Data Plane Acceleration Use-cases

Table of Contents

1 Introduction

As a result of convergence of various traffic types and increasing data rates, the performance requirements (both in terms of bandwidth and real-time-ness)on data plane devices within network infrastructure have been growing at significantly higher rates than in the past. As the traditional ‘bump-in-the-wire’ network functions evolve to a

virtualized paradigm with NFV,the focus will be even higher to deliver high performance within very competitive cost envelopes.

At the same time, application developers have, in some cases, taken advantage of various hardware and software acceleration capabilities, many of which are platform supplier dependent. There is a clear impetus to move away from proprietary data plane interfaces, in favor of more standardized interfaces to leveraging the data plane capability of underlying platforms –whether using specialized hardware accelerators or general purpose CPUs.

Therefore,a requirement project “dpacc” at OPNFV is formed to specify a general framework for VNF data plane acceleration (or DPA for short), including a common suite of abstract APIs at various OPNFV interfaces, to enable VNF portability and resource management across various underlying integrated SOCs that may include hardware accelerators or standard high volume (or SHV) server platforms that may include attached hardware accelerators.

This document is a work item of dpacc project to document usecases of the dpacc framework (https://etherpad.opnfv.org/p/dpacc_framework). For each use-case in scope, it elaborates in more detail regarding the application scenario, operation procedure instantiation and how to benchmark the performance.

For more information regarding the project, please refer to https://wiki.opnfv.org/dpacc/

Terminology

RAN	Radio Access Network
RRU	Remote Radio Unit
eNB	E-UTRAN NodeB
UE	User Equipment
L1/2/3	Layer 1/2/3
DL	Downlink
UL	Uplink
IP	Internet Protocol
LTE	Long Term Evolution
MAC	Medium Access Control
RLC	Radio Link Control
PDCP	Packet Data Convergence Protocol
RRC	Radio Resource Control
FFT	Fast Fourier Transform
iFFT	Inverse Fast Fourier Transform
DFT	Discrete Fourier Transform
iDFT	Inverse Discrete Fourier Transform
UL	Uplink

2 Acceleration Models

Dpacc considers 3 kinds of acceleration models, namely **Lookaside**, **Offload** and **External**.

It is noted that although different dpacc application scenarios can be implemented with one or several of the above acceleration models, the choice of acceleration models are not transparent to vNFs, and the portability of VNFs across different acceleration models are currently out of scope for dpacc.

2.1 Lookaside Model

The packet arrives at the vNF which offloads compute intensive operations to an accelerator by sending it data and receiving results back from the accelerator, the vNF then completes packet processing and transmits the resulting packet.

There can be several examples of lookaside accelerator functions such as Crypto, Channel encoding/decoding, PDCP, IPsec-header processing, SSL-Header processing, PME (Pattern Matching) and so on.

2.2 Offload Model (Inline Model)

In this model, the accelerator is in the packet data path at ingress and/or egress of the packet from the vNF. The VNF application typically pushes all the states to the offload accelerator and allows the accelerator to perform the packet processing in an autonomous fashion.

This model is also referred to as **cut-through model** or **inline model**. In this case the VNF application expects to process few packets such as exception packets.

Examples of offload accelerator functions include Firewall, SLB, NAT, IPsec, FFT/iFFT etc.

2.3 External Model (Data Path Offload Model)

The accelerator implements a complete data path with switching, flow manipulation and packet processing that completely offloads the data path processing from the vNF, vNF only configures the data plane with the desired actions to be performed.

3 Virtualized Acceleration Interfaces

As stated in Dpacc Architecture, there are two alternative interfaces for data path for packets in and out of vNFs:

- sio: short for software I/O interface, e.g. VirtIO to the underlying SW/HW, which enables binary compatibility with paravirtualized drivers and is optional for hio-only requirements.
- hio: short for Hardware I/O interface (e.g. SR-IOV, SoC-specific interfaces, etc.), which is referring to some type of pass-through design with support for virtualization and is optional for sio-only deployments.

Note: Pass-through access cannot achieve binary compatibility. It is offered to VNFs as an option to ensure optimal performance.Since binary compatibility is a “Should” requirement, there may be cases where this tradeoff is acceptable.

The functional abstraction layer (i.e. g-API over SAL) framework and architecture should support both, and provide a unified interface to the upper VNF. It will be the person configuring the VNF, hypervisor/host OS and hardware (policies) that decides which model to use.

In this following, Virtio and Passthrough will be used as exemplary implementations for sio and hio, respectively.

3.1 Passthrough-based Acceleration

In this “pass-through” model, the VNF is making use of a common suite of DPA APIs in discovering the hardware accelerators and/or generalized network interfaces (NWI) available and using the correct and specific “direct drivers” to directly access the allocated hardware resources. The features of this model include:

- It enables the most efficient use of hardware resources by bypassing the hypervisor/host OS, yielding higher performance than the other model.
- It cannot provide “absolute transparency” to the VNFs using hardware accelerators, as they have to upgrade to make changes to their VM image each time they are making use to a new type of hardware accelerator, to load the specific driver and make it known to the application.

Figure 1 shows a suggested implementation of standardized accelerator available for VNFs using Passthrough to gain direct access to accelerator devices, which is an instantiation of the general diagram for hio-enabled vNF acceleration in Dpacc Architecture.

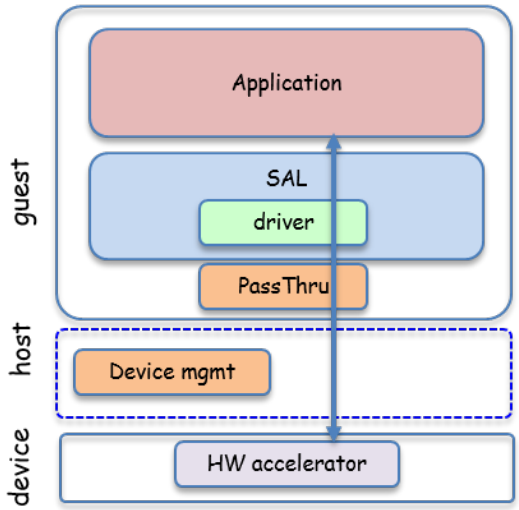


Figure 1 Passthrough-based Acceleration

3.2 Virtio-based Acceleration

Alternatively, there is the “fully intermediated” model where the VNF talks to a group of abstracted functional “synthetic drivers” or “frontend drivers”. These “frontend drivers” relays the call to a backend driver in the host that actually interacts with specific driver for the underlying HWA and/or NWI. The features of this model include:

- Through this intermediate layer in the host, a registration mechanism is possible for a new HWA to make them mapped to the backend driver and then be used automatically with no changes to the upper VNFs.
- Access control and/or resource scheduling mechanisms for HWA allocation to different VNFs can also be included in the hypervisor to enable flexible policies for operation considerations.

Figure 2 shows a suggested implementation of standardized accelerator available for VNFs using Virtio, which is an instantiation of the general diagram for sio-enabled vNF acceleration in Dpacc Architecture.

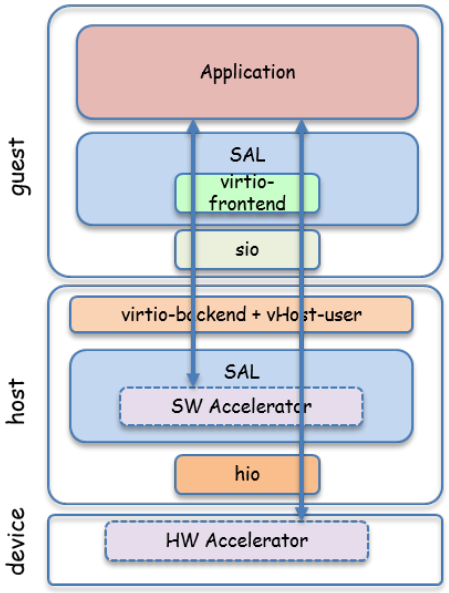


Figure 2 Virtio-based Acceleration

Two types of data paths are shown in the above figure, where the bi-directional arrow in the left representing the case where the software accelerator realized as part of the host SAL is utilized and the bi-directional arrow in the right representing the case where the hardware accelerator residing as part of the device is leveraged.

