OPNFV DPACC Architecture

Contents

1 Introduction

The project is to specify a general framework for VNF Data Plane Acceleration (DPA or DPACC), including a common suite of abstract APIs at various OPNFV interfaces.

The purpose of specifying such a framework is to enable VNF portability and resource management across various underlying integrated SOCs and standard high volume servers (SHV), with/without hardware accelerators.

It may desirable to design such DPA API framework to easily fit underneath existing prevalent APIs (e.g. sockets), as a design choice, for legacy designs.

The framework should not dictate what APIs an application must use, rather recognizing the API abstraction is likely a layered approach and developers can decide which layer to access directly.

2 Framework

2.1 Partitioned View

2.1.1 Acceleration Layer for Guest (VM's)

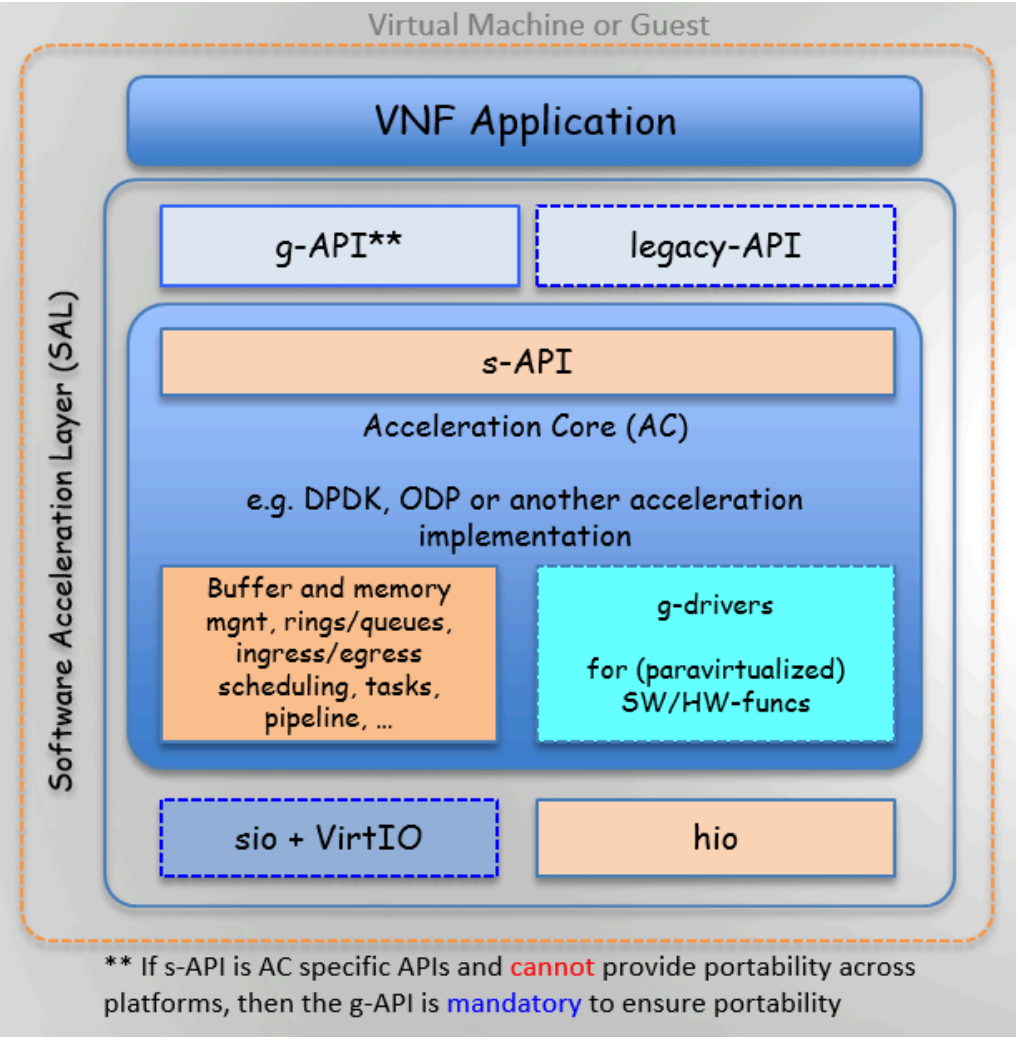

Figure 1 Acceleration Layer for Guest (VM's)

Figure 1 depicts the acceleration layer for VNF applications running in the guest. In particular, SAL is identified as the key component in this layering architecture.

