WHITE PAPER

Evaluating Dynamic Service Function Chaining for the Gi-LAN

The Gi-LAN is best served with dynamic service function chaining for flexibility, functionality, and low total cost of ownership. Is the technology ready today? Find out what to expect next.

Dynamic service function chaining (SFC) is ready for CSP evaluation and for small-scale proof of concepts, but it is not yet complete for large-scale deployment in early 2016. Intel is working with the open source community to further evolve dynamic SFC for the Gi-LAN use case.

AUTHORS

Software Defined Networking Division, Network Platforms Group, Data Center Group

Joseph Gasparakis
Open Networking Platform Software Architect

Kevin Smith
Solution Architect

Danny Zhou
SDN/NFV Software Architect

Dana Nehama
Product Marketing Manager

Gilbert F. Hyatt II
Product Owner—Intel ONP RA Integration

Jeff Oliver
Network Software Engineer

Trevor Cooper
Solution Architect

Rory Browne
Network Engineer

Executive Overview

Dynamic service function chaining (SFC) enables communications service providers (CSPs) to use software to automatically and dynamically set up, configure, and manage network services. Using dynamic SFC, CSPs can optimize services for the ever-changing mix of traffic introduced into the CSP network, based on their policies. Dynamic SFC allows optimized utilization of the CSP infrastructure because traffic is classified and directed to desired network services only. It further allows meeting more stringent service-level agreement requirements, helping CSPs to comply with regulatory demands and at the same time significantly reduce capital expenditures and operational expenditures. Recognizing the potential for dynamic SFC to increase business agility for CSPs, the Intel® Open Network Platform (Intel® ONP) program has invested in enabling the technology. As of early 2016, the Intel ONP is providing a snapshot evaluation of dynamic SFC as a potential technology in the Gi-LAN context.

Our findings suggest that dynamic SFC is ready for CSP evaluation and for small-scale proofs of concept, but it is not yet complete for large-scale deployment in early 2016. Our implementation of dynamic SFC as a potential option for the Gi-LAN use case, using the Intel ONP in support of the Internet Engineering Task Force Network Service Header draft, highlighted several technology gaps. Intel is working with the open source community to further evolve dynamic SFC, which shows great promise, and has proposed multiple software patches to address many of these gaps.

This paper provides the technical details about how Intel evaluated and tested dynamic SFC using the Intel ONP reference architecture. To support the evaluation, Intel developed a set of scripts for configuring the system, including OpenStack*, OpenDaylight* SDN Controller, and Open vSwitch* with the Intel® Data Plane Development Kit (Intel® DPDK). Scripts are provided to guide the reader, illustrating how a system can be configured. These scripts can be found at: https://download.01.org/packet-processing/ONPS2.0/Intel_ONP_Server_Release_2.0_Use_Case-GiLAN-1.0.tgz

Intel continues to study dynamic SFC in the context of the Gi-LAN as part of the Intel ONP program, where the Intel ONP reference architecture is used as the base platform for testing and benchmarking. We anticipate publishing additional reports in the future as our analysis continues and the industry evolves. We also welcome feedback and proposals as to other aspects of dynamic SFC that may require industry attention.

Table of Contents

Introduction

Mobile traffic is expected to grow at a compound annual growth rate of at least 45 percent.[1] Communications service providers (CSPs) and the industry are evaluating ways to efficiently scale to meet this growth, as well as meet consumers' demand for more services and CSPs' need for more cost-effective solutions. One such adaptation is to adopt network function virtualization (NFV).

Virtualizing services onto standard, off-the-shelf hardware and taking advantage of software-defined networking (SDN) can increase network flexibility and slash costs, as well as enable CSPs to quickly launch new services without major investment or risk. To illustrate how NFV is poised to transform the marketplace, analysts predict the current USD 2.3 billion NFV market to reach USD 11.6 billion in 2019.[2]

The Intel® Open Network Platform (Intel® ONP) provides reference architecture for SDN and NFV deployments. The Intel ONP supports the European Telecommunications Standards Institute (ETSI) NFV architecture and provides the robust NFV infrastructure (NFVI) necessary to virtualize functions, control, and orchestration. Ecosystem suppliers in CSP networks, enterprise environments, and cloud data centers can use the Intel ONP reference architecture to more easily build solutions using an open source software stack running on commercial, standard high-volume servers. With the Intel ONP reference architecture, Solution vendors can plan, evaluate, and benchmark designs in advance of NFV deployments. One important use case for NFV that the Intel ONP program is evaluating is the virtualized Gi-LAN.[3]

Today, Gi-LAN implementations are based on non-flexible, hardware-based architecture and error-prone manual configuration. CSPs are using a broad mix of vendors, making the Gi-LAN operationally challenging. Dynamic service function chaining (SFC), which relies on implementations of the Internet Engineering Task Force (IETF) Network Service Header (NSH) draft, has the potential to improve CSPs' infrastructure utilization and agility, especially in the Gi-LAN case, in order to accommodate mobile traffic growth. Dynamic SFC will enable CSPs to innovate, differentiate, and monetize services using unique capabilities provided by IP functions between the packet gateway and the Internet.

The Intel ONP program has conducted an evaluation of dynamic SFC for the Gi-LAN use case. In this evaluation, we configured release 2.0 of the Intel ONP with dynamic SFC and basic Gi-LAN capabilities. We then investigated the readiness of open source software to support this use case, identifying technology gaps. We found that dynamic SFC, when used in the Gi-LAN use case, is an industry area that is not yet ready for large-scale deployment. However, as the virtualized Gi-LAN use case and capabilities evolve, dynamic SFC is poised to become a powerful mechanism for CSPs to realize the benefits of NFV and SDN technologies. Intel is working with the open source community to close existing technology gaps through open source initiatives, which will move the SFC virtualized Gi-LAN[4] use case toward a fully functional state.

[1] Ericsson Mobility Report, June 2015. www.ericsson.com/res/docs/2015/ericsson-mobility-report-june-2015.pdf

[2] FierceWireLessTech, "Study: NFV market will hit $11.6B in 2019." www.fiercewireless.com/tech/story/study-nfv-market-will-hit-116b-2019/2015-07-20

[3] The "Gi" (Gateway-Internet) LAN interface (referred to as the sGi-LAN in 4G networks) is the reference point defined by the 3rd Generation Partnership Project (3GPP) as the interface between a communications service provider's mobile packet gateway and an external packet data network (such as the Internet). www.3gpp.org

[4] Subsequent occurrences of "Gi-LAN" in this paper refer to a virtualized Gi-LAN.

