PROTOCOL MODELING FOR CONFORMANCE TESTING: CASE STUDY FOR THE ISDN LAPD PROTOCOL

Mostafa Hashem Sherif and M. Ümit Uyar

Mostafa Hashem Sherif *is a member of technical staff in the Advanced Digital Signal Processing Department, and* ***M. Ümit Uyar*** *is a member of technical staff in the Network Interfaces and Standards Department. They are with AT&T Bell Laboratories in Holmdel, New Jersey. Mr. Sherif works on systems engineering and standardization of wideband packet protocols. He joined AT&T Bell Laboratories in 1983 and has both a B.Sc. in electronics and communications and an M.Sc. in circuits from the Faculty of Engineering, Cairo University, Egypt; and a Ph.D. in engineering from the University of California at Los Angeles. Mr. Uyar joined AT&T Bell Laboratories in 1986 and has a B.S. in electrical engineering from Istanbul Teknik Üniversitesi, Turkey, and both an M.S. and Ph.D. in electrical engineering from* (continued on page 70)

In this paper, we present a generic approach for modeling a communications protocol with state transitions and window and timer mechanisms, and for generating conformance test sequences automatically. Based on this model, minimum-cost (i.e., minimum run time) conformance test sequences can be generated by using a method based on *unique input/output* sequences and the *Rural Chinese Postman* tours. As a case study, this method is applied to the Integrated Services Digital Network link-access protocol on the D channel.

Introduction

It is well established that conformance testing of protocol implementations is needed to ensure reliable communications in computer networks. Recently, it became apparent that complex protocols require formal methods to define the test conditions and the conformance tests. Formal test-generation methods allow an increase in test coverage for a given number of tests and a reduction of the testing time. In particular, the method we describe, which is based on the unique input/output (UIO) sequences[1] and Rural Chinese Postman tours,[2–4] is suitable for many applications. This method takes into account constraints on the controllability and observability of protocol states. The tests generated by this method completely cover the protocol state transitions, while minimizing the run time of tests. (Refer to the paper by Dahbura, Sabnani, and Uyar in this issue.[5])

Although they vary in details, the increasing number of link- and network-layer protocols have the same basic function: to transfer information from one point to another in an orderly and error-free way. Furthermore, many protocols include similar mechanisms for window-flow control (i.e., window filling, window rotation), error-recovery procedures (such as retransmission), actions when the remote end is busy, timer mechanisms, and so on. The purpose of this paper is to discuss generic ways to model these mechanisms for conformance testing purposes through a case study based on the ISDN LAPD protocol. (ISDN is the Integrated Services Digital Network, and LAPD is the link-access protocol on the D channel. Panel 1 defines acronyms and terms

used in this paper.)

We present a model of LAPD using the *Protocol Specification Language* (PSL).[6] Based on this model, one can generate minimum-cost (i.e., minimum run time), conformance test sequences by using algorithmic methods.

First, we describe the principles of protocol modeling and survey various formal methods that are currently available. Next, we review the main properties of LAPD that are of interest and then discuss how to model these properties for conformance testing—especially the state transitions, window rotation, and timers. Then, we present LAPD conformance test sequences generated with the methods of the UIO sequences and the Rural Chinese Postman Tours. Finally, the main aspects of the case study are summarized.

The paper has three appendices. Appendix A gives the conformance model of LAPD when the network-layer primitives are accessible. Appendix B contains the UIO sequences of LAPD for two models. The first model applies when the network-layer interactions are controllable and observable. The second model corresponds to inaccessible network-layer primitives. [For definitions of the various layers, please refer to the Open System Interconnection (OSI) Reference Model of the International Organization for Standardization (ISO)[7] and to Tanenbaum.[8]] Finally, Appendix C illustrates the test-sequence table, with a sample from the complete sequence.

Protocol Models

A *communications protocol* defines a set of rules and conventions used in the communications among various systems interconnected by a network. A protocol model should offer a clear and unequivocal interpretation of the functions to be performed and should avoid implementation-specific details. However, different levels of abstractions are needed at various stages of development—such as specification, verification, implementation, debugging, and testing the protocol.

Protocol designers specify the overall behavior of a protocol, usually by a high-level specification model, such as flow diagrams and/or the graphical form of the CCITT-recommended Specification Description Language (SDL).[9,10] (CCITT is the International Telegraph and Telephone Consultative Committee.) This representation emphasizes the functions and services that the protocol provides.

Before a protocol is implemented, one must show that the specification delivers the expected services and that it is free of logical errors. A model for verification purposes must contain the interactions of all the protocol entities with each other and with their environment. Formal description techniques—such as SDL, LOTOS (Language of Temporal Ordering Specification),[11] and Estelle[12]—can be used at this stage.

During the implementation of a protocol, models based on these formal description techniques are preferable to high-level representations because formal descriptions provide details that can serve as guidelines for implementation. For debugging a protocol implementation, developers may want to use an "inverse" representation of the protocol to relate variables and state transitions to inputs.

Finally, in conformance testing, the implementation under test is viewed as a *black box* whose behavior is characterized by a set of observable actions, called *outputs*, that are generated by applying a set of externally controllable inputs. Black-box models contain more information than high-level flow diagrams, but are less detailed than models that depict internal events and variables.

A black-box model for a protocol is represented as a finite-state machine (FSM), which is defined by the following elements:

- A set of inputs
- A set of outputs (i.e., observable actions, including no action)
- A set of stable states
- The next-state function, which gives the new state of the FSM for a given input and a current state
- The output function, which defines the output of the

Panel 1. Acronyms and Terms

CCITT	International Telegraph and Telephone Consultative Committee
DISC frame	command frame to disconnect a logical link
DM frame	command response when the logical link is disconnected
EFSM	extended finite-state machine
Estelle	formal description technique defined by the ISO, and based on an extended finite-state machine model
F bit	final bit; a bit in the frame's header that is set to 1 in the response frame to a command frame with the P bit set to 1
FRMR	frame reject (response)
FSM	finite-state machine
I-frame	information frame (command): contains a layer-3 packet
I.441/Q.921	CCITT recommendation for the ISDN link-layer protocol, LAPD, to support network-layer protocols
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
IUT	implementation under test
LAPD	CCITT recommended link-access protocol on the D channel
LOTOS	Language of Temporal Ordering Specification, a formal description technique defined by the ISO, and based on temporal ordering of observed behavior
$N(R)$	receive-sequence number in a numbered LAPD frame
$N(S)$	send-sequence number in an I-frame
OSI	Open Systems Interconnection (model)
P bit	poll bit; a bit in the frame's header that is set to 1 in a command frame to force a response with the F bit set to 1
P/F bit	poll or final bit in a frame's header; called *P bit* for a command and *F bit* for a response
PSL	Protocol Specification Language
Q.931	CCITT recommendation that specifies the procedures for establishing, maintaining, and clearing connections at the ISDN user-network interface
REJ	reject (command or response)
RNR	receiver not ready (command or response)
RR	receiver ready (command or response)
SABME	command frame to establish a logical link
SDL	Specification Description Language; formal description technique recommended by the CCITT, and based on an extended finite-state machine model. It has a graphical form and a programming-language form.
T200	LAPD timer, helps ensure transmitted information frames are acknowledged
T203	LAPD timer, prevents the link from staying idle indefinitely
UIO	unique input/output (sequences)
X.25	CCITT recommended packet-switching protocol at link and packet layers
$V(A)$	lower window edge variable
$V(R)$	receive-state variable
$V(S)$	send-state variable
XID	exchange identification (frame)

FSM for a given input and a current state.

An FSM is said to be *deterministic* if, for any given input at any given state, the output and the next state are uniquely defined. In other words, in a deterministic FSM, there is only one state transition starting from a given state for a given input.

In this paper, our description of an FSM that represents a protocol is based on the Protocol Specification Language.[6] The PSL representation that is used here has the following format for declaring the entities of a protocol:

INPUTS	*<list of input messages>*
OUTPUTS	*<list of output messages>*
INITIAL_STATE	*<the protocol state at power-up>*
OTHER_STATES	*<list of remaining states>*

Each state transition is represented with the following format:

current_state : *next_state* WHEN *(input, output)*

In Appendix A, we give an example of this PSL-based description for a black-box description of LAPD.

