L4-L7 Service Function Chaining Solution Architecture

Version 1.0
14 June 2015

ONF TS-027

ONF Document Type: Technical Specifications
ONF Document Name: L4-L7 Service Function Chaining Architecture

Disclaimer

Contents

List of Figures

1. About This Document

1.1. Purpose of This Document

This document describes the architecture of L4-L7 service function chaining (SFC), and the information necessary for the SDN Controller's northbound interface (NBI) and southbound interface (SBI). The intent is to build a common base for concrete NBI specifications and OpenFlow (OF) extensions needed for SFC.

1.2. Intended Audience

ONF NBI, Extensibility (EXT) Working Groups, and ONF member companies.

1.3. Executive Summary

Sections 2 through 4 describe the problem statement, terminologies and provide a detailed look at the SFC architecture. The reviewer should review these sections for their accuracy and for baseline understanding of the concepts and terms as used in this document.

The reader is encouraged to review sections 5 and 6. Section 5 describes what is possible today with OF 1.4 and 1.5 to implement service function chaining. Section 6 illustrates such a use case which shows how it is already possible to implement SFC with OF 1.5 today. Section 5 also describes additional work necessary to support enhanced SFC features including L4-L7. The recommended enhancements include the NBI of the SDN controller, and a number of OpenFlow extensions (Section 5.5.1) that are necessary to support the controller's interaction with the classifiers, head-end and terminating service function forwarders (SFFs), SFF proxies and SFC headers. The recommendations here will entail further development work with other ONF working groups, in particular the NBI and EXT groups.

2. Problem Set and Drivers

End-to-end application traffic flows are often required to traverse various network services such as IPS/IDS, firewalls, WAN optimizers and load balancers along a predetermined path. For example, a user flow accessing an application server at a remote site may need to go through WAN optimizers for performance, firewalls and an IPS for security, load balancers for performance and high availability, and a VPN server before reaching the application server.

Traditional networks face a number of inefficiencies:[1]

1. **Topology constraints:** Traditional network service deployments are highly dependent on the network topology. Network elements are linked together by a complex and rigid set of configurations to create a service path. Making changes to this highly complex environment such as dynamically inserting a new service function, adding a service function instance for scaling, etc. tends to be a complex, resource intensive and time consuming effort.
2. **Fixed service chains:** Topology constraints force all traffic through the rigid set of service functions, even if some of the traffic may require only some or none of the service functions; or if only the first few packets may need to traverse certain service functions and subsequent packets may bypass them. This rigid behavior imposes unnecessary capacity and latency costs.
3. **Complex configuration:** Topological dependencies make the entire configuration far more complex, making troubleshooting harder.
    a) **Consistent ordering of service functions:** Ensuring traffic flows go through a set of service functions is often a slow and error-prone process.
    b) **Symmetric traffic flows:** Many network services such as firewalls require bi-directional traffic flows. Existing methods to ensure symmetric traffic flows often require complex configuration of each network service function along the path.
4. **Traffic classification and policy enforcement:** Classification capabilities on most service functions today are coarse: one or some combination of L1-L4 parameters (e.g., interface, VLAN, IP, port, etc.) Coarse classification often results in coarse policy enforcement. That is, the traffic that does not require service enforcement still has to traverse the service functions, and may unnecessarily be subject to policy enforcement.
5. **Constrained HA:** Since network services are often shared by several applications, any change in behavior or capacity requirements of an application flow could negatively impact flows of other applications traversing the same service.
6. **Complex scale-out architecture:** Certain service functions that require symmetric traffic flows and/or stateful flows such as L4-L7 load balancers and firewalls are notoriously difficult to scale out. It requires careful planning to add these services in highly rigid topology-constrained environments.
7. **Limited visibility for troubleshooting:** Troubleshooting L4-L7 service related issues requires administrators versed in both L2-L3 and L4-L7 service functions. Troubleshooting in virtual environments operating over multiple layers of overlay/underlay networks requires tools with better visibility into the underlay networks, as well as those that can provide end-to-end views.

---

[1] See [SFC-PS] for details on the SFC Problem Statement.

The degree of complexity increases exponentially in service provider environments in which several tenants/clients each with their own requirements co-exist. These various tenants may subscribe to their own set of services, which are often provided by service function devices shared amongst several tenants. The degree of complexity increases further still when these clients wish to dynamically insert, remove, or reroute traffic depending on their traffic and security policies as well as service level agreements.

SDN-based service function chaining needs to enable solutions for the above problem set.

3. Terminologies

Most of the terminologies used in this document are from [SFC-ARCH], [SFC-FWK] and [SFC-META], and are summarized here for convenience:

- **Service Function (SF):** A network function that provides a value-added service to traffic flows. A service function may perform its function(s) at one or more OSI layers. Service functions include: firewall, DPI, NAT, HTTP Header Enrichment function, TCP optimizer, load-balancer, IDS, IPS, NAT etc.
- **SF Chain (SFC):** An ordered list of Service Function instances. Each SF Chain is identified with a unique identifier called an SF Chain Identifier. The SFC is also called VNFFG (Virtual Network Function Forwarding Graph).
- **SF Chain Identifier:** Identifies an SF Chain.
- **SFC Classifier (or Classifier):** An entity that classifies traffic flows for service chaining according to classification rules defined in an SFC Policy Table. Frames/packets, of the flows, are then marked with the corresponding SF Chain Identifier. SFC Classifier can be embedded in an SFC Boundary (Ingress) Node or SF proxy node in which case the Classifier will do the encapsulation after getting the L2/L3 frame/packet from a Legacy SN. The SFC Classifier can run on an independent (physical or virtual) platform. The SFC classifier can be on a data path, and can also run as an application on top of a network controller.
- **SFC Header:** A header that is embedded into the flow packet by the SFC Classifier to facilitate the forwarding of flow packets along the service function chain path. This header also allows for the transport of metadata to support various service chain related functionality.
- **SF Node:** Denotes any node within an SFC-enabled domain that embeds one or multiple SFs.
- **Service Function Instance (SFI):** Denotes an instantiation of a service function, such as firewall, on a service node. A service function can have multiple service instances running on the same SF node with each service instance having its own service profile.
- **Service Function Forwarder (SFF):** Provides service layer forwarding. An SFF receives frames/packets carrying SFC header and forwards the frames/packets to the associated SF instances using information contained in the SFC header.
- **SF Proxy:** A Network Element along the SF Chain path to facilitate the operation of legacy SF nodes. It will strip off the SFC header from the flow packet and forward the original flow packet to the SFI running on the legacy SF node. When the legacy SF node returns the packet to the Proxy, the Proxy shall correlate the packet with the SF Chain and add back the SFC header. SF Proxy is usually placed right before a legacy SF node.
- **Metadata:** Provides contextual information about the data packets which traverse a service chain. Metadata can be used to convey contextual information not available at one location in the network to another location in the network where that information is not readily available. While primarily intended for consumption by SF(s), metadata MAY also be interpreted by other SFC entities.
- **SFC-enabled domain:** Denotes a network (or a region thereof) that implements SFC.
- **SF Identifier:** A unique identifier that unambiguously identifies an SF within an SFC-enabled domain. SF Identifiers are assigned, configured and managed by the administrative entity that operates the SFC-enabled domain. SF identifiers can be structured as strings, or in other formats. SF Identifiers are not required to be globally unique or be exposed to or used by another SF-enabled domain.

- **SFC Policy Table:** A table of classification rules. Each classification rule contains matching traffic parameters (such as 5-tuples, MAC addresses etc..) and action of the rule identifying the SFC and the service functions to be applied to the traffic.

4. SFC Architecture

The SFC architecture consists of management, control and data plane elements. The table below lists areas under which common SFC functional elements can be categorized. Note that the elements here are listed as separate entities from a functional perspective only. The functions may exist on separate platforms or on a single platform, which in turn may exist in physical, virtual, software or any other form factors.