3.2.1 Virtio-lookaside Acceleration

Figure 3 shows a suggested implementation of standardized lookaside accelerator available for VNFs using Virtio. Under the Virtio-lookaside model (umbrella of drivers), VNF can access several lookaside accelerator functions such as IPsec, Crypto, PME, DCE etc.

VNF applications in the Guest User space can make use of the SAL interface implementing Virtio-Accelerator specific frontend drivers to access the underlying SW accelerator in the host or the HW accelerator in the device.

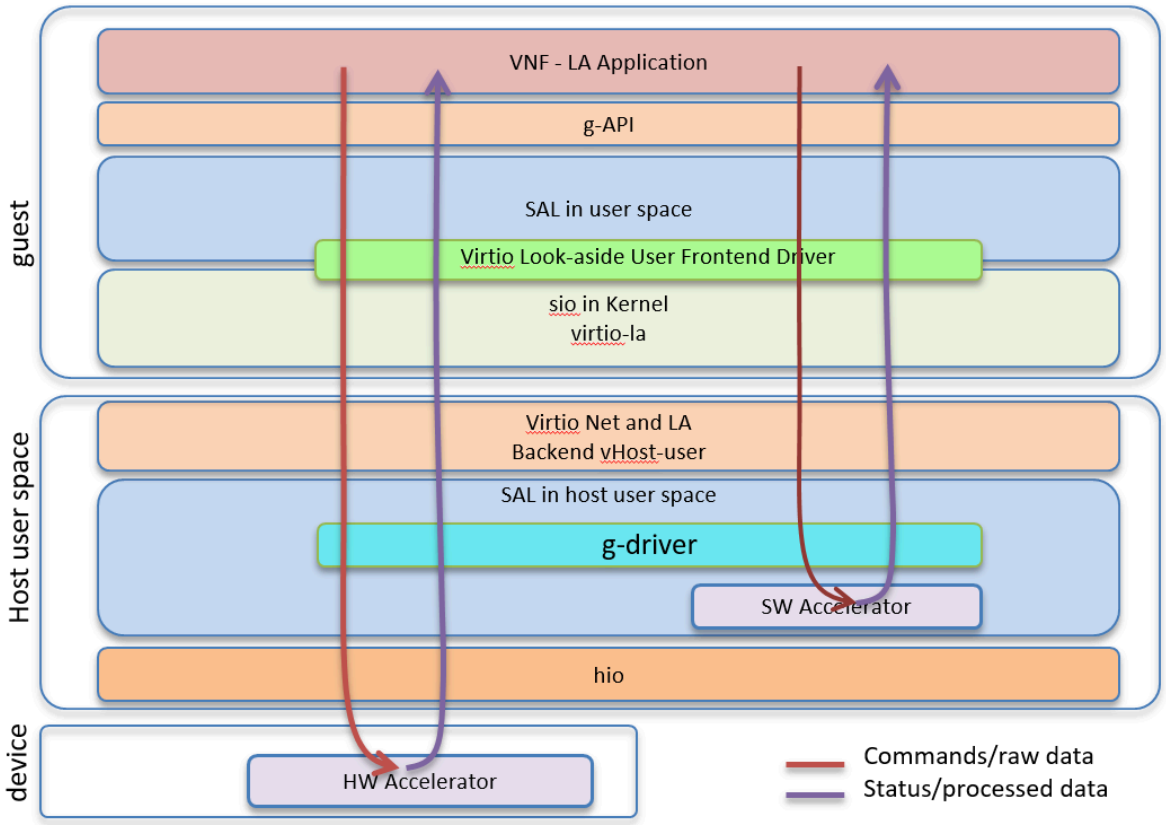


Figure 3 Virtio-Lookaside Acceleration

Figure 4 shows some typical examples of Virtio-lookaside function specific drivers enabling a VNF to access underlying hardware accelerators in a hardware agnostic fashion, where multiple virtualized virtio-based acceleration functionalities can be conducted by a single physical device through a common g-driver in the host SAL.

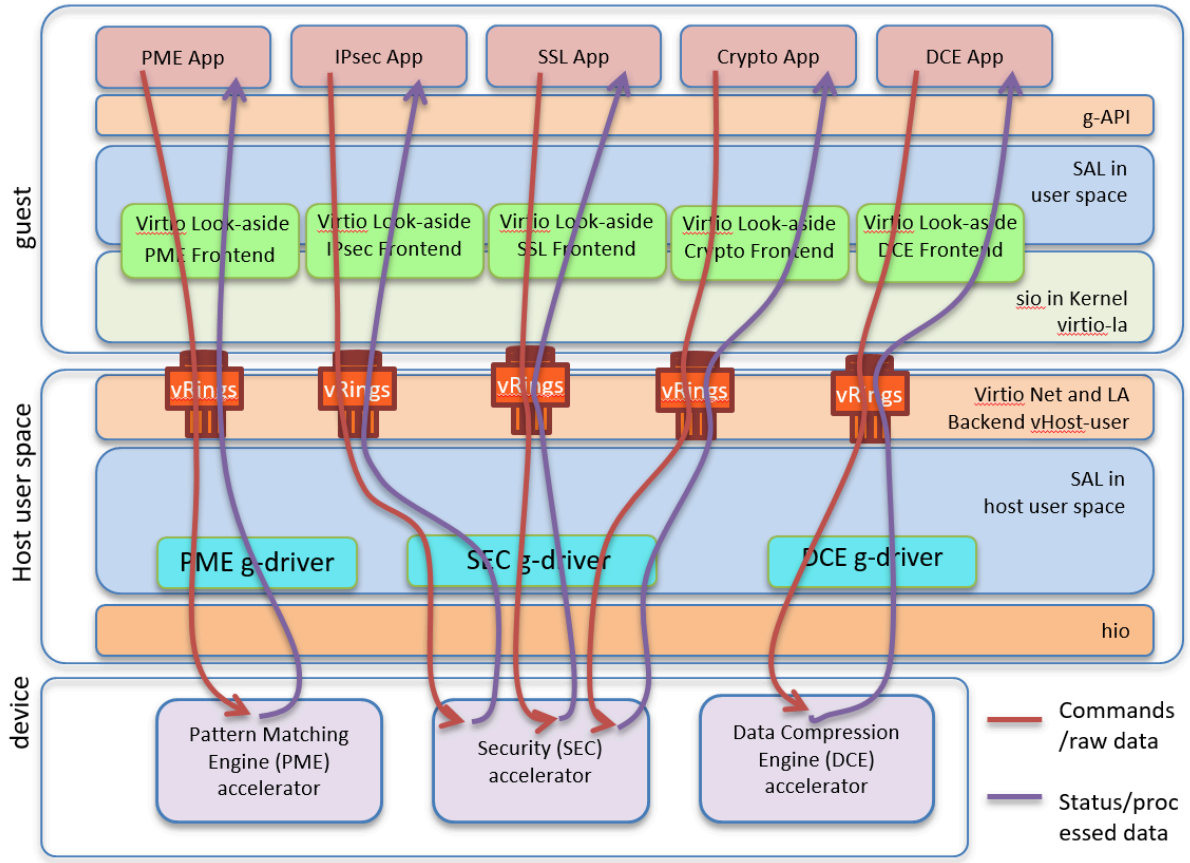


Figure 4 Exemplary Functions for Virtio-based Lookaside Acceleration

Several Virtio-accelerator function drivers (LA or LookAside model) are shown in the picture – namely Virtio LA Crypto (for Crypto operations), Virtio LA IPsec (for IPsec level acceleration), Virtio LA SSL (for SSL level operations), and Virtio LA PME (for pattern matching acceleration) and Virtio LA DCE (for compression and de-compression operations).

The backend would include vendor specific translators that translate the generic virtio messages to vendor specific messages, hence enabling the VNFs access to the underlying accelerators.

3.2.2 virtio-inline acceleration

Figure 5 shows the virtio-inline/offload accelerator interface. Under the Virtio-offload model (umbrella of drivers), the VNF can access several offload functions such as Firewall Offload function, SLB-NAT Offload function, IPSec Offload function etc.

