is not returned immediately. This mechanism is optional (strongly recommended) by the RFC 1122.

Nagle Algorithm
When an application issues many write() system calls with a single byte of data or so, TCP should not send data segments just carrying a single byte of data. In order to avoid this inefficiency, data should not be sent until the ACK of the prior data segment is received. This mechanism makes small segments accumulated into one big (reasonable amount) segment before it is sent out. This mechanism is optional (strongly recommended) by the RFC 1122.

Note: Certain applications, such as X-Windows, don′t work fine with this mechanism. Thus, there must be an option to disable this feature. You can use the TCP_NODELAY option of setsockopt() call.

Congestion Avoidance
When a segment is lost, the sender′s TCP module considers that this is due to the congestion of the network and reduces the send window size by a factor of 2. If the segment loss continues, the sender′s TCP module keeps reducing the send window using the previous procedure until it reaches 1. This mechanism is mandatory by the RFC 1122.

Slow Start
If a network congestion is resolved, the minimized send window should be recovered. The recovery should not be the opposite of shrinking (exponential backoff). This mechanism defines how to recover the send window. This mechanism is mandatory by the RFC 1122.

Our AIX V3.2 and V4.1 support all of above mechanisms.

Our Experience

We received a question from a customer. They were using a RS/6000 as a gateway between WAN (9.6 Kbps SNA) and LAN (Ethernet). The RS/6000 is a protocol gateway. They got a network trace by a protocol analyzer to tune their performance. They expected that the incoming WAN packets were immediately sent out to the Ethernet. But, actually packets were stored in the RS/6000 until the ACK of the prior segment was received. Their question was why the RS/6000 behaves in such way.

We read their trace data and concluded that they were watching the Nagle Algorithm in action. In this case, since each WAN packet arrives at the RS/6000 with a bit of a long interval (around 0.2 second), they could not see the accumulation effect of multiple packets. ACK was returned within 0.2 seconds from Ethernet before the second WAN packet arrived.

We believe that the commercial books and RFCs will help you. As we saw in the example, we can only review window size (th_win), sequence number (th_seq) and acknowledge number (th_ack) with the IP trace because the TCP header only contains this information. But, the TCP module (kernel) keeps more sophisticated information. It is defined in the header file /usr/include/netinet/tcp_var.h. For the complete list of this header file, refer to C.3, “/usr/include/netinet/tcp_var.h” on page 369.

```
/*
 * The following fields are used as in the protocol specification.
 * See RFC783, Dec. 1981, page 21.
 */
/* send sequence variables */
        tcp_seq snd_una;               /* send unacknowledged */
        tcp_seq snd_nxt;               /* send next */
        tcp_seq snd_up;                /* send urgent pointer */
        tcp_seq snd_wl1;               /* window update seg seq number */
        tcp_seq snd_wl2;               /* window update seg ack number */
        tcp_seq iss;                   /* initial send sequence number */
        u_long  snd_wnd;               /* send window */
/* receive sequence variables */
        u_long  rcv_wnd;               /* receive window */
        tcp_seq rcv_nxt;               /* receive next */
        tcp_seq rcv_up;                /* receive urgent pointer */
        tcp_seq irs;                   /* initial receive sequence number */
...
/*
 * Additional variables for this implementation.
 */
/* receive variables */
        tcp_seq rcv_adv;               /* advertised window */
/* retransmit variables */
        tcp_seq snd_max;               /* highest sequence number sent;
                                        * used to recognize retransmits
                                        */
/* congestion control (for slow start, source quench, retransmit after loss) */
        u_long  snd_cwnd;              /* congestion-controlled window */
        u_long snd_ssthresh;           /* snd_cwnd size threshhold for
                                        * slow start exponential to
                                        * linear switch
                                        */
```

If you want to review these kernel parameters during a communication, the socket-level trace function will help you. The following is a sample output:

```
488 ESTABLISHED:output (src=9.170.5.45,1336, dst=9.170.5.21,23)
        [157ac830..157ac831)@601b54c8(win=3e64)<ACK,PUSH> -> ESTABLISHED
        rcv_nxt 601b54c8 rcv_wnd 3e64 snd_una 157ac830
        snd_nxt 157ac831 snd_max 157ac831
        snd_wl1 601b54c7 snd_wl2 157ac830 snd_wnd 3e64
        REXMT=3 (t_rxtshft=0), KEEP=14400
488 ESTABLISHED:user SEND -> ESTABLISHED
        rcv_nxt 601b54c8 rcv_wnd 3e64 snd_una 157ac830
        snd_nxt 157ac831 snd_max 157ac831
        snd_wl1 601b54c7 snd_wl2 157ac830 snd_wnd 3e64
        REXMT=3 (t_rxtshft=0), KEEP=14400
488 ESTABLISHED:input (src=9.170.5.21,23, dst=9.170.5.45,1336)
        [601b54c8..601b54c9)@157ac831(win=3e64)<ACK,PUSH> -> ESTABLISHED
        rcv_nxt 601b54c9 rcv_wnd 3e64 snd_una 157ac831
        snd_nxt 157ac831 snd_max 157ac831
        snd_wl1 601b54c8 snd_wl2 157ac831 snd_wnd 3e64
        KEEP=14400
489 ESTABLISHED:user RCVD -> ESTABLISHED
        rcv_nxt 601b54c9 rcv_wnd 3e64 snd_una 157ac831
        snd_nxt 157ac831 snd_max 157ac831
        snd_wl1 601b54c8 snd_wl2 157ac831 snd_wnd 3e64
        KEEP=14400
```

If a sender is faster than the receiver, the receiver system sends back an ACK segment with a 0 window size. When the sender receives this ACK segment, it cannot send until the receiver system advertises more than a 0 window size by sending another ACK segment. This ACK segment is called the window update. Since the ACK segment is not reliably transmitted, the window update ACK segment may be lost. If this happened, the sender would never know that it can send the next data, and then the connection hangs indefinitely. In order to avoid this problem, a sender periodically sends a probe. If a receiver doesn′t respond to a probe, the sender repeatedly sends a probe after the timeout period. This timeout is called the persistent timeout. The sender system never gives up sending probes. There is no final timeout or connection abortion due to window probe activity.

5.7.4 Getting Window Status

Since the TCP sliding window is a dynamic mechanism and only trace (IP trace or socket level trace) reveals the details. Some activities are also counted and display the netstat -p tcp command. The following counters show the TCP statistics:

```
# LANG=C netstat -p tcp
tcp:
        111905 packets sent
                71310 data packets (235937 bytes)
                0 data packets (0 bytes) retransmitted
                40449 ack-only packets (40380 delayed)
                0 URG only packets
                0 window probe packets
                51 window update packets
                112 control packets
        80042 packets received
                71326 acks (for 235974 bytes)
                8 duplicate acks
                0 acks for unsent data
                74479 packets (2885666 bytes) received in-sequence
                2 completely duplicate packets (122 bytes)
                1 packet with some dup. data (98 bytes duped)
                1 out-of-order packet (0 bytes)
                0 packets (0 bytes) of data after window
                0 window probes
                0 window update packets
                2 packets received after close
                0 discarded for bad checksums
                0 discarded for bad header offset fields
                0 discarded because packet too short
        61 connection requests
        0 connection accepts
        39 connections established (including accepts)
        70 connections closed (including 0 drops)
        4 embryonic connections dropped
        71351 segments updated rtt (of 71382 attempts)
        14 retransmit timeouts
                0 connections dropped by rexmit timeout
        0 persist timeouts
        1 keepalive timeout
                0 keepalive probes sent
                0 connections dropped by keepalive
#
```

Refer to 3.5.1, “TCP Segment Statistics” on page 214 for details of each counters.

5.8 TCP/IP and Timeout

There are several timeouts in the TCP/IP world. Usually the adjustments of timeouts are crucial for performance tuning. Unfortunately, many TCP/IP parameters (including the timeout period) are fixed and there is no way to tune. This section deals mainly with TCP timeouts. The mechanisms and effects of those timeouts are covered next.

5.8.1 Timeout Basics

When a destination system doesn′t respond, the source system should take an action (for example, a retransmission or abortion). There are many reasons for no-responses. For example, a data packet could not reach to the destination, a destination system just delayed to respond due to heavy load, and the ACK packet was lost during the transmission, or the system really crashed. With the current communication technology, a source system cannot know what was happening. A timeout usually causes retransmission, but this may not be a correct action. A retransmission is only effective when a data packet was lost during the transmission. Be aware that retransmission after timeouts may not be a good approach for all occasions (sometimes it makes the situation worse).

There should be criteria to decide if a source system takes some recovery action. The timeout is for this purpose. If a response doesn′t come in within the timeout period, a source system decides that a problem has occurred and takes some recovery action (usually retransmits the segment again).

The concept of a timeout can be applied to any layer of the network except the physical layer. But, a timeout can only be implemented to connection oriented protocols because the protocol module has to keep the state of every packet transmission in order to detect the timeout. In the TCP/IP stack, IP and UDP are connectionless protocols. Only TCP is a connection oriented protocol. Thus, only TCP has a built in timeout mechanism. The TCP module is responsible for detecting a timeout and retransmission. The application program using TCP is free from implementing the timeout mechanism.

For an application using UDP, if the application requires reliable transmission, the application should implement timeout and retransmission mechanism. Well designed applications are actually doing this. NFS and DNS may be good examples. If the application doesn′t have a timeout mechanism, there are no timeout tuning tasks.

The tuning of the timeout period has the following two aspects or objectives:

1. If packets are lost during transmission, choosing a smaller timeout period makes it easy to detect problems and the source system can take a recovery action quickly. As a result, it improves performance.
2. If packets are delayed due to the destination system problem (maybe a heavy load), choosing a larger timeout period suppresses the unnecessary recovery action (retransmissions). As a result, it reduces the network and destination system loads.

You cannot satisfy both of the previous two objectives. For TCP, almost all the parameters which affect the timeout period are fixed or automatically adjusted by the TCP module. Currently, AIX V3.2 TCP has the following timeout mechanisms.

Connection Establishment Timeout

This timeout is for a TCP connection establishment sequence. Internally, the keep-alive timer is used. The final timeout (timer expiration) invites the abortion of the connection establishment sequence.

You can adjust the final timeout time with the no command.

Retransmission Timeout (RTO)

This timeout is for each TCP data segment before the sender receives the ACK segment. Internally, the retransmission timer is used. The final timeout (after 12 successive retransmission failures) invites the abortion of the connection. Each retransmission timeout time is dynamically adjusted, so the final timeout is not a fixed value (but it is usually around 10 minutes).

You cannot adjust this timer with AIX V3.2, but you can adjust this timer with AIX V4.1.

Keep-Alive Timeout

This timeout is used to start the keep-alive procedure. Internally, the keep-alive timer is used. When this timer expires, the keep-alive procedure is initiated. The final timeout (timer expiration) invites the abortion of the connection.

You can adjust the final timeout time with the no command.

Persist Timeout

This timeout is used to send a window probe segment. Internally, the persist timer is used. There is no final timeout or timer expiration. Until the response is received, the window probe segments are sent continuously at each timer expiration.

You *can not* adjust this timer.

FIN_STATE_2 Timeout

This timeout is to terminate a TCP connection which is suspended during the termination procedure. When a system sends a FIN segment and receives the ACK, the system gets in the FIN_STATE_2 state. Now the system is waiting for a FIN segment from the other end. If the other system will not send the FIN segment, the system will be in the FIN_STATE_2 forever. To avoid this situation, this timer is implemented. This timer expires after 10 minutes and 75 seconds (in the case of AIX V3.2) and forces the connection to the closed state.

Note: Although many systems implement this timer, this timer is violating the TCP protocol specification.

You cannot adjust this timer.

2MSL (TIME_WAIT) Timeout

After a TCP connection is closed, the port cannot be reused for a 2MSL period. Internally, the 2MSL timer is used. This timer defines 2MSL (60 seconds for AIX V3.2), and after the expiration, the port can be reused.

You cannot adjust this timer, but you can override this timer to reuse the port in certain occasions.

Delayed ACK′ Timeout

The ACK segments are delayed until this timer expires before it is sent back. This mechanism reduces network traffic because an ACK segment can be merged with a data segment going the same direction, which is generated by the timeout.

You cannot adjust this timer, and it is fixed to 200 ms.

5.8.2 TCP Timeout at Connection Establishment

When a system tries to open a TCP connection to another system by sending a SYN segment, a problem may occur. In the case of a daemon or an application program on the destination system not running, the TCP connection request is immediately reset because the TCP module of the destination system sends back an RST segment. But, if the destination system is not turned on, nobody can respond. For that case, a retransmission of the SYN segment is made six times before the timer expires. This is the implementation example of our AIX V3.2. As a result, in the case of TELNET, you would see the following error message.

```
# tn mat
Trying...
telnet: connect: Connection timed out
#
```

This behavior is to comply with the RFC 1122 although this is not mandatory requirement.

RFC 1122 Requirements for Internet Hosts, Page 95

4.2.3.1 Retransmission Timeout Calculation

A host TCP MUST implement Karn′s algorithm and Jacobson′s algorithm for computing the retransmission timeout (″RTO″)... This implementation also MUST include ″exponential backoff″ for successive RTO values for the same segment. Retransmission of SYN segments SHOULD use the same algorithm as data segments.

During this period, the TCP module is retransmitting the SYN segment using the exponential backoff algorithm. In this algorithm, the timeout period is doubled at each retransmission. If the initial timeout period is 5 seconds, the timeout period after the first retransmission would be 10 seconds. The successive timeout periods would be 20, 40, 80... seconds. But, the total retransmission period is fixed to 75 seconds. These retransmissions, based on the exponential backoff, continue for 75 seconds. So you should see the previous TELNET message 75 seconds after you have invoked the telnet command. Each retransmission is governed by the retransmission timer. But, the 75 seconds is governed by the keep-alive timer. In other words, the keep-alive timer defines the final TCP connection establishment timeout for abortion.

The following table shows the actual measured timeout related parameters. Notice that this is about the timeout of the first segment, and there are no measured RTTs (Round Trip Time). Then the initial REXMIT is set to 2 (1 second).

Note: The actual measured first timeout was invokded at 0.58 seconds. This is because all TCP timers are driven by the clock tick of 500 mseconds called slow timeout and defined as PR_SLOWHZ in the header file /usr/include/sys/protosw.h. Therefore, a deviation within 0.5 seconds is not a problem. This is a common characteristic of the BSD derivative system.

This is defined in the header file /usr/include/netinet/tcp_timer.h, as TCPTV_MIN. After the first timeout, REXMIT was incremented to 6. Because there were no calculated smoothed RTT or SRTT (this should be deduced from the measured RTT), the default SRTT was used. This default is also defined in the same header file as TCPTV_SRTTDFLT.

No	Time Stamp	shift	REXMIT	REXMIT(real)	KEEP
1	22:14:32.709636224	0	2	0.579539456	150
2	22:14:33.289175680	1	6	3.000247168	149
3	22:14:36.289422848	2	12	6.000443248	143
4	22:14:42.289860096	3	24	12.000883200	131
5	22:14:54.290743296	4	48	23.001226240	107
6	22:15:18.291969536	5	96	N/A	59

No = Packet Number in the IP trace report.
shift = Number of exponential backoff algorithm applied.
REXMIT = Retransmissions Timer Value (unit of 0.5 sec)
REXMIT(real) = Measured Retransmissions Time (unit of sec)
KEEP = Keep Alive Timer Value (unit of 0.5 sec)

While the TCP module tried to retransmit the SYN segment using the exponential backoff (this was clearly logged), the keep-alive timer value (KEEP), shown on the right most column, was being decremented. When it reaches 0, the connection (not established yet) is aborted. Unfortunately, this abortion was not logged by the IP trace or the socket-level trace (because it didn′t send any segments or cause socket level activity).

5.8.2.1 Socket-Level/IP Trace Example

The following is an example of a socket-level trace log during the retransmissions. The previous table is based on this trace log.

```
5a6aa00:
270 SYN_SENT:output (src=9.68.214.82,1096, dst=9.68.214.84,23)
        [d7fe5801..d7fe5805)@0(win=4000)<SYN> -> SYN_SENT
        REXMT=2 (t_rxtshft=0), KEEP=150
270 CLOSED:user CONNECT -> SYN_SENT
        REXMT=2 (t_rxtshft=0), KEEP=150
328 SYN_SENT:output (src=9.68.214.82,1096, dst=9.68.214.84,23)
        [d7fe5801..d7fe5805)@0(win=4000)<SYN> -> SYN_SENT
        REXMT=6 (t_rxtshft=1), KEEP=149
328 SYN_SENT:user SLOWTIMO<REXMT> -> SYN_SENT
        REXMT=6 (t_rxtshft=1), KEEP=149
628 SYN_SENT:output (src=9.68.214.82,1096, dst=9.68.214.84,23)
        [d7fe5801..d7fe5805)@0(win=4000)<SYN> -> SYN_SENT
        REXMT=12 (t_rxtshft=2), KEEP=143
628 SYN_SENT:user SLOWTIMO<REXMT> -> SYN_SENT
        REXMT=12 (t_rxtshft=2), KEEP=143
228 SYN_SENT:output (src=9.68.214.82,1096, dst=9.68.214.84,23)
        [d7fe5801..d7fe5805)@0(win=4000)<SYN> -> SYN_SENT
        REXMT=24 (t_rxtshft=3), KEEP=131
228 SYN_SENT:user SLOWTIMO<REXMT> -> SYN_SENT
```

```
        REXMT=24 (t_rxtshft=3), KEEP=131
429 SYN_SENT:output (src=9.68.214.82,1096, dst=9.68.214.84,23)
        [d7fe5801..d7fe5805)@0(win=4000)<SYN> -> SYN_SENT
        REXMT=48 (t_rxtshft=4), KEEP=107
429 SYN_SENT:user SLOWTIMO<REXMT> -> SYN_SENT
        REXMT=48 (t_rxtshft=4), KEEP=107
829 SYN_SENT:output (src=9.68.214.82,1096, dst=9.68.214.84,23)
        [d7fe5801..d7fe5805)@0(win=4000)<SYN> -> SYN_SENT
        REXMT=96 (t_rxtshft=5), KEEP=59
829 SYN_SENT:user SLOWTIMO<REXMT> -> SYN_SENT
        REXMT=96 (t_rxtshft=5), KEEP=59
```

The following packet trace was used to build the table. If you compare the timestamps of each packet, you can understand which packet you are looking at. Although, totally, six packets (TCP SYN segments) were transmitted, all the packets were completely identical to each other, so we just show you the first two packets.

1. This is the first packet.

```
Packet Number 1
TOK: ====( 66 bytes transmitted on interface tr0 )==== 22:14:32.709636224
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=7101, ip_off=0
IP:     ip_ttl=60, ip_sum=a3e0, ip_p = 6 (TCP)
TCP:    <source port=1096, destination port=23(telnet) >
TCP:    th_seq=d7fe5801, th_ack=0
TCP:    th_off=6, flags<SYN>
TCP:    th_win=16384, th_sum=64a0, th_urp=0
TCP: 00000000     020405ac                                 |....            |
```

2. This is the second packet and the first retransmitted packet. There are no differences from the first packet except the timestamp. The first retransmission was invoked after about 0.58 seconds.

```
Packet Number 2
TOK: ====( 66 bytes transmitted on interface tr0 )==== 22:14:33.289175680
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=7102, ip_off=0
IP:     ip_ttl=60, ip_sum=a3df, ip_p = 6 (TCP)
TCP:    <source port=1096, destination port=23(telnet) >
TCP:    th_seq=d7fe5801, th_ack=0
TCP:    th_off=6, flags<SYN>
TCP:    th_win=16384, th_sum=64a0, th_urp=0
TCP: 00000000     020405ac                                 |....            |
```

Be aware that the socket trace log seems to be clear evidence of the retransmission activity. Actually, in this example, the TCP module really sent out SYN segments six times. But this doesn′t mean that the corresponding packets were transferred on the physical cable, even when retransmissions were recorded in the socket-level trace log. If a destination system has not been turned on, it can never respond to an ARP request. Then, your system cannot get the destination MAC address, and as a result, an IP datagram (so TCP segment) cannot be sent out. But, the TCP module doesn′t notice that the data link layer is struggling with the ARP retransmissions. If you try to capture packets with an IP trace, you might not capture anything except for the ARP request packets.

5.8.2.2 Statistics Example by the netstat Command

We can use the command, netstat -p tcp, to review these timeouts. The previous activity is counted as follows:

```
# netstat -p tcp
...
        430 connection requests
        139 connection accepts
        516 connections established (including accepts)
        971 connections closed (including 9 drops)
        20 embryonic connections dropped
        576696 segments updated rtt (of 576800 attempts)
        84 retransmit timeouts
                1 connection dropped by rexmit timeout
        0 persist timeouts
        27304 keepalive timeouts
                88 keepalive probes sent
                5 connections dropped by keepalive
#
```

This output was taken at the telnet client system, mat. As already mentioned, this timeout was invoked by the keep-alive timer; the counters, keepalive timeouts and connections, dropped by keepalive, were incremented by one. Also notice that the other connection related counters are not affected.

Note: Remember, if a destination system is up, but a daemon (application) is not running, the connection request is returned with the RST segment. In such a case, it is counted at the counter and the connections are closed.

5.8.2.3 Parameter Configuration

Up to V3.2.5, we could do nothing with the final timeout of 75 seconds. Although we mentioned that the keep-alive timer is used for this timeout, we could not change this value with tcp_keepidle and tcp_keepintvl of the no command. Now we have a new option, tcp_keepinit, to keep-alive mechanism as follows:

```
# no -a | grep keep
            tcp_keepintvl = 150
             tcp_keepidle = 14400
             tcp_keepinit = 150
#
```

tcp_keepinit

This option is to configure TCPTV_KEEP_INIT, which defines the timeout period for the connection establishment.

You can specify this timeout value in units of 0.5 seconds. The default is 150 (75 seconds). If your destination system has a problem (for example, it is turned off), you have to waste more time to get a timeout with a longer timeout value. But, networks can be suddenly congested. Remember, too short a timeout value invites an unstable environment.

Note: You may need a PTF to get this new option with V3.2.5.

5.8.3 TCP Retransmission Timeout (RTO)

Here we explain the retransmission timeout or ACK timeout. Since TCP guarantees reliable transmission, any data sent out should be acknowledged by the destination. This doesn′t mean each data segment must be acknowledged individually. If a source system doesn′t receive an ACK segment within the retransmission timeout period, the source decides that the data segment was lost and retransmits it again. Then, how long is the retransmission timeout? As one of the sophisticated features of TCP, this timeout period is dynamically adjusted and is not a fixed value.

The current retransmission timeout value is calculated based on both the measured (actual) round-trip time and the estimated (smoothed) round-trip time of the prior segment. So, the latest timeout value dynamically reflects the current network status. When you use the slow network such as WAN, TCP adjusts the timeout value to be long. If the network suddenly gets congested, TCP also makes it longer. This mechanism was a mandatory requirement defined by the RFC 1122.

RFC 1122 Requirements for Internet Hosts, Page 95

4.2.3.1 Retransmission Timeout Calculation

A host TCP MUST implement Karn′s algorithm and Jacobson′s algorithm for computing the retransmission timeout (″RTO″).

- *Jacobsen′s algorithm for computing the smoothed round-trip (″RTT″) time incorporates a simple measure of the variance [TCP:6].*
- *Karn′s algorithm for selecting RTT measurements ensures that ambiguous round-trip times will not corrupt the calculation of the smoothed round-trip time [TCP:6].*

This implementation also MUST include ″exponential backoff″ for successive RTO values for the same segment. ...

This calculation and adjustment algorithm is well described in a lot of commercial books. Our recommendation is the following:

Internetworking With TCP/IP Vol I: Principles, Protocols, and Architecture

by Douglas E. Comer, Prentice Hall.

TCP/IP Illustrated, Volume 1 The Protocols

by W. Richard Stevens

Also, header files are good references. The following is quoted from the file /usr/include/netinet/tcp_timer.h. Refer to C.2, “/usr/include/netinet/tcp_timer.h” on page 367 for the complete list of this header file:

```
/*
 * The TCPT_REXMT timer is used to force retransmissions.
 * The TCP has the TCPT_REXMT timer set whenever segments
 * have been sent for which ACKs are expected but not yet
 * received.  If an ACK is received which advances tp->snd_una,
 * then the retransmit timer is cleared (if there are no more
 * outstanding segments) or reset to the base value (if there
 * are more ACKs expected).  Whenever the retransmit timer goes off,
 * we retransmit one unacknowledged segment, and do a backoff
 * on the retransmit timer.
 *
```

Another header file, /usr/include/netinet/tcp_var.h, helps you with the backoff algorithm. Refer to C.3, "/usr/include/netinet/tcp_var.h" on page 369 for the complete list of this header file. The key point is that an actual measured RTT and deduced SRTT (is used as current RTO) are used to calculate the timeout period (RTO) of the next segment.

```
/*
 * The smoothed round-trip time and estimated variance
 * are stored as fixed point numbers scaled by the values below.
 * For convenience, these scales are also used in smoothing the average
 * (smoothed = (1/scale)sample + ((scale-1)/scale)smoothed).
 * With these scales, srtt has 3 bits to the right of the binary point,
 * and thus an "ALPHA" of 0.875.  rttvar has 2 bits to the right of the
 * binary point, and is smoothed with an ALPHA of 0.75.
 */
#define TCP_RTT_SCALE          8      /* multiplier for srtt; 3 bits frac. */
#define TCP_RTT_SHIFT          3      /* shift for srtt; 3 bits frac. */
#define TCP_RTTVAR_SCALE       4      /* multiplier for rttvar; 2 bits */
#define TCP_RTTVAR_SHIFT       2      /* multiplier for rttvar; 2 bits */

/*
 * The initial retransmission should happen at rtt + 4 * rttvar.
 * Because of the way we do the smoothing, srtt and rttvar
 * will each average +1/2 tick of bias.  When we compute
 * the retransmit timer, we want 1/2 tick of rounding and
 * 1 extra tick because of +-1/2 tick uncertainty in the
 * firing of the timer.  The bias will give us exactly the
 * 1.5 tick we need.  But, because the bias is
 * statistical, we have to test that we don't drop below
 * the minimum feasible timer (which is 2 ticks).
 * This macro assumes that the value of TCP_RTTVAR_SCALE
 * is the same as the multiplier for rttvar.
 */
#define TCP_REXMTVAL(tp) \
        (((tp)->t_srtt >> TCP_RTT_SHIFT) + (tp)->t_rttvar)
```

Some constants are also defined in the file, /usr/include/netinet/tcp_timer.h, as follows:

```
...
/*
 * Time constants.
 */
#define TCPTV_MSL       ( 30*PR_SLOWHZ)         /* max seg lifetime (hah!) */
#define TCPTV_SRTTBASE  0                       /* base roundtrip time;
                                                   if 0, no idea yet */
#define TCPTV_SRTTDFLT  (  3*PR_SLOWHZ)         /* assumed RTT if no info */
...
```

```
#define TCPTV_MIN       (  1*PR_SLOWHZ)         /* minimum allowable value */
#define TCPTV_REXMTMAX  ( 64*PR_SLOWHZ)         /* max allowable REXMT value */
...
#define TCP_MAXRXTSHIFT 12                      /* maximum retransmits */
...
```

This mechanism is also involved in the initial connection establishment (SYN segment). For that moment there is no measured RTT, the minimum value TCPTV_MIN of 2 (1 second) and the default TCPTV_SRTTDFLT of 6 (3 seconds) are defined here. These defaults are due to the RFC 1122.

