Name-Based Service Function Forwarder (nSFF) component within SFC framework
draft-trossen-sfc-name-based-sff-03

Abstract

   Many stringent requirements are imposed on today's network, such as
   low latency, high availability and reliability in order to support
   several use cases such as IoT, Gaming, Content distribution, Robotics
   etc. Adoption of cloud and fog technology at the edge of the network
   allows operator to deploy a single "Service Function" to multiple
   "Execution locations". The decision to steer traffic to a specific
   location may change frequently based on load, proximity etc. Under
   the current SFC framework, steering traffic dynamically to the
   different execution end points require a specific 're-chaining',
   i.e., a change in the service function path reflecting the different
   IP endpoints to be used for the new execution points. This procedure
   maybe complex and take time. In order to simplify re-chaining and
   reduce the time to complete the procedure, we discuss separating the
   logical Service Function Path from the specific execution end points.
   This can be done by identifying the Service Functions using a name
   rather than a routable IP endpoint (or Layer 2 address). This draft
   describes the necessary extensions, additional functions and protocol
   details in SFF (Service Function Forwarder) to handle name based
   relationships.

Status of This Memo

   This Internet-Draft will expire on August 4, 2019.

Copyright Notice

Table of Contents

1. Introduction

   The requirements on today's networks are very diverse, enabling
   multiple use cases such as IoT, Content Distribution, Gaming and
   Network functions such as Cloud RAN and 5G control planes based on a
   service-based architecture . These network services are deployed,
   provisioned and managed using Cloud based techniques as seen in the
   IT world. Virtualization of compute and storage resources is at the
   heart of providing (often web) services to end users with the ability
   to quickly provisioning such virtualized service endpoints through,
   e.g., container based techniques. This creates a dynamicity with the
   capability to dynamically compose new services from available
   services as well as move a service instance in response to user
   mobility or resource availability where desirable. When moving from
   a pure 'distant cloud' model to one of localized micro data centers
   with regional, metro or even street level, often called 'edge' data
   centers, such virtualized service instances can be instantiated in
   topologically different locations with the overall 'distant' data
   center now being transformed into a network of distributed ones.

   The Service Function Chaining (SFC) framework [RFC7665] allows
   network operators as well as service providers to compose new
   services by chaining individual "Service Functions (SFs)". Such
   chains are expressed through explicit relationships of functional
   components (the service functions), realized through their direct
   Layer 2 (e.g., MAC address) or Layer 3 (e.g., IP address)
   relationship as defined through next hop information that is being
   defined by the network operator, see Section 3 for more background on
   SFC.

   In a dynamic service environment of distributed data centers as the
   one outlined above, with the ability to create and recreate service
   endpoints frequently, the SFC framework requires to reconfigure the
   existing chain through information based on the new relationships,
   causing overhead in a number of components, specifically the
   orchestrator that initiates the initial service function chain and
   any possible reconfiguration.

   This document describes how such changes can be handled without
   involving the initiation of new and reconfigured SFCs by lifting the
   chaining relationship from Layer 2 and 3 information to that of
   service function 'names', such as names for instance being expressed
   as URIs. In order to transparently support such named relationships,
   we propose to embed the necessary functionality directly into the
   Service Function Forwarder (SFF, see [RFC7665]). With that, the SFF
   described in this document allows for keeping an existing SFC intact,
   as described by its service function path (SFP), while enabling the

selection of an appropriate service function endpoint(s) during the
traversal of packets through the SFC.

2. Example use case: 5G control plane services

We exemplify the need for chaining service functions at the level of
a service name through a use case stemming from the current 3GPP Rel
16 work on Service Based Architecture (SBA) [_3GPP_SBA],
[_3GPP_SBA_ENHANCEMENT]. In this work, mobile network control planes
are proposed to be realized by replacing the traditional network
function interfaces with a fully service-based one. HTTP was chosen
as the application layer protocol for exchanging suitable service
requests [_3GPP_SBA]. With this in mind, the exchange between, say
the 3GPP (Rel. 15) defined Session Management Function (SMF) and the
Access and Mobility management Function (AMF) in a 5G control plane
is being described as a set of web service like requests which are in
turn embedded into HTTP requests. Hence, interactions in a 5G
control plane can be modelled based on service function chains where
the relationship is between the specific (IP-based) service function
endpoints that implement the necessary service endpoints in the SMF
and AMF. The service functions are exposed through URIs with work
ongoing to define the used naming conventions for such URIs.