A Closer Look at Gi-LAN and Traditional Service Function Chaining

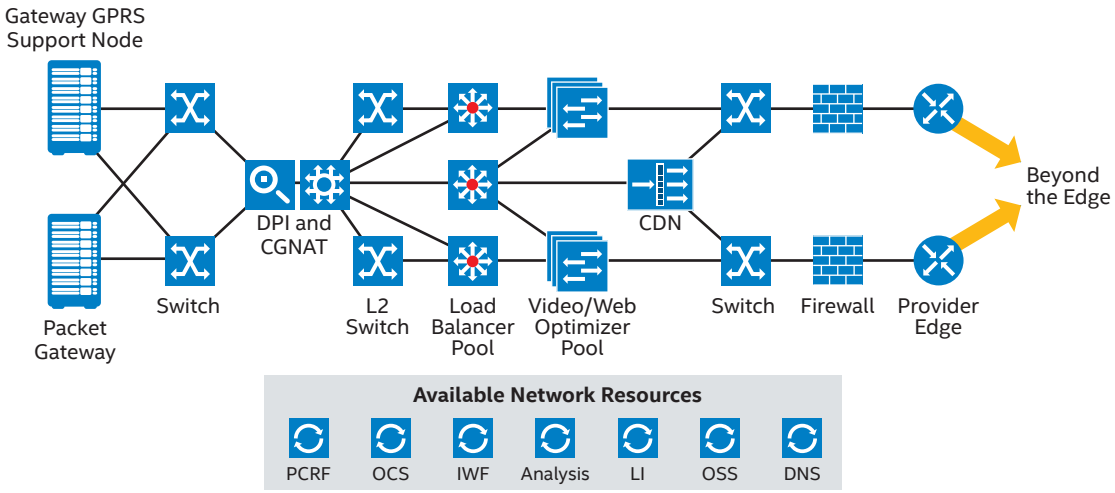
The main functionality of the Gi-LAN is to map subsets of network flows to specifically selected network services, in a given order to support a desired CSP policy and an appropriate service-level agreement. For a typical CSP, several million subscribers access services through the Gi-LAN. CSPs desire not to feed all network services with this high-volume traffic; therefore, they are interested in classifying the traffic and directing it only to necessary network services. To meet the increasing demands for service scaling while allowing for infrastructure cost optimization and network agility, the Gi-LAN network functions require increased flexibility, optimal routing, and efficient use of node capacity.

Static Service Chains in the Gi-LAN Context

Preconfigured paths, designated to the aforementioned network traffic subsets, direct traffic across disparate IP functions that lead from the mobile packet gateway through the Gi-LAN to the external packet data network (see Figure 1 below). These preconfigured paths are commonly referred to herewith as "Static service chains." A service chain requires the definition, configuration (which involves human intervention and can be error-prone) and instantiation of an ordered set of network functions, and the subsequent steering of traffic flows through those network functions. CSPs typically use access point names (APNs) as an initial means to define separate service chains for different subscriber pools. Within APNs, depending on the functionality of the gateway, additional preconfigured packet inspection logic can be used to further delineate unique service chains per subscriber session.

Some examples of Gi-LAN service functions include Firewall, Transmission Control Protocol (TCP) Proxy, Network Address Translation (NAT), Load Balancing, Content Delivery Optimization (such as Web, video, audio, and Internet television), Deep Packet Inspection (DPI), and Header Enrichment. The underlying capabilities of these Gi-LAN service functions enable CSPs to offer subscriber-facing, value-added services such as video optimization, content caching, parental control, URL filtering, and security services. Figure 1 shows a simplified Gi-LAN architecture. In the figure, note that traffic flows may take different paths traversing the Gi-LAN network, per CSP policy.

The lack of a standards developing organization with oversight for Gi-LANs has resulted in a wide range of Gi-LAN architectural approaches even within the same mobile network operator group. It has led to a broad vendor solution mix, making the management, maintenance, and evolution of the Gi-LAN operationally challenging. In traditional CSP mobile networks static SFC is deployed and the Gi-LAN consists of service functions based on rigidly defined physical appliances (a combination of proprietary software and hardware) that forward traffic to each node to inspect, steer, modify, monitor, or report on the mobile data packets.



CDN – content delivery network; **CGNAT** – Steering/Carrier Grade Network Address Translation; **DPI** – deep packet inspection; **DNS** – domain name system; **GPRS** - General Packet Radio Service; **IWF** – interworking function; **LI** – lawful interception; **OCS** – online charging system; **OSS** – operational support system; **PCRF** – policy and charging rules function

Figure 1. The architecture of a generic, simplified Gi-LAN.

These physical appliances use physical cabling and preconfigured static routing mechanisms and have to be designed for maximum capacity and throughput even when traffic volume is low.

While many CSPs implement service steering on the Gi-LAN's network underlays to steer traffic to the appropriate set of service nodes, today's static services deployment approach can't support the cost, flexibility, and scalability expected with emerging demands.

Drawbacks of Static Service Function Chaining

The inflexible nature of today's mostly static service function chaining in production Gi-LAN deployments limits CSPs' ability to scale, innovate, and monetize new services. Some of the most common challenges related to today's static SFC approaches[5] are as follows:

- Topological dependencies
- Configuration complexity
- Constrained high availability
- Inconsistent ordering of service functions
- Difficult application of service policy
- Transport dependence
- Inelastic service delivery
- Coarse and inflexible traffic selection criteria
- Limited end-to-end service visibility
- Inefficient classification and reclassification per service function
- Complexity associated with symmetric traffic flows
- Limited interoperability, lack of standard ways for inter-device exchange of metadata, and lack of interaction between devices on the chain resulting from multi-vendor service functions

These challenges can be divided into two key areas: lack of flexibility and increased costs.

Lack of Flexibility

IP networks rely on basic routing and forwarding functions and the combination of advanced functions for the delivery of added-value services. Historically, network functions are implemented by manipulating physical network configuration that are labor intensive, error-prone, and unable to react in real time to traffic changes. These rigid, inflexible network services also lack network infrastructure optimization because traffic has to flow to the physical appliance's location in the topology instead of the service being available close to the application that needs to consume it.

