Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 12, 2015

K. Ma
Ericsson
R. Parker
Affirmed Networks

August 11, 2014

SFC Service Decomposition
draft-ma-sfc-decomposition-02

Abstract

This document discusses the role of composite (monolithic) service functions in service function chaining, and describes a method for supporting composite service function decomposition.

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].

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 February 12, 2015.

Copyright Notice

Table of Contents

1. Introduction

Service function chaining (SFC), as defined in the SFC Problem Statement [I-D.quinn-sfc-problem-statement], affects "packets and/or frames selected as a result of classification. Packet and/or frame-based classification, however, is limited to information that is available in every packet and/or frame. Typical methods of using VLAN tags or TCP(UDP)/IP 5-tuples (i.e., source IP address, destination IP address, IP protocol, source port, and destination port) allow frame/packet flows to be easily identified and directed through an appropriate service function (SF) or chain. The service functions themselves, however, are often more complex and act upon higher level message constructs (e.g., HTTP/1.x messages or chunks [RFC2616], or HTTP/2.x frames [I-D.ietf-httpbis-http2]) whose boundaries do not typically align with L2/3 frame/packet boundaries. Particularly in the cases of application acceleration, deep packet inspection (DPI), server load balancing (SLB), anti-virus scanning, and various HTTP-based protocol optimizations (e.g., HTTP live streaming (HLS) [I-D.pantos-http-live-streaming] video delivery), the limitations of frame/packet-based classification necessitate flow termination and re-classification, in order to perform the service function. Given the overhead cost of flow termination and re-

classification, multiple service functions are often merged into a single monolithic service function (or composite service function) to take advantage of economies of scale.

Basic service function chaining allows multiple composite service functions (perhaps provided by different vendors) to be ordered and routed through, however, duplication of message-level re-classification is likely. There may also be duplication of component service functions by the independent composite service functions. Decomposition of the composite service functions could reduce the overhead that results from duplicated processing and provide more granular control over virtualized service functions. Decomposition also enables component function isolation which is useful for virtualization and scaling.

This document describes the challenges associated with composite service function decomposition and the requirements for supporting service function chaining with component services. The document uses an HLS video session to illustrate the decomposition concepts, but it should be understood that service function decomposition is applicable to a wide variety of application layer services.

1.1. Terminology

This document uses the terminology defined in the SFC Problem Statement [I-D.quinn-sfc-problem-statement]. Additional terminology is defined below:

- Composite Service Function: A service function that consumes a frame/packet flow, but is made up of multiple component service functions that act on message flows.

- Component Service Function: A service function that may be separated out from a composite service function.

- Decomposed Service Function Chain: A service function chain composed of component service functions, which begins and ends at a composite service function endpoint.

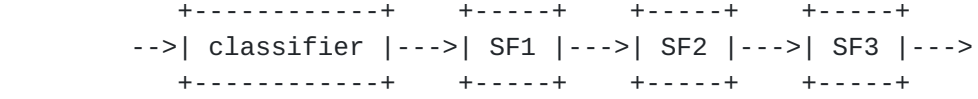
- Split Chain: A service function chain with multiple paths between two component service functions, consisting of different component service functions.

- Demux Service Function: A service function which performs stateful classification, separating incoming flow data, selecting different service function paths for different subsets of the flow data, with the understanding that the processed flow data must be reassembled at a further downstream service function.