2.1.1.1 SAL: software acceleration Layer

- SAL: short for Software Acceleration Layer, provides a target abstraction for application software, which is composed of the following components and interfaces:

2.1.1.2 North-bound interfaces from SAL to APP

- g-API (required): short for Generic API, meant to be a standard application interface and is required for application portability.
- legacy-API (optional): short for Legacy API, meant to be kept intact to help existing applications to remain functioning if they are not modified to be making explicit usage of g-API, but keep using APIs like sockets, libcrypto, …

2.1.1.3 Functional Components of SAL

- AC (required): short for Acceleration Core, meant to be the software library for acceleration implementation, e.g. DPDK and ODP, which in turn is composed of the following components and interfaces:
  - s-API: short for APIs for utilizing an AC (APIs from the AC);
  - basic functions: buffer and memory management, rings/queues, ingress/egress scheduling, tasks, pipelines, and so on;
  - g-drivers: short for General driver for each device type, which is implemented in software or the frontend to the hardware (may be different for different acceleration functions)

  AC is a required component for SAL implementation.

2.1.1.4 South-bound interfaces between guest and host

- 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.

2.1.2 Acceleration Layer for Host (Hypervisor)

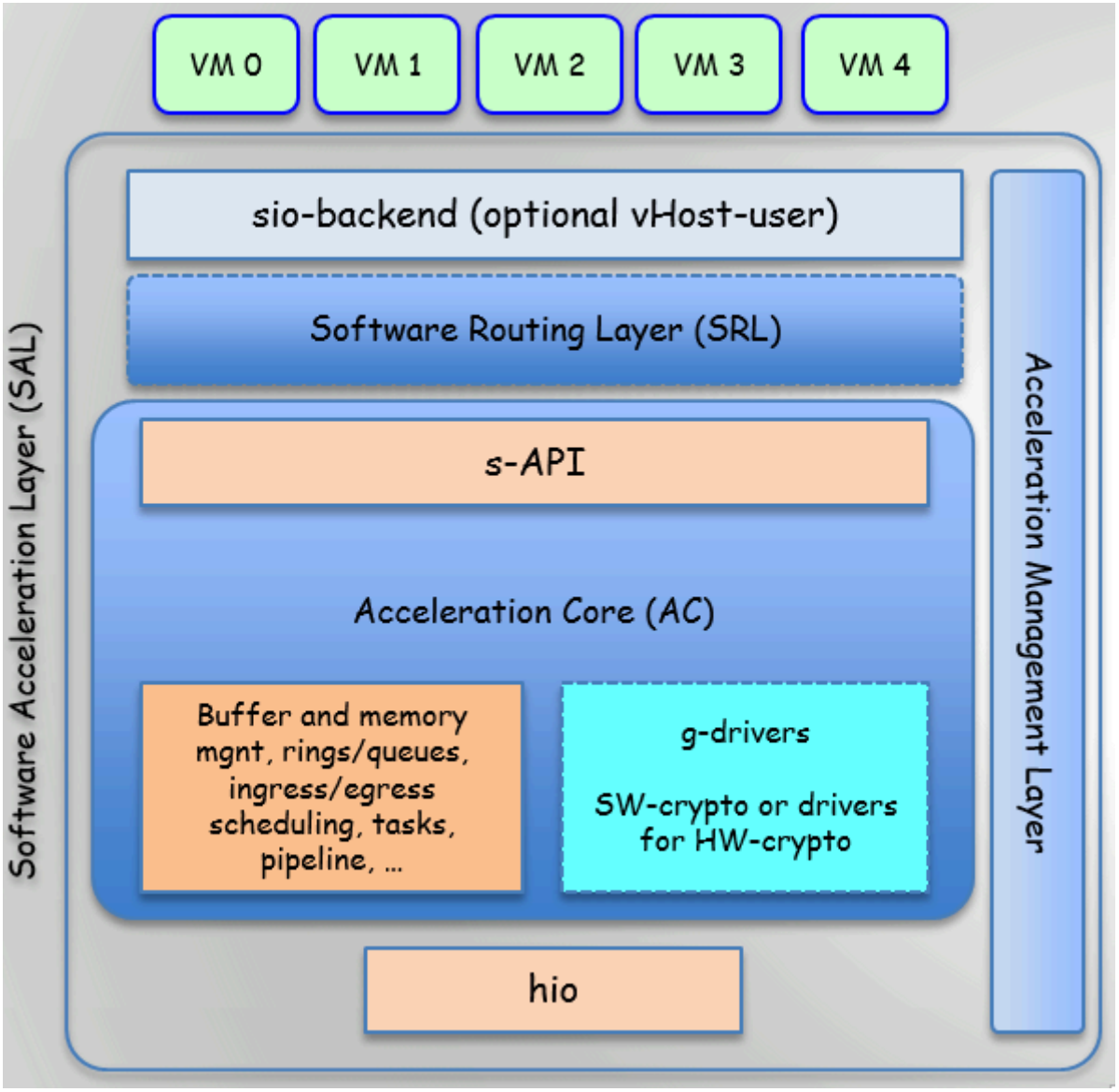

Figure 2 Acceleration Layer for Host (Hypervisor)

Figure 2 depicts the acceleration layer for applications running in the host. Once again, SAL is identified as the key component in this layering architecture, which is very similar to the guest SAL.

2.1.2.1 SAL: software acceleration Layer

- SAL: short for Software Acceleration Layer, provides an abstraction between SW and HW, which is composed of the following components and interfaces:

2.1.2.2 North-bound interfaces for SAL@host to guest