When static Gi-LAN service chains are defined by APNs, the CSP must configure a separate APN or alias APN[6] for each unique service chain. Each service function classifier node and APN configuration is unique to the vendor(s) in the network, which results in custom implementations for each CSP. Any network topology changes must be manually coordinated with service chain provisioning (a significant operational consideration), and all monitoring is also manual.

These attributes of static SFC make it difficult for CSPs to create differentiated service offerings and dynamically adapt services in response to traffic patterns or subscriber requests.

Increased Costs

Today's static approach to service chains also drives up costs. For any subscriber attached to a particular APN, that subscriber's mobile packets are steered across all the nodes in that service chain, regardless of whether that subscriber needs all the services in the chain. In other words, the physical infrastructure does not accurately reflect subscriber needs. Another source of cost burden with static SFC is scalability and reliability. While the mobile gateways may be able to handle significant traffic load increases, typical Gi-LAN functions do not necessarily scale in the same manner. Also, Gi-LANs must be significantly overbuilt in the appliance architecture to cope with site or unduplicated element failure.

[5] See the Internet Engineering Task Force (IETF) document "RFC7498 Problem Statement for Service Function Chaining."

[6] An alias access point name (APN) is a common way that communications service providers overwrite APNs in the network. Alias APNs provide unique policy and routing controls without changing the physical APN on the handset and other service nodes.

Dynamic Service Function Chaining

Dynamic SFC can address many of the drawbacks of static SFC mentioned earlier, such as topological dependencies and inelastic service delivery, and can therefore transform CSPs' ability to innovate and operate cost efficiently. This section describes how dynamic SFC works and discusses benefits and key challenges associated with this new technology.

Overview of Dynamic Service Function Chaining

Dynamic SFC enables CSPs to use software – the SDN Controller – to programmatically and dynamically configure network services. The result is that SFC no longer needs to change the network at the hardware level or be dependent on manual command-line interface configurations, thereby significantly reducing capital and operational expenditures, and speeding service delivery. Dynamic SFC allows the optimal use of network resources and supports easy provisioning of new services based on content type. The agility of dynamic SFC allows cloud-like elastic services to be used by allocating infrastructure resources necessary for the volume of traffic happening at any given point in time, which reduces cost and frees resources for other tasks.

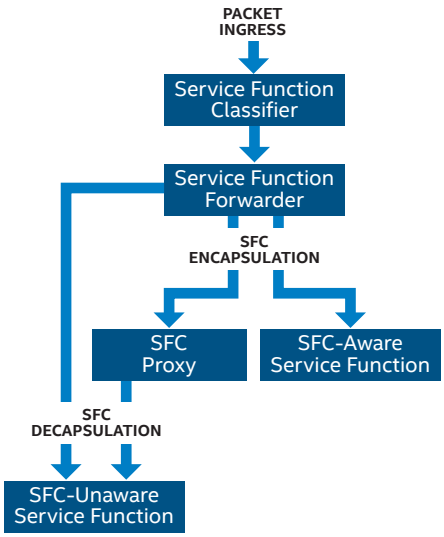


Figure 2. An overview of dynamic service function chaining (SFC).

Four fundamental components combine to enable dynamic SFC: a Service Function Classifier, a Service Function Forwarder, Service Functions, and an optional Service Function Proxy. Figure 2 shows a simple visualization of how SFC works based on the IETF SFC draft architecture, while Figure 3 shows how dynamic SFC works with the VNF orchestrator and SDN Controller. Software programmability through the SDN Controller provides the dynamic aspect, where a service chain can be initiated per policy, with changing external triggers or traffic. Therefore, using SFC can provide the necessary elasticity on the network by accessing resources when traffic needs to increase and by releasing them while these conditions disappear. Also, scalability, load balancing, and failure recovery are possible with a dynamic virtual network dedicated for SFC—meaning that service chaining is no longer dependent on manual command-line interface configurations.

To provide for transport-independent service (that is, a service function may reside anywhere and be tied to the network by various protocols), the packet is encapsulated and sent over a virtual network dedicated to that set of service functions. Encapsulation may use any industry-standard method, such as Virtual Extensible LAN-Generic Protocol Extension (VxLAN-GPE). After encapsulation, the packet is passed to the appropriate service function. Network Service Headers (discussed in detail in "State of the Industry") add the ability to efficiently signal among service chain elements by adding metadata fields, enabling new functionality and potentially preventing the need of reclassification by downstream service functions.

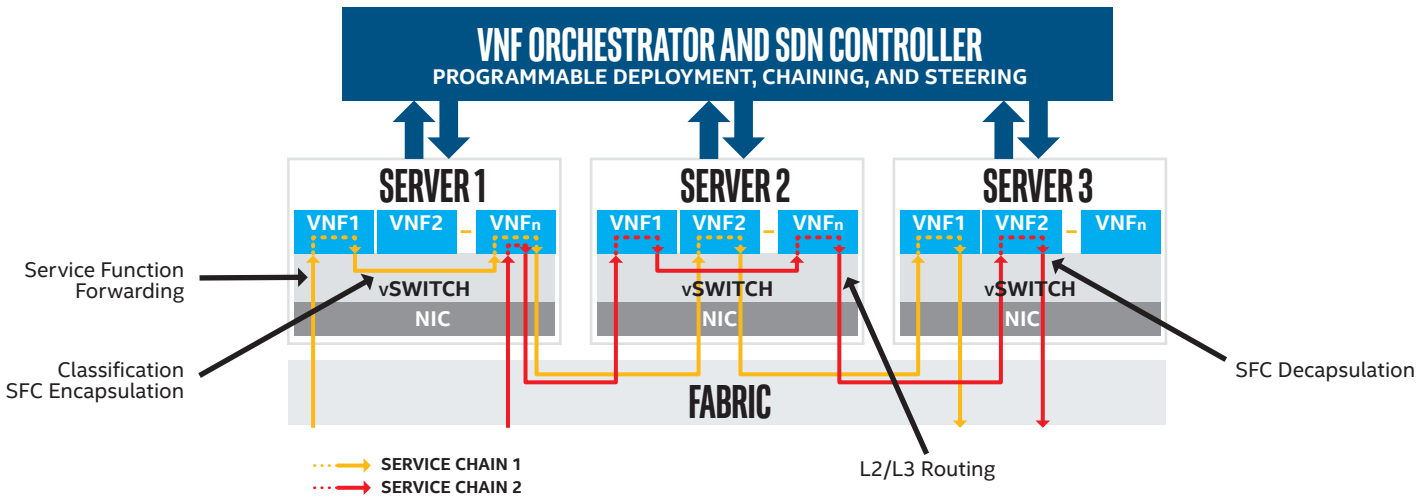


Figure 3. An example of dynamic service function chaining.