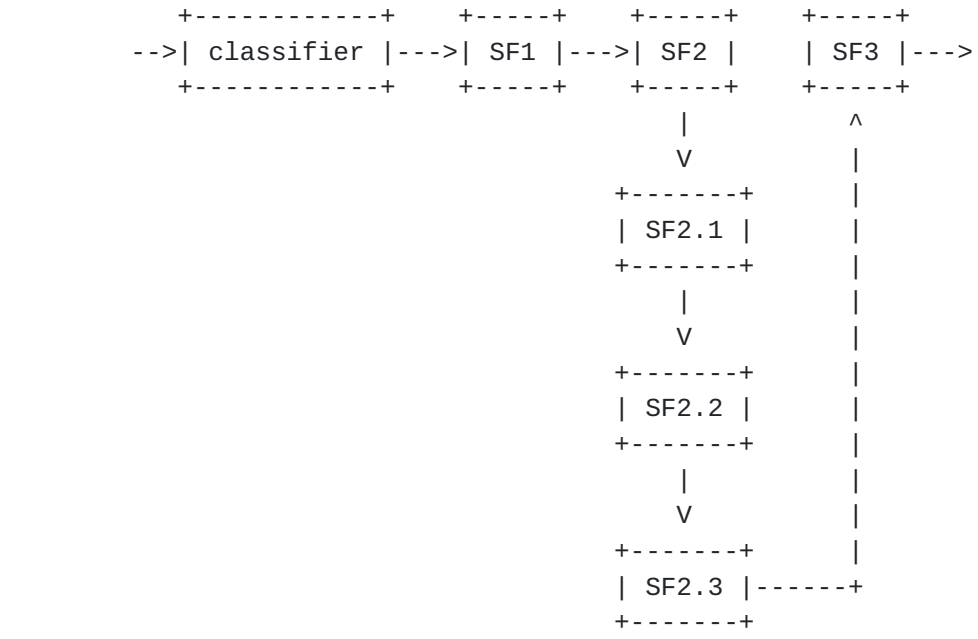
- Remux Service Function: A service function which recombines incoming flow data arriving from different service function paths, previously separated by an upstream demux service function.

1.2. Nested Service Function Chains

The concept of a single service function chain identified by a single ingress classifier (as described in the SFC Architecture [I-D.quinn-sfc-problem-statement] document and as shown in the figure below) is fairly well understood.



Though re-classification is mentioned in the SFC Architecture [I-D.quinn-sfc-problem-statement] document, composite service function decomposition provides a concrete use case for reclassification and highlights the hierarchical nature of service function chaining. The nesting of decomposed service function chains within a higher level encapsulating service chain (as shown in the figure below) illustrates this hierarchical chaining concept.



In this nested scenario, the ingress classifier is still only aware of the service function chain through SF1, SF2, and SF3. The fact that SF2 might perform re-classification and decompose into a lower level chain (i.e., SF2.1, SF2.2, and SF2.3) is hidden from the ingress classifier. In general, at any given level, a (re-)classifier is oblivious to any further downstream re-

classification. This type of hierarchical abstraction becomes a powerful tool for the design and insertion of new service functions into a given network deployment.

1.3. Ingress Demux Service Functions

The examples shown in the following sections show options for both "packets from client" and "from previous in chain". This distinction is to handle the following two separate use cases for service function decomposition, with respect to insertion of the service function into the client data path:

- Direct Addressing: where the destination IP address of the packets from the client explicitly route to the demux service function node, and

- SFC steering: where the composite service function is a member of a higher level service function chain and packets from the previous service function in the higher level chain are forwarded based on the configured SFC path resolution.

For many application level service functions, the selection of a service function may occur in the client at the application layer (e.g., localized DNS-based request routing may explicity resolve requests to a service function specific IP address). In these cases, the composite service function is not necessarily part of a hierarchical service function chain; the monolithic service node providing the composite service function would be a standalone proxy service. Though a frame/packet-based classifier is not required here, this still constitutes a valid SFC use case where a composite service function is decomposed into a set of service function chains. In this case, the demux service function, directly identified (e.g., via DNS), acts as the (message-based) ingress classifier for the decomposed service function chain. Packets arriving at the demux service function will not be SFC encapsulated, but will instead have IP headers containing locally owned destination addresses.

Alternatively, if the composite service function is nested within a service function chaining hierarchy, then packets destined for the composite service function could have been classified by an upstream (frame/packet-based) ingress classifier. In this case, the packets arriving from at the demux service function will have SFC encapsulation headers. Decomposed service function chain ID selection must take into account the need for any egress SFC encapsulation required for propagating data to a next service function in the higher level service function chain.

For simplicity, the examples in this document do not make any assumptions about the ingress path of packets destined for the demux service function or the egress next hop of packets released out of the remux service function; it should be understood that either direct addressing or SFC steering should work with decomposed service function chaining.

2. Service Function Decomposition

The following sections describe the use case, challenges, and solution requirements for decomposed service function chaining.