Plane	Examples
Management	SFC Orchestrator, i.e. SFC Manager and SF Instance Manager SFC applications (e.g., SFC OAM functions)
Control	SFC SDN controllers
Data	Classifier SF Forwarders, SF Proxy SF Nodes

The elements can be viewed as follows:

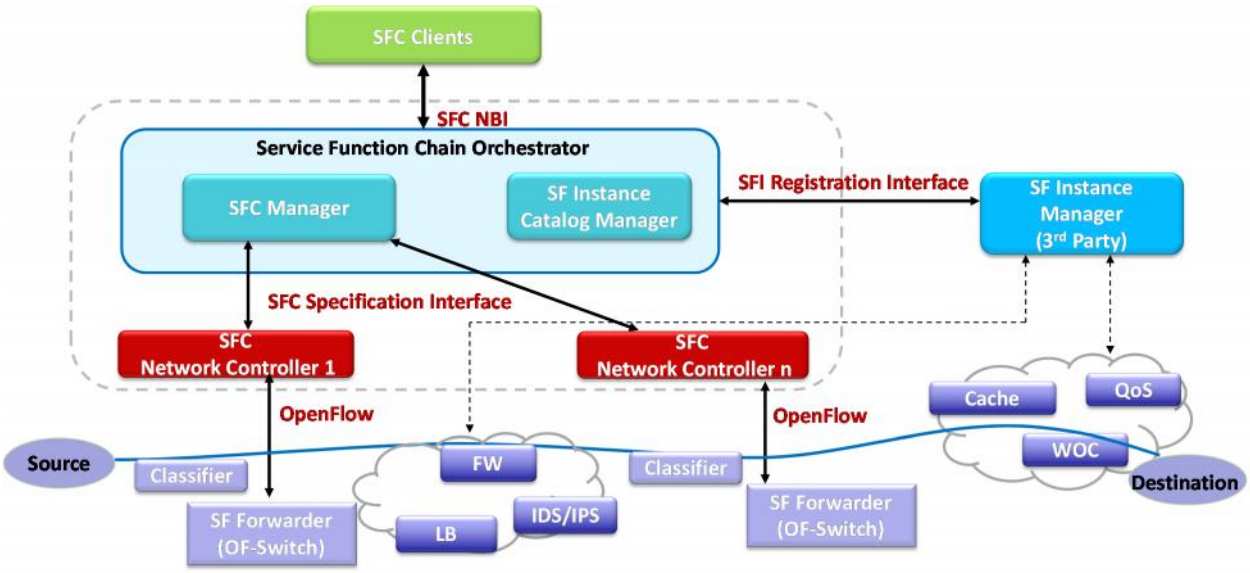

Figure 1: L4-L7 SDN SFC Architecture

SFC management plane elements together with the control plane elements are responsible for translating user/application requests; converting requisite policies into network topology dependent paths; and disseminating steering policies to relevant forwarding nodes. SFC data plane components are responsible for carrying out the steering policies.

The role of the SF Instance Manager is to manage instantiation, configuration and decommissioning of SFs, as well as registration with SFC orchestration. Since SFs by different vendors are typically configured

differently, it is necessary to have the SF Manager (usually vendor specific) to configure the SFs, and pass relevant network information (i.e. addresses, types, etc.) to SFC orchestration.

In the ONF SDN paradigm, the SFC Orchestrator requests necessary changes through the northbound interface of the Network Controller, which in turn implements the requested changes via OpenFlow. If service functions are not OpenFlow-speaking or are not under the domain of OpenFlow Network Controllers, the Orchestrator may need to interact directly with the service functions themselves or with the SF Instance Manager as shown in Figure 1.

4.1. SFC Orchestrator

The SFC Orchestrator translates clients' high-level abstract intent-based network-topology agnostic requirements into flow classification constructs, and sequences of L4-L7 service function constructs associated with the flows. The Orchestrator provides an intent-based NBI for service clients to specify SFC requirements including the source endpoint, the destination endpoint, sequence of service function treatment, policy constraints, etc.

The Orchestrator consists of two key components: the SF Chain Manager and the SF Catalog. The SF Chain Manager is responsible for translating the user's SFC intent/requirement on the sequence of service function treatment between a source endpoint and a destination endpoint into flow classification constructs and associated sequence of L4-L7 service function instance constructs and passing these constructs to the Network Controller. The SF Catalog hosts a pool of the L4-L7 service function instances that are available in the network domain. For each service chain request, the SF Chain Manager will find out all the service function instances from the pool whose capability satisfies the service chain requirement, and pass the list of available service function instances to the Network Controller.

4.2. SF Instance Manager

The SF Instance Manager is responsible for managing the life cycle of each service function instance, and for keeping track of each service function instance's locator and chain header encapsulation and transport capabilities. The Instance Manager will register each SF instance with the SF Chain Manager.

4.3. SFC Network Controller

The Network Controller is responsible for setting up service function steering/chaining paths. It will find out the best SF instance out of the available SF instances and translates the sequence of service function constructs into southbound OpenFlow programming commands. It locates the flow classifier, instructs it how to classify a flow, and encapsulates the flow packet with an appropriate SFC header. It is also responsible for locating the sequence of SF Forwarders associated with the sequence of SF instances so as to enforce the flow going through the sequence of designated SF Instances. That is, the Network Controller will use those OpenFlow programming commands to program the classifier to classify the traffic flows and program the SF Forwarders to steer the flow through the sequence of SF Instances.

4.4. SFC Classifier

This refers to a logical entity that classifies traffic flows based on policies (e.g., n-tuple fields) and inserts an SFC header containing a path ID into the flow packet headers [SFC-SCH] [SFC-NSH]. This logical entity can run on a separate device or on the same device as the Service Entity and/or Steering Entity. Traffic can be reclassified at any point along a service chain. An SFC may have multiple classifiers.

For legacy L4-L7 service functions, it is necessary to have proxy nodes to remove the SFC header that are not recognized by legacy L4-L7 functions, as shown in the figure below:

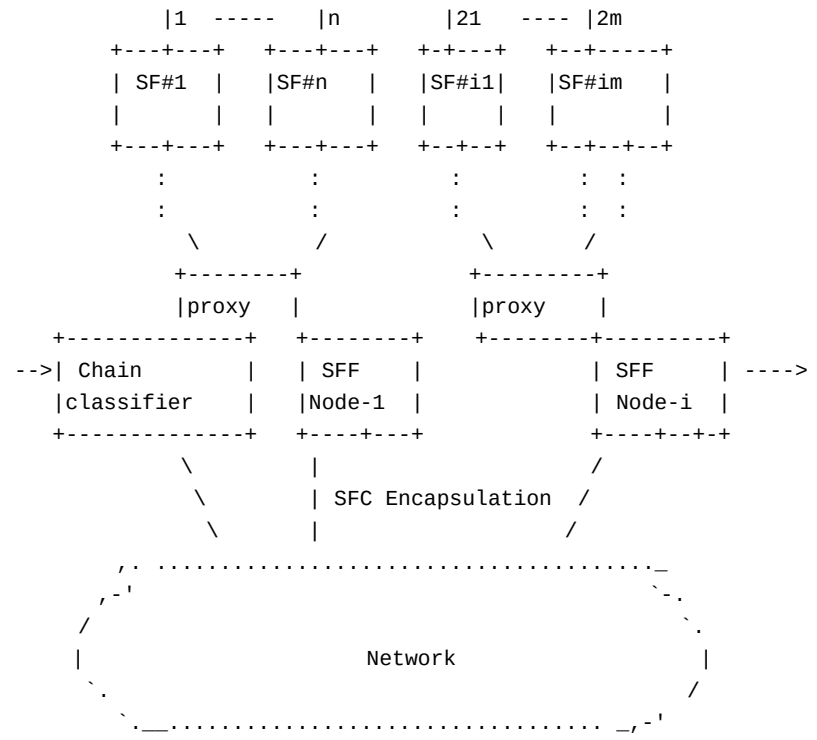

Figure 2. Legacy L4-L7 Service Function Chain Architecture

When the SFC header encapsulation is not supported, flows on different service chain can be differentiated by some fields in the packets.

4.5. Service Function Forwarder

The Service Function Forwarder (SFF) is responsible for forwarding data packets to their designated service function instances. As the case with other SFC entities, the SFF may come in physical, virtual or any other form factor. The SFF may make its forwarding/steering decision by matching the chain identification information carried in the SFC header with the next-hop information provided by the Network Controller. The SFC header should carry sufficient information for each SFF in the path to establish symmetric flows in forward and reverse directions, if necessary.

In this architecture, the SFF may be any OpenFlow-capable device that *preferably* understands the SFC header. To be sure, the SFF can still make its steering decisions based on just non-SFC-header-based fields, without using any SFC headers. But there are limits to this approach; for instance:

- OpenFlow 1.3+-capable SFFs may make forwarding decisions on L2-L3 header fields such as MAC or IP addresses. With an OpenFlow version that supports L4 fields such as TCP and UDP ports, the SFF will be able to make forwarding decisions at L4.
- It is far more difficult for SFFs to make forwarding decisions using fields above L4. It requires not only OpenFlow's support of L5-L7 fields but also L4-L7 proxy and DPI capabilities in the SFFs themselves.

