

Service Function Chaining  
Internet-Draft  
Intended status: Standards Track  
Expires: April 23, 2017

S. Kumar  
J. Guichard  
P. Quinn  
Cisco Systems, Inc.  
J. Halpern  
Ericsson  
S. Majee  
F5 Networks  
October 20, 2016

Service Function Simple Offloads  
draft-kumar-sfc-offloads-03

## Abstract

Service Function Chaining (SFC) enables services to be delivered by selective traffic steering through an ordered set of service functions. Once classified into an SFC, the traffic for a given flow is steered through all the service functions of the SFC for the life of the traffic flow even though this is often not necessary. Steering traffic to service functions only while required and not otherwise, leads to shorter SFC forwarding paths with improved latencies, reduced resource consumption and better user experience.

This document describes the rationale, techniques and necessary protocol extensions to achieve such optimization, with focus on one such technique termed "simple offloads".

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 23, 2017.

Internet-Draft

SFC SF Offloads

October 2016

## Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">3</a>
<a href="#">1.1.</a>	Requirements Language . . . . .	<a href="#">3</a>
<a href="#">2.</a>	Definition Of Terms . . . . .	<a href="#">3</a>
<a href="#">3.</a>	Service Function Path Reduction . . . . .	<a href="#">4</a>
<a href="#">3.1.</a>	Bypass . . . . .	<a href="#">4</a>
<a href="#">3.2.</a>	Simple Offload . . . . .	<a href="#">5</a>
<a href="#">3.2.1.</a>	Stateful SFF . . . . .	<a href="#">7</a>
<a href="#">3.2.2.</a>	Packet Re-ordering . . . . .	<a href="#">7</a>
<a href="#">3.2.3.</a>	Race Conditions . . . . .	<a href="#">8</a>
<a href="#">3.2.4.</a>	Policy Implications . . . . .	<a href="#">8</a>
<a href="#">3.2.5.</a>	Capabilities Exchange . . . . .	<a href="#">8</a>
<a href="#">4.</a>	Methods For SFP Reduction . . . . .	<a href="#">9</a>
<a href="#">4.1.</a>	SFP In-band Offload . . . . .	<a href="#">9</a>
<a href="#">4.1.1.</a>	Progression Of SFP Reduction . . . . .	<a href="#">11</a>
<a href="#">4.2.</a>	Service Controller Offload . . . . .	<a href="#">12</a>
<a href="#">5.</a>	Simple Offload Data-plane Signaling . . . . .	<a href="#">13</a>
<a href="#">5.1.</a>	Offload Flags Definition . . . . .	<a href="#">14</a>
<a href="#">6.</a>	Acknowledgements . . . . .	<a href="#">15</a>
<a href="#">7.</a>	IANA Considerations . . . . .	<a href="#">15</a>
<a href="#">7.1.</a>	Standard Class Registry . . . . .	<a href="#">15</a>
<a href="#">7.1.1.</a>	Simple Offloads TLV . . . . .	<a href="#">15</a>
<a href="#">8.</a>	Security Considerations . . . . .	<a href="#">16</a>
<a href="#">9.</a>	References . . . . .	<a href="#">16</a>
<a href="#">9.1.</a>	Normative References . . . . .	<a href="#">16</a>
<a href="#">9.2.</a>	Informative References . . . . .	<a href="#">16</a>
	Authors' Addresses . . . . .	<a href="#">16</a>

Internet-Draft

SFC SF Offloads

October 2016

## 1. Introduction

Service function chaining involves steering traffic flows through a set of service functions in a specific order. Such an ordered list of service functions is called a Service Function Chain (SFC). The actual forwarding path used to realize an SFC is called the Service Function Path (SFP).

Service functions forming an SFC are hosted at different points in the network, often co-located with different types of service functions to form logical groupings. Applying a SFC thus requires traffic steering by the SFC infrastructure from one service function to the next until all the service functions of the SFC are applied. Service functions know best what type of traffic they can service and how much traffic needs to be delivered to them to achieve complete delivery of service. As a consequence any service function may potentially request, within its policy constraints, traffic no longer be delivered to it or its function be performed by the SFC infrastructure, if such a mechanism is available.