2.1. HLS Use Case

In an HLS live streaming session, the client polls a sliding window manifest file which indicates the most recent MPEG-TS segment files available for retrieval. An HLS video optimization service function may perform CDN selection, request routing (RR), and/or SLB, as well as perform URL and HTTP header rewrite on both the manifest and segment requests. For manifest responses, manifest manipulation may be used to perform ad insertion, rate limiting, or other user entitlement enforcement services. For segment responses, segment manipulation may be used to perform internationalization (i.e., alternate audio and/or subtitles), rate limiting, watermarking, etc. service. Each of these service functions are applied at an application messaging layer, not at a L2/3 frame/packet layer.

In a secure session, manifest requests are protected by TLS (i.e., using HTTPS), while segment requests may not be if the segment files themselves are already encrypted. Manifest and segment requests do not typically use persistent HTTP/1.x connections but could. HTTP/2.0 muxed streams with server push could also be used. These each pose different issues for composite service function decomposition, as described below:

- HTTP/1.x (non-persistent): The different non-persistent connection flows (identified by TCP/IP 5-tuple) still need to be associated in order to apply user entitlements and to collect session analytics. Manifest requests may also require TLS termination.

- HTTP/1.x (persistent): The requests for manifests and segments occur over the same persistent connection flow (identified by TCP/IP 5-tuple) have different processing requirements for request routing and content transformation.

- HTTP/2.0 (muxed): In addition to the separate processing requirements for request routing and content transformation, the interleaving of data frames for manifest and segment responses

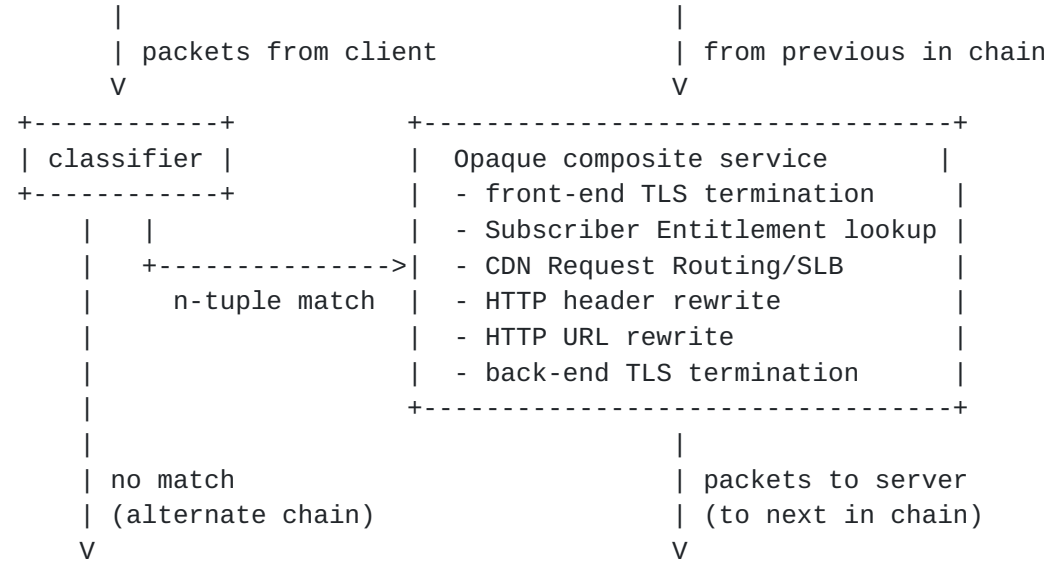
travelling over the same persistent connection flow (identified by TCP/IP 5-tuple) must be re-associated in real-time.

- HTTP/2.0 (server push): With server push, there is no longer a one-to-one correlation between request and response file which complicates reverse path mapping for bi-directional flows.