Even if more sophisticated capabilities were available in both OpenFlow and SFFs to support L4-L7 fields, it would be expensive to implement. The SFFs in the path would have to hold L4-L7 forwarding tables in addition to L2 and L3 forwarding tables. Depending on policy, even L4 tables, not to mention L7 ones, could easily explode in size. By using an SFC header, the task of classifying based on L4-L7 or any other metadata is offloaded to the classifier, enabling the SFFs in the path to match on just the SFC header alone.

Note that a single SFF may be attached to several SF types or several instances of the same SF, and may support multiple chains. SFFs attached to legacy SFs will need to perform SF Proxy functionality.

4.6. Service Function Node

The Service Function Node is a logical entity that may consist of one or more instances of a particular service function. SF instances may be hosted on various form-factors: physical appliance, virtual machine, container, software, etc. SF nodes (and instances) may or may not understand SFC. SFC proxies are needed to front non-SFC-speaking (legacy) SF nodes/instances.

4.7. Connectivity between SFF and SF

4.7.1. Topology

The SF instances can be connected to SFFs (steering nodes) in several ways: one-armed, bump-in-the-wire (BITW), embedded-in-the-SFF, etc.

Figure 2: Main SF to SFF Connection Methods

In one-armed mode, the SFF forwards the packets to the selected SF instance's link address (e.g., Ethernet MAC address). After performing its function, the SF instance returns the packets to the originating SFF's link address. The SFF then forwards them to their next destination.

In BITW mode, the SFF forwards the traffic to the link address of the SF instance. The SF instance performs its function, and forwards it to the next SFF's link address.

Note that SFF and SF instances may be embedded on the same platform (router/switch). Moreover, SF proxies will be needed if SF instances do not understand SFC.

4.7.2. Mechanisms

The link between the SFF and the SF can be a physical link or logical one. The SFC architecture is agnostic as to how the packets are transported between the SFF and SF. Nor does it require any support of overlay technologies by the SFs. Although SF instances are expected to be directly connected to the SFF in most cases, if the SFs and SFFs mutually support an overlay mechanism (e.g., VXLAN, MPLS over GRE) between them, the SFC architecture does not prohibit the use of said transport.

4.8. Connectivity between SF Proxy and SF

In many cases, SFs may not understand the SFC header. An SF proxy function is needed to support legacy SFs. The proxy function may be attached to or embedded in an SFF. An external SF Proxy serves as the proxy for SF instances.

4.8.1. SF Proxy to SF

The SF Proxy de-encapsulates the SFC header from the packets if the SFC header, extracts the service chain identifier from the SFC header, maps the service chain identifier to a locally significant tag or header that is recognizable by the legacy SF, and encapsulates the tag or the header to the data packets before sending the packets to the SF.

Locally significant means the tag or the header is only local to the link/path between the SF Proxy entity and the SF, and is capable of differentiating packets from different service chains that traverse the link/path. Examples of locally significant tags include VLAN IDs, GRE keys and logical/virtual port IDs such as VIF or Ether-Channel number, etc. Examples of locally significant headers include encapsulating IP headers, MAC headers, etc.

In some cases, the legacy SF may need the metadata inside the SFC header. The SF Proxy shall send the metadata to the SF if a mutually supported method of delivery exists.

4.8.2. SF to SF Proxy

The SF Proxy reconstructs the SFC header using the locally significant tag/header information of the returning packets from the SF, encapsulates the packets inside the reconstructed header, and forwards the packets to the next SFF.

4.9. SFC OAM Functions

The Service Chain OAM functions refer to the functionality of collecting and analyzing the SFC path status, detecting failures, and establishing SFC restoration paths.

4.10. SFC Identifier

The IETF SFC WG is working on a new header to carry the needed service chain identifier and metadata. However, it is also possible to use existing data packet headers, such as VLAN, MPLS label, source MAC addresses, etc. to differentiate service chains if carrying metadata is not required by the service chains [L4-L7-Chain]. For example, in the deployment scenario below: A large block of MAC addresses are allocated to the Chain Classifier node. The Chain Classifier node can use any of the allocated MAC addresses in the Source Address field of the data frame to differentiate flows belonging to different chains. An Out-of-band message(s) can be exchanged between the Chain Classifier node and SFF Nodes to signal the chain associated to each Source MAC address.

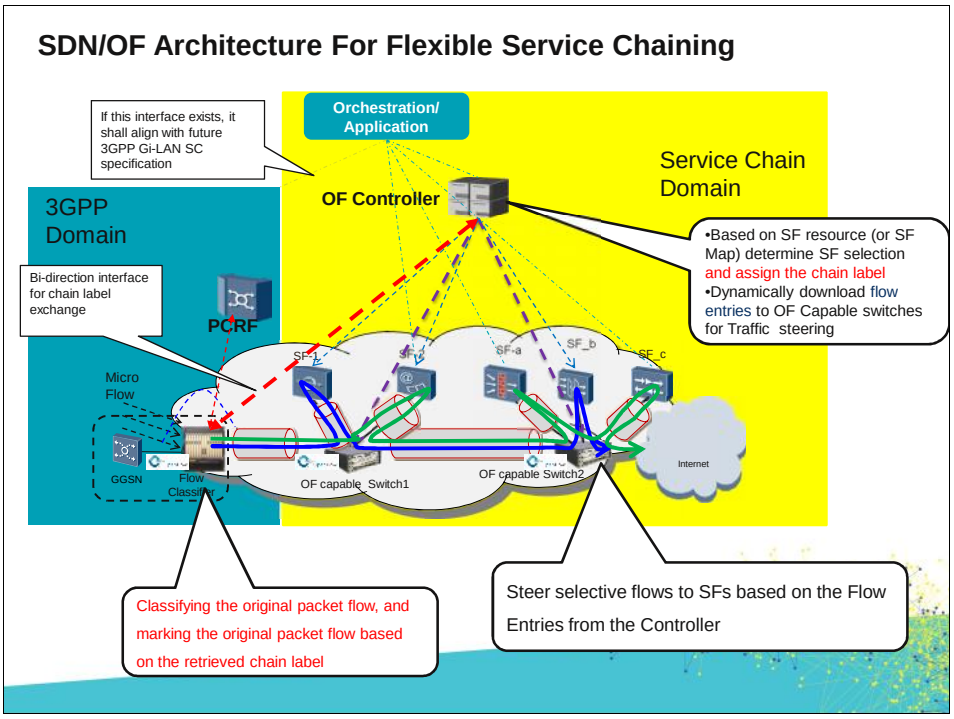

Figure 3: Classifier for Legacy L4-L7 SFCs

More extensive analysis of using existing data headers to differentiate service chains are analyzed by http://datatracker.ietf.org/doc/draft-boucadair-sfc-design-analysis/.

4.11. Metadata Considerations

Technically speaking, everything carrying the information that is not in the payload is metadata. The IETF has standardized several types of metadata exchanged among L2/L3 nodes such as IPv4 DSCP bits, MPLS labels, etc. Metadata in the SFC sense means something more: "the information added to the packet to be carried along with the packet for the consumption of the service function nodes along the chain".

This section describes relevant metadata concepts and considerations.

- Metadata may be transported in-band or out-of-band over various methods/protocols.
- In-band metadata may be transported inside existing header fields, or in additional SFC header metadata fields. Metadata can simply be carried as part of existing header fields—e.g., DSCP bits in the IPv4 header. This approach does not require adding any extra bytes to the packets. But sending more flexible metadata may require adding more bytes, for example by using TCP options fields, or the metadata fields in proposed IETF SFC header standards like the Service Chain Header [SFC-SCH] or the Network Service Header [SFC-NSH]. Adding additional information—whether via TCP options fields or metadata fields—must take path MTU and fragmentation issues into account.
- In-band metadata may be transported at different OSI layers, and not all elements in the SFC path may have the capability to recognize and/or act upon the metadata.

- Transport (L1-L4) metadata: Elements in the SFC path such as the SFF or SF Proxy can recognize and may use inherent L1-L4 header information to derive forwarding decisions or some other actions as required by prevailing policy. With frame/packet-layer metadata, SFC entities may apply policy on a frame-by-frame or packet-by-packet basis. Likewise, the entities may use flow-based metadata to apply policy on flows.
- Application (L5-L7) metadata: Recognition of metadata carried over upper layers by intermediate entities in the SFC path requires far more sophisticated capabilities in those entities. The entities need to be able to proxy the traffic at least at L4, and have DPI capabilities to inspect application layer protocols such as HTTP or SIP, e.g., to examine HTTP cookies or SIP session IDs. Even then, the entities may not be able to read further inside—e.g., if the payload is encrypted.
- Some proposed IETF SFC headers like the SCH and NSH allow the header to be carried inside L4 tunnels such as VXLAN. (The proposed headers are agnostic as to which layer it is carried over.) In such cases, the SFF needs to have the proxy capability to reconstruct the entire header including the metadata fields at L4.
- In most cases, SFC elements will not be able to recognize and act upon metadata carried over at upper layers above L3.