Service Function Classifier

The Classifier analyzes traffic in real time and matches it against a pre-defined policy or set of policies, assigning the traffic to a service chain (which comprises the desired ordered service functions). Each service chain is further identified by a service chain header that contains a service chain path ID. The format of this header (see the IETF Network Service Header draft) is undergoing standardization. The OpenDaylight* code used in our evaluation complies with this draft; see State of the Industry for further discussion. While the Classifier can be provisioned in several different ways, and can reside anywhere, typically it is convenient to locate it in a virtual switch (vSwitch), such as Open vSwitch*, subject to SDN Controller programming.

Service Function Forwarder

The Forwarder is responsible for forwarding data packets to their designated service function instances along the service chain path. Each service function, upon fulfilling its designated service, sends the packet back to the Service Function Forwarder for the next hop on the service chain or to its final destination (when the service chain is complete). The Forwarder uses the path ID contained in the service chain header and potentially additional data such as the metadata that identifies the next service function. The Forwarder further encapsulates the packet with the next-hop information (how to reach the next service function) provided by the SDN Controller.

Service Function Proxy

The Service Function Proxy removes and inserts the service chain headers when it forwards the packets to and from SFC-unaware service function, respectively. It is mapping the service function chain headers into one (but not limited to) VLAN, IP in IP, GRE or VxLAN. A Service Function Forwarder usually acts as an SFF-proxy when needed.

Service Functions

These are the individual network functions attached to service chains, such as Firewall, TCP Proxy, NAT, Load Balancing, Content Delivery Optimization, DPI, and Header Enrichment.

Dynamic Service Function Chaining Key Challenges

Dynamic SFC is designed to address the challenges described in RFC7665. While it is a promising technology, dynamic SFC requires further support for virtualization and virtual networks, partially deployed by CSPs today. Use of SDN technology, as well availability of a rich set of virtual appliances that replace dedicated physical appliances and a more complete and mature dynamic SFC technology, is required. The dynamic nature of SFC may introduce some challenges for packet flow, L2/L3 processing, and L4–L7 manipulation and processing. The key industry challenges associated with dynamic SFC exist in both the orchestration and control plane, and in the data plane.

SFC Orchestration and Control Plane

Challenges at this level include the following:

- **Overall NFV and SDN integration challenges being addressed by broader NFV and SDN efforts impact dynamic SFC.** These challenges include loose-coupled integration of the Orchestrator, VNF Manager, Virtual Infrastructure Manager, and SDN Controller to enable VNF provisioning and load-balancing in a dynamic fashion, and high availability and reliability support in centralized orchestration and control plane.
- **Support for defining a service chain using abstract service function types in any order.** In an NFV environment, when the orchestrator instantiates an abstract service function, it needs to place VNFs intelligently by leveraging placement hints (such as the network topology of the overlay and underlay networks and the current utilization of service functions), and use proper service function and service path selection algorithms to ensure end-to-end quality of service. The current OpenDaylight implementation supports a framework to use abstract types and adopt selection algorithms, but needs more features required by a production environment.
- **Operators require inserting or removing a service function on an instantiated service chain without losing state information stored in existing service functions.** The current OpenDaylight implementation of dynamic SFC supports only destroying the existing service chain completely, followed by creating a new service chain instance that includes service function instances on the destroyed chain. This approach creates numerous problems such as removing and recreating classification rules and traffic steering rules.

SFC Data Plane

At this level, the challenges associated with dynamic SFC are as follows:

- **Preservation of metadata fields.** Dynamic SFC enables the exchange of context information (such as classification results and control of symmetric flows) among Service Function Classifiers and service functions, and between service functions. However, there is currently not a well-defined method for enabling service functions from different vendors to properly share (instead of overwrite or pollute) metadata fields in the SFC data plane encapsulation header.
- **Hybrid environments.** As NFV and dynamic SFC evolve, CSPs will probably gradually transition from physical appliances to virtualized networks. Therefore, CSPs will want to integrate existing paid-for physical network functions (PNFs) into a dynamic service chain. This requires a complex hybrid architecture that mixes VNFs with PNFs. In particular, provisioning and traffic steering challenges will arise when CSPs begin mixing different types of service functions, some of which need proxies, while others don't. The architecture must include dynamic SFC proxy functions that mediate between the Service Function Forwarder and NSH-unaware PNFs—this integration may requires assistance and custom code from the PNF vendor. Modeling and traffic steering for this type of hybrid environment is not currently supported in the OpenDaylight implementation of dynamic SFC.

State of the Industry

Today, SFC solutions deployed by CSPs and in data centers are mostly based on static SFC. Static SFC deployment models are coupled to network topology and physical resources that reduce the operator's ability to introduce new services or remove dynamically existing services. The industry is developing various dynamic SFC solutions. A number of open source efforts are underway with the intent of driving the industry toward convergence on a SDN/NFV methodology integrated with the CSP's network; OpenDaylight SFC is one approach.[7] Such a convergence will enable open, interoperable, and interchangeable solutions for broad market adoption and CSP deployments.

Some of the desired technological improvements include the following: metadata format, SFC encapsulation, service chain provisioning, and integration with existing mobile network policy infrastructure. The metadata in the packets' header provides the ability to exchange context information between classifiers and service functions and between one service function and another.