RFC 1122 Requirements for Internet Hosts, Page 96

The following values SHOULD be used to initialize the estimation parameters for a new connection.

(a) RTT = 0 seconds.

(b) RTO = 3 seconds. *(The smoothed variance is to be initialized to the value that will result in this RTO).*

The important constant, PR_SLOWHZ, is defined in the header file, /usr/include/sys/protosw.h, as follows:

```
...
 * A protocol is called through the pr_init entry before any other.
 * Thereafter it is called every 200ms through the pr_fasttimo entry and
 * every 500ms through the pr_slowtimo for timer based actions.
 * The system will call the pr_drain entry if it is low on space and
 * this should throw away any non-critical data.
...
#define PR_SLOWHZ       2               /* 2 slow timeouts per second */
#define PR_FASTHZ       5               /* 5 fast timeouts per second */
...
```

The slow timeout happens every 500ms, then the PR_SLOWHZ is defined as 2. Notice that all the time constants defined in the above header file are represented by the unit of pr_slowtimo or 0.5 seconds. Therefore, the retransmission timeout period must be between 1 and 64 seconds.

As we have seen, there are many parameters and the TCP module does a bit of complicated work. Unfortunately, almost all parameters are hard coded and you can not make any adjustments.

The following table is actually a measured retransmission timeout. We established a TELNET session between two RS/6000′s and then let the interface of one RS/6000 down. The result shows that the TCP module got timeouts 12 times and retransmits the same segment 12 times. Notice that the final segment (No.27) is the reset (RST flag on) segment to abort the connection, and this segment is not the subject of the retransmission timer. You could then know that the timeout period had been expanded using the backoff algorithm and finally reached 64 seconds, which was the upper limit of the retransmission timer. Remember, all that was defined in the header files. The retransmission timer value was saturated at 128 and this is what was defined by the TCPTV_REXMTMAX. The retransmission was made 12 times and this is what is defined by the TCP_MAXRXTSHIFT.

No	Time Stamp	shift	REXMIT	REXMIT(real)	KEEP
62	16:13:17.086509952	0	3	1.255081472	14165
63	16:13:18.341591424	1	6	3.000216832	14163
64	16:13:21.341808256	2	12	6.000395008	14157
65	16:13:27.342203264	3	24	12.004708096	14145
66	16:13:39.346911360	4	48	24.001780600	14121
67	16:14:03.348691968	5	96	48.002619648	14073
68	16:14:51.351311616	6	128	64.003326336	13977
69	16:15:55.354637952	7	128	64.003434112	13849
70	16:16:59.358072064	8	128	64.003592192	13721
71	16:18:03.361664256	9	128	64.003363968	13593
72	16:19:07.365028224	10	128	64.003641344	13465
73	16:20:11.368669568	11	128	64.003480704	13337
74	16:21:15.372150272	12	128	64.003491328	13209
75	16:22:19.375641600	12	128	N/A	13081

No = Packet Number in the IP trace report.
shift = Number of exponential backoff algorithm applied.
REXMIT = Retransmission Timer Value (unit of 0.5 sec)
REXMIT(real) = Measured Retransmission Time (unit of sec)
KEEP = Keep Alive Timer Value (unit of 0.5 sec)

5.8.3.1 Socket-Level/IP Trace Example

Also, we show you how a socket-level trace looks during the retransmission. The mat is the TELNET client and the zero is the TELNET server. We made the zero′s interface down and this trace was logged at the mat.

```
# netstat -f inet -A
Active Internet connections
PCB/ADDR Proto Recv-Q Send-Q  Local Address      Foreign Address    (state)
 5a5fe00 tcp        0      0  mat.hakozaki.1035  zero.hakozak.telne ESTABLISHED
 5a4b800 tcp        0      0  mat.hakozaki.cppbr mat.hakozaki.1026  ESTABLISHED
 5a4bd00 tcp        0      0  mat.hakozaki.1026  mat.hakozaki.cppbr ESTABLISHED
 5a48900 tcp        0      0  mat.hakozaki.cppbr *.*                LISTEN

#

# trpt -p 5a5fe00 -t -a -f

...
708 ESTABLISHED:output (src=9.68.214.82,1035, dst=9.68.214.84,23)
        [2288d23f..2288d240)@228dbc8f(win=3e64)<ACK,PUSH> -> ESTABLISHED
        REXMT=3 (t_rxtshft=0), KEEP=14165
708 ESTABLISHED:user SEND -> ESTABLISHED
        REXMT=3 (t_rxtshft=0), KEEP=14165
767 ESTABLISHED:user SEND -> ESTABLISHED
        REXMT=2 (t_rxtshft=0), KEEP=14164
789 ESTABLISHED:user SEND -> ESTABLISHED
        REXMT=1 (t_rxtshft=0), KEEP=14163
834 ESTABLISHED:output (src=9.68.214.82,1035, dst=9.68.214.84,23)
        [2288d23f..2288d242)@228dbc8f(win=3e64)<ACK,PUSH> -> ESTABLISHED
        REXMT=6 (t_rxtshft=1), KEEP=14163
834 ESTABLISHED:user SLOWTIMO<REXMT> -> ESTABLISHED
```

```
       REXMT=6 (t_rxtshft=1), KEEP=14163
854 ESTABLISHED:user SEND -> ESTABLISHED
       REXMT=6 (t_rxtshft=1), KEEP=14162
134 ESTABLISHED:output (src=9.68.214.82,1035, dst=9.68.214.84,23)
       [2288d23f..2288d244)@228dbc8f(win=3e64)<ACK,PUSH> -> ESTABLISHED
       REXMT=12 (t_rxtshft=2), KEEP=14157
134 ESTABLISHED:user SLOWTIMO<REXMT> -> ESTABLISHED
       REXMT=12 (t_rxtshft=2), KEEP=14157
734 ESTABLISHED:output (src=9.68.214.82,1035, dst=9.68.214.84,23)
       [2288d23f..2288d244)@228dbc8f(win=3e64)<ACK,PUSH> -> ESTABLISHED
       REXMT=24 (t_rxtshft=3), KEEP=14145
734 ESTABLISHED:user SLOWTIMO<REXMT> -> ESTABLISHED
       REXMT=24 (t_rxtshft=3), KEEP=14145
...
136 ESTABLISHED:output (src=9.68.214.82,1035, dst=9.68.214.84,23)
       [2288d23f..2288d244)@228dbc8f(win=3e64)<ACK,PUSH> -> ESTABLISHED
       REXMT=128 (t_rxtshft=11), KEEP=13337
136 ESTABLISHED:user SLOWTIMO<REXMT> -> ESTABLISHED
       REXMT=128 (t_rxtshft=11), KEEP=13337
537 ESTABLISHED:output (src=9.68.214.82,1035, dst=9.68.214.84,23)
       [2288d23f..2288d244)@228dbc8f(win=3e64)<ACK,PUSH> -> ESTABLISHED
       REXMT=128 (t_rxtshft=12), KEEP=13209
537 ESTABLISHED:user SLOWTIMO<REXMT> -> ESTABLISHED
       REXMT=128 (t_rxtshft=12), KEEP=13209
937 CLOSED:output (src=9.68.214.82,1035, dst=9.68.214.84,23)
       2288d244@228dbc8f(win=3e64)<ACK,RST> -> CLOSED
       REXMT=128 (t_rxtshft=12), KEEP=13081
```

Here we show you the corresponding IP trace log. After disabling the interface of zero, a data segment was sent to the zero from the mat by just typing garbage characters. This is only the first three and the last three segments including the RST segment. You could know that the completely identical segments were sent repeatedly. The IP trace log is as follows:

1. We have already established a TELNET connection between mat and zero. First, we detach the interface at the TELNET server zero, and then type a pwd command at the TELNET client mat. This packet contains the first character of the pwd, p; since the TELNET server (telnetd) echoes every typed character one by one, this p should have been echoed by the zero.

   ```
   Packet Number 62
   TOK: ====( 63 bytes transmitted on interface tr0 )==== 16:13:17.086509952
   TOK: 802.5 packet
   TOK: 802.5 MAC header:
   TOK: access control field = 0, frame control field = 40
   TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
   TOK: 802.2 LLC header:
   TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
   IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
   IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
   IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=41, ip_id=579, ip_off=0
   IP:     ip_ttl=60, ip_sum=bd5d, ip_p = 6 (TCP)
   TCP:    <source port=1035, destination port=23(telnet) >
   TCP:    th_seq=2288d23f, th_ack=228dbc8f
   TCP:    th_off=5, flags<PUSH | ACK>
   TCP:    th_win=15972, th_sum=6a31, th_urp=0
   TCP: 00000000     70                                      |p                 |
   ```

2. Due to the no echo (no response) from the TELNET server zero, the mat retransmitted the character. At this moment, subsequent characters were already typed, and they were sent together as pwd.

```
Packet Number 63
TOK: ====( 65 bytes transmitted on interface tr0 )==== 16:13:18.341591424
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=43, ip_id=580, ip_off=0
IP:     ip_ttl=60, ip_sum=bd5a, ip_p = 6 (TCP)
TCP:    <source port=1035, destination port=23(telnet) >
TCP:    th_seq=2288d23f, th_ack=228dbc8f
TCP:    th_off=5, flags<PUSH | ACK>
TCP:    th_win=15972, th_sum=5b8, th_urp=0
TCP: 00000000     707764                                    |pwd             |
```

3. This is the second retransmission segment. This time, more characters (due to pressing the enter key) were added.

```
Packet Number 64
TOK: ====( 67 bytes transmitted on interface tr0 )==== 16:13:21.341808256
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=45, ip_id=581, ip_off=0
IP:     ip_ttl=60, ip_sum=bd57, ip_p = 6 (TCP)
TCP:    <source port=1035, destination port=23(telnet) >
TCP:    th_seq=2288d23f, th_ack=228dbc8f
TCP:    th_off=5, flags<PUSH | ACK>
TCP:    th_win=15972, th_sum=5a9, th_urp=0
TCP: 00000000     7077640d 00                               |pwd..           |
```

4. This is the segment of the 11th retransmission. When you compare an IP trace log with a socket level trace log, you can use the field of th_seq and th_ack as a key to match the same segment.

```
Packet Number 73
TOK: ====( 67 bytes transmitted on interface tr0 )==== 16:20:11.368669568
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=45, ip_id=595, ip_off=0
IP:     ip_ttl=60, ip_sum=bd49, ip_p = 6 (TCP)
TCP:    <source port=1035, destination port=23(telnet) >
TCP:    th_seq=2288d23f, th_ack=228dbc8f
```

```
TCP:    th_off=5, flags<PUSH | ACK>
TCP:    th_win=15972, th_sum=5a9, th_urp=0
TCP: 00000000     7077640d 00                              |pwd..            |
```

5. This is the segment of the 12th retransmission. All the information in the TCP header is completely identical for each retransmission.

```
Packet Number 74
TOK: ====( 67 bytes transmitted on interface tr0 )==== 16:21:15.372150272
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=45, ip_id=597, ip_off=0
IP:     ip_ttl=60, ip_sum=bd47, ip_p = 6 (TCP)
TCP:    <source port=1035, destination port=23(telnet) >
TCP:    th_seq=2288d23f, th_ack=228dbc8f
TCP:    th_off=5, flags<PUSH | ACK>
TCP:    th_win=15972, th_sum=5a9, th_urp=0
TCP: 00000000     7077640d 00                              |pwd..            |
```

6. This is the connection abort (reset) segment. Finally, the TCP module on the mat gave up and sent a segment with the RST flag. Since this is abortion, no TIME_WAIT period is set.

```
Packet Number 75
TOK: ====( 62 bytes transmitted on interface tr0 )==== 16:22:19.375641600
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=598, ip_off=0
IP:     ip_ttl=60, ip_sum=bd4b, ip_p = 6 (TCP)
TCP:    <source port=1035, destination port=23(telnet) >
TCP:    th_seq=2288d244, th_ack=228dbc8f
TCP:    th_off=5, flags<RST | ACK>
TCP:    th_win=15972, th_sum=da31, th_urp=0
```

5.8.3.2 Statistics Example by netstat Command

We can review the previous activity with the netstat -p tcp command. After the previous retransmission timeout sequence was completed, the following counters were incremented. The retransmit timeouts were increased by 12. Other counters were increased by one.

```
# netstat -p tcp
tcp:
...
        5 connection requests
        1 connection accept
        6 connections established (including accepts)
        5 connection closed (including 1 drops)
        0 embryonic connections dropped
        46 segments updated rtt (of 49 attempts)
        12 retransmit timeouts
                1 connection dropped by rexmit timeout
        0 persist timeouts
        0 keepalive timeouts
                0 keepalive probes sent
                0 connections dropped by keepalive
#
```

5.8.3.3 Parameter Configuration

The important point you should not forget is, it takes almost 10 minutes to drop a TCP connection after a problem occurs. In this example, it took 10 minutes and 3 seconds. At the very least, the application programs will know about it for about 10 minutes if a cable was disconnected or the destination system suddenly turned off. The AIX V4.1 provides the capability to adjust the retransmission timeout parameters. The following new options are introduced to the no command:

```
# no -a | grep rto
                  rto_low = 1
                 rto_high = 64
                rto_limit = 7
               rto_length = 13
#
```

rto_low
: Defines the minimum value of the retransmission timeout value (RTO) in seconds.

rto_high
: Defines the maximum value of the retransmission timeout value (RTO) in seconds.

rto_limit
: Defines the limit of numbers that the back-off algorithm is applied in the timeout value calculation. In our example, the RTO value was increased to 64 seconds and fixed there because the back-off algorithm was applied 7 times.

rto_length
: Defines the limit of the timeout number. The default value 13 means that after 13 RTO timeouts (and 12 retransmission), the connection is closed.

Refer to the header file, /usr/include/netinet/tcp_timer.h, and see how the back-off calculation is defined for RTO.

```
#ifdef _KERNEL
extern int tcp_keepidle;            /* time before keepalive probes begin */
extern int tcp_keepintvl;           /* time between keepalive probes */
extern int tcp_maxidle;             /* time to drop after starting probes */
extern int tcp_ttl;                 /* time to live for TCP segs */
extern int tcp_rtolow;              /* TCP RTO backoff low watermark */
extern int tcp_rtohigh;             /* TCP RTO backoff high watermark */
extern int tcp_rtolimit;            /* TCP RTO backoff exponential mark */
extern int tcp_rtolength;           /* TCP RTO backoff total# elements */
extern int tcp_rtoshift;            /* TCP RTO backoff delta shift; the */
                                    /*  number of bit shitfs from */
                                    /*  tcp_rtolow to tcp_rtohigh inclusive
*/
#define TCP_BACKOFF(x) \
                ( (((x)+1) >= (tcp_rtolimit)) ? (tcp_rtohigh) : \
                        (tcp_rtolow) << (((tcp_rtoshift)*(x))/(tcp_rtolimit)) )
#endif
```

Difference between V4.1 and V3.2

With V3.2, all the parameters which are involved in this mechanism are hard coded in the kernel. You can not tune the retransmission timeout mechanism with AIX V3.2.

If you need to close or drop the connection more rapidly, use the keep-alive timer. Look at the socket level trace log again and you will notice that the keep-alive timer is also working. Set the option tcp_keepidle and tcp_keepintvl of the no command small enough. You can compensate for the long retransmission timeout sequence.

5.8.4 TCP Keep-Alive Timeout

In this section, we describe a timeout mechanism called keep-alive. The keep-alive is a mechanism which drops hang-up TCP connections. As you know, the TCP module sends a data segment and waits for an ACK segment. But, when you don′t send any data, then the TCP module has nothing to do. A TCP connection can stay idle forever if there is no traffic. TCP itself doesn′t drop a connection just because it is idle (not used). But why is this a problem? Usually a server program accepts a connection from a client program. In the server program, a certain amount of resource is assigned to the client. Also, the OS (AIX) or TCP module consumes some system resource (for example, a socket port is a limited system resource). When the client crashes or is just turned off during the idle term, it is not aware of it and just keeps the connection in the established state and stays indefinitely. So far, the resources allocated at the server side, are no longer necessary, but they are never released by the native TCP mechanism. The keep-alive is to compensate this nature.

With the keep-alive procedure, the TCP module can decide if the current no traffic is due to just no data to be sent or due to some problems. If a problem is detected, the TCP module aborts the connection. The procedure is as follows:

1. Whenever a data segment is received, the TCP module starts the keep-alive timer. Every time a new data or ACK segment is received, the keep-alive timer is reset.

2. When the keep-alive timer reaches 120 minutes (this is the default configuration), the TCP module starts to send a probe segment. This means for the last 120 minutes, no data or ACK segment has not been received.
3. The probe segment is repeatedly retransmitted if there are no responses. The interval of each probe segment is 75 seconds (this is the default configuration). If a response is received for any of the probes, the keep-alive timer is reset.
4. If the probe segment is retransmitted eight times and finally there are no responses, the TCP module aborts the connection. Be aware that the retransmissions are sent eight times and probes are sent nine times (totally) before the connection is aborted.

Although the keep-alive looks like a nice feature, it has certain drawbacks. Due to these drawbacks, the keep-alive is purely an optional capability and not a part of the formal TCP specification. They are describe in the RFC 1122.

RFC 1122 Requirements for Internet Hosts, Page 102

DISCUSSION:

A ″Keep-alive″ mechanism periodically probes the other end of a connection when the connection is otherwise idle, even when there is no data to be sent. The TCP specification does not include a keep-alive mechanism because it could: (1) cause perfectly good connections to break during transient Internet failures; (2) consume unnecessary bandwidth (″if no one is using the connection, who cares if it is still good?″); and (3) cost money for an Internet path that charges for packets.

With all the default parameters, if there is a problem, the TCP connection is dropped 130 minutes after the last succeeded communication. These are defined in the header file, /usr/include/netinet/tcp_timer.h. You can find an excellent explanation in it as follows:

```
...
 *
 * The TCPT_KEEP timer is used to keep connections alive.  If an
 * connection is idle (no segments received) for TCPTV_KEEP_INIT amount of time,
 * but not yet established, then we drop the connection.  Once the connection
 * is established, if the connection is idle for TCPTV_KEEP_IDLE time
 * (and keepalives have been enabled on the socket), we begin to probe
 * the connection.  We force the peer to send us a segment by sending:
 *      <SEQ=SND.UNA-1><ACK=RCV.NXT><CTL=ACK>
 * This segment is (deliberately) outside the window, and should elicit
 * an ack segment in response from the peer.  If, despite the TCPT_KEEP
 * initiated segments we cannot elicit a response from a peer in TCPT_MAXIDLE
 * amount of time probing, then we drop the connection.
 */
...
/*
 * Time constants.
 */
...
#define TCPTV_KEEP_INIT ( 75*PR_SLOWHZ)        /* initial connect keep alive */
#define TCPTV_KEEP_IDLE (120*60*PR_SLOWHZ)     /* dflt time before probing */
#define TCPTV_KEEPINTVL ( 75*PR_SLOWHZ)        /* default probe interval */
#define TCPTV_KEEPCNT   8                      /* max probes before drop */
...
```

Refer to C.2, "/usr/include/netinet/tcp_timer.h" on page 367 for the complete list of this header file.

5.8.4.1 Parameter Configuration

Both TCPTV_KEEP_IDLE and TCPTV_KEEPINTVL are configurable by the no command. Also, since AIX V3.2.5, TCPTV_KEEP_INIT is now configurable. They correspond to each of the following options. The unit is 0.5 seconds.

```
# no -a | grep keep
            tcp_keepintvl = 150
             tcp_keepidle = 14400
             tcp_keepinit = 150
#
```

tcp_keepintvl
This option is to configure TCPTV_KEEPINTVL, which defines the interval between each probe.

tcp_keepidle
This option is to configure TCPTV_KEEP_IDLE, which defines the timeout period for the first probe.

If you need to change any of them, issue the no command as follows:

1. To change this option, for example, to 60 (30 seconds), issue the following command:

   ```
   # no -o tcp_keepidle=60
   #
   ```

2. Confirm if it was successfully made:

   ```
   # no -o tcp_keepidle
   tcp_keepidle = 60
   #
   ```

If you need to restore a default value, you can use the no -d command. As usual with no, this update is not saved for reboot. If you make this permanent, write this command somewhere in a start up script. The best place is at the end of the /etc/rc.net, as follows:

```
###################################################
# The socket default buffer size (initial advertised TCP window) is being
# set to a default value of 16k (16384). This improves the performance
# for Ethernet and token-ring networks.  Networks with lower bandwidth
# such as SLIP (Serial Line Internet Protocol) and X.25 or higher bandwidth
# such as Serial Optical Link and FDDI would have a different optimum
# buffer size.
# ( OPTIMUM WINDOW = Bandwidth * Round Trip Time )
###################################################
if [ -f /usr/sbin/no ] ; then
        /usr/sbin/no -o tcp_sendspace=16384
        /usr/sbin/no -o tcp_recvspace=16384
        /usr/sbin/no -o tcp_keepidle=60
        /usr/sbin/no -o tcp_keepintvl=10
fi
```

5.8.4.2 TCP Application Behavior

It is completely optional if you use keep-alive. Therefore, socket API has the option to allow you this selection. Use the SO_KEEPALIVE option of setsockopt() to enable this feature for your writing applications. If you use the system provided application (such as TELNET or FTP) , read manual or InfoExplorer. Some applications do have an option flag to enable the keep-alive. For example, with TELNET, telnetd has the -n flag to disable the keep-alive (the default is ON). Although the telnet client or telnet command doesn′t have a similar option, this is reasonable because the main purpose of the keep-alive is to release server resource in the case of client crash. For FTP, the server ftpd has the -k flag to enable the keep-alive (the default is OFF). The client or ftp command doesn′t have this option.

As quoted below, our telnetd implementation is violating the RFC.

RFC Requirements for Internet Hosts, Page 101

4.2.3.6 TCP Keep-Alives

Implementers MAY include ″Keep-Alives″ in their TCP implementations, although this practice is not universally accepted. If keep-alives are included, the application MUST be able to turn them on or off for each TCP connection, and they MUST default to off.

5.8.4.3 Socket-Level/IP Trace Example

This time, we review an IP trace example. This is also the TELNET session between the mat and zero. The mat is the TELNET server running telnetd, and the zero is the TELNET client. For this experiment, we set the keep-alive related parameters, as follows:

```
# no -a
...
            tcp_keepintvl = 10
             tcp_keepidle = 60
...
#
```

The first probe would be sent 30 seconds after the last data segment. The probes would be retransmitted every 5 seconds.

1. This is a usual TCP data segment from the mat to the zero. We just logged in to mat from zero, and this is the message and prompt displayed at the zero′s screen. In this segment, the PUSH flag is set. This means that the data carried in this segment should be passed to the application (telnet client) immediately without buffering. Since TELNET needs an interactive nature, this flag is used. Also, the ACK flag is set and this implies that there was data segment(s) from the zero to the mat (from TCP module any TELNET activity is data segment). This segment is also acknowledging for this data.

   ```
   Packet Number 46
   TOK: ====( 1128 bytes transmitted on interface tr0 )==== 18:23:10.714830080
   TOK: 802.5 packet
   TOK: 802.5 MAC header:
   TOK: access control field = 0, frame control field = 40
   TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
   TOK: 802.2 LLC header:
   TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
   ```

```
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=1106, ip_id=2181, ip_off=0
IP:     ip_ttl=60, ip_sum=b2f2, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1030 >
TCP:    th_seq=5eedc2d0, th_ack=5eedc23a
TCP:    th_off=5, flags<PUSH | ACK>
TCP:    th_win=15972, th_sum=a9af, th_urp=0
TCP: 00000000     2a2a2a2a 2a2a2a2a 2a2a2a2a 2a2a2a2a     |****************|
TCP: ********
TCP: 00000040     2a2a2a2a 2a2a2a2a 2a2a2a2a 2a2a2a0d     |***************.|
...
TCP: 000003f0     3a31343a 35382031 39393520 6f6e202f     |:14:58 1995 on /|
TCP: 00000400     6465762f 7074732f 36206672 6f6d207a     |dev/pts/6 from z|
TCP: 00000410     65726f2e 68616b6f 7a616b69 2e69626d     |ero.hakozaki.ibm|
TCP: 00000420     2e636f6d 0d0a0d0a 2320                  |.com....#       |
```

2. This is the ACK segment from the zero to the mat. The th_seq=5eedc23a is the same with the th_ack=5eedc23a of the prior segment. This is correct. The th_ack=5eedc6fa means that this is the assumed sequence number of the next segment. The prior segment has the th_seq=5eedc2d0 and ip_len=1106. Since the IP header is 20 bytes (ip_hl=20) and the TCP header is 20 bytes (th_off=5), the TCP data length is 1066 bytes. Adding 1066 to 5eedc2d0 equals to 5eedc6fa. Therefore, the next sequence number would be 5eedc6fa and this coincide with the header (th_ack=5eedc6fa). This means there is no pending data on the way. All data was safely received and acknowledged by the zero.

   Now all the sends and receives were completed and the connection was in a quiet state.

```
Packet Number 47
TOK: ====( 62 bytes received on interface tr0 )==== 18:23:10.914401408
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 18, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=3326, ip_off=0
IP:     ip_ttl=60, ip_sum=b2a3, ip_p = 6 (TCP)
TCP:    <source port=1030, destination port=23(telnet) >
TCP:    th_seq=5eedc23a, th_ack=5eedc6fa
TCP:    th_off=5, flags<ACK>
TCP:    th_win=15972, th_sum=6714, th_urp=0
```

3. Refer to the time stamp of the first line and this segment was sent 30 seconds after the prior segment. This is the keep-alive probe sent from the mat. An important feature of the probe segment is that it intentionally has the wrong sequence number. Notice it is th_seq=5eedc6f9 and this is one smaller than the th_ack=5eedc6fa of the prior segment. This is the most elaborated part of the keep-alive. Notice that this probe carries one garbage data byte “T.”

```
Packet Number 48
TOK: ====( 63 bytes transmitted on interface tr0 )==== 18:23:40.734537088
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=41, ip_id=2182, ip_off=0
IP:     ip_ttl=60, ip_sum=b71a, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1030 >
TCP:    th_seq=5eedc6f9, th_ack=5eedc239
TCP:    th_off=5, flags<ACK>
TCP:    th_win=15972, th_sum=1315, th_urp=0
TCP: 00000000     54                                      |T                |
```

4. The zero received the probe segment and acknowledged it with the sequence number that the zero or the TELNET client was expecting. The zero discarded the garbage byte (included in the probe) because the probe′s data is corresponding to a part of the received data due to the sequence number assigned to the probe.

```
Packet Number 49
TOK: ====( 62 bytes received on interface tr0 )==== 18:23:40.736268544
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 18, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=3327, ip_off=0
IP:     ip_ttl=60, ip_sum=b2a2, ip_p = 6 (TCP)
TCP:    <source port=1030, destination port=23(telnet) >
TCP:    th_seq=5eedc23a, th_ack=5eedc6fa
TCP:    th_off=5, flags<ACK>
TCP:    th_win=15972, th_sum=6714, th_urp=0
```

From a probe receiver′s point of view, it already receives the received sequence number again. In such case, the TCP module just discard the duplicated segment and responds with an acknowledgement segment with the ACK number which it wants to receive next. This is a regular TCP behavior. In other words, the TCP module of the proved side never needed the special feature to support the keep-alive. Any correct TCP implementation should respond to the keep-alive probe segment.

