Authors contact info:

Paul Quinn
paulq@cisco.com
Distinguished Engineer
Cisco Systems
55 Cambridge Parkway
Cambridge, MA 02141

Tel: 408 527 3560

Jim Guichard
jguichar@cisco.com
Principal Engineer
Cisco Systems
170 W. Tasman Drive
San Jose, CA 02481

Tel: 703 484 4555

Author Bios:

Paul Quinn:

Paul Quinn is a Cisco Distinguished Engineer who during his tenure at Cisco has worked on a wide-range of technologies including service provider and platform security, data center services and video infrastructure.

Paul's current focus is on service chaining, more specifically the development of protocols, control, policy and data plane, used for deploying network services. Paul is active in the IETF Service Function Chaining Working Group.

Paul has also been a frequent presenter at customer and industry events.

Jim Guichard:

Jim is a Principal Network Architect at Cisco Systems with 20+ years of industry experience and expertise in areas such as Multiprotocol Label Switching (MPLS), IP routing, L3VPN's, and Software Defined Networking (SDN). In his current role within Cisco's central software organization he is responsible for defining new networking protocols and steering these through industry standardization bodies as well as internal engineering development.

Jim is the inventor of 50+ US & international patents relating to IP, MPLS, and network services, and is widely respected for his MPLS/IP expertise. This knowledge has resulted in his co-authorship of several books including "MPLS & VPN Architectures, volumes I & II" and "Definitive MPLS Network Designs".

Abstract:

The term “service chaining” has emerged to describe the deployment of composite services that are constructed from one or more L4-L7 services using Software Defined Networking (SDN) and Network Function Virtualization (NFV) paradigms; these technologies are helping to move service deployment into the realm of modern networks.

In this paper, provide an overview of the service chaining problem space, and delve into the details of a newly proposed standard for implementing an SDN-centric service chaining dataplane.

Keywords:

Service chaining, service function chaining, SDN, software defined networking/networks, network function virtualization, NfV, network service header, NSH, network overlay, service overlay, IETF SFC, service plane

Service Function Chaining

Creating a Service Plane Using Network Service Header (NSH)

Authors contact info:

Paul Quinn
paulq@cisco.com
Distinguished Engineer
Cisco Systems
170 W. Tasman Drive
San Jose, CA 02481

Tel: 408 527 3560

Jim Guichard
jguichar@cisco.com
Principal Engineer
Cisco Systems
170 W. Tasman Drive
San Jose, CA 02481

Tel: 703 484 4555

Abstract

Modern networks require a more agile method for network service deployment and delivery. Service chaining is a broad term that describes a new paradigm for service deployment and is predicated on several key concepts including topological independence for services and elastic scaling of service elements. This paper provides an overview of service chaining and specific details about a proposed standard data plane format used to create a service plane for network service chaining.

Service Chaining Primer

Networks, servers, storage, and applications, have all undergone significant change in recent years with the widespread adoption of virtualization technologies, network overlays, and orchestration. However, the deployment of critical services (often referred to as L4-L7 services) into a network has remained largely unchanged, and continues to be a complex and risk fraught technical and organizational challenge. This directly impacts the speed at which critical applications can be deployed and significantly increases operational costs for network operators.

The term "service chaining" has emerged to describe the deployment of composite services that are constructed from one or more L4-L7 services using Software Defined Networking (SDN) and Network Function Virtualization (NFV); these technologies are helping to move service deployment into the realm of modern networks.

A service chain is fundamentally a policy construct: a series of service functions that a packet must traverse. For instance, a service chain may define that all TCP port 80 traffic must pass through a firewall (FW), then intrusion prevention (IPS), and finally server load balancing (SLB). In order to realize this requirement in a network, a service path, which is an instantiation of the service chain, must be created. The service path is formed using an overlay topology between the needed service functions. Using the same example as above, an instantiated service path might be FW2 → IPS12 → SLB5. There is typically a 1:n relationship between service chains and service paths.

Given the widespread applicability of service chaining, the Internet Engineering Task Force (IETF) has formed a new working group, entitled Service Function Chaining (SFC)[1]. This paper uses the IETF adopted term, service function, to describe a discrete element that provides specific service functionality. Examples of service functions include, but are not limited to, firewalls, load balancers, deep packet inspectors, etc.