While there are several possible means to achieve this, one of the most flexible, directly connected to functional semantics, is based on allowing service functions themselves to evaluate a particular flow and reflect the result of this evaluation back to the SFC infrastructure.

This document outlines the "simple offloads" mechanism that avoids steering traffic to service functions on flow boundary, on request from the service functions, while still ensuring compliance to the instantiated policy that mandates the SFC.

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

## 2. Definition Of Terms

This document uses the following terms. Additional terms are defined in [[RFC7498](#)], [[I-D.ietf-sfc-architecture](#)] and [[I-D.ietf-sfc-nsh](#)].

Service Controller (SC): The entity responsible for managing the service chains, including create/read/update/delete actions as well as programming the service forwarding state in the network - SFP distribution.

Kumar, et al.

Expires April 23, 2017

[Page 3]

---

Internet-Draft

SFC SF Offloads

October 2016

Classifier (CF): The entity, responsible for selecting traffic as well as SFP, based on policy, and forwarding the selected traffic on the SFP after adding the necessary encapsulation. Classifier is implicitly an SFF.

Offload: A request or a directive from the SF to alter the SFP so as to remove the requesting SF from the SFP while maintaining the effect of the removed SF on the offloaded flow.

## 3. Service Function Path Reduction

The packet forwarding path of a SFP involves the classifier, one or more SFFs and all the SFs that are part of the SFP. Packets of a flow are forwarded along this path to each of the SFs, for the life of the flow, whether SFs perform the full function in treating the packet or reapply the cached result, from the last application of the function, on the residual packets of the flow. In other words, every packet on the flow incurs the same latency and the end-to-end SFP latency remains more or less constant subject to the nature of the SFs involved. If an SF can be removed from the SFP, for a specific flow, traffic steering to the SF is avoided for that flow; thus leading to a shorter SFP for the flow. When multiple SFs in a SFP are removed, the SFP starts to converge towards the optimum path, incurring a fraction of the latency associated with traversing the SFP.

Although SFs are removed from the SFP, the corresponding SFC is not changed - this is subtle but an important characteristic of this mechanism. In other words, this mechanism does not alter the SFC and

still uses the SFP associated with the SFC.

There are two primary approaches to removing an SF from the SFP. Namely,

- o Bypass: Mechanism that alters the SFC. Described in this draft for completeness.
- o Simple Offload: Mechanism that alters the SFP alone, does not affect the SFC. This is the primary focus of this draft.

### [3.1.](#) Bypass

Many service functions do not deliver service to certain types of traffic. For instance, typical WAN optimization service functions are geared towards optimizing TCP traffic and add no value to non-TCP traffic. Non-TCP traffic thus can bypass such a service function. Even in the case of TCP, a WAN optimization SF may not be able to service the traffic if the corresponding TCP flow is not seen by it

from inception. In such a situation a WAN optimization SF can avoid the overhead of processing such a flow or reserving resources for it, if it had the ability to request such flows not be steered to it. In other words such service functions need the ability to request they be bypassed for a specified flow from a certain time in the life of that flow.

A seemingly simple alternative is to require service functions pre specify the traffic flow types they add value to, such as the one-tuple: IP protocol-type described above. A classifier built to use such data exposed by SFs, may thus enable bypassing such SFs for specific flows by way of selecting a different SFC that does not contain the SF being removed.

Although knowledge of detailed SF profiles helps SFC selection at the classifier starting the SFC, it leads to shortcomings.

- o It adds to the overhead of classification at that classifier as all SF classification requirements have to be met by the classifier.
- o It leads to conflicts in classification requirements between the

classifier and the SFs. Classification needs of different SFs in the same SFC may vary. A classifier thus cannot classify traffic based on the classification of one of the SFs in the chain. For instance, even though a flow is uninteresting to one SF on an SFC, it may be interesting to another SF in the same SFC.