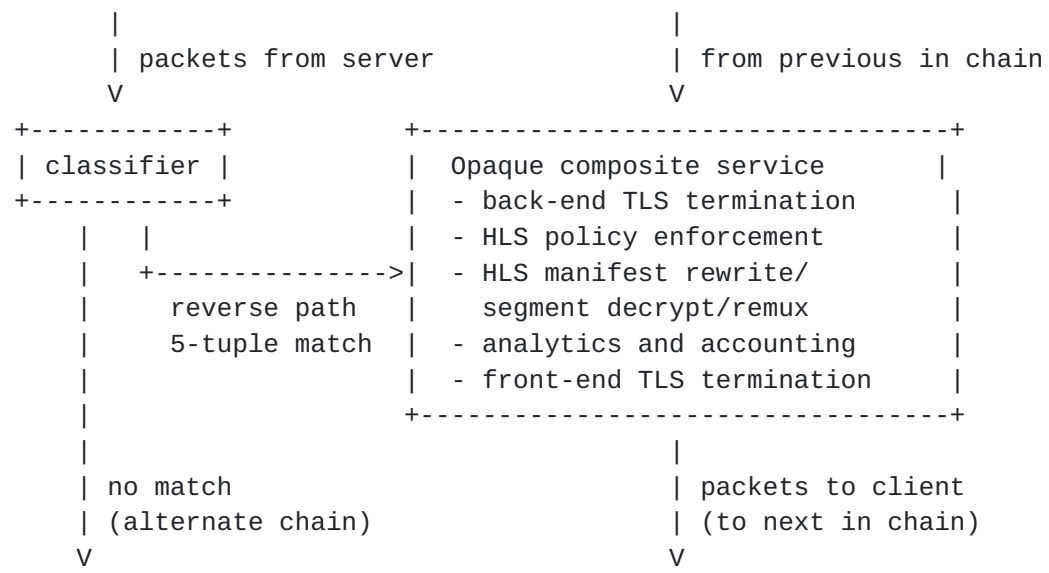
For an HLS video optimization service to be decomposed, the individual packets associated with each message need to be identified and marked in a way that frame/packet-based SFC-aware devices can (and will) use to forward the frames/packets to the next component service function in the chain.

2.2. Composite Service Function

Using the most interesting case of HTTP/2.0 over TLS, with manifests and segments muxed on the same TCP connection and using server push, we can create a bidirectional composite service function as shown in the figures below.



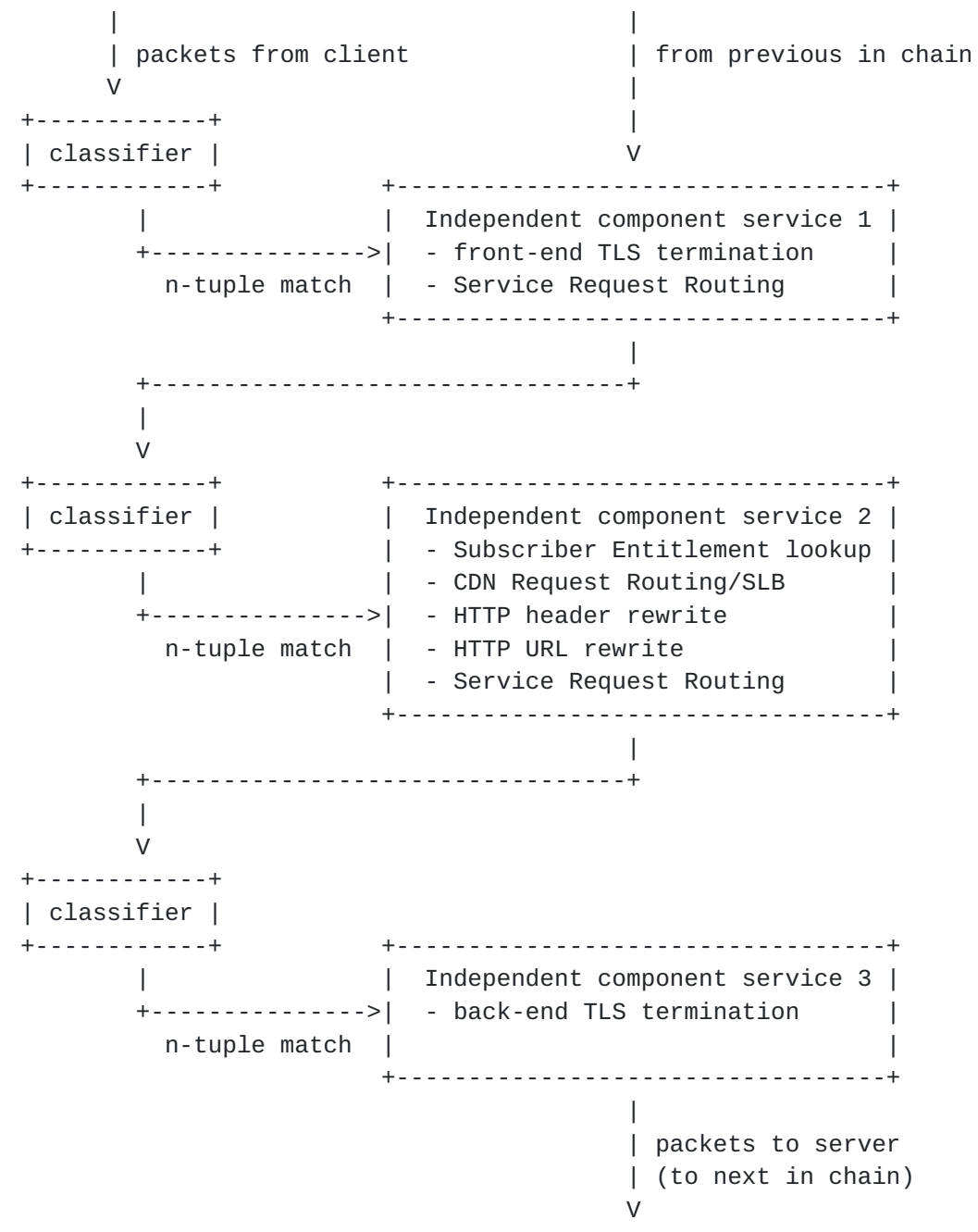
The ingress path (classified using a 3 or 5-tuple match on the front-end TLS connection) terminates TLS, extracts user credentials, looks up subscriber information, enforces subscriber entitlements, performs RR/SLB, updates the URL and request headers then forwards the request to the server, along the chain, over TLS.



