None

Release draft (d7fe651)

OPNFV

January 25, 2016

CONTENTS

The goal of Software Fastpath service Quality Metrics (SFQM) is to develop the utilities and libraries in DPDK to support:

- Measuring Telco Traffic and Performance KPIs. Including:
  - Packet Delay Variation (by enabling TX and RX time stamping).
  - Packet loss (by exposing extended NIC stats).
- Performance Monitoring of the DPDK interfaces (by exposing extended NIC stats + collectd Plugin).
- Detecting and reporting violations that can be consumed by VNFs and higher level management systems (through DPDK Keep Alive).

After all **the ability to measure and enforce Telco KPIs (Service assurance) in the data-plane will be mandatory for any Telco grade NFVI implementation**.

All developed features will be upstreamed to DPDK or other Open Source projects relevant to telemetry such as collectd and fault management such as Monasca and/or Ceilometer.

The OPNFV project wiki can be found @ SFQM

CHAPTER

ONE

PROBLEM STATEMENT

The OPNFV platform (NFVI) requires functionality to:

- Create a low latency, high performance packet processing path (fast path) through the NFVI that VNFs can take advantage of;
- Measure Telco Traffic and Performance KPIs through that fast path;
- Detect and report violations that can be consumed by VNFs and higher level EMS/OSS systems

Examples of local measurable QoS factors for Traffic Monitoring which impact both Quality of Experience and 5'9s availability would be (using Metro Ethernet Forum Guidelines as reference):

- Packet loss
- Packet Delay Variation
- Uni-directional frame delay

Other KPIs such as Call drops, Call Setup Success Rate, Call Setup time etc. are measured by the VNF.

In addition to Traffic Monitoring, the NFVI must also support Performance Monitoring of the physical interfaces themselves (e.g. NICs), i.e. an ability to monitor and trace errors on the physical interfaces and report them.

All these traffic statistics for Traffic and Performance Monitoring must be measured in-service and must be capable of being reported by standard Telco mechanisms (e.g. SNMP traps), for potential enforcement actions.

CHAPTER

TWO

SCOPE

The output of the project will provide interfaces and functions to support monitoring of Packet Latency and Network Interfaces while the VNF is in service.

The DPDK interface/API will be updated to support:

- Exposure of NIC MAC/PHY Level Counters
- Interface for Time stamp on RX
- Interface for Time stamp on TX

Specific testing and integration will be carried out to cover:

- Unit/Integration Test plans: A sample application provided to demonstrate packet latency monitoring and interface monitoring

The following list of features and functionality will be developed:

- DPDK APIs and functions for latency and interface monitoring
- A sample application to demonstrate usage

The scope of the project is limited to the DPDK APIs and a sample application to demonstrate usage.

In the figure above, the interfaces 1, 2, 3 are implemented along with a sample application. The sample application will support monitoring of NIC counters/status and will also support measurement of packet latency using DPDK provided interfaces.

VNF specific processing, Traffic Monitoring, Performance Monitoring and Management Agent are out of scope. The scope is limited to Intel 10G Niantic support.

The Proposed MAC/PHY Interface Counters include:

- Packet RX
- Packet TX
- Packet loss
- Interface errors + other stats

The Proposed Packet Latency Monitor include:

- Cycle accurate ‘stamping’ on ingress
- Supports latency measurements on egress

Support for additional types of Network Interfaces can be added in the future.