- sio-backend (optional): backend of paravirtualized drivers, e.g. vHost-user (User space based VirtIO interface), which is optional interface between VMs and the host.

Note:

The sio-backend could be a set of paravirtualized drivers plus it can contain the implementation of the VirtIO backend called vHost-user.

2.1.2.3 Functional Components of SAL

- SRL: short for Software Routing Layer, is referring to the entity to perform forwarding or switching of packets to external or internal system, which is needed in some cases to give support of VM to VM communication without having to leave the system, as in the case of an external router or switch. Possible SRL applications could be Open vSwitch or a vRouter design. SRL is an optional layer for the host.
- AC: short for Acceleration Core, meant to be the software library for acceleration implementation, e.g. DPDK and ODP, which in turn is composed of the following components and interfaces:
  - s-API: short for APIs for utilizing an AC (APIs from the AC);
  - basic functions: buffer and memory management, rings/queues, ingress/egress scheduling, tasks, pipelines, and so on;
  - g-drivers: short for General driver for each device type, which is implemented in software or the frontend to the hardware (may be different for different acceleration functions)

  AC is a required component for SAL implementation.
- AML: short for Acceleration Management Layer, to be defined for orchestration and management later in Sec 2.1.2.5 and spans more than the SAL. It is expected that AML is to interface to the orchestration layer to allow the orchestration and management layer to help manage the VNF applications as well as local acceleration SW/HW resources. The AML is present in the host SAL only.

2.1.2.4 South-bound interfaces between host and HW

- hio: short for Hardware I/O interface, which is referring to the Non-virtualized interface, accessed only by host SAL in the case of a SAL in the host. The guest may also contain a hio if the guest requires direct access to the hardware.