5. SFC Information Model and Interfaces

This section discusses the overall SFC information model, and identifies required components and interfaces to access them. The required information between the Network Controller and the Service Chain Orchestrator will be provided to the ONF NBI Working Group. The required information between the Controller and the forwarding/service function nodes will be passed on to the ONF EXT Working Group as needed extensions to OpenFlow.

5.1. Interface between SFC Orchestrator and SFC Client

As shown in Figure 1, this interface refers to the interface between the SFC Client and the SFC Orchestrator.

This is the interface for a client to specify its high-level requirements/intention of an ordered sequence of service function types and flavors that should be applied to a traffic flow between a source and a destination. To make it easy for the user and reduce the OPEX, this interface should be network technology unaware and network topology unaware. The SFC Orchestrator consumes this high level intent specification and translates it into the next abstraction level SFC representation. This Intent API is based on an Intent model which includes the following entities:

1. Service Function Chain Intent. This entity specifies which sequence of service function treatment needs to be applied to the traffic between a source and a destination. It consists of the source Composite Endpoint, the destination Composite Endpoint, and an Intent Operation.
2. Composite Endpoint (CEP). A composite endpoint consists of a Boolean expression consisting of operands (terms) and the Boolean operators: and, or, not. The Boolean operands are registered endpoint descriptor IDs and/or the IDs of other Composite Endpoints. Parentheses may be used for the purpose of grouping. Each Composite Endpoint can be of any granularity. The source Composite Endpoint combined with the Destination Composite Endpoint identifies the flow in between. Following is a Composite Endpoint Description:

   Composite Endpoint 3 = "(EPD1 or EPD2) and not CEP22"

3. Endpoint Descriptor (EPD). Each endpoint descriptor has a type and value that are used to partially identify the endpoint. The Endpoint Descriptor ID is used as an operand (term) in a Composite Endpoint's Boolean expression. The type of the Endpoint Descriptor may be a base networking identifier, such as an IP address, or a user-defined type. The value may be a single value, a range of values, or a group of values.
4. Operation. This specifies the operation applied to the traffic that matches the source Composite Endpoint and the Destination Composite Endpoint. The Operation type is set to “service chain”.
5. Service Function Chain Specification. This is a Service Function Chain specification associated with the “ServiceChain” type Operation. The specification consists of an ordered list of Service Functions with each represented by service type and flavor

5.2. Interface between SFC Orchestrator and SF Instance Manager

As shown in Figure 1, this interface refers to the interface between the SFC Orchestrator and the SF Instance Manager which could be any L4-L7 Service Provider. The SF Instance Manager registers the SF instance’s information with the SF Orchestrator. This interface should include the following SF instance information:

- SF Instance Type: The type of service function; e.g., FW, LB, IPS
- SF Instance Locator: Represents the addressing information used to reach an SF instance.
- SF Instance Flavor: Different flavors of a service function type.
- SF Instance Group: Each SF instance can be either associated with an SF group or not associated with any SF group. An SF Group is used to provide a local load-balancing capability over a group of SF instances.
- SF Chain Header Support: This includes information about whether an SF Instance support a standard SFC header format or not. In the data-plane, an SF instance may be SFC header aware or SFC header unaware. If it is unaware, then an SF Proxy needs to front the SF instance. It may also include information on what kind of chain ID-flow correlation mechanism it supports.
- Mechanism between the Service Instance and its connecting switch for chain correlation if the Service Instance cannot interpret the service chain header which includes the chain identifier.

5.3. Interface between SFC Manager and Network Controller

As shown in Figure 1, this interface refers to the interface between the SFC Manager (part of the SFC Orchestrator) and the Network Controller. The SFC Manager will pass the client's abstract intent-based service function requirements as well as a list of available SF Instances to the Network Controller. All the management of SFFs and setup of traffic steering path for service chain are done internally within the Network Controller.

This interface should include the following entities:

- Sequence of Requested Service Functions: This specifies the sequence of service function treatment for a tenant's traffic flow.
- Traffic Flow Classification rules: It consists of a set of flow descriptors.
- List of Available Service Function Instances: each SF Instance contains information such as the Instance's Locator, the Instance's Flavor, Instance's Group, Instance's Tunnel Transport Capability. An example of an SF Instance is a firewall or an IDS.
- A weighing factor for each SF instance within an SF Group that can be used for load distribution. The weight may be derived in a number of ways—e.g., a loading level of an SF instance, or a policy decision made at the SFC Manager.

  From an OpenFlow implementation standpoint, this SF instance-level weight can be used in conjunction with OpenFlow Groups (OF Groups) to implement load distribution. OF Groups already support the selection of "buckets" within a Group using a "switch-based selection algorithm" such as hashing. Each bucket has a weight associated with it to specify the relative precedence of that bucket. It means that an SF Group can be implemented as an OF Group in which individual member buckets represent corresponding SF instances; and that the weight values for each SF instance supplied by the SFC Manager can be mapped directly to those of the buckets in the OF Group.

5.4. Network Controller SBI – What is Possible Today

The Network Controller may enable service function chains by affecting some or all of the following components in an SFC: (1) SFC Classifier; (2) SF Forwarder; (3) SF Proxy; and (4) SF. In this architecture, the controller could use an enhanced version of OpenFlow to affect all of the aforementioned components.

This section will discuss what enhancements are needed, and their implications. The recommended extensions to OpenFlow will be passed to the ONF EXT Area.

5.4.1. With OpenFlow 1.4

OpenFlow 1.4 already supports classifying applications with dynamic TCP/UDP Source/Destination port numbers. OpenFlow needs to support TCP and UDP header fields—TCP flags and packet size, in particular.

- TCP flags matching
  - `EXT 109: Match on TCP flags`
  - `Add OFPXMT_OFB_TCP_FLAGS to oxm_ofb_match_fields`
- Packet size matching:
  - `EXT-441: Need a new matching criterion for packet size`
  - `Add new field “packet size” to oxm_ofb_match_fields`

5.4.2. With OpenFlow 1.5

Although the use of SFC classifiers and headers will greatly enhance SFC capabilities, the standards are still being worked out in the IETF.

With OpenFlow 1.5, it is still possible to implement SFCs using:

- Fixed L2-L3 header-based forwarding: E.g., L2-L3 header fields such as the MAC or IP source/destination address, MPLS label, VLAN ID.
- TCP/UDP port based forwarding

It means the Network controller can program the forwarding elements (switches, routers) up to L4 (ports). Without an SFC header, a classifier is not needed. Each forwarding plane element will classify the incoming traffic up to L4 ports, and make forwarding decisions based on its forwarding table programmed by the Network controller.

5.5. OpenFlow Extensions

5.5.1. Overview

OpenFlow needs to support at least one kind of SFC header. As discussed in previous sections, the SFC header is calculated and inserted into data packets by the SFC classifier. (The header in theory can be inserted at L4 and L5 but in practice will likely be inserted at L2 or L3.) The header alleviates the rest of components in the chain (SFFs, SF Proxies, SFs) from having to maintain the same classification capabilities as the classifier, and allows them instead to understand just the header. The header also frees the Controller from the need to supply the policy database to each component in the path; instead, the Controller can now provide the policy database just to the classifier.

The use of SFC headers simplifies the job of OpenFlow—not just but especially with L4-L7 service functions. Instead of needing to understand myriad L4-L7 variations, OpenFlow needs to understand just the SFC header.

Even if more sophisticated capabilities were available in both OpenFlow and SFFs to support L4-L7 fields, it would be expensive to implement. The SFFs in the path would have to hold L4-L7 forwarding tables in

addition to L2 and L3 forwarding tables. Depending on policy, even L4 tables, not to mention L7 ones, could easily explode in size. By using the SFC header, the task of classifying based on L4-L7 or any other metadata is offloaded to the classifier, enabling the SFFs in the path to match on just the SFC header alone.

In sum:

- The classifier needs to be able to insert the SFC header including metadata inside the SFC header;
- The elements in the SFC (SFFs, SF Proxies, non-legacy SF instances) must be able to read the SFC header;
- OpenFlow needs a field that allows the Controller to program the forwarding table of SFC elements.

The OpenFlow control plane interface is between the Network Controller and the Classifiers, Service Forwarding Functions, Service Function appliances and Proxy Devices. Existing OpenFlow match commands can be used to support the Classifier N-tuple matching. Existing OpenFlow action commands can be used to forward packets out a specific interface.

The OpenFlow Experimenter Message, match and action extensions to the OpenFlow messages are required to transport the SFC Header (SFCH, e.g., SCH/NSH) information. A new Experimenter ID will be registered with the ONF for these structures.