Service Insertion Today

Existing service insertion models suffer from a number of limitations. Service functions that must be applied to traffic for a given service are physically inserted on the data-forwarding path between communicating peers, and traffic is directed through them using VLANs and policy-based routing techniques. Consequently, services are tightly coupled to the physical network topology creating constraints on service delivery and potentially inhibiting the network operator from optimally utilizing their service resources. New application deployment or the addition of new services into the network is constrained and this topological coupling limits scale, capacity, and redundancy. If the necessary service functions are not available to support a new application, then the network operator has no other option but to deploy additional hardware resources and re-configure the network to accommodate the new service requirements.

Service functions are not easily moved, created or removed even when virtualized service functions are deployed. This rigidity is the antithesis of highly elastic environments that demand rapid creation, destruction or movement of the service functions required for application delivery. Additionally, the transition to virtual platforms requires an agile service insertion model that supports elastic and very granular service insertion, and post-facto modification as well the movement of service functions and application workloads in the existing network, all the while retaining the network and service policies and the ability to easily bind service policy to granular network-centric identifiers such as subscriber information.

These factors provide motivation for a simplified and flexible service insertion model that addresses many of the current shortcomings and provides new, much needed functionality to enable service deployments in modern network environments. Service chaining accomplishes this by changing the notion of a service function. Instead of fixed, static elements that require network and routing changes, service functions are treated as resources, with associated attributes, available for scheduled consumption. Selective traffic, subject to policy, may then be "steered" to the requisite service resources, along with any "extra" information – often referred to as metadata – that may be valuable for policy enforcement.

Service “Plane”

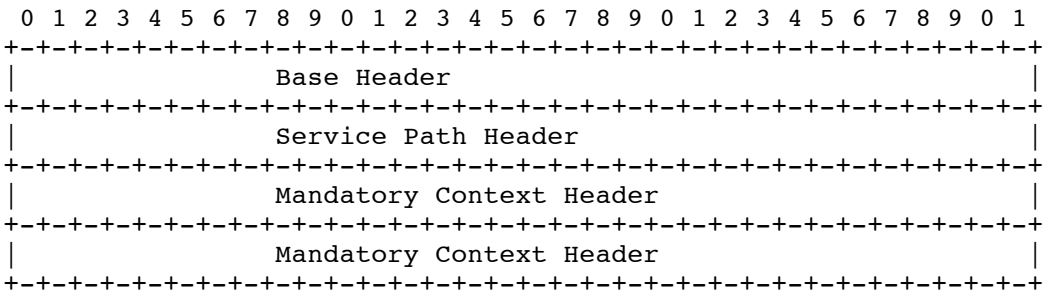
A basic form of service chaining may be realized using existing transport encapsulations. This method of chaining relies upon the tunneling of selected data between service functions. Using this model, packets are selected, based on some classification criteria, and then encapsulated in a network transport header (such as GRE [2] or VXLAN[3], or VXLAN-GPE[4]) and delivered to the initial service function in a service chain. Post service function execution, packets must be re-classified and if further service functions need to be applied, the packets are encapsulated again in a tunnel for transport to the next service function. This process is repeated until all required service functions have been applied.

Although this form of service chaining achieves some level of abstraction from the underlying topology, it does not truly create a service plane: a distinct identifiable plane that can be used across all transports to create a service chain, and exchange metadata along the chain. From an architectural standpoint a number of significant limitations remain for complex deployments as follows:

- Configuration complexity due to the per-hop re-classification policy and tunnel selection configuration.
- Limited service chain construction capabilities; rigid service ordering that prevents the formation of true service graphs.
- No ability to pass additional metadata information.
- Limited chain visibility and troubleshooting: largely a hop-by-hop debugging exercise.