- o The trigger for bypassing an SF may be dynamic as opposed to the static classification at the classifier – it may originate at the SFs themselves and involve the control and policy planes. The policy and control planes may react to such a trigger by instructing the classifier to select a different SFC for the flow, thereby achieving SF bypass.

### 3.2. Simple Offload

Service delivery by a class of service functions involves inspecting the initial portion of the traffic and determining whether traffic should be permitted or dropped. In some service functions, such an inspection may be limited to just the five tuple, in some others it may involve protocol headers, and in yet others it may involve inspection of the byte stream or application content based on the policy specified. Firewall service functions fall into such a class, for example. In all such instances, servicing involves determining whether to permit the traffic to proceed onwards or to deny the traffic from proceeding onwards and drop the traffic. In some cases, dropping of the traffic may be accompanied with the generation of a

response to the originator of traffic or to the destination or both. Once the service function determines the result – permit or deny (or drop), it simply applies the same result to the residual packets of the flow by caching the result in the flow state.

In essence, the effect of service delivery is a PERMIT or a DENY action on the traffic of a flow. This class of service functions can avoid all the overhead of processing such traffic at the SF, by simply requesting another entity in the SFP, to assume the function of performing the action determined by the service function. Since PERMIT and DENY are very simple actions, other entities in the SFP are very likely to be able to perform them on behalf of the requesting SF. A service function can thus offload simple functions to other entities in the SFP.

As with PERMIT and DENY actions, there are others which are simple enough to be supported. Some are listed here for illustration.

**Unidirectional Offload:** Client-Server communication, typical of HTTP request-response transactions, imposes higher cost on SFs in one direction. Responses often carry more bytes, sometimes orders of magnitude more, as compared to requests. SFs could avoid the cost of moving the bits in the response direction to which it may add no value, once the policy is satisfied, if the response flow can be offloaded. Hence Offloads must be requestable on a unidirectional flow boundary.

**TCP Control Exception Offload:** Most SFs maintain flow state and would like to know when a flow terminates, so SFs can cleanup the flow state and associated resources. Such SFs need to receive the TCP control packets, the ones with control flags [[RFC0793](#)] set, on the flow even when the flow itself is offloaded, in order to perform such activity. Hence Offloads must be predicatable to offload all but the TCP control packets of a flow.

**Time Limited Offload:** SF policy may dictate flows be limited to certain period of time among other reasons to optimize SF load. SFs can request a flow be offloaded for a specific time duration after which, all traffic on that flow gets redirected to the SF as was done before the offload was initiated. Hence Offloads must be requestable on a time limit.

**Volume Limited Offload:** As with time limited offloads, SF policy may dictate flows be limited to certain volume of data. SFs can request a flow be offloaded until a specified number of bytes traverse the flow. Hence Offloads must be requestable on a volume limit.

Since SFF is the one steering traffic to the SFs and hence is on the SFP, it is a natural entity to assume the offload function. A SF not interested in traffic being steered to it can simply perform a simple offload by indicating a PERMIT action along with an OFFLOAD request. The SFF responsible for steering the traffic to the SF takes note of the ACTION and offload request. The OFFLOAD directive and the ACTION received from the requesting SF are cached against the SF for that flow. Once cached, residual packets on the flow are serviced by the

cached directive and action as if being serviced by the corresponding SF.

#### [3.2.1.](#) Stateful SFF

SFFs are the closest SFC infrastructure entities to the service functions. SFFs may be state-full and hence can cache the offload and action in both of the unidirectional flows of a connection. As a consequence, action and offload become effective on both the flows simultaneously and remain so until cancelled or the flow terminates.