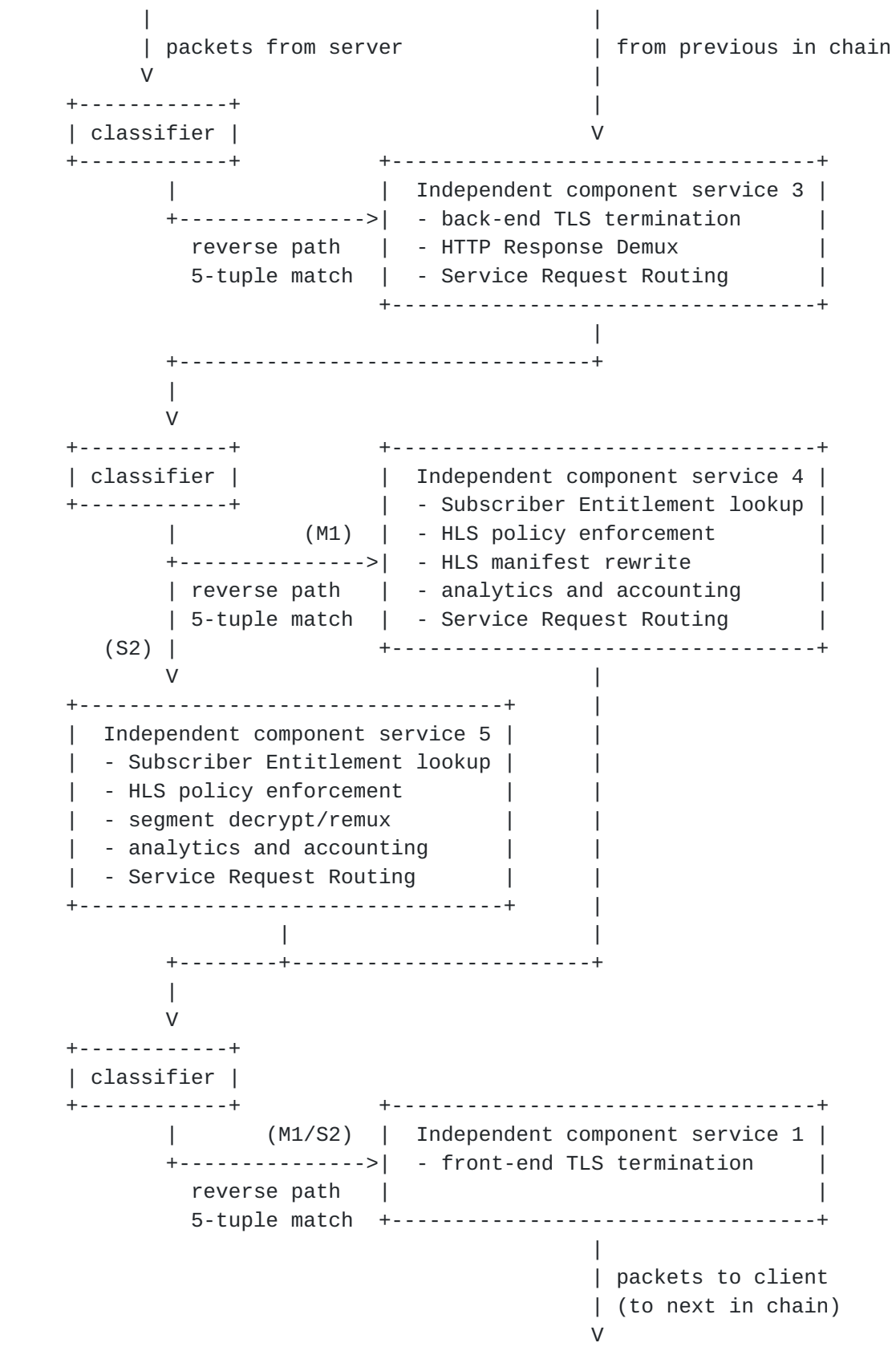
The egress path (classified using a 5-tuple match on the back-end TLS connection) receives content on the TLS connection it initiated, reconstructs the manifest/segment file, looks up subscriber information, enforces subscriber entitlements, and depending on the type of file, performs manifest/segment manipulation and records session analytics, then forwards the request to the client (along the chain), over the original TLS connection.