Network Service Header (NSH)

NSH[5] is a data plane header format used to create a dedicated service plane that is both independent of the underlying transport but is also able to take advantage of the service landscape transitions highlighted previously. NSH, illustrated in figure 1, is comprised of a base header, a service path header, 16-bytes of mandatory fixed context information and optional variable length context data:



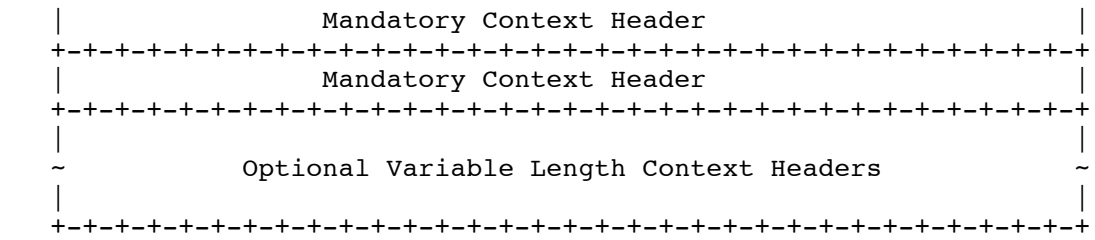


Figure 1: Network Service Header

NSH is imposed between the original packet or frame, and an outer network transport encapsulation such as MPLS, VXLAN/VXLAN-GPE, GRE or IP in IP. NSH is transport agnostic, and can be carried by many widely deployed transport protocols.

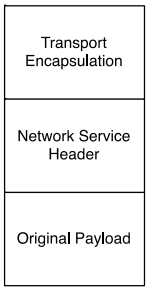


Figure 2: Protocol Stack

NSH Base Header

As shown in figure 3, the 4-byte base header is used to provide information about NSH itself, as well as the encapsulated packet. More specifically, the base header contains the following fields:

- Version number: provides data plane versioning.
- Flags bits: used to define header changes or parsing behavior within a version. To date, two flags are defined: i) O-bit: when set, the packet is an operation, administration and management (OAM) packet, ii) C-bit: indicates the presence of a critical metadata TLV.
- Length: total length of NSH, including optional variable TLVs, in 4 byte words. The length must be equal to or greater than 6.
- MD Type: NSH defines type 0x1 as NSH.
- Next Protocol: indicates the protocol type of the original packet. Currently, three types are defined: 0x1 for IPv4, 0x2 for IPv6 and 0x3 for Ethernet.

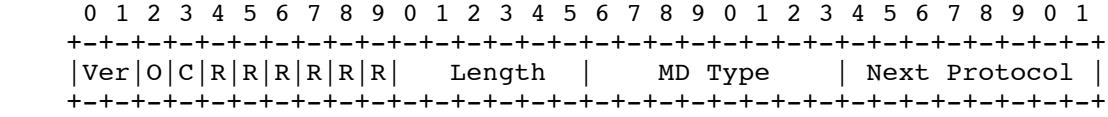


Figure 3: NSH Base Header

NSH Service Path Header

A 4-byte service path header follows the base header and defines two fields used to construct a service path: service path identifier, and service index. Figure 4 depicts the service path header.

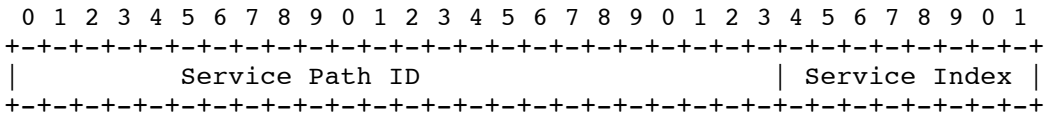


Figure 4: NSH Path Header

The service path identifier (SPI) is used to identify the service path that interconnects the needed service functions. It provides a level of abstraction allowing nodes to utilize the identifier to select the appropriate network transport protocol and forwarding techniques. The service index (SI) identifies the location of a packet within a service path. As packets traverse a service path, the SI is decremented post-service.

Additionally, the combination of SPI and SI provides operators with clear visibility into where packets are in relation to the service functions on a service path.

Service Path Index and Re-Classification

Given that the SPI represents the service path, altering the path identifier results in a change of a service path. An SPI change is a result of classification (strictly speaking, re-classification since initial classification occurred to select packets that have been forwarded along a service path): that is a node in the service path determined – based on policy – that the initial classification was incorrect, or incomplete.