SFFs may not always honor the offload requests received from SFs. This does not affect the correctness of the SFP in any way. It implies that the SFs can expect traffic to arrive on a flow, which it offloaded, and hence must service them, which may involve requesting an offload again. It is natural to think of an acknowledgement mechanism to provide offload guarantees to the SFs but such a mechanism just adds to the overhead while not providing significant benefit. Offload serves as a best effort mechanism.

#### [3.2.2.](#) Packet Re-ordering

The simple offload mechanism creates short time-windows where packet re-ordering may occur. While SFs request flows be offloaded to SFFs, packets may still be in flight at various points along the SFP, including some between the SFF and the SF. Once the offload decision is received and committed into the flow entry at the SFF, any packets arriving after and destined to the offloading SF are treated to the offload decision and forwarded along (if it is a PERMIT action). Inflight packets to the offloading SF may arrive at the SFF after one or more packets are already treated to the offload decision and forwarded along.

This is a transitional effect and may not occur in all cases. For instance, if the decision to offload a flow by an SF is based on the first packet of TCP flow, a reasonable time window exists between the offload action being committed into the SFF and arrival of subsequent packet of the same flow at that SFF. Likewise, request/response based protocols such as HTTP may not always be subject to the re-ordering effects.

#### [3.2.3.](#) Race Conditions



The tuple that make up an end-to-end flow or connection, such as a five tuple TCP connection, may be reused in a very short span of time when very high performing end points are involved. A very remote manifestation of this behavior may involve the wrong incarnation of a flow at the SFF receiving the flow offload request from a SF.

Implementations of simple offloads must thus be aware of such a possibility and include appropriate measures to address it. It is important to note that a SFF must maintain correctness and hence it is acceptable to not honor a simple offloads request to resolve such an occurrence. After all SFs exist with right security posture to protect against malicious traffic.

A simple and widely used method to serialize reuse of tuples is to use an incarnation number in addition to the five-tuple. The steering SFF can pass an opaque cookie, which in its simplest form could be the incarnation number, that is preserved by the SF and passed along with the simple offload request. SFF can thus correctly identify the right incarnation of the flow. SYN detection at the SFF to take corrective action is another option. The SFF implementations may employ any technique deemed appropriate.

#### [3.2.4.](#) Policy Implications

Offload mechanism may be controlled by the policy layer. The SFs themselves may have a static policy to utilize the capability offered by the SFC infrastructure. They could also be dynamic and controlled by the specific policy layer under which the SFs operate.

Similarly, the SFC infrastructure, specifically the classifiers and the SFFs, may be under the SFC infrastructure control plane policy controlling the decision to honor offloads from an SF. This policy in turn may be coarse-grain, at the SF level, and hence static. It can also be fine grain and hence dynamic but it adds to the overhead of policy distribution.

Policy model related to offloads is out of scope of this document.

#### [3.2.5.](#) Capabilities Exchange

Simple offloads can be exposed and negotiated a priori as a capability between the SFFs and the SFs or the corresponding control layers. In the simplest of the implementations, this is provided by the SFC infrastructure and the SFs are statically configured to utilize them without capabilities negotiation, within the constraints of the SF specific policies.

Capabilities exchange is outside the scope of this document.

#### [4.](#) Methods For SFP Reduction

There are a number of different models that may be used to facilitate SFP shortening.

The methods discussed in the following sections require signaling among the participant components to communicate offload and permit/deny actions. The signaling may be performed in the data-plane or in the control plane.

- a. Data-plane: A SFC specific communication channel is needed for SFs to communicate the offload request along with the SF treated packet. [NSH] defines a header specifically for carrying SFP along with metadata and provides such a channel for use with offloads. Necessary bits need to be allocated in NSH to convey the action as well as the offload directive. This signaling may be limited to SF and SFF or may continue from one SFF to another SFF or the classifier. It may also involve signaling directly from the SF to the classifier.
- b. Control-plane: Messages are required between the SF and the service controller as well as between the SFF and the service controller. Service controller messaging is out of scope of this document and it is assumed to be service controller specific, which may include open or standard interfaces.