The following Experimenter extensions are used to support Service Function Chaining:

1. Flow Match: This is used by an SFF to match the SFCH Path ID and Path Index fields in incoming packets.
2. Push SFCH Action: The Push SFCH action is used by a Classifier to insert an SFC header and any optional metadata into the outgoing packets at the start of a service chain. The Push SFCH action is also used by a Proxy Device to insert an SFCH into packets received from a legacy Service Function appliance.
3. The Pop SFCH action is used by a Terminating SFF to remove the SFCH from the outgoing packets at the end of a Service Chain. The Pop SFCH action is also used by a Proxy Device to remove the SFCH from packets that are delivered to a legacy Service Function appliance.
4. Set Service Path ID, Service Path Index and metadata fields in the SFCH.
5. Increment the Service Path Index.

5.5.2.Flow Match

The Experimenter Flow Match will be used to match the Path ID and Path Index in an SFCH in a packet. The Network Controller sends a Match SFCH to an SFF to match on the SFCH in incoming packets. There is a header structure following by a body structure:

```
/* Header for OXM experimenter match fields. */
struct ofp_oxm_experimenter_header {
    uint32_t oxm_header;   /* oxm_class = OFPXMC_EXPERIMENTER */
    uint32_t experimenter; /* Experimenter ID which takes the same
                                  form as in struct ofp_experimenter_header. */
};
OFP_ASSERT(sizeof(struct ofp_oxm_experimenter_header) == 8);
```

The following match operations are defined:

```
enum oxm_ofb_experimenter_sfch_match {
    OXM_OF_SFCH_PATH_ID = 0,    /* Match Path ID in SFCH */
```



```
    OXM_OF_SFCH_PATH_INDEX = 1, /* Match Path Index in SFCH */
};
```

The Experimenter Match SFCH Path ID body has the following structure and fields:

```
struct ofp_oxm_experimenter_sfch_path_id {
    uint32_t operation;     /* Operation; OXM_OF_SFCH_PATH_ID */
    uint32_t path_id;       /* Path ID */
};
```

The Experimenter Match SFCH Path Index body has the following structure and fields:

```
struct ofp_oxm_experimenter_sfch_path_index {
    uint32_t operation;     /* Operation; OXM_OF_SFCH_PATH_INDEX */
    uint8_t path_index;     /* Path Index */
};
```

It is expected that these matches will eventually become standard OpenFlow matches `OXM_OF_SFCH_PATH_ID` and `OXM_OF_SFCH_PATH_INDEX`.

5.5.3. Action

An Experimenter Action will be used to push the Service Chain Header onto a packet or pop the SFCH off a packet. The Network Controller sends a Push SFCH action to the Classifier or a Proxy Device to insert an SFCH into an outgoing packet. The Network Controller sends a Pop SFCH action to the Proxy Device or the terminating SFF to remove an SFCH from outgoing packets.

The Experimenter action header has the following structure and fields:

```
/* Action header for OFPAT_EXPERIMENTER.
* The rest of the body is experimenter-defined. */
struct ofp_action_experimenter_header {
    uint16_t type; /* OFPAT_EXPERIMENTER. */
    uint16_t len;  /* Length is a multiple of 8. */
    uint32_t experimenter; /* Experimenter ID which takes the same
                              form as in struct ofp_experimenter_header. */
};
OFP_ASSERT(sizeof(struct ofp_action_experimenter_header) == 8);
```

The following actions are defined:

```
enum ofp_experimenter_sfch_action {
    OFPAT_PUSH_SFCH = 0,         /* Push SFCH onto packet */
    OFPAT_POP_SFCH = 1,          /* Pop SFCH off packet */
    OFPAT_SET_SFP = 2,          /* Set Service Path ID */
    OFPAT_SET_SFI = 3,          /* Set Service Index */
    OFPAT_INC_SFI = 4,          /* Increment Service Index */
    OFPAT_SET_MD = 5,           /* Set Metadata */
};
```

The Experimenter `Push/Pop SFCH` action body has the following structure and fields:

```
struct ofp_action_experimenter_sfch {
    uint32_t operation;     /* Operation; OFPAT_PUSH_SFCH and OFPAT_POP_SFCH */
    uint32_t path_id;      /* Path ID */
    uint8_t path_index;    /* Path Index */
    uint8_t metadata[];    /* Metadata TLVs */
};
OFP_ASSERT(sizeof(struct ofp_action_experimenter_header) == 8);
```

Only the operation field is used in the `Pop SFCH` operation. It is expected that these actions will eventually become a standard OpenFlow action `OFPAT_PUSH_SFCH` and `OFPAT_POP_SFCH`.

5.6. Rules for OpenFlow Extensions

The Network Controller installs OpenFlow rules on the Classifier, SFF, terminating SFF, and Proxy devices to manage the service chains. The OpenFlow match and action enumerations used with each flow rule are listed.

5.6.1. On Classifiers

A Service Classifier is used to classify packet flows to different Service Chains. The Network Controller installs OpenFlow rules on the Classifier to match packet flows identified by N-tuple flow descriptors, adds the SFCH, any optional metadata TLVs and then forwards packets to the first SFF. The matching items may be different depending on the classification selected by the client.

```
Match:
     Input port (OXM_OF_IN_PORT): port A
     Source IP (OXM_OF_IPV4_SRC): source IP address
     Destination IP (OXM_OF_IPV4_DST): destination IP address
     Protocol (OXM_OF_IP_PROTO): TCP
     Source port (OXM_OF_TCP_SRC): source TCP port
     Destination port (OXM_OF_TCP_DST): destination TCP port

Action:
     Set Field (OFPAT_SET_FIELD/OXM_OF_TUNNEL_ID) = VXLAN VNI
     Push SFCH (OFPAT_PUSH_SFCH)
     Output port (OFPAT_OUTPUT): Port B
```

5.6.2. On Head-End Service Function Forwarders

Each SFF is responsible for forwarding the traffic to their designated local Service Functions and for forwarding the traffic to the next hop SFF after the local service function processing.

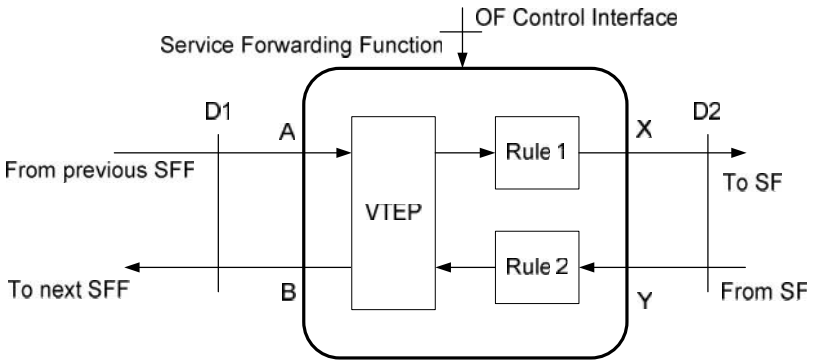

Figure 4: SFF interaction model with OF extensions

The Network Controller installs the following OpenFlow rules to an SFF to match the SFCH in incoming packets received from a previous SFF on interface D1 and then forward those packets to a specified output port on interface D2 to a Service Function.

Previous SFF to SF Flow Rule

This flow rule matches the packets with SFCH received from a previous SFF via an input OpenFlow logical port representing a VXLAN tunnel and forwards the packets to an output port to the Service Function.


```
Match:
     Input port (OFP_OF_IN_PORT) = tunnel port A
     Tunnel ID (OXM_OF_TUNNEL_ID) = VXLAN VNI
     Service Function Chain Header: Path ID (OXM_OF_SFCH_PATH_ID), Path Index
     (OXM_OF_SFCH_PATH_INDEX)

Action:
      Output port (OFPAT_OUTPUT) = SF port X
```

SF to Next SFF Flow Rule

This flow rule matches the packets with SFCH received from a locally connected SF via an OpenFlow input port and forwards the packets to an output logical port representing a VXLAN tunnel port to the next SFF.

```
Match:
     Input port (OXM_OF_IN_PORT) = port Y
     Service Function Chain Header: Path ID (OXM_OF_SFCH_CHAIN_ID), Path Index
     (OXM_OF_SFCH_PATH_INDEX)

Action:
     Set Field (OFPAT_SET_FIELD/OXM_OF_TUNNEL_ID) = VXLAN VNI
     Output port (OFPAT_OUTPUT) = Tunnel port B
```

5.6.3.On Terminating Service Function Forwarders

The terminating SFF removes the Service Chain Header before delivering the packet to its final destination using normal routing. The Network Controller installs the following OpenFlow rules on the Terminating SFF.