For protocols that have a large number of states and state transitions, extended FSM (EFSM) models can be used. These models associate each state transition with a condition or a variable assignment. EFSMs reduce the number of distinct states by grouping some of them and using context variables to distinguish among those states that have been coalesced.

Formal Methods for Conformance Testing

During the conformance test of a protocol implementation, every state transition of the FSM that represents the protocol should be tested. Testing a state transition from state v_i to state v_j takes place in three steps:

1. Put the implementation into state v_i.
2. Apply the required input and compare the output generated with the one defined in the specification.
3. Verify that the new state of the FSM is v_j.

Each state transition has an associated cost value that represents the relative difficulty of the state transition. The cost value assigns a larger penalty to those transitions that require a long time (e.g., to wait for expiration of a long timer) or human intervention. (For further details, see the paper by Dahbura, Sabnani, and Uyar in this issue.[5])

The test generation method used in this work relies on UIO sequences and the Rural Chinese Postman tours for LAPD conformance testing. The UIO sequence of a state v_i is a set of inputs that, when applied to the FSM implementation, generate a set of outputs that are unique to state v_i. The UIO sequences are used to verify the new state (step 3 of the above test procedure). The Rural Chinese Postman tours are used to concatenate the state transition tests to minimize the total cost of the resulting test sequence (i.e., the sum of the costs for all tests in the test sequence).[5]

Overview of LAPD

LAPD is a link-layer protocol defined for ISDN applications by CCITT Recommendation I.441/Q.921 to support network-layer protocols, such as the CCITT X.25 packet-layer protocol and Q.931. (Q.931 specifies the procedures for establishing, maintaining, and clearing connections at the ISDN user-network interface.) From a functional viewpoint, LAPD consists of data-link procedures and of a multiplexing and demultiplexing procedure.

The transfer of data to each network-layer entity occurs on a separate *logical link*. A data-control procedure provides flow control and both error detection and error recovery for each logical link. The data-link procedure also responds to service primitives from the network layer by transmitting appropriate commands and responses. (Service primitives indicate the exchange of information needed for layer n to provide its services to layer $n+1$. They can be function calls in a software implementation, or the exchange of electrical signals.)

The multiplexing and demultiplexing procedure

Panel 2. The Nine States of the LAPD Subset

TEI ASSIGN state (s1). This is the implementation's default state after initialization; the LAPD address has been assigned but the data link is not established.

AWAIT EST state (s2). The implementation enters this state after transmitting a SABME command with the P bit set to 1 to establish the data link and starting the acknowledgment timer, T200.

AWAIT REL state (s3). The implementation moves into this state after transmitting a DISC command with the P bit set to 1 and starting the acknowledgment timer, T200.

MF EST NORM state (s4). On receipt of a SABME command, the implementation moves from the TEI ASSIGN state to the MF EST NORM state. In this state, the data link is established.

MF EST REJ state (s5). During data transfer in the MF EST NORM state, the arrival of an out-of-sequence I-frame forces a transition to this state.

MF EST BUSY state (s6). The implementation enters this state when a local-busy condition occurs in the MF EST NORM or the MF EST REJ state.

TM REC NORM state (s7). Expiration of the acknowledgment timer, T200, causes a transition from the MF EST NORM state to the TM REC NORM state.

TM REC REJ state (s8). This state is entered from the MF EST REJ state following expiration of the T200 timer, or from the TM REC NORM state following the reception of an out-of-sequence I-frame.

TM REC BUSY state (s9). The implementation enters this state when T200 expires in the MF EST BUSY state, or when a higher layer entity initiates a local-busy condition in the TM REC REJ or the TM REC NORM state.

regulates access of the various logical links to the shared physical resources. Some options are a *round-robin* scheme and a *priority* scheme. Strictly speaking, examining the multiplexing and demultiplexing is not part of a conformance test because I.441/Q.921 does not recommend a specific procedure.[13] But in some implementations, it may be necessary to treat the traffic on various logical links differently. If so, it is important to verify the conformance of the implementation to the required policy as Sherif, Sabnani, and Nyquist discussed.[14]

LAPD defines two timers, T200 and T203, which do not run concurrently. The function of timer T203 is to prevent the link from remaining indefinitely idle in the MF EST NORM state (see Panel 2). There are no state transitions after timer T203 expires, but the action that follows depends on the link's initial state. The function of timer T200 is to prevent some transmitted information-frame commands (*I-frames*) from remaining unacknowledged. Note that each I-frame contains a network-layer message called a *packet*.

The formal specification presented in this paper assumes the following:

- LAPD addresses for each logical link have already been assigned.
- The poll bit (P bit) of an I-frame is always set to 0.
- The implementation will accept a valid *frame reject* response (FRMR) to signal some error conditions, but it does not transmit them.
- Exchange-identification (XID) frames are not considered.

LAPD Control Portion. This section assumes some familiarity with various LAPD frames. Note that the poll or final (P/F) bit is the same bit in the frame header; it is called *P bit* for a command and *F bit* for a response. The LAPD subset under consideration can be represented in the nine states defined in Panel 2. Three of these states—AWAIT REL, MF EST BUSY, and TM REC BUSY—cannot be entered without the intervention of the network layer.

Window-Flow Control. During normal data transfer, the logical link is in the MF EST NORM state. If the transmission queue contains data packets, the implementation sends new I-frames as long as all the following conditions exist:

- There are data packets to transmit.
- The transmission window is not closed.
- The implementation is not expecting a response to a poll to the other end.
- The remote side is not busy.

Whenever it transmits or retransmits an I-frame, the implementation starts timer T200 if it was not running, and increments the send-state variable $V(S)$ in modulo 128 arithmetic.

When an I-frame arrives with a send-sequence number $N(S)$ that is different from the receive-state variable $V(R)$, the receiver must transmit a *reject* response (REJ) to indicate an out-of-sequence condition to the sender. The receiving implementation will use the receive-sequence number $N(R)$ in the incoming frame to update the lower window edge $V(A)$ and then move to the MF EST REJ state.

If the I-frame is in sequence, the receiver remains in the MF EST NORM state and responds according to the value of the P bit in the I-frame. For example, the receiver must check and restart the timers if the remote side is not busy and any transmitted I-frames have not been acknowledged.

To check the operation of window filling and window rotation, as well as timer recovery in the MF EST NORM state and for any LAPD address, the window size is set to, say, the value 4. With the implementation in the MF EST NORM state, successive I-frames are sent (which requires that network-layer primitives be controllable). The implementation must then send a sequence of four I-frames. If the tester withholds the corresponding acknowledgments, the implementation must refrain from sending the fifth and sixth frames.

After expiration of the timer acknowledgment T200, the implementation must poll the tester by transmitting a *receiver ready* (RR) command with the P bit set to 1. The tester then clears the timer-recovery condition by sending an RR response, with the F bit set to 1 and with $N(R) = 2$ to acknowledge the first two frames. The implementation must now send the fifth and sixth frames.

When the timer T200 expires again, the implementation is expected to poll the tester. The tester will then acknowledge the remaining four frames through an RR response, with the F bit set to 1 and with $N(R) = 6$.

Modeling LAPD for Conformance Testing

Here, we introduce an FSM model for conformance testing of LAPD. Among the modeled features of LAPD are the state transitions (after valid, inopportune, and illegal frames) and the window rotation and timer mechanisms. This section can be easily adopted for other protocols that have similar characteristics.

Modeling State Transitions. A deterministic FSM that represents a protocol needs to include all state transitions defined in the protocol specification. The controllable inputs that can be applied to a protocol implementation fall into three groups:

- *Valid* inputs. These inputs are defined for the "normal" (or expected) behavior of the sender entity. For example, in LAPD, receiving a DISC frame at any connected state (`s4` to `s9`) is an expected stimulus because the far end of a connection is allowed to disconnect at any given state.
- *Inopportune* inputs. These inputs are semantically and syntactically correct, but they arrive unexpectedly (or out of sequence) in a given state. They correspond to an "unexpected" behavior from the far-end entity. For example, I-frames with an invalid $N(S)$ or response frames with an unsolicited F bit are inopportune in any MF EST state (`s4` to `s6`). Also, in state `s3`, reception of a DISC frame is an inopportune event because it indicates the collision of DISC frames; in state `s3`, it indicates a DISC/SABME frame collision—i.e., collision of a DISC with a SABME. (DISC is used to disconnect the logical link, but SABME is used to establish the logical link.)
- *Illegal* inputs. These inputs do not meet the syntax requirements of a protocol and signals a "faulty" behavior from the far-end entity or from the transmission channel. For example, a numbered frame is illegal if its length (excluding flags and the frame-check sequence) is shorter than four octets.