2.1.2.5 Host AML

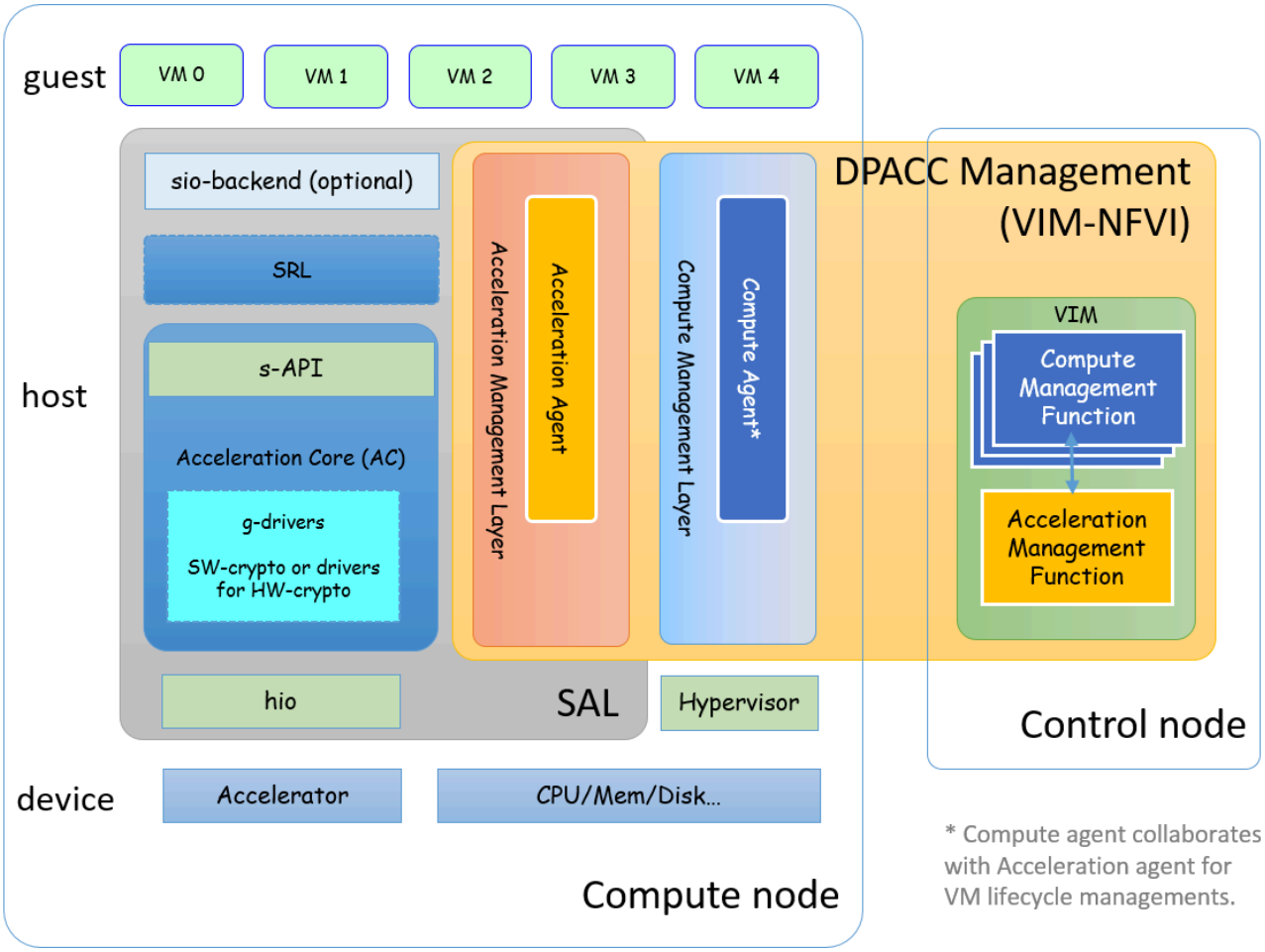

Figure 3 Host AML: VIM-NFVI acceleration management architecture[2]

As depicted in Figure 3, there are two dedicated components for DPACC acceleration management, including the acceleration management function as part of VIM and the acceleration agent as part of host AML on NFVi platform.