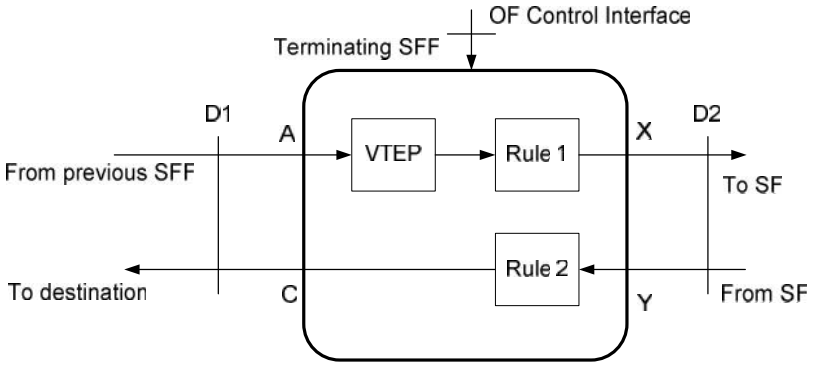

Figure 5: Terminating SFF interaction with OF extensions

Previous SFF to SF Flow Rule

This flow rule matches the packets with SFCH received from a previous SFF via an input OpenFlow logical port representing a VXLAN tunnel and forwards the packets to an output port to the Service Function.

```
Match:
     Input port (OXM_OF_IN_PORT) = tunnel port A
     Tunnel Id (OXM_OF_TUNNEL_ID) = VXLAN VNI
     Service Function Chain Header: Path ID (OXM_OF_SFCH_PATH_ID), Path Index
     (OXM_OF_SFCH_PATH_INDEX)

Action:
      Output port (OFPAT_OUTPUT) = SF port X
```


SF to Destination Flow Rule

This flow rule matches the packets with SFCH received from a locally connected SF via an OpenFlow input port, pops the SFCH and forwards the packets to an output logical port to the flow destination.

```
Match:
     Input port (OXM_OF_IN_PORT) = port X
     Service Function Chain Header: Path ID (OXM_OF_SFCH_PATH_ID), Path Index
     (OXM_OF_SFCH_PATH_INDEX)

Action:
     Pop SFCH (OFPAT_POP_SFCH)
     Output port (OFPAT_OUTPUT) = Port C
```

5.6.4.On Proxies

The Proxy Device is used to interface with a non-SFCH-aware Service Function appliance over the D3 interface. The Network Controller installs the following OpenFlow rules on the Proxy Device.

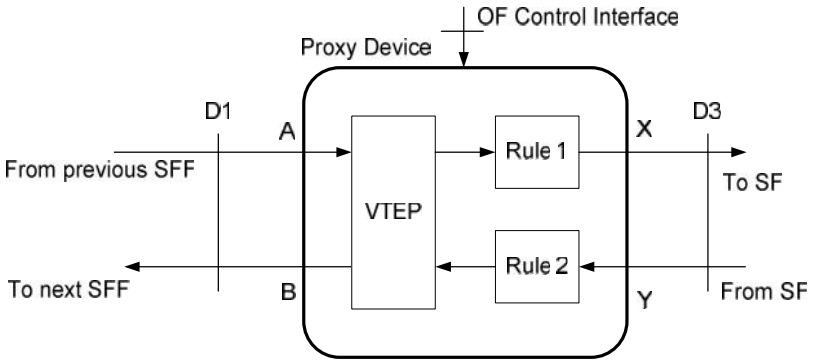

Figure 6: SFC Proxies with OF extensions

Previous SFF to SF Flow Rule

This flow rule matches the packets with SFCH received from a previous SFF via an input OpenFlow logical port representing a VXLAN tunnel, pops the SFCH and forwards the packets to an output port to the Service Function.

```
Match:
     Input port (OXM_OF_IN_PORT) = tunnel port A
     Tunnel ID (OXM_OF_TUNNEL_ID) = VXLAN VNI
     Service Function Chain Header: Path ID (OXM_OF_SFCH_PATH_ID), Path Index
     (OXM_OF_SFCH_PATH_INDEX)

Action:
     Pop SFCH (OFPAT_POP_SFCH)
     Output port (OFPAT_OUTPUT) = SF port X
```

SF to Next SFF Flow Rule

This flow rule matches the packets with SFCH received from a locally connected SF via an OpenFlow input port, pushes the SFCH and forwards the packets to an output logical port to the next SFF.

```
Match:
     Input port (OXM_OF_IN_PORT) = port Y

Action:
     Set Field (OFPAT_SET_FIELD/OXM_OF_TUNNEL_ID) = VXLAN VNI
```



```
Push SFCH (OFPAT_PUSH_SFCH)
Output port (OFPAT_OUTPUT) = Port C
```

5.7. L5-L7 Classification Considerations

Classifying the traffic at L5-L7 is a more complex operation than that at L1-L4. To perform L5-L7 classification, the classifier at a *minimum* needs to: (1) be able to decrypt the traffic, if necessary; and (2) have deep packet inspection (DPI) capabilities. These two capabilities are sufficient if the SFs in an SFC do not need to see the entire flow from start to finish. The initial traffic flow—e.g., the TCP handshake and perhaps a few subsequent packets—will traverse the default chain path since the classifier's DPI engine has not yet seen the L5-L7 value (e.g., the URI) by which it is supposed to classify the traffic. When the DPI engine eventually sees the L5-L7 value, it will classify the traffic by tagging it with an SFC header, which may force the traffic to take a separate SFC.

To be sure, this midway SFC switch method will not work if one or more SFs in the new SFC need to see the entire TCP flow—as do firewalls and (most common) WAN optimizers. Here, the classifier needs to know the L5-L7 value *before* the initial SYN can traverse down the right SFC. One way to solve this is to have the classifier perform TCP and application proxy functions. For example, to be able to classify packets containing HTTP traffic based on the URI, the classifier must be able to proxy HTTP (and TCP), and read the URI. This task is more difficult when the HTTP traffic is encrypted, e.g., with SSL/TLS. In this case, the classifier will need to be able to terminate SSL/TLS, which is a far more complex operational process.

In practice, L5-L7 classification may take place on the end hosts instead of on an in-path SFC classifier. The classifier on hosts will need to have access to non-encrypted traffic, have DPI capabilities, and be able to glean the L5-L7 value prior to the initial SYN leaving the host. It will be difficult for classifiers deployed inside service provider networks to have all three capabilities.

6. Sample Implementation: L2 Transparent Network Service Chaining with Traffic Steering using SDN

In "traditional" networks, inserting a network service involves physically altering the network topology. The physical appliance that performs the L2 network service must be inserted in the path of the traffic flows. The advent of virtualized network services (i.e. network services that are offered on a software form factor such as virtual machines, containers, etc.) enables the possibility that the services can be brought up dynamically. But the form factor of the services itself does not solve the problem of inserting the service without changing the physical network topology.

Enter service function chaining (SFC). It enables services regardless of their form factor—physical or software/virtual—to be inserted dynamically without having to change the underlying network topology. OpenFlow's ability to override traditional IP routing makes it suitable for traffic steering. Ideally, SF chaining will leverage an SFC header such as the SCH or NSH, which alleviates the complexity and load on the switches in the network. However, most network services today do not support an SFC header. Without such a header, service functions will have to evaluate header fields that are universally available today such as the classic five-tuple, VLAN IDs, etc. In this section we will discuss how service function chaining can be achieved for certain use cases today without the use of an SFC header. Transparent network services will be used to illustrate the points.

6.1. L2 Transparent Network Services

L2 transparent network services such as L2 security services allow the ability to inspect L2 Traffic transparently. In the physical world, L2 transparent network services appear as L2 bridges (tunnels) to the rest of the network. L2 transparent network services do their operations while bridging the packets. and they do not need to run any unicast or multicast routing protocols. Note that although L2 services can be deployed on physical appliances, for the purpose of this discussion, we will use the VM form factor to add the capability to insert the services more dynamically.

6.1.1. Virtual Server deployments

In virtual deployments, servers are run as virtual machines on compute nodes. A few important points to note are:

- Multiple servers could be instantiated on a physical server (compute node).
- Servers of various privileges or levels can be instantiated on the same compute node. For example, Web servers and database servers could be instantiated on a compute node.
- Servers can be live migrated from one compute node to another.

6.2. Using SDN/OF

Introducing network services in virtual environments should be simple, fast and less error prone. There should be no restrictions on server VM placement and server VM movement and no dependency on specific network virtualization technologies.

The solution leverages two technologies. One is the NFV technology where network services are also deployed as VMs and another is the SDN/OF technology which can be used to steer the inter server-VM traffic across multiple network service VMs.