A protocol specification should define the actions that an *implementation under test* (IUT) is to take

after receiving any input from the three groups of stimuli at any given state. Because the number of illegal inputs is infinitely large, only a subset of illegal inputs can be included in the protocol model. The choice of illegal inputs will focus on those errors that are more likely to affect operation when they occur individually (i.e, no multiple errors).

Similarly, for complex protocols such as LAPD, the set of inopportune messages can be large. Here, the model should incorporate a subset of inopportune inputs. Preferably, these inopportune inputs should be those agreed on by both the standards organizations (which define the protocol specification, as well as the testing standards) and the manufacturers.

Valid-frames testing. If the protocol specification is well written, modeling the behavior for valid input frames should be straightforward. At any given state, each distinct input to an IUT corresponds to a state transition in the corresponding FSM model that starts at the IUT's current state and ends at the state defined by the specification. Note that generating no output (i.e., the *null* output) is a valid response.

When defining the model for conformance testing, controllability of a particular protocol implementation is a major issue. Typically, a protocol specification assumes that interactions with the upper entity of the protocol is controllable and observable. The Open Systems Interconnection (OSI) model, however, specifies interfaces among systems and not within a system. Therefore, in a multilayer implementation, the layer boundaries may not be accessible from the outside.

For example, a LAPD implementation resides between (and interacts separately with) the physical- and network-layer protocols of the ISO protocol stack.[7] During conformance testing, the tester may be unable to control and observe the interactions between the network-layer protocol and LAPD. In such a case, some aspects cannot be tested. Therefore, the model for conformance testing should be modified to exclude features and states that are not testable because of implementation restrictions. For example, a model that does not include the service interactions of LAPD with the network layer cannot be used to test the states `s3`, `s6`, and `s9`. Transitions to these states require the intervention of the network layer.

Typically, a protocol specification defines various implementation options to fit various service requirements, and a core set to enable interoperability. Each implementation option corresponds to a different FSM model where the state transitions that represent the options are either different or absent. Therefore, a different conformance testing model is required for each IUT that implements a particular set of options. The model in Appendix B does not include any implementation options.

Inopportune-frames testing. Suppose the specification defines the behavior of an IUT for all or a subset of inopportune messages (i.e., the output and the next state of an IUT). Then, each action that follows an inopportune input is modeled as a state transition. The starting and ending states are defined similar to those for valid inputs.

For simplicity, the LAPD model in Appendix A includes only a subset of the inopportune frames. CCITT Recommendation I.441/Q.921 defines the expected behavior of an IUT for the remaining inopportune cases.

Illegal-frames testing. Modeling illegal behavior requires that we include the definition of the illegal inputs to restrict them to a finite number. In this work, we define a *bad frame* as a frame that is too long or that has an invalid receive-sequence number $N(R)$ or an invalid control field. (According to Section 5.8.5 of I.441/Q.921, it is possible to discard a frame whose length exceeds twice the longest permissible frame length plus one octet.) Hence, in Appendix A, the illegal inputs are modeled as self-loops (i.e., transitions that start and end at the same state) shown as `bad_frame/null`, which means that there is a `null` response to a `bad_frame`.

Parameter-values testing. If all parameter values of every input are tested, then an infeasibly large number

of tests must be implemented. Therefore, the number of inputs is restricted through equivalence partitioning and boundary-value analysis techniques.

Equivalent partitioning is a black-box technique that identifies classes of inputs for which the protocol's behavior is similar in one sense. The basic assumption is that, if a representative value is selected for a class of inputs, then the response of the IUT will be the same for all other values of this class.[15] Because these equivalence classes do not overlap, one set of test data is selected to represent each equivalence class. In the model in Appendix A, the default value of a parameter is considered to represent all normal values. Similarly, a value outside the allowable range is assumed to represent all other invalid values.

Boundary-value analysis concentrates on the boundary of the equivalence class, because practical experience has shown that these values usually cause the majority of errors. For example, only the maximum window size, as opposed to every combination of window sizes, is considered during the window-rotation tests, as described next.

Modeling the Window-Rotation Mechanism. The purpose of this model is to represent the window-rotation mechanism with a deterministic FSM so that it can be incorporated in the transition tour. This model applies to any multiframe established (MF EST) state—i.e., states `s4`, `s5`, or `s6`.

LAPD specifies that the implementation can transmit I-frames as long as the following conditions are satisfied:

1. There are data packets in the queue.
2. The window is not closed.
3. No response frame with the F bit set to 1 is expected.
4. The remote side is not busy.

From the tester's side, the user can always control conditions 2 and 3. Therefore, to model the window-rotation mechanism, it is enough to consider the primitive DL DATA REQUEST, which is modeled as `13_e1`. Through this primitive, the network layer notifies the link layer that there are data packets ready for transmission. Hence, the link layer should transmit I-frames that contain network-layer data as long as the window is not closed.

Panel 3. The Three Pseudo-States of the LAPD Subset

MF EST NORM WIND CLOS state (`s4.1`). This is a pseudo-state used for testing the window mechanism. It is the same as state `s4` but with the window closed.

MF EST REJ WIND CLOS state (`s5.1`). This is a pseudo-state used for testing the window mechanism. It is the same as state `s5` but with the window closed.

MF EST BUSY WIND CLOS state (`s6.1`). This is a pseudo-state used for testing the window mechanism. It is the same as state `s6` but with the window closed.

To model the state where the window is closed, we define a companion for each MF EST state as follows (the "*x*.1" states are fictitious states we introduced just for the analysis): `s4.1` is attached to `s4`, `s5.1` to `s5`, and `s6.1` to `s6` (see Panel 3). We also add a pseudo-primitive, `13_e6_max`, through which the network layer can force transition of the IUT from any MF EST state to its respective companion state and transmission of the last I-frame in the window. If the window mechanism is correctly implemented, the primitive `13_e6`, which would cause transmission of an I-frame in the original state, is not supposed to cause transmission of an I-frame in the companion state. (The pseudo-primitive `13_e6_max` closes the window to prevent a response to `13_e6`.) The implementation can return to the original state after receiving a frame (RR, REJ, RNR, or I) with the receive-sequence number that can allow the window to slide. (RNR is a *receiver not ready*.) We denote these frames differently in the model in Appendix A. Clearly, the window-rotation mechanism cannot be tested if the network-layer primitive `13_e6` and the pseudo-primitive `13_e6_max` are not controllable.

Panel 4. PSL Description of LAPD Timers

```
s1    :    s1 WHEN (timeout_t200, null)
       :    s1 WHEN (timeout_t203, null),

s2    :    s2 WHEN (timeout_t200, sabme_p1)
       :    s1 WHEN (timeout_t200_N200, l3_e7),

s3    :    s3 WHEN (timeout_t200, l3_e7)
       :    s1 WHEN (timeout_t200_N200, disc_p1),

s4    :    s7 WHEN (timeout_t200, rr_p1)
       :    s4 WHEN (timeout_t203, rr_p1),

s5    :    s8 WHEN (timeout_t200, rr_p1)
       :    s5 WHEN (timeout_t203, rr_p1),

s6    :    s9 WHEN (timeout_t200, rnr_p1),
       :    s6 WHEN (timeout_t203, rr_p1),

s7    :    s7 WHEN (timeout_t200, rr_p1),
       :    s2 WHEN (timeout_t200_N200, sabme_p1)

s8    :    s8 WHEN (timeout_t200, rr_p1),
       :    s2 WHEN (timeout_t200_N200, sabme_p1)

s9    :    s9 WHEN (timeout_t200, rnr_p1)
       :    s2 WHEN (timeout_t200_N200, sabme_p1)
```


Modeling the Timers. In any MF EST state (`s4`, `s5`, `s6`), a state transition occurs when the timer T200 expires for the first time. In contrast, the state transition in states `s2`, `s3`, `s7`, `s8`, and `s9` occurs only after T200 has expired a predefined number of times (called N200).

As an illustration, suppose we start from the MF EST NORM state (`s4`). The state changes to TM REC NORM (`s7`) after the first expiration of T200, and the implementation remains in `s7` until the timer expires N200 times. The implementation then moves to the AWAIT EST state (`s2`).

