NFV Infrastructure Metrics for Monitoring Virtualized Network Deployments

Alliance for Telecommunications Industry Solutions
January 2018

ATIS-I-0000062

Abstract

Service Providers are adopting advanced network virtualization technologies like SDN and NFV to benefit from time-to-market flexibility and service agility. Virtual Network Functions (VNFs) may be deployed across diverse, multi-vendor execution environments with variable configuration options and capabilities impacting performance and capacity. This report presents an overview of existing performance metrics/KPIs to assist in dimensioning the NFV infrastructure to meet the needs of the NFV applications it supports.

NFV Characterization and Capacity Planning Focus Group Leadership

Intel Corporation:
Rajesh Gadiyar
Tim Verrall

Juniper:
Qasim Arham
Pavan Kurapati

About ATIS

As a leading technology and solutions development organization, the Alliance for Telecommunications Industry Solutions (ATIS) brings together the top global ICT companies to advance the industry's business priorities. ATIS' 150 member companies are currently working to address 5G, cybersecurity, robocall mitigation, IoT, artificial intelligence-enabled networks, the all-IP transition, network functions virtualization, smart cities, emergency services, network evolution, quality of service, billing support, operations, and much more. These priorities follow a fast-track development lifecycle – from design and innovation through standards, specifications, requirements, business use cases, software toolkits, open source solutions, and interoperability testing.

ATIS is accredited by the American National Standards Institute (ANSI). ATIS is the North American Organizational Partner for the 3rd Generation Partnership Project (3GPP), a founding Partner of the oneM2M global initiative, a member of the International Telecommunication Union (ITU), and a member of the Inter-American Telecommunication Commission (CITEL). For more information, visit www.atis.org.

Notice of Disclaimer and Limitation of Liability

The information provided in this document is directed solely to professionals who have the appropriate degree of experience to understand and interpret its contents in accordance with generally accepted engineering or other professional standards and applicable regulations. No recommendation as to products or vendors is made or should be implied.

NO REPRESENTATION OR WARRANTY IS MADE THAT THE INFORMATION IS TECHNICALLY ACCURATE OR SUFFICIENT OR CONFORMS TO ANY STATUTE, GOVERNMENTAL RULE OR REGULATION, AND FURTHER, NO REPRESENTATION OR WARRANTY IS MADE OFMERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR AGAINST INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS. ATIS SHALL NOT BE LIABLE, BEYOND THE AMOUNT OF ANY SUM RECEIVED IN PAYMENT BY ATIS FOR THIS DOCUMENT, AND IN NO EVENT SHALL ATIS BE LIABLE FOR LOST PROFITS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES. ATIS EXPRESSLY ADVISES THAT ANY AND ALL USE OF OR RELIANCE UPON THE INFORMATION PROVIDED IN THIS DOCUMENT IS AT THE RISK OF THE USER.

NOTE - The user's attention is called to the possibility that compliance with this standard may require use of an invention covered by patent rights. By publication of this standard, no position is taken with respect to whether use of an invention covered by patent rights will be required, and if any such use is required no position is taken regarding the validity of this claim or any patent rights in connection therewith. Please refer to [http://www.atis.org/legal/patentinfo.asp] to determine if any statement has been filed by a patent holder indicating a willingness to grant a license either without compensation or on reasonable and non-discriminatory terms and conditions to applicants desiring to obtain a license.

Copyright Information

Alliance for Telecommunications Industry Solutions
1200 G Street, NW, Suite 500
Washington, DC 20005

Contents

1. Introduction 2

2. Infrastructure Resource requirements across Service types 2

3. NFV Infrastructure Resources 12

4. Summary 64

5. References 64

1. Introduction

Driven by the requirements for agile service delivery Service Providers (SPs) are adopting Network Function Virtualization (NFV) and Software Defined Networks (SDN) technology to deploy a wide range of network functions and applications across cloud, mobility, enterprise, and core and edge network services. The supporting Infrastructure for these technologies includes industry standard computing systems, storage systems, networking systems, virtualization support systems (such as hypervisors), and management systems for the virtual and physical resources. This approach delivers significant improvements in service velocity and network flexibility while also reducing costs.

There are many potential suppliers of NFV Infrastructure components and significant flexibility in configuring these components for best performance. There are also many potential suppliers of Virtual Network Functions (VNFs), adding to the combinations possible in this environment. With this added flexibility, SPs must work to ensure that their infrastructure is correctly dimensioned and that resources are matched to services so that the performance requirements of all services are met.

This report provides a reference framework for NFV deployment planning that aims to assist in all aspects of ensuring, across a range of key performance indicators (KPIs), that the NFV Infrastructure is dimensioned to meet the needs of the NFV applications it supports. The framework will address:

- The characterization of different service types in terms of their critical KPIs and the parameters that describe their infrastructure requirements.
- An analysis of how to measure KPIs to assess the resource utilization and headroom on operational NFV infrastructure.

2. Infrastructure Resource Requirements across Service Types

The computational and networking requirements across VNFs can vary significantly depending on the type of network function. For example, latency and packets-per-second rate are key factors for VNFs that require packet-forwarding performance (i.e., firewalls and routers). Other network functions such as Session Border Controllers (SBCs) are bounded by network and compute resources for session state management. Sometimes, different sub-functions can be combined to form a higher-level, multi-

component VNF, such as a virtual router. Each of these components can exhibit different performance impacts and scalability requirements.

In the table below, several different VNF service types are examined to identify key infrastructure requirements. For any specific type of VNF deployed there are fixed and limited resources that must be tracked and trended to understand the capacity and performance of those functions.

Type of VNF	Example Usage	Example KPIs and Metrics Networking Virtualized Resources	Example KPIs and Metrics for Application Services	Example KPIs and Metrics for Virtualized Storage
General applicability (not specific to a particular VNF or network plugin type or host)	NVFI	VIF egress.drops VIF egress errors VIF bps In/Out VIF PPS In/Out VIF total bytes VIF total packets VIF Flows/sec VNF egress.drops VNF egress errors VNF bps In/Out VNF PPS In/Out VNF total bytes VNF total packets VNF Flows/sec VNF Number of Active Flows VNF Throughput BPS & PPS Host egress.drops Host egress errors Host bps In/Out Host PPS In/Out Host total bytes Host total packets Host Flows/sec Host Number of Active Flows Host Throughput BPS & PPS Host ipv4tables.rule_count Host ipv6tables.rule_count	Number of VCPUs (IO, Processing Cores) VNF VCPU Utilization VNF VCPU Pinning to Physical Cores Hyper-Threading NUMA VCPU Poll/Event Driven Memory size & Hugepages/ Default L1/L2/L3 Cache utilization & Cache Misses Per VF VLAN range VF Anti-Spoofing Count VF Multicast/Broadcast Host.cpu.io_wait Host.cpu.ipc Host.cpu.l3_cache.miss Host.cpu.l3_cache.usage Host.cpu.mem_bw.local Host.cpu.mem_bw.remote Host.cpu.mem_bw.total Host.cpu.usage	host.disk.io.read host.disk.io.write host.disk.response_time host.disk.read_response_time host.disk.write_response_time host.disk.smart.hdd.command_timeout host.disk.smart.hdd.current_pending_ sector_count host.disk.smart.hdd.offline_ uncorrectable host.disk.smart.hdd.reallocated_sector_ count host.disk.smart.hdd.reported_ uncorrectable_errors host.disk.smart.ssd.available_reserved_ space host.disk.smart.ssd.media_wearout_i ndicator host.disk.smart.ssd.reallocated_sector_ count host.disk.smart.ssd.wear_leveling_ count host.disk.usage.bytes host.disk.usage.percent host.memory.usage host.memory.swap.usage host.memory.dirty.rate host.memory.page_fault.rate host.memory.page_in_out.rate

Type of VNF	Example Usage	Example KPIs and Metrics Networking Virtualized Resources	Example KPIs and Metrics for Application Services	Example KPIs and Metrics for Virtualized Storage
Carrier grade Virtual Router	**Virtual Provider Edge for:** Business VPN Business Internet Virtual Broadband Network Gateway For residential internet	DPDK_Ring_RX_PKTS DPDK_Ring_TX_PKTS DPDK_Ring_Drops **Pipeline Model:** IO_n_PPE_n_Enqueues IO_n_PPE_n_Dequeues IO_n_PPE_n_Drops PPE_n_IO_n_Enqueues PPE_n_IO_n_Dequeues PPE_n_IO_n_Drops PPE_n_Exception_Packets_TX PPE_n_Exception_Pckets_RX PPE_n_Exception_Packets_Drop Number_of_Active_Flows Number_of_Flows_Cached Flow_n_Action_List	**Defined in OpenConfig/IETF:** BGP Nbr Up/Down BGP Peers per Peer-Group BGP Advertised Prefixes/Received Prefixes/Suppressed prefixes MPLS LSP Statistics OSPF LSAs OSPF Session count ACL drop count ACL accept count Policer drops Tail dropped packets RED dropped packets	
Virtual Switch/ Virtual Router Plugin	**OpenContrail, OpenvSwitch**: Software-forwarding element running on the host that is responsible for building overlay and provide network connectivity to VNFs	**Physical Interface:** vRouter.pInterface.drop_stats vRouter.pInterface.in_bytes vRouter.pInterface.in_pkts vRouter.pInterface.out_bytes vRouter.pInterface.out_pkts vRouter Interface: vRouter.total_flows vRouter.aged_flows vRouter.exception_packets vRouter.drop_stats_flow_queue_limit_exceeded vRouter.drop_stats_flow_table_full vRouter.drop_stats_vlan_fwd_enq vRouter.drop_stats_vlan_fwd_tx vRouter.flow_export_drops vRouter.flow_export_sampling_drops vRouter.flow_rate_active_flows vRouter.flow_rate_added_flows vRouter.flow_rate_deleted_flows vRouter.Interface.in_pkts_ewm vRouter.Interface.out_pkts_ewmv vRouter.phy_band_in_bps vRouter.phy_band_out_bps		

Type of VNF	Example Usage	Example KPIs and Metrics Networking Virtualized Resources	Example KPIs and Metrics for Application Services	Example KPIs and Metrics for Virtualized Storage
		vRouter.in_bps_ewm vRouter.out_bps_ewmv **Virtual Machine Interface (tap interface):** vRouter.Virtual_Machine_Interface.drop_stats vRouter.Virtual_Machine_Interface.in_bytes vRouter.Virtual_Machine_Interface.in_pkts vRouter.Virtual_Machine_Interface.out_bytes vRouter.Virtual_Machine_Interface.out_pkts vRouter.Virtual_Machine_Interface.in_pkts_ewm vRouter.Virtual_Machine_Interface.out_pkts_ewm vRouter.Virtual_Machine_Interface.active_flows_ewm vRouter.Virtual_Machine_Interface.added_flows_ewm vRouter.Virtual_Machine_Interface.deleted_flows_ewm vRouter.Virtual_Machine_Interface.flow_rate vRouter.Virtual_Machine_Interface.if_stats vRouter.Virtual_Machine_Interface.if_in_pkts_ewm vRouter.Virtual_Machine_Interface.if_out_pkts_ewm **Virtual Network Interface:** vRouter.VirtualNetwork.vn_stats vRouter.VirtualNetwork.in_bytes vRouter.VirtualNetwork.in_tpkts vRouter.VirtualNetwork.out_bytes vRouter.VirtualNetwork.out_tpktsvRouter.VirtualNetwork.in_tpkts vRouter.VirtualNetwork.out_bytes vRouter.VirtualNetwork.out_tpkts		
Proxy		**Physical Interface:** Proxy.pInterface.drop_stats Proxy.pInterface.in_bytes Proxy.pInterface.in_pkts Proxy.pInterface.out_bytes Proxy.pInterface.out_pkts **Proxy Interface:** Proxy.total_flows Proxy.aged_flows Proxy.exception_packets Proxy.drop_stats_flow_queue_limit_exceeded Proxy.drop_stats_flow_table_full Proxy.drop_stats_vlan_fwd_enq		

Type of VNF	Example Usage	Example KPIs and Metrics Networking Virtualized Resources	Example KPIs and Metrics for Application Services	Example KPIs and Metrics for Virtualized Storage
		Proxy.drop_stats_vlan_fwd_tx Proxy.flow_rate_active_flows Proxy.flow_rate_added_flows Proxy.flow_rate_deleted_flows Proxy.Interface.in_pkts_ewm Proxy.Interface.out_pkts_ewm Proxy.phy_band_in_bps Proxy.phy_band_out_bps Proxy.in_bps_ewm Proxy.out_bps_ewm **Virtual Machine Interface (tap interface):** Proxy.Virtual_Machine_Interface.drop_stats Proxy.Virtual_Machine_Interface.in_bytes Proxy.Virtual_Machine_Interface.in_pkts Proxy.Virtual_Machine_Interface.out_bytes Proxy.Virtual_Machine_Interface.out_pkts Proxy.Virtual_Machine_Interface.in_pkts_ewm Proxy.Virtual_Machine_Interface.out_pkts_ewm Proxy.Virtual_Machine_Interface.active_flows_ewm Proxy.Virtual_Machine_Interface.added_flows_ewm Proxy.Virtual_Machine_Interface.deleted_flows_ewm Proxy.Virtual_Machine_Interface.flow_rate Proxy.Virtual_Machine_Interface.if_stats Proxy.Virtual_Machine_Interface.if_in_pkts_ewm Proxy.Virtual_Machine_Interface.if_out_pkts_ewm **Virtual Network Interface:** Proxy.VirtualNetwork.vn_stats Proxy.VirtualNetwork.in_bytes Proxy.VirtualNetwork.in_tpkts Proxy.VirtualNetwork.out_bytes Proxy.VirtualNetwork.out_tpkts		

3. NFV Infrastructure Resources

The NFV Infrastructure consists of physical and virtual compute, storage and networking resources that VNFs will utilize. Utilization and performance metrics can be collected on these components to support performance monitoring, and to address capacity planning. This document contains a list of metrics that may be used in performing these tasks.