This move from a network function model (in pre-Rel 15 systems of
3GPP) to a service-based model is motivated through the proliferation
of data center operations for mobile network control plane services.
In other words, typical IT-based methods to service provisioning, in
particular that of virtualization of entire compute resources, are
envisioned to being used in future operations of mobile networks.
Hence, operators of such future mobile networks desire to virtualize
service function endpoints and direct (control plane) traffic to the
most appropriate current service instance in the most appropriate
(local) data centre, such data centre envisioned as being
interconnected through a software-defined wide area network (SD-WAN).
'Appropriate' here can be defined by topological or geographical
proximity of the service initiator to the service function endpoint.
Alternatively, network or service instance compute load can be used
to direct a request to a more appropriate (in this case less loaded)
instance to reduce possible latency of the overall request. Such
data center centric operation is extended with the trend towards
regionalization of load through a 'regional office' approach, where
micro data centers provide virtualizable resources that can be used
in the service execution, creating a larger degree of freedom when
choosing the 'most appropriate' service endpoint for a particular
incoming service request.

While the move to a service-based model aligns well with the
framework of SFC, choosing the most appropriate service instance at

runtime requires so-called 're-chaining' of the SFC since the
relationships in said SFC are defined through Layer 2 or 3
identifiers, which in turn are likely to be different if the chosen
service instances reside in different parts of the network (e.g., in
a regional data center).

Hence, when a traffic flow is forwarded over a service chain
expressed as an SFC-compliant Service Function Path (SFP), packets in
the traffic flow are processed by the various service function
instances, with each service function instance applying a service
function prior to forwarding the packets to the next network node.
It is a Service layer concept and can possibly work over any Virtual
network layer and an Underlay network, possibly IP or any Layer 2
technology. At the service layer, Service Functions are identified
using a path identifier and an index. Eventually this index is
translated to an IP address (or MAC address) of the host where the
service function is running. Because of this, any change of service
function instance is likely to require a change of the path
information since either IP address (in the case of changing the
execution from one data centre to another) or MAC address will change
due to the newly selected service function instance.

Returning to our 5G Control plane example, a user's connection
request to access an application server in the internet may start
with signaling in the Control Plane to setup user plane bearers. The
connection request may flow through service functions over a service
chain in the Control plane, as deployed by network operator. Typical
SFs in a 5G control plane may include "RAN termination / processing",
"Slice Selection Function", "AMF" and "SMF". The Classifier, as
described in SFC architecture, may reside in the user terminal or at
the eNB. These service functions can be configured to be part of a
Service Function Chain. We can also say that some of the
configurations of the Service Function Path may change at the
execution time. E.g. SMF may be relocated as user moves and a new
SMF may be included in the Service Function Path based on user
location. The following diagram in Figure 1 shows the example
Service Function Chain described here.

```
        +------+   +---------+  +-----+   +-----+
        | User |   | Slice   |  |     |   |     |     |
        | App  |-->| Control |->| AMF |-->| SMF |-->
        | Fn   |   | Function|  |     |   |     |     |
        +------+   +---------+  +-----+   +-----+
```
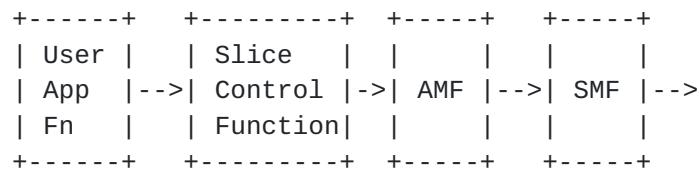
Figure 1: Mapping SFC onto Service Function Execution Points along a
                      Service Function Path

3. Background

   [RFC7665] describes an architecture for the specification, creation,
   and ongoing maintenance of Service Function Chains (SFCs). It
   includes architectural concepts, principles, and components used in
   the construction of composite services through deployment of SFCs.
   In the following, we outline the parts of this SFC architecture
   relevant for our proposed extension, followed by the challenges with
   this current framework in the light of our example use case.