VNF applications in the Guest User space can make use of the SAL interface implementing Virtio-Accelerator specific frontend drivers to access the underlying hardware accelerator. VNF Applications residing the Guest Kernel space can make use of the virtio frontend driver in the kernel level to access the underlying hardware accelerator.

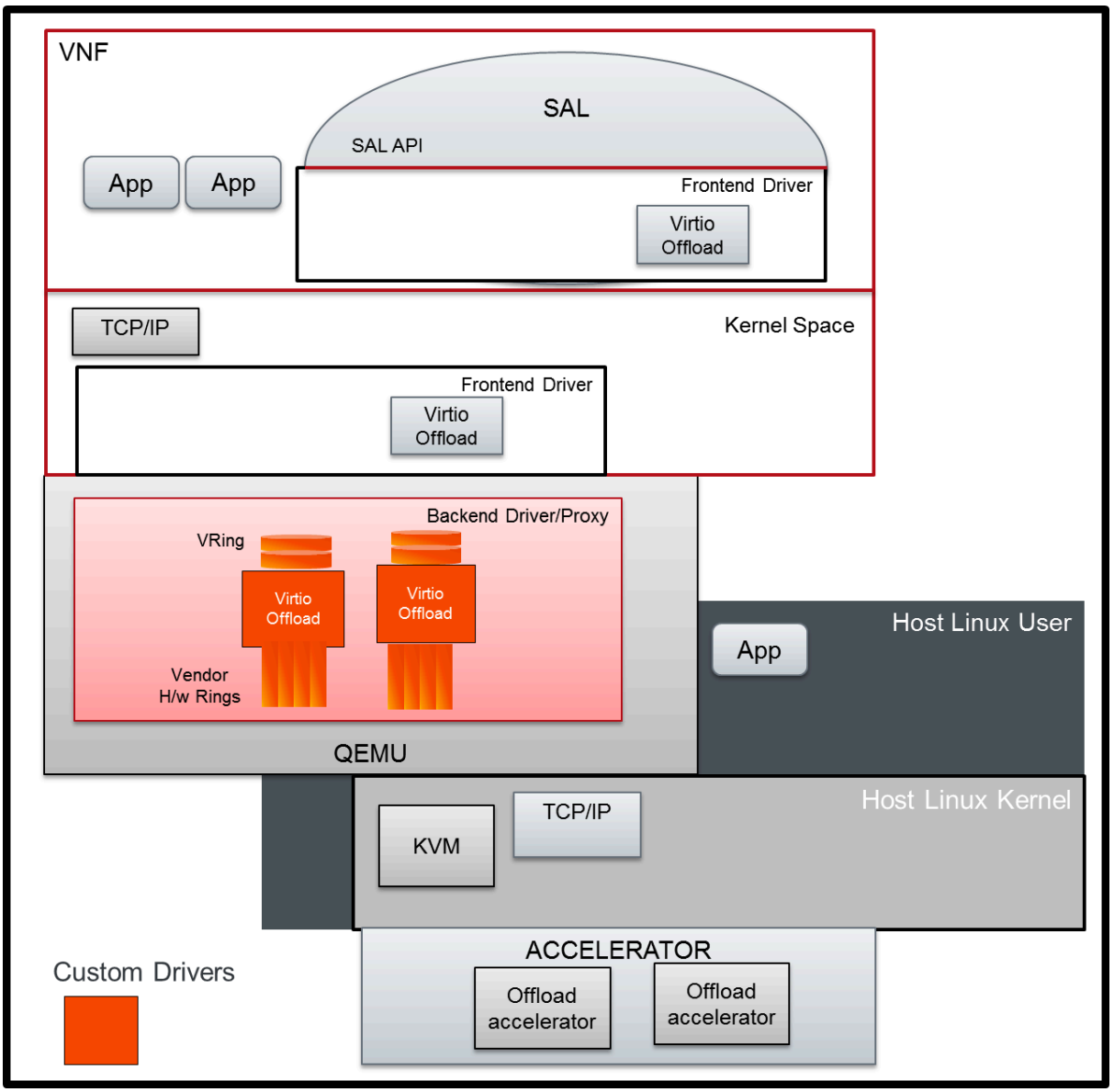


Figure 5 Virtio-based inline/offload Acceleration

Figure 6 shows a suggested implementation of standardized offload model interfaces available to VNFs using Virt-IO drivers.

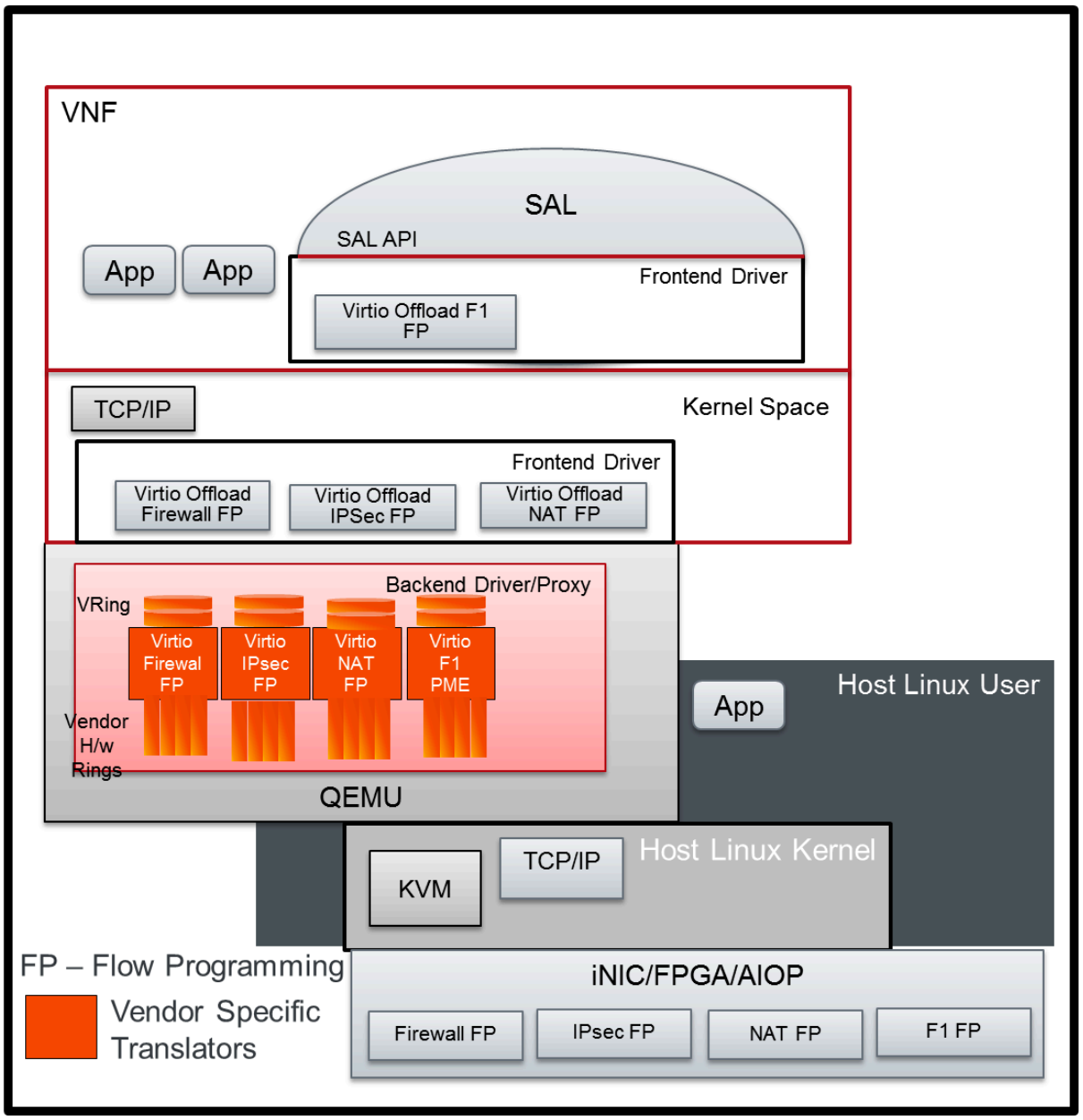


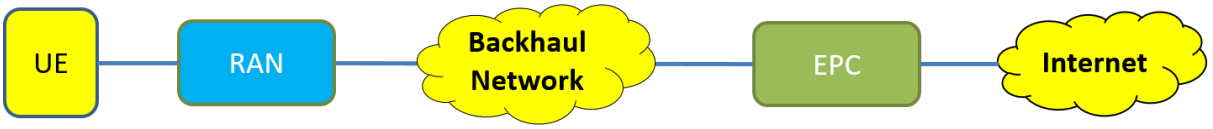
Figure 6 Exemplary Acceleration for Virtio-inline Acceleration

Several virtio-offload function drivers are shown in the picture – namely Virtio Firewall flow programming, Virtio IPsec Flow Programming, Virtio NAT flow programming etc.