As mentioned earlier, the format and encapsulation of this metadata and how it is exchanged is a topic addressed in standards and industry efforts. The NSH approach used in our evaluation is gaining interest. The current IETF NSH draft provides a mechanism to carry dedicated service chain path information (such as Service Path and Service Index) in the newly added header, rather than mapping VLAN or Multiprotocol Label Switching (MPLS) tags to path information. NSH can also carry shared metadata between network devices and service functions, and between one service function and another. NSH works with various encapsulation protocols, including Ethernet and VxLAN-GPE; details are included in the NSH IETF draft. There is industry and open source community support for NSH as a valid data plane encapsulation approach to support dynamic SFC; examples include an Open Platform for NFV* (OPNFV) proof of concept, OpenDaylight, and Open vSwitch.

While NSH-based dynamic SFC is a potential approach, the industry does not unanimously support NSH. Some in the industry make arguments against NSH's adoption. For the telecommunications equipment manufacturers that plan to support NSH, multiple implementations have already been publicly announced. For example, F5 Networks and Citrix Systems demonstrated NSH-based VNFs at the February 2015 Mobile World Conference. Other vendors, such as Cisco and Sandvine Corporation, are publicly showing NSH use cases.[8] However, broader market acceptance will take time based on product life cycles and CSPs' deployments. The greatest benefits provided by NSH are network transport independence, standardization (the only protocol discussed by the IETF SFC workgroup), a path to incorporate existing PNFs, and new capability to add metadata. While adoption of metadata may take time, especially across different vendors, it provides a mechanism that can enrich dynamic SFC, make the infrastructure more efficient and agile, and lower the cost. For example, in the Gi-LAN use case, where subscriber-specific actions need to be taken, metadata is extremely valuable.

NSH and SDN address some of the more common LAN requirements for dynamic SFC; however, the Gi-LAN and mobile networks add complexities. For example, mobile networks have comprehensive policy and charging controls and policy enforcement architecture (for details, refer to the 3GPP Technical Specification 23.228). Dynamic SFC introduces additional policy controls that need to be coordinated with the existing policy architecture.

[7] Other open source efforts include OpenStack* and Internet Engineering Task Force (IETF) projects.

[8] For more information see the following sites: F5 PR in MWC 2015; Citrix blog; Cisco on NSH; Citrix and Cisco leading NSH development; and Sandvine

The 3GPP (3rd Generation Partnership Project) is also working on several technical reports and studies to evaluate approaches to better integrate dynamic service chain provisioning with the existing mobile network policy control architecture and traffic steering policies. For example, 3GPP Technical Review 23.718 defines a new interface (called the St interface) between the Policy and Charging Rules function (PCRF) and a new Service Chain Traffic Controller function (SCTCF). This interface, among other capabilities, enables the PCRF to interface to the SFC controller functions to provide traffic description filters for more comprehensive and coordinated implementation of dynamic SFC in the Gi-LAN.

Intel is contributing to several open source communities, including Open vSwitch, OpenDaylight, OpenStack*, and OPNFV to help close the gaps. OPNFV is developing a proof of concept with NSH through its Service Function Chaining project. The Intel ONP program drives open source integration efforts to help validate and integrate the underlying dynamic SFC technologies. Intel is also working with ecosystem partners and suppliers on proofs of concept to validate end-to-end solutions that demonstrate the capabilities and state of the technologies. These end-to-end solutions will provide the industry with visibility into the great value of dynamic SFC and the gaps impeding broader adoption of dynamic SFC using SDN/NFV technologies for CSPs' Gi-LAN services.

Service Function Chaining Evaluation by the Intel® Open Network Platform Program

Today's CSPs need more Gi-LAN agility. Dynamic SFC classification and forwarding, implemented using the concepts of SDN, could be the answer because SDN enables virtual networks that provide transport independence as well as granular, scalable, and dynamic traffic steering capabilities. While technology gaps exist and standards are not completed, the technology is far enough along to warrant conducting a formal evaluation by the Intel ONP program and supporting CSP evaluation, proofs of concept, and small-scale deployments. The Intel ONP program has conducted such an evaluation, and Intel is contributing technology to and working with the SDN/NFV ecosystem to optimize packet processing performance for VNFs.

Gi-LAN Implementation on the Intel Open Network Platform

In order to emulate an SDN/NFV-based Gi-LAN on the Intel ONP, we added dynamic SFC to the system architecture to allow for classifying and routing packets to VNFs. Figure 4 shows a high-level diagram of the architecture with two service chains. Note that this architecture represents a highly oversimplified Gi-LAN implementation using 10-GbE links while underlay connections for the Gi-LAN use case are typically 100 GbE.

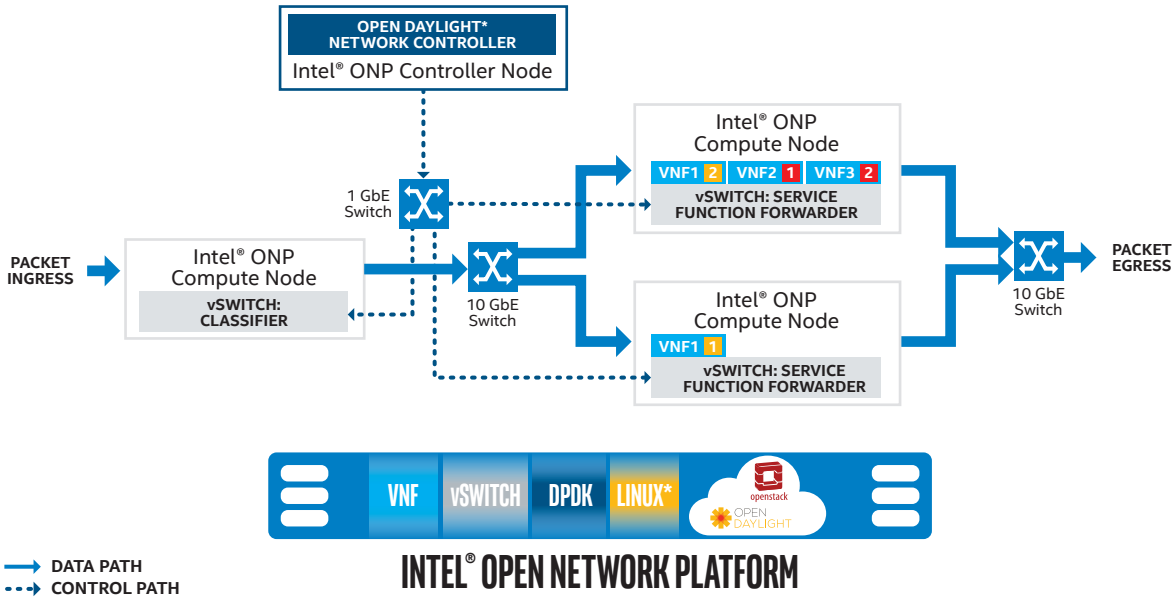


Figure 4. Intel® Open Network Platform (Intel® ONP) configuration for dynamic service function chaining. One service chain includes two virtual machines (VMs) on **different** compute nodes (the orange boxes denote a chain hop number), and the other includes two VMs on the **same** compute node (the red boxes denote a chain hop number).