For the purposes of Platform Service Assurance, it's important to distinguish between metrics and events as well as how they are measured (from a timing perspective).

A **metric** is a (standard) definition of a quantity describing the performance and/or reliability of a monitored function, which has an intended utility and is carefully specified to convey the exact meaning of the measured value. The measured value of a metric is produced in an assessment of a monitored function according to a method of measurement. For example, the number of dropped packets for a networking interface is a metric.

An **event** is defined as an important state change in a monitored function. The monitoring system is notified that an event has occurred using a message with a standard format. The event notification describes the significant aspects of the event, such as the name and ID of the monitored function, the type of event, and the time the event occurred. For example, an event notification would take place if the link status of a networking device on a compute node suddenly changes from up to down on a node hosting VNFs in an NFV deployment.

The metrics listed may be categorized as either generic, conditional, or component specific.

Generic metrics are intended to be independent of a specific implementation and should be available on all implementations.

Conditional metrics are those metrics that are related to the support of a particular functional capability. Conditional metrics should be available on implementations that support the related functional capability. An example condition based on a functional capability might be whether a Layer 2 network interface function supports MPLS or not.

If MPLS is supported, then the metrics that are conditional on MPLS support should be provided.

Component-specific metrics are dependent on the use of specific hardware or software components. They are available if the specific components are used. These could be implementation-specific metrics that are not covered in the first two categories.

In the current industry ecosystem, differences between the metrics provided by different software and hardware components may present difficulties when managing deployments that contain a mixture of different components. Improving the coverage and granularity of generic and conditional metrics can help promote multivendor deployments in the NFV area.

The subsequent sections list various metrics across the compute, networking, and storage domains. The metrics definitions are independent of the location in which they are collected. The metrics can be collected from the guest Virtual Machine (VM), or multiple components residing in the host. The relevance of location for each metric will be elaborated upon in the section tables.

Compute

CPU

"The Central Processing Unit (CPU) is an essential component of every coherent compute domain. Each CPU is a limited resource in terms of the instructions per second it can execute. It is valuable to monitor the utilization of the CPU resource to fulfill the goals of maintaining continued and efficient operations, and for troubleshooting abnormal behavior to find root causes. For many uses, it is helpful to categorize the CPU's execution time into multiple execution contexts, such as system and user contexts. A compute node may include additional processors beyond the main CPU; the metrics specified in this case can also be used to measure and report the usage of such processors.

VNFs also have a view of CPU resources in terms of execution time they have used during a measurement interval. However, the configured instantiation of the VNF determines how to map its view of virtual processor resource usage to actual hardware CPU resources available and used. For example, a VNF's processes may be pinned to one or more CPU cores, or the VNF may be sharing the resources of many CPU cores with other VNFs." [1]

CPU metrics are relevant in both the "host" and the "guest" domains. CPU metrics collected at the host provide the current utilization of host CPUs and hyper threads (vCPUs).

For guests that use poll mode drivers and Data Plane Development Kit (DPDK) hardware acceleration, the CPUs on the host are always utilized 100%. Hence, CPU utilization should always be collected from the guest as well as the host. Both these metrics should be correlated for accurate measurement. In addition, metrics such as PMU counts in the host that records IPC, memory bandwidth, etc., also provide useful information.

Units and Instance	**Description**	**Comment**	**Collection Location**
Idle, percent/ nanoseconds	Time CPU spends idle.	Can be per CPU/aggregate across all the CPUs. For more info, please see: http://man7.org/linux/man-pages/man1/top.1.html http://blog.scoutapp.com/articles/2015/02/24/understanding-linuxs-cpu-stats Note that jiffies operate on a variable time base, HZ. The default value of HZ should be used (100), yielding a jiffy value of 0.01 seconds [time (7)]. Also, the actual number of jiffies in each second is subject to system factors, such as use of virtualization. Thus, the percent calculation based on jiffies will nominally sum to 100% plus or minus error.	• Host Machine. • Guest Virtual Machine. • Network Plugins using DPDK (e.g., OpenContrail). • Any other DPDK based application running in user space.
Nice, percent/ nanoseconds	Time the CPU spent running user space processes that have been niced. The priority level a user space process can be tweaked by adjusting its niceness.		
Interrupt, percent/ nanoseconds	Time the CPU has spent servicing interrupts.		
Softirq,percent/ nanoseconds	Time spent handling interrupts that are synthesized, and almost as important as Hardware interrupts (above). "In current kernels, there are 10 softirq vectors defined; two for tasklet processing, two for networking, two for the block layer, two for timers, and one each for the scheduler and read-copy-update processing. The kernel maintains a per-CPU bitmask indicating which softirqs need processing at any given time." See: https://lwn.net/Articles/520076/		
Steal, percent/ nanoseconds	CPU steal is a measure of the fraction of time that a machine is in a state of "involuntary wait." It is time for which the kernel cannot otherwise account in one of the traditional classifications like user, system, or idle. It is time that went missing, from the perspective of the kernel. http://www.stackdriver.com/understanding-cpu-steal-experiment/		

Units and Instance	Description	Comment	Collection Location
System, percent/ nanoseconds	Time that the CPU spent running the kernel.		
User, percent/ nanoseconds	Time CPU spends running un-niced user space processes.		
Wait, percent/ nanoseconds	The time the CPU spends idle while waiting for an I/O operation to complete.		

Interface

"Network Interfaces exist at many points on the communication paths through the NFVI. It is valuable to monitor the traffic load of arbitrary interfaces — both the physical interfaces with hardware resources and their virtual counterparts. Additional information can assist troubleshooting and resolution, so interfaces also indicate problems with attempted transmission or reception of packets". [1]

In the life of a packet, after it enters the host, several interfaces (physical and virtual) are encountered. The generic interface metrics apply to all the interface types, whereas some conditional metrics apply only to certain implementations.

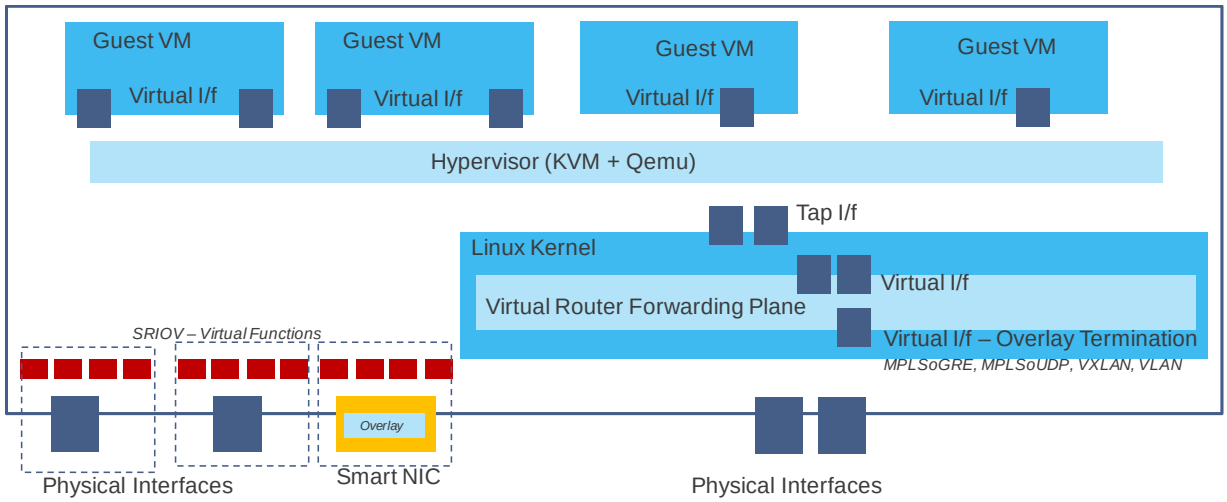


Figure 1: Location of Network Interfaces in a Host

The figure above shows a simplified view of a host with different interfaces. Physical interfaces connect to the underlay switch fabric. Underlay infrastructure in data centers

are typical leaf-spine Clos networks. They provide basic connectivity between compute and DC infrastructure. Since the physical interfaces on the host are the entry and exit points to the host, metrics collected on these interfaces give critical information for the compute.

Network plugins such as OpenContrail [2], OpenvSwitch [3] or Linux bridges may run inside the Linux kernel. These plugins have southbound (towards underlay) and northbound (towards guest virtual machines) virtual interfaces. The interface facing the underlay may support different types of overlay technologies such as VXLAN, MPLSoGRE, MPLSoUDP. Overlays are tunnels riding on top of the underlay infrastructure. Overlays enable the applications to communicate with each other. Depending on the overlay method, certain conditional metrics can be collected on these virtual interfaces. Similarly, the tap interfaces on the host give important statistics for packets on the path between the kernel and the guest VM.

A guest VM might support different types of network interfaces. Guest virtual machines (VMs) can run Ethernet and do Layer 2 operations. Or, guest VMs can support IPv4, IPv6 on their virtual interfaces. Some carrier-grade virtualized routers acting as guest VMs (VPE, VBNG etc.) can even terminate MPLS on these interfaces. The metrics for IP/MPLS become conditional on the guest VM.

Lastly, there are Network Interface Controllers (NICs) that support hardware acceleration, such as Single Root IO Virtualization (SR-IOV). One can carve a physical NIC into multiple slices called Virtual functions (VFs). Guest VMs can have direct access to these slices (VFs). As such, to collect per-VF metrics is also important for service assurance.

Generic (applies to all types of Interfaces: Physical, Tap, Virtual Interface, etc.)

Units and Instance	Description	Comment	Collection Location
If_dropped, in	The total number of received dropped packets.		• Physical interface of the host. • Virtual interface on Network Plugins (OpenvSwitch, OpenContrail, bridge interfaces etc.). • Tap interfaces between the network plugin and the guest. • Guest virtual interfaces.
If_errors, in	The total number of received error packets.	http://www.onlamp.com/pub/a/linux/2000/11/16/LinuxAdmin.html	
If octets, in	The total number of received bytes.		
If packets, in	The total number of received packets.		
If_dropped, out	The total number of transmit packets dropped.		
If_errors, out	The total number of transmit error packets. (This is the total of error conditions encountered when attempting to transmit a packet. The code here explains the possibilities, but this code is no longer present in /net/core/dev.c master at present – it appears to have moved to /net/core/net-procfs.c.).		
If_octets, out	The total number of bytes transmitted.		
If_packets, out	The total number of transmitted packets.		
If_multicast_packets, in			
If_multicast_packets, out			
If_broadcast_packets, in			
If_broadcast_packets,out			
If_Frame_Alignment_errors			
If_rx_fragmented_errors			
If_rx_overrun_errors			
If_invalid_ether_type	Invalid ether type field.		
If_Invalid_If	Packet arrived on Invalid interface.		
If_ds_invalid_protocol	Invalid Protocol field.		
If_phy_bandwidth_in_bps	Interface Bandwidth ingress bits/sec.		
If_phy_bandwidth_out_bps	Interface Bandwidth Egress bits/sec.		

Conditional

Units and Instance	Description	Comment
If_invalid_label	Invalid label received on Interface.	Any implementation that supports MPLS or its derivatives (such as MPLSoGRE/MPLSoUDP).
If_labeled_packets, in	Number of MPLS labeled packets received.	
If_labeled_packets, out	Number of MPLS labeled packets transmitted.	
If_invalid_vnid	Received invalid VXLAN ID.	Any implementation that supports VXLAN and act as a VTEP (VXLAN Tunnel End Point).
If_invalid_vlan	Received Invalid VLAN (Not in range).	Any implementation that supports VLAN tags. Includes QinQ VLANs.
If_untagged_packets, in	Number of untagged packets received on a VLAN enabled interface.	
If_invalid_mcast_source	Multicast RPF check failed.	Any implementation that supports processing IP Multicast frames (Note: This is different to the metric if they are multicast packets. An implementation must be able to display multicast/broadcast statistics. However, to identify and perform RPF check, interface must have IP multicast state information).