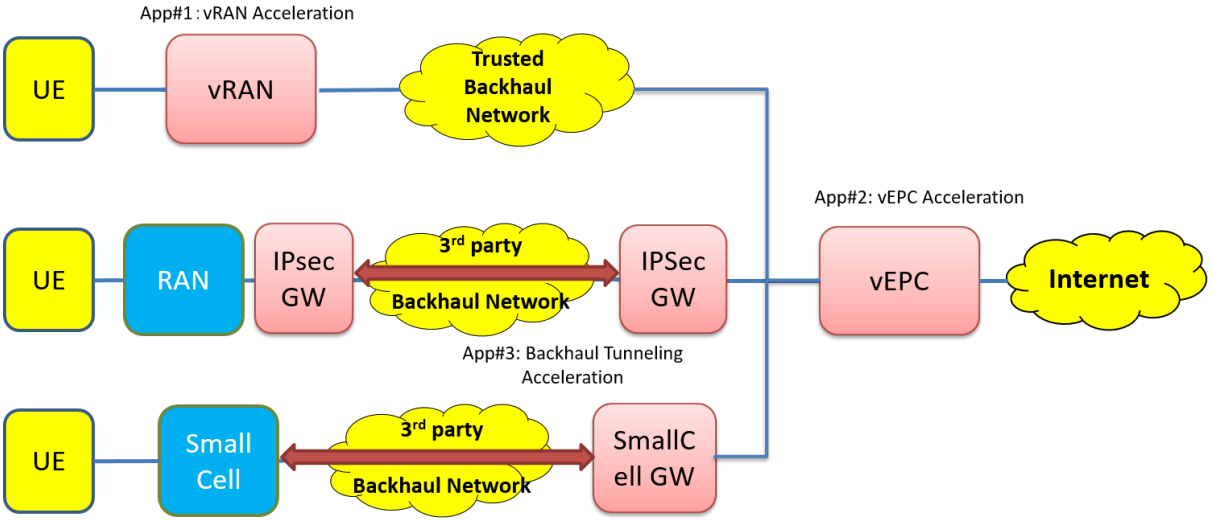
The backend would include vendor specific translators that translate the generic virtio messages to vendor specific messages, hence enabling the VNFs access to the underlying accelerators.

4 Targeting Applications

[Editor Note] description of the targeted application scenario (e.g. small cell GW VNF), including the performance metrics and benchmarking methodology.

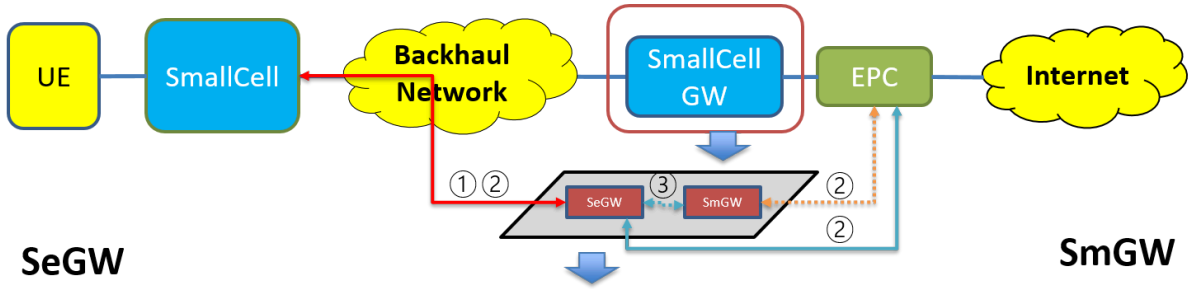


The above figure is depicting various segments of the mobile network, which can be further refined into the following diagram, differentiating different DPACC applications on the end-to-end traffic path in a mobile network case.



In the above figure, three main applications are identified: vRAN acceleration, Backhaul Tunneling acceleration and vEPC acceleration.

4.1 SmallCell Gateway



SeGW

Authentication: realize mutual authentication between small cell and GW.

Security Protection: establish IPSec tunnels between small cell and GW.

QoS Inheritance: copies the inner IP ToS/DSCP tags onto the outerIP header during encapsulation.

Performance Factors

① I/O packet processing
② IPSEC performance
③ vSwitch performance

Performance Metrics

Throughput(Mbps, Mpps)
Concurrency(tunnels#)
Latency(tunnel setup time)

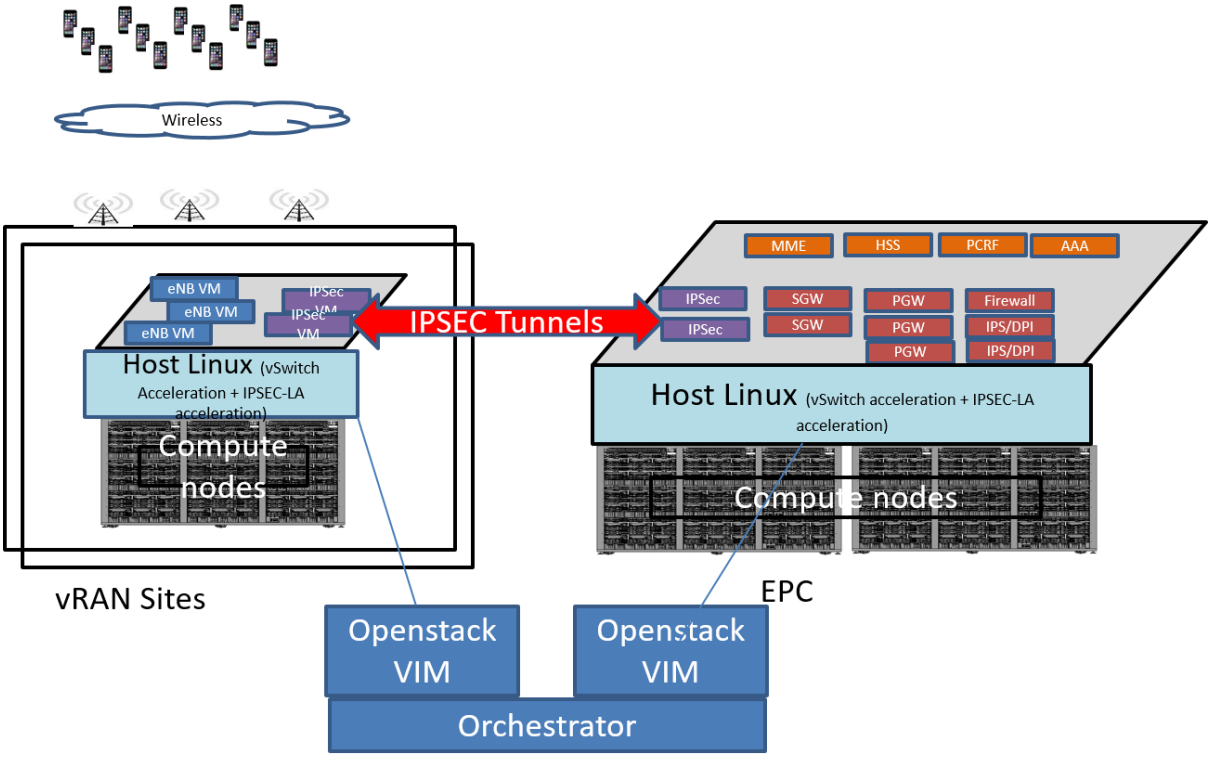
SmGW

Signaling Routing: selects a proper MME for an attaching UE.