Proposed Software Contributions for Dynamic Service Function Chaining

Intel has proposed patches[9] to address dynamic SFC mechanism support for the key software ingredients of the Intel ONP (Open vSwitch, OpenDaylight, and OpenStack). The patches and evaluation are a work in progress. The following list describes the various challenges to be addressed in order to enable a dynamic SFC vertical solution:

- **SDN Controller.** Currently, OpenDaylight supports configuring the underlying virtual switches to act as the Classifier or the Service Function Forwarders for dynamic SFC. Specifically, the OpenDaylight Group Based Policy (GBP) project controls the Classifier and the OpenDaylight SFC project controls the Forwarder. In the next OpenDaylight release, code-named Beryllium (targeted for February 2016), the SFC project will be able to control the Classifier for the CSP NFV use case where there is no tenant virtual machine (VM) that needs GBP control. Standalone SFC patches are already merged into the OpenDaylight mainline and will ship with the Beryllium release.
- **NSH in service functions.** The Linux* networking stack currently does not support NSHs, so VNFs (in this case, the implementation of the actual service function) cannot be implemented using the Linux kernel. VNFs based on the Intel® Data Plane Performance Demonstrators (Intel® DPPD)[10] now support NSH and are used to demonstrate NSH value; other commercial VNFs must do the same in order to be deployed in NSH-based SFC environments.
- **SFC management plane.** If we leverage OpenStack as the Virtual Infrastructure Manager (VIM) defined in the ETSI NFV standard, the VM scheduler in Nova* uses a default scheduling algorithm and filters that are designed for regular tenant VMs to place virtual service functions in a network, in an SFC-unaware manner which is not ideal. The OpenStack Neutron* community is slow to adopt dynamic SFC despite multiple efforts. For some CSPs, direct interaction and control of the SDN Controller (and SFC) from higher layers (such as the management and orchestration layer) is the right architectural approach, somewhat limiting the significance of lack of Neutron support.

  As mentioned before, service function lifecycle management is a necessary component. In the open source community, the OpenStack Tacker project is building an Open NFV Orchestrator with a built-in VNF Manager that manages the lifecycle of VNF instances on the Neutron network. The Tacker project proposes the ability to enable SFC for Tacker and is gaining industry momentum. This approach will be used by the OPNFV SFC community in the “Brahmaputra” release, but is not yet an official OpenStack project.
- **SFC data plane.** The Intel ONP uses OpenDaylight to control Open vSwitch-based vSwitches. Therefore, extension of the OpenFlow and/or the Open vSwitch Database Management Protocol (OVSDB) used by OpenDaylight to control Open vSwitch may be needed. The open source community has asked Intel to use the Nicira Extended Match (NXM) extension to OpenFlow to support push/pop NSHs and decouple them from the transport header push/pop in Open vSwitch; these patches are being reworked. Upstreaming NSH support to the Open vSwitch kernel data plane as well as to the DPDK-netdev user space data plane is an ongoing process that takes time and depends on community acceptance.

  CSPs generally have deployed many NSH-unaware service functions in the SFC production environment. To protect their investment, CSPs may want to mix NSH-aware and NSH-unaware service functions in a hybrid data plane. This raises two challenges for developing an SFC solution:

  – A NSH proxy must be located between the Service Function Forwarder and NSH-unaware service functions to manipulate NSHs on behalf of the service functions.

  – In the Service Function Classifier’s Yang information model in OpenDaylight, there is not yet an NSH proxy entity modeled.
- **Security.** In a production environment, deploying and operating dynamic SFC securely (like any other infrastructure or service element) is a mandatory requirement. Attacks may be performed from the SFC data plane, SFC control plane, SFC management plane, or SFC tenants’ user plane. More sophisticated attacks may involve a coordination of attackers in multiple planes. Vendors may use “standard” security mechanisms and products like those used for any other IP-based technology. However, these days, additional consideration is given in the IETF SFC WG to ensure no new threats are enabled by use of dynamic SFC.

[9] Patches created for this evaluation are currently Intel internal work that is intended to be upstreamed with the respective open source communities; however adoption of those patches cannot be guaranteed.

[10] Intel® Data Plane Performance Demonstrators (Intel® DPPD) are Linux user space applications based on the Intel® Data Plane Development Kit (Intel® DPDK). Intel DPPD is a configurable data plane environment leveraged to characterize actual VNF performance see https://01.org/intel-data-plane-performance-demonstrators/downloads.

Evaluation Setup

Appendix A provides the technical details about how Intel evaluated and tested dynamic SFC in the Gi-LAN context on the Intel ONP. Sections include hardware and software components, general description of the system configuration, bootstrap hosts, bootstrap VMs, and VNF configuration. Scripts that allow system configuration are provided on https://download.01.org/packet-processing/ONPS2.0/Intel_ONP_Server_Release_2.0_Use_Case-GiLAN-1.0.tgz.

Conclusion

While dynamic SFC shows great potential to help CSPs provide new, cost-effective services, the technology ecosystem as a whole is evolving. Intel's effort to implement dynamic SFC using the Intel ONP and NSH highlights several technology gaps:

- Patches for Open vSwitch to become aware of the necessary protocols and to be able to be programmed through the Open vSwitch database are available but not yet committed.
  - Currently, Intel has provided these patches to the Open vSwitch community and is working to implement the community's feedback.
  - Push/pop actions are needed for NSH and VxLAN-GPE headers.
- The Linux network stack must be aware of the presence of NSH when it runs inside a VM. As a workaround, we have used the Intel ONP with an NSH-aware application (Intel DPPD) that is based on the Intel DPDK. Because this approach is NSH-aware and accelerates Open vSwitch, it improves the end-to-end SFC performance for both Service Function Forwarders and service functions.
- Our evaluation at this time does not use OpenStack or Neutron, as mentioned above. We intend to continue our work to include OpenStack as part of the solution stack.