VNFs and plugins will inherit all "generic" metrics, and support some or all the specific metrics. They may add more interface metrics that are unique to that application.

Memory

"Memory is a key resource of NFVI. It is valuable to monitor the utilization and management of system and user memory and their virtual counterparts. This information can assist troubleshooting and resolution." [1]

Units and Instance	Description	Comment
Buffered	The amount, in kibibytes, of temporary storage for raw disk blocks.	https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s2-proc-meminfo.html
Cached	The amount of physical RAM, in kibibytes, left unused by the system.	
Free	The amount of physical RAM, in kibibytes, left unused by the system.	
slab_recl	The part of Slab that can be reclaimed, such as caches.	Slab — The total amount of memory, in kibibytes, used by the kernel to cache data structures for its own use.

Units and Instance	Description	Comment
slab_unrecl	The part of Slab that cannot be reclaimed even when lacking memory.	https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s2-proc-meminfo.html
Used	mem_used = mem_total - (mem_free + mem_buffered + mem_cached + mem_slab_total).	https://github.com/collectd/collectd/blob/master/src/memory.c#L349

Disk

The disk plugin collects performance statistics for hard disks for load, Input/Out Per Second (IOPS), pending operations and traffic.

Units and Instance	Description	Comment
disk_io_time, io_time	Time spent doing I/Os (ms). You can treat this metric as a device load percentage (value of 1 sec time spent matches 100% of load).	
disk_io_time, weighted_io_time	Measure of both I/O completion time and the backlog that may be accumulating.	
Disk_merged, read	The number of operations that could be merged into other, already queued operations – i.e., one physical disk access served two or more logical operations. Of course, the higher that number, the better.	
Disk_merged, write	The number of operations that could be merged into other, already-queued operations – i.e., one physical disk access served two or more logical operations. Of course, the higher that number, the better.	
Disk_octets, read	The number of octets read from a disk or partition.	
Disk_octets, write	The number of octets written to a disk or partition.	
Disk_ops, read	The number of read operations issued to the disk.	
Disk_ops, write	The number of write operations issued to the disk.	
Disk_time, read	The average time an I/O-operation took to complete. Note from collectd: Since this is a little messy to calculate, take the actual values with a grain of salt.	
Disk_time, write	The average time an I/O-operation took to complete. Note from collectd: Since this is a little messy to calculate, take the actual values with a grain of salt.	https://collectd.org/wiki/index.php/Plugin:Disk
Pending_operations	Shows queue size of pending I/O operations.	http://lxr.free-electrons.com/source/include/uapi/linux/if_link.h#L43

Ping

The ping plugin measures round-trip network latency using ICMP "echo requests" or ping.

Units and Instance	Description	Comment
Ping	Network latency is measured as a round-trip time in milliseconds. An ICMP "echo request" is sent to a host and the time needed for its echo-reply to arrive is measured.	Latency
Ping_droprate	droprate = ((double) (pkg_sent - pkg_recv)) / ((double) pkg_sent);	https://github.com/collectd/collectd/blob/master/src/ping.c#L703
Ping_stddev	if pkg_recv > 1 latency_stddev = sqrt (((((double) pkg_recv) * latency_squared) - (latency_total * latency_total)) / ((double) (pkg_recv * (pkg_recv - 1))));	https://github.com/collectd/collectd/blob/master/src/ping.c#L698 pkg_recv = # of echo-reply messages receivedlatency_squared = latency * latency (for a received echo-reply message) latency_total = the total latency for received echo-reply messages

Load

The load plugin measures the system load defined as the number of runnable tasks in the run-queue and is reported over short term (averaged over 1 minute), medium term (last five minutes) and long term (last fifteen minutes).

Units and Instance	Description	Comment
Shortterm	Load average figures giving the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over one minute.	http://man7.org/linux/man-pages/man5/proc.5.html
Ping_droprate	Measured CPU and IO utilization for one minute using /proc/loadavg.	https://github.com/collectd/collectd/blob/master/src/load.c
Midterm	Load average figures giving the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 5 minutes.	

Open vSwitch (OVS) Stats

The OVS stats plug-in collects statistics related to OVS-connected interfaces, ports, and bridges and uses the OVS database to get those statistics.

Units and Instance	Description	Comment
If_collisions	Number of collisions.	Per interface.
if_packets rx: 1_to_64_packets	The total number of packets (including bad packets) received that were 64 octets in length (excluding framing bits but including FCS octets).	Supported in ovs v2.6+ and dpdk ports only.
if_packets rx: 65_to_127_packets	The total number of packets (including bad packets) received that were between 128 and 255 octets in length inclusive (excluding framing bits but including FCS octets).	Supported in ovs v2.6+ and dpdk ports only.
if_packets rx: 128_to_255_packets	The total number of packets (including bad packets) received that were between 256 and 511 octets in length inclusive (excluding framing bits but including FCS octets).	Supported in ovs v2.6+ and dpdk ports only.
if_packets rx: 256_to_511_packets	The total number of packets (including bad packets) received that were between 512 and 1023 octets in length inclusive (excluding framing bits but including FCS octets).	Supported in ovs v2.6+ and dpdk ports only.
if_packets rx: 512_to_1023_packets	The total number of packets (including bad packets) received that were between 1024 and 1518 octets in length inclusive (excluding framing bits but including FCS octets).	Supported in ovs v2.6+ and dpdk ports only.
if_packets rx: 1024_to_1522_packets	The total number of packets (including bad packets) received that were between 1523 and max octets in length inclusive (excluding framing bits but including FCS octets).	Supported in ovs v2.6+ and dpdk ports only.
if_packets rx: 1523_to_max_packets	The total number of packets (including bad packets) received that were between 1523 and max octets in length inclusive (excluding framing bits but including FCS octets).	Supported in ovs v2.6+ and dpdk ports only.
if_packets tx: 1_to_64_packets	The total number of packets transmitted that were 64 octets in length.	Supported in ovs v2.6+ and dpdk ports only.
if_packets tx: 65_to_127_packets	The total number of packets received that were between 65 and 127 octets in length inclusive.	Supported in ovs v2.6+ and dpdk ports only.
if_packets tx: 128_to_255_packets	The total number of packets received that were between 128 and 255 octets in length inclusive.	Supported in ovs v2.6+ and dpdk ports only.
if_packets tx: 256_to_511_packets	The total number of packets received that were between 256 and 511 octets in length inclusive.	Supported in ovs v2.6+ and dpdk ports only.
if_packets tx: 512_to_1023_packets	The total number of packets received that were between 512 and 1023 octets in length inclusive.	Supported in ovs v2.6+ and dpdk ports only.
if_packets tx: 1024_to_1522_packets	The total number of packets received that were between 1024 and 1518 octets in length inclusive.	Supported in ovs v2.6+ and dpdk ports only.
if_packets tx: 1523_to_max_packets	The total number of packets received that were between 1523 and max octets in length inclusive.	Supported in ovs v2.6+ and dpdk ports only.

Units and Instance	Description	Comment
If_rx_errors: rx_undersized_errors	The total number of packets received that were less than 64 octets long (excluding framing bits but including FCS octets) and were otherwise well formed.	Supported in ovs v2.6+ and dpdk ports only.
If_rx_errors: rx_oversize_errors	The total number of packets received that were longer than max octets (excluding framing bits but including FCS octets) and were otherwise well formed.	Supported in ovs v2.6+ and dpdk ports only
If_rx_errors: rx_fragmented_errors	The total number of packets received that were less than 64 octets in length (excluding framing bits but including FCS octets) and had either a bad Frame Check Sequence (FCS) with an integral number of octets (FCS Error) or a bad FCS with a non-integral number of octets (Alignment Error). **Note**: It is entirely normal for rx_fragmented_errors to increment, due to the fact it counts both runts (which are normal occurrences due to collisions) and noise hits.	Supported in ovs v2.6+ and dpdk ports only
If_rx_errors: Rx_jabber_errors	The total number of jabber packets received that had either a bad Frame Check Sequence (FCS) with an integral number of octets (FCS Error) or a bad FCS with a non-integral number of octets (Alignment Error).	Supported in ovs v2.6+ and dpdk ports only

Opencontrail Plugin

Units and Instance	Description
vRouter.total_flows	vRouter Total number of flows on the vrouter
vRouter.aged_flows	vRouter Aged flows on the vrouter
vRouter.exception_packets	vRouter Exception packets (Control) on the vrouter
vRouter.drop_stats_flow_queue_limit_exceeded	vRouter Number of packets dropped due to flow queue limit exceeded
vRouter.drop_stats_flow_table_full	vRouter Number of packets dropped due to flow table being full
vRouter.drop_stats_vlan_fwd_enq	vRouter Number of packets dropped for VLAN forwarding enqueue
vRouter.drop_stats_vlan_fwd_tx	vRouter Number of packets dropped for VLAN forwarding transmit
vRouter.flow_export_drops	vRouter Number of packets dropped for flow export
vRouter.flow_export_sampling_drops	vRouter Number of packets dropped for flow export sampling
vRouter.flow_rate_active_flows	vRouter Active flow rate

Units and Instance	Description
vRouter.flow_rate_added_flows	vRouter Added flow rate
vRouter.flow_rate_deleted_flows	vRouter Deleted flow rate
vRouter.Virtual_Machine_Interface.active_flows_ewm	vRouter per VMI active flows EWM
vRouter.Virtual_Machine_Interface.added_flows_ewm	vRouter per VMI added flows EWM
vRouter.Virtual_Machine_Interface.deleted_flows_ewm	vRouter per VMI deleted flows EWM
vRouter.Virtual_Machine_Interface.flow_rate	vRouter per VMI flow rate
vRouter.VirtualNetwork.vn_stats	vRouter per Virtual Network stats
vRouter.VirtualNetwork.in_bytes	Router per Virtual Network ingress bytes
vRouter.VirtualNetwork.in_tpkts	Router per Virtual Network ingress packets
vRouter.VirtualNetwork.out_bytes	vRouter per Virtual Network egress bytes
vRouter.VirtualNetwork.out_tpkts	vRouter per Virtual Network egress packets

Hugepages Plugin

Virtual memory makes it easy for several processes to share memory [4]. *"Each process has its own virtual address space, which is mapped to physical memory by the operating system." [5]* The process views the virtual memory address space as a contiguous/linear address space, but in reality the virtual addresses need to be mapped to physical addresses. This is typically done by the Memory Management Unit (MMU) on the CPU.

"*There are two ways to enable the system to manage large amounts of memory:*

- *Increase the number of page table entries in the hardware memory management unit*
- *Increase the page size*

The first method is expensive, since the hardware memory management unit in a modern processor only supports hundreds or thousands of page table entries. Additionally, hardware and memory management algorithms that work well with thousands of pages (megabytes of memory) may have difficulty performing well with millions (or even billions) of pages. This results in performance issues: When an application needs to use more memory pages than the memory management unit supports, the system falls back to slower, software-based memory management, which causes the entire system to run more slowly.

Huge pages are blocks of memory that come in 2MB and 1GB sizes. The page tables used by the 2MB pages are suitable for managing multiple gigabytes of memory, whereas the page tables of 1GB pages are best for scaling to terabytes of memory." [6]

The purpose of this feature is to report the number of free and used huge pages on a system for intelligent workload placement. Huge pages can be used by applications to improve performance such as the memory backing for VMs. They are allocated per socket across a platform through configuration files or systemctl. Huge pages help improve the performance of applications by reducing the TLB lookups as the page size is increased from 4KB to 2MB or 1GB.

Units and Instance	Description	Comment
bytes	used	Number of used hugepages in bytes.
bytes	free	Number of free hugepages in bytes.
vmpage_number	used	Number of used hugepages in numbers.
vmpage_number	free	Number of free hugepages in numbers.
percent	used	Number of used hugepages in percent.
percent	free	Number of free hugepages in percent.

Processes Plugin

The processes plugin collects the number of processes, grouped by their state (e.g., running, sleeping, zombies).

Units and Instance	Description	Comment
Fork_rate	The number of threads created since the last reboot.	The information comes mainly from /proc/PID/status, /proc/PID/psinfo and /proc/PID/usage.
Ps_state blocked	The number of processes in a blocked state.	https://collectd.org/wiki/index.php/Plugin:Processes
Ps_state paging	The number of processes in a paging state.	http://man7.org/linux/man-pages/man5/proc.5.html
Ps_state running	The number of processes in a running state.	
Ps_state sleeping	The number of processes in a sleeping state.	
Ps_state stopped	The number of processes in a stopped state.	
Ps_state zombies	The number of processes in a Zombie state.	