Difference between V4.1 and V3.2

As you see, our V4.1 keep-alive probe segment carries one byte of meaningless data. But, with V3.2, no data was carried. Some incorrect TCP implementations don′t respond to a keep-alive probe if the probe doesn′t carry data. 4.2 BSD may be the most famous example. Thus, keep-alive doesn′t work if we use V3.2 with 4.2 BSD. Our V4.1 is enhanced to support this situation. This problem is describe in the RFC 1122.

RFC 1122 Requirements for Internet Hosts, Page 102

Unfortunately, some misbehaved TCP implementations fail to respond to a segment with SEG.SEQ = SND.NXT-1 unless the segment contains data. Alternatively, an implementation could determine whether a peer responded correctly to keep-alive packets with no garbage data octet.

The following is the corresponding socket-level trace log. This trace was obtained at the mat because the telnetd was running there. Remember that keep-alive is enabled by an application (mostly by a server side).

Note: The regular ACK segment was not logged with socket-level trace. Therefore, Packet Number 47 of above IP trace log was not shown.

```
071 ESTABLISHED:output (src=9.68.214.82,23, dst=9.68.214.84,1030)
        [5eedc2d0..5eedc6fa)@5eedc23a(win=3e64)<ACK,PUSH> -> ESTABLISHED
        REXMT=4 (t_rxtshft=0), KEEP=60
073 ESTABLISHED:user SLOWTIMO<KEEP> -> ESTABLISHED
        KEEP=10
073 ESTABLISHED:input (src=9.68.214.84,1030, dst=9.68.214.82,23)
        5eedc23a@5eedc6fa(win=3e64)<ACK> -> ESTABLISHED
        KEEP=60
```

When the data segment was sent, the keep-alive timer was KEEP=60. (This is also defined by the unit of 0.5 seconds.) This timer value is what we set by tcp_keepidle=60 of the no command. When this timer was expired, the probe was sent and at this moment the keep-alive timer was set to KEEP=10. This is what we set by tcp_keepintvl=10. If this timer would have expired, the second probe would have been sent. In the previous experiment, the grover responded to the ACK and the keep-alive timer was again refreshed to KEEP=60.

The following is an IP trace example for when the keep-alive failed. In AIX V4.1, the probes are sent up to nine times. In this experiment, we made the interface of the zero detach after the successful data and acknowledgement transmission. We just show the last two probe′s segments and the RST segment. All the keep-alive parameters are the same as the previous experiment.

1. This is the 8th probe segment.

   ```
   Packet Number 76
   TOK: ====( 63 bytes transmitted on interface tr0 )==== 18:26:47.244921088
   TOK: 802.5 packet
   TOK: 802.5 MAC header:
   TOK: access control field = 0, frame control field = 40
   TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
   TOK: 802.2 LLC header:
   TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
   IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
   IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
   IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=41, ip_id=2218, ip_off=0
   IP:     ip_ttl=60, ip_sum=b6f6, ip_p = 6 (TCP)
   TCP:    <source port=23(telnet), destination port=1030 >
   TCP:    th_seq=5eedc862, th_ack=5eedc23d
   TCP:    th_off=5, flags<ACK>
   TCP:    th_win=15972, th_sum=23a8, th_urp=0
   TCP: 00000000     42                                       |B               |
   ```

2. This is the 9th probe segment. The time stamp tells that this probe was sent five seconds after the 8th probe. This is what we expected.

```
Packet Number 77
TOK: ====( 63 bytes transmitted on interface tr0 )==== 18:26:52.245235840
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=41, ip_id=2219, ip_off=0
IP:     ip_ttl=60, ip_sum=b6f5, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1030 >
TCP:    th_seq=5eedc862, th_ack=5eedc23d
TCP:    th_off=5, flags<ACK>
TCP:    th_win=15972, th_sum=21a8, th_urp=0
TCP: 00000000     44                                        |D                |
```

3. No responses were received by the mat for five seconds after the last (9th) probe, the keep-alive timer, expired and the RST segment was sent. Be aware that this is no longer a probe but actually just the RST segment. Therefore, the sequence number is set to the correct number th_seq=2bce1275 (one bigger than that of probes).

```
Packet Number 78
TOK: ====( 62 bytes transmitted on interface tr0 )==== 18:26:57.245547648
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=2220, ip_off=0
IP:     ip_ttl=60, ip_sum=b6f5, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1030 >
TCP:    th_seq=5eedc863, th_ack=5eedc23e
TCP:    th_off=5, flags<RST | ACK>
TCP:    th_win=15972, th_sum=65a3, th_urp=0
```

Again, this is the corresponding socket-level trace log. The keep-alive timer was set to KEEP=10 each time a probe was sent. Notice that the final RST segment was also kicked by the keep-alive timer.

```
724 ESTABLISHED:user SLOWTIMO<KEEP> -> ESTABLISHED
        KEEP=10
224 ESTABLISHED:user SLOWTIMO<KEEP> -> ESTABLISHED
        KEEP=10
724 CLOSED:output (src=9.68.214.82,23, dst=9.68.214.84,1030)
        5eedc863@5eedc23e(win=3e64)<ACK,RST> -> CLOSED
```

5.8.4.4 Statistics Example by netstat Command

The keep-alive activity is counted by the kernel and you can review it by the netstat -p tcp command. The following counters get incremented with the keep-alive. These counters were taken at the mat (TELNET server). If a connection is dropped by the keep-alive mechanism, the counter, connections closed, and connections dropped by keepalive are all incremented by one.

```
# netstat -p tcp
tcp:
...
        15 connection requests
        4 connection accepts
        19 connections established (including accepts)
        18 connections closed (including 3 drops)
        0 embryonic connections dropped
        1499 segments updated rtt (of 1505 attempts)
        24 retransmit timeouts
                2 connections dropped by rexmit timeout
        0 persist timeouts
        17 keepalive timeouts
                14 keepalive probes sent
                1 connection dropped by keepalive
#
```

5.8.5 TCP Persist Timeout

The persist timeout is to ensure that the TCP window flow control works safely. TCP guarantee reliable data transmission and any data must be acknowledged by the receiver. But, TCP doesn′t guarantees the reliable ACK segment transmission. In other words, TCP only watches if the data has arrived safely at the destination. This can be done by receiving the acknowledgement from the destination. But, the destination system doesn′t care if the acknowledgement arrived at the source system safely. Then, if the acknowledgement is lost during the transmission, what will happen? The source system concludes that the data was not delivered to the destination, because no acknowledgements were returned (but this is not correct). Then, the source system retransmits the data again, and the destination system receives the same data twice. The destination system just discards the duplicated data and sends the acknowledgement again. As you already know, the sequence number and acknowledgement number in each segment have a crucial role in this mechanism.

Note: Since the TCP connection is bidirectional, an acknowledgement segment can carry data (more accurately, a data segment can carry acknowledgement). In this case, due to this piggy-backed data, this segment is subject to be reliable transmission.

In the TCP flow control mechanism, if a receiver system informed a 0 window size, the sender must stop the data transmission until the receiver informs more than a 0 window size. This window update from the receiver is made by an acknowledgement segment. If an acknowledgement only segment (no data) is used for this purpose, there are no reliable mechanisms to confirm if the window update is delivered. What happens if it is lost? The sender just considers that the receiver′s window remains 0 and cannot send any data. The receiver would not receive any new data, but this is just considering that the sender no longer has any data to send. As a result, the TCP connection hangs indefinitely. There are no reasons to initiate a new transmission for both sides. The persist timeout makes up for this problem.

The overview of the persist timeout is as follows:

1. A receiver acknowledges with the window field 0, when the receiver window is full.
2. The sender stops to send the next data segment. At this moment, the sender starts the persist timer.

3. When the receive buffer gets some empty space, the receiver sends an acknowledgement segment with the updated window size (more than 0). This segment is called the window update. Otherwise, the receiver remains silent.

   Note: This window update can be lost during the transmission and there are no ways to know if it is lost.

4. When the persist timer at the sender expires, the sender sends the next data segment carrying one byte data. This is called the window probe. If the sender receives the window update segment safely, the persist timer is reset and no probes are sent.

5. When the probe arrived at the receiver, the receiver responds with the same acknowledgement segment advertising 0 window size if the receive buffer is still full.

   Note: In case the window has already been updated and the window update segment was lost, the receiver responds a regular acknowledgement telling that the one data byte was received and it informs the new updated window size.

6. If the sender receives the same 0 window acknowledgement, it resets the persist timer and repeats the previous procedure. But this time, the persist timer is set with a longer timeout value.

The persist timer value is incremented when it expires. The exponential backoff algorithm is used here. As you can see following, the necessary constants are defined in the /usr/include/netinet/tcp_timer.h header file. Refer to C.2, "/usr/include/netinet/tcp_timer.h" on page 367 for the complete list of this header file. The minimum persist timer value is 5 seconds and the maximum is 60 seconds. Between this minimum and maximum, the value is updated. Although it is not a mandatory requirement, our AIX V3.2 implementation uses the same algorithm with the retransmit timeout. With our experiment, the timer PERSIST increased from 10 to 120 with the following order. 10, 10, 12, 24, 48, 96 and 120 (all are described by a unit of 0.5 seconds). Why did 10 (5 seconds) repeat twice? This because the minimum is 10, so any retransmit timeouts lower than 10 were round up to 10. There is no final timeout to drop or abort the connection for the persist timeout. This is a big difference from other timeouts.

```
...
 *
 * The TCPT_PERSIST timer is used to keep window size information
 * flowing even if the window goes shut.  If all previous transmissions
 * have been acknowledged (so that there are no retransmissions in progress),
 * and the window is too small to bother sending anything, then we start
 * the TCPT_PERSIST timer.  When it expires, if the window is nonzero,
 * we go to transmit state.  Otherwise, at intervals send a single byte
 * into the peer′s window to force him to update our window information.
 * We do this at most as often as TCPT_PERSMIN time intervals,
 * but no more frequently than the current estimate of round-trip
 * packet time.  The TCPT_PERSIST timer is cleared whenever we receive
 * a window update from the peer.
 *
...
/*
 * Time constants.
 */
...
#define TCPTV_PERSMIN   (  5*PR_SLOWHZ)         /* retransmit persistance */
#define TCPTV_PERSMAX   ( 60*PR_SLOWHZ)         /* maximum persist interval */
...
```

5.8.5.1 Socket-Level/IP Trace Example

Here is an example of the persist timeout. Again, we used TELNET. The mat is the TELNET server and the zero is the TELNET client. During this TELNET session, the receive window of the client side was filled up.

1. We issued the following command at the zero, just to make the TELNET server mat to send data continuously:

   ```
   # cat smit.log
   ```

   This segment is carrying data of a smit.log to be displayed at the TELNET client window at the zero.

   ```
   Packet Number 178
   TOK: ====( 1002 bytes transmitted on interface tr0 )==== 20:02:10.533591552
   TOK: 802.5 packet
   TOK: 802.5 MAC header:
   TOK: access control field = 0, frame control field = 40
   TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
   TOK: 802.2 LLC header:
   TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
   IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
   IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
   IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=980, ip_id=3762, ip_off=0
   IP:     ip_ttl=60, ip_sum=ad43, ip_p = 6 (TCP)
   TCP:    <source port=23(telnet), destination port=1034 >
   TCP:    th_seq=8b2b7646, th_ack=8b29fa62
   TCP:    th_off=5, flags<ACK>
   TCP:    th_win=15972, th_sum=cebd, th_urp=0
   TCP: 00000000     41342082 c582b781 428fda8d d782c982     |A4 .....B.......|
   TCP: 00000010     c282a282 c482cd81 4182b182 cc0d0a20     |........A...... |
   TCP: 00000020     20834383 93835883 67815b83 8b82cc8f     | .C...X.g.·.....|
   ...
   ```

2. In the middle of continuous data transmission, we just enter the TELNET escape sequence, Ctrl-T, at the client zero. With this operation, the TELNET client displayed the TELNET command prompt (tn>) and stalled the command input and stopped the scrolling of the file smit.log file. Inside the TELNET client, the read operation from the socket receive buffer was suspended and the receive buffer was easily filled up. As a result, the TCP module on the zero was forced to advertise the 0 receive window. Therefore th_win=0 is in this segment.

   Now the mat knows the receive window of the zero got to 0 and starts the persist timer.

```
Packet Number 179
TOK: ====( 62 bytes received on interface tr0 )==== 20:02:10.538320640
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 18, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=5263, ip_off=0
IP:     ip_ttl=60, ip_sum=ab12, ip_p = 6 (TCP)
TCP:    <source port=1034, destination port=23(telnet) >
TCP:    th_seq=8b29fa62, th_ack=8b2b79f2
TCP:    th_off=5, flags<ACK>
TCP:    th_win=0, th_sum=61da, th_urp=0
```

3. The persist timer (5 seconds) at the mat expired, but still no window update segments came in. Then the mat sends a window probe. This segment has the th_seq=8b2b79f2 with only one byte of data. If this one byte is received by the zero, the ACK number in the ACK segment from the zero should be 8b2b79f3.

```
Packet Number 180
TOK: ====( 63 bytes transmitted on interface tr0 )==== 20:02:15.533966208
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=41, ip_id=3763, ip_off=0
IP:     ip_ttl=60, ip_sum=b0ed, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1034 >
TCP:    th_seq=8b2b79f2, th_ack=8b29fa62
TCP:    th_off=5, flags<ACK>
TCP:    th_win=15972, th_sum=9874, th_urp=0
TCP: 00000000     8b                                      |.                |
```

4. At the zero, the receive window was still filled completely. So the zero cannot receive even one byte on the data. It again advertised specifying th_win=0 and doesn't increase the ACK number. Notice it says th_ack=8b2b79f2 and lets the mat know that the data was not received.

   The mat restarts the persist timer again. Until the zero advertises the receive window greater than 0, this procedure is repeated.

```
Packet Number 181
TOK: ====( 62 bytes received on interface tr0 )==== 20:02:15.535704704
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 18, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=5264, ip_off=0
IP:     ip_ttl=60, ip_sum=ab11, ip_p = 6 (TCP)
```

```
    TCP:    <source port=1034, destination port=23(telnet) >
    TCP:    th_seq=8b29fa62, th_ack=8b2b79f2
    TCP:    th_off=5, flags<ACK>
    TCP:    th_win=0, th_sum=61da, th_urp=0
```

This is the corresponding socket-level trace log at the mat. Before Packet Number 178 (time stamp 053) was sent, several SEND requests were buffered. In this example, only two are shown (time stamp 662). An interesting thing is that the buffer flush was also invoked by the persist timer. The ACK Packet Number 179 was logged at the time stamp 053. The window probe Packet Number 180 was logged at the time stamp 553.

```
...
662 ESTABLISHED:user SEND -> ESTABLISHED
        PERSIST=8, KEEP=14398
662 ESTABLISHED:user SEND -> ESTABLISHED
        PERSIST=8, KEEP=14398
053 ESTABLISHED:output (src=9.68.214.82,23, dst=9.68.214.84,1034)
        [8b2b7646..8b2b79f2)@8b29fa62(win=3e64)<ACK> -> ESTABLISHED
        REXMT=4 (t_rxtshft=0), KEEP=14391
053 ESTABLISHED:user SLOWTIMO<PERSIST> -> ESTABLISHED
        REXMT=4 (t_rxtshft=0), KEEP=14391
053 ESTABLISHED:input (src=9.68.214.84,1034, dst=9.68.214.82,23)
        8b29fa62@8b2b79f2<ACK> -> ESTABLISHED
        KEEP=14400
553 ESTABLISHED:output (src=9.68.214.82,23, dst=9.68.214.84,1034)
        [8b2b79f2..8b2b79f3)@8b29fa62(win=3e64)<ACK> -> ESTABLISHED
        PERSIST=10, KEEP=14391
553 ESTABLISHED:user SLOWTIMO<PERSIST> -> ESTABLISHED
        PERSIST=10, KEEP=14391
553 ESTABLISHED:input (src=9.68.214.84,1034, dst=9.68.214.82,23)
        8b29fa62@8b2b79f2<ACK> -> ESTABLISHED
        PERSIST=10, KEEP=14400
```

5.8.5.2 Statistics Example by the netstat Command

We can review the above activity with the netstat -p tcp command. The activity is counted, as described. For the receiving side, whose window gets filled, it would get several counters incremented. On the sending side, which sends window probes, it would get one counter incremented. It would not be a good sign if you see these counters getting incremented because it implies that someone (your system or your destination) got its receive window run out. You may need to expand the receive buffer(s).

- Here are the counters in the window probe sender mat before the experiment:

  ```
  mat # LANG=C netstat -p tcp
  tcp:
          3048 packets sent
                  2663 data packets (561095 bytes)
                  24 data packets (116 bytes) retransmitted
                  327 ack-only packets (290 delayed)
                  0 URG only packets
                  0 window probe packets
                  0 window update packets
                  34 control packets
          4069 packets received
  ...
  ```

- Here are the counters in the window probe sender mat after the experiment:

```
mat # LANG=C netstat -p tcp
tcp:
        3233 packets sent
                2800 data packets (633204 bytes)
                24 data packets (116 bytes) retransmitted
                359 ack-only packets (314 delayed)
                0 URG only packets
                12 window probe packets
                0 window update packets
                38 control packets
        4266 packets received
...
```

- Here are the counters in the window probe receiver zero before the experiment:

```
zero # netstat -p tcp
tcp:
        3970 packets sent
                2201 data packets (17034 bytes)
                12 data packets (72 bytes) retransmitted
                1740 URG only packets
                0 URG only packets
                0 window probe packets
                1 window update packet
                23 control packets
        3162 packets received
                2217 acks (for 17050 bytes)
                11 duplicate acks
                0 acks for unsent data
                2634 packets (560523 bytes) received in-sequence
                252 completely duplicate packets (250 bytes)
                0 packets with some dup. data (0 bytes duped)
                7 out-of-order packets (0 bytes)
                0 packets (0 bytes) of data after window
                0 window probes
                0 window update packets
                0 packets received after close
...
```

- Here are the counters in the window probe receiver zero after the experiment:

```
zero # netstat -p tcp
tcp:
        4135 packets sent
                2279 data packets (17200 bytes)
                12 data packets (72 bytes) retransmitted
                1818 URG only packets
                0 URG only packets
                0 window probe packets
                6 window update packets
                27 control packets
        3314 packets received
                2297 acks (for 17218 bytes)
                21 duplicate acks
                0 acks for unsent data
                2759 packets (632104 bytes) received in-sequence
                252 completely duplicate packets (250 bytes)
                0 packets with some dup. data (0 bytes duped)
                7 out-of-order packets (0 bytes)
```

```
                 10 packets (10 bytes) of data after window
                 10 window probes
                 0 window update packets
                 1 packet received after close
...
```

For the receiving side, whose window gets filled, several counters get incremented. On the sending side, which sends window probes, one counter gets incremented. It would not be a good sign if you see these counters get incremented because it implies that someone (your system or your destination) had their receive window run out. You may need to expand the receiver buffer(s).

Since the window probe contains one data byte which is out of the receive window if the window is still full, then, the counter (packets of data after window) was incremented. Also the window probe has the ACK flag and all the probes are the same. Then they were also counted as the counter duplicate ACKs.

5.9 New Option rfc1323 to Implement RFC1323

Since TCP was designed, our network environment has been updated dramatically. Still, TCP is one of the best choices for networking. But in certain instances, enhancements are needed in order to maintain reasonable performance. One example is a network called long, fat pipe or LFN, (pronounced "elephan(t)"). The LFN has the big product of bandwidth × delay. For example, a satellite link and a fiber cable between countries is considered to be LFN. They could have a bandwidth of tens of megabits and delays of tens of msec; the products would be the order of 10•. The RFC 1323 introduced new options to enhance TCP to work well enough with the LFN environment.

5.9.1 New Options Defined by RFC1323

In order to achieve reasonable performance and enough reliability for LFN, the following new TCP options have been defined by the *RFC 1323 Extensions for High Performance.*

TCP Window Scale Option
The current TCP window has the limitation of 64 KB. This option expands the limitation to 2 GB in order to exploit the advantage of the window scheme for LFN.

TCP Timestamps Option
This option is designed for two purposes. One is Round-Trip Time Measurement (RTTM) to provide accurate time measurement. The other is Protect Against Wrapped Sequence Number (PAWS). With a large window size, the sequence number and acknowledge number can run out and can be reused within the same TCP connection. The timestamp in each segment makes this possible.

For details of options and backgrounds, refer to the RFC. After AIX V3.2.5, we have a new no command option, rfc1323. With this option you can enable the previous two new TCP options for your system. These options do not impact current connectivity. TCP negotiates options when it establishes a connection (remember MSS option). Therefore, if the destination system doesn't support RFC 1323, these options are not used even when you set the rfc1323. Only if both systems agree to use these options are they used.

One drawback is that these options are designed for LFN. Enabling these options for a usual environment, may give you performance degradation. You should not have any connectivity problems, but options may need some overhead.

rfc1323
: Setting this option to 1 enables both the TCP window scale option and the TCP timestamp option to be defined by the RFC 1323. The default is 0.

If you need to change any of them, issue the no command, as follows:

1. Check the current configuration:

   ```
   # no -o rfc1323
   rfc1323 = 0
   #
   ```

2. To change this option to effective, set it to 1, as follows:

   ```
   # no -o rfc1323=1
   #
   ```

3. Confirm if the change was successfully made:

   ```
   # no -o rfc1323
   rfc1323 = 1
   #
   ```

If you need to restore the default value, you can use no -d. As is usual of the no command, this update is not saved for reboot. If you make this permanent, write this command some where in a start up script. The best place is the end of /etc/rc.net, as follows:

```
###################################################
# The socket default buffer size (initial advertised TCP window) is being
# set to a default value of 16k (16384). This improves the performance
# for Ethernet and token-ring networks.  Networks with lower bandwidth
# such as SLIP (Serial Line Internet Protocol) and X.25 or higher bandwidth
# such as Serial Optical Link and FDDI would have a different optimum
# buffer size.
# ( OPTIMUM WINDOW = Bandwidth * Round Trip Time )
###################################################
if [ -f /usr/sbin/no ] ; then
        /usr/sbin/no -o tcp_sendspace=16384
        /usr/sbin/no -o tcp_recvspace=16384
        /usr/sbin/no -o rfc1323=1
fi
```

5.9.2 Option Negotiation Examples

TCP has the option field formatted as follows. All the TCP options comply with this format, and all the option fields consist of the following three parts:

Kind (1 byte)
: The first byte represents what option this is.

Length (1 byte)
: The second byte represents the length of this option field. The length includes both the kind field and length field.

Option data (variable)
: The third byte and following bytes represent data used by this option.

Table 3. Current Available TCP Options

Option Name	Kind	Length (byte)	Option Field
N/A	0	N/A	End of option list
N/A	1	N/A	No-operation
MSS	2	4	Max segment size
Window scale	3	3	Shift count
Timestamp	8	10	Timestamp value (4 bytes) and timestamp echo reply (4 bytes)

Some options use the option field at connection establishment only, in order to negotiate the option parameters. Other options need the option field in every segment.

5.9.2.1 New Options Ignored

In this section, we show you the failed option negotiation procedure of the TCP window scale option and the TCP timestamp option. We used TELNET. The mat is the TELNET client and the zero is the TELNET server (telnetd). At the mat we configured rfc1323=1 and at the zero we configured rfc1323=0. Therefore these options should not be effective.

1. The active open side, mat, sent the SYN segment with the options. Notice that the th_off=10, and this means that this segment has 20 bytes of total option fields.

0204079C	MSS
01	No-Operation
030300	Window Scale
01	No-Operation
01	No-Operation
080A 303775EA 00000000	Timestamp

```
Packet Number 1
TOK: ====( 82 bytes transmitted on interface tr0 )==== 20:53:13.871623936
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=60, ip_id=4557, ip_off=0
IP:     ip_ttl=60, ip_sum=adc0, ip_p = 6 (TCP)
TCP:    <source port=1043, destination port=23(telnet) >
TCP:    th_seq=a3b14601, th_ack=0
TCP:    th_off=10, flags<SYN>
TCP:    th_win=16384, th_sum=b7e2, th_urp=0
TCP: 00000000     020405ac 01030300 0101080a 303775ea     |............07u.|
TCP: 00000010     00000000                                |....            |
```

2. The passive open side, zero, sent the SYN/ACK segment with the only option (MSS).

```
Packet Number 2
TOK: ====( 66 bytes received on interface tr0 )==== 20:53:13.874240640
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 18, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=44, ip_id=6242, ip_off=0
IP:     ip_ttl=60, ip_sum=a73b, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1043 >
TCP:    th_seq=a3ae5801, th_ack=a3b14602
TCP:    th_off=6, flags<SYN | ACK>
TCP:    th_win=15972, th_sum=b0fd, th_urp=0
TCP: 00000000     020405ac                                |....            |
```

3. This is the ACK from the mat and the completed TCP three way handshake. As a result, only the MSS option was effectively used. All the other options were just ignored. This segment and all the following segments don′t have an option field.

```
Packet Number 3
TOK: ====( 62 bytes transmitted on interface tr0 )==== 20:53:13.874291840
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=4558, ip_off=0
IP:     ip_ttl=60, ip_sum=add3, ip_p = 6 (TCP)
TCP:    <source port=1043, destination port=23(telnet) >
TCP:    th_seq=a3b14602, th_ack=a3ae5802
TCP:    th_off=5, flags<ACK>
TCP:    th_win=15972, th_sum=c8b2, th_urp=0
```

5.9.2.2 New Options Effective

In this section, we show you the successful option negotiation procedure of the TCP Window Scale option and the TCP Timestamp option. We used TELNET. The mat is the TELNET client and the zero is the TELNET server (telnetd). At the mat we configured rfc1323=1 and at the zero we configured rfc1323=1. Therefore these options should be effective.

1. The active open side, mat, sent SYN segment with the options. Notice that the th_off=10 and this means this segment has 20 bytes of total option fields.

0204079C	MSS
01	No-Operation
030300	Window Scale
01	No-Operation
01	No-Operation
080A 2E82BC24 00000000	Timestamp

```
Packet Number 1
TOK: ====( 82 bytes transmitted on interface tr0 )==== 20:56:28.147303936
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=60, ip_id=4852, ip_off=0
IP:     ip_ttl=60, ip_sum=ac99, ip_p = 6 (TCP)
TCP:    <source port=1044, destination port=23(telnet) >
TCP:    th_seq=a52d2801, th_ack=0
TCP:    th_off=10, flags<SYN>
TCP:    th_win=16384, th_sum=d2e1, th_urp=0
TCP: 00000000     020405ac 01030300 0101080a 3037776e     |............07wn|
TCP: 00000010     00000000                                |....            |
```

2. The passive open side, zero, sent SYN/ACK segment with the options. Notice that the th_off=10, and this means that this segment has 20 bytes of total option fields.