The full description of both timers in the PSL-based notation is given in Panel 4.

Conformance Test Sequences for LAPD

Because the LAPD model described above is a deterministic FSM, we can readily apply transition-tour techniques for automatic generation of tests. In particular, a software tool that implements the test-generation technique based on the UIO sequences and the Rural Chinese Postman tours can generate LAPD conformance test sequences.[12]

As discussed earlier, the first step is to compute the model UIO sequences that, in general, should not include any optional features. Appendix B gives the UIO sequences for the model defined in Appendix A and for the restricted model described below.

In Appendix B, we can see that states `s2` and `s3` can be identified by a single input/output operation, but the remaining states require two input/output operations. (Each operation is separated by a comma in Appendix B.) Furthermore, the UIO sequences may include LAPD frames, timer expirations, and network-layer primitives. Therefore, a separate model is needed for an IUT that does not offer

access to the network-layer interface.

To obtain this restricted model from the model in Appendix A, we delete:

- States `s3`, `s6`, and `s9`
- The primitives between the link and network layers
- The state transitions that correspond to these primitives.

The conformance test sequence for the LAPD model in Appendix A has a total of 1499 input/output message pairs. For the restricted model, the length of the test sequence drops to 545 input/output message pairs. Appendix C presents a sample of the corresponding test execution table. As this table shows, the tester sends the input listed in the "Message to IUT" column to an IUT whose initial state is listed in the "Current state" column. The expected output and the next state of the IUT are given in the "Message from IUT" and "Next state" columns, respectively. The inputs are applied to an IUT in the order indicated by the "Step" column. A test body consists of sending an input, observing the expected output, and verifying the next state. If any response differs from what is expected, then the IUT fails that test.

Let us follow the first two steps of the sample test execution table. These steps correspond to the exchange `dm_f0/sabme_p1` in state `s1`. In step 1, the tester sends a DM frame with the F bit set to 0 to the IUT, which is initially in state `s1`. The IUT should send a SABME frame with the P bit set to 1. According to the model of Appendix A, at this point, the new state of the IUT should be `s2`.

Step 2 consists of applying the UIO sequence for `s2`. The tester sends a DM frame with the F bit set to 1, and the IUT should respond by sending the primitive `13_e7` to the network layer to indicate that the connection attempt has failed. After completion of step 2, the IUT moves back to state `s1`. In a similar way, each modeled aspect of LAPD is tested by checking that the response and the state transition are as defined in the specification.

Various issues about the implementation of such test cases in a test laboratory are discussed in several papers that appear in this issue.[16–18]

Summary

This paper has outlined the general ideas behind a formal method to evaluate conformance to a subset of LAPD. The approach is based on the FSM model of the control portion of LAPD and use of the UIO sequences as signatures of various states. Application of the Rural Chinese Postman algorithm results in a minimum-cost test sequence. This test sequence exercises:

- Various state transitions for valid, inopportune, and illegal inputs
- The window-filling and rotation mechanisms
- Timers.

The general principles outlined here are applicable to the generation of conformance tests for other communications protocols.

Acknowledgments

We would like to thank Stephen J. Griesmer of AT&T Bell Laboratories for his initial contributions and his valuable suggestions throughout the work.

References

1. K. K. Sabnani and A. T. Dahbura, "A protocol test generation procedure," *Computer Networks and ISDN Systems*, Vol. 15, No. 4, September 1988, pp. 285-297.
2. J. K. Lenstra and A. H. G. Rinnooy Kan, "On general routing problems," *Networks*, Vol. 6, No. 3, July 1976, pp. 273-280.
3. A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman tours," *Protocol Specification, Testing and Verification VIII*, Elsevier (North-Holland Publishing), Amsterdam, Holland, 1988, pp. 75-86.
4. M. U. Uyar and A. T. Dahbura, "Optimal test sequence generation for protocols: the Chinese Postman algorithm applied to Q.931," *Proceedings of the IEEE Global Communications Conference*, Houston, Texas, December 1-4, 1986, Vol. 1, pp. 68-72.

5. A. T. Dahbura, K. K. Sabnani, and M. U. Uyar, "Algorithmic Generation of Protocol Conformance Test Sequences," *AT&T Technical Journal*, Vol. 69, No. 1, January/February 1990, pp. 101-118.
6. K. K. Sabnani and A. Lapone, "Protocol analyzer and verifier," *Protocol Specification, Testing and Verification VI*, B. Sarikaya and G. V. Bochmann (eds.), Elsevier (North-Holland Publishing), Amsterdam, Holland, 1987, pp. 29-34.
7. ISO/TC97/SC21, "Information processing systems—Open systems interconnection, Basic reference model," ISO 7498, International Organization for Standardization, Geneva, Switzerland, 1984.
8. A. S. Tanenbaum, *Computer Networks*, second edition, Prentice Hall, Englewood Cliffs, New Jersey, 1988.
9. International Telegraph and Telephone Consultative Committee, "Recommendation Z.101—Z.104 SDL," *CCITT Red Book*, 8th Plenary Assembly, Malaga-Torremolinos, International Telecommunications Union, October 8 to 19, 1984, Geneva, Switzerland, 1985.
10. R. Saracco and P. A. J. Tilanus, "CCITT SDL: Overview of the language and its applications," *Computer Networks and ISDN Systems*, Vol. 13, No. 2, 1987, pp. 65-74.
11. *LOTOS, A Formal Description Technique Based on Temporal Ordering of Observed Behavior*, DIS 8807, International Organization for Standardization, Geneva, Switzerland, August 1987.
12. *ESTELLE, A Formal Description Technique Based on an Extended State Transition Model*, DIS 9074, International Organization for Standardization, Geneva, Switzerland, August 1987.
13. International Telegraph and Telephone Consultative Committee, "Recommendation I.441—ISDN User-Network Interface—Data Link Layer Specification," *CCITT Blue Book*, XIth Plenary Assembly, Melbourne, Australia, November 11 to 25, 1988, International Telecommunications Union, Geneva, Switzerland, 1989.
14. M. H. Sherif, K. K. Sabnani, and J. S. Nyquist, "Conformance tests for LAPD," *30th Midwest Symposium on Circuits and Systems*, Syracuse, New York, August 16 to 18, 1987, G. Glasford and K. Jabbour (eds.), North-Holland Publishing, Amsterdam, Holland, 1988, pp. 1415-1423.
15. *Testing in Software Development*, M. A. Ould and C. Unwin (eds.), Cambridge University Press, New York, 1986, p. 80.
16. H. V. Bertine, W. B. Elsner, K. T. Tewani, and P. K. Verma, "Overview of Protocol Testing Programs, Methodologies, and Standards," *AT&T Technical Journal*, Vol. 69, No. 1, January/February 1990, pp. 7-16.
17. M. Bush, K. Rasmussen, and F. Wong, "Conformance Testing Methodology for OSI Protocols," *AT&T Technical Journal*, Vol. 69, No. 1, January/February 1990, pp. 84-100.
18. D. Hubbard, "Deterministic Execution Testing of FSM-Based Protocols," *AT&T Technical Journal*, Vol. 69, No. 1, January/February 1990, pp. 119-128.

Biographies (continued)

Cornell University. His interests include formal testing and verification of communications protocols, expert systems, and parallel processing.