##### [4.1.](#) SFP In-band Offload

SFs receive traffic on an overlay from the SFF. SFs service the traffic and turn them back to the SFF on an overlay or forward the traffic on the underlay. In the former case, along with returning the traffic to SFF, they can perform simple offload by signaling OFFLOAD and ACTION to the SFF. SFF caches the OFFLOAD and ACTION while forwarding the serviced packet onwards to the next service hop on the SFP or dropping it as per the ACTION. This may continue from one hop to the next on the SFP. SFF can now enforce the OFFLOAD and ACTION on the residual packets of the flow.

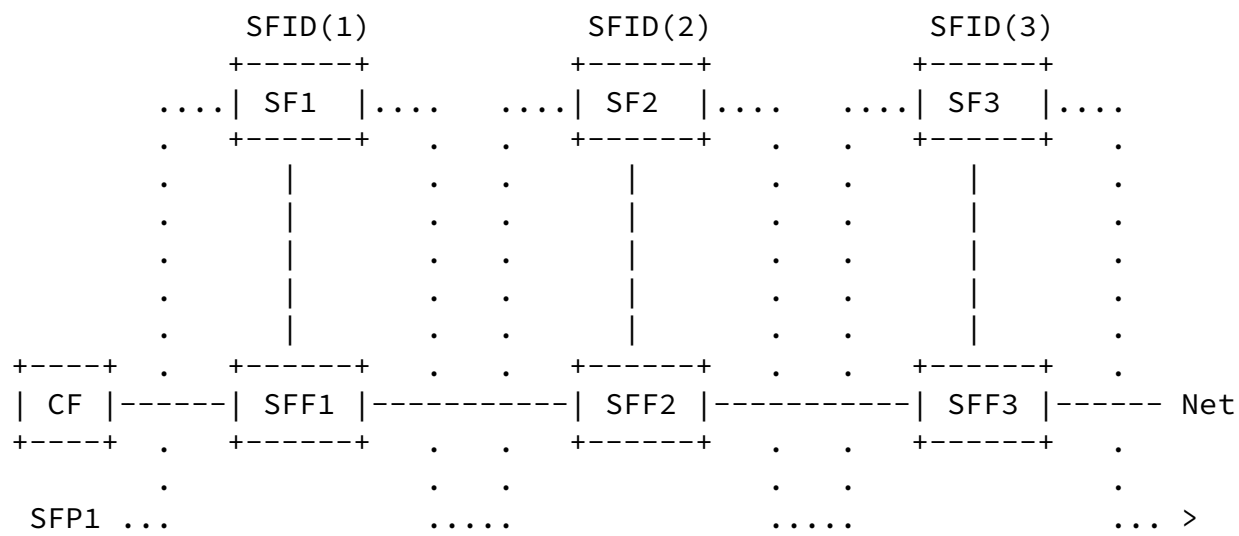
By performing such hop-by-hop offloads, SFP can be reduced from its original length, steering traffic to only the SFFs and the SFs that really need to see the traffic.

Figure 1 to Figure 3 show an example of SF and SFF performing offload operations, with PERMIT action, and the effect thereafter on the SFP.