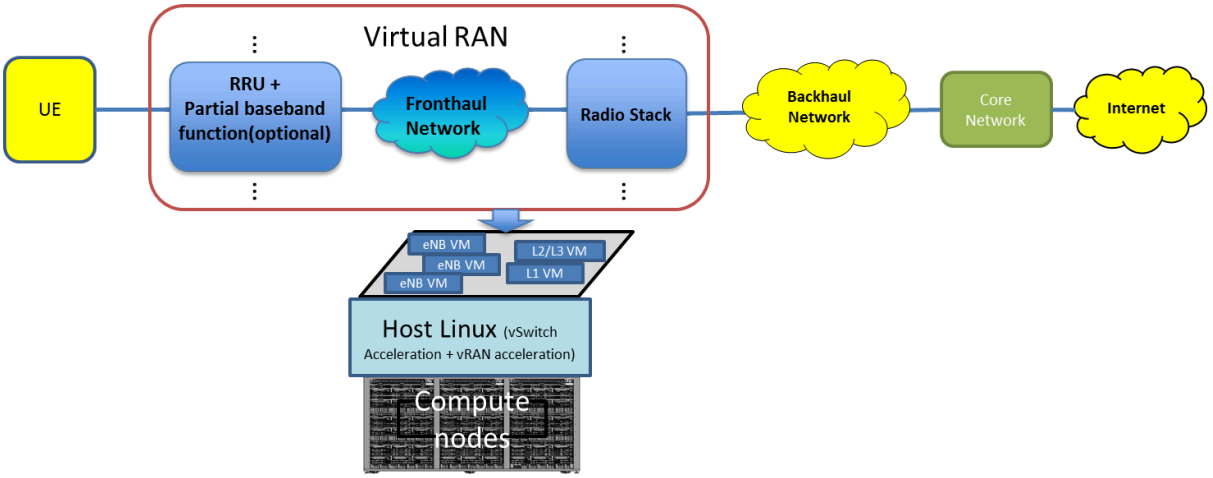
Signaling Pooling: pools the interfaces to MME for a large group of small cells.

Optional GTP Termination: terminate the GTP tunneling for packets before EPC.

4.2 IPSec between vRAN and vEPC



4.3 VRAN



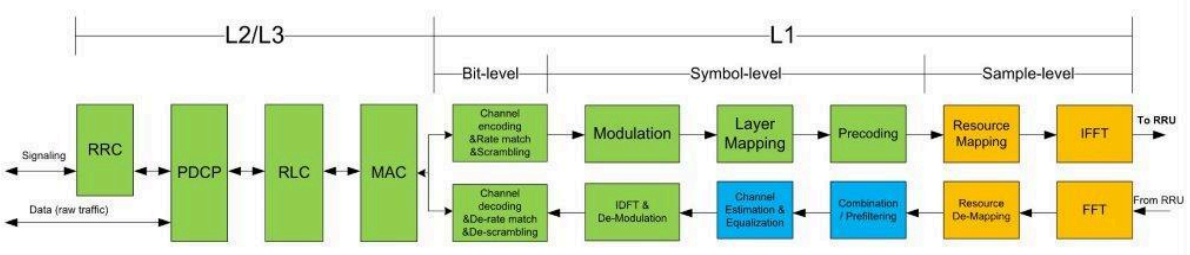
As shown in the previous figure, with the evolution of RAN, there is a wide consensus that future radio access network should be a virtual RAN (vRAN) with the benefits of flexibility, quicker time-to-market, unified management and flourishing applications. In vRAN, the main part of baseband radio stack is deployed on the virtual platform.
Currently, there are two main vRAN VM design schemes.

- One is that all the radio access network protocol functions (including L1, L2, L3) are in a eNB VM.

- The other is that different layers of radio access network protocol are in different VMs, such as L2/L3 VM, L1 VM, etc. The functions of L2/L3 VM may include MAC, RLC, PDCP, RRC. The functions of L1 VM may include bit-level processing (e.g. channel edcoing/decoding), symbol-level processing(e.g. modulation/de-modulation) and sample-level processing (FFT/iFFT).

Based on the evaluation of RAN processing, vRAN accelerator is necessary in order to improve the efficiency of vRAN. vRAN accelerator mainly refers to the compute-intensive functions acceleration.

As shown in the following figure, taking LTE protocol stack as an example, the compute-intensive functions includes PDCP ciphering/deciphering, PDCP Robust Header Compression, channel encoding/decoding and FFT/iFFT, etc.



What's more, vRAN accelerator can be divided into two classes according to the characteristics of load.

- First, it is radio load dependent vRAN accelerator, such as PDCP accelerator, channel encoding/decoding accelerator. Different carriers/cells can share the processing capacity of the vRAN accelerator. The packet arrives at the vNF which offloads PDCP ciphering/deciphering or PDCP robust header compression or channel encoding/decoding to an accelerator by sending it data and receiving results back from the accelerator, the vNF then completes packet processing and transmits the resulting packet.
- Second, it is radio load independent vRAN accelerator, such as FFT/iFFT accelerator. FFT accelerator is in the packet data path at ingress of the packet from the vNF. Accordingly, iFFT accelerator is in the packet data path at egress of the packet from the vNF. The VNF application typically pushes all the states to the offload FFT/iFFT accelerator and allows the accelerator to perform the packet processing in an autonomous fashion.

The performance metrics of vRAN accelerator mainly include throughput and latency. For the numerical requirements, it depends on RAN protocol parameters configuration and the functions of accelerator.

4.4 Summary

4.4.1 IPsec Packet Processing Acceleration

Method 1: Crypto Look Aside Offload
Method 2: Crypto acceleration using native instructions

Method 3: IPSec Look Aside Offload
Method 4: IPSec Inline/fast-path acceleration

4.4.2 NFVI Packet Processing Acceleration

Method 1: Ingress/Outgress Inline
Method 2: OVS Offload

4.4.3 PDCP Packet Processing Acceleration

Method 1: PDCP Look Aside Offload

5 Use-cases

5.1 IPsec Look Aside Accelerator

Figure X shows the flow of packets when IPsec Look aside accelerator is used. F1 and F2 stand for other packet processing functions that is implemented in the vNF, such as Firewall, NAT etc.