2.3. Non-SFC Service Decomposition

Service decomposition could be accomplished without SFC, using the traditional means of either VLAN stitching or statically configured next hop IP addresses. These methods suffer from all the limitations described in the SFC Problem Statement [I-D.quinn-sfc-problem-statement] document. An example decomposition is shown in the figures below.



The ingress path is divided into three independent component service functions, splitting out the front-end and back-end TLS processing which is not HLS video delivery specific and may be commoditized. (The two TLS processing service functions may actually be two separate instantiations of the same service.) The subscriber entitlement enforcement, RR/SLB, and URL/headers updates are performed by a separate independent service function.


packets from server
from previous in chain
classifier
reverse path
5-tuple match
Independent component service 3
- back-end TLS termination
- HTTP Response Demux
- Service Request Routing
classifier
(M1)
reverse path
5-tuple match
(S2)
Independent component service 4
- Subscriber Entitlement lookup
- HLS policy enforcement
- HLS manifest rewrite
- analytics and accounting
- Service Request Routing
Independent component service 5
- Subscriber Entitlement lookup
- HLS policy enforcement
- segment decrypt/remux
- analytics and accounting
- Service Request Routing
classifier
(M1/S2)
reverse path
5-tuple match
Independent component service 1
- front-end TLS termination
packets to client
(to next in chain)

The egress path is divided into four independent component service functions, splitting out the front-end and back-end TLS processing (identical to the ingress path, just in reverse order) as well as separating manifest and segment manipulation functions, given the different resource requirements associated with each function.

Note: The HTTP response demux could be separated out from the back-end TLS service function, however, they are left combined in this example for simplicity. The key point is that demuxing from a single TCP session occurs and the traffic is split between more than one downstream path.

It should be noted that after each of the independent service functions, re-classification is required. In the case of the two TLS SFs, TCP termination is required in order to terminate the front-end and back-end TLS sessions. In the case of the RR/SLB SF, though TCP termination is not explicitly required to deal with the differences that result from content rewrite, the decision as to whether or not back-end TLS is required relies on message-level processing and cannot be determined by the frame/packet-based classifier. Given these considerations, the sequence of SFs 1, 2, and 3 cannot be assembled into an single service function chain. Instead, each of the three hops in the path must be evaluated as independent service function chains, each composed of a single SF.

Similar to the ingress RR/SLB case, the return path manifest and segment processing requires message-level parsing to determine the next hop service function. This cannot be accomplished by the frame/packet-based classifier. Unique to the egress case, in this example, are the split service functions for manifest and segment processing which require independent TCP connections from the back-end TLS service function to the next hop SF for each file received. Multiplexed server response data must be separated and forwarded on the proper TCP connection to support downstream re-classification.