The acceleration management function, which corresponds to the global management as specified in [2], is the centralised orchestrator of acceleration resources. It receives requests from the compute/network/storage management functions of VIM, arranges those requests and communicates with the acceleration agents to fulfil the requests.

The acceleration agent, which corresponds to the core of local management as specified in [2], manages communications with locally/remotely attached accelerators. It collaborates with the Acceleration Core (i.e. the g-driver defined above) and/or the hypervisor to collect information of and/or conduct operations on the accelerators. It also collaborates with other local agents of VIM management functions (e.g. compute agent) in response of acceleration orchestration as part of NS/VNF lifecycle management events.

2.2 Usage Analysis

2.2.1 Usage: VNF Acceleration types

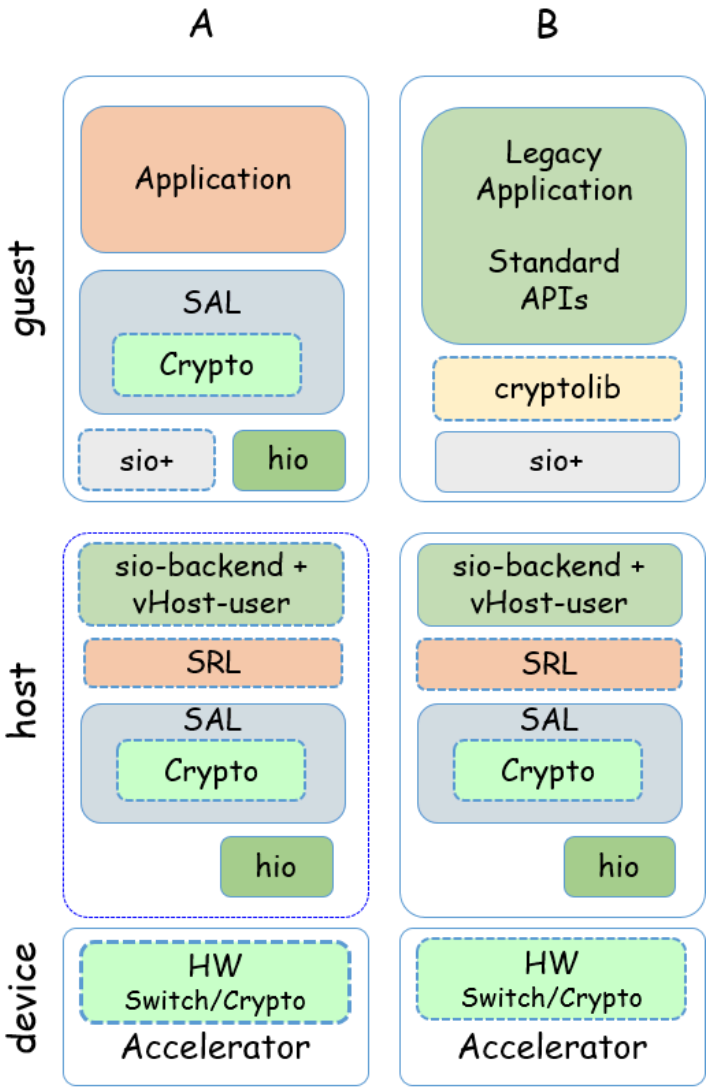

Figure 4 VNF Acceleration types

Figure 4 depicts two typical usage options for crypto accelerated VNFs employing DPACC architecture. Both options are attempting to show a number of options or layers a VNF could deploy in many different configurations.

Example A on the left is meant for legacy application using crypto library via VirtIO to accelerate crypto operations in the host. Crypto support is contained on the guest SAL, potentially using external hardware accelerator. In this example, explicit acceleration layer for VM to VM communication is missing, but can be supported by Host-SAL or Guest-SAL via external vSwitch accelerator. vHost-user shown here (normally in the SAL) for sio+ access and host/sio+ layers are optional in some configurations.