Appendix A. Example of PSL-based Description

1. Model for LAPD with Network-Layer Interactions

The layer-2 frames considered are:

Frame	Comment
`sabme_p0,`	SABME with the P bit set to 0
`sabme_p1,`	SABME with the P bit set to 1
`disc_p0,`	DISC with the P bit set to 0
`disc_p1,`	DISC with the P bit set to 1
`dm_f0,`	DM with the F bit set to 0
`dm_f1,`	DM with the F bit set to 1
`frmr_f0,`	FRMR with the F bit set to 0
`frmr_f1,`	FRMR with the F bit set to 1
`ua_f1,`	UA with the F bit set to 1
`i_p0,`	I with the P bit set to 0
`i_p1,`	I with the P bit set to 1
`i_p0_cond,`	I with the P bit set to 0 and cond (Note 1)
`i_p0_unexp_ns,`	I with the P bit set to 1 and unexpected $N(S)$
`i_p1_unexp_ns,`	I with the P bit set to 1 and unexpected $N(S)$
`rej_f0,`	REJ response with the F bit set to 0
`rej_f1,`	REJ response the F bit set to 1
`rej_p0,`	REJ command with the P bit set to 0
`rej_p1,`	REJ command with the P bit set to 1
`rnr_f0,`	RNR response with the F bit set to 0
`rnr_f1,`	RNR response with the F bit set to 1
`rnr_p0,`	RNR command with the P bit set to 0
`rnr_p1,`	RNR command with the P bit set to 1
`rr_f0,`	RR response with the F bit set to 0
`rr_f1,`	RR response the F bit set to 1
`rr_p0,`	RR command with the P bit set to 0
`rr_p1,`	RR command with the P bit set to 1
`bad_frame,`	(Note 2)
`timeout_t200,`	T200 expires
`timeout_t203,`	T203 expires
`timeout_t200_N200;`	T200 expires N200 times

NOTES:

1. In the "Frame" column, `cond` means that the implementation can send I-frames because a response with the F bit set to 1 is not expected, data are in the transmit queue, the remote side is not busy, *and* the window is not closed.
2. See the discussion about illegal frames in this paper.

The following is a list of the service primitives between the network layer and the link layer that should be included in the general model:

Input	Comment
13_e1	DL ESTABLISH REQUEST
13_e2	DL RELEASE REQUEST
13_e3	local busy detected
13_e4_cond1	local busy cleared and no received I-frames have been discarded
13_e4_cond2	local busy cleared and I-frames have been discarded: send a REJ frame
13_e4_cond3	local busy cleared with I-frames discarded and a REJ frame transmitted
13_e5	data in transmit queue with window not closed, remote not busy, and response with the F bit set to 1 not expected
13_e6	DL DATA REQUEST
13_e7	DL RELEASE INDICATION
13_e8	DL ESTABLISH INDICATION
13_e9	DL ESTABLISH CONFIRMATION
13_e10	DL RELEASE CONFIRMATION
13_e11	DL DATA INDICATION

Outputs with network-layer interactions include the following combinations of service primitives between the network layer and the link layer, as well as the following layer-2 frames:

Output	Comment
dm_f0_13_e7	DM with F bit set to 0 and DL RELEASE INDICATION
dm_f1_13_e7	DM with F bit set to 1 and DL RELEASE INDICATION
rnr_f0_13_e11	RNR with F bit set to 0 and DL DATA INDICATION
rnr_f1_13_e11	RNR with F bit set to 1 and DL DATA INDICATION
rr_f0_13_e11	RR with F bit set to 0 and DL DATA INDICATION
rr_f1_13_e11	RR with F bit set to 1 and DL DATA INDICATION
i_p0_13_e11	I with P bit set to 0 and DL DATA INDICATION
ua_f0_13_e7	UA with F bit set to 0 and DL RELEASE INDICATION
ua_f0_13_e8	UA with F bit set to 0 and DL ESTABLISH INDICATION
ua_f1_13_e7	UA with F bit set to 1 and DL RELEASE INDICATION
ua_f1_13_e8	UA with F bit set to 1 and DL ESTABLISH INDICATION

• **Pseudo-Frames and Pseudo-Primitives for Window Testing**

Pseudo-frame/ primitive	Comment
i_p0_max	I with the P bit set to 0 and window reopened
i_p1_max	I with the P bit set to 1 and window reopened
rr_p0_max	RR command with the P bit set to 0 and window reopened
rr_p1_max	RR command with the P bit set to 1 and window reopened
rr_f0_max	RR response with the F bit set to 0 and window reopened
rnr_p0_max	RNR command with the P bit set to 0 and window reopened
rnr_p1_max	RNR command with the P bit set to 1 and window reopened
rnr_f0_max	RNR response with the F bit set to 0 and window reopened
rej_p0_max	REJ command with the P bit set to 0 and window reopened
rej_f0_max	REJ response with the F bit set to 0 and window reopened
rej_p1_max	REJ command with the P bit set to 1 and window reopened
l3_e6_max	DL DATA REQUEST with window closed

2. Model for LAPD with Network-Layer and Link-Layer Interactions and Window Rotation

```
INITIAL_STATE     s1;       /* TEI ASSIGN    */
OTHER_STATES
                  s2,       /* AWAIT EST     */
                  s3,       /* AWAIT REL     */
                  s4,       /* MF EST NORM   */
                  s4.1,     /* MF EST NORM WITH WINDOW CLOSED  */
                  s5,       /* MF EST REJ    */
                  s5.1,     /* MF EST REJ WITH WINDOW CLOSED   */
                  s6,       /* MF EST BUSY   */
                  s6.1,     /* MF EST BUSY WITH WINDOW CLOSED  */
                  s7,       /* TM REC NORM   */
                  s8,       /* TM REC REJ    */
                  s9;       /* TM REC BUSY   */

s1   :    s1 WHEN (disc_p0,dm_f0)          inopportune
     :    s1 WHEN (disc_p1,dm_f1)          inopportune
     :    s4 WHEN (sabme_p1,ua_f1_l3_e8)   valid
     :    s4 WHEN (sabme_p0,ua_f0_l3_e8)   valid
     :    s2 WHEN (l3_e1,sabme_p1)         valid
     :    s2 WHEN (dm_f0,sabme_p1)         valid
     :    s1 WHEN (timeout_t200, null)     timer
     :    s1 WHEN (timeout_t203, null),    timer
```

```
s2    :    s1 WHEN (dm_f1, l3_e7)                  valid
      :    s2 WHEN (sabme_p0, ua_f0)               inopportune
      :    s2 WHEN (sabme_p1, ua_f1)               inopportune
      :    s2 WHEN (disc_p0, dm_f0)                inopportune
      :    s2 WHEN (disc_p1, dm_f1)                inopportune
      :    s4 WHEN (ua_f1, l3_e9)                  valid
      :    s2 WHEN (timeout_t200, sabme_p1)        timer
      :    s1 WHEN (timeout_t200_N200, l3_e7),     timer

s3    :    s1 WHEN (ua_f1, l3_e10)                 valid
      :    s1 WHEN (dm_f1, l3_e10)                 valid
      :    s3 WHEN (sabme_p1, dm_f1)               inopportune
      :    s3 WHEN (sabme_p0, dm_f0)               inopportune
      :    s3 WHEN (disc_p1, ua_f1)                inopportune
      :    s3 WHEN (disc_p0, ua_f0)                inopportune
      :    s3 WHEN (timeout_t200, l3_e7)           timer
      :    s1 WHEN (timeout_t200_N200, disc_p1),   timer

s4    :    s4 WHEN (sabme_p0, ua_f0_l3_e8)         valid
      :    s4 WHEN (sabme_p1, ua_f1_l3_e8)         valid
      :    s1 WHEN (disc_p0, ua_f0_l3_e7)          valid
      :    s1 WHEN (disc_p1, ua_f1_l3_e7)          valid
      :    s2 WHEN (l3_e1, sabme_p1)               valid
      :    s3 WHEN (l3_e2, disc_p1)                valid
      :    s6 WHEN (l3_e3, rnr_p0)                 valid
      :    s4 WHEN (l3_e5, i_p0)                   valid
      :    s4.1 WHEN (l3_e6_max, i_p0)             valid
      :    s4 WHEN (l3_e6, i_p0)                   valid
      :    s2 WHEN (dm_f0, sabme_p1)               inopportune
      :    s2 WHEN (dm_f1, sabme_p1)               inopportune
      :    s2 WHEN (frmr_f0, sabme_p1)             inopportune
      :    s2 WHEN (frmr_f1, sabme_p1)             inopportune
      :    s4 WHEN (i_p1, rr_f1_l3_e11)            valid
      :    s4 WHEN (i_p0, rr_f0_l3_e11)            valid
      :    s4 WHEN (i_p0_cond, i_p0_l3_e11)        valid
      :    s5 WHEN (i_p0_unexp_ns, rej_f0)         inopportune
      :    s5 WHEN (i_p1_unexp_ns, rej_f1)         inopportune
      :    s4 WHEN (rr_p0, null)                   valid
      :    s4 WHEN (rr_f0, null)                   valid
      :    s4 WHEN (rr_f1, null)                   inopportune
      :    s4 WHEN (rr_p1, rr_f1)                  valid
      :    s4 WHEN (rnr_p0, null)                  valid
      :    s4 WHEN (rnr_f0, null)                  valid
      :    s4 WHEN (rnr_f1, null)                  inopportune
```