020405AC	MSS
01	No-Operation
030300	Window Scale
01	No-Operation
01	No-Operation
080A 30377675 3037776E	Timestamp

Now both systems can agree to use all the options. Be aware that the Timestamp option field is carrying the timestamp at this system with the original timestamp.

```
Packet Number 2
TOK: ====( 82 bytes received on interface tr0 )==== 20:56:28.149961984
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 18, frame control field = 40
TOK: [ src = 40:00:7e:08:66:70, dst = 10:00:5a:a8:b5:c1]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     < DST =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=60, ip_id=6448, ip_off=0
IP:     ip_ttl=60, ip_sum=a65d, ip_p = 6 (TCP)
TCP:    <source port=23(telnet), destination port=1044 >
TCP:    th_seq=a52b3401, th_ack=a52d2802
TCP:    th_off=10, flags<SYN | ACK>
TCP:    th_win=15840, th_sum=5517, th_urp=0
TCP: 00000000     020405ac 01030300 0101080a 30377675     |............07vu|
TCP: 00000010     3037776e                                |07wn            |
```

3. This is the ACK from newton and the complete TCP three way handshake. As a result all options are used effectively. MSS and Window Scale negotiation were already finished, this segment and all the following segments have only the timestamp option field.

```
Packet Number 3
TOK: ====( 74 bytes transmitted on interface tr0 )==== 20:56:28.150108416
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 10:00:5a:a8:b5:c1, dst = 40:00:7e:08:66:70]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =     9.68.214.82 >  (mat.hakozaki.ibm.com)
IP:     < DST =     9.68.214.84 >  (zero.hakozaki.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=52, ip_id=4853, ip_off=0
IP:     ip_ttl=60, ip_sum=aca0, ip_p = 6 (TCP)
TCP:    <source port=1044, destination port=23(telnet) >
TCP:    th_seq=a52d2802, th_ack=a52b3402
TCP:    th_off=8, flags<ACK>
TCP:    th_win=15840, th_sum=80d3, th_urp=0
TCP: 00000000     0101080a 3037776e 30377675              |....07wn07vu    |
```

5.9.2.3 Option Field Padding Consideration

As you see in the previous sections, the implementation of our AIX V4.1 uses the dummy option, No-operation, in order to align the ends of other option fields to a multiple of 4 bytes. Be aware that this is a possible implementation but not a mandatory requirement. Other implementations may use the End of Option List. Refer to the *RFC 793 TRANSMISSION CONTROL PROTOCOL SPECIFICATION*.

RFC 793 TRANSMISSION CONTROL PROTOCOL, Page 18

No-Operation... This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Chapter 6. Performance Tuning Tools

We have several useful tools for performance tuning on the RS/6000. In this chapter, three of them are explained briefly. They are ping, spray and netpmon. The ping and spray are active tools. Active means that they send packets to their destination system and monitor what happens. On the contrary, the netpmon is a passive tool because it just gathers a lot of data from various counters in the kernel and generates a summary report.

It is recommended that you practice using these tools, because the most difficult part of using the tools is understanding or interpreting their responses and data.

6.1 ping

We have already explained some functions of this command in a previous chapter. The ping has a lot of options. In this section we just mention some convenient options relevant to performance tuning.

6.1.1 Convenient Options

The following are typical options to be used in performance tuning. We already used -c and -s in another part of this book. Notice that this is not the complete list of options. Refer to the manual or InfoExplorer for details.

-c Specify the number of packets. This option is useful when you get an IP trace log. You can capture a minimum of ping packets.

-s Specify the length of packets. You can use this option to check fragmentation and reassembly.

-f Send the packets at 10 ms intervals or immediately after each response. Only the root user can use this option.

If you need to load your network or systems, the -f option is convenient. For example, if you suspect that your problem is caused by a heavy load, you want to load your environment intentionally in order to confirm your thought. Open several aixterms and run the ping -f command in each aixterm. Your Ethernet utilization quickly gets to around 100 % (you could see this if you have a protocol analyzer). See the following example:

```
# date ; ping -c 1000 -f mat ; date
Sun Aug 20 21:09:01 1995
PING mat.hakozaki.ibm.com: (9.68.214.82): 56 data bytes
.
----mat.hakozaki.ibm.com PING Statistics----
1000 packets transmitted, 1000 packets received, 0% packet loss
round-trip min/avg/max = 2/3/22 ms
Sun Aug 20 21:09:06 1995
#
```

In this example, 1000 packets were sent for 5 seconds. Be aware that this command uses IP and ICMP protocol. No transport protocol (UDP/TCP) and application activities are involved. Do not forget that the measured data such as round-trip time, doesn′t reflect the total performance characteristics.

6.1.2 Considerations

When you try to send a flood of packets to your destination, you should consider some points. Sending packets also puts a load on your system. You have to monitor the status of your network interface during the experiment. The netstat -i command is convenient for this purpose. You may find the system of dropping packets during a send, as follows:

```
# netstat -i
Name  Mtu   Network     Address         Ipkts    Ierrs Opkts    Oerrs Coll
lo0   1536  <Link>                          616     0       616     0     0
lo0   1536  127         loopback            616     0       616     0     0
tr0   1492  <Link>                      1135982     0   435773   104     0
tr0   1492  9.170.5     newton          1135982     0   435773   104     0
#
```

You should watch other resources such as mbuf and send/receive queue. If the previous happens, the destination system just receives filtered packets. It can be difficult to place a heavy load to the destination system. Your system may give up first.

Pay attention to relativity of the results. If you want to monitor or test just one destination system, do the same experiment to some other systems for comparison, because your network or router may have a problem.

6.2 spray

The spray command is a client-server style application. You need the spray command on your system and the sprayd server daemon running on the destination system. This tool is an application of Sun ONC/RPC. Your environment must support ONC/RPC or you can't use this tool.

6.2.1 Configuration

If you invoke the spray command without preparation at the destination system, you will get the following error message:

```
# spray grover
sending 1162 packets of length 86 to
        grover ...
SPRAYPROC_CLEAR RPC: 1832-019 Program not registered
#
```

This tool needs to be configured at the server side before its use. The daemon sprayd is a subserver of the inetd. The configuration procedure is below, if the destination (sprayd) system is an RS/6000. With the default configuration, sprayd should have been configured if it is RS/6000. The following procedure explains this further.

1. First, you have to make sure that the portmap daemon is running on the system where the sprayd is invoked. In the RPC scheme, the portmap daemon has a crucial role. You can check the status with the lssrc command as follows:

   ```
   # lssrc -s portmap
   Subsystem         Group             PID     Status
    portmap          portmap           5654    active
   #
   ```

With default configuration, the portmap is invoked from the script /etc/rc.tcpip at system boot and you should not worry about it. You can find the following lines in the script.

```
...
# Start up Portmapper
USR_NFS=mount | awk '{if($3=="/usr") print $4}'
if [ "$USR_NFS" != "nfs" ]
then
REMOTE_USR="N"
start /usr/sbin/portmap "$src_running"
fi
...
```

Note: The portmap daemon is necessary if you run more than one ONC/RPC application on your system. NIS and NFS are good examples

2. Edit the /etc/inetd.conf file and remove the comment mark (#) from the line, as follows:

```
...
#rexd    sunrpc_tcp    tcp    wait    root    /usr/sbin/rpc.rexd rexd 100017 1
rstatd   sunrpc_udp    udp    wait    root    /usr/sbin/rpc.rstatd rstatd 100001 1-3
rusersd  sunrpc_udp    udp    wait    root    /usr/lib/netsvc/rusers/rpc.rusersd rusersd 100002 1-2
rwalld   sunrpc_udp    udp    wait    root    /usr/lib/netsvc/rwall/rpc.rwalld rwalld 100008 1
sprayd   sunrpc_udp    udp    wait    root    /usr/lib/netsvc/spray/rpc.sprayd sprayd 100012 1
pcnfsd   sunrpc_udp    udp    wait    root    /usr/sbin/rpc.pcnfsd pcnfsd 150001 1-2
...
```

3. Import the update into the ODM InetServ object class:

```
# inetimp
#
```

4. Refresh the inetd to load the updated information from the ODM:

```
# refresh -s inetd
0513-095 The request for subsystem refresh was completed successfully.
#
```

You can confirm if the sprayd daemon has successfully configured, as follows. This command can get information from a remote system:

```
# rpcinfo -u mat spray
program 100012 version 1 ready and waiting
#
```

The spray is a simple RPC application. You can learn how ONC/RPC works with this tool.

6.2.2 Convenient Options

The spray uses ONC/RPC. An RPC is mapped to an UDP datagram or datagrams. Remember that the ping uses the ICMP echo and reply, and each ping packet (ICMP echo) is returned by the destination system. On the contrary, the destination (sprayd daemon) receives a flood of spray RPCs (UDP datagrams) and responds to only one RPC to inform it of the result.

Notice this is not the complete list of options. Refer to the manual or InfoExplorer for details.

-c Specify number of RPC. The default is the number of RPCs required to make the total 100,000 bytes.

-l Specify length of RPC in byte. You can use this option to check fragmentation and reassembly. The default is 86 bytes and this is the minimum length. The maximum length is 8842 bytes (this is the limit of current ONC/RPC).

-d Specify the time interval between each RPC in microseconds. The default is 1.

See the following for a series of examples. The default interval of 1 microsecond is usually too fast for many systems. Changing the interval shows you the destination system′s performance characteristics.

```
# spray inoki5
sending 1162 packets of length 86 to inoki5 ...

        571 packets (49.139%) dropped by inoki5
        53 packets/second, 4633 bytes/second
# spray -d 2 inoki5
sending 1162 packets of length 86 to inoki5 ...

        62 packets (5.336%) dropped by inoki5
        794 packets/second, 68308 bytes/second
# spray -d 10 inoki5
sending 1162 packets of length 86 to inoki5 ...

        61 packets (5.250%) dropped by inoki5
        796 packets/second, 68493 bytes/second
# spray -d 20 inoki5
sending 1162 packets of length 86 to inoki5 ...

no packets dropped by inoki5
        819 packets/second, 70473 bytes/second
#
```

To analyze a result, pay attention to relativity. Dropping packets can mean that the network is congested and the destination system has no problem. The best way is to try the spray command on several systems and compare the results.

6.2.3 Considerations

All considerations explained for the ping should also be applied to the spray/sprayd.

In addition to the above considerations, the spray is an application and a lot of system environments, such as the sprayd daemon process priority, affect the spray operation and the result. Always keep in mind your are using ONC/RPC on the top of UDP. You should use FTP or some other tools for the TCP performance experiment.

6.3 netpmon

The netpmon is a very convenient tool. This tool uses a system trace function to gather data, and summarizes the results in a very useful format. As you have seen, the netstat command is another good tool. With the netstat you can gather various data for each network interface, protocol, adapter, etc., but it can not show you the relationship between those network activities and processes. The netpmon generates a report which the netstat can not provide. For example, the following information is provided by the netpmon:

- Process CPU usage statistics
- First level interrupt handler CPU usage statistics
- Second level interrupt handler CPU usage statistics
- Network device-driver statistics (by device)
- Network device-driver transmit statistics (by destination Host)
- TCP socket call statistics (by process)
- UDP socket call statistics (by process)
- NFS server statistics (by client)

Difference between V4.1 and V3.2

With AIX V4.1, you need explicit installation of the fileset perfagent.tools, to use netpmon. Not only the netpmon, but some other useful tools are included in this fileset. rmss, svmon, filemon and stem for example.

6.3.1 Operation

The netpmon can be invoked in two ways. There are no substantial differences between the following two methods. One method allows you to invoke the netpmon more interactively. Here is the first method:

1. Invoke the netpmon as follows. You can specify the output report file with the -o flag. If you don′t specify the output file, the standard output is used.

```
# netpmon -o /tmp/netpmon.log

Enter the ″trcstop″ comand to complete netpmon processing

#
```

2. Select something you want to monitor.
3. Stop the netpmon with the trcstop command, as follows:

```
# trcstop
[netpmon: Reporting started]

[netpmon: Reporting completed]

[netpmon: 103.569 seconds in measured interval]
#
```

   The report is automatically generated and placed in the specified file.

   Difference between V4.1 and V3.2

   With V3.2, you have to issue the kill command to stop the netpmon.

```
# kill $(ps -e | grep netpmon | grep -v grep | awk ′{print $1}′)
[netpmon: Reporting started]

[netpmon: Reporting completed]

[netpmon: 43.891 seconds in measured interval]
#
```

4. Use your prefered editor to review the output report file.

This is an alternative method:

1. Invoke the netpmon as follows. You can specify the output report file with the -o flag. If you don′t specify the output file, the standard output is used. With the -d flag, though, the netpmon starts, but it doesn′t start the system trace now.

   ```
   # netpmon -d -o /tmp/netpmon.log
   #
   ```

2. Issue the trcon command to start the system trace function, and how the data is gathered.

   ```
   # trcon
   Enter the ″trcstop″ comand to complete netpmon processing

   #
   ```

3. Select something when you want to monitor.

4. Stop the netpmon with the trcstop command. The report is automatically generated and placed in the specified file.

   ```
   # trcstop
   [netpmon: Reporting started]

   [netpmon: Reporting completed]

   [netpmon: 49.688 seconds in measured interval]
   #
   ```

5. Use your preferred editor to review the output report file.

6.3.2 Report Example

The following is a netpmon sample output report. This sample may not be a good example because only a few network activities were reported. The purpose of this example is to give you a concrete picture of the report format and the content. If you run the netpmon on a busy server system, you may get a huge report file.

```
Sun Aug 20 21:45:48 1995
System: AIX zero Node: 4 Machine: 000970044D00

2246.650 seconds in measured interval

========================================================================

Process CPU Usage Statistics:
-----------------------------
                                                        Network
Process (top 20)             PID  CPU Time   CPU %   CPU %
----------------------------------------------------------
info_gr                    24470 2072.3083  92.240   0.001
X                           9376   48.8685   2.175   0.000
aixterm                    19268   15.5105   0.690   0.000
aixterm                     2618    9.9855   0.444   0.000
vi                         25872    9.8809   0.440   0.000
netpmon                    25592    8.0261   0.357   0.000
gil                         1032    6.0008   0.267   0.267
ksh                        18246    4.1132   0.183   0.000
init                           1    3.5578   0.158   0.000
swapper                        0    2.4563   0.109   0.000
```

```
dtwm                       18484    2.1209   0.094   0.000
trace                      25074    1.8302   0.081   0.000
dlci                       12996    1.6701   0.074   0.000
vi                         18976    1.5983   0.071   0.000
syncd                      11258    0.8424   0.037   0.000
vi                         18982    0.7915   0.035   0.000
aixterm                    21506    0.6088   0.027   0.000
spray                      11002    0.5875   0.026   0.005
vi                         18984    0.4937   0.022   0.000
aixterm                    25856    0.4537   0.020   0.000
==========================================================
Total (all processes)            2199.4901  97.901   0.278
Idle time                          25.9099   1.153

========================================================================

First Level Interrupt Handler CPU Usage Statistics:
---------------------------------------------------
                                                     Network
FLIH                              CPU Time   CPU %   CPU %
----------------------------------------------------------
PPC decrementer                    14.5158   0.646   0.000
data page fault                     1.6037   0.071   0.000
external device                     1.1192   0.050   0.005
UNKNOWN                             0.3680   0.016   0.001
floating point                      0.0406   0.002   0.000
instruction page fault              0.0068   0.000   0.000
----------------------------------------------------------
Total (all FLIHs)                  17.6540   0.786   0.006

========================================================================

Second Level Interrupt Handler CPU Usage Statistics:
----------------------------------------------------
                                                     Network
SLIH                              CPU Time   CPU %   CPU %
----------------------------------------------------------
itokdd                              1.5741   0.070   0.062
mousedd                             1.1444   0.051   0.000
kbddd                               0.8536   0.038   0.000
ncr810dd                            0.1816   0.008   0.000
----------------------------------------------------------
Total (all SLIHs)                   3.7538   0.167   0.062

========================================================================

Network Device-Driver Statistics (by Device):
---------------------------------------------
                        ----------- Xmit -----------   -------- Recv ---------
Device                   Pkts/s  Bytes/s  Util  QLen   Pkts/s  Bytes/s   Demux
------------------------------------------------------------------------------
token ring 0               0.64       66  0.0%10.016     1.46      530  0.0001

========================================================================

Network Device-Driver Transmit Statistics (by Destination Host):
----------------------------------------------------------------
```

```
Host                     Pkts/s  Bytes/s
----------------------------------------
mat.hakozaki.ibm.com       0.54       50
9.68.214.1                 0.10       15
guru.hakozaki.ibm.com      0.00        0
ts.hakozaki.ibm.com        0.00        0

========================================================================

TCP Socket Call Statistics (by Process):
----------------------------------------
                                   ------ Read -----   ----- Write -----
Process (top 20)             PID   Calls/s   Bytes/s   Calls/s   Bytes/s
------------------------------------------------------------------------
tn                         19056      0.02       195      0.02         0
msgchk                     10758      0.00         9      0.00         0
netstat                    25864      0.00         0      0.00         0
------------------------------------------------------------------------
Total (all processes)                 0.02       204      0.02         0

========================================================================

NFS Client Statistics for Server ts.hakozaki.ibm.com (by File):
---------------------------------------------------------------
                         ------ Read -----   ----- Write -----
File (top 20)            Calls/s   Bytes/s   Calls/s   Bytes/s
--------------------------------------------------------------
<vnode=814aa0c>             0.00         9      0.00         0
<vnode=814620c>             0.00         7      0.00         0
<vnode=8149e0c>             0.00         7      0.00         0
<vnode=814fa0c>             0.00         7      0.00         0
<vnode=814680c>             0.00         5      0.00         0
<vnode=8146e0c>             0.00         5      0.00         0
<vnode=814740c>             0.00         5      0.00         0
<vnode=8147a0c>             0.00         5      0.00         0
<vnode=814700c>             0.00         5      0.00         0
<vnode=814860c>             0.00         5      0.00         0
<vnode=8148c0c>             0.00         5      0.00         0
<vnode=814920c>             0.00         5      0.00         0
<vnode=814980c>             0.00         5      0.00         0
<vnode=814a40c>             0.00         5      0.00         0
<vnode=814a00c>             0.00         5      0.00         0
<vnode=814b60c>             0.00         5      0.00         0
<vnode=814bc0c>             0.00         5      0.00         0
<vnode=814c20c>             0.00         5      0.00         0
<vnode=814c80c>             0.00         5      0.00         0
<vnode=814ce0c>             0.00         5      0.00         0
--------------------------------------------------------------
Total (all files)           0.04       157      0.00         0

========================================================================

NFS Client RPC Statistics (by Server):
--------------------------------------

Server                     Calls/s
----------------------------------
ts.hakozaki.ibm.com           0.09
```

```
------------------------------------------------------------------------
Total (all servers)          0.09

========================================================================

NFS Client Statistics (by Process):
-----------------------------------
                                   ------ Read -----   ----- Write -----
Process (top 20)             PID   Calls/s   Bytes/s   Calls/s   Bytes/s
------------------------------------------------------------------------
info_gr                    24470      0.04       164      0.00         0
------------------------------------------------------------------------
Total (all processes)                 0.04       164      0.00         0

========================================================================

Detailed Second Level Interrupt Handler CPU Usage Statistics:
-------------------------------------------------------------

SLIH: itokdd
count:                  6270
  cpu time (msec):      avg 0.251   min 0.020   max 1.443   sdev 0.302

SLIH: mousedd
count:                  22740
  cpu time (msec):      avg 0.050   min 0.009   max 0.254   sdev 0.036

SLIH: kbddd
count:                  6910
  cpu time (msec):      avg 0.124   min 0.023   max 0.274   sdev 0.046

SLIH: ncr810dd
count:                  1739
  cpu time (msec):      avg 0.104   min 0.026   max 0.291   sdev 0.071

COMBINED (All SLIHs)
count:                  37659
  cpu time (ms):        avg 0.100   min 0.009   max 1.443   sdev 0.148

========================================================================

Detailed Network Device-Driver Statistics:
------------------------------------------

DEVICE: token ring 0
recv packets:           3290
  recv sizes (bytes):   avg 362.1   min 50      max 1516    sdev 517.7
  recv times (ms):      avg 0.268   min 0.046   max 1.074   sdev 0.334
  demux times (ms):     avg 0.074   min 0.012   max 0.547   sdev 0.075
xmit packets:           1448
  xmit sizes (bytes):   avg 102.4   min 50      max 166     sdev 24.9
  xmit times (ms):      avg 15540.184 min 0.666   max 1861567.078 sdev 132414.877

========================================================================

Detailed Network Device-Driver Transmit Statistics (by Host):
-------------------------------------------------------------
```

```
HOST: mat.hakozaki.ibm.com
xmit packets:          1213
  xmit sizes (bytes):   avg 93.3    min 52      max 106     sdev 4.0
  xmit times (ms):      avg 991.048 min 0.666   max 129372.221 sdev 10472.941

HOST: 9.68.214.1
xmit packets:          221
  xmit sizes (bytes):   avg 155.0   min 62      max 166     sdev 25.5
  xmit times (ms):      avg 73262.825 min 0.671   max 1574711.153 sdev 284046.818

HOST: guru.hakozaki.ibm.com
xmit packets:          8
  xmit sizes (bytes):   avg 66.8    min 62      max 75      sdev 5.4
  xmit times (ms):      avg 61757.527 min 7.334   max 122283.414 sdev 60487.592

========================================================================

Detailed TCP Socket Call Statistics (by Process):
-------------------------------------------------

PROCESS: tn   PID: 19056
reads:                 41
  read sizes (bytes):   avg 10692.0 min 10692   max 10692   sdev 0.0
  read times (ms):      avg 0.789   min 0.084   max 25.565  sdev 3.918
writes:                37
  write sizes (bytes):  avg 2.2     min 1       max 25      sdev 4.4
  write times (ms):     avg 0.555   min 0.219   max 1.867   sdev 0.263

PROCESS: msgchk   PID: 10758
reads:                 5
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 1018.040 min 2.285   max 5031.800 sdev 2006.968
writes:                4
  write sizes (bytes):  avg 9.5     min 6       max 13      sdev 3.5
  write times (ms):     avg 0.396   min 0.363   max 0.443   sdev 0.032

PROCESS: netstat   PID: 25864
reads:                 4
  read sizes (bytes):   avg 36.0    min 2       max 77      sdev 34.4
  read times (ms):      avg 74.399  min 5.457   max 189.261 sdev 76.188
writes:                2
  write sizes (bytes):  avg 2.0     min 2       max 2       sdev 0.0
  write times (ms):     avg 0.531   min 0.402   max 0.659   sdev 0.129

PROTOCOL: TCP (All Processes)
reads:                 50
  read sizes (bytes):   avg 9179.9  min 2       max 10692   sdev 3338.9
  read times (ms):      avg 108.403 min 0.084   max 5031.800 sdev 703.990
writes:                43
  write sizes (bytes):  avg 2.9     min 1       max 25      sdev 4.8
  write times (ms):     avg 0.539   min 0.219   max 1.867   sdev 0.250

========================================================================

Detailed NFS Client Statistics for Server ts.hakozaki.ibm.com (by File):
------------------------------------------------------------------------

FILE: <vnode=814aa0c>
reads:                 5
```

```
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 28.620  min 0.410   max 42.746  sdev 14.916

FILE: <vnode=814620c>
reads:                  4
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 46.573  min 26.194  max 56.575  sdev 12.079

FILE: <vnode=8149e0c>
reads:                  4
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 53.276  min 27.325  max 87.470  sdev 23.098

FILE: <vnode=814fa0c>
reads:                  4
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 46.538  min 26.950  max 76.927  sdev 18.496

FILE: <vnode=814680c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 49.508  min 27.076  max 78.983  sdev 21.769

FILE: <vnode=8146e0c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 41.686  min 27.334  max 54.589  sdev 11.174

FILE: <vnode=814740c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 35.768  min 21.148  max 59.982  sdev 17.244

FILE: <vnode=8147a0c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 29.672  min 21.223  max 46.558  sdev 11.940

FILE: <vnode=814700c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 46.353  min 42.461  max 48.428  sdev 2.754

FILE: <vnode=814860c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 54.286  min 48.084  max 58.688  sdev 4.512

FILE: <vnode=8148c0c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 26.882  min 21.183  max 38.242  sdev 8.033

FILE: <vnode=814920c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 31.956  min 26.568  max 42.197  sdev 7.245
```

```
FILE: <vnode=814980c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 30.227  min 21.209  max 42.477  sdev 8.978

FILE: <vnode=814a40c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 40.879  min 21.830  max 72.307  sdev 22.389

FILE: <vnode=814a00c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 37.235  min 27.673  max 49.330  sdev 9.021

FILE: <vnode=814b60c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 43.509  min 27.019  max 54.849  sdev 11.931

FILE: <vnode=814bc0c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 35.491  min 21.162  max 47.207  sdev 10.792

FILE: <vnode=814c20c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 27.025  min 21.372  max 37.383  sdev 7.334

FILE: <vnode=814c80c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 41.712  min 27.486  max 49.984  sdev 10.104

FILE: <vnode=814ce0c>
reads:                  3
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 41.609  min 21.172  max 78.935  sdev 26.433

SERVER: ts.hakozaki.ibm.com (All Files)
reads:                  86
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 38.059  min 0.410   max 87.470  sdev 16.704

========================================================================

Detailed NFS Client RPC Statistics (by Server):
-----------------------------------------------

SERVER: ts.hakozaki.ibm.com
calls:                  193
  call times (ms):      avg 24.545  min 5.492   max 58.044  sdev 10.864

COMBINED (All Servers)
calls:                  193
  call times (ms):      avg 24.545  min 5.492   max 58.044  sdev 10.864

========================================================================
Detailed NFS Client Statistics (by Process):
--------------------------------------------

PROCESS: info_gr   PID: 24470
reads:                  90
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 38.259  min 0.410   max 87.470  sdev 16.754

COMBINED (All Processes)
reads:                  90
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (ms):      avg 38.259  min 0.410   max 87.470  sdev 16.754
```

6.4 no and nfso

The no command may not be an accurate tuning or monitoring tool, but many parameters are adjusted with this command. The advantages of the no command are obvious. For example, a change is effective immediately without a system reboot.

There is another command, the nfso, for NFS parameters. The nfso has the same advantage as the no command. This section is for your reference. We just list the options of these commands. Some of them have been explained in this book. You can refer to manuals and InfoExplorer for the remaining options.

6.4.1 no Command of V3.2

```
# no -a
                dog_ticks = 60
                 lowclust = 29
                  lowmbuf = 88
                  thewall = 2048
              mb_cl_hiwat = 58
                compat_43 = 1
                   sb_max = 131072
             detach_route = 1
          subnetsarelocal = 1
                   maxttl = 255
                ipfragttl = 60
          ipsendredirects = 1
             ipforwarding = 1
                  udp_ttl = 30
                  tcp_ttl = 60
               arpt_killc = 20
            tcp_sendspace = 16384
            tcp_recvspace = 16384
            udp_sendspace = 9216
            udp_recvspace = 41600
           loop_check_sum = 1
           rfc1122addrchk = 0
           nonlocsrcroute = 1
            tcp_keepintvl = 150
             tcp_keepidle = 14400
             tcp_keepinit = 150
          icmpaddressmask = 0
                  rfc1323 = 0
               tcp_mssdflt = 512
```

```
       directed_broadcast = 0
                ipqmaxlen = 50
#
```

6.4.2 no Command of V4.1

```
# no -a
                  thewall = 4092
                   sb_max = 131072
         net_malloc_police = 0
                  rto_low = 1
                 rto_high = 64
                rto_limit = 7
               rto_length = 13
              arptab_bsiz = 7
                arptab_nb = 25
               tcp_ndebug = 100
                   ifsize = 8
          subnetsarelocal = 1
                   maxttl = 255
                ipfragttl = 60
          ipsendredirects = 1
             ipforwarding = 0
                  udp_ttl = 30
                  tcp_ttl = 60
               arpt_killc = 20
            tcp_sendspace = 16384
            tcp_recvspace = 16384
            udp_sendspace = 9216
            udp_recvspace = 41600
           rfc1122addrchk = 0
           nonlocsrcroute = 0
            tcp_keepintvl = 150
             tcp_keepidle = 14400
                bcastping = 0
                 udpcksum = 1
              tcp_mssdflt = 512
          icmpaddressmask = 0
             tcp_keepinit = 150
ie5_old_multicast_mapping = 0
                  rfc1323 = 0
                ipqmaxlen = 100
       directed_broadcast = 1
#
```

6.4.3 nfso Command of V3.2

```
# nfso -a
              nfs_portmon = 0
              nfsudpcksum = 0
                nfs_chars = 100000
         nfs_setattr_error = 0
      nfs_gather_threshold = 4096
#
```

Note: You need root privilege to run the nfso command at V3.2.