Support for failover of DPDK enabled cores is also out of scope of the current proposal. However, this is an important requirement and must-have functionality for any DPDK enabled framework in the NFVI. To that end, a second phase of this project will be to implement DPDK “Keep Alive” functionality that would address this and would report to a

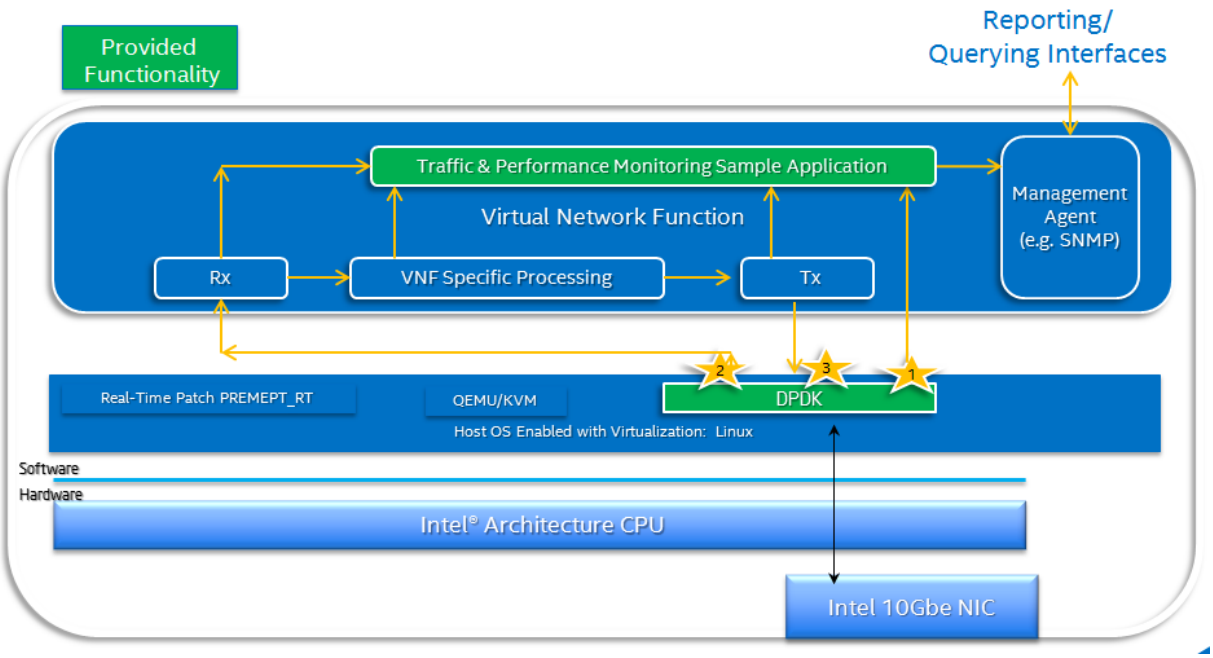

Fig. 2.1: Architecture overview (showing provided functionality in green)

VNF-level Failover and High Availability mechanism that would then determine what actions, including failover, may be triggered.

CHAPTER

THREE

CONSUMPTION MODELS

Fig 1.1 shows how a sample application will be provided to demonstrate usage. In reality many VNFs will have an existing performance or traffic monitoring utility used to monitor VNF behavior and report statistics, counters, etc.

To consume the performance and traffic related information provided within the scope of this project should in most cases be a logical extension of any existing VNF performance or traffic monitoring utility, in most cases it should not require a new utility to be developed. We do not see the Software Fastpath Service Quality Metrics data as major additional effort for VNFs to consume, this project would be sympathetic to existing VNF architecture constructs. The intention is that this project represents a lower level interface for network interface monitoring to be used by higher level fault management entities (see below).

Allowing the Software Fastpath Service Quality Metrics data to be handled within existing VNF performance or traffic monitoring utilities also makes it simpler for overall interfacing with higher level management components in the VIM, MANO and OSS/BSS. The Software Fastpath Service Quality Metrics proposal would be complementary to the Fault Management and Maintenance project proposal ("Doctor") which is also in flight, which addresses NFVI Fault Management support in the VIM. To that end, the project committers and contributors for the Software Fastpath Service Quality Metrics project wish to work in sync with the "Doctor" project – to facilitate this, one of the "Doctor" contributors has also been added as a contributor to the Software Fastpath Service Quality Metrics project.