Example B on the right is meant for legacy applications making use of legacy APIs to cryptolib or even being agnostic to the traffic encryption handled in the host/accelerator. The guest has only the standard sio support, but is accelerated by the acceleration layer in the

host. In this example, an explicit layer for SRL (vSwitch/vRouter) is added in the host for VM to VM communication. The SRL in the host layer must support the SIO in the guest to be able to switch packets from VM to VM. In this case, SRL contains the vHost-user interface instead of SAL. HW-Crypto or HW vSwitch at the device layer are being utilized for acceleration. Host SAL is optional in some configurations.

Note that both Example A and Example B above are attempting to show a number of options or layers a VNF could deploy in many different configurations.

- If the guest does not require host support the host layer are optional. The host layer allows the guest to have better acceleration support without having to have a SAL layer in the guest.
- For cases where host SAL is being used, an explicit layer for SRL (vSwitch/vRouter) can be added in the host for VM to VM communication. The SRL in the host layer must support the SIO in the guest to be able to switch packets from VM to VM. In this case, SRL contains the vHost-user interface instead of SAL.
- In some cases, HW-Crypto or HW vSwitch at the device layer are being utilized for acceleration via HIO in the host. While in other cases, the acceleration functionalities are either missing or can be realized by SW-Crypto accelerators within guest or host SAL.
- The guest can have crypto support in software (either guest SAL as in Example A or host SAL as in Example B) or be using a external crypto hardware via the hio interface.
- The sio is optional in the cases where the guest does not require direct access to the host for any reason (e.g. using HW-Crypto or HW vSwitch at the device layer via guest hio and bypass the host userspace entirely), but as the sio layer is normally present it can be used for management access to the guest.

2.2.2 Usage: VNF Acceleration Examples

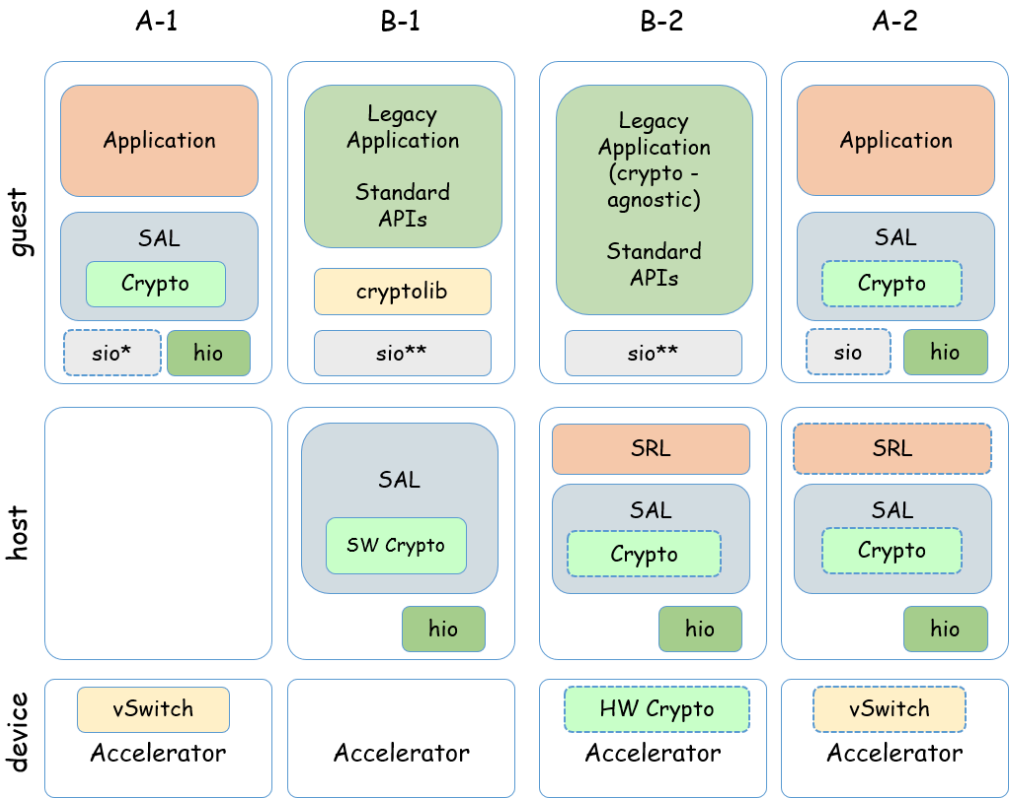

Figure 5 VNF Acceleration (e.g. crypto)

Figure 5 further depicts four typical usage options for crypto accelerated VNFs employing DPACC architecture.