	:	s4 WHEN (rnr_p1, rr_f1)	valid
	:	s4 WHEN (rej_p0, null)	valid
	:	s4 WHEN (rej_f0, null)	valid
	:	s4 WHEN (rej_f1, null)	inopportune
	:	s4 WHEN (rej_p1, rr_f1)	valid
	:	s2 WHEN (bad_frame, sabme_p1)	illegal
	:	s7 WHEN (timeout_t200, rr_p1)	timer
	:	s4 WHEN (timeout_t203, rr_p1),	timer
s4.1	:	s4 WHEN (sabme_p0, ua_f0_l3_e8)	valid
	:	s4 WHEN (sabme_p1, ua_f1_l3_e8)	valid
	:	s1 WHEN (disc_p0, ua_f0_l3_e7)	inopportune
	:	s1 WHEN (disc_p1, ua_f1_l3_e7)	inopportune
	:	s2 WHEN (l3_e1, sabme_p1)	inopportune
	:	s3 WHEN (l3_e2, disc_p1)	inopportune
	:	s6.1 WHEN (l3_e3, rnr_p0)	inopportune
	:	s4.1 WHEN (l3_e6, null)	inopportune
	:	s2 WHEN (dm_f0, sabme_p1)	inopportune
	:	s2 WHEN (dm_f1, sabme_p1)	inopportune
	:	s2 WHEN (frmr_f0, sabme_p1)	inopportune
	:	s2 WHEN (frmr_f1, sabme_p1)	inopportune
	:	s4.1 WHEN (i_p1, rr_f1_l3_e11)	inopportune
	:	s4.1 WHEN (i_p0, rr_f0_l3_e11)	inopportune
	:	s4 WHEN (i_p1_max, rr_f1_l3_e11)	valid
	:	s4 WHEN (i_p0_max, rr_f0_l3_e11)	valid
	:	s4 WHEN (rr_p0_max, null)	valid
	:	s4 WHEN (rr_f0_max, null)	valid
	:	s4 WHEN (rr_p1_max, null)	valid
	:	s4 WHEN (rnr_p0_max, null)	valid
	:	s4 WHEN (rnr_f0_max, null)	valid
	:	s4 WHEN (rnr_p1_max, null)	valid
	:	s4 WHEN (rej_p0_max, null)	valid
	:	s4 WHEN (rej_f0_max, null)	valid
	:	s4 WHEN (rej_p1_max, null)	valid
	:	s5.1 WHEN (i_p0_unexp_ns, rej_f0)	inopportune
	:	s5.1 WHEN (i_p1_unexp_ns, rej_f1)	inopportune
	:	s4.1 WHEN (rr_p0, null)	inopportune
	:	s4.1 WHEN (rr_f0, null)	inopportune
	:	s4.1 WHEN (rr_f1, null)	inopportune
	:	s4.1 WHEN (rr_p1, rr_f1)	inopportune
	:	s4.1 WHEN (rnr_p0, null)	inopportune
	:	s4.1 WHEN (rnr_f0, null)	inopportune
	:	s4.1 WHEN (rnr_f1, null)	inopportune
	:	s4.1 WHEN (rnr_p1, rr_f1)	inopportune

```
        :   s4.1 WHEN (rej_p0, null)          inopportune
        :   s4.1 WHEN (rej_f0, null)          inopportune
        :   s4.1 WHEN (rej_f1, null)          inopportune
        :   s4.1 WHEN (rej_p1, rr_f1)         inopportune
        :   s2 WHEN (bad_frame, sabme_p1)     illegal
        :   s7 WHEN (timeout_t200, rr_p1)     timer
        :   s4.1 WHEN (timeout_t203, rr_p1),  timer

s5      :   s1 WHEN (disc_p0, ua_f0_13_e7)    valid
        :   s1 WHEN (disc_p1, ua_f1_13_e7)    valid
        :   s4 WHEN (sabme_p0, ua_f0_13_e8)   valid
        :   s4 WHEN (sabme_p1, ua_f1_13_e8)   valid
        :   s2 WHEN (dm_f0, sabme_p1)         inopportune
        :   s2 WHEN (dm_f1, sabme_p1)         inopportune
        :   s2 WHEN (frmr_f0, sabme_p1)       inopportune
        :   s2 WHEN (frmr_f1, sabme_p1)       inopportune
        :   s2 WHEN (13_e1, sabme_p1)         valid
        :   s3 WHEN (13_e2, disc_p1)          valid
        :   s6 WHEN (13_e3, rnr_p0)           valid
        :   s5 WHEN (13_e5, i_p0)             valid
        :   s5 WHEN (13_e6, i_p0)             valid
        :   s5.1 WHEN (13_e6_max, i_p0)       valid
        :   s4 WHEN (i_p1, rr_f1_13_e11)      valid
        :   s4 WHEN (i_p0, rr_f0_13_e11)      valid
        :   s4 WHEN (i_p0_cond, i_p0_13_e11)  valid
        :   s5 WHEN (i_p0_unexp_ns, null)     inopportune
        :   s5 WHEN (i_p1_unexp_ns, rr_f1)    inopportune
        :   s5 WHEN (rr_p0, null)             inopportune
        :   s5 WHEN (rr_p1, rr_f1)            inopportune
        :   s5 WHEN (rr_f0, null)             inopportune
        :   s5 WHEN (rr_f1, null)             inopportune
        :   s5 WHEN (rnr_p0, null)            inopportune
        :   s5 WHEN (rnr_f0, null)            inopportune
        :   s5 WHEN (rnr_f1, null)            inopportune
        :   s5 WHEN (rnr_p1, rr_f1)           inopportune
        :   s5 WHEN (rej_f0, null)            inopportune
        :   s5 WHEN (rej_f1, null)            inopportune
        :   s5 WHEN (rej_p0, null)            inopportune
        :   s5 WHEN (rej_p1, rr_f1)           inopportune
        :   s2 WHEN (bad_frame, sabme_p1)     illegal
        :   s8 WHEN (timeout_t200, rr_p1)     timer
        :   s5 WHEN (timeout_t203, rr_p1),    timer

s5.1    :   s1 WHEN (disc_p0, ua_f0_13_e7)    inopportune
```

```
        :   s1 WHEN (disc_p1, ua_f1_13_e7)      inopportune
        :   s4 WHEN (sabme_p0, ua_f0_13_e8)     valid
        :   s4 WHEN (sabme_p1, ua_f1_13_e8)     valid
        :   s5 WHEN (i_p1_max, rr_f1_13_e11)    valid
        :   s5 WHEN (i_p0_max, rr_f0_13_e11)    valid
        :   s5 WHEN (rr_p0_max, null)           valid
        :   s5 WHEN (rr_f0_max, null)           valid
        :   s5 WHEN (rr_p1_max, null)           valid
        :   s5 WHEN (rnr_p0_max, null)          valid
        :   s5 WHEN (rnr_f0_max, null)          valid
        :   s5 WHEN (rnr_p1_max, null)          valid
        :   s5 WHEN (rej_p0_max, null)          valid
        :   s5 WHEN (rej_f0_max, null)          valid
        :   s5 WHEN (rej_p1_max, null)          valid
        :   s2 WHEN (dm_f0, sabme_p1)           inopportune
        :   s2 WHEN (dm_f1, sabme_p1)           inopportune
        :   s2 WHEN (frmr_f0, sabme_p1)         inopportune
        :   s2 WHEN (frmr_f1, sabme_p1)         inopportune
        :   s2 WHEN (13_e1, sabme_p1)           inopportune
        :   s3 WHEN (13_e2, disc_p1)            inopportune
        :   s6.1 WHEN (13_e3, rnr_p0)           inopportune
        :   s5.1 WHEN (13_e6, null)             inopportune
        :   s4.1 WHEN (i_p1, rr_f1_13_e11)      inopportune
        :   s4.1 WHEN (i_p0, rr_f0_13_e11)      inopportune
        :   s5.1 WHEN (i_p0_unexp_ns, null)     inopportune
        :   s5.1 WHEN (i_p1_unexp_ns, rr_f1)    inopportune
        :   s5.1 WHEN (rr_p0, null)             inopportune
        :   s5.1 WHEN (rr_p1, rr_f1)            inopportune
        :   s5.1 WHEN (rr_f0, null)             inopportune
        :   s5.1 WHEN (rr_f1, null)             inopportune
        :   s5.1 WHEN (rnr_p0, null)            inopportune
        :   s5.1 WHEN (rnr_f0, null)            inopportune
        :   s5.1 WHEN (rnr_f1, null)            inopportune
        :   s5.1 WHEN (rnr_p1, rr_f1)           inopportune
        :   s5.1 WHEN (rej_f0, null)            inopportune
        :   s5.1 WHEN (rej_f1, null)            inopportune
        :   s5.1 WHEN (rej_p0, null)            inopportune
        :   s5.1 WHEN (rej_p1, rr_f1)           inopportune
        :   s2 WHEN (bad_frame, sabme_p1)       illegal
        :   s8 WHEN (timeout_t200, rr_p1)       timer
        :   s5.1 WHEN (timeout_t203, rr_p1),    timer

s6      :   s1 WHEN (disc_p0, ua_f0_13_e7)      valid
        :   s1 WHEN (disc_p1, ua_f1_13_e7)      valid
```

```
                :   s2 WHEN (l3_e1, sabme_p1)          valid
                :   s3 WHEN (l3_e2, disc_p1)           valid
                :   s4 WHEN (l3_e4_cond1, rr_p0)       valid
                :   s5 WHEN (l3_e4_cond2, rej_p0)      valid
                :   s5 WHEN (l3_e4_cond3, rr_p0)       valid
                :   s6 WHEN (l3_e5, i_p0)              valid
                :   s6 WHEN (l3_e6, i_p0)              valid
                :   s6.1 WHEN (l3_e6_max, i_p0)        valid
                :   s2 WHEN (dm_f0, sabme_p1)          inopportune
                :   s2 WHEN (dm_f1, sabme_p1)          inopportune
                :   s2 WHEN (frmr_f1, sabme_p1)        inopportune
                :   s2 WHEN (frmr_f0, sabme_p1)        inopportune
                :   s4 WHEN (sabme_p0, ua_f0_l3_e8)    valid
                :   s4 WHEN (sabme_p1, ua_f1_l3_e8)    valid
                :   s6 WHEN (i_p1, rnr_f1_l3_e11)      inopportune
                :   s6 WHEN (i_p0, rnr_f0_l3_e11)      inopportune
                :   s6 WHEN (i_p0_unexp_ns, rnr_f0)    inopportune
                :   s6 WHEN (i_p1_unexp_ns, rnr_f1)    inopportune
                :   s6 WHEN (rr_p0, null)              valid
                :   s6 WHEN (rr_f0, null)              valid
                :   s6 WHEN (rr_f1, null)              valid
                :   s6 WHEN (rr_p1, rnr_f1)            valid
                :   s6 WHEN (rnr_p0, null)             valid
                :   s6 WHEN (rnr_f0, null)             valid
                :   s6 WHEN (rnr_f1, null)             valid
                :   s6 WHEN (rnr_p1, rnr_f1)           valid
                :   s6 WHEN (rej_f0, null)             valid
                :   s6 WHEN (rej_f1, null)             valid
                :   s6 WHEN (rej_p0, null)             valid
                :   s6 WHEN (rej_p1, rnr_f1)           valid