Libvirt Plugin

Libvirt is an open source project, which aims to provide a simple and convenient way of managing VMs created by various different hypervisors in a unified manner. It is an abstraction layer that provides a common application programming interface (API) for numerous functionalities implemented by hypervisors. Even if particular functionality is at the moment implemented only by a single hypervisor, it has to be exposed in a generic way to enable support of multiple hypervisors in the future should they choose to implement the same feature. It eliminates the need to learn about hypervisor-specific tools — a desirable characteristic and significant advantage in cloud-based environments. Major libvirt features include:

- VM management — domain lifecycle management, device hotplug operations.
- Remote machine support — support for multiple network transports for remote connection.
- Storage management — creation of file images, mounting NFS shares.
- Network management — physical and logical network interfaces management.
- Virtual NAT and route-based networking — management and creation of virtual networks.

Libvirt consists of three software components:

- API library.
- Daemon – **libvirtd.**
- Command line utility – **virsh.**

Libvirt is based on client-server architecture, which requires a daemon to be installed only on machines that will host virtualized guests. For the purpose of controlling domains remotely, libvirt uses a custom protocol to establish connections with remote libvirt instances. It should be noted that domains can also be controlled locally where client and server can be run on the same physical host.

The virt plugin collects statistics by using virtualization API. Metrics are gathered directly from the hypervisor on a host system, which means that collectd does not have to be installed and configured on a guest system.

Note: Certain metrics and events have a requirement on a minimal libvirt API version. For more information, please see collectd virt plugin documentation.

Units and Instance	**Description**	**Comment**
Disk_octets DISK	Number of read/write bytes as unsigned long long.	
Disk_ops DISK	Number of read/write requests .	
Disk_time flush_DISK	Total time spend on cache reads/writes in nano-seconds.	
If-dropped INTERFACE	Packets dropped on rx/tx as unsigned long long.	
If-errors INTERFACE	Rx/tx errors as unsigned long long.	
If-octets INTERFACE	Bytes received/transmitted as unsigned long long.	
If-packets INTERFACE	Packets received/transmitted as unsigned long long.	
Memory actual_balloon	Resident Set Size of the process running the domain. This value is in kB.	https://libvirt.org/html/libvirt-libvirt-domain.html#virDomainMemoryStatStruct
Memory rss	How much the balloon can be inflated without pushing the guest system to swap, corresponds to 'Available' in /proc/meminfo.	
Memory swap_in	The total amount of memory written out to swap space (in kB).	
Memory total	the memory in KBytes used by the domain.	
Virt_cpu_total	the CPU time used in nanoseconds.	
Virt_cpu VCPU_NR	the CPU time used in nanoseconds per CPU.	
Cpu_affinity vcpu_NR-cpu_NR	pinning of domain VCPUs to host physical CPUs.	Value stored is a boolean.
Job status *	Information about progress of a background/completed job on a domain.	Number of metrics depend on job type. Check API documentation for more information: virDomainGetJobStats.
Disk_error DISK_NAME	Disk error code.	Metric is not dispatched for disk with no errors.
Percent virt_cpu_total	CPU utilization in percentage per domain.	

Units and Instance	Description	Comment
Perf *	Performance monitoring events.	Number of metrics depends on libvirt API version. Following perf metric are available in libvirt API version 2.4. To collectd perf metric they must be enabled in domain and supported by the platform.
perf_cmt	Usage of l3 cache in bytes by applications running on the platform.	
perf_ mbmt	Total system bandwidth from one level of cache.	
perf_ mbml	Bandwidth of memory traffic for a memory controller.	
perf_cpu_cycles	The count of CPU cycles (total/elapsed).	
perf_instructions	The count of instructions by applications running on the platform.	
perf_cache_references	The count of cache hits by applications running on the platform.	
perf_cache_misses	The count of cache misses by applications running on the platform.	
Ps_cputime	Physical user/system CPU time consumed by the hypervisor.	
Total_requests flush-DISK	Total flush requests of the block device.	
Total_requests flush-DISK	Total time spend on cache flushing in milliseconds.	

Intel® RDT Plugin

Cache Monitoring Technology (CMT), Memory Bandwidth Monitoring (MBM), Cache Allocation Technology (CAT) and Code and Data Prioritization (CDP) Technology provide the hardware framework for monitoring and controlling the utilization of shared resources, such as last-level cache and memory bandwidth. These technologies comprise Intel® Resource Director Technology (RDT). As multithreaded and multicore platform architectures emerge — running workloads in single-threaded, multithreaded, or complex virtual machine environments — the last-level cache and memory bandwidth are key resources to manage. Intel introduces CMT, MBM, CAT, and CDP to manage these various workloads across shared resources.

CMT and MBM

CMT and MBM are new features that allow an operating system (OS) or Hypervisor/virtual machine monitor (VMM) to determine the usage of cache and

memory bandwidth by applications running on the platform. CMT and MBM can be used to do the following:

- Detect if the platform supports these monitoring capabilities (via CPUID).
- Allow an OS or VMM to assign an ID for each application or VMs that are scheduled to run on a core. This ID is called the Resource Monitoring ID (RMID).
- Monitor cache occupancy and memory bandwidth on a per-RMID basis.
- Read LLC occupancy and memory bandwidth for a given RMID at any time, for an OS or VMM.

CAT and CDP

CAT and CDP are new features that allow an OS or Hypervisor/VMM to control allocation of CPUs shared last level cache. Once CAT or CDP is configured, the processor allows access to portions of the cache according to the established class of service (COS). The processor obeys the COS rules when it runs an application thread or application process. This can be accomplished by performing these steps:

- Determine if the CPU supports the CAT and CDP feature.
- Configure the COS to define the amount of resources (cache space) available. This configuration resides at the processor level and is common to all logical processors.
- Associate each logical processor with an available COS.
- Run the application on the logical processor that uses the desired COS.

The **intel_rdt** plugin collects information provided by monitoring features of Intel Resource Director Technology (Intel® RDT): Cache Monitoring Technology (CMT), and Memory Bandwidth Monitoring (MBM).

Units and Instance	Description	Comment
Ipc	Number of instructions per clock per core group.	per core group
Memory_bandwidth local	Local Memory Bandwidth utilization.	
Memory_bandwidth remote	Remote Memory Bandwidth utilization.	
Bytes llc	Last Level Cache occupancy.	

DPDKStats Plugin

DPDK provides a number of features to expose statistics and events relevant to DPDK interfaces and packet processing cores. A collectd read plugin should take advantage of these features to expose the key telco KPI statistics that provide a monitor for the performance of DPDK interfaces.

In terms of stats, DPDK provides two functions through which to access packet-forwarding statistics: (1) generic statistics API, and (2) extended statistics (xstats) API. The first API returns generic aggregate packet-forwarding statistics that are common across all NIC drivers such as: packets received, packets sent, bytes received, bytes sent, etc. Although valuable information, modern NICs collect and expose much more detailed information, which cannot be accessed through the generic statistics interface. The xstats API was designed to transparently expose this information. The extended statistics API allows each individual NIC Driver to expose a unique set of statistics. The xstats API also exposes the generic stats API metrics. The collectd plugin will take advantage of the xstats API to expose all the DPDK interface metrics. These metrics are listed below for the ixgbe driver.

Units and Instance	**Description**	**Comment**
Derive rx_l3_l4_xsum_error	Number of receive IPv4, TCP, UDP or SCTP XSUM errors.	
Errors flow_director_filter_add_errors	Number of failed added filters.	Compatible with DPDK 16.04, 16.07 (based on ixgbe, vhost support will be enabled in DPDK 16.11).
Errors flow_director_filter_remove_errors	Number of failed removed filters.	
Errors mac_local_errors	Number of faults in the local MAC.	
Errors mac_remote_errors	Number of faults in the remote MAC.	
If_rx_dropped rx_fcoe_dropped	Number of Rx packets dropped due to lack of descriptors.	
If_rx_dropped rx_mac_short_packet_dropped	Number of MAC short packet discard packets received.	

Units and Instance	Description	Comment
If_rx_dropped rx_management_dropped	Number of management packets dropped. This register counts the total number of packets received that pass the management filters and then are dropped because the management receive FIFO is full. Management packets include any packet directed to the manageability console (such as RMCP and ARP packets).	
If_rx_dropped rx_priorityX_dropped	Number of dropped packets received per UP.	Where X is 0 to 7.
If_rx_errors rx_crc_errors	Counts the number of receive packets with CRC errors. In order for a packet to be counted in this register, it must be 64 bytes or greater (from <Destination Address> through <CRC>, inclusively) in length.	
If_rx_errors rx_errors	Number of errors received.	
If_rx_errors rx_fcoe_crc_errors	FC CRC Count.	
If_rx_errors	Count the number of packets with good Ethernet CRC and bad FC CRC.	
If_rx_errors rx_fcoe_mbuf_allocation_errors	Number of fcoe Rx packets dropped due to lack of descriptors.	
If_rx_errors rx_fcoe_no_direct_data_placement		Number of metrics depend on job type. Check API documentation for more information: virDomainGetJobStats.
If_rx_errors rx_fcoe_no_direct_data_placement_ext_buff		Metric is not dispatched for disk with no errors.
If_rx_errors rx_fragment_errors	Number of receive fragment errors (frame shorted than 64 bytes from <Destination Address> through <CRC>, inclusively) that have bad CRC (this is slightly different from the Receive Undersize Count register).	
If_rx_errors rx_illegal_byte_errors	Counts the number of receive packets with illegal bytes errors (such as there is an illegal symbol in the packet).	Number of metrics depends on libvirt API version. Following perf metric are available in libvirt API version 2.4. To collectd perf metric they must be enabled in domain and supported by the platform.

Units and Instance	Description	Comment
If_rx_errors rx_jabber_errors	Number of receive jabber errors. This register counts the number of received packets that are greater than maximum size and have bad CRC (this is slightly different from the Receive Oversize Count register). The packets length is counted from <Destination Address> through <CRC>, inclusively.	
If_rx_packets rx_priorityX_xoff_packets	Number of XOFF packets received per UP.	Where X is 0 to 7.
If_rx_packets rx_priorityX_xon_packets	Number of XON packets received per UP.	Where X is 0 to 7.
If_rx_packets rx_q0_packets	Number of packets received for the queue.	Per queue.
If_rx_packets rx_size_1024_to_max_packets	Number of packets received that are 1024-max bytes in length (from <Destination Address> through <CRC>, inclusively). This register does not include received flow control packets. The maximum is dependent on the current receiver configuration and the type of packet being received. If a packet is counted in receive oversized count, it is not counted in this register. Due to changes in the standard for maximum frame size for VLAN tagged frames in 802.3, packets can have a maximum length of 1522 bytes.	
If_rx_packets rx_size_128_to_255_packets	Number of packets received that are 128-255 bytes in length (from <Destination Address> through <CRC>, inclusively).	
If_rx_packets rx_size_256_to_511_packets	Number of packets received that are 256-511 bytes in length (from <Destination Address> through <CRC>, inclusively).	
If_rx_packets rx_size_512_to_1023_packets	Number of packets received that are 512-1023 bytes in length (from <Destination Address> through <CRC>, inclusively).	
If_rx_packets rx_size_64_packets	Number of good packets received that are 64 bytes in length (from <Destination Address> through <CRC>, inclusively).	

Units and Instance	Description	Comment
If_rx_packets rx_size_65_to_127_packets	Number of packets received that are 65-127 bytes in length (from <Destination Address> through <CRC>, inclusively).	
If_rx_packets rx_total_missed_packets	The total number of rx missed packets, that is a packet that was correctly received by the NIC but because it was out of descriptors and internal memory, the packet had to be dropped by the NIC itself.	
If_rx_packets rx_total_packets	Number of all packets received. This register counts the total number of all packets received. All packets received are counted in this register, regardless of their length, whether they are erred, but excluding flow control packets.	
If_rx_packets rx_xoff_packets	Number of XOFF packets received. Sticks to 0xFFFF. XOFF packets can use the global address or the station address. This register counts any XOFF packet whether it is a legacy XOFF or a priority XOFF. Each XOFF packet is counted once even if it is designated to a few priorities. If a priority FC packet contains both XOFF and XON, only this counter is incremented.	
If_rx_packets rx_xon_packets	Number of XON packets received. XON packets can use the global address, or the station address. This register counts any XON packet whether it is a legacy XON or a priority XON. Each XON packet is counted once even if it is designated to a few priorities. If a priority FC packet contains both XOFF and XON, only the LXOFFRXCNT counter is incremented.	
if_tx_errors tx_errors	Total number of TX error packets.	
if_tx_errors tx_fcoe_bytes	Number of fcoe bytes transmitted.	
if_tx_octets tx_good_bytes	Counter of successfully transmitted octets. This register includes transmitted bytes in a packet from the <Destination Address> field through the <CRC> field, inclusively.	