CHAPTER

FOUR

MEASURING TELCO TRAFFIC AND PERFORMANCE KPIS

This section will look at what SFQM has done to enable Measuring Telco Traffic and Performance KPIs.

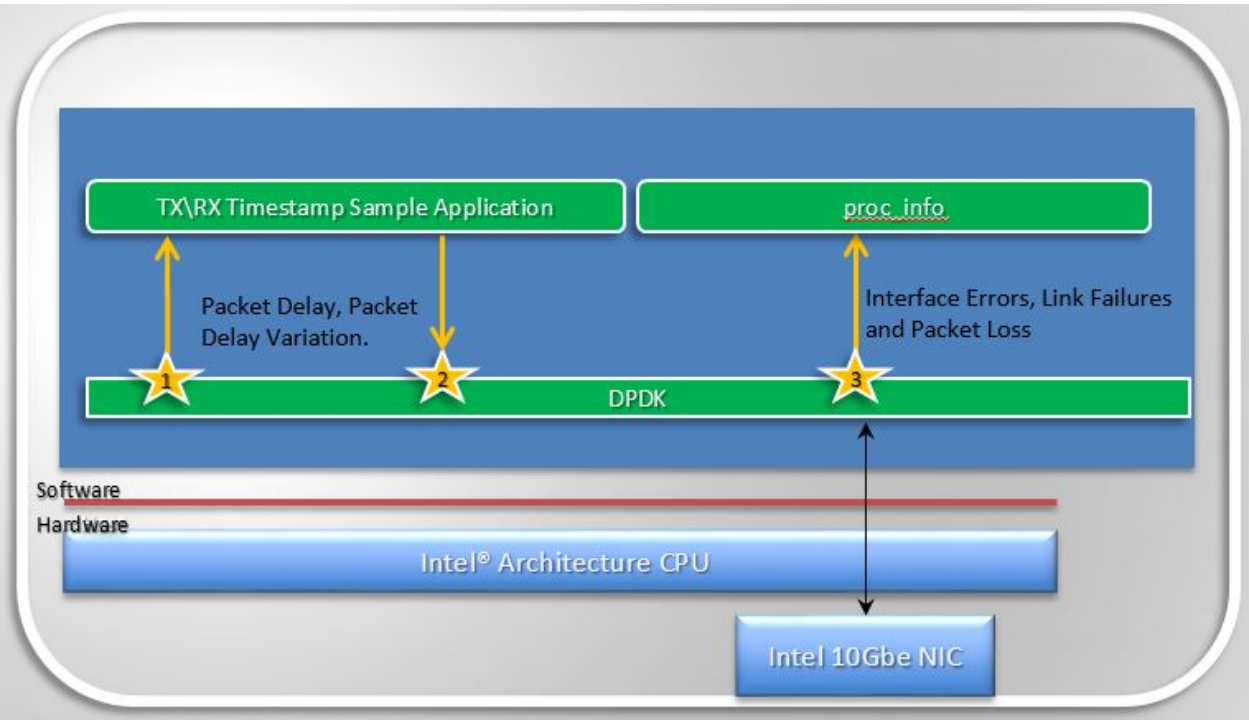

Fig. 4.1: Measuring Telco Traffic and Performance KPIs

- The very first thing SFQM enabled was a call-back API in DPDK and an associated application that used the API to demonstrate how to timestamp packets and measure packet latency in DPDK (the sample app is called rxtx_callbacks). This was upstreamed to DPDK 2.0 and is represented by the interfaces 1 and 2 in Figure 1.2.
- The second thing SFQM implemented in DPDK is the extended NIC statistics API, which exposes NIC stats including error stats to the DPDK user by reading the registers on the NIC. This is represented by interface 3 in Figure 1.2.
  - For DPDK 2.1 this API was only implemented for the ixgbe (10Gb) NIC driver, in association with a sample application that runs as a DPDK secondary process and retrieves the extended NIC stats.
  - For DPDK 2.2 the API was implemented for igb, i40e and all the Virtual Functions (VFs) for all drivers.