                :   s2 WHEN (bad_frame, sabme_p1)      illegal
                :   s6 WHEN (timeout_t203, rnr_p1)     timer
                :   s9 WHEN (timeout_t200, rnr_p1),    timer

s6.1            :   s1 WHEN (disc_p0, ua_f0_l3_e7)     inopportune
                :   s1 WHEN (disc_p1, ua_f1_l3_e7)     inopportune
                :   s2 WHEN (l3_e1, sabme_p1)          inopportune
                :   s3 WHEN (l3_e2, disc_p1)           inopportune
                :   s4.1 WHEN (l3_e4_cond1, rr_p0)     inopportune
                :   s5.1 WHEN (l3_e4_cond2, rej_p0)    inopportune
                :   s5.1 WHEN (l3_e4_cond3, rr_p0)     inopportune
                :   s6.1 WHEN (l3_e6, null)            inopportune
                :   s4 WHEN (sabme_p0, ua_f0_l3_e8)    valid
                :   s4 WHEN (sabme_p1, ua_f1_l3_e8)    valid
```

```
      :   s6 WHEN (i_p1_max, rr_f1_13_e11)     valid
      :   s6 WHEN (i_p0_max, rr_f0_13_e11)     valid
      :   s6 WHEN (rr_p0_max, null)            valid
      :   s6 WHEN (rr_f0_max, null)            valid
      :   s6 WHEN (rr_p1_max, null)            valid
      :   s6 WHEN (rnr_p0_max, null)           valid
      :   s6 WHEN (rnr_f0_max, null)           valid
      :   s6 WHEN (rnr_p1_max, null)           valid
      :   s6 WHEN (rej_p0_max, null)           valid
      :   s6 WHEN (rej_f0_max, null)           valid
      :   s6 WHEN (rej_p1_max, null)           valid
      :   s2 WHEN (dm_f0, sabme_p1)            inopportune
      :   s2 WHEN (dm_f1, sabme_p1)            inopportune
      :   s2 WHEN (frmr_f1, sabme_p1)          inopportune
      :   s2 WHEN (frmr_f0, sabme_p1)          inopportune
      :   s6.1 WHEN (i_p1, rnr_f1_13_e11)      inopportune
      :   s6.1 WHEN (i_p0, rnr_f0_13_e11)      inopportune
      :   s6.1 WHEN (i_p0_unexp_ns, rnr_f0)    inopportune
      :   s6.1 WHEN (i_p1_unexp_ns, rnr_f1)    inopportune
      :   s6.1 WHEN (rr_p0, null)              inopportune
      :   s6.1 WHEN (rr_f0, null)              inopportune
      :   s6.1 WHEN (rr_f1, null)              inopportune
      :   s6.1 WHEN (rr_p1, rnr_f1)            inopportune
      :   s6.1 WHEN (rnr_p0, null)             inopportune
      :   s6.1 WHEN (rnr_f0, null)             inopportune
      :   s6.1 WHEN (rnr_f1, null)             inopportune
      :   s6.1 WHEN (rnr_p1, rnr_f1)           inopportune
      :   s6.1 WHEN (rej_f0, null)             inopportune
      :   s6.1 WHEN (rej_f1, null)             inopportune
      :   s6.1 WHEN (rej_p0, null)             inopportune
      :   s6.1 WHEN (rej_p1, rnr_f1)           inopportune
      :   s2 WHEN (bad_frame, sabme_p1)        illegal
      :   s6.1 WHEN (timeout_t203, rnr_p1)     timer
      :   s9 WHEN (timeout_t200, rnr_p1),      timer

s7    :   s1 WHEN (disc_p1, ua_f1_13_e7)       valid
      :   s1 WHEN (disc_p0, ua_f0_13_e7)       valid
      :   s2 WHEN (dm_f0, sabme_p1)            inopportune
      :   s2 WHEN (dm_f1, sabme_p1)            inopportune
      :   s2 WHEN (frmr_f0, sabme_p1)          inopportune
      :   s2 WHEN (frmr_f1, sabme_p1)          inopportune
      :   s2 WHEN (13_e1, sabme_p1)            valid
      :   s3 WHEN (13_e2, disc_p1)             valid
      :   s9 WHEN (13_e3, rnr_p0)              valid
```

```
        :   s4 WHEN (sabme_p0, ua_f0_13_e8)          valid
        :   s4 WHEN (sabme_p1, ua_f1_13_e8)          valid
        :   s7 WHEN (i_p0, rr_f0_13_e11)             valid
        :   s7 WHEN (i_p1, rr_f1_13_e11)             valid
        :   s8 WHEN (i_p0_unexp_ns, rej_f0)          inopportune
        :   s8 WHEN (i_p1_unexp_ns, rej_f1)          inopportune
        :   s7 WHEN (rr_p0, null)                    inopportune
        :   s7 WHEN (rr_f0, null)                    inopportune
        :   s4 WHEN (rr_f1, null)                    valid
        :   s7 WHEN (rr_p1, rr_f1)                   inopportune
        :   s7 WHEN (rnr_p0, null)                   inopportune
        :   s7 WHEN (rnr_f0, null)                   inopportune
        :   s4 WHEN (rnr_f1, null)                   valid
        :   s7 WHEN (rnr_p1, rr_f1)                  inopportune
        :   s7 WHEN (rej_f0, null)                   inopportune
        :   s7 WHEN (rej_p0, null)                   inopportune
        :   s4 WHEN (rej_f1, null)                   valid
        :   s7 WHEN (rej_p1, rr_f1)                  inopportune
        :   s2 WHEN (bad_frame, sabme_p1)            illegal
        :   s2 WHEN (timeout_t200_N200, sabme_p1)    timer
        :   s7 WHEN (timeout_t200, rr_p1),           timer
s8      :   s2 WHEN (dm_f0, sabme_p1)                inopportune
        :   s2 WHEN (dm_f1, sabme_p1)                inopportune
        :   s1 WHEN (disc_p1, ua_f1_13_e7)           inopportune
        :   s1 WHEN (disc_p0, ua_f0_13_e7)           inopportune
        :   s4 WHEN (sabme_p0, ua_f0_13_e8)          inopportune
        :   s4 WHEN (sabme_p1, ua_f1_13_e8)          inopportune
        :   s2 WHEN (frmr_f1, sabme_p1)              inopportune
        :   s2 WHEN (frmr_f0, sabme_p1)              inopportune
        :   s2 WHEN (13_e1, sabme_p1)                valid
        :   s3 WHEN (13_e2, disc_p1)                 valid
        :   s9 WHEN (13_e3, rnr_p0)                  valid
        :   s7 WHEN (i_p1, rr_f1_13_e11)             valid
        :   s7 WHEN (i_p0, rr_f0_13_e11)             valid
        :   s8 WHEN (i_p0_unexp_ns, null)            inopportune
        :   s8 WHEN (i_p1_unexp_ns, rr_f1)           inopportune
        :   s8 WHEN (rr_p0, null)                    inopportune
        :   s8 WHEN (rr_f0, null)                    inopportune
        :   s5 WHEN (rr_f1, null)                    valid
        :   s8 WHEN (rr_p1, rr_f1)                   inopportune
        :   s8 WHEN (rnr_p0, null)                   inopportune
        :   s8 WHEN (rnr_f0, null)                   inopportune
        :   s5 WHEN (rnr_f1, null)                   valid
```

```
      :   s8 WHEN (rnr_p1, rr_f1)                 inopportune
      :   s8 WHEN (rej_f0, null)                  inopportune
      :   s8 WHEN (rej_p0, null)                  inopportune
      :   s8 WHEN (rej_f1, null)                  valid
      :   s8 WHEN (rej_p1, rr_f1)                 inopportune
      :   s2 WHEN (bad_frame, sabme_p1)           illegal
      :   s2 WHEN (timeout_t200_N200, sabme_p1)   timer
      :   s8 WHEN (timeout_t200, rr_p1),          timer

s9    :   s1 WHEN (disc_p0, ua_f0_13_e7)          valid
      :   s1 WHEN (disc_p1, ua_f1_13_e7)          valid
      :   s2 WHEN (dm_f0, sabme_p1)               inopportune
      :   s2 WHEN (dm_f1, sabme_p1)               inopportune
      :   s2 WHEN (frmr_f0, sabme_p1)             inopportune
      :   s2 WHEN (frmr_f1, sabme_p1)             inopportune
      :   s4 WHEN (sabme_p0, ua_f0_13_e8)         valid
      :   s4 WHEN (sabme_p1, ua_f1_13_e8)         valid
      :   s2 WHEN (13_e1, sabme_p1)               valid
      :   s3 WHEN (13_e2, disc_p1)                valid
      :   s7 WHEN (13_e4_cond1, rr_p0)            valid
      :   s8 WHEN (13_e4_cond2, rej_p0)           valid
      :   s8 WHEN (13_e4_cond3, rr_p0)            valid
      :   s9 WHEN (i_p1, rnr_f1_13_e11)           valid
      :   s9 WHEN (i_p0, rnr_f0_13_e11)           valid
      :   s9 WHEN (i_p0_unexp_ns, rnr_f0)         inopportune
      :   s9 WHEN (i_p1_unexp_ns, rnr_f1)         inopportune
      :   s9 WHEN (rr_f0, null)                   inopportune
      :   s6 WHEN (rr_f1, null)                   valid
      :   s9 WHEN (rr_p0, null)                   inopportune
      :   s9 WHEN (rr_p1, rnr_f1)                 inopportune
      :   s9 WHEN (rnr_f0, null)                  inopportune
      :   s6 WHEN (rnr_f1, null)                  valid
      :   s9 WHEN (rnr_p0, null)                  inopportune
      :   s9 WHEN (rnr_p1, rnr_f1)                inopportune
      :   s8 WHEN (rej_f0, null)                  inopportune
      :   s6 WHEN (rej_f1, null)                  valid
      :   s8 WHEN (rej_p0, null)                  inopportune
      :   s9 WHEN (rej_p1, rnr_f1)                inopportune
      :   s2 WHEN (bad_frame, sabme_p1)           illegal
      :   s9 WHEN (timeout_t200, rnr_p1)          timer
      :   s2 WHEN (timeout_t200_N200, sabme_p1)   timer
```