Though a service function chain could be used between the back-end and front-end TLS SFs, front-end TLS session reuse requires correlation between the individual TCP connections from the back-end TLS SF with the associated front-end TLS session. Though it has been suggested that this is a control plane issue, such a suggestion implies that the back-end TLS SF would need to dynamically create a different service function chain for each response file so that their association to the front-end TLS session might be propagated. Such an approach, however, would likely be performance prohibitive. An alternate approach described herein is to use SFC metadata to encapsulate mid-chain flow identifiers for de/re-multiplexing of split-chain data.

2.4. SFC Service Decomposition

In order to take advantage of SFC for the egress path in the example use case described above, two additional pieces of information are required, i.e., an identifier for the front-end TLS session and message identifiers for the individual manifest and segment response files. Each frame/packet sent between the back-end TLS service function and the front-end TLS service function should include:

- Split Chain ID: The decomposed service function chain identifier associated with one of the split paths selected by the demux service function.

- Ingress Flow ID: An identifier provided by the remux service function to the demux service function prior to reverse direction demuxing and split chain delivery.

- Message Sequence Number: An identifier added by the demux service function which associate the frames/packets for a given message, as well as conveys the order in which messages were received by the demux service function.

On the ingress path, the ingress flow ID (associated with the front-end TLS session) needs to be conveyed by the front-end TLS service function and propogated by all service functions along the ingress path. When application layer connection termination or re-classification occurs, the ingress flow ID must either be persistent across the intermediate service function, or reverse path translated and mapped. A reserved service function chain identifier may be used to accompany metadata across re-classifications.

On the ingress path, for a front-end TLS session X, the following table shows the SFC ID, flow ID, and message sequence number, for two HTTP messages (each consisting of multiple TCP/IP packets) traversing service functions 1, 2, and 3. Along each hop, the FLOW_RES SFC ID is used to convey the ingress flow ID X from service function 1 to service function 3.

src SF	dst SF	SFC ID	Flow ID	Mesg Seq
1	2	FLOW_RES	X	1
2	3	FLOW_RES	X	1
1	2	FLOW_RES	X	2
2	3	FLOW_RES	X	2

On the egress path, for front-end TLS session X, the following table table shows the SFC ID, flow ID, and message sequence number, for four HTTP messages (each consisting of multiple TCP/IP packets) traversing service functions 3, 4, 5, and 1. Two of the messages (i.e., 1 and 3) are statefully classified as manifest files for flow X, traversing service functions 3, 4, and 1, represented by SFC ID M1. The other two messages (i.e., 2 and 4) are statefully classified as segment files for flow X, traversing service functions 3, 5, and 1, represented by SFC ID S2. In all cases, the flow ID X is included, corresponding to the ingress HTTP flow ID.

src SF	dst SF	SFC ID	Flow ID	Mesg Seq
3	4	M1	X	1
4	1	M1	X	1
3	5	S2	X	2
5	1	S2	X	2
3	4	M1	X	3
4	1	M1	X	3
3	5	S2	X	4
5	1	S2	X	4

3. IANA Considerations

This memo includes no request to IANA.

4. Security Considerations

TBD.

5. Privacy Considerations

TBD.

6. Acknowledgements

The authors would like to thank Dan Biagini for his helpful review comments.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

7.2. Informative References

[I-D.ietf-httpbis-http2]
Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-14 (work in progress), July 2014.

[I-D.pantos-http-live-streaming]
Pantos, R., "HTTP Live Streaming", draft-pantos-http-live-streaming-13 (work in progress), April 2014.

[I-D.quinn-sfc-arch]
Quinn, P. and J. Halpern, "Service Function Chaining (SFC) Architecture", draft-quinn-sfc-arch-05 (work in progress), May 2014.

[I-D.quinn-sfc-problem-statement]
Quinn, P., "Service Function Chaining Problem Statement", draft-quinn-sfc-problem-statement-02 (work in progress), December 2013.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

Authors' Addresses

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.j.ma@ericsson.com

Ron Parker
Affirmed Networks
35 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-268-0800
Email: ron_parker@affirmednetworks.com