CHAPTER

FIVE

MONITORING DPDK INTERFACES

With the features SFQM enabled in DPDK to enable measuring Telco traffic and performance KPIs, we can now retrieve NIC statistics including error stats and relay them to a DPDK user. The next step is to enable monitoring of the DPDK interfaces based on the stats that we are retrieving from the NICs, and relay the information to a higher level Fault Management entity. To enable this SFQM has been enabling a number of plugins for collectd.

collectd is is a daemon which collects system performance statistics periodically and provides mechanisms to store the values in a variety of ways. It supports more than 90 different plugins to retrieve platform information, such as CPU utilization, and is capable of publishing/writing the information is gathers to a number of endpoints through its write plugins.

SFQM has been enabling two collectd plugins to collect DPDK NIC statistics and push the stats to Ceilometer:

- dpdkstat plugin: A read plugin that retrieve stats from the DPDK extended NIC stats API.
- ceilometer plugin: A write plugin that pushes the retrieved stats to Ceilometer. It's capable of pushing any stats read through collectd to Ceilometer, not just the DPDK stats.

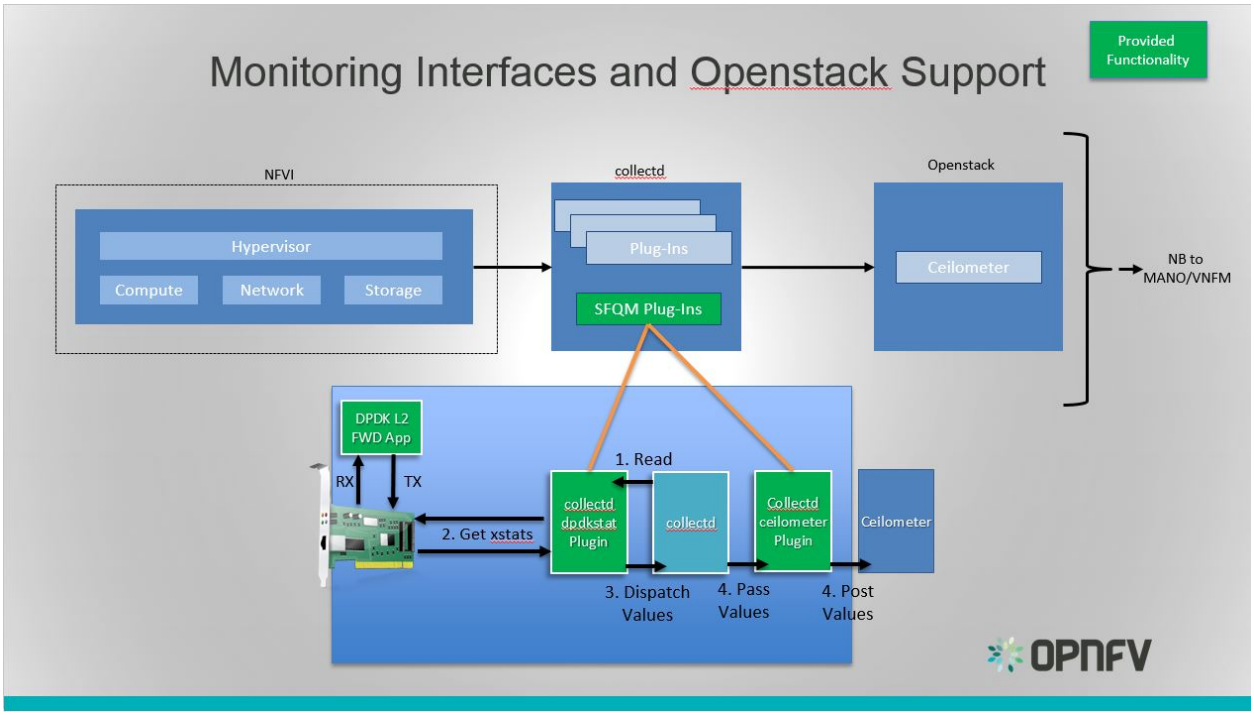

Fig. 5.1: Monitoring Interfaces and Openstack Support

The figure above shows the DPDK L2 forwarding application running on a compute node, sending and receiving

traffic. collectd is also running on this compute node retrieving the stats periodically from DPDK through the dpdkstat plugin and publishing the retrieved stats to Ceilometer through the ceilometer plugin.