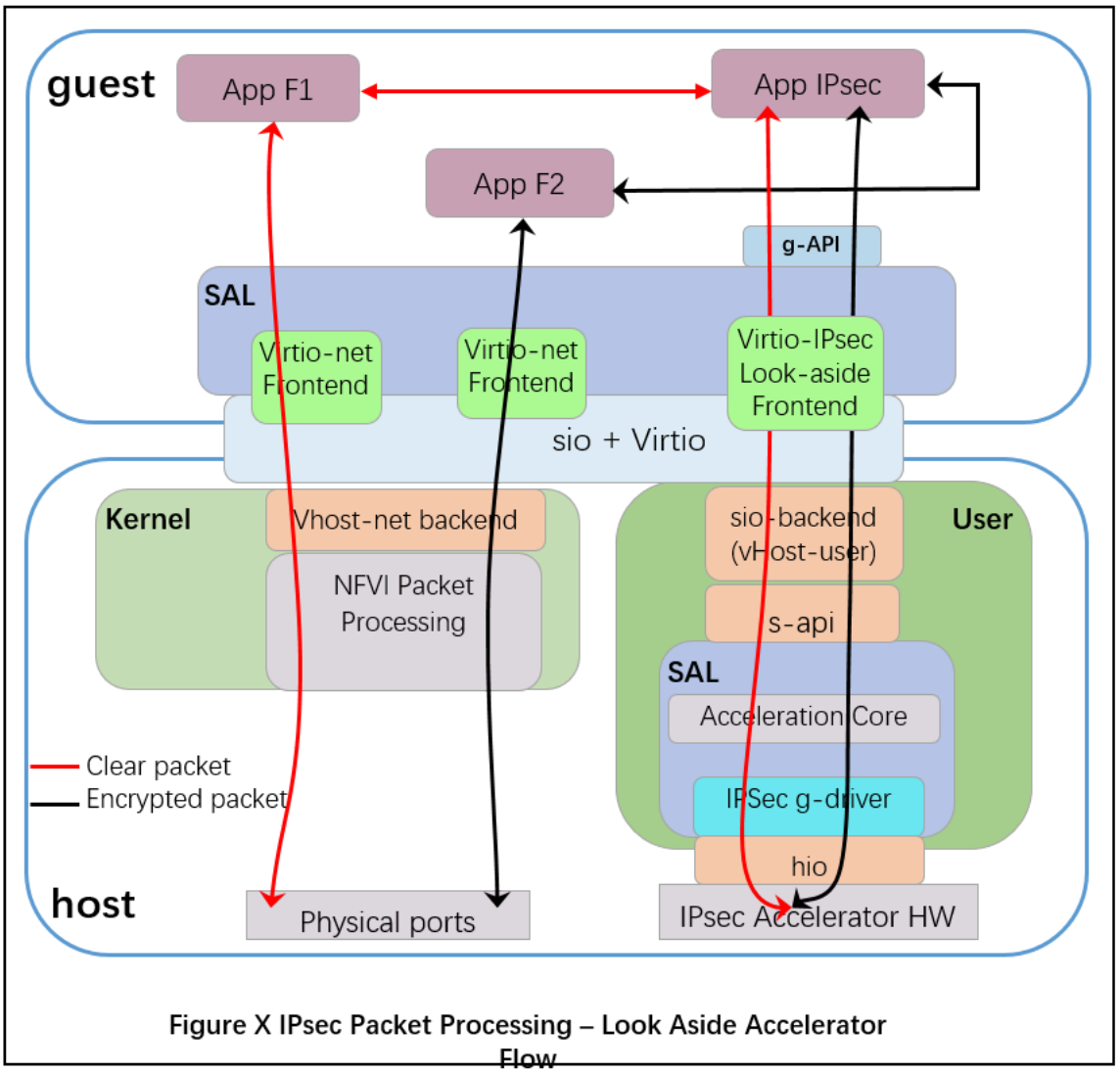


Figure X IPsec Packet Processing – Look Aside Accelerator Flow

Figure X IPsec Packet Processing –Look Aside Accelerator Flow

Ingress Packet Flow:

- Packets processed by VXLAN/VLAN, OVS Data Path, IPTables, Vhost-Net.
- Packet announced to VNF through Virtio-Net driver.
- Packets under several function processing such as Firewall etc.
- Packets arrive at the IPsec module for IPsec Packet Processing.
- As packets are submitted by the IPsec Module to the Virtio-IPsec Frontend driver, the buffers are put in the Virtio Descriptor Vrings or Virt Qs to be transferred to the Virtio-IPsec Backend.
- The Virtio IPSec Backend is responsible for translating the packets from Virt Q Descriptor to the actual hardware accelerator in a message that the accelerator understands and vice-versa.

- The Virtio IPsec Backend is also responsible for picking up processed packets from the hardware accelerator, updating the VirtQ rings and notifying the Guest VNF.

- The processed packets under further processing functions (F2 etc.) before being sent out through the Virtio-net interface.

5.2 NFVI Packet Processing Offload Accelerator

Figure Y shows the packet flow for a NFVI Packet Processing Accelerator.

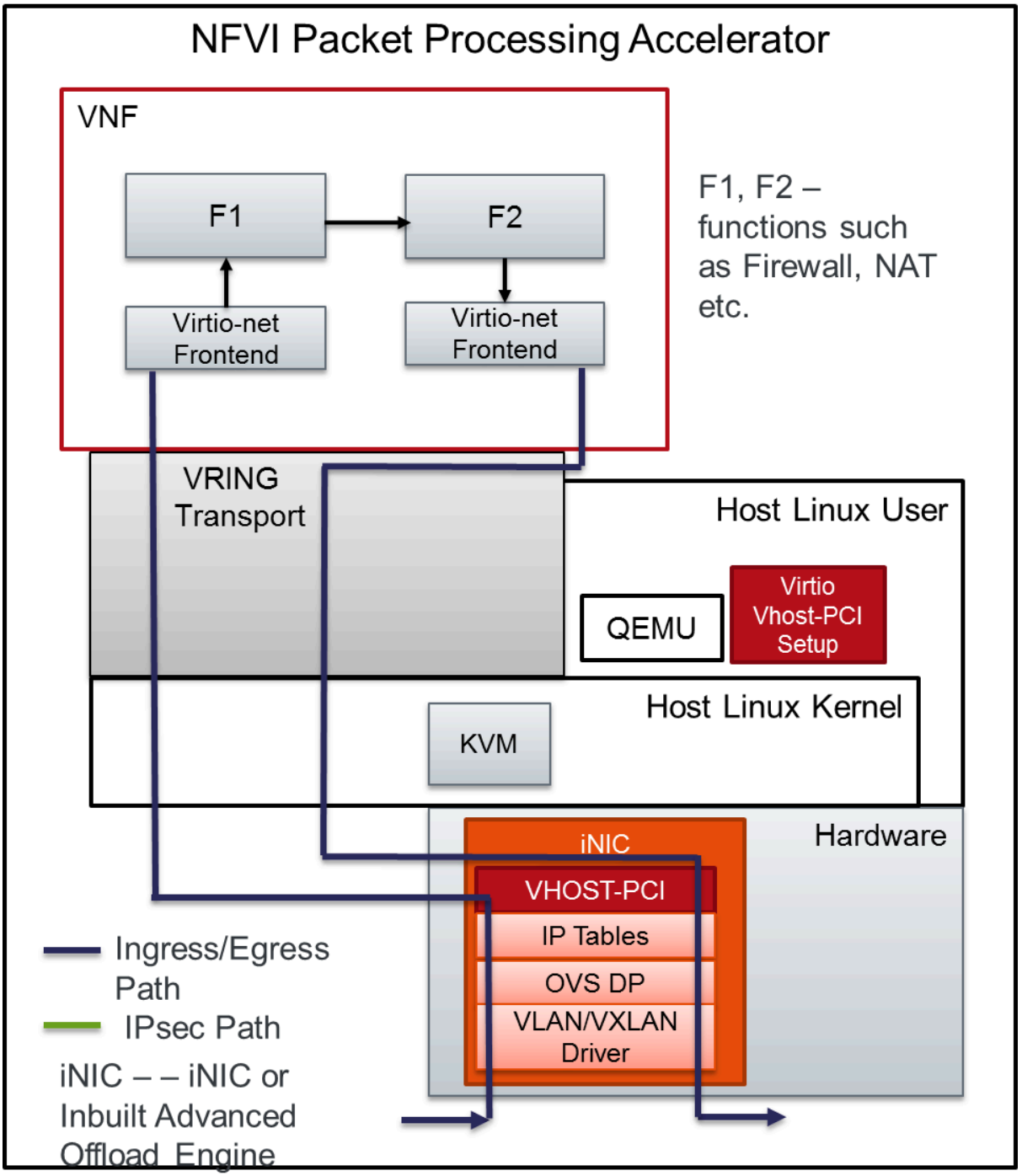


Figure Y NFVI Packet Processing Accelerator

The packet flow in this case is as follows:

- VXLAN/VLAN, OVS Data Plane and IP Tables processing are handled by the Intelligent NIC (iNIC). The Vhost-PCI backend presents the packets to the VNF using the Virtio-net interface.

- Packets that need to be transmitted out are submitted through the Virtio-Interface. The Vhost-PCI backend handles the packet, does the necessary processing before sending the packet out.

5.3 Combined NFVI Inline and IPsec LA Accelerator

Figure Z shows the packet flow of a combined case of NFVI inline Acceleration and IPSec Look Aside Acceleration.