6.4.4 nfso Command of V4.1

```
# nfso -a
portcheck= 0
udpchecksum= 1
nfs_socketsize= 60000
nfs_setattr_error= 0
nfs_gather_threshold= 4096
nfs_repeat_messages= 0
nfs_duplicate_cache_size= 0
nfs_server_base_priority= 0
nfs_dynamic_retrans= 1
nfs_iopace_pages= 32
#
```

Appendix A. Network Configuration Startup Scripts

A.1 /etc/rc.net

```
#!/bin/ksh
# @(#)90;1.22  src/bos/usr/sbin/netstart/rc.net, cmdnet, bos411, 9428A410j 4/19/94 09:17:52
#
# COMPONENT_NAME: CMDNET;(/etc/rc.net)
#
# ORIGINS: 27
#
# (C) COPYRIGHT International Business Machines Corp. 1985, 1989
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#

##################################################################
# rc.net - called by cfgmgr during 2nd boot phase.
#
# Configures and starts TCP/IP interfaces.
# Sets hostname, default gateway and static routes.
# Note: all the stdout should be redirected to a file (e.g. /dev/null),
#;because stdout is used to pass logical name(s) back to the cfgmgr
#;to be configured.  The LOGFILE variable specifies the output file.
# The first section of rc.net configures the network via the new
#;configuration methods.  These configuration methods require that
#;the interface and protocol information be entered in the ODM
#;database (with either SMIT or the high level configuration commands
#;(mkdev, chdev).
# The second section (commented out) is an example of the equivalent
#;traditional commands used to perform the same function.  You may
#;use the traditional commands instead of the configuration methods
#;if you prefer.  These commands do NOT use the ODM database.
# The third section performs miscellaneous commands which are
#;compatible with either of the previous two sections.
##################################################################
#
# Close file descriptor 1 and 2 because the parent may be waiting
# for the file desc. 1 and 2 to be closed.  The reason is that this shell
# script may spawn a child which inherit all the file descriptor from the parent
# and the child process may still be running after this process is terminated.
# The file desc. 1 and 2 are not closed and leave the parent hanging
# waiting for those desc. to be finished.
#LOGFILE=/dev/null;# LOGFILE is where all stdout goes.
LOGFILE=/tmp/rc.net.out;# LOGFILE is where all stdout goes.
>$LOGFILE;;# truncate LOGFILE.
exec 1<&-;;# close descriptor 1
exec 2<&-;;# close descriptor 2
exec 1< /dev/null;# open descriptor 1
exec 2<;/dev/null;# open descriptor 2



##################################################################
```

```
# Part I - Configuration using the data in the ODM database:
# Enable network interface(s):
##################################################################
# This should be done before routes are defined.
# For each network adapter that has already been configured, the
# following commands will define, load and configure a corresponding
# interface.
# NOTE: If you are using a diskless/dataless machine, you may want to
#       disable the logging of messages to the LOGFILE by the cfgif
#       routine. On some diskless/dataless machines, the message
#       logging causes the client to hang on LED 581 when booting.

/usr/lib/methods/defif ;;;>>$LOGFILE 2>&1
/usr/lib/methods/cfgif  $*;;;>>$LOGFILE 2>&1
# If a diskless or dataless machine uses this configuration method, you
# may want to replace the previous line with the following.
#
# /usr/lib/methods/cfgif  $*

##################################################################
# Configure the Internet protocol kernel extension (netinet):
##################################################################
# The following commands will also set hostname, default gateway,
# and static routes as found in the ODM database for the network.
/usr/lib/methods/definet ;;;>>$LOGFILE 2>&1
/usr/lib/methods/cfginet ;;;>>$LOGFILE 2>&1


##################################################################
# Part II - Traditional Configuration.
##################################################################
# An alternative method for bringing up all the default interfaces
# is to specify explicitly which interfaces to configure using the
# ifconfig command.  Ifconfig requires the configuration information
# be specified on the command line.  Ifconfig will not update the
# information kept in the ODM configuration database.
#
# Valid network interfaces are:
# lo=local loopback, en=standard ethernet, et=802.3 ethernet
# sl=serial line IP, tr=802.5 token ring, xs=X.25
#
# e.g., en0 denotes standard ethernet network interface, unit zero.
#
# Below are examples of how you could bring up each interface using
# ifconfig.  Since you can specify either a hostname or a dotted
# decimal address to set the interface address, it is convenient to
# set the hostname at this point and use it for the address of
# an interface, as shown below:
#
#/bin/hostname robo.austin.ibm.com;>>$LOGFILE 2>&1
#
# (Remember that if you have more than one interface,
# you'll want to have a different IP address for each one.
# Below, xx.xx.xx.xx stands for the internet address for the
# given interface).
#
#/usr/sbin/ifconfig lo0 inet loopback    up >>$LOGFILE 2>&1
#/usr/sbin/ifconfig en0 inet hostname  up >>$LOGFILE 2>&1
#/usr/sbin/ifconfig et0 inet xx.xx.xx.xx  up >>$LOGFILE 2>&1
```

```
#/usr/sbin/ifconfig tr0 inet xx.xx.xx.xx  up >>$LOGFILE 2>&1
#/usr/sbin/ifconfig sl0 inet xx.xx.xx.xx  up >>$LOGFILE 2>&1
#/usr/sbin/ifconfig xs0 inet xx.xx.xx.xx  up >>$LOGFILE 2>&1
#
#
# Now we set any static routes.
#
# /usr/sbin/route add 0 gateway ;;>>$LOGFILE 2>&1
# /usr/sbin/route add 192.9.201.0 gateway ;>>$LOGFILE 2>&1


##################################################################
# Part III - Miscellaneous Commands.
##################################################################
# Set the hostid and uname to hostname, where hostname has been
# set via ODM in Part I, or directly in Part II.
# (Note it is not required that hostname, hostid and uname all be
# the same).
/usr/sbin/hostid hostname;;>>$LOGFILE 2>&1
/bin/uname -Shostname|sed 's/\..*$//';>>$LOGFILE 2>&1

##################################################################
#  Special SLIP handling
##################################################################
if [ -f /etc/rc.net.serial ] ; then
;/etc/rc.net.serial
fi

##################################################################
#  Special X25 handling
##################################################################
if [ -f /etc/rc.net.x25 ] ; then
;/etc/rc.net.x25
fi

###################################################
# The socket default buffer size (initial advertized TCP window) is being
# set to a default value of 16k (16384). This improves the performance
# for ethernet and token ring networks.  Networks with lower bandwidth
# such as SLIP (Serial Line Internet Protocol) and X.25 or higher bandwidth
# such as Serial Optical Link and FDDI would have a different optimum
# buffer size.
# ( OPTIMUM WINDOW = Bandwidth * Round Trip Time )
###################################################
if [ -f /usr/sbin/no ] ; then
;/usr/sbin/no -o tcp_sendspace=16384
;/usr/sbin/no -o tcp_recvspace=16384
fi
```

A.2 Startup Script /etc/rc.bsdnet

```
#!/bin/ksh
# @(#)36;1.6  src/bos/usr/sbin/netstart/rc.bsdnet, cmdnet, bos411, 9428A410j 4/19/94 09:16:37
#
# COMPONENT_NAME: CMDNET;(/etc/rc.bsdnet)
#
# ORIGINS: 27
#
```

```
# (C) COPYRIGHT International Business Machines Corp. 1985, 1991
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
LOGFILE=/tmp/rc.net.out;    # LOGFILE is where all stdout goes.
>$LOGFILE;;    # truncate LOGFILE.


/bin/hostname aoot.austin.ibm.com ;;>>$LOGFILE 2>&1

##################################################################
# Valid network interfaces are:
# lo=local loopback, en=standard ethernet, et=802.3 ethernet
# sl=serial line IP, tr=802.5 token ring, xs=X.25
##################################################################

/usr/sbin/ifconfig lo0 inet 127.0.0.1 up ;>>$LOGFILE 2>&1
/usr/sbin/ifconfig en0 inet hostname up ;>>$LOGFILE 2>&1

#/usr/sbin/route add 0 gateway ;;;>>$LOGFILE 2>&1
#/usr/sbin/route add 192.9.201.0 gateway ;>>$LOGFILE 2>&1

/usr/sbin/hostid hostname;;;>>$LOGFILE 2>&1
/bin/uname -Shostname|sed ′s/\..*$//′;;>>$LOGFILE 2>&1

##################################################################
#  Special SLIP handling
##################################################################
if [ -f /etc/rc.net.serial ] ; then
;/etc/rc.net.serial
fi

##################################################################
#  Special X25 handling
##################################################################
if [ -f /etc/rc.net.x25 ] ; then
;/etc/rc.net.x25
fi

##################################################
# The socket default buffer size (initial advertized TCP window) is being
# set to a default value of 16k (16384). This improves the performance
# for ethernet and token ring networks.  Networks with lower bandwidth
# such as SLIP (Serial Line Internet Protocol) and X.25 or higher bandwidth
# such as Serial Optical Link and FDDI would have a different optimum
# buffer size.
# ( OPTIMUM WINDOW = Bandwidth * Round Trip Time )
##################################################
if [ -f /usr/sbin/no ] ; then
;/usr/sbin/no -o tcp_sendspace=16384
;/usr/sbin/no -o tcp_recvspace=16384
fi
```

A.3 Startup Script /etc/rc.tcpip

```
#! /bin/bsh
# @(#)95       1.55  src/tcpip/etc/rc.tcpip, tcpip, tcpip411, GOLD410 6/24/94 11:19:36
#
# COMPONENT_NAME: TCPIP rc.tcpip
#
# FUNCTIONS:
#
# ORIGINS: 26  27
#
# (C) COPYRIGHT International Business Machines Corp. 1985, 1994
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
####################################################################
# rc.tcpip -
#;assumes interfaces are brought up by /etc/rc.net
#;starts TCP/IP daemons (sendmail, inetd, etc.)
####################################################################
# start -
#;starts daemons using either src or command-line method
# args:
#;$1: pathname of daemon
#;$2: non-null if we should use src to start the daemon
#;$3: any arguments to pass it
#
start()
{
;# just return if the daemon doesn′t exist
;#
;[ -x $1 ] || return 0

;# start the daemon using either src or command-line method
;#
;cmd=basename $1
;if [ -n ″$2″ ] ; then
;;startsrc -s $cmd -a ″$3″
;else
;;if [ $cmd = ″portmap″ ] ; then
;;;$1 $3 &; # portmap must start in background
;;else
;;;$1 $3
;;fi
;;echo ″\t$cmd″
;fi
}

# check the bootup_option flag in the configuration database
option=lsattr -E -l inet0 -a bootup_option -F value
if [ ″$option″ = ″no″  ]
then
####################################################################
#
# Check to see if srcmstr is running; if so, we try to use it;
# otherwise, we start the daemons without src
#
i=3  # make sure init has time to start it
while [ $i != 0 ] ; do
;if [ -n ″ps -e | awk ′$NF == ″srcmstr″ { print $1; exit }′″ ] ; then
;;src_running=1  # set flag
;;break
;fi
;i=expr $i - 1  # decrement count
done

# If srcmstr is running, ensure that it is active before issuing the
# startsrc commands
#
if [ -n ″$src_running″ ] ; then
;echo ″Checking for srcmstr active...\c″
;i=10  # try ten times to contact it
;while [ $i != 0 ] ; do
;;lssrc -s inetd >/dev/null 2>&1 && break  # break out on success
;;sleep 1  # otherwise wait a second and try again
;;echo ″.\c″
;;i=expr $i - 1  # decrement count
;done
;if [ $i = 0 ] ; then
;;echo ″\n\nERROR: srcmstr is not accepting connections.\n″
;;exit 1
;fi
;echo ″complete″
fi

else
;src_running=″″
fi
# Start up the daemons
#
echo ″Starting tcpip daemons:″

# Start up syslog daemon (for error and event logging)
start /usr/sbin/syslogd ″$src_running″

# Start up print daemon
#start /usr/sbin/lpd ″$src_running″

# Start up routing daemon (only start ONE)
#start /usr/sbin/routed ″$src_running″ -q
#start /usr/sbin/gated ″$src_running″

# Start up the sendmail daemon.
#
# Sendmail will automatically build the configuration and alias
# data bases the first time it is invoked.  You may wish to update
# the alias source file /usr/lib/aliases with local information,
# and then rebuild the alias data base by issuing the command
# ″/usr/lib/sendmail -bi″ or ″/usr/ucb/newaliases″.
#
# When the configuration or alias data bases are changed, the
# sendmail daemon can be made to rebuild and re-read them by
# issuing the command ″kill -1 cat /etc/sendmail.pid″ or, if
# SRC was used to start the daemon, ″refresh -s sendmail″.
```

```
#
# The "qpi", or queue processing interval, determines how
# frequently the daemon processes the message queue.
#
qpi=30m  # 30 minute interval
#
start /usr/lib/sendmail "$src_running" "-bd -q${qpi}"

# Start up Portmapper
USR_NFS=mount | awk '{if($3=="/usr") print $4}'
if [ "$USR_NFS" != "nfs" ]
then
REMOTE_USR="N"
start /usr/sbin/portmap "$src_running"
fi

# Start up socket-based daemons
start /usr/sbin/inetd "$src_running"

# Start up Domain Name daemon
#start /usr/sbin/named "$src_running"

# Start up time daemon
#start /usr/sbin/timed "$src_running"

# Start up rwhod daemon (a time waster)
#start /usr/sbin/rwhod "$src_running"

# Start up the Simple Network Management Protocol (SNMP) daemon
#start /usr/sbin/snmpd "$src_running"

```

A.4 Startup Script /etc/rc.nfs

```
#!/bin/bsh
# @(#)47;1.34  src/nfs/etc/rc.nfs.sh, cmdnfs, nfs411, GOLD410 5/31/94 08:05:21
# COMPONENT_NAME: (CMDNFS) Network File System Commands
#
# FUNCTIONS:
#
# ORIGINS: 27
#
# (C) COPYRIGHT International Business Machines Corp. 1988, 1993
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#

#
# start() has the following logic
# 1) if srcmstr is running, use it to start the daemon
# 2) if srcmstr is NOT running, lookup the daemon path and arguments from
# srcmstr's config database in odm (SRCsubsys).
# 3) if the config info can not be found, attempt to start the daemon
# from the default parameters passed into start().
#
```

```
start()
{

daemon=$1 ;;;;;    # Subsystem name
default_path=$2;;;;;    # full path w/ cmdname
shift;shift
default_arg=$*;;;;;    # default arguments

#
# get the path to the daemon from the SRC ODM config info
#
daemon_path=odmget -q subsysname=$daemon SRCsubsys 2>/dev/null | \
      awk ′ $1 == ″path″ { print $NF }′ 2>/dev/null | sed ′s/″//g′

#
# if daemon_path not set (length zero) then try synonymname
#
if [ -z ″$daemon_path″ ] ; then ;
;daemon_path=odmget -q synonym=$daemon SRCsubsys 2>/dev/null | \
      ;;awk ′ $1 == ″path″ { print $NF }′ 2>/dev/null | sed ′s/″//g′
fi

#
# get the arguments to the daemon from the SRC ODM config info
#
cmdargs=odmget -q subsysname=$daemon SRCsubsys 2>/dev/null | \
      awk ′$1  == ″cmdargs″ { print $NF }′ 2>/dev/null | sed ′s/″//g′


if [ -n ″$src_running″ -a -n ″$daemon_path″ ] ; then ;
;#
;#if srcmstr is running and there is an entry in SRCsubsys - use src
;#
;startsrc -s $daemon
else;;;;;#if srcmstr not running, start manually
;if [ -n ″$daemon_path″ ] ; then
;;if [ -n ″$cmdargs″ ] ; then
;;;$daemon_path $cmdargs  &;;# issue cmd
;;else
;;;$daemon_path $default_arg  & ;# issue cmd
;;fi
;else
;;$default_path $default_arg &  ;;#issue cmd
;fi
;
fi

} ;# end start()

#
# determine of srcmstr is running
#
if [ -n ″ps -e | awk ′$NF == ″srcmstr″ {print $1} ′″ ] ; then
       src_running=1
else
       src_running=″″
fi


```

```
# Check the mount of /.  If it is remote, do not start statd,lockd.
REMOTE_ROOT=″N″
/usr/sbin/mount | /usr/bin/grep ′ /  *jfs ′ 2>&1 > /dev/null
if [ ″$?″ != 0 ]
then
;REMOTE_ROOT=″Y″
fi

# Check the mount of /usr.  If it is remote, do not start statd.
REMOTE_USR=″N″
/usr/sbin/mount | /usr/bin/grep ′ /usr *jfs ′ 2>&1 > /dev/null
if [ ″$?″ != 0 ]
then
;REMOTE_USR=″Y″
fi

# Uncomment the following lines  and change the domain
# name to define your domain (domain must be defined
# before starting NIS).
#if [ -x /usr/bin/domainname ]; then
#;/usr/bin/domainname ibm
#fi

#
# Clear all servers′ rmtab files in case we went down abnormally.
#
if [ -s /sbin/helpers/nfsmnthelp ]; then
;/sbin/helpers/nfsmnthelp B 0
fi

#dspmsg cmdnfs.cat -s 8 2 ″starting NIS services:\n″
#if [ -x /usr/lib/netsvc/yp/ypserv -a -d /var/yp/domainname ]; then
#;start ypserv /usr/lib/netsvc/yp/ypserv
#fi

#if [ -x /usr/lib/netsvc/yp/ypbind ]; then
#;start ypbind /usr/lib/netsvc/yp/ypbind
#fi

#if [ -x /usr/sbin/keyserv ]; then
#;start keyserv /usr/sbin/keyserv
#fi

#if [ -x /usr/lib/netsvc/yp/rpc.ypupdated -a -d /var/yp/domainname ]; then
#;start ypupdated /usr/lib/netsvc/yp/rpc.ypupdated
#fi

dspmsg cmdnfs.cat -s 8 1 ″starting nfs services:\n″
if [ -x /usr/sbin/biod ]; then
;start biod /usr/sbin/biod 8
fi

#
# If nfs daemon is executable and /etc/exports, become nfs server.
#
if [ -x /usr/sbin/nfsd -a -f /etc/exports ]; then
;> /etc/xtab
;/usr/sbin/exportfs -a
;start nfsd /usr/sbin/nfsd 8
```

```
;start rpc.mountd /usr/sbin/rpc.mountd
fi

#
# start up status monitor and locking daemon if present
#
if [ -x /usr/sbin/rpc.statd -a $REMOTE_ROOT = ″N″ -a $REMOTE_USR = ″N″ ]; then
;start rpc.statd /usr/sbin/rpc.statd
fi

if [ -x /usr/sbin/rpc.lockd -a $REMOTE_ROOT = ″N″ ]; then
;start rpc.lockd /usr/sbin/rpc.lockd
fi

#
#Uncomment the following lines to start up the NIS
#yppasswd daemon.
#DIR=/etc
#if [ -x /usr/lib/netsvc/yp/rpc.yppasswdd -a -f $DIR/passwd ]; then
#;start rpc.yppasswdd /usr/lib/netsvc/yp/rpc.yppasswdd /etc/passwd -m
#fi

```

A.5 Startup Script /etc/inittab

```
: @(#)49  1.28.2.7  src/bos/etc/inittab/inittab, cmdoper, bos411, 9430C411a 7/26/94 16:27:45
:
:  COMPONENT_NAME: CMDOPER
:
:  ORIGINS: 3, 27
:
:  (C) COPYRIGHT International Business Machines Corp. 1989, 1993
:  All Rights Reserved
:  Licensed Materials - Property of IBM
:
:  US Government Users Restricted Rights - Use, duplication or
:  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
:
: Note - initdefault and sysinit should be the first and second entry.
:
init:2:initdefault:
brc::sysinit:/sbin/rc.boot 3 >/dev/console 2>&1 # Phase 3 of system boot
powerfail::powerfail:/etc/rc.powerfail 2>&1 | alog -tboot > /dev/console # Power Failure Detection
rc:2:wait:/etc/rc 2>&1 | alog -tboot > /dev/console # Multi-User checks
fbcheck:2:wait:/usr/sbin/fbcheck 2>&1 | alog -tboot > /dev/console # run /etc/firstboot
srcmstr:2:respawn:/usr/sbin/srcmstr # System Resource Controller
rcsna:2:wait:/etc/rc.sna > /dev/console 2>&1 # Start sna daemons
rctcpip:2:wait:/etc/rc.tcpip > /dev/console 2>&1 # Start TCP/IP daemons
rcnfs:2:wait:/etc/rc.nfs > /dev/console 2>&1 # Start NFS Daemons
rchttpd:2:wait:/etc/rc.httpd > /dev/console 2>&1 # Start HTTP daemon
:lafs:2:once:/usr/vice/etc/lafs
rcx25:2:wait:/etc/rc.net.x25 > /dev/console 2>&1 # Load X.25 translation table
cron:2:respawn:/usr/sbin/cron
piobe:2:wait:/usr/lib/lpd/pio/etc/pioinit >/dev/null 2>&1  # pb cleanup
qdaemon:2:wait:/usr/bin/startsrc -sqdaemon
writesrv:2:wait:/usr/bin/startsrc -swritesrv
uprintfd:2:respawn:/usr/sbin/uprintfd
infod:2:once:startsrc -s infod
```

```
dt:2:wait:/etc/rc.dt
cons:0123456789:respawn:/usr/sbin/getty /dev/console
diagd:2:once:/usr/lpp/diagnostics/bin/diagd >/dev/console 2>&1
tty0:2:off:/usr/sbin/getty /dev/tty0
hcon:2:once:/etc/rc.hcon
```

A.6 Configuration File /etc/inetd.conf

```
## @(#)62;1.17.1.6  src/tcpip/etc/inetd.conf, tcpip, tcpip411, GOLD410 7/13/94 10:53:12
##
## COMPONENT_NAME: TCPIP inetd.conf
##
## FUNCTIONS:
##
## ORIGINS: 26  27
##
## (C) COPYRIGHT International Business Machines Corp. 1993
## All Rights Reserved
## Licensed Materials - Property of IBM
##
## US Government Users Restricted Rights - Use, duplication or
## disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
##
#######################################################################
##
##                Internet server configuration database
##
##;Services can be added and deleted by deleting or inserting a
##;comment character (ie. #) at the beginning of a line  If inetd
##;is running under SRC control then the "refresh -s inetd" command
##;needs to be executed for inetd to re-read the inetd.conf file.
##
##;NOTE: The TCP/IP servers do not require SRC and may be started
##;by invoking the service directly (i.e. /etc/inetd). If inetd
##;has been invoked directly, after modifying this file, send a
##;hangup signal, SIGHUP to inetd (ie. kill -1 "pid_of_inetd").
##
##;NOTE: The services with socket type of "sunrpc_tcp" and "sunrpc_udp"
##;require that the portmap daemon be running.
##;Also please use ## to designate comments in this file so that
## ;the smit commands can edit this file correctly.
##
## service  socket  protocol  wait/  user    server    server program
##  name     type             nowait         program     arguments
##
ftp     stream  tcp     nowait  root    /usr/sbin/ftpd        ftpd
telnet  stream  tcp     nowait  root    /usr/sbin/telnetd     telnetd
shell   stream  tcp     nowait  root    /usr/sbin/rshd        rshd
login   stream  tcp     nowait  root    /usr/sbin/rlogind     rlogind
exec    stream  tcp     nowait  root    /usr/sbin/rexecd      rexecd
#comsat dgram   udp     wait    root    /usr/sbin/comsat      comsat
#uucp   stream  tcp     nowait  root    /usr/sbin/uucpd       uucpd
#bootps;dgram   udp     wait    root    /usr/sbin/bootpd      bootpd /etc/bootptab
##
## Finger, systat and netstat give out user information which may be
## valuable to potential "system crackers."  Many sites choose to disable
## some or all of these services to improve security.
##
```

```
51;#finger stream  tcp     nowait  nobody  /usr/sbin/fingerd     fingerd
52;#systat;stream;tcp;nowait;nobody;/usr/bin/ps           ps -ef
53;#netstat stream;tcp;nowait;nobody;/usr/bin/netstat      netstat -f inet
54;#
55;#tftp; dgram;udp;nowait;nobody;/usr/sbin/tftpd;;tftpd -n
56;#talk   dgram   udp     wait    root    /usr/sbin/talkd         talkd
57;ntalk   dgram   udp     wait    root    /usr/sbin/talkd         talkd
58;#
59;# rexd uses very minimal authentication and many sites choose to disable
60;# this service to improve security.
61;#
62;#rexd; sunrpc_tcp;tcp;wait;root;/usr/sbin/rpc.rexd rexd 100017 1
63;rstatd; sunrpc_udp;udp;wait;root;/usr/sbin/rpc.rstatd rstatd 100001 1-3
64;rusersd sunrpc_udp;udp;wait;root;/usr/lib/netsvc/rusers/rpc.rusersd rusersd 100002 1-2
65;rwalld; sunrpc_udp;udp;wait;root;/usr/lib/netsvc/rwall/rpc.rwalld rwalld 100008 1
66;sprayd; sunrpc_udp;udp;wait;root;/usr/lib/netsvc/spray/rpc.sprayd sprayd 100012 1
67;pcnfsd; sunrpc_udp;udp;wait ;root;/usr/sbin/rpc.pcnfsd pcnfsd 150001 1-2
68;echo;stream;tcp;nowait;root;internal
69;discard;stream;tcp;nowait;root;internal
70;chargen;stream;tcp;nowait;root;internal
71;daytime;stream;tcp;nowait;root;internal
72;time;stream;tcp;nowait;root;internal
73;echo;dgram;udp;wait;root;internal
74;discard;dgram;udp;wait;root;internal
75;chargen;dgram;udp;wait;root;internal
76;daytime;dgram;udp;wait;root;internal
77;time;dgram;udp;wait;root;internal
78;## The following line is for installing over the network.
79;#instsrv stream;tcp;nowait;netinst;/u/netinst/bin/instsrv instsrv -r
/tmp/netinstalllog /u/netinst/scripts
80;dtspc;stream;tcp;nowait;root;/usr/dt/bin/dtspcd /usr/dt/bin/dtspcd
81;cmsd;sunrpc_udp;udp;wait;root;/usr/dt/bin/rpc.cmsd cmsd 100068 2-4
```