To see this demo in action please checkout: SFQM OPNFV Summit demo

Future enahancements to the DPDK stats plugin include:

- Integration of DPDK Keep Alive functionality.
- Implementation of the ability to retrieve link status.

CHAPTER

SIX

DPDK KEEP ALIVE OVERVIEW

SFQM aims to enable fault detection within DPDK, the very first feature to meet this goal is the DPDK Keep Alive Sample app that is part of DPDK 2.2.

DPDK Keep Alive or KA is a sample application that acts as a heartbeat/watchdog for DPDK packet processing cores, to detect application thread failure. The application supports the detection of 'failed' DPDK cores and notification to a HA/SA middleware. The purpose is to detect Packet Processing Core fails (e.g. infinite loop) and ensure the failure of the core does not result in a fault that is not detectable by a management entity.

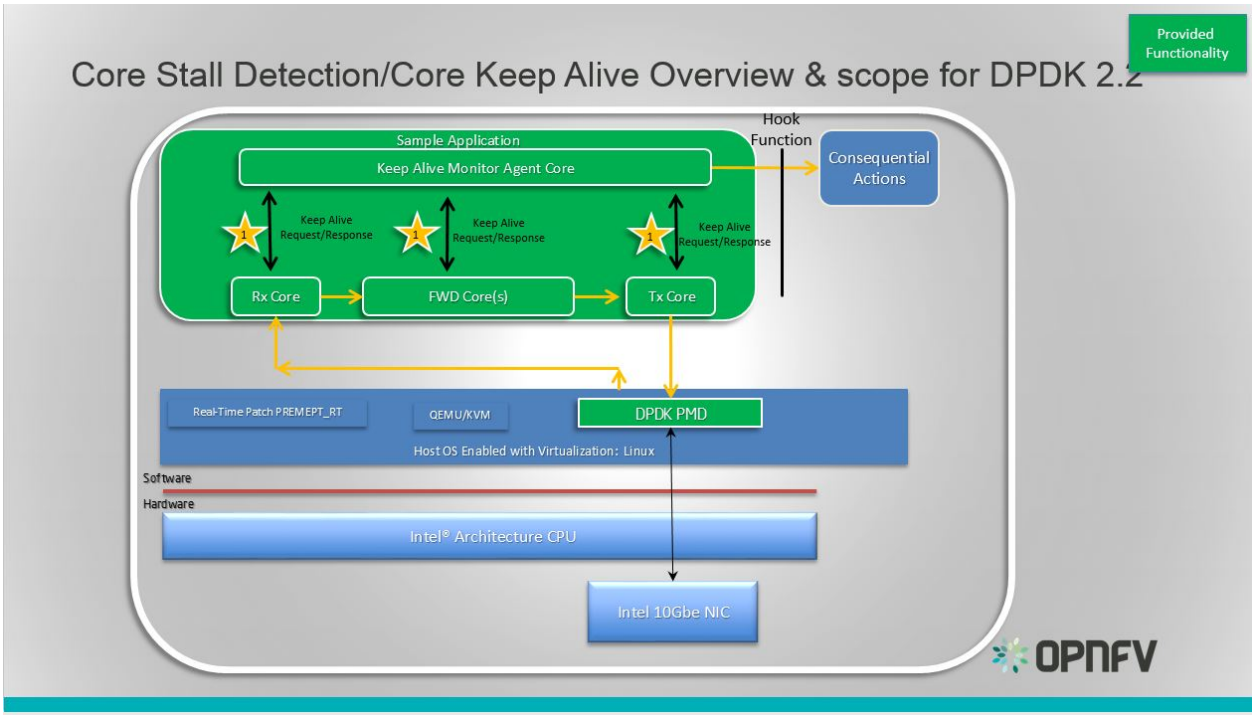

Fig. 6.1: DPDK Keep Alive Sample Application

Essentially the app demonstrates how to detect 'silent outages' on DPDK packet processing cores. The application can be decomposed into two specific parts: detection and notification.