6.2.1. SDN/OF virtual switches

Modern hypervisors introduced virtual switches that are controllable from outside. Linux has adopted the SDN/OF protocol-based virtual switch as part of its virtual environment. Each compute node has one virtual switch. All the traffic from/to virtual machines on the compute node passes through these virtual switches. Unlike older and proprietary virtual switches, SDN/OF based virtual switches are programmable from an external controller. Since all the traffic across VMs within a compute node or across compute nodes passes through programmable virtual switches, traffic steering application on top of a central controller becomes a possibility.

OVS (Open Virtual Switch) is a virtual switch implementation in recent Linux distributions. OVS supports the OF 1.3 protocol, which allows external controllers to create a pipeline of packet processing using multiple tables. Each table can be programmed with multiple entries. These tables support any combination of match fields. OVS also supports many instructions/actions which enable multiple virtual network technologies including VXLAN, NVGRE and of course VLAN.

6.2.2. Role of SDN/OF in Service Function Chaining Traffic Steering

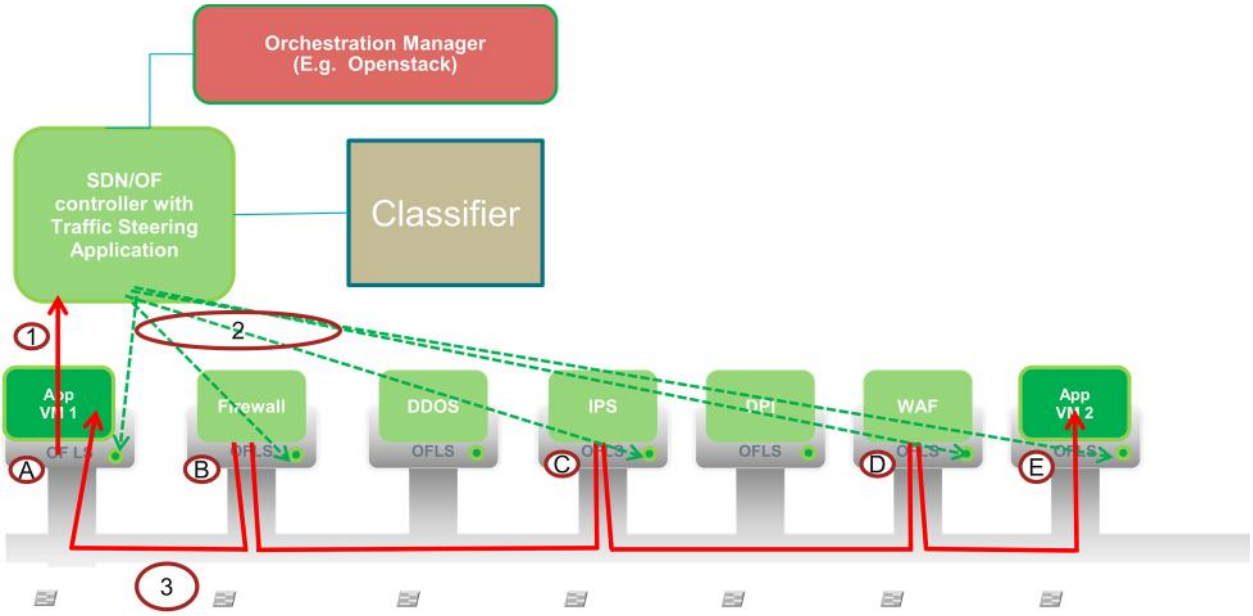

Figure 7: Service Function Chaining Components Overview[1]

The picture only shows the traffic flow, but the following configuration steps are expected to be done by the administrator using an orchestration tool such as OpenStack. Configuration steps typically involve:

[1] Diagram courtesy of Freescale

- Uploading network service VM images into orchestration tool. In this example, it is expected that ”firewall”, “DDOS prevention”, "IPS', "DPI and "WAF" VM images are uploaded in the orchestration tool.
- Creation of network service chains.
- Adding network services to the chain. In this example, "firewall", "DDOS prevention", "IPS", "DPI" and "WAF" services are added to the chain.
- Creating bypass rules. In this example, web traffic is expected to be passed through firewall, IPS and WAF services. And all other traffic is expected to be passed through Firewall, DDoS, IPS and DPI services. Hence, two bypass rules would need to be created:
  - Rule 1: Web traffic, bypass "DDOS", "DPI".
  - Rule 2: All traffic, bypass "WAF".
- Attaching the network service chain to a virtual network.

Once above steps are done, the orchestrator is expected to instantiate network service VMs (one for each service) upon the first application VM in that virtual network is brought up.

Orchestrator programs the SDN/OF controller with the above configuration.

Packet flow: For simplicity, the diagram above shows each VM on a separate compute node. In reality, multiple VMs can be instantiated on a compute node. Each compute node has an OFLS (OpenFlow Logical switch) virtual switch. All virtual switches of compute nodes are configured to communicate with a centralized SDN/OF controller. The SDN/OF controller runs the traffic steering application.

- Step1: APPVM on the left side initiates an HTTP connection to the APPVM on the right side. When the virtual switch on local compute node sees the first packet, it does not find any flow in its tables. Hence, it sends that packet to the SDN/OF controller via OF protocol as an exception packet.
- Step 2: Traffic steering application in SDN/OF controller, refers to the service chain configuration associated with the virtual network. It also finds out the services to be applied on the traffic of this connection. Since the rule (Rule 1) indicates that DDOS and DPI services need to be bypassed for HTTP traffic, it chooses to create flows in the virtual switches where application VMs, "firewall", "IPS" and 'WAF" VMs are present. In this example, virtual switches A, E, B, C, D are populated with the flows by the controller and it is functioning also as a classifier. These flows are meant to redirect the traffic from one SF to another. It then sends the packet back to the virtual switch.
- Step 3: Once the flows are created, all packets of the connection between these two VMs are passed through the network service VMs.

Details of flows that are created as part of Step 2 by the SDN/OF controller:

Assumptions:

- AppVM1 IP address = I1 and AppVM2 IP address = I2.
- Compute node IP addresses: A's address = O1, E's address = O2 B's address = O3, C's IP address = O4 and D's IP address = O5.
- HTTP Connection: TCP Source Port = 2000 and Destination Port = 80

  - Client to Server 5-tuple: I1, 2000->I2, 80 with TCP.
  - Server to Client 5 tuple I2,80->I1, 2000 with TCP.
- All VMs in each compute node are connected via port VM-p to the respective logical switches.
- Each of the network service instances across compute nodes are assigned with a distinct pair of VLAN IDs within a virtual network uniquely. The network service hosted by Compute Node B is assigned with the pair VLAN100/VLAN200 , that of Compute Node C with VLAN300/VLAN400 and that of compute node D with VLAN500/600.
- VXLAN based virtual network is used. VNI = 50
- Flow based tunnelling mechanism is used.
- Network ports in all virtual switches are denoted as N-p

OF flows:

Traffic flows in virtual switch A:

- Flow 1 (Client to Server flow)
  - Match fields:
    - Input Port = VM-p
    - Source IP = I1, Destination IP = I2, Protocol = TCP, Source port = 2000 and Destination Port = 80
  - Instructions/Actions: Next hop if virtual switch B and firewall leg VLAN 100.
    - Set fields: Tunnel_ID = 50 (VNI), Tunnel Destination IP = O3
    - Push VLAN - VLAN100.
    - Output: N-p
- Flow 2: (Server to Client flow)
  - Match fields:
    - Input port = N-p
    - Source IP = I2, Destination IP = I1, Protocol = TCP, Source Port = 80 and Destination port = 2000
  - Instructions/actions:
    - Output: VM-p

Traffic flows in virtual switch B: Here four flows are required - one for receiving client-to-server traffic from the network to give it to the firewall, second flow is for receiving the traffic from the firewall VM to send it onto the network towards next service, third flow is for receiving server-to-client traffic from the network to give it to the firewall and the fourth flow is for receiving the packet from firewall to send it onto the next service/application VM.