Internet-Draft

SFC SF Offloads

October 2016



SFC1 = {SF1, SF2, SF3}  
 SFC1 -> SFP1

Where,

SFC1 is a service function chain

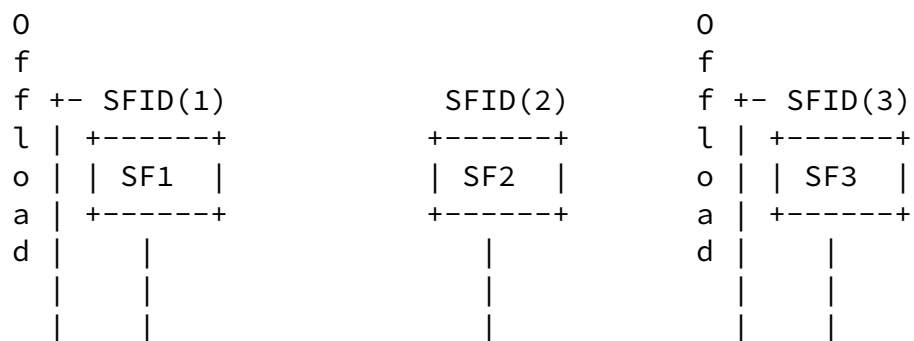
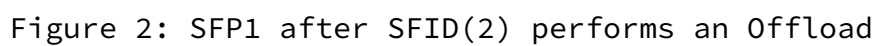
SF1, SF2 and SF3 are three service functions

SFP1 is the service function path for SFC1

CF is the classifier starting SFP1 based on policy

Note: Network forwarders are omitted from the figure for simplicity

Figure 1: SFC1 with corresponding SFP1



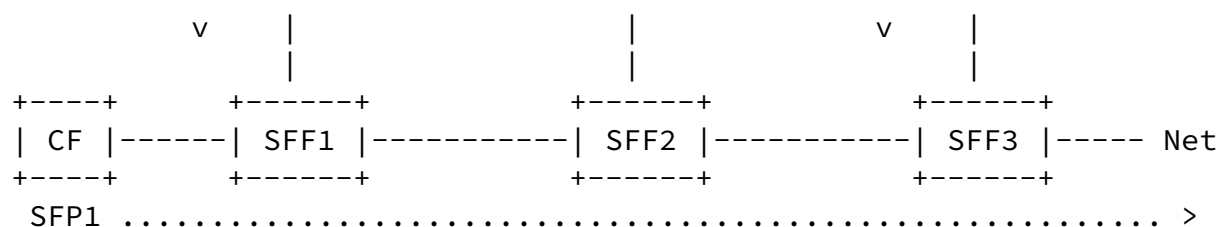


Figure 3: SFP1 after SFID(1) and SFID(3) perform Offloads

#### [4.1.1.](#) Progression Of SFP Reduction

SFP reduction happens one SFF at a time: by collapsing the SFF-to-SF hops into the SFF or the SFC infrastructure.

Figure 1 to Figure 3 show one sequence of offload events that lead to a shorter SFP.

Corresponding transformation of the actual forwarding path is captured by the states below.

Stage-1: Prior to any offloads, service function path SFP1 (corresponding to SFC1) has the following actual forwarding path as shown in Figure 1:

```

CF ->
SFF1 -> SF1 -> SFF1 ->
SFF2 -> SF2 -> SFF2 ->
SFF3 -> SF3 -> SFF3 ->

```

Stage-2: After SF2 performs a simple offload, service function path SFP1 changes to the one represented below, as also shown in Figure 2:

```

CF ->
SFF1 -> SF1 -> SFF1 ->
SFF2 ->
SFF3 -> SF3 -> SFF3 ->

```

Stage-3: After SF1 and SF3 both perform simple offloads, service function path SFP1 changes to the one represented below, as also show in Figure 3:

CF ->  
SFF1 ->  
SFF2 ->  
SFF3 ->

When all the SFs in a SFP perform offloads the forwarding path is reduced to pass through just the SFFs.

#### [4.2.](#) Service Controller Offload

Each SF signals the service controller of the OFFLOAD and ACTION via control plane messaging for a specific flow. The service controller then signals the appropriate SFFs to offload the requested SFs, thereby achieving the hop-by-hop offload behavior.