Units and Instance	Description	Comment
if_tx_octets tx_q0_bytes	Number of bytes transmitted by the queue.	Per queue.
If_tx_packets tx_broadcast_packets	Number of broadcast packets transmitted count. This register counts all packets, including standard packets, secure packets, FC packets and manageability packets.	
If_tx_packets tx_fcoe_packets	Number of fcoe packets transmitted.	
If_tx_packets tx_flow_control_xoff_packets	Link XOFF Transmitted Count.	
If_tx_packets tx_flow_control_xon_packets	Link XON Transmitted Count.	
If_tx_packets tx_good_packets	Number of good packets transmitted.	
If_tx_packets tx_management_packets	Number of management packets transmitted.	
If_tx_packets tx_multicast_packets	Number of multicast packets transmitted. This register counts the number of multicast packets transmitted. This register counts all packets, including standard packets, secure packets, FC packets and manageability packets.	
If_tx_packets tx_priorityX_xoff_packets	Number of XOFF packets transmitted per UP.	Where X is 0 to 7.
If_tx_packets tx_priorityX_xon_packets	Number of XON packets transmitted per UP.	Where X is 0 to 7.
If_tx_packets tx_q0_packets	Number of packets transmitted for the queue. A packet is considered as transmitted if it is forwarded to the MAC unit for transmission to the network and/or is accepted by the internal Tx to Rx switch enablement logic. Packets dropped due to anti-spoofing filtering or VLAN tag validation (as described in Section 7.10.3.9.2) are not counted.	Per queue.
If_tx_packets tx_size_1024_to_max_packets	Number of packets transmitted that are 1024 or more bytes in length (from <Destination Address> through <CRC>, inclusively). This register counts all packets, including standard packets, secure packets, and manageability packets.	

Units and Instance	Description	Comment
If_tx_packets tx_size_128_to_255_packets	Number of packets transmitted that are 128-255 bytes in length (from <Destination Address> through <CRC>, inclusively). This register counts all packets, including standard packets, secure packets, and manageability packets.	
If_tx_packets tx_size_256_to_511_packets	Number of packets transmitted that are 256-511 bytes in length (from <Destination Address> through <CRC>, inclusively). This register counts all packets, including standard packets, secure packets, and manageability packets.	
If_tx_packets tx_size_512_to_1023_packets	Number of packets transmitted that are 512-1023 bytes in length (from <Destination Address> through <CRC>, inclusively). This register counts all packets, including standard packets, secure packets, and manageability packets.	
If_tx_packets tx_size_64_packets	Number of packets transmitted that are 64 bytes in length (from <Destination Address> through <CRC>, inclusively). This register counts all packets, including standard packets, secure packets, FC packets, and manageability packets.	
If_tx_packets tx_size_65_to_127_packets	Number of packets transmitted that are 65-127 bytes in length (from <Destination Address> through <CRC>, inclusively). This register counts all packets, including standard packets, secure packets, and manageability packets.	
If_tx_packets tx_total_packets	Number of all packets transmitted. This register counts the total number of all packets transmitted. This register counts all packets, including standard packets, secure packets, FC packets, and manageability packets.	
If_tx_packets tx_xoff_packets	Number of XOFF packets transmitted.	
If_tx_packets tx_xon_packets	Number of XON packets transmitted.	
Operations flow_director_added_filters	This field counts the number of added filters to the flow director filters logic.	

Units and Instance	Description	Comment
Operations flow_director_matched_filters	This field counts the number of matched filters to the flow director filters logic.	
Operations flow_director_missed_filters	This field counts the number of missed filters to the flow director filters logic.	
Operations flow_director_removed_filters	This field counts the number of removed filters from the flow director filters logic.	
Errors corrected_memory_errors	The total number of hardware errors that were corrected by the hardware (e.g., using a single bit data corruption that was correctible using ECC). These errors do not require immediate software actions but are still reported for accounting and predictive failure analysis.	Memory (RAM) errors are among the most common errors in typical server systems. They also scale with the amount of memory: the more memory the more errors. In addition, large clusters of computers with tens or hundreds (or sometimes thousands) of active machines increase the total error rate of the system. http://www.mcelog.org/memory.html
Errors uncorrected_memory_error	The total number of uncorrected hardware errors detected by the hardware. Data corruption has occurred. These errors require software reaction.	
Errors corrected_memory_errors_in_%s	The total number of hardware errors that were corrected by the hardware in a certain period of time.	Where %s is a timed period like 24 hours.
Errors uncorrected_memory_errors_in_%s	The total number of uncorrected hardware errors detected by the hardware in a certain period of time.	Where %s is a timed period like 24 hours.

VNF specific (Example: Virtual Provider Edge (vPE)/VMX)

Units and Instance	Description	Comment
RX lcore to PPE lcore dequeues	Pipeline model: number of packets dequeued from RX lcore to Packet Processing Engine lcore. RX lcore can be per RX queue.	
RX lcore to PPE lcore enqueues	Number of packets enqueued by PPE lcore.	
RX lcore to PPE lcore drops	Packets dropped due to PPE lcore being busy.	

Units and Instance	Description	Comment
PPE lcore to TX lcore dequeues	Number of packets dequeued from PPE to TX lcore. TX lcore can represent TX queue of the NIC Virtual function.	
PPE lcore to TX lcore enqueues	Number of packets enqueued from PPE to TX lcore. TX lcore can represent TX queue of the NIC Virtual function.	
PPE lcore to TX lcore drops	Packets dropped due to TX core being busy.	
PPE lcore to Exception lcore dequeues	Number of packets dequeued from PPE to Exception lcore. Exception lcore is responsible for processing control packets.	
PPE lcore to Exception lcore enqueues	Number of packets enqueued from PPE to Exception lcore.	
PPE lcore to Exception lcore drops	Packets dropped due to host bound lcore being busy.	
Number of Active flows	Number of active flows present in the VNF.	
Number_of_Flows_Cached	Number of flows cached with action list.	
Number_of_Flows_Not_Cached	Number of flows that are not cached and go through microcode processing.	
Number_of_Flows_Aged	Number of flows aged out.	
Flow Action List	Flow action table for a given flow index.	String of flow actions.
Number_of_IO_Lcores	Number of lcores assigned for Packet Input/Output.	
Number_of_PPE_Lcores	Number of lcores assigned for packet processing engine.	
Number_of_Scheduler_Lcores	Number of lcores assigned for QoS scheduler.	

IPMI Plugin

A baseboard management controller (BMC) is a specialized service processor that uses sensors to monitor the physical state of computer, network server or other hardware device. It also communicates with the system administrator through an independent connection. The BMC is part of the Intelligent Platform Management Interface (IPMI) and is usually contained in the motherboard or main circuit board of the device to be monitored. The sensors of a BMC measure internal physical variables such as temperature, humidity, power-supply voltage, fan speeds, communications parameters, and operating system (OS) functions.

If any of these variables deviate from specified limits, the administrator is notified and takes corrective action through remote control. This means the monitored device can be power cycled or rebooted as necessary, empowering a single administrator to remotely manage numerous servers and other devices simultaneously. This saves on the overall operating cost of the network and helps ensure its reliability.

IPMI defines two basic types of sensors. Threshold sensors monitor "analog" variables, such as temperature, voltage, or fan speed. Discrete sensors monitor events or states, such as entity presence, software initialization progress, or whether external power is applied to the system.

Both threshold and discrete sensors can generate events. Sensor events are stored in system event log (SEL). Most entries will display the SEL record id, date of event, time of event, sensor group, sensor name, and the sensor event occurrence. Some timestamps in the SEL may report a date of 1-Jan-1970. This timestamp is not necessarily incorrect. It usually indicates a hardware event that occurred before a timestamp in firmware has been initialized. For example, certain hardware components will have their internal clocks reset during a power cycle. The IPMI plugin collects information about sensors provided by BMC and generates notifications when events are received from BMC. The sensors are specific to a given BMC, so these will change depending on what the BMC supports. Below is an example for the Intel® S2600WT2R platform.

Units and Instance	**Description**	**Comment**
Percent MTT CPU2	IPMI defines many types of sensors, but groups them into two main categories: threshold and discrete. Threshold sensors are "analog", they have continuous (or mostly continuous) readings. Variables such as fans speed, voltage, or temperature. Discrete sensors have a set of binary readings that may each be independently zero or one. In some sensors, these may be independent. For instance, a power supply may have both an external power failure and a predictive failure at the same time. In other cases, they may be mutually exclusive. For instance, each bit may represent the initialization state of a piece of software.	The IPMI plugin supports analog sensors of type voltage, temperature, fan and current + analog sensors that have VALUE type WATTS, CFM and percentage (%). http://openipmi.sourceforge.net/IPMI.pdf
Percent MTT CPU1		
Percent P2 Therm Ctrl %		
Percent P1 Therm Ctrl %		
Percent PS1 Curr Out %		
Voltage BB +3.3V Vbat		
Voltage BB +12.0V		
Temperature Agg Therm Mgn 1		
Temperature DIMM Thrm Mrgn 4		
Temperature DIMM Thrm Mrgn 3		
Temperature DIMM Thrm Mrgn 2		
Temperature DIMM Thrm Mrgn 1		

Units and Instance	Description	Comment
Temperature P2 DTS Therm Mgn		
Temperature P1 DTS Therm Mgn		
Temperature P2 Therm Ctrl %		
Temperature P1 Therm Ctrl %		
Temperature P2 Therm Margin		
Temperature P1 Therm Margin		
Temperature PS1 Temperature		
Temperature LAN NIC Temp		
Temperature Exit Air Temp		
Temperature HSBP 1 Temp		
Temperature I/O Mod Temp		
Temperature BB Lft Rear Temp		
Temperature BB Rt Rear Temp		
Temperature BB BMC Temp		
Temperature SSB Temp		
Temperature Front Panel Temp		
Temperature BB P2 VR Temp		
Temperature BB P1 VR Temp		
Fan System Fan 6B		
Fan System Fan 6A		
Fan System Fan 5B		
Fan System Fan 5A		

Units and Instance	Description	Comment
Fan System Fan 4B		
Fan System Fan 4A		
Fan System Fan 3B		
Fan System Fan 3A		
Fan System Fan 2B		
Fan System Fan 2A		
Fan System Fan 1B		
Fan System Fan 1A		
CFM System Airflow		
Watts PS1 Input Power		

Intel® PMU Plugin

Performance counters are CPU hardware registers that count hardware events such as instructions executed, cache-misses suffered, or branches mis-predicted. They form a basis for profiling applications to trace dynamic control flow and identify hotspots. The Linux perf interface provides rich generalized abstractions over hardware-specific capabilities.

PMU Tools

PMU tools are a collection of tools for profiling and performance analysis on Intel CPUs on top of Linux perf. This uses performance counters in the CPU. These tools are developed and maintained on https://github.com/andikleen/pmu-tools. In addition to a number of tools for profiling and performance analysis, this package provides jevents library. The intel_pmu plugin collects information provided by Linux perf interface. The metrics it collects are shown below.

Units and Instance	Description	Comment
cpu-cycles	[Hardware Event]	The types of events are: Hardware Events — These are instrument low-level processor activity based on CPU performance counters. For example, CPU cycles, instructions retired, memory stall cycles, level 2 cache misses, etc. Some will be listed as Hardware Cache Events. Software Events — These are low-level events based on kernel counters. For example, CPU migrations, minor faults, major faults, etc. http://www.brendangregg.com/perf.html#Events
instructions		
cache-references		
cache-misses		
branch-instructionsORbranches		
branch-misses		
cpu-clock	[Software Event]	
task-clock		
page-faultsORfaults		
minor-faults		
major-faults		
context-switchesORcs		
cpu-migrationsORmigrations		
alignment-faults		
emulation-faults		
L1-dcache-loads	[Hardware Cache Event]	
L1-dcache-load-misses		
L1-dcache-stores		
L1-dcache-store-misses		
L1-dcache-prefetches		
L1-dcache-prefetch-misses		
L1-icache-loads		
L1-icache-load-misses		
L1-icache-prefetches		
L1-icache-prefetch-misses		
LLC-loads		
LLC-load-misses		
LLC-stores		
LLC-store-misses		
LLC-prefetch-misses		
dTLB-loads		
dTLB-load-misses		
dTLB-stores		
dTLB-store-misses		
dTLB-prefetches		
dTLB-prefetch-misses		

Events

Collectd is an open source daemon that gathers metrics from various sources, such as the operating system, applications, logfiles and external devices. It stores this information or makes it available over the network, the statistics of which can be used to monitor systems, find performance bottlenecks (i.e., *performance analysis*) and predict future system load (i.e., *capacity planning*).