- The detection period is programmable/configurable but defaults to 5ms if no timeout is specified.
- The Notification support is enabled by simply having a hook function that where this can be 'call back support' for a fault management application with a compliant heartbeat mechanism.

CHAPTER

SEVEN

DPDK KEEP ALIVE SAMPLE APP INTERNALS

This section provides some explanation of the The Keep-Alive/'Liveliness' conceptual scheme as well as the DPDK Keep Alive App. The initialization and run-time paths are very similar to those of the L2 forwarding application (see L2 Forwarding Sample Application (in Real and Virtualized Environments) for more information).

There are two types of cores: a Keep Alive Monitor Agent Core (master DPDK core) and Worker cores (Tx/Rx/Forwarding cores). The Keep Alive Monitor Agent Core will supervise worker cores and report any failure (2 successive missed pings). The Keep-Alive/'Liveliness' conceptual scheme is:

- DPDK worker cores mark their liveliness as they forward traffic.
- A Keep Alive Monitor Agent Core runs a function every N Milliseconds to inspect worker core liveliness.
- If keep-alive agent detects time-outs, it notifies the fault management entity through a call-back function.

Note: Only the worker cores state is monitored. There is no mechanism or agent to monitor the Keep Alive Monitor Agent Core.

CHAPTER

EIGHT

DPDK KEEP ALIVE SAMPLE APP CODE INTERNALS

The following section provides some explanation of the code aspects that are specific to the Keep Alive sample application.

The heartbeat functionality is initialized with a struct rte_heartbeat and the callback function to invoke in the case of a timeout.

```
rte_global_keepalive_info = rte_keepalive_create(&dead_core, NULL);
if (rte_global_hbeat_info == NULL)
    rte_exit(EXIT_FAILURE, "keepalive_create() failed");
```

The function that issues the pings hbeat_dispatch_pings() is configured to run every check_period milliseconds.

```
if (rte_timer_reset(&hb_timer,
        (check_period * rte_get_timer_hz()) / 1000,
        PERIODICAL,
        rte_lcore_id(),
        &hbeat_dispatch_pings, rte_global_keepalive_info
        ) != 0 )
    rte_exit(EXIT_FAILURE, "Keepalive setup failure.\n");
```

The rest of the initialization and run-time path follows the same paths as the the L2 forwarding application. The only addition to the main processing loop is the mark alive functionality and the example random failures.

```
rte_keepalive_mark_alive(&rte_global_hbeat_info);
cur_tsc = rte_rdtsc();

/* Die randomly within 7 secs for demo purposes.. */
if (cur_tsc - tsc_initial > tsc_lifetime)
break;
```

The rte_keepalive_mark_alive() function simply sets the core state to alive.

```
static inline void
rte_keepalive_mark_alive(struct rte_heartbeat *keepcfg)
{
    keepcfg->state_flags[rte_lcore_id()] = 1;
}
```

Keep Alive Monitor Agent Core Monitoring Options The application can run on either a host or a guest. As such there are a number of options for monitoring the Keep Alive Monitor Agent Core through a Local Agent on the compute node:

Application Location	DPDK KA	LOCAL AGENT
HOST	X	HOST/GUEST
GUEST	X	HOST/GUEST

For the first implementation of a Local Agent SFQM will enable:

Application Location	DPDK KA	LOCAL AGENT
HOST	X	HOST

Through extending the dpdkstat plugin for collectd with KA functionality, and integrating the extended plugin with Monasca for high performing, resilient, and scalable fault detection.