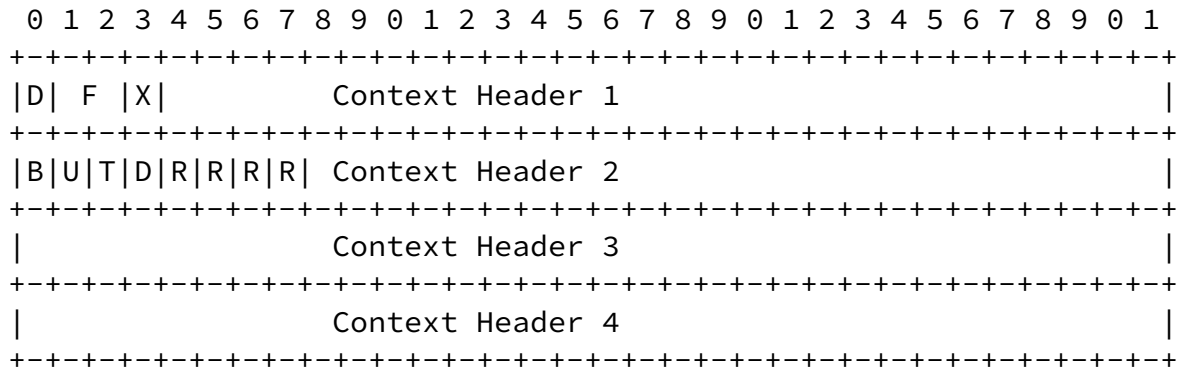
The service controller has full knowledge of all the SFs of the SFP offloading the flow and hence can determine the optimum SFP within the Service Controller and program the appropriate SFFs to achieve SFP optimization.

### [5.](#) Simple Offload Data-plane Signaling

Since Offload and action are signaled at the time of returning the traffic to SFF, post servicing the traffic, such signaling can be integrated into the SFC service header of the packet.

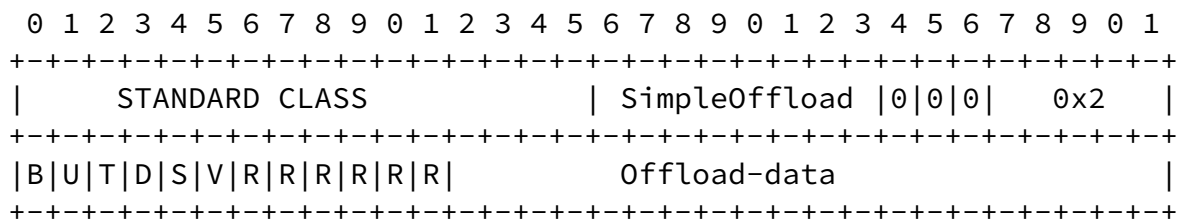
Figure 4 and Figure 5 show the bits necessary to achieve the signaling using the SFC encapsulation as described in [\[I-D.ietf-sfc-nsh\]](#). In particular, for NSH MD-Type1 header format, the offload bits are communicated via the flags field in the very first byte of the fixed context headers. For NSH MD-Type2 header format, the offload bits are communicated via a new standard TLV - Simple Offload TLV. The standard TLV is requested to be allocated from the TLV Class, "Standard Class", from the IANA.

By integrating the signaling with the packets, the simple offloads scale with the traffic in the data plane.



X : Extend flags into first byte of "Context Header 2"  
 B : Bidirectional Offload  
 U : Unidirectional Offload  
 T : TCP-control Exception Offload  
 D : Drop Offload

Figure 4: NSH Type-1 Offload Bits shown for DC Allocation



B : Bidirectional Offload  
 U : Unidirectional Offload  
 T : TCP-control Exception Offload  
 D : Drop Offload

S : Time Limited Offload  
V : Volume Limited Offload

Figure 5: NSH Type-2 Offload Bits

### [5.1.](#) Offload Flags Definition

#### Offload Control Flags:

B, Bidirectional Offload: SF requests both flows in the connection, described by the payload, be offloaded, by setting B=1. B=0 otherwise.

U, Unidirectional Offload: SF requests only the current flow in the connection, described by the payload, be offloaded, by setting U=1. U=0 otherwise.

One and only one of 'B' and 'U' MUST be specified to indicate offload. In the event a NSH encapsulated packet is received with both 'B' and 'U' offload flags set to 1, 'B' MUST take precedence.