3.1. Relevant part of SFC architecture

   SFC Architecture [RFC7665], describes architectural components such
   as Service Function (SF), Classifier, and Service Function Forwarder
   (SFF). It describes the Service Function Path (SFP) as the logical
   path of an SFC. Forwarding traffic along such SFP is the
   responsibility of the SFF. For this, the SFFs in a network maintain
   the requisite SFP forwarding information. Such SFP forwarding
   information is associated with a service path identifier (SPI) that
   is used to uniquely identify an SFP. The service forwarding state is
   represented by the Service Index (SI) and enables an SFF to identify
   which SFs of a given SFP should be applied, and in what order. The
   SFF also has information that allows it to forward packets to the
   next SFF after applying local service functions.

   The operational steps to forward traffic are then as follows: Traffic
   arrives at an SFF from the network. The SFF determines the
   appropriate SF the traffic should be forwarded to via information
   contained in the SFC encapsulation. After SF processing, the traffic
   is returned to the SFF, and, if needed, is forwarded to another SF
   associated with that SFF. If there is another non-local hop (i.e.,
   to an SF with a different SFF) in the SFP, the SFF further
   encapsulates the traffic in the appropriate network transport
   protocol and delivers it to the network for delivery to the next SFF
   along the path. Related to this forwarding responsibility, an SFF
   should be able to interact with metadata.

3.2. Challenges with current framework

   As outlined in previous section, the Service Function Path defines an
   ordered sequence of specific Service Functions instances being used
   for the interaction between initiator and service functions along the
   SFP. These service functions are addressed by IP (or any L2/MAC)
   addresses and defined as next hop information in the network locator
   maps of traversing SFF nodes.

   As outlined in our use case, however, the service provider may want
   to provision SFC nodes based on dynamically spun up service function
   instances so that these (now virtualized) service functions can be
   reached in the SFC domain using the SFC underlay layer.

   Following the original model of SFC, any change in a specific
   execution point for a specific Service Function along the SFP will
   require a change of the SFP information (since the new service
   function execution point likely carries different IP or L2 address
   information) and possibly even the Next Hop information in SFFs along
   the SFP. In case the availability of new service function instances
   is rather dynamic (e.g., through the use of container-based
   virtualization techniques), the current model and realization of SFC
   could lead to reducing the flexibility of service providers and
   increasing the management complexity incurred by the frequent changes
   of (service) forwarding information in the respective SFF nodes.
   This is because any change of the SFP (and possibly next hop info)
   will need to go through suitable management cycles.

   To address these challenges through a suitable solution, we identify
   the following requirements:

   o Relations between Service Execution Points MUST be abstracted so
      that, from an SFP point of view, the Logical Path never changes.

   o Deriving the Service Execution Points from the abstract SFP SHOULD
      be fast and incur minimum delay.

   o Identification of the Service Execution Points SHOULD not use a
      combination of Layer 2 or Layer 3 mechanisms.

   The next section outlines a solution to address the issue, allowing
   for keeping SFC information (represented in its SFP) intact while
   addressing the desired flexibility of the service provider.

4. **Name based operation in SFF**

4.1. **General Idea**

   The general idea is two-pronged. Firstly, we elevate the definition
   of a Service Function Path onto the level of 'name-based
   interactions' rather than limiting SFPs to Layer 3 or Layer 2
   information only. Secondly, we extend the operations of the SFF to
   allow for forwarding decisions that take into account such name-based
   interaction while remaining backward compatible to the current SFC
   architecture [RFC7665]. In the following sections, we outline these
   two components of our solution.

   If the next hop information in the Network Locator Map is described
   using L2/L3 identifier, the nSFF MAY operate as described in
   [RFC7665]. On the other hand, if the next hop information in the NLM
   is described as a name, then the nSFF operates as described in the
   following sections.

   In the following sections, we outline the two components of our
   solution.