Collectd can generate events based on thresholds for any of the metrics reported in the table above. For more info please consult the following link: https://collectd.org/documentation/manpages/collectd.conf.5.shtml#threshold_configuration

OVS Events

The OVS events plug-in uses an events-based mechanism to retrieve link status and receive interface status change events from the OVS database. When the connection to the OVS database is lost, the plugin generates the appropriate notification.

Units and Instance	**Severity**	**Description**
Gauge Link_status	Warning on Link Status Down. OKAY on link Status Up.	Link status of the OvS interface: UP or DOWN. Severity will be configurable by the end user.

DPDK Events

A read plugin that retrieves DPDK link status and DPDK forwarding cores status (DPDK Keep Alive).

Units and Instance	**Severity**	**Description**	**Comment**
Link_status	Warning on Link Status Down, OKAY on link Status Up.	Link status of the OvS interface: UP or DOWN. Severity will be configurable by the end user.	Depending on plugin configuration, can be dispatched as a metric or event.
keep_alive	OKAY: if core status is ALIVE, UNUSED, DOZING, SLEEP Warning: if core status is MISSING. Failure: if core status is DEAD or GONE.	Reflects the state of DPDK packet processing cores.	Protects against packet processing core failures for DPDK → no silent packet drops. Depending on plugin configuration, can be dispatched as a metric or event.

PCIE Events

The plugin creates the list of available PCIe devices using sysfs access to PCI devices and their config space. The list is enumerated on every interval and config space is polled to read available errors register. If new error is set, the plugin is sent notification. The type_instance and severity depend on error category.

Units and Instance	Severity	Description	Comment
Pcie_error correctable	Notification (Warning) in case of PCIe correctable error occurrence. Message contains short error description.	correctable	Correctable Errors include: Receiver Error Status Bad TLP Status Bad DLLP Status REPLAY_NUM Rollover Replay Timer Timeout Advisory Non-Fatal Corrected Internal Header Log Overflow Uncorrectable Errors include: Data Link Protocol Surprise Down Poisoned TLP Flow Control Protocol Completion Timeout Completer Abort Unexpected Completion Receiver Overflow Malformed TLP ECRC Error Status Unsupported Request ACS Violation Internal MC blocked TLP Atomic egress blocked TLP prefix blocked
Pcie_error fatal	Notification (Failure) in case of PCIe uncorrectable fatal error occurrence. Message contains short error description.	fatal	
Pcie_error non_fatal	Notification (Warning) in case of PCIe uncorrectable non-fatal error occurrence. Message contains short error description.	non_fatal	

Mcelog Events

The Mcelog plugin sends notifications and statistics relevant to Machine Check Exceptions (MCE) when they occur. The plugin leverages the mcelog Linux utility to detect that an exception has occurred.

Units and Instance	Severity	Description	Comment
mcelog (RAS memory) errors	Warning for Corrected Memory Errors. Failure for Uncorrected Memory Errors.	Failure on failure to connect to the mcelog socket/if connection is lost. OK on connection to mcelog socket.	Reports Corrected and Uncorrected DIMM Failures.

IPMI Events

The IPMI plugin generates notifications when an event is received from BMC; it exposes as much information as it is provided using notifications.

Severity	Description	Comment
OKAY - upper non-critical OKAY - lower non-critical WARNING- lower critical WARNING - upper critical FAILURE - upper non-recoverable FAILURE - lower non-recoverable Discrete sensor status changes are also reported out via OKAY, WARNING and FAILURE notifications. Examples of discrete sensors can be found under the "IPMI Sensors for S2600WT2R" tab.	Each IPMI sensor may have six different thresholds: upper non-recoverable upper critical upper non-critical lower non-critical lower critical lower non-recoverable.	Events may be on a threshold sensor by specifying values (called thresholds) where it is desirable that the sensor report an event. Then events can be enabled for the specific thresholds. Not all sensors support all thresholds, some cannot have their events enabled and others cannot have them disabled. The capabilities of a sensor may all be queried by the user to determine what it can do. When the value of the sensor goes outside the threshold an event may be generated. An event may be generated when the value goes back into the threshold.

Platform Specific Events

Platform specific events such as mcelog RAS System, CPU, QPI, OI (specific to a Platform) will change depending on what's supported by the Platform.

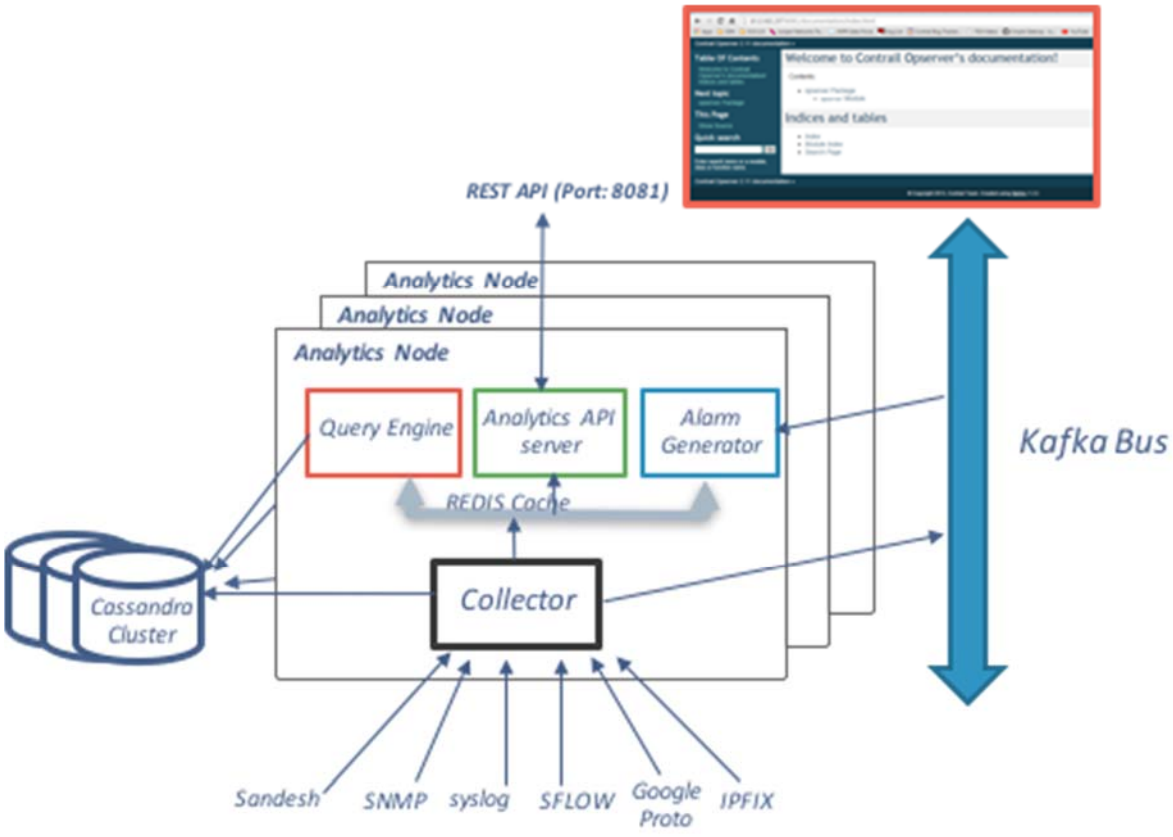
Severity	Description	Comment
WARNING - Correctable errors FAILURE - Uncorrectable Errors	Servers based on Intel® Architecture, are generally designed for use in mission critical environments. Reliability, Availability and Serviceability (RAS) features, are integrated into the servers to address the error handling and memory mirroring and sparing required by these environments. The goal of this feature is to expose the RAS features provided by the Broadwell or newer platform to higher level fault management applications. The Features to be exposed fall under the following categories: **Reliability Features**: -System attributes to ensure Data integrity. -Capability to prevent, detect, correct and contain faults over a given time interval. **Availability Features**: -System attributes to help stay operational in the presence of faults in the system. -Capability to map out failed units, ability to operate in a degraded mode. **Serviceability Features**: -System attributes to help system service, repair. -Capability to identify failed units, and facilitates repair. **Generic Error Handling** The Silicon supports corrected, uncorrected (recoverable, unrecoverable), fatal, and catastrophic error types.	/* See IA32 SDM Vol3B Chapter 16*/ Integrated Memory Controller Machine Check Errors "Address parity error", "HA write data parity error", "HA write byte enable parity error", "Corrected patrol scrub error", "Uncorrected patrol scrub error", "Corrected spare error", "Uncorrected spare error", "Any HA read error", "WDB read parity error", "DDR4 command address parity error", "Uncorrected address parity error" "Unrecognized request type", "Read response to an invalid scoreboard entry", "Unexpected read response", "DDR4 completion to an invalid scoreboard entry", "Completion to an invalid scoreboard entry", "Completion FIFO overflow", "Correctable parity error", "Uncorrectable error", "Interrupt received while outstanding interrupt was not ACKed", "ERID FIFO overflow", "Error on Write credits", "Error on Read credits", "Scheduler error", "Error event", "MscodDataRdErr", "Reserved", "MscodPtlWrErr", "MscodFullWrErr", "MscodBgfErr", "MscodTimeout", "MscodParErr", "MscodBucket1Err" Interconnect(QPI) Machine Check Errors "UC Phy Initialization Failure", "UC Phy detected drift buffer alarm", "UC Phy detected latency buffer rollover", "UC LL Rx detected CRC error: unsuccessful LLR: entered abort state", "UC LL Rx unsupported or undefined packet", "UC LL or Phy control error", "UC LL Rx parameter exchange exception", "UC LL detected control error from the link-mesh interface",

Severity	Description	Comment
	Corrected Errors Errors that are corrected by either hardware or software, corrected error information is used in predictive failure analysis by the OS. MCA Banks corrected errors (except selected memory corrected errors) are handled directly by the OS. HASWELL-EP PROCESSOR triggers CMCI for the corrected errors, or CMCI OS can read the MCA Banks and collect error status. All the other platform-related corrected errors can either be ignored or can be logged into BMC SEL based on platform policy. **Memory Corrected Errors** Memory corrected errors such as mirror fail over and memory read errors can be configured to trigger SMI using BIOS setup options. On memory mirror fail over, BIOS logs the error for the OS as per the UEFI error record format. On memory read errors, BIOS does the following memory RAS operations in order to correct the error. **Rank Sparing** SDDC/Device tagging Uncorrected Non-Fatal Errors. Errors that are not corrected by hardware generally trigger machine check exception and in turn trigger SMI. The BIOS SMI handler logs this error information, clears the error status and passes the error log to the OS. The OS can recover from the error, or in cases where recovery is not an option, can trigger a system reset.	"COR Phy initialization abort", "COR Phy reset", "COR Phy lane failure, recovery in x8 width", "COR Phy L0c error corrected without Phy reset", "COR Phy L0c error triggering Phy Reset", "COR Phy L0p exit error corrected with Phy reset", "COR LL Rx detected CRC error - successful LLR without Phy Reinit", "COR LL Rx detected CRC error - successful LLR with Phy Reinit" "Phy Control Error", "Unexpected Retry.Ack flit", "Unexpected Retry.Req flit", "RF parity error", "Routeback Table error", "unexpected Tx Protocol flit (EOP, Header or Data)", "Rx Header-or-Credit BGF credit overflow/underflow", "Link Layer Reset still in progress when Phy enters L0", "Link Layer reset initiated while protocol traffic not idle", "Link Layer Tx Parity Error"

Severity	Description	Comment
	Uncorrected Fatal Errors For errors that are neither corrected by hardware nor recovered by the s/w, the system is not in a reliable state and needs a reset to bring it back up to normal.	

Networking — OpenContrail

This section describes how to collect KPI metrics from OpenContrail SDN controller. Contrail Analytics provide full statistical data of overlay networks and host networking data via the REST API interface. Any OSS/BSS application or tool can be further integrated into Contrail Analytics.



Contrail Analytics collectors support different protocols for data connection. For data center compute nodes, it uses the Sandesh protocol, which collects data from compute nodes and makes it available via REST APIs. This section highlights the main networking KPIs and how to collect that data via REST APIs.