If the updated classification results in the necessity of a new service path, the node updates the SPI and SI fields accordingly, and the new identifier is then used to select the appropriate overlay topology.

This level of indirection (SPI → overlay) allows service functions to alter the path of a packet without having to participate in the network topology and its associated control plane(s).

NSH Context Headers

NSH defines 16 bytes of mandatory opaque context headers. The context headers are used to pass information – metadata – between classification nodes and service functions, between service functions, and between service functions and network nodes. The exchange of classification information in NSH provides several keys benefits:

1. Implementation simplicity
2. Flexible service function policy
3. Compound context information

Generally speaking, metadata reflects the result of an antecedent, or external (to the system) classification. This classification information is then provided via the NSH context fields to service chain participant for local policy decisions.

A control plane defines the semantics of the fixed context headers.

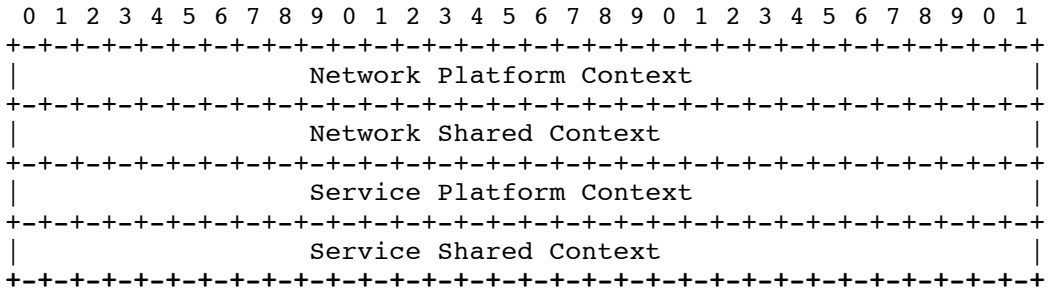


Figure 5: NSH Mandatory Context Headers

In addition to the mandatory 16 bytes of context, NSH defines optional variable length, typed context space as per figure 6.

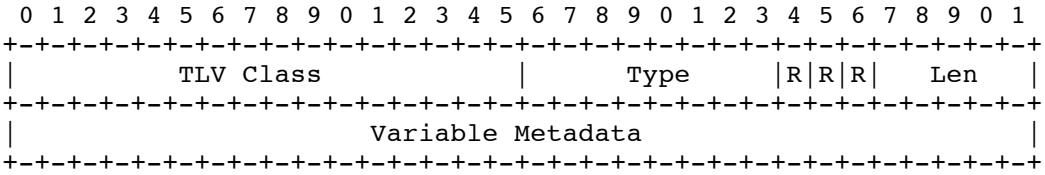


Figure 6: NSH Path Header

These optional variable length headers contain a class, which defines the scope of the metadata. For example, this class might indicate a vendor; otherwise it may define how to interpret the typed data.

NSH and Control Plane(s)

A control plane is required in order to exchange NSH values with participating nodes, and to provision the same nodes with requisite information such as service path ID to overlay mapping.

NSH is applicable to many different environments, and as such does not mandate a single control protocol. Rather, widely used control planes must add support for service chaining via NSH. More specifically, the IETF SFC explicitly defines a working group action: "A document will be developed to describe requirements for conveying information between control or management elements and SFC implementation points"[6].

Commonly used protocols – centralized and distributed – support or will evolve to support NSH-based service chaining, and will vary based on deployment and operator requirements and several models will be viable.

The open-source OpenDaylight (ODL) Service Function Chaining project[7] provides an SDN approach to service chaining via a controller. Furthermore, the controller platform is agnostic with respect to southbound provisioning protocols and uses the appropriate protocols for a given device / service chain combination. These southbound protocols include Openflow with Open vSwitch [8]and Netconf/YANG[9].

Another approach is described in detail in Service Function Chaining Use Cases in Mobile Networks[10]. In the mobile environment, Policy and Charging Rules Function (PCRF) is one of the network infrastructure elements used to control the service chains.

Architectural Benefits

As described above, network service are currently tightly linked to topology, further limitations of existing deployments are discussed in details in the IETF SFC Problem Statement draft[11].

NSH provides a substrate needed to address most, if not all, of the limitations identified by providing a vastly simplified and unified dataplane.