4.2. **Name-Based Service Function Path (nSFP)**

   In the existing SFC framework [RFC7665], as outlined in Section 3,
   the SFP information is representing path information based on Layer 2
   or 3 information, i.e., MAC or IP addresses , causing the
   aforementioned frequent adaptations in cases of execution point
   changes. Instead, we introduce the notion of a 'name-based service
   function path (nSFP)'

   In today's networking terms, any identifier can be treated as a name
   but we will illustrate the realization of a "Name based SFP" through
   extended SFF operations (see Section 5) based on URIs as names and
   HTTP as the protocol of exchanging information . Here, URIs are being
   used to name for a Service Function along the nSFP. It is to be
   noted that the Name based SFP approach is not restricted to HTTP (as
   the protocol) and URIs (as next hop identifier within the SFP).
   Other identifiers such as an IP address itself can also be used and
   are interpreted as a 'name' in the nSFP . With this, our notion of
   the nSFP goes beyond the initial proposals made in
   [I-D.purkayastha-sfc-service-indirection], which limited the notion
   of a 'name' to a URL (uniform resource locator), commonly used in the
   addressing of HTTP requests. In other words, IP addresses as well as
   fully qualified domain names forming complex URIs (uniform resource
   identifiers), such as www.foo.com/service_name1, are all captured by
   the notion of 'name' in this draft.

Generally, nSFPs are defined as an ordered sequence of the "name" of
Service Functions (SF) and a typical name-based Service Function Path
may look like: 192.168.x.x -> www.foo.com -> www.foo2.com/service1 ->
www.foo2.com/service2

Our use case in Section 2 can then be represented as an ordered named
sequence. An example for a session initiation that involves an
authentication procedure, this could look like 192.168.x.x ->
smf.3gpp.org/session_initiate -> amf.3gpp.org/auth -> smf.3gpp.org/
session_complete -> 192.168.x.x [Note that this example is only a
conceptual one, since the exact nature of any future SBA-based
exchange of 5G control plane functions is yet to be defined by
standardization bodies such as 3GPP]

In accordance with our use case in Section 2, any of these named
services can potentially be realized through more than one replicated
SF instances. This leads to make dynamic decision on where to send
packets along the SAME service function path information, being
provided during the execution of the SFC. Through elevating the SFP
onto the notion of name-based interactions, the SFP will remain the
same even if those specific execution points change for a specific
service interaction.

The following diagram in Figure 2, describes this name-based SFP
concept and the resulting mapping of those named interactions onto
(possibly) replicated instances.

```
  +-------------------------------------------------------------+
  |SERVICE LAYER                                                |
  | 192.168.x.x --> www.foo.com --> www.foo2.com --> www.fooN.com |
  |                     ||               ||                      |
  +---------------------||---------------||----------------------+
                        ||               ||
                        ||               ||
  +---------------------||---------------||----------------------+
  | Underlay network    \/               \/                     |
  |             +--+ +--+ +--+    +--+ +--+ +--+                 |
  |             |  | |  | |  |    |  | |  | |  |                 |
  |             +--+ +--+ +--+    +--+ +--+ +--+                 |
  |             Compute and       Compute and                   |
  |             storage nodes     storage nodes                 |
  +-------------------------------------------------------------+
```
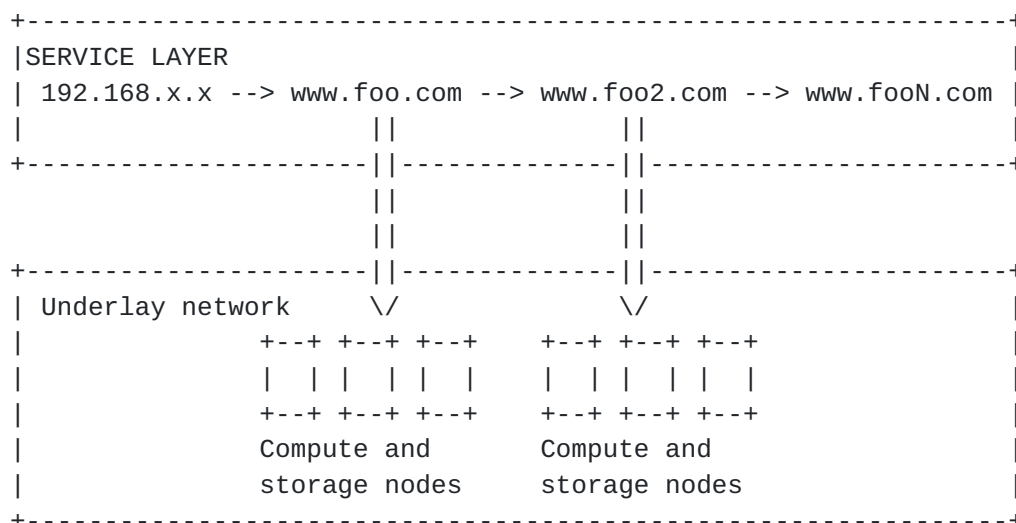
Figure 2: Mapping SFC onto Service Function Execution Points along a
Service Function Path based on Virtualized Service Function Instance

4.3. Name Based Network Locator Map (nNLM)

   In order to forward a packet within a name-based SFP, we need to
   extend the network locator map as originally defined in [RFC8300]
   with the ability to consider name relations based on URIs as well as
   high-level transport protocols such as HTTP for means of SFC packet
   forwarding. Another example for SFC packet forwarding could be that
   of CoAP.