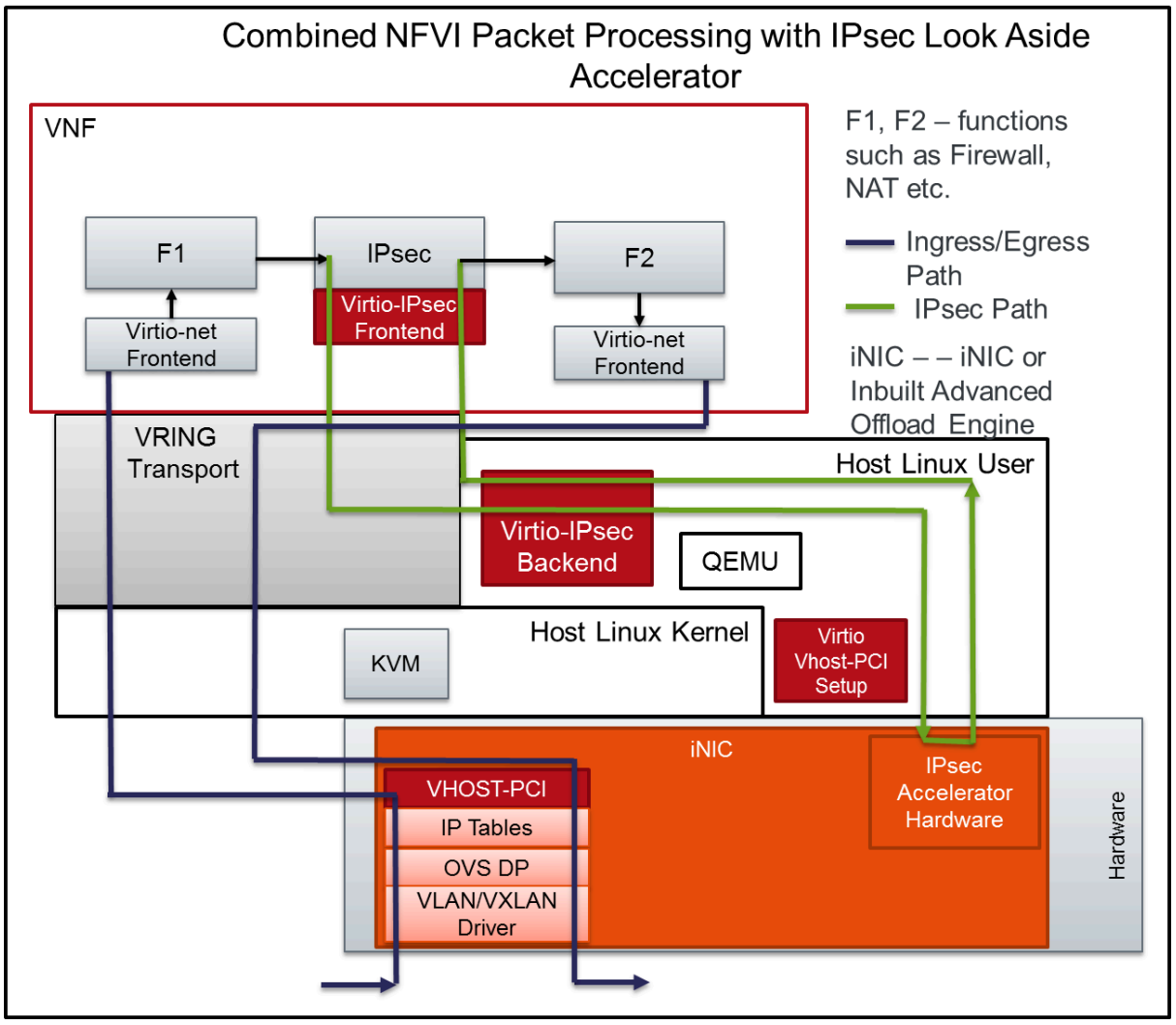


Figure Z Combined NFVI Packet Processing with IPsec Look Aside Accelerator

5.4 NFVI and IPsec Offload Accelerator

Complete Offload – Packet flow

Figure 7 shows the packet flow, where the VNF is able to set up flow programming in the iNIC to apply IPsec Processing on specific flows. As a result of this, for packets matching the programmed flows, packet processing happens in the iNIC itself.

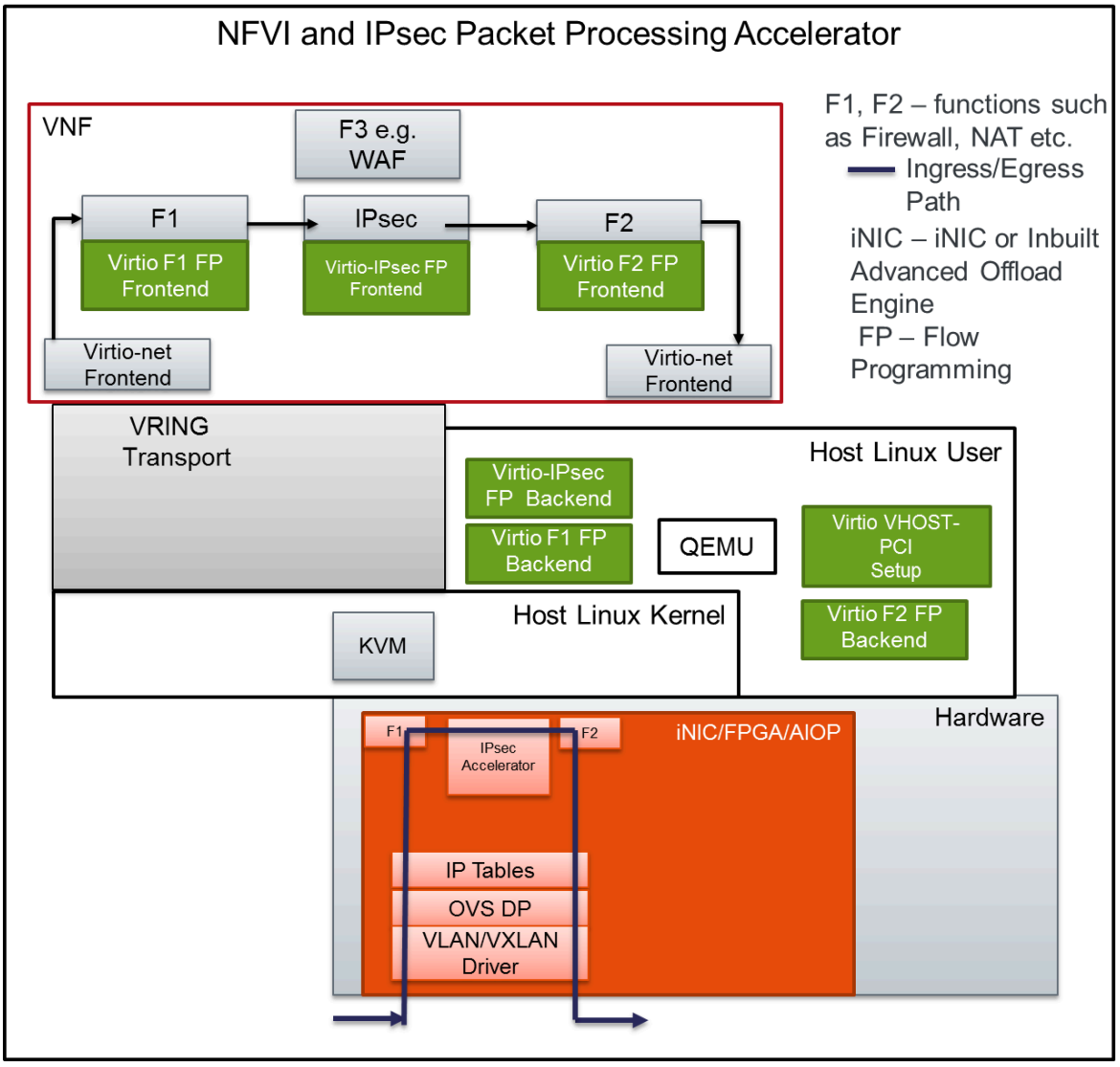


Figure 7 NFVI IPsec Packet Processing Accelerator – Complete Offload Data Path Flow

Inline Offload – Packet flow

Figure W shows the packet flow, in the case, where the packets failing to meet the set criteria for complete offload, have to be processed by the VNF, e.g. TCP port 80 traffic.