First and foremost, NSH provides the de-coupling of the service topology and the actual network topology. The concept of a service function morphs: rather than being viewed as "bump in the wire", a service function becomes an identifiable resources available for consumption, with only the appropriate traffic (based on classification) being sent to that resource.

The service path ID and index combination provides the needed indirection and represents service-plane identifiers. The network transport layer is selected based on the combination off service path ID and index. Rather than creating network segments such as VLANs to "insert" a service, traffic, based on NSH information, is redirected to the required services.

Topological independence further enables complex service creation since the service topology defined by NSH is dynamic and created as needed without requiring underlying changes. Similarly, per service function elasticity is viable since a service function is simply identified as a hop within the NSH service topology.

Although the creation of a dedicate service topology is the cornerstone of service chaining with NSH, the value of NSH context carriage cannot be overstated.

NSH context information provides metadata used for service-specific policy creation. The NSH metadata reflect the result of classification – both derived directly from the packet/payload as well as from external sources. This metadata is then used, per service function for policy enforcement.

The Service Function Chaining (SFC) Architecture document [12] provides a comprehensive overview of service chaining, the required components and their interaction.

NSH In Practice

Path Selection and Forwarding

Figure 7 below depicts NSH control/policy plane setup and subsequent use of the service path for forwarding packets through the necessary services of an associated service chain.

Figure 7 shows a number of steps that are used for service path creation and packet forwarding through an associated service chain:

1. An SDN controller platform is responsible for configuration of the classification policies associated with a given service chain; each network element involved in the service path is provisioned and this may be achieved via any number of mechanisms such as OpenFlow, PCRF, NETCONF/YANG, or direct device programming.

2. NSH is imposed on packets that match the classification policy; in this example, a NSH with service path identifier (10), and service index (2) is added to the packets at the ingress service classifier.

3. Based on the SPI/SI combination, a lookup is performed and the appropriate transport encapsulation is imposed and used to deliver the packet. In this case VXLAN is used, and the traffic is sent to the (physical or virtual) Top of Rack (TOR) switch that provides connectivity for the initial service function to be applied.

   In addition to the service path identifier and path index, the classification rules also provide information about the packet. In this example, the application is identified, as is the user. Both classification results are encoded and inserted into the NSH context headers.

4. TOR^1 removes the transport encapsulation and consults the NSH. Using the SPI/SI, the packet and the NSH is then delivered to the initial service function (SF_1). SF_1 may opt to use the context headers for policy enforcement, and must update the NSH SI field to reflect that the service function is complete.

5. Post-SF_1, packets and NSH are returned to TOR^1, which again uses the NSH SPI/SI to determine the next service function to apply, in this case SF_2. The encapsulated packet is then forwarded to TOR^2.