Example A-1 on the left includes a VM with a SAL for direct access to external device and VM to VM support only supplied by kernel based vSwitch or external device. The sio is optional for host access for management, and non-hio packets may have poor performance in some cases.

Example B-1 in the middle includes a legacy application using crypto lib via VirtIO to accelerate crypto operations in the host, where VM to VM is still missing, but can be supported by SAL to external switch accelerator. The design could also be enhanced by adding a kernel based vSwitch for VM to VM traffic.

Example B-2 in the middle includes a legacy application being agnostic to the encrypted traffic being handled in the host software or in external accelerator, which means it sends and receives packet in clear text format as the host layer or the device is encrypting or decrypting the packets unbeknownst to the guest application. An SRL (vSwitch/vRouter) is added for VM to VM communication.

Example A-2 on the right includes accelerated application using SAL in guest to access crypto accelerator directly, as well as flexible vSwitch or vRouter support in SW or HW. SAL allows for some/all crypto operations to be done in the guest or passed to the host for processing.

2.3 Aggregated View

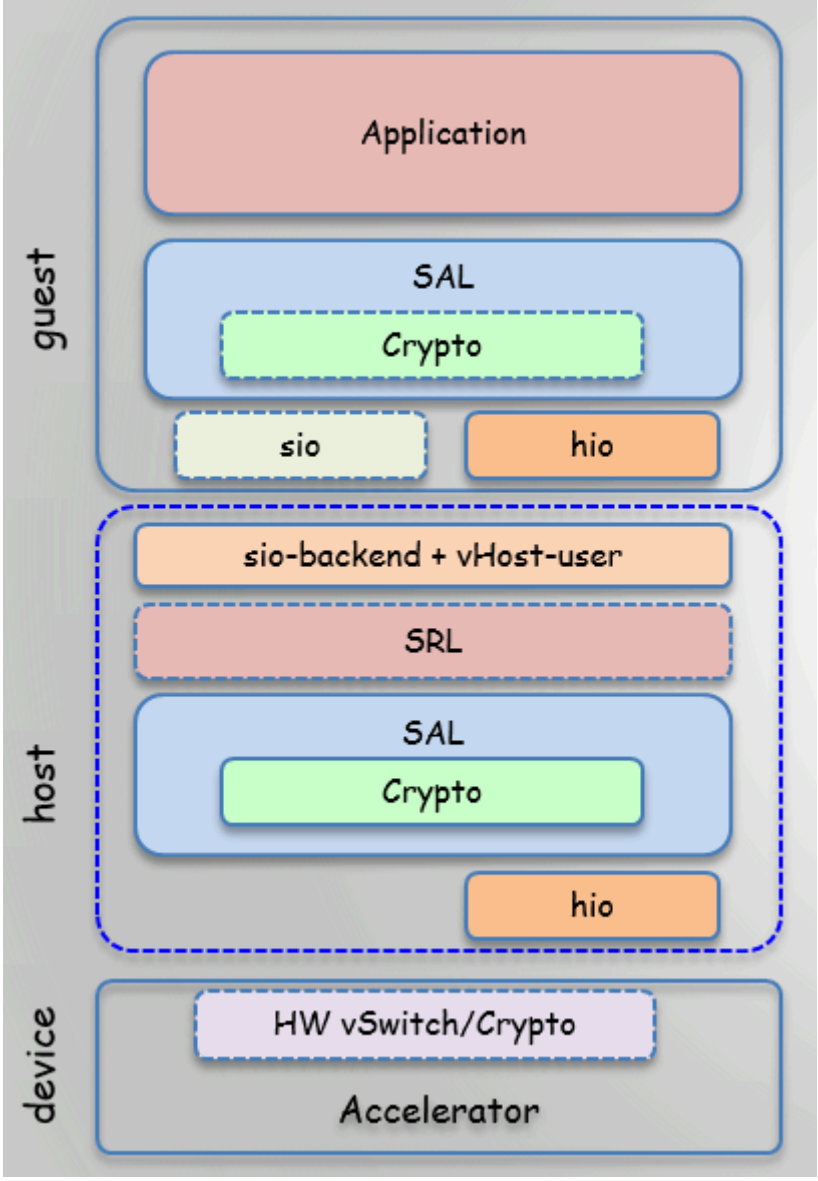

Figure 6 DPACC Architecture: NFV High Level View

Figure 6 focus on one as a basic high level view. This view is attempting to how the DPACC layer and configuration can be designed by the developer to give his application the best accelerated performance.

- SAL is the software to hardware abstraction layer, which makes possible additional services which can be controlled by the orchestration layer. SAL can be deployed to either guest or host. SAL is required in all DPACC usages identified in this document.
- sio-backend + vHost-user is normally in the SAL or SRL layer, but shown here to illustrate vHost in the host.
- A SAL in the guest allows for the best performance selection as it enables
  - direct access to hardware acceleration via SR-IOV, SOC-specific interface or other pass-through mechanism, and/or
  - software acceleration in the guest.

- SRL or HW vSwitch adds VM to VM routing or switching of packets.