We have made good progress, but development of dynamic SFC, and, in particular, capabilities for the Gi-LAN use case, is needed before technology matures and broad commercialization can occur. Intel continues to work with the open source community to make dynamic SFC for the Gi-LAN use case a viable large-scale deployable solution. In addition, Intel continues the study of the Gi-LAN as part of the Intel ONP program, where the Intel ONP reference architecture is used as the base platform for testing and benchmarking. We anticipate publishing additional reports over time as part of this analysis.

Call to Action

Please join us in advancing the open source and industry standards, and available technology for dynamic SFC. Intel is looking forward to collaborating with CSPs and solution vendors in proofs of concept on SFC.

For more information, visit the following websites:

- Dynamic SFC cross-community efforts: datatracker.ietf.org/doc/draft-ietf-sfc-nsh
- OpenDaylight efforts related to SFC: wiki.opendaylight.org/view/Service_Function_Chaining:Main#SFC_101
- SFC integration work done by the OPNFV community: wiki.opnfv.org/service_function_chaining
- Manageability of SFC by OpenStack Tacker: https://wiki.openstack.org/wiki/Tacker
- The Intel ONP and the integrated solution: 01.org/packet-processing/intel%C2%AE-onp-servers and networkbuilders.intel.com/onp

Appendix A: Evaluation Details

Hardware and Software Components

Intel® server reference platforms (code-named Wildcat Pass) were used in this evaluation. The switch hardware used was an Extreme Networks 10 GbE switch. Table 1 provides the server hardware configuration. Table 2 lists the software components. Note that OpenStack was not used in this setup.

Overview of the System Configuration

As part of the evaluation, we developed scripts (see Table 3) for configuring the system. A sample `prepare_sys.sh` script is provided for guidance to the reader as to how the base system can be configured. We also used scripts to configure the OpenDaylight controller server and Service Function Forwarder nodes, as documented in the `prepare_controller.sh` and `prepare_node.sh` sample scripts. We used the sample script `prepare_json.sh` to control runtime configuration options. This script uses OpenDaylight's JSON REST-API. Refer to the OpenDaylight JSON REST-API documentation for expanded details on the fields available.

We ran tests to evaluate the functionality of a dynamic service chain using the system. To validate this functionality, we used pktgen-DPDK to create and send packets to the traffic classifier across 10 GbE ports. We used Wireshark*[11] to inspect the accuracy of the traffic in the VNF VMs, and to verify the integrity of the original packets, by inspecting the final target.

[11] We used TShark*, a non-GUI version of Wireshark* designed for capturing and displaying packets when an interactive user interface is not necessary or available.

Table 1. Server Hardware Configuration Details

COMPONENT	TYPE	NETWORK CARD
OpenDaylight* Controller	Dell R730*	Intel® Ethernet Adaptor X710
Classifier	Intel® Server Board S2600WTT I	Intel Ethernet Adaptor X710
Service Function Forwarder	Intel Server Board S2600WTT I	Intel Ethernet Adaptor X710
Packet Generator	Dell R730	Intel Ethernet Adaptor X710

Table 2. Software Component Configurations

COMPONENT	VERSION	CONFIGURATION URL
OpenDaylight*	Beryllium with patches to enable IP-less VNFs	git.opendaylight.org/gerrit/#/c/27331/
Open vSwitch*	Commit ID `88058f19ed9aadb1b22d26d93e46b3fd5eb1ad32` and patches that allow it to work as a Service Function Classifier and/or Forwarder	comments.gmane.org/gmane.network.openvswitch.devel/53788
NSH-aware VNFs	Instances of Intel® Data Plane Performance Demonstrators	01.org/zh/intel-data-plane-performance-demonstrators/downloads/prox-application-v021?langredirect=1
Data Plane Development Kit	2.1	www.dpdk.org/browse/dpdk/snapshot/dpdk-2.1.0.tar.gz

Table 3. Sample Automated Configuration Scripts

SCRIPT NAME	SCRIPT DESCRIPTION
`prepare_sys.sh`	Performs the base configuration of a system that is to be used for Gi-LAN testing. This script sets up base networking, installs needed packages, and installs the version of the kernel that is needed for testing. This script adds an exclude line to the updates section of DNF (Fedora* package manager), preventing newer kernels from being installed on the system.
`prepare_node.sh`	Downloads the Data Plane Development Kit[1] (DPDK), Open vSwitch*, and needed patches. The script then compiles and sets up DPDK and Open vSwitch to be used for Gi-LAN testing.
`prepare_controller.sh`	Downloads DPDK, Open vSwitch, OpenDaylight*, and needed patches. The script then compiles and sets up DPDK, Open vSwitch, and OpenDaylight to be used for Gi-LAN testing.
`prepare_flow.sh`	Creates a `.pcap` file for pktgen-DPDK and sets up pktgen-DPDK to generate the traffic.
`prepare_json.sh`	Contains runtime configuration options that are controlled using OpenDaylight's JSON REST-API.

[1] *The Data Plane Development Kit (DPDK) is the industry-standard for packet processing acceleration.*
Source URL for all scripts: https://01.org/packet-processing/intel%C2%AE-onp-servers

We used the following types of traffic in the tests:

- Simple pings of 48 and 64 bytes
- Small flows of 100 packets each
- Large flows of 1,000 packets each

Each flow was captured using Wireshark,[12] listening to the Open vSwitch bridge to verify accuracy. Note that our Gi-LAN test system uses a single Open vSwitch bridge. We found during testing that a dual bridge setup provided significant throughput improvement under a heavy load. As we make progress and address solution gaps, a future report will provide an upgraded Gi-LAN setup script and instructions for the dual bridge setup and will also provide performance results for that setup. We plan to do additional optimization to address issues found during the benchmarking process.

Our initial performance testing focused on measuring the overhead of Service Function Classification and Service Function Forwarding. Important test metrics included throughput, latency, and packet-delay-variation throughout the service chain—in cases where VNFs were on the same node and also where they were on different nodes. We anticipate providing performance results in an updated version of this paper. See comments in the scripts for additional technical details about our Gi-LAN test system.