6. TOR^2 uses the SPI/SI to determine that the packet should be delivered to SF_2.

7. When the packet exits the last service function of the chain it is forwarded toward its initial destination.

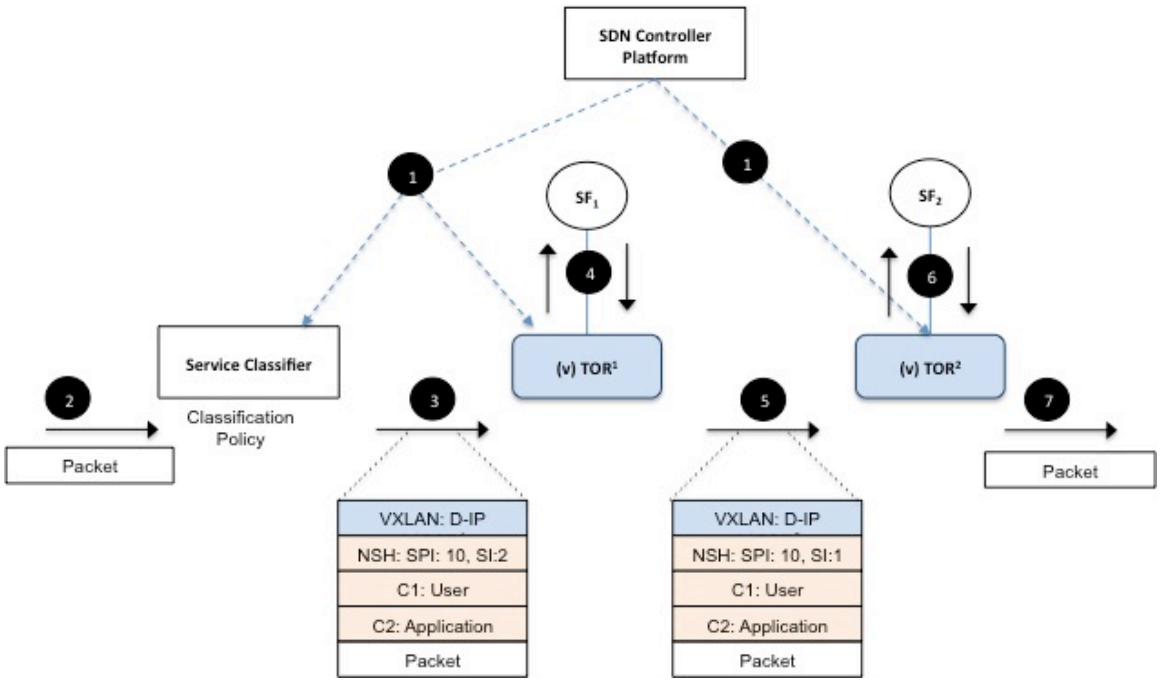


Figure 7: NSH Service Path Forwarding

Metadata and Service Function Policy

The metadata carried in NSH is primarily used for policy enforcement at service functions. In figure 7 above, encoded classification results are delivered to the SFs via NSH, and can then be used locally. For example, SF_1 might take some action (permit/deny) based on application information.

Figure 8 depicts a more detailed example of metadata and policy.

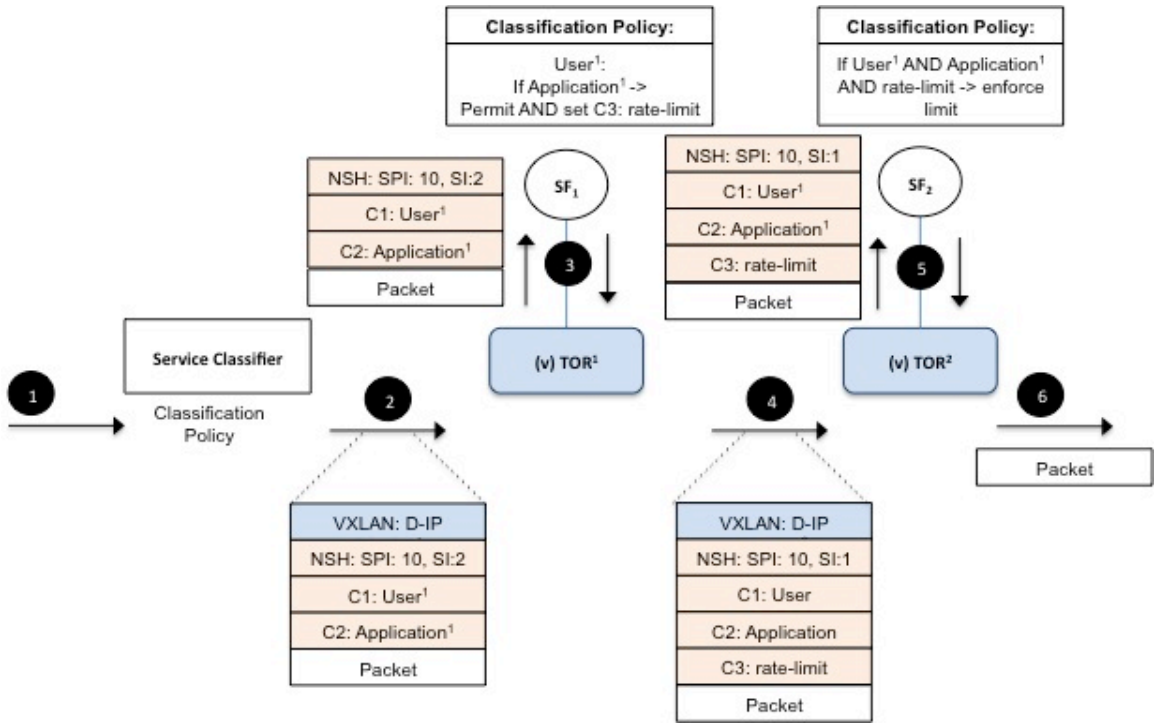


Figure 8: NSH Metadata & Policy

Figure 8 illustrates a number of steps that show service functions use of NSH metadata to apply complex policy:

1. Initial classification of traffic occurs at the service classifier and is encapsulated with NSH.
2. Based on the SPI/SI combination, a lookup is performed and the appropriate transport encapsulation is imposed and used to deliver the packet to the first service function of the service chain; SF_1.
3. When SF_1 receives the NSH context information, the service function applies local policy that requires per-user, per-application policy enforcement. SF_1 reclassifies the traffic and adds context information based on local classification (when the packet arrived at SF_1 there was no indication that User[1] + Application[1] needed to be rate-limited. However, SF_1's policy dictates that such packets need to be rate-limited). SF_1 does not enforce rate limiting but rather updates the NSH context with its classification result.
4. Post-SF_1, packets and NSH are returned to TOR[1], which again uses the NSH SPI/SI to determine the next service function to apply, in this case SF_2. The encapsulated packet is then forwarded to TOR[2].
5. SF_2 applies local policy that determines the traffic should be rate-limited based on the NSH context header imposed at SF_1.
6. When the packet exits the last service function of the chain it is forwarded toward its initial destination.

Conclusion

Agile, elastic insertion of a service is a core component of any modern network, and a comprehensive service chaining architecture is needed to address application requirements. As service chaining matures, interoperability is a must. Network Service Headers, a data plane protocol submitted to the IETF, provide the required data plane information needed to construct topologically independent service paths, and pass opaque metadata between classification and service functions.

References

[1] IETF Service Function Chaining Working Group: http://datatracker.ietf.org/wg/sfc/

[2] D. Farinacci, T. Li, S. Hanks, D. Meyer and P. Traina (2000, March). *Generic Routing Encapsulation (GRE)*, [online]. Available: http://tools.ietf.org/html/rfc2784.html

[3] D. Dutt, M. Mahalingam, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. (2014, April 10). *VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*". Available: https://datatracker.ietf.org/doc/draft-mahalingam-dutt-dcops-vxlan/

[4] P. Agarwal, et al. (2013, December 16). *Generic Protocol Extension for VXLAN*. Available: https://datatracker.ietf.org/doc/draft-quinn-vxlan-gpe/

[5] P. Quinn, et al. (2014, Feb 14). *Network Service Header (NSH).* Available: http://datatracker.ietf.org/doc/draft-quinn-sfc-nsh/

[6] Service Function Chaining Working Group. *Charter for Working Group.* Available*:* http://datatracker.ietf.org/wg/sfc/charter/

[7] OpenDaylight Service Function Chaining Project. (May 2014). Available: https://wiki.opendaylight.org/view/Service_Function_Chaining:Main

[8] V. Ermagan, L. Jakab, P. Kothari and K. Mestery. (2013). *LISP and NSH in Linux and Open vSwitch*. Available: http://www.slideshare.net/mestery/lisp-and-nsh-in-open-vswitch

[9] Penno, R., Quinn, P. (2014, May 29). *Yang Data Model for Service Function Chaining*. Available: http://datatracker.ietf.org/doc/draft-penno-sfc-yang/

[10] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, J. Uttaro. (2014, May 5). *Service Function Chaining Use Cases in Mobile Networks*. Available: http://datatracker.ietf.org/doc/draft-ietf-sfc-use-case-mobility/

[11] Quinn, P., Nadeau, T. (2014, Augurst 8). *Service Function Chaining Problem Statement.* Available: http://datatracker.ietf.org/doc/draft-ietf-sfc-problem-statement/

[12] P. Quinn, P. and J. Halpern, Eds (2014, May 5). *Service Function Chaining (SFC) Architecture*. Available: http://datatracker.ietf.org/doc/draft-quinn-sfc-arch/