Appendix B. UIO Sequences for Complete and Restricted LAPD Models

Complete model (includes network-layer interactions)

```
UIO sequence for s1   : disc_p0/dm_f0, l3_e1/sabme_p1
UIO sequence for s2   : dm_f1/l3_e7
UIO sequence for s3   : ua_f1/l3_e10
UIO sequence for s4   : l3_e5/i_p0, i_p0_unexp_ns/rej_f0
UIO sequence for s4.1 : i_p1_max/rr_f1_l3_e11, i_p0_unexp_ns/rej_f0
UIO sequence for s5   : l3_e5/i_p0, i_p0_unexp_ns/null
UIO sequence for s5.1 : i_p1_max/rr_f1_l3_e11, i_p0_unexp_ns/null
UIO sequence for s6   : l3_e4_cond1/rr_p0, l3_e5/i_p0
UIO sequence for s6.1 : l3_e4_cond1/rr_p0, i_p1_max/rr_f1_l3_e11
UIO sequence for s7   : i_p0_unexp_ns/rej_f0, timeout_t200_N200/null, dm_f1/l3_e7
UIO sequence for s8   : i_p0_unexp_ns/null, timeout_t200_N200/null, dm_f1/l3_e7
UIO sequence for s9   : l3_e4_cond1/rr_p0, timeout_t200_N200/null, dm_f1/l3_e7
```

Restricted model (without network-layer interactions)

```
UIO sequence for s1 : disc_p0/dm_f0, dm_f0/sabme_p1
UIO sequence for s2 : dm_f1/null
UIO sequence for s4 : i_p0_unexp_ns/rej_f0, i_p0_cond/i_p0
UIO sequence for s5 : i_p0_unexp_ns/null, i_p0_cond/i_p0
UIO sequence for s7 : i_p0_unexp_ns/rej_f0, timeout_t200_N200/null, dm_f1/null
UIO sequence for s8 : i_p0_unexp_ns/null, timeout_t200_N200/null, dm_f1/null
```

Appendix C. Sample Test Execution Table

This is the test execution table for the LAPD model in Appendix A.

Test sequence table

Step number	Current state	Next state	Message to IUT	Message from IUT
			Test body for dm_f0/sabme_p1 in s1	
1	s1	s2	dm_f0	sabme_p1
2	s2	s1	dm_f1	13_e7
			Test body for 13_e1/sabme_p1 in s1	
3	s1	s2	13_e1	sabme_p1
4	s2	s1	dm_f1	13_e7
			Test body for disc_p0/dm_f0 in s1	
5	s1	s1	disc_p0	dm_f0
6	s1	s1	disc_p0	dm_f0
7	s1	s2	13_e1	sabme_p1
			Test body for timeout_t200/null in s2	
8	s2	s1	timeout_t200	null
9	s1	s1	disc_p0	dm_f0
10	s1	s2	13_e1	sabme_p1
			Test body for dm_f1/13_e7 in s2	
11	s2	s1	dm_f1	13_e7
12	s1	s1	disc_p0	dm_f0
13	s1	s2	13_e1	sabme_p1
			Test body for sabme_p1/ua_f1 in s2	
14	s2	s2	sabme_p1	ua_f1
15	s2	s1	dm_f1	13_e7

(Manuscript received July 5, 1989)