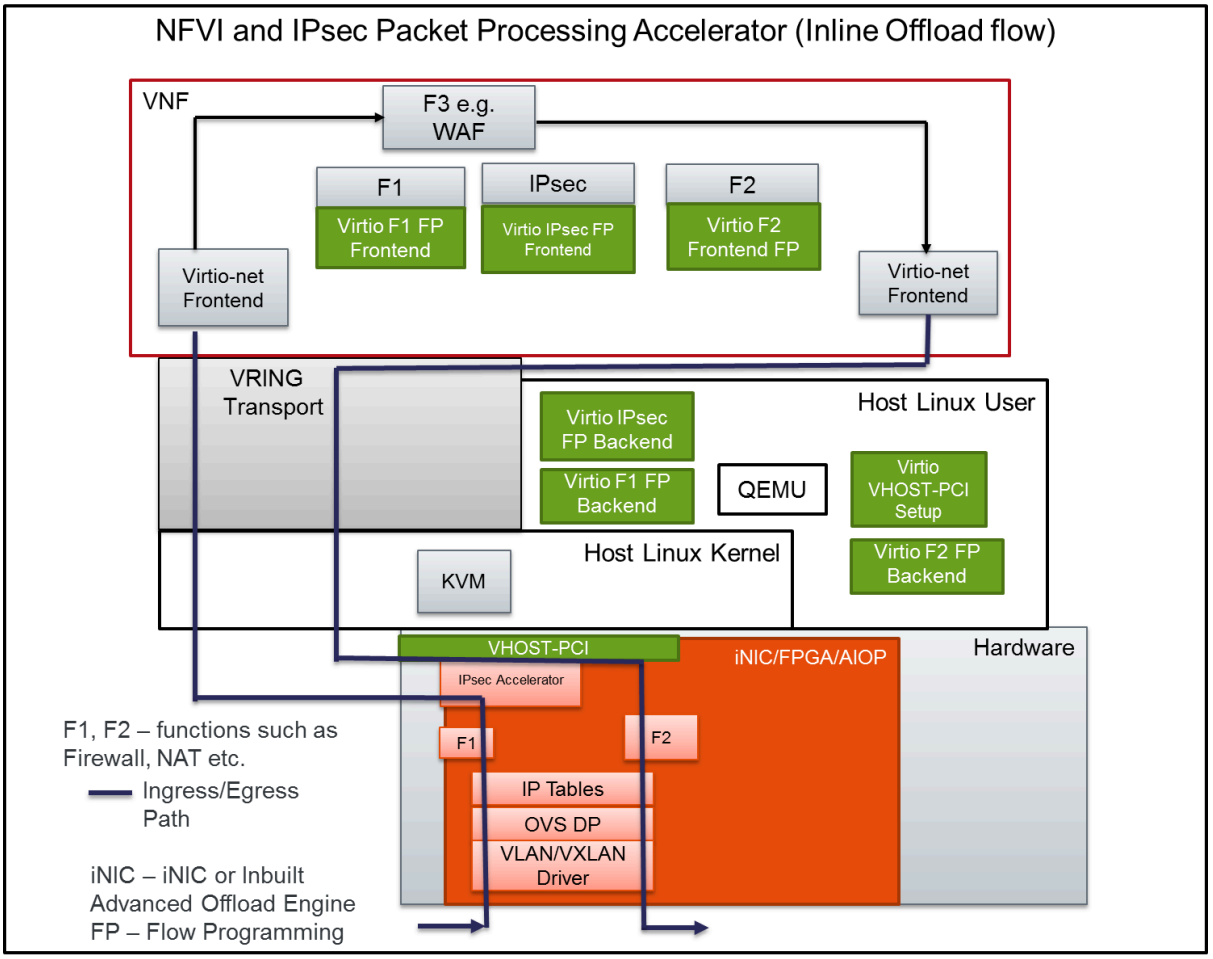
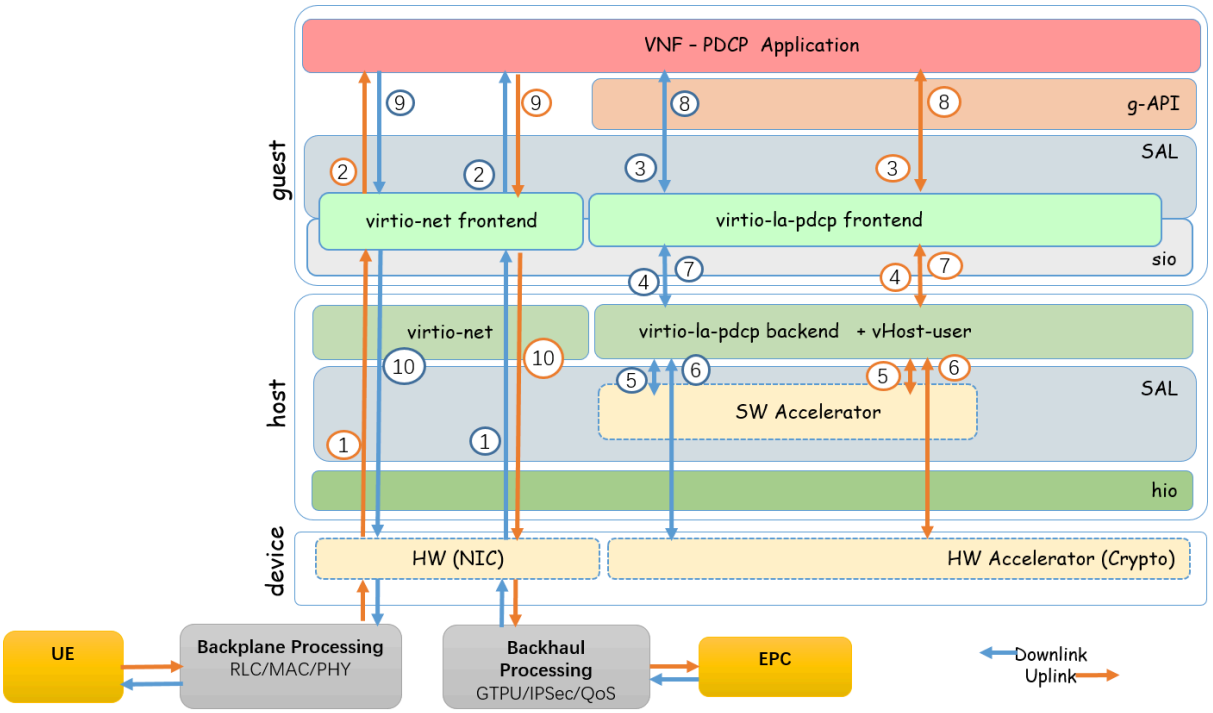


Figure W NFVI and IPsec Packet Processing –Inline Accelerator Flow

In this case, part of the packet processing happens in the iNIC as part of the offload operation, namely F1 processing and IPsec processing (Inbound processing or decryption). The packets are then sent to the VNF through the Virtio-net frontend to the F3 (WAF) for processing. Subsequent to processing by F3, the packets are sent via Virtio-net Frontend for subsequent processing (Outbound IPsec processing) and F2 before being sent out.

5.5 PDCP Look Aside Accelerator



This figure shows Packet flow between EPC – eNodeB - RLC-MAC:

1. Packets processed by Vhost-Net
2. Packet announced to VNF through Virtio-Net driver
3. Packets arrive at the PDCP module for Processing
4. As packets are submitted by the PDCP Module to the Virtio-pdcp front end driver, the buffers are put in the Virtio Descriptor Vrings or Virt Qs to be transferred to the Virtio-pdcp Backend.
5. The Virtio-pdcp Backend is responsible for translating the packets from Virt Q Descriptor to the actual hardware accelerator in a message that the accelerator understands and vice-versa.
6. The Virtio-pdcp Backend is also responsible for picking up processed packets from the hardware accelerator, updating the VirtQ rings and notifying the Guest VNF.
7. The processed packets are sent out through the Virtio-net interface for post security processing.

6 History

1/31/2016	Initial draft from etherpad discussion and earlier contributions.
2/5/2016	Add vRAN as the third targeting application; Add PDCP Offload accelerator as a new use-case.

7 Editors

Lingli Deng

8 Contributors

Srini Addepalli
Mario Cho
Saikrishna Kotha
Argy Krikelis
Kin-Yip Liu
Abdel Rabi
Subhashini Venkataramanan
Keith Wiles
Fahd
Francois
Rajesh
Yannan Yuan

9 References

[1] OPNFV DPACC Architecture:
https://docs.google.com/document/d/1O4rtCh1vbTOO5cMwmRwfv3UJb_bVWZrqXQS_-QJAk10/sharing