Bootstrap Hosts

All host machines used in the lab setup (Figure 4) were bootstrapped with the same procedures. All machines ran Fedora* 21, using a default install without running yum update afterwards. After installation, we installed kernel version 4.1.10-200,[13] and all network configurations were set up. We used automated scripts to perform the configuration following the installation. We performed the following steps:

- We installed the Fedora Basic Install package because it installs minimal packaging, allowing for the installation of dependencies later.
- Next, we installed basic packaging dependencies using the following command:
  `yum install @development-tools autoconf automake`
- We used two 10GbE interfaces per service node.

As mentioned earlier, we developed scripts to configure the system. A sample `prepare _ sys.sh` script is provided for guidance as to how the base system can be configured. The OpenDaylight controller server and Service Function Forwarder nodes can be configured with scripting as well. Please also see the `prepare _ controller.sh` and `prepare _ node.sh` sample scripts. Table 3 describes the available scripts and provides the source URL for download.

Bootstrap Virtual Machines

We manually prepared VM images used for dynamic SFC on a separate system running libvirt and Kernel-based Virtual Machine (see the previous section, Bootstrap Hosts, for the default setup). We used a Virtual Machine Manager to create the image files and control the VM during installation of Fedora 21. After Fedora 21 was installed on the system, we modified the `prepare _ sys.sh` script for the VM and then ran the script.

We applied the following configuration to the image:

- Base VM configuration
  - Disable the firewall:
    `#systemctl disable firewalld.service`
  - Edit the grub default configuration:
    `#vi /etc/default/grub`
  - Add the following to GRUB_CMDLINE_LINUX_DEFAULT:
    `… noirqbalance intel _ idle.max _ cstate=0 processor.max _ cstate=0`
    `ipv6.disable=1 default _ hugepagesz=1G hugepages=4`
    `isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19`

[12] Ibid.

[13] To use this specific version of the kernel, it will need to be downloaded from the Koji archive. (Koji is the Fedora* build, packaging, and archival system. See comments in the scripts for more details.)

  - Rebuild grub config and reboot the system:
    ```
    #grub2-mkconfig -o /boot/grub2/grub.cfg
    #reboot
    ```
  - Verify that hugepages are available in the VM:
    ```
    #cat /proc/meminfo
    HugePages _ Total:4
    HugePages _ Free:4
    Hugepagesize:1048576 kB
    ```

- Build the Data Plane Development Kit (DPDK)
  ```
  $ wget http://dpdk.org/browse/dpdk/snapshot/dpdk-2.1.0.tar.gz
  $ tar xvzf dpdk-2.1.0.tar.gz
  $ export RTE _ SDK=$(pwd)/dpdk-2.1.0
  $ export RTE _ TARGET=x86 _ 64-native-linuxapp-gcc
  $ export DPDK _ BUILD=${RTE _ SDK}/${RTE _ TARGET}
  $ cd ${RTE _ SDK}
  # make install T=${RTE _ TARGET}
  ```
- Bind the virtual ports to DPDK
  *Note: Replace **<nic 1>** and **<nic 2>** with the named interfaces of your system.*
  ```
  # modprobe uio
  # insmod $RTE _ SDK/$RTE _ TARGET/kmod/igb _ uio.ko
  # $RTE _ SDK/tools/dpdk _ nic _ bind.py -b igb _ uio <nic 1>
  # $RTE _ SDK/tools/dpdk _ nic _ bind.py -b igb _ uio <nic 2>
  ```
- Build the Intel DPPD
  ```
  $ wget -c https://01.org/sites/default/files/downloads/intel-data-plane-performance-
  demonstrators/dppd-prox-v021.zip
  $ unzip dppd-prox-v021.zip
  $ cd dppd-PROX-v021
  $ export PROX _ DIR=$(pwd)
  $ make
  # ./build/prox -f ./config/nop.cfg
  ```

VNF Configuration

To boot the created VM image, we used the following command line on one of the VNF nodes:

```
sudo qemu-system-x86 _ 64 -hda [image _ file.qcow2] -vnc :9 -m 4096M -cpu host -enable-kvm -boot c
```

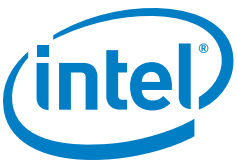
After booting the VM, we attached to it with a Virtual Network Computing (VNC) client. The address to connect to is the IP address of the VNF server, followed by the port for the VNC connection. The command above specifies port 9 with the `-vnc :9` option. So the address to connect to is `serverip:9`.

In some cases, the full VNC port must be specified. VNC starts connections at port 5900 and increments according to the specification. In the case of the above command, the full port number would be 5909.

After rebooting and connecting to the VNF, we made sure that we bound the VNF network interface cards (NICs) to the DPDK (as stated in the "Bind the virtual ports to DPDK" bullet above). We also made sure that we executed the `prox` command as shown in the "Build the Intel DPPD" bullet above. These two steps are necessary because neither of these processes is persistent and neither will survive a VNF reboot without any extra scripting.

Acronyms

3GPP	3rd Generation Partnership Project
APN	access point name
Intel® DPPD	Intel® Data Plane Performance Demonstrators
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
ETSI	European Telecommunications Standards Institute
GBP	OpenDaylight Group Based Policy
IETF	Internet Engineering Task Force
Intel® ONP	Intel® Open Network Platform
MPLS	Multiprotocol Label Switching
NAT	Network Address Translation
NFV	network function virtualization
NFVI	network function virtualization infrastructure
NIC	network interface card
NSH	network service header
NXM	Nicira Extended Match
OPNFV	Open Platform for NFV
OVSDB	Open vSwitch Database Management Protocol
PCRF	Policy and Charging Rules function
PNFs	paid-for physical network functions
SCTCF	Service Chain Traffic Controller function
SDN	software-defined networking
SFC	service function chaining
TCP	Transmission Control Protocol
VIM	Virtual Infrastructure Manager
VM	virtual machine
VNF	virtualized network function
VNC	Virtual Network Computing
vSwitch	virtual switch
VxLAN-GPE	Virtual Extensible LAN-Generic Protocol Extension



Printed in USA 0116/JLAW/KC/PDF ♻ Please Recycle 333802-001