- A SAL in the host gives scalability for non-accelerated VMs and/or native applications, which would provide the best performance and flexibility for a VNF application.

3 Related Upstream Work Plans

OPNFV targets at integrate various open-source components from upstream projects to deliver an open-source NFVi platform. To fulfill the functional requirements as specified in [1] for various interfaces as defined in the above described DPACC architecture, the following enhancements to existing upstream projects are expected:

First of all, sio interface, (e.g. VirtIO as specified by OASIS) [3], needs to be enhanced to provide better control and adding more device types, including

- adding various device features (e.g. Crypto) support to VirtIO as an acceleration feature;
- to support legacy VirtIO API for backward compatibility;
- to support exporting VNF metadata needs for acceleration and orchestration; and
- enhancing performance as a requirement for the solution

Secondly, Software Acceleration Layer [4] needs to be enhanced to

- help locate/find hardware/software acceleration mechanisms for the VNFs;
- support a number of software and/or hardware accelerators; and
- help enhance support for orchestration layer along with the VIM for plumbing the data flows.

Thirdly, VIM implementations (e.g. Openstack) [2] needs to be enhanced for acceleration orchestration and management (e.g., OpenStack Nomad [5]), including:

- Acceleration resource lifecycle management; and
- Acceleration requirement description and orchestration.

4 References

[1] DPACC High Level Requirements:
https://docs.google.com/document/d/1YexfnLRZ9gZnj-5PNOnrJ5CVMhbzZ7mIJogW6nyIGq0/edit

[2] Gap Analysis of OpenStack for DPACC:
https://docs.google.com/document/d/1_fOinIQNcPwNODZPzGK5vRMPJQLwL7iLds4NFnjXSms/

[3] Gap Analysis of VirtIO for DPACC
TBA

[4] Gap Analysis of DPDK/ODP for DPACC

[5] OpenStack Nomad Project: https://wiki.openstack.org/wiki/Nomad

5 History

2015/12/21	Initial draft from wiki page version dated May 2015.
2016/1/22	Added Section 2.1.2.5 for host AML.
2016/2/10	Added Nomad as a reference in Section 3.
2016/3/12	Updated Section 2.2 to clarify the usage types and usage examples.

6 Editors

Keith Wiles
Bob Monkman
Lingli Deng
Zhipeng Huang

7 Contributors

Peng Yuan
Ola Liljedahl
Michele Paolino