Appendix B. Well-Known Numbers

B.1 Well-Known Protocols in /etc/protocols

```
 1;# @(#)26;1.6  src/bos/etc/protocols/protocols, cmdnet, bos411, 9428A410j 5/6/94 13:38:13
 2;#
 3;# COMPONENT_NAME: (CMDNET) Network commands.
 4;#
 5;# FUNCTIONS:
 6;#
 7;# ORIGINS: 27
 8;#
 9;# (C) COPYRIGHT International Business Machines Corp. 1988, 1989
10;# All Rights Reserved
11;# Licensed Materials - Property of IBM
12;#
13;# US Government Users Restricted Rights - Use, duplication or
14;# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
15;#
16;# /etc/protocols file for AIX
17;#
18;# offical name, protocol number, aliases
19;
20;ip;;0;IP;;# dummy for the Internet Protocol
21;icmp;;1;ICMP;;# Internet control message protocol
22;igmp;;2;IGMP;;# internet group multicast protocol
23;# ggp;;3;GGP;;# gateway-gateway protocol
24;# ip;;4;IP;;# IP in IP (encapsulation)
25;# st;;5;ST;;# Stream
26;tcp;;6;TCP;;# transmission control protocol
27;# ucl;;7;UCL;;# UCL
28;# egp;;8;EGP;;# exterior gateway protocol
29;# igp;;9;IGP;;# any private interior gateway
30;# bbn-rcc-mon;10;BBN-RCC-MON;# BBN RCC Monitoring
31;# nvp-ii;11;NVP-II;;# Network Voice Protocol
32;# pup;;12;PUP;;# PARC universal packet protocol
33;# argus;;13;ARGUS;;# ARGUS
34;# emcon;;14;EMCON;;# EMCON
35;# xnet;;15;XNET;;# XNET
36;# chaos;;16;CHAOS;;# CHAOS
37;udp;;17;UDP;;# user datagram protocol
38;# mux;;18;MUX;;# Multiplexing
39;# dcn-meas;19;DCN-MEAS;# DCN Measurement Subsystems
40;# hmp;;20;HMP;;# Host Monitoring
41;# prm;;21;PRM;;# Packet Radio Measurement
42;# idp;;22;IDP;;# xns idp
43;# trunk-1;23;TRUNK-1;;# Trunk-1
44;# trunk-2;24;TRUNK-2;;# Trunk-2
45;# leaf-1;25;LEAF-1;;# Leaf-1
46;# leaf-2;26;LEAF-2;;# Leaf-2
47;# rdp;;27;RDP;;# Reliable Data Protocol
48;# irtp;;28;IRTP;;# Internet Reliable Transaction
49;# iso-tp4;29;ISO-TP4;;# ISO Transport Protocol Class 4
50;# netblt;30;NETBLT;;# Bulk Data Transfer Protocol
51;# mfe-nsp;31;MFE-NSP;;# MFE Network Services Protocol
52;# merit-inp;32;MERIT-INP;# Merit Internodal Protocol
53;# sep;;33;SEP;;# Sequential Exchange Protocol
```

```
# 3pc;;34;3PC;;# Third Party Connect Protocol
# idpr;;35;IDPR;;# Inter-Domain Policy Routing Protocol
# xtp;;36;XTP;;# XTP
# ddp;;37;DDP;;# Datagram Delivery Protocol
# idpr-cmtp;38;IDPR-CMTP;# IDPR Control Message Transport Proto
# tp++;;39;TP++;;# TP++ Transport Protocol
# il;;40;IL;;# IL Transport Protocol
#;;41-60;;;# Unassigned
# ;;61;;;# any host internal protocol
# cftp;;62;CFTP;;# CFTP
# ;;63;;;# any local network
# sat-expak;64;SAT-EXPAK;# SATNET and Backroom EXPAK
# kryptolan;65;KRYPTOLAN;# Kryptolan
# rvd;;66;RVD;;# MIT Remote Virtual Disk Protocol
# ippc;;67;IPPC;;# Internet Pluribus Packet Core
# ;;68;;;# any distributed file system
# sat-mon;69;SAT-MON;;# SATNET Monitoring
# visa;;70;VISA;;# VISA Protocol
# ipcv;;71;IPCV;;# Internet Packet Core Utility
# cpnx;;72;CPNX;;# Computer Protocol Network Executive
# cphb;;73;CPHB;;# Computer Protocol Heart Beat
# wsn;;74;WSN;;# Wang Span Network
# pvp;;75;PVP;;# Packet Video Protocol
# br-sat-mon;76;BR-SAT-MON;# Backroom SATNET Monitoring
# sun-nd;77;SUN-ND;;# SUN ND PROTOCOL-Temporary
# wb-mon;78;WB-MON;;# WIDEBAND Monitoring
# wb-expak;79;WB-EXPAK;# WIDEBAND EXPAK
# iso-ip;80;ISO-IP;;# ISO Internet Protocol
# vmtp;;81;VMTP;;# VMTP
# secure-vmtp;82;SECURE-VMTP;# SECURE-VMTP
# vines;;83;VINES;;# VINES
# ttp;;84;TTP;;# TTP
# nfsnet-igp;85;NSFNET-IGP;# NSFNET-IGP
# dgp;;86;DGP;;# Dissimilar Gateway Protocol
# tcf;;87;TCF;;# TCF
# igrp;;88;IGRP;;# IGRP
# ospfigp;89;OSPFIGP;;# OSPFIGP
# sprite-rpc;90;Sprite-RPC;# Sprite RPC Protocol
# larp;;91;LARP;;# Locus Address Resolution Protocol
# mtp;;92;MTP;;# Multicast Transport Protocol
# ax.25;;93;AX.25;;# AX.25 Frames
# ipip;;94;IPIP;;# IP-within-IP Encapsulation Protocol
# micp;;95;MICP;;# Mobile Internetworking Control Pro.
# aes-sp3-d;96;AES-SP3-D;# AES Security Protocol 3-D
# etherip;97;ETHERIP;;# Ethernet-within-IP Encapsulation
# encap;;98;ENCAP;;# Encapsulation Header
# ;;99-254;;;# Unassigned
# ;;255;;;# Reserved
```

B.2 Well-Known Ports in /etc/services

```
# @(#)27;1.17  src/bos/etc/services/services, cmdnet, bos411, 9428A410j 5/20/94 09:56:14
#
# COMPONENT_NAME: (CMDNET) Network commands.
#
# FUNCTIONS:
#
# ORIGINS: 26 27
```

```
 8;#
 9;# (C) COPYRIGHT International Business Machines Corp. 1988, 1989
10;# All Rights Reserved
11;# Licensed Materials - Property of IBM
12;#
13;# US Government Users Restricted Rights - Use, duplication or
14;# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
15;#
16;#
17;# Network services, Internet style
18;#
19;tcpmux;;1/tcp    ;;;# TCP Port Service Multiplexer
20;tcpmux;;1/udp    ;;;# TCP Port Service Multiplexer
21;compressnet;2/tcp    ;;;# Management Utility
22;compressnet;2/udp    ;;;# Management Utility
23;compressnet;3/tcp    ;;;# Compression Process
24;compressnet;3/udp    ;;;# Compression Process
25;echo;;7/tcp
26;echo;;7/udp
27;discard;;9/tcp;;sink null
28;discard;;9/udp;;sink null
29;systat;;11/tcp;;users
30;daytime;;13/tcp
31;daytime;;13/udp
32;netstat;;15/tcp
33;qotd;;17/tcp;;quote
34;msp;;18/tcp;;;;# Message Send Protocol
35;msp;;18/udp;;;;# Message Send Protocol
36;chargen;;19/tcp;;ttytst source
37;chargen;;19/udp;;ttytst source
38;ftp-data;20/tcp
39;ftp;;21/tcp
40;telnet;;23/tcp
41;smtp;;25/tcp;;mail
42;nsw-fe;;27/tcp;;;;# NSW User System FE
43;nsw-fe;;27/udp;;;;# NSW User System E
44;msg-icp;;29/tcp;;;;# MSG ICP
45;msg-icp;;29/udp;;;;# MSG ICP
46;msg-auth;31/tcp;;;;# MSG Authentication
47;msg-auth;31/udp;;;;# MSG Authentication
48;dsp;;33/tcp;;;;# Display Support Protocol
49;dsp;;33/udp;;;;# Display Support Protocol
50;time;;37/tcp;;timserver
51;time;;37/udp;;timserver
52;rlp;;39/udp;;resource;# resource location
53;graphics;41/tcp;;;;# Graphics
54;graphics;41/udp;;;;# Graphics
55;nameserver;42/udp;;name;;# IEN 116
56;whois;;43/tcp;;nicname
57;mpm-flags;44/tcp;;;;# MPM FLAGS Protocol
58;mpm-flags;44/udp;;;;# MPM FLAGS Protocol
59;mpm;;45/tcp;;;;# Message Processing Module
60;mpm;;45/udp;;;;# Message Processing Module
61;mpm-snd;;46/tcp;;;;# MPM ·default send‘
62;mpm-snd;;46/udp;;;;# MPM ·default send‘
63;ni-ftp;;47/tcp;;;;# NI FTP
64;ni-ftp;;47/udp;;;;# NI FTP
65;re-mail-ck;50/tcp;;;;# Remote Mail Checking Protocol
66;re-mail-ck;50/udp;;;;# Remote Mail Checking Protocol
```

```
 67;la-maint;51/tcp;;;;# IMP Logical Address Maint
 68;la-maint;51/udp;;;;# IMP Logical Address Maint
 69;xns-time;52/tcp;;;;# XNS Time Protocol
 70;xns-time;52/udp;;;;# XNS Time Protocol
 71;domain;;53/tcp;;nameserver;# name-domain server
 72;domain;;53/udp;;nameserver
 73;xns-ch;;54/tcp;;;;# XNS Clearinghouse
 74;xns-ch;;54/udp;;;;# XNS Clearinghouse
 75;isi-gl;;55/tcp;;;;# ISI Graphics Language
 76;isi-gl;;55/udp;;;;# ISI Graphics Language
 77;xns-auth;56/tcp;;;;# XNS Authentication
 78;xns-auth;56/udp;;;;# XNS Authentication
 79;mtp;;57/tcp;;;;# deprecated
 80;xns-mail;58/tcp;;;;# XNS Mail
 81;xns-mail;58/udp;;;;# XNS Mail
 82;ni-mail;;61/tcp;;;;# NI MAIL
 83;ni-mail;;61/udp;;;;# NI MAIL
 84;acas;;62/tcp;;;;# ACA Services
 85;acas;;62/udp;;;;# ACA Services
 86;via-ftp;;63/tcp;;;;# VIA Systems - FTP
 87;via-ftp;;63/udp;;;;# VIA Systems - FTP
 88;covia;;64/tcp;;;;# Communications Integrator
 89;covia;;64/udp;;;;# Communications Integrator
 90;tacacs-ds;65/tcp;;;;# TACACS-Database Service
 91;tacacs-ds;65/udp;;;;# TACACS-Database Service
 92;sql*net;;66/tcp;;;;# Oracle SQL*NET
 93;sql*net;;66/udp;;;;# Oracle SQL*NET
 94;bootps;;67/udp;;;;# bootp server port
 95;bootpc;;68/udp;;;;# bootp client port
 96;tftp;;69/udp
 97;gopher;;70/tcp;;;;# Gopher
 98;gopher;;70/udp;;;;# Gopher
 99;rje;;77/tcp;;netrjs
100;vettcp;;78/tcp;;;;# vettcp
101;vettcp;;78/udp;;;;# vettcp
102;finger;;79/tcp
103;www;;80/tcp;;;;# World Wide Web HTTP
104;www;;80/udp;;;;# World Wide Web HTTP
105;hosts2-ns;81/tcp;;;;# HOSTS2 Name Server
106;hosts2-ns;81/udp;;;;# HOSTS2 Name Server
107;xfer;;82/tcp;;;;# XFER Utility
108;xfer;;82/udp;;;;# XFER Utility
109;mit-ml-dev;83/tcp;;;;# MIT ML Device
110;mit-ml-dev;83/udp;;;;# MIT ML Device
111;ctf;;84/tcp;;;;# Common Trace Facility
112;ctf;;84/udp;;;;# Common Trace Facility
113;mit-ml-dev;85/tcp;;;;# MIT ML Device
114;mit-ml-dev;85/udp;;;;# MIT ML Device
115;mfcobol;;86/tcp;;;;# Micro Focus Cobol
116;mfcobol;;86/udp;;;;# Micro Focus Cobol
117;link;;87/tcp;;ttylink
118;kerberos;88/tcp;;;;# Kerberos
119;kerberos;88/udp;;;;# Kerberos
120;su-mit-tg;89/tcp;;;;# SU/MIT Telnet Gateway
121;su-mit-tg;89/udp;;;;# SU/MIT Telnet Gateway
122;dnsix;;90/tcp;;;;# DNSIX Security Attr. Token Map
123;dnsix;;90/udp;;;;# DNSIX Security Attr. Token Map
124;mit-dov;;91/tcp;;;;# MIT Dover Spooler
125;mit-dov;;91/udp;;;;# MIT Dover Spooler;
```

```
126;npp;;92/tcp;;;;# Network Printing Protocol
127;npp;;92/udp;;;;# Network Printing Protocol
128;dcp;;93/tcp;;;;# Device Control Protocol
129;dcp;;93/udp;;;;# Device Control Protocol
130;objcall;;94/tcp;;;;# Tivoli Object Dispatcher
131;objcall;;94/udp;;;;# Tivoli Object Dispatcher
132;supdup;;95/tcp
133;dixie;;96/tcp;;;;# DIXIE Protocol Specification
134;dixie;;96/udp;;;;# DIXIE Protocol Specification
135;swift-rvf;97/tcp;;;;# Swift Remote Vitural File Pro.
136;swift-rvf;97/udp;;;;# Swift Remote Vitural File Pro.
137;tacnews;;98/tcp;;;;# TAC News
138;tacnews;;98/udp;;;;# TAC News
139;metagram;99/tcp;;;;# Metagram Relay
140;metagram;99/udp;;;;# Metagram Relay
141;hostnames;101/tcp;;hostname;# usually from sri-nic
142;iso_tsap;102/tcp
143;x400;;103/tcp
144;x400-snd;104/tcp
145;csnet-ns;105/tcp
146;3com-tsmux;106/tcp;;;;# 3COM-TSMUX
147;3com-tsmux;106/udp;;;;# 3COM-TSMUX
148;rtelnet;;107/tcp;;;;# Remote Telnet Service
149;rtelnet;;107/udp;;;;# Remote Telnet Service
150;snagas;;108/tcp;;;;# SNA Gateway Access Server
151;snagas;;108/udp;;;;# SNA Gateway Access Server
152;pop;;109/tcp;;postoffice;# Post Office Protocol Ver. 2
153;pop3;;110/tcp;;postoffice3;# Post Office Protocol Ver. 3
154;sunrpc;;111/tcp
155;sunrpc;;111/udp
156;mcidas;;112/tcp;;;;# McIDAS Data Transmission Prot.
157;mcidas;;112/udp;;;;# McIDAS Data Transmission Prot.
158;auth;;113/tcp;;authentication
159;audionews;114/tcp;;;;# Audio News Multicast
160;audionews;114/udp;;;;# Audio News Multicast
161;sftp;;115/tcp
162;ansanotify;116/tcp;;;;# ANSA REX Notify
163;ansanotify;116/udp;;;;# ANSA REX Notify
164;uucp-path;117/tcp
165;sqlserv;;118/tcp;;;;# SQL Services
166;sqlserv;;118/udp;;;;# SQL Services
167;nntp;;119/tcp;;readnews untp;# USENET News Transfer Protocol
168;cfdptkt;;120/tcp;;;;# CFDPTKT
169;cfdptkt;;120/udp;;;;# CFDPTKT
170;erpc;;121/tcp;;;;# Encore Expedited Remote Pro.
171;erpc;;121/udp;;;;# Encore Expedited Remote Pro.
172;smakynet;122/tcp;;;;# SMAKYNET
173;smakynet;122/udp;;;;# SMAKYNET
174;ntp;;123/tcp
175;ansatrader;124/tcp;;;;# ANSA REX Trader
176;ansatrader;124/udp;;;;# ANSA REX Trader
177;locus-map;125/tcp;;;;# Locus PC-Interface Net Map Ser
178;locus-map;125/udp;;;;# Locus PC-Interface Net Map Ser
179;unitary;;126/tcp;;;;# Unisys Unitary Login
180;unitary;;126/udp;;;;# Unisys Unitary Login
181;locus-con;127/tcp;;;;# Locus PC-Interface Conn Server
182;locus-con;127/udp;;;;# Locus PC-Interface Conn Server
183;gss-xlicen;128/tcp;;;;# GSS X License Verification
184;gss-xlicen;128/udp;;;;# GSS X License Verification
```

```
185;pwdgen;;129/tcp;;;;# Password Generator Protocol
186;pwdgen;;129/udp;;;;# Password Generator Protocol
187;cisco-fna;130/tcp;;;;# cisco FNATIVE
188;cisco-fna;130/udp;;;;# cisco FNATIVE
189;cisco-tna;131/tcp;;;;# cisco TNATIVE
190;cisco-tna;131/udp;;;;# cisco TNATIVE
191;cisco-sys;132/tcp;;;;# cisco SYSMAINT
192;cisco-sys;132/udp;;;;# cisco SYSMAINT
193;statsrv;;133/tcp;;;;# Statistics Service
194;statsrv;;133/udp;;;;# Statistics Service
195;ingres-net;134/tcp;;;;# INGRES-NET Service
196;ingres-net;134/udp;;;;# INGRES-NET Service
197;loc-srv;;135/tcp;;;;# Location Service
198;loc-srv;;135/udp;;;;# Location Service
199;profile;;136/tcp;;;;# PROFILE Naming System
200;profile;;136/udp;;;;# PROFILE Naming System
201;netbios-ns;137/tcp;;;;# NETBIOS Name Service;
202;netbios-ns;137/udp;;;;# NETBIOS Name Service;
203;netbios-dgm;138/tcp;;;;# NETBIOS Datagram Service
204;netbios-dgm;138/udp;;;;# NETBIOS Datagram Service
205;netbios-ssn;139/tcp;;;;# NETBIOS Session Service
206;netbios-ssn;139/udp;;;;# NETBIOS Session Service
207;emfis-data;140/tcp;;;;# EMFIS Data Service
208;emfis-data;140/udp;;;;# EMFIS Data Service
209;emfis-cntl;141/tcp;;;;# EMFIS Control Service
210;emfis-cntl;141/udp;;;;# EMFIS Control Service
211;bl-idm;;142/tcp;;;;# Britton-Lee IDM
212;bl-idm;;142/udp;;;;# Britton-Lee IDM
213;imap2;;143/tcp;;;;# Interim Mail Access Pro. v2
214;imap2;;143/udp;;;;# Interim Mail Access Pro. v2
215;NeWS;;144/tcp
216;uaac;;145/tcp;;;;# UAAC Protocol
217;uaac;;145/udp;;;;# UAAC Protocol
218;iso-tp0;;146/tcp;;;;# ISO-IP0
219;iso-tp0;;146/udp;;;;# ISO-IP0
220;iso-ip;;147/tcp;;;;# ISO-IP
221;iso-ip;;147/udp;;;;# ISO-IP
222;cronus;;148/tcp;;;;# CRONUS-SUPPORT
223;cronus;;148/udp;;;;# CRONUS-SUPPORT
224;aed-512;;149/tcp;;;;# AED 512 Emulation Service
225;aed-512;;149/udp;;;;# AED 512 Emulation Service
226;sql-net;;150/tcp;;;;# SQL-NET
227;sql-net;;150/udp;;;;# SQL-NET
228;hems;;151/tcp;;;;# HEMS
229;hems;;151/udp;;;;# HEMS
230;bftp;;152/tcp;;;;# Background File Transfer Prg
231;bftp;;152/udp;;;;# Background File Transfer Prg
232;sgmp;;153/tcp;;;;# SGMP
233;sgmp;;153/udp;;;;# SGMP
234;netsc-prod;154/tcp;;;;# NETSC
235;netsc-prod;154/udp;;;;# NETSC
236;netsc-dev;155/tcp;;;;# NETSC
237;netsc-dev;155/udp;;;;# NETSC
238;sqlsrv;;156/tcp;;;;# SQL Service
239;sqlsrv;;156/udp;;;;# SQL Service
240;knet-cmp;157/tcp;;;;# KNET/VM Command/Message Pro.
241;knet-cmp;157/udp;;;;# KNET/VM Command/Message Pro.
242;pcmail-srv;158/tcp;;;;# PCMail Server
243;pcmail-srv;158/udp;;;;# PCMail Server
```

```
244;nss-routing;159/tcp;;;;# NSS-Routing
245;nss-routing;159/udp;;;;# NSS-Routing
246;sgmp-traps;160/tcp;;;;# SGMP-TRAPS
247;sgmp-traps;160/udp;;;;# SGMP-TRAPS
248;snmp;;161/tcp;;;;# snmp request port
249;snmp;;161/udp;;;;# snmp request port
250;snmp-trap;162/tcp;;;;# snmp monitor trap port
251;snmp-trap;162/udp;;;;# snmp monitor trap port
252;cmip-man;163/tcp;;;;# CMIP/TCP Manager
253;cmip-man;163/udp;;;;# CMIP/TCP Manager
254;cmip-agent;164/tcp;;;;# CMIP/TCP Agent
255;smip-agent;164/udp;;;;# CMIP/TCP Agent
256;xns-courier;165/tcp;;;;# Xerox
257;xns-courier;165/udp;;;;# Xerox
258;s-net;;166/tcp;;;;# Sirius Systems
259;s-net;;166/udp;;;;# Sirius Systems
260;namp;;167/tcp;;;;# NAMP
261;namp;;167/udp;;;;# NAMP
262;rsvd;;168/tcp;;;;# RSVD
263;rsvd;;168/udp;;;;# RSVD
264;send;;169/tcp;;;;# SEND
265;send;;169/udp;;;;# SEND
266;print-srv;170/tcp;;;;# Network PostScript
267;print-srv;170/udp;;;;# Network PostScript
268;multiplex;171/tcp;;;;# Network Innovations Multiplex
269;multiplex;171/udp;;;;# Network Innovations Multiplex
270;xyplex-mux;173/tcp;;;;# Xyplex
271;xyplex-mux;173/udp;;;;# Xyplex
272;mailq;;174/tcp;;;;# MAILQ
273;mailq;;174/udp;;;;# MAILQ
274;vmnet;;175/tcp;;;;# VMNET
275;vmnet;;175/udp;;;;# VMNET
276;genrad-mux;176/tcp;;;;# GENRAD-MUX
277;genrad-mux;176/udp;;;;# GENRAD-MUX
278;xdmcp;;177/tcp;;;;# X Display Manager Control Pro.
279;xdmcp;;177/udp;;;;# X Display Manager Control Pro.
280;nextstep;178/tcp;;;;# NextStep Window Server
281;NextStep;178/udp;;;;# NextStep Window Server
282;bgp;;179/tcp;;;;# Border Gateway Protocol
283;bgp;;179/udp;;;;# Border Gateway Protocol
284;ris;;180/tcp;;;;# Intergraph
285;ris;;180/udp;;;;# Intergraph
286;unify;;181/tcp;;;;# Unify
287;unify;;181/udp;;;;# Unify
288;audit;;182/tcp;;;;# Unisys Audit SITP
289;audit;;182/udp;;;;# Unisys Audit SITP
290;ocbinder;183/tcp;;;;# OCBinder
291;ocbinder;183/udp;;;;# OCBinder
292;ocserver;184/tcp;;;;# OCServer
293;ocserver;184/udp;;;;# OCServer
294;remote-kis;185/tcp;;;;# Remote-KIS
295;remote-kis;185/udp;;;;# Remote-KIS
296;kis;;186/tcp;;;;# KIS Protocol
297;kis;;186/udp;;;;# KIS Protocol
298;aci;;187/tcp;;;;# Application Comm. Interface
299;aci;;187/udp;;;;# Application Comm. Interface
300;mumps;;188/tcp;;;;# Plus Five's MUMPS
301;mumps;;188/udp;;;;# Plus Five's MUMPS
302;qft;;189/tcp;;;;# Queued File Transport
```

```
303;qft;;189/udp;;;;# Queued File Transport
304;gacp;;190/tcp;;;;# Gateway Access Control Pro.
305;cacp;;190/udp;;;;# Gateway Access Control Pro.
306;prospero;191/tcp;;;;# Prospero
307;prospero;191/udp;;;;# Prospero
308;osu-nms;;192/tcp;;;;# OSU Network Monitoring System
309;osu-nms;;192/udp;;;;# OSU Network Monitoring System
310;srmp;;193/tcp;;;;# Spider Remote Monitoring Pro.
311;srmp;;193/udp;;;;# Spider Remote Monitoring Pro.
312;irc;;194/tcp;;;;# Internet Relay Chat Protocol
313;irc;;194/udp;;;;# Internet Relay Chat Protocol
314;dn6-nlm-aud;195/tcp;;;;# DNSIX Net. Level Module Audit
315;dn6-nlm-aud;195/udp;;;;# DNSIX Net. Level Module Audit
316;dn6-smm-red;196/tcp;;;;# DNSIX Sess. Mgt Mod. Audit
317;dn6-smm-red;196/udp;;;;# DNSIX Sess. Mgt Mod. Audit
318;dls;;197/tcp;;;;# Directory Location Service
319;dls;;197/udp;;;;# Directory Location Service
320;dls-mon;;198/tcp;;;;# Directory Location Serv. Mon.
321;dls-mon;;198/udp;;;;# Directory Location Serv. Mon.
322;smux;;199/tcp;;;;# snmpd smux port
323;src;;200/udp;;;;# IBM System Resource controller
324;at-rtmp;;201/tcp;;;;# AppleTalk Routing Maint.
325;at-rtmp;;201/udp;;;;# AppleTalk Routing Maint.
326;at-nbp;;202/tcp;;;;# AppleTalk Name Binding
327;at-nbp;;202/udp;;;;# AppleTalk Name Binding
328;at-3;;203/tcp;;;;# AppleTalk Unused
329;at-3;;203/udp;;;;# AppleTalk Unused
330;at-echo;;204/tcp;;;;# AppleTalk Echo
331;at-echo;;204/udp;;;;# AppleTalk Echo
332;at-5;;205/tcp;;;;# AppleTalk Unused
333;at-5;;205/udp;;;;# AppleTalk Unused
334;at-zis;;206/tcp;;;;# AppleTalk Zone Information
335;at-zis;;206/udp;;;;# AppleTalk Zone Information
336;at-7;;207/tcp;;;;# AppleTalk Unused
337;at-7;;207/udp;;;;# AppleTalk Unused
338;at-8;;208/tcp;;;;# AppleTalk Unused
339;at-8;;208/udp;;;;# AppleTalk Unused
340;tam;;209/tcp;;;;# Trivial Auth. Mail Protocol
341;tam;;209/udp;;;;# Trivial Auth. Mail Protocol
342;z39.50;;210/tcp;;;;# ANSI Z39.50
343;z39.50;;210/udp;;;;# ANSI Z39.50
344;anet;;212/tcp;;;;# ATEXSSTR
345;anet;;212/udp;;;;# ATEXSSTR
346;ipx;;213/tcp;;;;# IPX
347;ipx;;213/udp;;;;# IPX
348;vmpwscs;;214/tcp;;;;# VM PWSCS
349;vmpwscs;;214/udp;;;;# VM PWSCS
350;softpc;;215/tcp;;;;# Insignia Solutions
351;softpc;;215/udp;;;;# Insignia Solutions
352;atls;;216/tcp;;;;# Access Tech. License Server
353;atls;;216/udp;;;;# Access Tech. License Server
354;dbase;;217/tcp;;;;# dBASE Unix
355;dbase;;217/udp;;;;# dBASE Unix
356;mpp;;218/tcp;;;;# Netix Message Posting Protocol
357;mpp;;218/udp;;;;# Netix Message Posting Protocol
358;uarps;;219/tcp;;;;# Unisys ARPs
359;uarps;;219/udp;;;;# Unisys ARPs
360;imap3;;220/tcp;;;;# Interactive Mail Acces Pro. v3
361;imap3;;220/udp;;;;# Interactive Mail Acces Pro. v3
```

```
362;fln-spx;;221/tcp;;;;# Berkeley rlogind w/ SPX Auth.
363;fln-spx;;221/udp;;;;# Berkeley rlogind w/ SPX Auth.
364;fsh-spx;;222/tcp;;;;# Berkeley rshd w/ SPX Auth.
365;fsh-spx;;222/udp;;;;# Berkeley rshd w/ SPX Auth.
366;cdc;;223/tcp;;;;# Certificate Dist. Center
367;cdc;;223/udp;;;;# Certificate Dist. Center
368;sur-meas;243/tcp;;;;# Survey Measurement
369;sur-meas;243/udp;;;;# Survey Measurement
370;dsp3270;;246/tcp;;;;# Display System Protocol
371;dsp3270;;246/udp;;;;# Display System Protocol
372;pawserv;;345/tcp;;;;# Perf Analysis Workbench
373;pawserv;;345/udp;;;;# Perf Analysis Workbench
374;zserv;;346/tcp;;;;# Zebra Server
375;zserv;;346/udp;;;;# Zebra Server
376;fatserv;;347/tcp;;;;# Fatmen Server
377;fatserv;;347/udp;;;;# Fatmen Server
378;clearcase;371/tcp;;;;# Clearcase
379;clearcase;371/udp;;;;# Clearcase
380;ulistserv;372/tcp;;;;# Unix Listserv
381;ulistserv;372/udp;;;;# Unix Listserv
382;legent-1;373/tcp;;;;# Legent Corporation
383;legent-1;373/udp;;;;# Legent Corporation
384;legent-2;374/tcp;;;;# Legent Corporation
385;legent-2;374/udp;;;;# Legent Corporation
386;#
387;# UNIX specific services
388;#
389;exec;;512/tcp
390;biff;;512/udp;;comsat
391;login;;513/tcp
392;who;;513/udp;;whod
393;shell;;514/tcp;;cmd;;# no passwords used
394;syslog;;514/udp
395;printer;;515/tcp;;spooler;;# line printer spooler
396;talk;;517/udp
397;ntalk;;518/udp
398;utime;;519/tcp;;;;# unixtime
399;utime;;519/udp;;;;# unixtime
400;efs;;520/tcp;;;;# for LucasFilm
401;route;;520/udp;;router routed
402;timed;;525/udp;;timeserver
403;tempo;;526/tcp;;newdate
404;courier;;530/tcp;;rpc
405;conference;531/tcp;;chat
406;netnews;;532/tcp;;readnews
407;netwall;;533/udp;;;;# -for emergency broadcasts
408;uucp;;540/tcp;;uucpd;;# uucp daemon
409;klogin;;543/tcp
410;kshell;;544/tcp;;krcmd
411;new-rwho;550/udp
412;dsf;;555/tcp
413;dsf;;555/udp
414;remotefs;556/tcp;;rfs_server rfs;# Brunhoff remote filesystem
415;rmonitor;560/udp
416;monitor;;561/udp
417;chshell;;562/tcp;;chcmd
418;chshell;;562/udp;;chcmd
419;9pfs;;564/tcp;;;;# plan 9 file service
420;9pfs;;564/udp;;;;# plan 9 file service
```

```
421;whoami;;565/tcp
422;whoami;;565/udp
423;meter;;570/tcp;;demon
424;meter;;570/udp;;demon
425;meter;;571/tcp;;udemon
426;meter;;571/udp;;udemon
427;ipcserver;600/tcp;;;;# Sun IPC Server
428;ipcserver;600/udp;;;;# Sun IPC Server
429;nqs;;607/tcp
430;nqs;;607/udp
431;mdqs;;666/tcp
432;mdqs;;666/udp
433;elcsd;;704/tcp;;;;# errlog copy/server daemon
434;elcsd;;704/udp;;;;# errlog copy/server daemon
435;netcp;;740/tcp;;;;# NETscout Control Protocol
436;netcp;;740/udp;;;;# NETscout Control Protocol
437;netgw;;741/tcp;;;;# netGW
438;netgw;;741/udp;;;;# netGW
439;netrcs;;742/tcp;;;;# Network Based Rev. Cont Sys.
440;netrcs;;742/udp;;;;# Network Based Rev. Cont Sys.
441;flexlm;;744/tcp;;;;# Flexible License Manager
442;flexlm;;744/udp;;;;# Flexible License Manager
443;fujitsu-dev;747/tcp;;;;# Fujitsu Device Control
444;fujitsu-dev;747/udp;;;;# Fujitsu Device Control
445;ris-cm;;748/tcp;;;;# Russell Info Sci Calendar Man.
446;ris-cm;;748/udp;;;;# Russell Info Sci Calendar Man.
447;kerberos-adm;749/tcp;;;;# kerberos administration
448;kerberos-adm;749/udp;;;;# kerberos administration
449;rfile;;750/tcp
450;loadav;;750/udp
451;pump;;751/tcp
452;pump;;751/udp
453;qrh;;752/tcp
454;qrh;;752/udp
455;rrh;;753/tcp
456;rrh;;753/udp
457;tell;;754/tcp
458;tell;;754/udp
459;nlogin;;758/tcp
460;nlogin;;758/udp
461;con;;759/tcp
462;con;;759/udp
463;ns;;760/tcp
464;ns;;760/udp
465;rxe;;761/tcp
466;rxe;;761/udp
467;quotad;;762/tcp
468;quotad;;762/udp
469;cycleserv;763/tcp
470;cycleserv;763/udp
471;omserv;;764/tcp
472;omserv;;764/udp
473;webster;;765/tcp
474;webster;;765/udp
475;phonebook;767/tcp;;phone
476;phonebook;767/udp;;phone
477;vid;;769/tcp
478;vid;;769/udp
479;cadlock;;770/tcp
```

```
480;cadlock;;770/udp
481;rtip;;771/tcp
482;rtip;;771/udp
483;cycleserv2;772/tcp
484;cycleserv2;772/udp
485;submit;;773/tcp
486;notify;;773/udp
487;rpasswd;;774/tcp
488;acmaint_dbd;774/udp
489;entomb;;775/tcp
490;acmaint_transd;775/udp
491;wpages;;776/tcp
492;wpages;;776/udp
493;wpgs;;780/tcp
494;wpgs;;780/udp
495;hp-collector;781/tcp;;;;# HP Perf. Data Collector
496;hp-collector;781/udp;;;;# HP Perf. Data Collector
497;hp-managed-node;782/tcp;;;;# HP Perf. Data Managed Node
498;hp-managed-node;782/udp;;;;# HP Perf. Data Managed Node
499;hp-alarm-mgr;783/tcp;;;;# HP Perf. Data Alarm Manager
500;hp-alarm-mgr;783/udp;;;;# HP Perf. Data Alarm Manager
501;mdbs_daemon;800/tcp
502;mdbs_daemon;800/udp
503;device;;801/tcp
504;device;;801/udp
505;xtreelic;996/tcp;;;;# XTREE License Server
506;xtreelic;996/udp;;;;# XTREE License Server
507;maitrd;;997/tcp
508;maitrd;;997/udp
509;busboy;;998/tcp
510;puparp;;998/udp
511;garcon;;999/tcp
512;applix;;999/udp;;;;# Applix ac
513;puprouter;999/tcp
514;puprouter;999/udp;
515;ock;;1000/udp
516;#
517;blackjack;1025/tcp;;;# Network BlackJack
518;blackjack;1025/udp;;;# Network BlackJack
519;instsrv;;1234/tcp;;;# Network Install Service
520;hermes;;1248/tcp
521;hermes;;1248/udp
522;bbn-mmc;;1347/tcp;;;# multi Media Conferencing
523;bbn-mmc;;1347/udp;;;# multi Media Conferencing
524;bbn-mmx;;1348/tcp;;;# multi Media Conferencing
525;bbn-mmx;;1348/udp;;;# multi Media Conferencing
526;sbook;;1349/tcp;;;# Registration Network Protocol
527;sbook;;1349/udp;;;# Registration Network Protocol
528;editbench;1350/tcp;;;# Registration Network Protocol
529;editbench;1350/udp;;;# Registration Network Protocol
530;equationbuilder;1351/tcp;;;# Digital Tool Works (MIT)
531;equationbuilder;1351/udp;;;# Digital Tool Works (MIT)
532;lotusnote;1352/tcp;;;# Lotus Note
533;lotusnote;1352/udp;;;# Lotus Note
534;ingreslock;1524/tcp;;;# ingres
535;orasrv;;1525/tcp;;;# oracle
536;orasrv;;1525/udp;;;# oracle
537;prospero-np;1525/tcp;;;# prospero non-privileged
538;prospero-np;1525/udp;;;# prospero non-privileged
```

```
539;tlisrv;;1527/tcp;;;# oracle
540;tlisrv;;1527/udp;;;# oracle
541;coauthor;1529/tcp;;;# oracle
542;coauthor;1529/udp;;;# oracle
543;issd;;1600/tcp
544;issd;;1600/udp
545;nkd;;1650/tcp
546;nkd;;1650/udp
547;callbook;2000/tcp
548;callbook;2000/udp
549;dc;;2001/tcp
550;wizard;;2001/udp;curry
551;globe;;2002/tcp
552;globe;;2002/udp
553;mailbox;;2004/tcp
554;emce;;2004/udp;;;# CCWS mm conf
555;berknet;;2005/tcp
556;oracle;;2005/udp
557;invokator;2006/tcp
558;raid-cc;;2006/udp
559;dectalk;;2007/tcp
560;raid-am;;2007/udp
561;conf;;2008/tcp
562;terminaldb;2008/udp
563;news;;2009/tcp
564;whosockami;2009/udp
565;search;;2010/tcp
566;pipe_server;2010/udp
567;raid-cc;;2011/tcp
568;servserv;2011/udp
569;ttyinfo;;2012/tcp
570;raid-ac;;2012/udp
571;raid-am;;2013/tcp
572;raid-cd;;2013/udp
573;troff;;2014/tcp
574;raid-sf;;2014/udp
575;cypress;;2015/tcp
576;raid-cs;;2015/udp
577;bootserver;2016/tcp
578;bootserver;2016/udp
579;cypress-stat;2017/tcp
580;bootclient;2017/udp
581;terminaldb;2018/tcp
582;rellpack;2018/udp
583;whosockami;2019/tcp
584;about;;2019/udp
585;xinupageserver;2020/tcp
586;xinupageserver;2020/udp
587;servexec;2021/tcp
588;xinuexpansion1;2021/udp
589;down;;2022/tcp
590;xinuexpansion2;2022/udp
591;xinuexpansion3;2023/tcp
592;xinuexpansion3;2023/udp
593;xinuexpansion4;2024/tcp
594;xinuexpansion4;2024/udp
595;ellpack;;2025/tcp
596;xribs;;2025/udp
597;scrabble;2026/tcp
```

```
598;scrabble;2026/udp
599;shadowserver;2027/tcp
600;shadowserver;2027/udp
601;submitserver;2028/tcp
602;submitserver;2028/udp
603;device2;;2030/tcp
604;device2;;2030/udp
605;blackboard;2032/tcp
606;blackboard;2032/udp
607;glogger;;2033/tcp
608;glogger;;2033/udp
609;scoremgr;2034/tcp
610;scoremgr;2034/udp
611;imsldoc;;2035/tcp
612;imsldoc;;2035/udp
613;objectmanager;2038/tcp
614;objectmanager;2038/udp
615;lam;;2040/tcp
616;lam;;2040/udp
617;interbase;2041/tcp
618;interbase;2041/udp
619;isis;;2042/tcp
620;isis;;2042/udp
621;isis-bcast;2043/tcp
622;isis-bcast;2043/udp
623;rimsl;;2044/tcp
624;rimsl;;2044/udp
625;cdfunc;;2045/tcp
626;cdfunc;;2045/udp
627;sdfunc;;2046/tcp
628;sdfunc;;2046/udp
629;shilp;;2049/tcp
630;shilp;;2049/udp
631;writesrv;2401/tcp;;;# Temporary Port Number
632;www-dev;;2784/tcp;;;# World Wide Web - development
633;www-dev;;2784/udp;;;# World Wide Web - development
634;NSWS;;3049/tcp
635;NSWS;;3049/udp
636;rfa;;4672/tcp;;;# Remote File Access Server
637;rfa;;4672/udp;;;# Remote File Access Server
638;commplex-main;5000/tcp
639;commplex-main;5000/udp
640;commplex-link;5001/tcp
641;commplex-link;5001/udp
642;rfe;;5002/tcp;;;# Radio Free Ethernet
643;rfe;;5002/udp;;;# Radio Free Ethernet
644;rmonitor_secure;5145/tcp
645;rmonitor_secure;5145/udp
646;padl2sim;5236/tcp
647;padl2sim;5236/udp
648;sub-process;6111/tcp;;;# HP SoftBench Sub-Process Cntl.
649;sub-process;6111/udp;;;# HP SoftBench Sub-Process Cntl.
650;xdsxdm;;6558/udp
651;xdsxdm;;6558/tcp
652;afs3-fileserver;7000/tcp;;;# File Server Itself
653;afs3-fileserver;7000/udp;;;# File Server Itself
654;afs3-callback;7001/tcp;;;# Callbacks to Cache Managers
655;afs3-callback;7001/udp;;;# Callbacks to Cache Managers
656;afs3-prserver;7002/tcp;;;# Users & Groups Database
```

```
657;afs3-prserver;7002/udp;;;# Users & Groups Database
658;afs3-vlserver;7003/tcp;;;# Volume Location Database
659;afs3-vlserver;7003/udp;;;# Volume Location Database
660;afs3-kaserver;7004/tcp;;;# AFS/Kerberos Auth. Service
661;afs3-kaserver;7004/udp;;;# AFS/Kerberos Auth. Service
662;afs3-volser;7005/tcp;;;# Volume Managment Server
663;afs3-volser;7005/udp;;;# Volume Managment Server
664;afs3-errors;7006/tcp;;;# Error Interpretation Service
665;afs3-errors;7006/udp;;;# Error Interpretation Service
666;afs3-bos;7007/tcp;;;# Basic Overseer Process
667;afs3-bos;7007/udp;;;# Basic Overseer Process
668;afs3-update;7008/tcp;;;# Server-To-Server Updater
669;afs3-update;7008/udp;;;# Server-To-Server Updater
670;afs3-rmtsys;7009/tcp;;;# Remote Cache Manager Service
671;afs3-rmtsys;7009/udp;;;# Remote Cache Manager Service
672;graPHIGS;8000/tcp;; ;# graPHIGS Remote Nucleus
673;x_st_mgrd;9000/tcp;; ;# IBM X Terminal
674;man;;9535/tcp
675;man;;9535/udp
676;isode-dua;17007/tcp
677;isode-dua;17007/udp
678;dtspc;;6112/tcp;
```