Packet Loss/Drop Stats

vRouter (Neutron Plugin) Measure the packet loss and drop stats UVE (User Visible Entities)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/vrouter | python -m json.tool**
	curl http://<ip>:<port>/analytics/uves/vrouter/<compute>?flat | python -m json.tool
	
	"drop_stats": {
	"ds_cksum_err": 0,
	"ds_discard": 0,
	"ds_drop_new_flow": 0,
	"ds_drop_pkts": 0,
	"ds_duplicated": 0,
	"ds_flow_action_drop": 0,
	"ds_flow_action_invalid": 0,
	"ds_flow_evict": 0,
	"ds_flow_invalid_protocol": 0,
	"ds_flow_nat_no_rflow": 0,
	"ds_flow_no_memory": 0,
	"ds_flow_queue_limit_exceeded": 0,
	"ds_flow_table_full": 0,
	"ds_flow_unusable": 0,
	"ds_frag_err": 0,
	"ds_fragment_queue_fail": 0,
	"ds_head_alloc_fail": 0,
	"ds_interface_drop": 0,
	"ds_interface_rx_discard": 0,
	"ds_interface_tx_discard": 0,
	"ds_invalid_arp": 0,
	"ds_invalid_if": 0,
	"ds_invalid_label": 0,
	"ds_invalid_mcast_source": 0,
	"ds_invalid_nh": 0,
	"ds_invalid_packet": 0,
	"ds_invalid_protocol": 0,
	"ds_invalid_source": 0,
	"ds_invalid_vnid": 0,
	"ds_l2_no_route": 0,
	"ds_mcast_clone_fail": 0,
	"ds_mcast_df_bit": 0,
	"ds_misc": 0,
	"ds_no_fmd": 0,
	"ds_no_memory": 0,
	"ds_nowhere_to_go": 0,
	"ds_pcow_fail": 0,
	"ds_pull": 0,
	"ds_push": 0,
	"ds_rewrite_fail": 0,
	"ds_trap_no_if": 0,
	"ds_trap_original": 0,
	"ds_ttl_exceeded": 0,
	"ds_vlan_fwd_enq": 0,
	"ds_vlan_fwd_tx": 0
	}

Virtual Machine Interface (VM Tap Interface-Level KPIs)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/virtual-machine-interface/default-domain:vPE:c92cc8ca-eaf6-4ebb-a13a-0a3fd563944a?flat | python -m json.tool**
	
	"drop_stats_1h": {
	"ds_cksum_err": 0,
	"ds_discard": 0,
	"ds_drop_new_flow": 0,
	"ds_drop_pkts": 0,
	"ds_duplicated": 0,
	"ds_flow_action_drop": 0,
	"ds_flow_action_invalid": 0,
	"ds_flow_evict": 0,
	"ds_flow_invalid_protocol": 0,
	"ds_flow_nat_no_rflow": 0,
	"ds_flow_no_memory": 0,
	"ds_flow_queue_limit_exceeded": 0,
	"ds_flow_table_full": 0,
	"ds_flow_unusable": 0,
	"ds_frag_err": 0,
	"ds_fragment_queue_fail": 0,
	"ds_head_alloc_fail": 0,
	"ds_interface_drop": 0,
	"ds_interface_rx_discard": 0,
	"ds_interface_tx_discard": 0,
	"ds_invalid_arp": 0,
	"ds_invalid_if": 0,
	"ds_invalid_label": 0,
	"ds_invalid_mcast_source": 0,
	"ds_invalid_nh": 0,
	"ds_invalid_packet": 0,
	"ds_invalid_protocol": 0,
	"ds_invalid_source": 0,
	"ds_invalid_vnid": 0,
	"ds_l2_no_route": 0,
	"ds_mcast_clone_fail": 0,
	"ds_mcast_df_bit": 0,
	"ds_misc": 0,
	"ds_no_fmd": 0,
	"ds_no_memory": 0,
	"ds_nowhere_to_go": 0,
	"ds_pcow_fail": 0,
	"ds_pull": 0,
	"ds_push": 0,
	"ds_rewrite_fail": 0,
	"ds_trap_no_if": 0,
	"ds_trap_original": 0,
	"ds_ttl_exceeded": 0,
	"ds_vlan_fwd_enq": 0,
	"ds_vlan_fwd_tx": 0
	}

Host Physical Interface Level	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/vrouter/<compute>?flat | python -m json.tool**
	
	Result:
	
	"phy_flow_rate": {
	"p6p1": {
	"active_flows": 0,
	"added_flows": 0,
	"deleted_flows": 0,
	"max_flow_adds_per_second": 0,
	"max_flow_deletes_per_second": 0,
	"min_flow_adds_per_second": 0,
	"min_flow_deletes_per_second": 0
	}
	},
	"phy_if_5min_usage": [
	{
	"in_bandwidth_usage": 7482,
	"name": "p6p1",
	"out_bandwidth_usage": 16703
	}
	],
	"phy_if_drop_stats_1h": {
	"p6p1": {
	"ds_cksum_err": 0,
	"ds_discard": 0,
	"ds_drop_new_flow": 0,
	"ds_drop_pkts": 0,
	"ds_duplicated": 0,
	"ds_flow_action_drop": 0,
	"ds_flow_action_invalid": 0,
	"ds_flow_evict": 0,
	"ds_flow_invalid_protocol": 0,
	"ds_flow_nat_no_rflow": 0,
	"ds_flow_no_memory": 0,
	"ds_flow_queue_limit_exceeded": 0,
	"ds_flow_table_full": 0,
	"ds_flow_unusable": 0,
	"ds_frag_err": 0
	"ds_fragment_queue_fail": 0,
	"ds_head_alloc_fail": 0,
	"ds_interface_drop": 0,
	"ds_interface_rx_discard": 0,
	"ds_interface_tx_discard": 0,
	"ds_invalid_arp": 0,
	"ds_invalid_if": 0,
	"ds_invalid_label": 0,
	"ds_invalid_mcast_source": 0,
	"ds_invalid_nh": 0,
	"ds_invalid_packet": 0,
	"ds_invalid_protocol": 0,
	"ds_invalid_source": 0,
	"ds_invalid_vnid": 0,
	"ds_l2_no_route": 0,
	"ds_mcast_clone_fail": 0,

Host Physical Interface Level	

	"ds_mcast_df_bit": 0,
	"ds_misc": 0,
	"ds_no_fmd": 0,
	"ds_no_memory": 0,
	"ds_nowhere_to_go": 0,
	"ds_pcow_fail": 0,
	"ds_pull": 0,
	"ds_push": 0,
	"ds_rewrite_fail": 0,
	"ds_trap_no_if": 0,
	"ds_trap_original": 0,
	"ds_ttl_exceeded": 0,
	"ds_vlan_fwd_enq": 0,
	"ds_vlan_fwd_tx": 0
	}
	},
	"phy_if_info": {
	"p6p1": {
	"duplexity": 1,
	"speed": 10000
	}
	},
	"phy_if_stats": {
	"p6p1": {
	"in_bytes": 41428,
	"in_pkts": 252,
	"out_bytes": 77001,
	"out_pkts": 141
	}
	},

Packets per Second

PPS per vRouter	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/vrouter/<compute>?flat | python -m json.tool**
	
	Result:
	
	"phy_if_stats": {
	"p6p1": {
	"in_bytes": 41428,
	"in_pkts": 252,
	"out_bytes": 77001,
	"out_pkts": 141
	}
	},

PPS per Virtual Machine Interface (VM Tap Interface)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/virtual-machine-interface/default-domain:vPE:c92cc8ca-eaf6-4ebb-a13a-0a3fd563944a?flat | python -m json.tool**
	
	Result:
	
	"if_stats": {
	"in_bytes": 0,
	"in_pkts": 0,
	"out_bytes": 126,
	"out_pkts": 3
	}
	
	"drop_stats_1h": {
	"ds_cksum_err": 0,
	"ds_discard": 0,
	"ds_drop_new_flow": 0,
	"ds_drop_pkts": 0,
	"ds_duplicated": 0,
	"ds_flow_action_drop": 0,
	"ds_flow_action_invalid": 0,
	"ds_flow_evict": 0,
	"ds_flow_invalid_protocol": 0,
	"ds_flow_nat_no_rflow": 0,
	"ds_flow_no_memory": 0,
	"ds_flow_queue_limit_exceeded": 0,
	"ds_flow_table_full": 0,
	"ds_flow_unusable": 0,
	"ds_frag_err": 0,
	"ds_fragment_queue_fail": 0,
	"ds_head_alloc_fail": 0,
	"ds_interface_drop": 0,
	"ds_interface_rx_discard": 0,
	"ds_interface_tx_discard": 0,
	"ds_invalid_arp": 0,
	"ds_invalid_if": 0,
	"ds_invalid_label": 0,
	"ds_invalid_mcast_source": 0,
	"ds_invalid_nh": 0,
	"ds_invalid_packet": 0,
	"ds_invalid_protocol": 0,
	"ds_invalid_source": 0,
	"ds_invalid_vnid": 0,
	"ds_l2_no_route": 0,
	"ds_mcast_clone_fail": 0,
	"ds_mcast_df_bit": 0,
	"ds_misc": 0,
	"ds_no_fmd": 0,
	"ds_no_memory": 0,
	"ds_nowhere_to_go": 0,
	"ds_pcow_fail": 0,
	"ds_pull": 0,
	"ds_push": 0,
	"ds_rewrite_fail": 0,
	"ds_trap_no_if": 0,
	"ds_trap_original": 0,
	"ds_ttl_exceeded": 0,
	"ds_vlan_fwd_enq": 0,
	"ds_vlan_fwd_tx": 0
	}
	},

PPS per Virtual Network	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/virtual-network/default-domain:vPE:<VN>?flat| python -m json.tool**
	
	Results:
	
	"vn_stats": [
	{
	"in_bytes": 2510752,
	"in_tpkts": 5062,
	"out_bytes": 0,
	"out_tpkts": 0
	“vrouter”: “CC3x-Compute3”
	}
	
	{
	"in_bytes": 613383360,
	"in_tpkts ": 1236660,
	"other_vn:": “default-domain:vPE:MPLSoUDP”,
	"out_bytes": 0,
	"out_tpkts": 0
	“vrouter”: “CC3x-Compute3”
	},
	
	{
	"in_bytes": 0,
	"in_tpkts ": 0,
	"other_vn:": “default-domain:vPE:MPLSoUDP-D2IPE2”,
	"out_bytes": 599309642,
	"out_tpkts": 1243381
	“vrouter”: “CC3x-Compute3”
	},

PPS per Physical Interface (Host Interface)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/virtual-network/default-domain:vPE:<VN>?flat| python -m json.tool**
	
	Results:
	
	"phy_flow_rate": [
	"bond0": {
	"active_flows": 0,
	"added_flows": 0,
	"deleted_flows": 0
	"max_flow_adds_per_second": 0
	"max_flow_deletes_per_second": 0
	"min_flow_adds_per_second": 0
	"min_flow_deletes_per_second": 0
	}
	},
	
	"phy_if_5min_usage": [
	{
	"in_bandwidth_usage": 355317939,
	"name": "bond0",
	"out_bandwidth_usage": 205136091
	}
	],
	"phy_if_stats": {
	"bond0": {
	"drop_pkts": 1800005,
	"in_bytes": 951228689,
	"in_pkts": 1803450,
	"out_bytes": 8298689,
	"out_pkts": 5887
	}
	},

Flows Per Sec KPI

Flows per Sec vRouter	

REST API Call:	**Curl http://<ip>:<port>/analytics/uves/vrouter/CC3x-Compute3?flat| python -m json.tool**
	
	Results:
	
	"VrouterStatsAgent": {
	"active_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2741
	"sigma": 0.899238
	"state": {
	"mean": "819.15",
	"stddev": "201.115"
	}
	},
	
	"flow_rate": {
	"active_flows": 1000,
	"added_flows": 0,
	"deleted_flows": 0
	"max_flow_adds_per_second": 0
	"max_flow_deletes_per_second": 0
	"min_flow_adds_per_second": 0
	"min_flow_deletes_per_second": 0
	}
	
	"added_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2741
	"sigma": -1.37669
	"state": {
	"mean": "18966.6",
	"stddev": "13776.9"
	}
	},
	
	"deleted_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2741
	"sigma": -1.35325
	"state": {
	"mean": "32748.9",
	"stddev": "24200.2"
	}
	},

Flows per Sec Virtual Machine Interface (VM Tap Interface)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/virtual-machine-interface/default-domain:<project>:<UUID>?flat| python -m json.tool**
	
	Results:
	
	"UveVMInterfaceAgent": {
	"active": true,
	"active_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 41300,
	"sigma": -0.899939
	"state": {
	"mean": "781.899",
	"stddev": "19.8894"
	}
	},
	
	"added_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 41300
	"sigma": -0.661435
	"state": {
	"mean": "10.8857",
	"stddev": "16.4577"
	}
	},
	