- Flow1 (Client to Server traffic from network)
  - Match fields:
    - Input Port = N-p
    - VLAN = VLAN100

        - Source IP = I1, Destination IP = I2, Protocol = TCP, Source port = 2000 and Destination Port = 80
    - Instructions/Actions:
        - Output = VM-p
- Flow 2 (C-S traffic from firewall VM)
    - Match fields:
        - Input port = VM-p
        - VLAN = VLAN200
        - Source IP = I1, Destination IP = I2, Protocol = TCP, Source port = 2000 and Destination Port = 80
    - Instructions/Actions: Next hop if virtual switch C and IPS leg VLAN 300.
        - Set fields: Tunnel_ID = 50 (VNI), Tunnel Destination IP = O4
        - PopVLAN - VLAN200
        - Push VLAN - VLAN300
        - Output: N-p
- Flow 3: (S-C traffic from network)
    - Match fields:
        - Input Port = N-p
        - VLAN = VLAN200
        - Source IP = I2, Destination IP = I1, Protocol = TCP, Source Port = 80 and Destination port = 2000
    - Instructions/actions:
        - Output = VM-p
- Flow 4 (S-C traffic from firewall VM)
    - Match fields:
        - Input port = VM-p
        - VLAN = VLAN100
        - Source IP = I2, Destination IP = I1, Protocol = TCP, Source Port = 80 and Destination port = 2000
    - Instructions/Actions:
        - Set fields: Tunnel_ID = 50 (VNI), Tunnel Destination IP = O1
        - PopVLAN - VLAN100
        - Output: N-p

Traffic flows on virtual switch C and D look similar to the flows on virtual switch B.

Traffic flows in virtual switch E:

- Flow 1 (Client to Server flow)

- Match fields:
  - Input Port = N-p
  - Source IP = I1, Destination IP = I2, Protocol = TCP, Source port = 2000 and Destination Port = 80
- Instructions/Actions: Next hop if virtual switch B and firewall leg VLAN 100.
  - Output: VM-p

- Flow 2: (Server to Client flow)
  - Match fields:
    - Input port = VM-p
    - Source IP = I2, Destination IP = I1, Protocol = TCP, Source Port = 80 and Destination port = 2000
  - Instructions/actions:
    - Set fields: Tunnel_ID = 50 (VNI), Tunnel Destination IP = O5
    - Push VLAN – VLAN600.
    - Output: N-p

6.2.3. OF-1.3 extensions used

The following Nicira (OVS)-defined set field actions, which are not supported by OF 1.3, are used.

A. Tunnel ID – represented by VNI in case of VXLAN

B. Tunnel destination or remote IP Address

C. Register (`NXM_NX_REG0, NXM_NX_REG1, NXM_NX_REG2`) for `ApplicationId`, `ProtocolPath` and `SuscriberClass`

6.3. Scale-out

The SDN/OF controller can also acts as an implicit load balancer, thereby avoiding any external load balancers to increase the scale of network services. In the following picture, there are three IPS service VMs instantiated and two WAF VMs instantiated. For every new connection, the SDN/OF controller chooses the IPS VM and WAF VM that is least loaded. Accordingly, it programs the flows in appropriate OF logical switches.

Note that doing even L4 Load balancing without an SFC header will be expensive for the SFFs as they may need to hold an extensive five-tuple based forwarding tables.

In the picture shown below, red connection traffic is passed through one IPS VM and blue connection traffic is passed through another IPS VM.

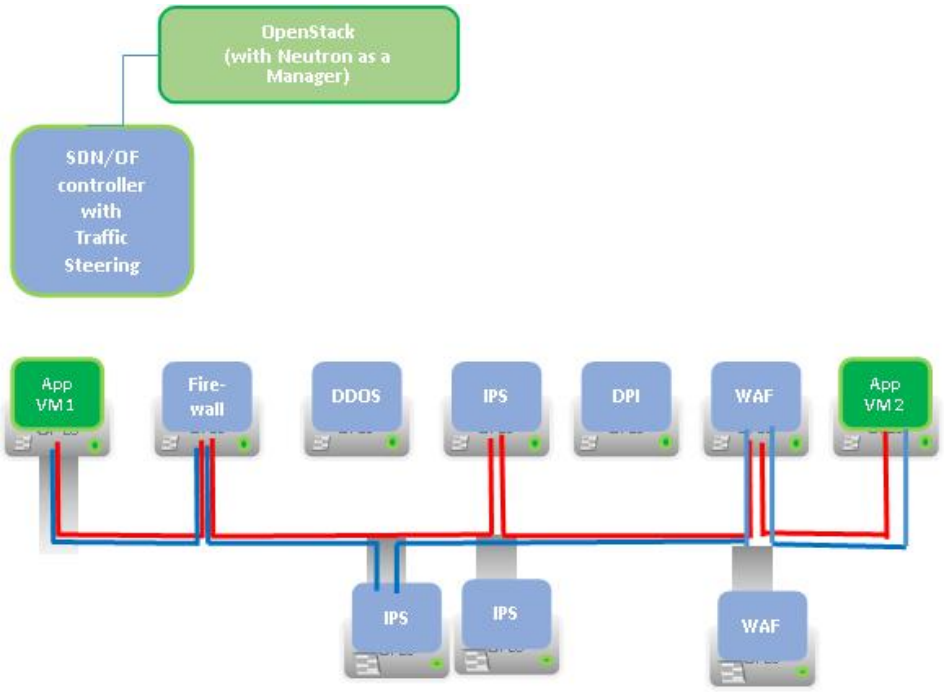

Figure 8: Sample Traffic Flows

6.4. Requirements to Management and Orchestration

NFV Management and Orchestration requires an SDN/OF controller within VIM. The SDN/OF controller may need to talk to the virtual switches of compute nodes to program flows. Hence there is a need for an interface point between the SDN/OF controller (VIM) and compute nodes (NFVI). Or-Vi and Nf-Vi would serve this purpose. The SDN/OF controller also needs to be informed of the network service chain configuration. Hence there is a need for an interface point between the orchestration system (VNFM) and the SDN/OF controller system (VIM). Vi-Vnfm and Ve-Vnfm would serve this purpose.

NFV management and orchestration needs to be able to satisfy the following requirements:

- Service Chain Management
  - Ability to manage (Create/Delete/Update) network service chains.
  - Ability to manage (Add/Delete/Update) network functions in the chain - Service Insertion/Deletion [MANO service graph update workflows]
  - Mitigate service interruption to existing traffic when VNF instance fails in the chain.
  - Ability to define traffic rules to bypass network functions in the chain.
- Management of network service chain with virtual networks.
  - Ability to associate multiple network service chains to a network.
  - Ability to define traffic rules for selecting the chain out of multiple chains.
- Scale-Out:
  - Ability to bring up/down multiple network function VMs on demand basis

  - Ability to balance the traffic across multiple VMs of a network function with no special load balancers.

- Fast Path
  - Ability for network service VMs to offload connections to some forwarding entity.

One orchestrations system to deploy both server and network service VMs.

6.4.1.Gap Analysis with ONF

OF 1.3 support multiple tables. The traffic steering application can use a few of the tables to install traffic redirection flow entries. A few extensions in OF 1.3 are required to support a flow based tunneling mechanism. Though those extensions are not part of the OF 1.3 specifications, they are widely implemented in OVS. These extensions are required to be standardized.

7. Acknowledgments

We would like to acknowledge Nicolas Bouthors, Marie-Paule Odini and Diego Lopez Garcia for their thorough review and feedback.

8. References

[SFC-ARCH] Halpern, J. (ed.) and C. Pignataro (ed.), Service Function Chaining (SFC) Architecture, https://tools.ietf.org/html/draft-ietf-sfc-architecture-05, February 2015.

[SFC-DESIGN] Boucadair, M., et al, "Service Function Chaining: Design Considerations, Analysis & Recommendations", https://tools.ietf.org/html/draft-boucadair-sfc-design-analysis-02, February 2014.

[SFC-L4-L7] Dunbar, L., et al, "Architecture for Chaining Legacy Layer 4-7 Service Functions", https://datatracker.ietf.org/doc/draft-dunbar-sfc-legacy-l4-l7-chain-architecture/, July 2014.

[SFC-META] Rijsman, B. and J. Moisand, "Metadata Considerations", https://datatracker.ietf.org/doc/draft-rijsman-sfc-metadata-considerations/, February 2014.

[SFC-PS] Quinn, P., Ed. and T. Nadeau, Ed., "Service Function Chain Problem Statement", https://datatracker.ietf.org/doc/draft-ietf-sfc-problem-statement/, August 2014.

[SFC-SCH] Zhang, C., et al, "Service Chain Header", https://datatracker.ietf.org/doc/draft-zhang-sfc-sch/, December 2014.

[SFC-NSH] Quinn, P., et al, "Network Service Header", http://www.ietf.org/archive/id/draft-quinn-sfc-nsh-07.txt, February 2015.

9. List of Authors

- Cathy Zhang: cathy.h.zhang@huawei.com
- Srini Addepalli: saddepalli@freescale.com
- NS Murthy: nsmurthy@freescale.com
- Louis Fourie: louis.fourie@huawei.com
- Myo Zarny: myo.zarny@gs.com
- Linda Dunbar: linda.dunbar@huawei.com