B.3 ONC/RPC Program Numbers in /etc/rpc

```
 1;# @(#)44;1.3  src/nfs/etc/rpc, cmdnfs, nfs411, GOLD410 4/22/94 13:20:59
 2;#
 3;# COMPONENT_NAME: (CMDNFS) Network File System Commands
 4;#
 5;# FUNCTIONS:
 6;#
 7;# ORIGINS: 24
 8;#
 9;# Copyright (c) 1988 Sun Microsystems, Inc.
10;#
11;portmapper;100000;portmap sunrpc
12;rstatd;;100001;rstat rup perfmeter
13;rusersd;;100002;rusers
14;nfs;;100003;nfsprog
15;ypserv;;100004;ypprog
16;mountd;;100005;mount showmount
17;ypbind;;100007
18;walld;;100008;rwall shutdown
19;yppasswdd;100009;yppasswd
20;etherstatd;100010;etherstat
21;rquotad;;100011;rquotaprog quota rquota
22;sprayd;;100012;spray
23;3270_mapper;100013
24;rje_mapper;100014
25;selection_svc;100015;selnsvc
26;database_svc;100016
27;rexd;;100017;rex
28;alis;;100018
29;sched;;100019
30;llockmgr;100020
31;nlockmgr;100021
32;x25.inr;;100022
33;statmon;;100023
```

```
34;status;;100024
35;bootparam;100026
36;ypupdated;100028;ypupdate
37;keyserv;;100029;keyserver
38;sunlink_mapper;100033
39;tfsd;;100037
40;nsed;;100038
41;nsemntd;;100039
42;showfhd;;100043;showfh
43;cmsd;;100068;dtcalendar
44;ypxfrd;;100069;ypxfr
45;ttdbserverd;100083;tooltalk
46;pcnfsd;;150001
```

Appendix C. Important Header Files

C.1 /usr/include/netinet/tcp.h

```
/* @(#)54;1.8.1.3  src/bos/kernext/inet/tcp.h, sockinc, bos411, 9428A410j 12/20/93 11:55:55 */
/*
 *   COMPONENT_NAME: SYSXINET
 *
 *   FUNCTIONS:
 *
 *   ORIGINS: 26,27,85,90
 *
 *
 *   (C) COPYRIGHT International Business Machines Corp. 1988,1993
 *   All Rights Reserved
 *   Licensed Materials - Property of IBM
 *   US Government Users Restricted Rights - Use, duplication or
 *   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */

/*
 * (c) Copyright 1990, 1991, 1992, 1993 OPEN SOFTWARE FOUNDATION, INC.
 * ALL RIGHTS RESERVED
 */
/*
 * OSF/1 1.2
 */
/*
 * Copyright (C) 1988,1989 Encore Computer Corporation.  All Rights Reserved
 *
 * Property of Encore Computer Corporation.
 * This software is made available solely pursuant to the terms of
 * a software license agreement which governs its use. Unauthorized
 * duplication, distribution or sale are strictly prohibited.
 *
 */
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted provided
 * that: (1) source distributions retain this entire copyright notice and
 * comment, and (2) distributions including binaries display the following
 * acknowledgement:  This product includes software developed by the
 * University of California, Berkeley and its contributors'' in the
 * documentation or other materials provided with the distribution and in
 * all advertising materials mentioning features or use of this software.
 * Neither the name of the University nor the names of its contributors may
 * be used to endorse or promote products derived from this software without
 * specific prior written permission.
 * THIS SOFTWARE IS PROVIDED AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 *;Base:;tcp.h;7.5 (Berkeley) 6/29/88
 *;Merged:;tcp.h;7.7 (Berkeley) 6/28/90
 */
```

```

#ifndef;_NETINET_TCP_H
#define;_NETINET_TCP_H

#include <netinet/ip.h>

typedef;u_long;tcp_seq;

/*
 * TCP header.
 * Per RFC 793, September, 1981.
 */
struct tcphdr {
;u_short;th_sport;;;/* source port */
;u_short;th_dport;;;/* destination port */
;tcp_seq;th_seq;;;;/* sequence number */
;tcp_seq;th_ack;;;;/* acknowledgement number */
;struct;ip_firstfour ip_ff;;/* see <netinet/ip.h> */
#define;th_off;;ip_v;;/* offset */
#define;th_x2;;ip_hl;;/* unused */
#define;th_xoff;;ip_vhl;;/* offset+unused */
#define;th_flags;ip_tos;;/* flags */
#define;th_win;;ip_ff.ip_fwin;/* window (unsigned) */
#define;TH_FIN;0x01
#define;TH_SYN;0x02
#define;TH_RST;0x04
#define;TH_PUSH;0x08
#define;TH_ACK;0x10
#define;TH_URG;0x20
;u_short;th_sum;;;;/* checksum */
;u_short;th_urp;;;;/* urgent pointer */
};

#define;TCPOPT_EOL;;0
#define;TCPOPT_NOP;;1
#define;TCPOPT_MAXSEG;;2
#define TCPOPT_WINDOWSCALE;3;/* RFC 1323 window scale option */
#define TCPOPT_TIMESTAMP;8;/* RFC 1323 timestamp option */

#define TCP_MAXWINDOWSCALE;14;/* RFC 1323 max shift factor */

/*
 * RFC 1323 - this define is used for fast timstamp option parsing...
 */
#define TCP_FASTNAME;;;0x0101080A
#define TCP_TIMESTAMP_OPTLEN;;12

/*
 * RFC 1323 - Used by PAWS algorithm.
 */
#define TCP_24DAYS_WORTH_OF_SLOWTICKS;(24 * 24 * 60 * 60 * PR_SLOWHZ)


/*
 * Default maximum segment size for TCP.
 * With an IP MSS of 576, this is 536,
 * but 512 is probably more convenient.
 * This should be defined as MIN(512, IP_MSS - sizeof (struct tcpiphdr)).
 */
```

```
#define;TCP_MSS;512

#define;TCP_MAXWIN;65535;;/* largest value for window */

/*
 * User-settable options (used with setsockopt).
 */
#define;TCP_NODELAY;0x01;/* don′t delay send to coalesce packets */
#define;TCP_MAXSEG;0x02;/* set maximum segment size */
#define TCP_RFC1323;0x04;/* Use RFC 1323 algorithms/options */
#endif
```

C.2 /usr/include/netinet/tcp_timer.h

```
/* @(#)38;1.11  src/bos/kernext/inet/tcp_timer.h, sockinc, bos411, 9428A410j 9/15/93 16:48:43 */
/*
 *   COMPONENT_NAME: SYSXINET
 *
 *   FUNCTIONS: TCPT_RANGESET
 *;;
 *
 *   ORIGINS: 26,27,85
 *
 *
 *   (C) COPYRIGHT International Business Machines Corp. 1988,1993
 *   All Rights Reserved
 *   Licensed Materials - Property of IBM
 *   US Government Users Restricted Rights - Use, duplication or
 *   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */

/*
 *
 * (c) Copyright 1991, OPEN SOFTWARE FOUNDATION, INC.
 * ALL RIGHTS RESERVED
 *
 */
/*
 * OSF/1 1.1 Snapshot 2
 */
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted provided
 * that: (1) source distributions retain this entire copyright notice and
 * comment, and (2) distributions including binaries display the following
 * acknowledgement:  This product includes software developed by the
 * University of California, Berkeley and its contributors′′ in the
 * documentation or other materials provided with the distribution and in
 * all advertising materials mentioning features or use of this software.
 * Neither the name of the University nor the names of its contributors may
 * be used to endorse or promote products derived from this software without
 * specific prior written permission.
 * THIS SOFTWARE IS PROVIDED AS IS′′ AND WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 *;Base::;tcp_timer.h;7.6 (Berkeley) 6/29/88
 *;Merged::;tcp_timer.h;7.8 (Berkeley) 6/28/90
 */

/*
 * Definitions of the TCP timers.  These timers are counted
 * down PR_SLOWHZ times a second.
 */
#define;TCPT_NTIMERS;4

#define;TCPT_REXMT;0;;/* retransmit */
#define;TCPT_PERSIST;1;;/* retransmit persistance */
#define;TCPT_KEEP;2;;/* keep alive */
#define;TCPT_2MSL;3;;/* 2*msl quiet time timer */

/*
 * The TCPT_REXMT timer is used to force retransmissions.
 * The TCP has the TCPT_REXMT timer set whenever segments
 * have been sent for which ACKs are expected but not yet
 * received.  If an ACK is received which advances tp->snd_una,
 * then the retransmit timer is cleared (if there are no more
 * outstanding segments) or reset to the base value (if there
 * are more ACKs expected).  Whenever the retransmit timer goes off,
 * we retransmit one unacknowledged segment, and do a backoff
 * on the retransmit timer.
 *
 * The TCPT_PERSIST timer is used to keep window size information
 * flowing even if the window goes shut.  If all previous transmissions
 * have been acknowledged (so that there are no retransmissions in progress),
 * and the window is too small to bother sending anything, then we start
 * the TCPT_PERSIST timer.  When it expires, if the window is nonzero,
 * we go to transmit state.  Otherwise, at intervals send a single byte
 * into the peer's window to force him to update our window information.
 * We do this at most as often as TCPT_PERSMIN time intervals,
 * but no more frequently than the current estimate of round-trip
 * packet time.  The TCPT_PERSIST timer is cleared whenever we receive
 * a window update from the peer.
 *
 * The TCPT_KEEP timer is used to keep connections alive.  If an
 * connection is idle (no segments received) for TCPTV_KEEP_INIT amount of time,
 * but not yet established, then we drop the connection.  Once the connection
 * is established, if the connection is idle for TCPTV_KEEP_IDLE time
 * (and keepalives have been enabled on the socket), we begin to probe
 * the connection.  We force the peer to send us a segment by sending:
 *;<SEQ=SND.UNA-1><ACK=RCV.NXT><CTL=ACK>
 * This segment is (deliberately) outside the window, and should elicit
 * an ack segment in response from the peer.  If, despite the TCPT_KEEP
 * initiated segments we cannot elicit a response from a peer in TCPT_MAXIDLE
 * amount of time probing, then we drop the connection.
 */

#define;TCP_TTL;;60;;/* default time to live for TCP segs */
/*
 * Time constants.
 */
#define;TCPTV_MSL;( 30*PR_SLOWHZ);;/* max seg lifetime (hah!) */
#define;TCPTV_SRTTBASE;0;;;/* base roundtrip time;
;;;;;;   if 0, no idea yet */
#define;TCPTV_SRTTDFLT;(  3*PR_SLOWHZ);;/* assumed RTT if no info */
```

```

#define;TCPTV_PERSMIN;(  5*PR_SLOWHZ);;/* retransmit persistance */
#define;TCPTV_PERSMAX;( 60*PR_SLOWHZ);;/* maximum persist interval */

#define;TCPTV_KEEP_INIT;( 75*PR_SLOWHZ);;/* initial connect keep alive */
#define;TCPTV_KEEP_IDLE;(120*60*PR_SLOWHZ);/* dflt time before probing */
#define;TCPTV_KEEPINTVL;( 75*PR_SLOWHZ);;/* default probe interval */
#define;TCPTV_KEEPCNT;8;;;/* max probes before drop */

#define;TCPTV_MIN;(  1*PR_SLOWHZ);;/* minimum allowable value */
#define;TCPTV_REXMTMAX;( 64*PR_SLOWHZ);;/* max allowable REXMT value */

#define;TCP_LINGERTIME;120;;;/* linger at most 2 minutes */

#ifndef;CONST
#define;CONST
#endif

#ifdef;TCPTIMERS
CONST;char *tcptimers[] =
    { "REXMT", "PERSIST", "KEEP", "2MSL" };
#endif

/*
 * Force a time value to be in a certain range.
 */
#define;TCPT_RANGESET(tv, value, tvmin, tvmax) { \
;(tv) = (value); \
;if ((tv) < (tvmin)) \
;;(tv) = (tvmin); \
;else if ((tv) > (tvmax)) \
;;(tv) = (tvmax); \
}

#ifdef _KERNEL
extern int tcp_keepidle;;;/* time before keepalive probes begin */
extern int tcp_keepintvl;;;/* time between keepalive probes */
extern int tcp_maxidle;;;;/* time to drop after starting probes */
extern int tcp_ttl;;;;/* time to live for TCP segs */
extern int tcp_rtolow;;;;/* TCP RTO backoff low watermark */
extern int tcp_rtohigh;;;;/* TCP RTO backoff high watermark */
extern int tcp_rtolimit;;;/* TCP RTO backoff exponential mark */
extern int tcp_rtolength;;;/* TCP RTO backoff total# elements */
extern int tcp_rtoshift;;;/* TCP RTO backoff delta shift; the */
;;;;;/*  number of bit shitfs from */
;;;;;/*  tcp_rtolow to tcp_rtohigh inclusive */
#define TCP_BACKOFF(x) \
                ( (((x)+1) >= (tcp_rtolimit)) ? (tcp_rtohigh) : \
;;;(tcp_rtolow) << (((tcp_rtoshift)*(x))/(tcp_rtolimit)) )
#endif
```