	"deleted_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 41300
	"sigma": -0.578272
	"state": {
	"mean": "12.8745",
	"stddev": "22.2638"
	}
	},
	"fixed_ip4_list": [
	"5.5.5.3",
	"5.5.5.4",
	]
	"flow_rate": {
	"active_flows": 764,
	"added_flows": 0,
	"deleted_flows": 0
	"max_flow_adds_per_second": 0
	"max_flow_deletes_per_second": 0
	"min_flow_adds_per_second": 0
	"min_flow_deletes_per_second": 0
	}

Flows per Sec Physical Interface (Host Interface)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/vrouter/CC3x-Compute3?flat| python -m json.tool**
	
	Results:
	
	"phy_active_flows_ewm": {
	"bond0": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2752,
	"sigma": 0.0
	"state": {
	"mean": "0",
	"stddev": "0"
	}
	},
	},
	"phy_added_flows_ewm": {
	"bond0": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2752
	"sigma": 0.0
	"state": {
	"mean": "0",
	"stddev": "0"
	}
	},
	},
	"phy_deleted_flows_ewm": {
	"bond0": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2752
	"sigma": 0.0
	"state": {
	"mean": "0",
	"stddev": "0"
	}
	},
	},
	"phy_flow_rate": {
	"bond0": {
	"active_flows": 0,
	"added_flows": 0,
	"deleted_flows": 0
	"max_flow_adds_per_second": 0
	"max_flow_deletes_per_second": 0
	"min_flow_adds_per_second": 0
	"min_flow_deletes_per_second": 0
	}
	},

Bandwidth Usage

Bandwidth per vRouter	

REST API Call:	**Curl http://<ip>:<port>/analytics/uves/vrouter/<Compute>?flat| python -m json.tool**
	
	Results:
	
	"phy_band_in_bps": {
	"bond0": "402030026"
	},
	"phy_band_out_bps ": {
	"bond0": "30720889"
	},
	
	"in_bps_ewm": {
	"bond0": {
	"mean": 351624000.0
	"samples": 1372
	"sigma": -1.17127,
	"stddev": 83697500.0
	}
	},
	"out_bps_ewm": {
	"bond0": {
	"mean": 196841000.0
	"samples": 1372
	"sigma": -1.17583,
	"stddev": 165682000.0
	}
	},

Bandwidth Utilization per Virtual Machine Interface (VM tap Interface)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/virtual-machine-interface/default-domain:vPE:c92cc8ca-eaf6-4ebb-a13a-0a3fd563944a?flat | python -m json.tool**
	
	Results:
	
	"if stats": {
	"drop_pkts": 2386
	"in_bytes": 358040080
	"in_pkts": 721855
	"out_bytes": 356976160
	"out_pkts": 719710
	},
	
	"if_in_pkts_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2571
	"sigma": -0.107204
	"state": {
	"mean": "791501",
	"stddev": "649661"
	}
	},
	
	"if_out_pkts_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2571
	"sigma": -0.11034
	"state": {
	"mean": "791382",
	"stddev": "649560"
	}
	},

Bandwidth Utilization per Virtual Network	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/virtual-network/default-domain:vPE:<VN>?flat| python -m json.tool**
	
	Results:
	
	"vn_stats": [
	{
	"in_bytes": 2510752,
	"in_tpkts": 5062,
	"other_vn": "__UNKNOWN__"
	"out_bytes": 0,
	"out_tpkts": 0,
	"vrouter": "CC3x-Compute3"
	},
	
	{
	"in_bytes": 613383360,
	"in_tpkts": 1236660,
	"other_vn": "default-domain:vPE:MPLSoUDP",
	"out_bytes": 0,
	"out_tpkts": 0,
	"vrouter": "CC3x-Compute3"
	},
	
	{
	"in_bytes": 0,
	"in_tpkts": 0,
	"other_vn": "default-domain:vPE:MPLSoUDP-D2IPE2",
	"out_bytes": 599309642,
	"out_tpkts": 1243381,
	"vrouter": "CC3x-Compute3"
	}

Bandwidth utilization per Physical Interface	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/vrouter/<Compute>?flat| python -m json.tool**
	
	Results:
	
	"phy_band_in_bps": {
	"bond0": "402030026"
	},
	"phy_band_out_bps ": {
	"bond0": "30720889"
	},
	
	"in_bps_ewm": {
	"bond0": {
	"mean": 351624000.0
	"samples": 1372
	"sigma": -1.17127,
	"stddev": 83697500.0
	}
	},
	"out_bps_ewm": {
	"bond0": {
	"mean": 196841000.0
	"samples": 1372
	"sigma": -1.17583,
	"stddev": 165682000.0
	}
	},

Concurrent Session KPI

Concurrent Sessions per vRouter	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/vrouter/CC3x-Compute3?flat| python -m json.tool**
	
	Results:
	
	"VrouterStatsAgent": {
	"active_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2741
	"sigma": 0.899238
	"state": {
	"mean": "819.15",
	"stddev": "201.115"
	}
	},
	
	"flow_rate": {
	"active_flows": 1000,
	"added_flows": 0,
	"deleted_flows": 0
	"max_flow_adds_per_second": 0
	"max_flow_deletes_per_second": 0
	"min_flow_adds_per_second": 0
	"min_flow_deletes_per_second": 0
	}
	
	"added_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2741
	"sigma": -1.37669
	"state": {
	"mean": "18966.6",
	"stddev": "13776.9"
	}
	},
	
	"deleted_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2741
	"sigma": -1.35325
	"state": {
	"mean": "32748.9",
	"stddev": "24200.2"
	}
	},

Concurrent Session per Virtual Machine Interface (VM Tap Interface)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/virtual-machine-interface/default-domain:<project>:<UUID>?flat| python -m json.tool**
	
	Results:
	
	"UveVMInterfaceAgent": {
	"active": true,
	"active_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 41300,
	"sigma": -0.899939
	"state": {
	"mean": "781.899",
	"stddev": "19.8894"
	}
	},
	
	"added_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 41300
	"sigma": -0.661435
	"state": {
	"mean": "10.8857",
	"stddev": "16.4577"
	}
	},
	
	"deleted_flows_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 41300
	"sigma": -0.578272
	"state": {
	"mean": "12.8745",
	"stddev": "22.2638"
	}
	},
	"fixed_ip4_list": [
	"5.5.5.3",
	"5.5.5.4",
	]
	"flow_rate": {
	"active_flows": 764,
	"added_flows": 0,
	"deleted_flows": 0
	"max_flow_adds_per_second": 0
	"max_flow_deletes_per_second": 0
	"min_flow_adds_per_second": 0
	"min_flow_deletes_per_second": 0
	}

Concurrent Session per Physical Interface (Host Interface)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/vrouter/CC3x-Compute3?flat| python -m json.tool**
	
	Results:
	
	"phy_active_flows_ewm": {
	"bond0": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2752,
	"sigma": 0.0
	"state": {
	"mean": "0",
	"stddev": "0"
	}
	},
	},
	"phy_added_flows_ewm": {
	"bond0": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2752
	"sigma": 0.0
	"state": {
	"mean": "0",
	"stddev": "0"
	}
	},
	},
	"phy_deleted_flows_ewm": {
	"bond0": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2752
	"sigma": 0.0
	"state": {
	"mean": "0",
	"stddev": "0"
	}
	},
	},
	"phy_flow_rate": {
	"bond0": {
	"active_flows": 0,
	"added_flows": 0,
	"deleted_flows": 0
	"max_flow_adds_per_second": 0
	"max_flow_deletes_per_second": 0
	"min_flow_adds_per_second": 0
	"min_flow_deletes_per_second": 0
	}
	},

Network Utilization KPI

Network Utilization KPI per vRouter	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/vrouter/<Compute>?flat| python -m json.tool**
	
	Results:
	
	"phy_band_in_bps": {
	"bond0": "402030026"
	},
	"phy_band_out_bps ": {
	"bond0": "30720889"
	},
	
	"in_bps_ewm": {
	"bond0": {
	"mean": 351624000.0
	"samples": 1372
	"sigma": -1.17127,
	"stddev": 83697500.0
	}
	},
	"out_bps_ewm": {
	"bond0": {
	"mean": 196841000.0
	"samples": 1372
	"sigma": -1.17583,
	"stddev": 165682000.0
	}
	},
	
	"vhost_stats": {
	"duplexity": -1,
	"in_bytes": 21918478801,
	"in_pkts": 15741656,
	"name": "vhost0",
	"out_bytes": 2828696248,
	"out_pkts": 7462236,
	"speed": -1
	}

Network Utilization per Virtual Machine Interface (VM Tap Interface)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/virtual-machine-interface/default-domain:vPE:c92cc8ca-eaf6-4ebb-a13a-0a3fd563944a?flat | python -m json.tool**
	
	Results:
	
	"if stats": {
	"drop_pkts": 2386
	"in_bytes": 358040080
	"in_pkts": 721855
	"out_bytes": 356976160
	"out_pkts": 719710
	},
	
	"if_in_pkts_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2571
	"sigma": -0.107204
	"state": {
	"mean": "791501",
	"stddev": "649661"
	}
	},
	
	"if_out_pkts_ewm": {
	"algo": "EWM",
	"config": "0.1",
	"samples": 2571
	"sigma": -0.11034
	"state": {
	"mean": "791382",
	"stddev": "649560"
	}
	},
	
	"vhost_stats": {
	"duplexity": -1,
	"in_bytes": 21918478801,
	"in_pkts": 15741656,
	"name": "vhost0",
	"out_bytes": 2828696248,
	"out_pkts": 7462236,
	"speed": -1
	}

Network Utilization per Physical Interface (Host Interface)	

REST API Call:	**curl http://<ip>:<port>/analytics/uves/vrouter/<Compute>?flat| python -m json.tool**
	
	Results:
	
	"phy_band_in_bps": {
	"bond0": "402030026"
	},
	"phy_band_out_bps ": {
	"bond0": "30720889"
	},
	
	"in_bps_ewm": {
	"bond0": {
	"mean": 351624000.0
	"samples": 1372
	"sigma": -1.17127,
	"stddev": 83697500.0
	}
	},
	"out_bps_ewm": {
	"bond0": {
	"mean": 196841000.0
	"samples": 1372
	"sigma": -1.17583,
	"stddev": 165682000.0
	}
	},

Storage

Three basic storage performance metrics are common to all environments: latency, throughput (or bandwidth) and IOPS (input/output operations per second). Below are the key storage metrics, as defined by ETSI GS NFV-TST001 [7]:

Performance/Speed Metrics

- Sequential read/write IOPS
- Random read/write IOPS
- Latency for storage read/write operations
- Throughput for storage read/write operations

Capacity/Scale Metrics

- Storage/Disk size
- Capacity allocation (block-based, object-based)
- Block size
- Maximum sequential read/write IOPS
- Maximum random read/write IOPS
- Disk utilization (max, average, standard deviation)

4. Summary

This report provides insight into a number of telemetry concepts for monitoring, measuring, and managing virtualized network deployments. It explores the current set of metrics defined for NFVI across a multi-vendor environment and characterizes the specific metrics key to performance and capacity of the VNFs running on top of the infrastructure.

As part of the analysis, related standards such as ETSI NFV TST008 [1], which support an independent and consistent look at NFVI performance metrics, were examined. Also included were metrics from open source projects such as OPNFV Barometer [8], as well as KPIs provided by other open source network plugins (e.g., OpenContrail Analytics, Open vSwitch).

As NFV Infrastructure continues to evolve to address the rapid growth of applications and services enabled by 5G, the requirement for real-time and accurate telemetry information will become increasingly critical. Additional NFVI metrics will be driven by network slicing architecture and other technologies, such as new hardware acceleration and container-based virtualization. The continued evolution of NFVI metrics will be essential to reliable virtualized network deployments.

5. References

[1] ETSI GS NFV-TST 008 V2.1.1, "Network Functions Virtualisation (NFV) Release 2; Testing; NFVI Compute and Network Metrics Specification".

[2] OpenContrail, http://www.opencontrail.org.

[3] The Linux Foundation, Open vSwitch, http://openvswitch.org.

[4] Rusling, David A, The Linux Documentation Project: Guides, The Linux Kernel, http://www.tldp.org/LDP/tlk/mm/memory.html.

[5] The Linux Kernel Archives, https://www.kernel.org/doc/gorman/html/understand/understand007.html.

[6] Red Hat Enterprise Linux Performance Tuning Guides, https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-memory-transhuge.

[7] ETSI GS NFV-TST 001 V1.1.1, "Network Functions Virtualisation (NFV); Pre-deployment Testing; Report on Validation of NFV Environments and Services".

[8] OPNFV. (28 September 2016). Barometer Home. Retrieved 4 December 2017, from https://wiki.opnfv.org/display/fastpath/Barometer+Home.