#### Offload Function Flags:

B|U, Permit Offload: When either B=1 or U=1, the implicit function is to PERMIT or allow all packets on the flow(s) to traverse along the SFP, unless over-ridden by other functional flags.

D, Drop Offload: Setting D=1, requests packets on the offloaded flow(s) be dropped; D MUST be set to 0 otherwise. D=1 modifies the default PERMIT behavior of 'B' and 'U' flags.

T, TCP-control Exception Offload: Setting T=1 requests TCP control packets to be exempted from Offload behavior. TCP control packets MUST continue to be forwarded to the SF while the rest of the packets must be allowed to bypass the SF contingent upon the

application of other offload flags. T MUST be set to 0 otherwise.

S, Time Limited Offload: Setting S=1 requests the flow(s) to be



offloaded for the duration specified, in seconds, in offload-data field. After that duration, offload behavior must be cancelled and affected flow(s) MUST be redirected to the SF. S MUST be set to 0 otherwise.

V, Volume Limited Offload: Setting V=1 requests the flow(s) to be offloaded until the volume of data specified, in Kilo Bytes, in offload-data field has traversed the flow(s). After that volume of data has traversed, offload behavior must be cancelled and affected flow(s) MUST be redirected to the SF. V MUST be set to 0 otherwise.

## 6. Acknowledgements

The authors would like to thank Abhjit Patra, Nagaraj Bagepalli, Kent Leung, Erik Nordmark, Diego Lopez for their comments, thoughtful questions and suggestions, review, etc.

## 7. IANA Considerations

### 7.1. Standard Class Registry

IANA is requested to allocate a "STANDARD" class from the TLV Class registry. Allocation of the registry values under this class shall follow the "IETF Review" policy defined in [RFC 5226](#) [[RFC5226](#)].

#### 7.1.1. Simple Offloads TLV

IANA is requested to allocate TLV type with value 0x1 from the STANDARD TLV class registry. The format of the "Simple Offloads" TLV is as defined in this draft.

TLV#	Name	Description	Reference
1	Simple Offloads	SF Flow Offload to SFF	This document

Table 1: Standard Class Registry

## 8. Security Considerations

Security of the offload signaling mechanism is very important. This document does not advocate any additional security mechanisms beyond the data plane and control plane signaling security mechanisms.

## 9. References

### 9.1. Normative References

- [I-D.ietf-sfc-architecture]  
Halpern, J. and C. Pignataro, "Service Function Chaining (SFC) Architecture", [draft-ietf-sfc-architecture-11](#) (work in progress), July 2015.
- [I-D.ietf-sfc-nsh]  
Quinn, P. and U. Elzur, "Network Service Header", [draft-ietf-sfc-nsh-10](#) (work in progress), September 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

### 9.2. Informative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, [RFC 793](#), DOI 10.17487/RFC0793, September 1981, <<http://www.rfc-editor.org/info/rfc793>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC7498] Quinn, P., Ed. and T. Nadeau, Ed., "Problem Statement for Service Function Chaining", [RFC 7498](#), DOI 10.17487/RFC7498, April 2015, <<http://www.rfc-editor.org/info/rfc7498>>.

## Authors' Addresses

Surendra Kumar  
Cisco Systems, Inc.  
170 W. Tasman Dr.  
San Jose, CA 95134

Email: smkumar@cisco.com

Kumar, et al.

Expires April 23, 2017

[Page 16]

---

Internet-Draft

SFC SF Offloads

October 2016

Jim Guichard  
Cisco Systems, Inc.

Email: jguichar@cisco.com

Paul Quinn  
Cisco Systems, Inc.

Email: paulq@cisco.com

Joel Halpern  
Ericsson

Email: joel.halpern@ericsson.com

Sumandra Majee  
F5 Networks  
90 Rio Robles  
San Jose, CA  
US

Email: S.Majee@F5.com

Kumar, et al.

Expires April 23, 2017

[Page 17]