C.3 /usr/include/netinet/tcp_var.h

```
/* @(#)55;1.14.2.5  src/bos/kernext/inet/tcp_var.h, sockinc, bos411, 9428A410j 3/15/94 17:00:24 */
/*
 *   COMPONENT_NAME: SYSXINET
 *
 *   FUNCTIONS: REASS_MBUF
 *;;TCPMISC_LOCK
 *;;TCPMISC_LOCKINIT
 *;;TCPMISC_UNLOCK
 *;;TCP_REXMTVAL
 *;;intotcpcb
 *;;sototcpcb
 *;;
 *
 *   ORIGINS: 26,27,85,90
 *
 *
 *   (C) COPYRIGHT International Business Machines Corp. 1988,1993
 *   All Rights Reserved
 *   Licensed Materials - Property of IBM
 *   US Government Users Restricted Rights - Use, duplication or
 *   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */

/*
 * (c) Copyright 1990, 1991, 1992, 1993 OPEN SOFTWARE FOUNDATION, INC.
 * ALL RIGHTS RESERVED
 */
/*
 * OSF/1 1.2
 */
/*
 * Copyright (C) 1988,1989 Encore Computer Corporation.  All Rights Reserved
 *
 * Property of Encore Computer Corporation.
 * This software is made available solely pursuant to the terms of
 * a software license agreement which governs its use. Unauthorized
 * duplication, distribution or sale are strictly prohibited.
 *
 */
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted provided
 * that: (1) source distributions retain this entire copyright notice and
 * comment, and (2) distributions including binaries display the following
 * acknowledgement:  This product includes software developed by the
 * University of California, Berkeley and its contributors'' in the
 * documentation or other materials provided with the distribution and in
 * all advertising materials mentioning features or use of this software.
 * Neither the name of the University nor the names of its contributors may
 * be used to endorse or promote products derived from this software without
 * specific prior written permission.
 * THIS SOFTWARE IS PROVIDED AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 *;Base:;tcp_var.h;7.8 (Berkeley) 6/29/88
 *;Merged:;tcp_var.h;7.10 (Berkeley) 6/28/90
 */

/*
 * Kernel variables for tcp.
 */
```

```

/*
 * Tcp control block, one per tcp; fields:
 */
struct tcpcb {
;struct;tcpiphdr *seg_next;;/* sequencing queue */
;struct;tcpiphdr *seg_prev;
;short;t_state;;;/* state of this connection */
;short;t_softerror;;;/* possible error not yet reported */
;short;t_timer[TCPT_NTIMERS];;/* tcp timers */
;short;t_rxtshift;;;/* log(2) of rexmt exp. backoff */
;short;t_rxtcur;;;/* current retransmit value */
;short;t_dupacks;;;/* consecutive dup acks recd */
;u_short;t_maxseg;;;/* maximum segment size */
;char;t_force;;;/* 1 if forcing out a byte */
;u_short;t_flags;
#define;TF_ACKNOW;0x0001;;/* ack peer immediately */
#define;TF_DELACK;0x0002;;/* ack, but try to delay it */
#define;TF_NODELAY;0x0004;;/* don′t delay packets to coalesce */
#define;TF_NOOPT;0x0008;;/* don′t use tcp options */
#define;TF_SENTFIN;0x0010;;/* have sent FIN */
#define TF_RFC1323;0x0020;;/* Use RFC1323 TCP options */
#define TF_SENT_WS;0x0040;;/* TCP has sent a windowscale option */
#define TF_RCVD_WS;0x0080;;/* TCP has rcvd a windowscale option */
#define TF_SENT_TS;0x0100;;/* TCP has sent a timestamp option */
#define TF_RCVD_TS;0x0200;;/* TCP has rcvd a timestamp option */

/* out-of-band data */
;char;t_oobflags;;;/* have some */
;char;t_iobc;;;;/* input character */
#define;TCPOOB_HAVEDATA;0x01
#define;TCPOOB_HADDATA;0x02

;struct;tcpiphdr t_template;;/* skeletal packet for transmit
;;;;; * (used to be mbuf)
;;;;; */
;struct;inpcb *t_inpcb;;;/* back pointer to internet pcb */
;tcp_seq;t_timestamp;;;/* used by slowtimo */
/*
 * The following fields are used as in the protocol specification.
 * See RFC783, Dec. 1981, page 21.
 */
/* send sequence variables */
;tcp_seq;snd_una;;;/* send unacknowledged */
;tcp_seq;snd_nxt;;;/* send next */
;tcp_seq;snd_up;;;;/* send urgent pointer */
;tcp_seq;snd_wl1;;;/* window update seg seq number */
;tcp_seq;snd_wl2;;;/* window update seg ack number */
;tcp_seq;iss;;;;/* initial send sequence number */
;u_long;snd_wnd;;;/* send window */
/* receive sequence variables */
;u_long;rcv_wnd;;;/* receive window */
;tcp_seq;rcv_nxt;;;/* receive next */
;tcp_seq;rcv_up;;;;/* receive urgent pointer */
;tcp_seq;irs;;;;/* initial receive sequence number */
/* RFC 1323 - variables */
;short;snd_wnd_scale;;;/* snd window scale */
;short;rcv_wnd_scale;;;/* rcv window scale */
;short;req_scale_sent;
;short;req_scale_rcvd;
;tcp_seq;last_ack_sent;;;/* seqno of last ACK sent (RTTM) */
;u_long;timestamp_recent;;/* most recent timestamp rcved (RTTM) */
;int;timestamp_age;;;/* age of timestamp_recent */
/*
 * Additional variables for this implementation.
 */
/* receive variables */
;tcp_seq;rcv_adv;;;/* advertised window */
/* retransmit variables */
;tcp_seq;snd_max;;;/* highest sequence number sent;
;;;;; * used to recognize retransmits
;;;;; */
/* congestion control (for slow start, source quench, retransmit after loss) */
;u_long;snd_cwnd;;;/* congestion-controlled window */
;u_long snd_ssthresh;;;/* snd_cwnd size threshhold for
;;;;; * slow start exponential to
;;;;; * linear switch
;;;;; */
/*
 * transmit timing stuff.  See below for scale of srtt and rttvar.
 * "Variance" is actually smoothed difference.
 */
;short;t_idle;;;;/* inactivity time */
;short;t_rtt;;;;/* round trip time */
;tcp_seq;t_rtseq;;;/* sequence number being timed */
;short;t_srtt;;;;/* smoothed round-trip time */
;short;t_rttvar;;;/* variance in round-trip time */
;u_short;t_rttmin;;;/* minimum rtt allowed */
;u_long ;max_rcvd;;;/* most peer has sent into window */
;u_long;max_sndwnd;;;/* largest window peer has offered */
};

#define;intotcpcb(ip);((struct tcpcb *)(ip)->inp_ppcb)
#define;sototcpcb(so);(intotcpcb(sotoinpcb(so)))

/*
 * The smoothed round-trip time and estimated variance
 * are stored as fixed point numbers scaled by the values below.
 * For convenience, these scales are also used in smoothing the average
 * (smoothed = (1/scale)sample + ((scale-1)/scale)smoothed).
 * With these scales, srtt has 3 bits to the right of the binary point,
 * and thus an "ALPHA" of 0.875.  rttvar has 2 bits to the right of the
 * binary point, and is smoothed with an ALPHA of 0.75.
 */
#define;TCP_RTT_SCALE;;8;/* multiplier for srtt; 3 bits frac. */
#define;TCP_RTT_SHIFT;;3;/* shift for srtt; 3 bits frac. */
#define;TCP_RTTVAR_SCALE;4;/* multiplier for rttvar; 2 bits */
#define;TCP_RTTVAR_SHIFT;2;/* multiplier for rttvar; 2 bits */

/*
 * The initial retransmission should happen at rtt + 4 * rttvar.
 * Because of the way we do the smoothing, srtt and rttvar
 * will each average +1/2 tick of bias.  When we compute
 * the retransmit timer, we want 1/2 tick of rounding and
 * 1 extra tick because of +-1/2 tick uncertainty in the
 * firing of the timer.  The bias will give us exactly the
 * 1.5 tick we need.  But, because the bias is
 * statistical, we have to test that we don't drop below
 * the minimum feasible timer (which is 2 ticks).
 * This macro assumes that the value of TCP_RTTVAR_SCALE
 * is the same as the multiplier for rttvar.
 */
#define;TCP_REXMTVAL(tp) \
;(((tp)->t_srtt >> TCP_RTT_SHIFT) + (tp)->t_rttvar)

/* XXX
 * We want to avoid doing m_pullup on incoming packets but that
 * means avoiding dtom on the tcp reassembly code.  That in turn means
 * keeping an mbuf pointer in the reassembly queue (since we might
 * have a cluster).  As a quick hack, the source & destination
 * port numbers (which are no longer needed once we've located the
 * tcpcb) are overlayed with an mbuf pointer.
 */
#define REASS_MBUF(ti) (*(struct mbuf **)&((ti)->ti_t))

/*
 * RFC 1323 - In the spirit of Header Prediction, we use this struct to
 * ;;check for the TS option.  If we match, then we avoid
*;;tcp_dooption() parsing...
 */
struct tcp_ts {
;u_long;  ts_name;
;u_long;  ts_val;
;u_long    ts_ecr;
};

/*
 * TCP statistics.
 * Many of these should be kept per connection,
 * but that's inconvenient at the moment.
 */
struct;tcpstat {
;u_long;tcps_connattempt;;/* connections initiated */
;u_long;tcps_accepts;;;/* connections accepted */
;u_long;tcps_connects;;;/* connections established */
;u_long;tcps_drops;;;/* connections dropped */
;u_long;tcps_conndrops;;;/* embryonic connections dropped */
;u_long;tcps_closed;;;/* conn. closed (includes drops) */
;u_long;tcps_segstimed;;;/* segs where we tried to get rtt */
;u_long;tcps_rttupdated;;/* times we succeeded */
;u_long;tcps_delack;;;/* delayed acks sent */
;u_long;tcps_timeoutdrop;;/* conn. dropped in rxmt timeout */
;u_long;tcps_rexmttimeo;;/* retransmit timeouts */
;u_long;tcps_persisttimeo;;/* persist timeouts */
;u_long;tcps_keeptimeo;;;/* keepalive timeouts */
;u_long;tcps_keepprobe;;;/* keepalive probes sent */
;u_long;tcps_keepdrops;;;/* connections dropped in keepalive */

;u_long;tcps_sndtotal;;;/* total packets sent */
;u_long;tcps_sndpack;;;/* data packets sent */
;u_long;tcps_sndbyte;;;/* data bytes sent */
;u_long;tcps_sndrexmitpack;;/* data packets retransmitted */
;u_long;tcps_sndrexmitbyte;;/* data bytes retransmitted */
;u_long;tcps_sndacks;;;/* ack-only packets sent */
;u_long;tcps_sndprobe;;;/* window probes sent */
;u_long;tcps_sndurg;;;/* packets sent with URG only */
;u_long;tcps_sndwinup;;;/* window update-only packets sent */
;u_long;tcps_sndctrl;;;/* control (SYN|FIN|RST) packets sent */

;u_long;tcps_rcvtotal;;;/* total packets received */
;u_long;tcps_rcvpack;;;/* packets received in sequence */
;u_long;tcps_rcvbyte;;;/* bytes received in sequence */
;u_long;tcps_rcvbadsum;;;/* packets received with ccksum errs */
;u_long;tcps_rcvbadoff;;;/* packets received with bad offset */
;u_long;tcps_rcvshort;;;/* packets received too short */
;u_long;tcps_rcvduppack;;/* duplicate-only packets received */
;u_long;tcps_rcvdupbyte;;/* duplicate-only bytes received */
;u_long;tcps_rcvpartduppack;;/* packets with some duplicate data */
;u_long;tcps_rcvpartdupbyte;;/* dup. bytes in part-dup. packets */
;u_long;tcps_rcvoopack;;;/* out-of-order packets received */
;u_long;tcps_rcvoobyte;;;/* out-of-order bytes received */
;u_long;tcps_rcvpackafterwin;;/* packets with data after window */
;u_long;tcps_rcvbyteafterwin;;/* bytes rcvd after window */
;u_long;tcps_rcvafterclose;;/* packets rcvd after "close" */
;u_long;tcps_rcvwinprobe;;/* rcvd window probe packets */
;u_long;tcps_rcvdupack;;;/* rcvd duplicate acks */
;u_long;tcps_rcvacktoomuch;;/* rcvd acks for unsent data */
;u_long;tcps_rcvackpack;;/* rcvd ack packets */
;u_long;tcps_rcvackbyte;;/* bytes acked by rcvd acks */
;u_long;tcps_rcvwinupd;;;/* rcvd window update packets */
;u_long ;tcps_pawsdrop;;;/* (RFC 1323) pkts dropped because
;;;;;;of PAWS */
#if;defined(_KERNEL) && LOCK_NETSTATS
;simple_lock_data_t tcps_lock;;/* statistics lock */
#endif
};

#ifdef _KERNEL
#if;NETSYNC_LOCK
extern;simple_lock_data_t;misc_tcp_lock;
#define TCPMISC_LOCKINIT();{;;;;;\
;lock_alloc(&misc_tcp_lock, LOCK_ALLOC_PIN, TCPMISC_LOCK_FAMILY, -1);\
;simple_lock_init(&misc_tcp_lock);;;;;\
}
#define TCPMISC_LOCK_DECL();int;_tcpml;
#define TCPMISC_LOCK();;_tcpml = disable_lock(PL_IMP, &misc_tcp_lock)
#define TCPMISC_UNLOCK();unlock_enable(_tcpml, &misc_tcp_lock)
#else;/* !NETSYNC_LOCK */
#define TCPMISC_LOCKINIT()
#define TCPMISC_LOCK()
#define TCPMISC_UNLOCK()
#endif

extern;int tcp_compat_42;
extern;struct;inpcb tcb;;;/* head of queue of active tcpcb's */
extern;struct;tcpstat tcpstat;;/* tcp statistics */
#endif
```

C.4 /usr/include/sys/protosw.h

```
/* @(#)97;1.12  src/bos/kernel/sys/protosw.h, sockinc, bos411, 9433A411a 8/12/94 10:09:06 */
/*
 *   COMPONENT_NAME: SOCKINC
 *
 *   FUNCTIONS: PRC_IS_REDIRECT
 *;;
 *
 *   ORIGINS: 26,27,85
 *
 *
 *   (C) COPYRIGHT International Business Machines Corp. 1988,1993
 *   All Rights Reserved
 *   Licensed Materials - Property of IBM
 *   US Government Users Restricted Rights - Use, duplication or
 *   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */
/*
 * (c) Copyright 1990, 1991, 1992, 1993 OPEN SOFTWARE FOUNDATION, INC.
 * ALL RIGHTS RESERVED
 */
/*
 * OSF/1 1.2
 */
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted provided
 * that: (1) source distributions retain this entire copyright notice and
 * comment, and (2) distributions including binaries display the following
 * acknowledgement:  This product includes software developed by the
 * University of California, Berkeley and its contributors'' in the
 * documentation or other materials provided with the distribution and in
 * all advertising materials mentioning features or use of this software.
 * Neither the name of the University nor the names of its contributors may
 * be used to endorse or promote products derived from this software without
 * specific prior written permission.
 * THIS SOFTWARE IS PROVIDED AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 *;Base:;protosw.h;7.4 (Berkeley) 9/4/89
 *;Merged: protosw.h;7.6 (Berkeley) 6/28/90
 */

#ifndef;_SYS_PROTOSW_H_
#define;_SYS_PROTOSW_H_

/*
 * Protocol switch table.
 *
 * Each protocol has a handle initializing one of these structures,
 * which is used for protocol-protocol and system-protocol communication.
 *
 * A protocol is called through the pr_init entry before any other.
 * Thereafter it is called every 200ms through the pr_fasttimo entry and
 * every 500ms through the pr_slowtimo for timer based actions.
 * The system will call the pr_drain entry if it is low on space and
 * this should throw away any non-critical data.
 *
 * Protocols pass data between themselves as chains of mbufs using
 * the pr_input and pr_output hooks.  Pr_input passes data up (towards
 * UNIX) and pr_output passes it down (towards the imps); control
 * information passes up and down on pr_ctlinput and pr_ctloutput.
 * The protocol is responsible for the space occupied by any the
 * arguments to these entries and must dispose it.
 *
 * The userreq routine interfaces protocols to the system and is
 * described below.
 */
 struct protosw {
 ;short;pr_type;;;/* socket type used for */
 ;struct;domain *pr_domain;;/* domain protocol a member of */
 ;short;pr_protocol;;;/* protocol number */
 ;short;pr_flags;;;/* see below */
 #ifdef;_KERNEL
 ;void;(*pr_input)();;;/* prototypes per-domain */
 ;int;(*pr_output)();;;/* prototypes per-domain */
 ;void;(*pr_ctlinput)(int, struct sockaddr *, caddr_t);
 ;int;(*pr_ctloutput)(int, struct socket *, int, int, struct mbuf **);
 ;int;(*pr_usrreq)(struct socket *, int,
 ;;;;struct mbuf *, struct mbuf *, struct mbuf *);
 ;int;(*pr_send)(struct socket *, struct mbuf *, struct uio *,
 ;;;;struct mbuf *, struct mbuf *, int);
 ;int;(*pr_receive)(struct socket *, struct mbuf **, struct uio *,
 ;;;;struct mbuf **, struct mbuf **, int *);
 ;void;(*pr_init)(void);
 ;void;(*pr_fasttimo)(void);
 ;void;(*pr_slowtimo)(void);
 ;void;(*pr_drain)(void);
 #else
 /* protocol-protocol hooks */
 ;void;(*pr_input)();;;/* input to protocol (from below) */
 ;int;(*pr_output)();;;/* output to protocol (from above) */
 ;void;(*pr_ctlinput)();;/* control input (from below) */
 ;int;(*pr_ctloutput)();;/* control output (from above) */
 /* user-protocol hook */
 ;int;(*pr_usrreq)();;;/* user request: see list below */
 ;int ;(*pr_send)();;;/* protocol-specific sosend */
 ;int;(*pr_receive)();;/* protocol-specific soreceive */
 /* utility hooks */
 ;void;(*pr_init)();;;/* initialization hook */
 ;void;(*pr_fasttimo)();;/* fast timeout (200ms) */
 ;void;(*pr_slowtimo)();;/* slow timeout (500ms) */
 ;void;(*pr_drain)();;;/* flush any excess space possible */
 #endif
 };

 #define;PR_SLOWHZ;2;;/* 2 slow timeouts per second */
 #define;PR_FASTHZ;5;;/* 5 fast timeouts per second */

 /*
 * Values for pr_flags.
 * PR_ADDR requires PR_ATOMIC;
 * PR_ADDR and PR_CONNREQUIRED are mutually exclusive.
 */
 #define;PR_ATOMIC;0x01;;/* exchange atomic messages only */
 #define;PR_ADDR;;0x02;;/* addresses given with messages */
 #define;PR_CONNREQUIRED;0x04;;/* connection required by protocol */
 #define;PR_WANTRCVD;0x08;;/* want PRU_RCVD calls */
 #define;PR_RIGHTS;0x10;;/* passes capabilities */
 #define PR_NOEOR        0x4000          /* Records not supported */
 #define PR_INTRLEVEL    0x8000          /* Protocol runs on interrupt level */
```

```

/*
 * The arguments to usrreq are:
 *;(*protosw[].pr_usrreq)(up, req, m, nam, control);
 * where up is a (struct socket *), req is one of these requests,
 * m is a optional mbuf chain containing a message,
 * nam is an optional mbuf chain containing an address,
 * and control is a pointer to a control chain or nil.
 * The protocol is responsible for disposal of the mbuf chain m,
 * the caller is responsible for any space held by nam and control.
 * A non-zero return from usrreq gives an
 * UNIX error number which should be passed to higher level software.
 */
#define;PRU_ATTACH;;0;/* attach protocol to up */
#define;PRU_DETACH;;1;/* detach protocol from up */
#define;PRU_BIND;;2;/* bind socket to address */
#define;PRU_LISTEN;;3;/* listen for connection */
#define;PRU_CONNECT;;4;/* establish connection to peer */
#define;PRU_ACCEPT;;5;/* accept connection from peer */
#define;PRU_DISCONNECT;;6;/* disconnect from peer */
#define;PRU_SHUTDOWN;;7;/* won′t send any more data */
#define;PRU_RCVD;;8;/* have taken data; more room now */
#define;PRU_SEND;;9;/* send this data */
#define;PRU_ABORT;;10;/* abort (fast DISCONNECT, DETACH) */
#define;PRU_CONTROL;;11;/* control operations on protocol */
#define;PRU_SENSE;;12;/* return status into m */
#define;PRU_RCVOOB;;13;/* retrieve out of band data */
#define;PRU_SENDOOB;;14;/* send out of band data */
#define;PRU_SOCKADDR;;15;/* fetch socket′s address */
#define;PRU_PEERADDR;;16;/* fetch peer′s address */
#define;PRU_CONNECT2;;17;/* connect two sockets */
/* begin for protocols internal use */
#define;PRU_FASTTIMO;;18;/* 200ms timeout */
#define;PRU_SLOWTIMO;;19;/* 500ms timeout */
#define;PRU_PROTORCV;;20;/* receive from below */
#define;PRU_PROTOSEND;;21;/* send to below */

#define;PRU_NREQ;;21

#ifndef;CONST
#define CONST
#endif

#ifdef PRUREQUESTS
CONST char;*prurequests[] = {
;″ATTACH″,;″DETACH″,;″BIND″,;;″LISTEN″,
;″CONNECT″,;″ACCEPT″,;″DISCONNECT″,;″SHUTDOWN″,
;″RCVD″,;;″SEND″,;;″ABORT″,;″CONTROL″,
;″SENSE″,;″RCVOOB″,;″SENDOOB″,;″SOCKADDR″,
;″PEERADDR″,;″CONNECT2″,;″FASTTIMO″,;″SLOWTIMO″,
;″PROTORCV″,;″PROTOSEND″
};
#endif

/*
 * The arguments to the ctlinput routine are
 *;(*protosw[].pr_ctlinput)(cmd, sa, arg);
 * where cmd is one of the commands below, sa is a pointer to a sockaddr,
 * and arg is an optional caddr_t argument used within a protocol family.
 */
#define;PRC_IFDOWN;;0;/* interface transition */
#define;PRC_ROUTEDEAD;;1;/* select new route if possible ??? */
#define;PRC_QUENCH2;;3;/* DEC congestion bit says slow down */
#define;PRC_QUENCH;;4;/* some one said to slow down */
#define;PRC_MSGSIZE;;5;/* message size forced drop */
#define;PRC_HOSTDEAD;;6;/* host appears to be down */
#define;PRC_HOSTUNREACH;;7;/* deprecated (use PRC_UNREACH_HOST) */
#define;PRC_UNREACH_NET;;8;/* no route to network */
#define;PRC_UNREACH_HOST;9;/* no route to host */
#define;PRC_UNREACH_PROTOCOL;10;/* dst says bad protocol */
#define;PRC_UNREACH_PORT;11;/* bad port # */
/* was;PRC_UNREACH_NEEDFRAG;12;   (use PRC_MSGSIZE) */
#define;PRC_UNREACH_SRCFAIL;13;/* source route failed */
#define;PRC_REDIRECT_NET;14;/* net routing redirect */
#define;PRC_REDIRECT_HOST;15;/* host routing redirect */
#define;PRC_REDIRECT_TOSNET;16;/* redirect for type of service & net */
#define;PRC_REDIRECT_TOSHOST;17;/* redirect for tos & host */
#define;PRC_TIMXCEED_INTRANS;18;/* packet lifetime expired in transit */
#define;PRC_TIMXCEED_REASS;19;/* lifetime expired on reass q */
#define;PRC_PARAMPROB;;20;/* header incorrect */
#define PRC_IFATTACH;;21;/* if attach notification */
#define PRC_IFDETACH;;22;/* if detach notification */

#define;PRC_NCMDS;;23

#define;PRC_IS_REDIRECT(cmd);\
;((cmd) >= PRC_REDIRECT_NET && (cmd) <= PRC_REDIRECT_TOSHOST)

#ifdef PRCREQUESTS
CONST char;*prcrequests[] = {
;"IFDOWN", "ROUTEDEAD", "#2", "DEC-BIT-QUENCH2",
;"QUENCH", "MSGSIZE", "HOSTDEAD", "#7",
;"NET-UNREACH", "HOST-UNREACH", "PROTO-UNREACH", "PORT-UNREACH",
;"#12", "SRCFAIL-UNREACH", "NET-REDIRECT", "HOST-REDIRECT",
;"TOSNET-REDIRECT", "TOSHOST-REDIRECT", "TX-INTRANS", "TX-REASS",
;"PARAMPROB", "IFATTACH", "IFDETACH"
};
#endif

/*
 * The arguments to ctloutput are:
 *;(*protosw[].pr_ctloutput)(req, so, level, optname, optval);
 * req is one of the actions listed below, so is a (struct socket *),
 * level is an indication of which protocol layer the option is intended.
 * optname is a protocol dependent socket option request,
 * optval is a pointer to a mbuf-chain pointer, for value-return results.
 * The protocol is responsible for disposal of the mbuf chain *optval
 * if supplied,
 * the caller is responsible for any space held by *optval, when returned.
 * A non-zero return from usrreq gives an
 * UNIX error number which should be passed to higher level software.
 */
#define;PRCO_GETOPT;0
#define;PRCO_SETOPT;1

#define;PRCO_NCMDS;2

#ifdef PRCOREQUESTS
```

```
CONST char;*prcorequests[] = {
;"GETOPT", "SETOPT"
};
#endif

#endif
```

Index

ITSO Redbook Evaluation

International Technical Support Organization
Learning Practical TCP/IP for AIX V3.2/V4.1 Users:
Hints and Tips for Debugging and Tuning
May 1996

Publication No. SG24-4381-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____

Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a)	Are you an employee of IBM or its subsidiaries:	Yes____	No____
b)	Do you work in the USA?	Yes____	No____
c)	Was this redbook published in time for your needs?	Yes____	No____
d)	Did this redbook meet your needs?	Yes____	No____

If no, please explain:

____

____

What other topics would you like to see in this redbook?

____

____

What other redbooks would you like to see published?

____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

______________________________ ______________________________
Name Address

______________________________ ______________________________
Company or Organization

______________________________ ______________________________
Phone No.

ITSO Redbook Evaluation
SG24-4381-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department HZ8, Building 678
P.O. BOX 12195
RESEARCH TRIANGLE PARK NC
USA 27709-2195

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.

SG24-4381-00