   The extended Network Locator Map or name-based Network Locator Map
   (nNLM) is shown in Figure 3 as an example for www.foo.com being part
   of the nSFP. Such extended nNLM is stored at each SFF throughout the
   SFC domain with suitable information populated to the nNLM during the
   configuration phase

```
     +------+------+--------------------+------------------------+
     | SPI  | SI   | Next Hop(s)        | Transport Encapsulation |
     +------+------+--------------------+------------------------+
     | 10   | 255  | 192.0.2.1          | VXLAN-gpe              |
     |      |      |                    |                        |
     | 10   | 254  | 198.51.100.10      | GRE                    |
     |      |      |                    |                        |
     | 10   | 253  | www.foo.com        | HTTP                   |
       ------------------------------------------------------------
     |      |      |                    |                        |
     | 40   | 251  | 198.51.100.15      | GRE                    |
     |      |      |                    |                        |
     | 50   | 200  | 01:23:45:67:89:ab  | Ethernet               |
     |      |      |                    |                        |
     | 15   | 212  | Null (end of path) | None                   |
     +------+------+--------------------+------------------------+
```

                   Figure 3: Name-based Network Locator Map

   Alternatively, the extended network locator map may be defined with
   implicit name information rather than explicit URIs as in Figure 3.
   In the example of Figure 4 below, the next hop is represented as a
   generic HTTP service without a specific URI being identified in the
   extended network locator map. In this scenario, the SFF forwards the
   packet based on parsing the HTTP request in order to identify the
   host name or URI. It retrieves the URI and may apply policy
   information to determine the destination host/service.

```
+------+------+--------------------+------------------------+
| SPI  | SI   | Next Hop(s)        | Transport Encapsulation |
+------+------+--------------------+------------------------+
| 10   | 255  | 192.0.2.1          | VXLAN-gpe              |
|      |      |                    |                        |
| 10   | 254  | 198.51.100.10      | GRE                    |
|      |      |                    |                        |
| 10   | 253  | HTTP Service       | HTTP                   |
  ---------------------------------------------------------
|      |      |                    |                        |
| 40   | 251  | 198.51.100.15      | GRE                    |
|      |      |                    |                        |
| 50   | 200  | 01:23:45:67:89:ab  | Ethernet               |
|      |      |                    |                        |
| 15   | 212  | Null (end of path) | None                   |
+------+------+--------------------+------------------------+
```

Figure 4: Name-based Network Locator Map with Implicit Name
information

4.4. Name-based Service Function Forwarder (nSFF)

While [I-D.purkayastha-sfc-service-indirection] outlined the
realization of forwarding packets in URL-based interaction through
HTTP via a specific function (called Service Request Routing (SRR) in
[I-D.purkayastha-sfc-service-indirection]), it is desirable to
extend the SFF of the SFC underlay in order to handle nSFPs
transparently and without the need to insert a special (SRR) service
function into the nSFP. Such extended name-based SFF would then be
responsible for forwarding a packet in the SFC domain as per the
definition of the (extended) nSFP.

In our exemplary realization for an extended SFF, the solution
described in this document uses HTTP as the protocol of forwarding
SFC packets to the next (name-based) hop in the nSFP. The URI in the
HTTP transaction are the names in our nSFP information, which will be
used for name based forwarding.

Following our reasoning so far, HTTP requests (and more specifically
the plain text encoded requests above) are the equivalent of Packets
that enter the SFC domain. In the existing SFC framework, typically
an IP payload is assumed to be a packet entering the SFC domain.
This packet is forwarded to destination nodes using the L2
encapsulation. Any layer 2 network can be used as an underlay
network. This notion is now extended to packets being possibly part
of a entire higher layer application, such as HTTP requests. The
handling of any intermediate layers such as TCP, IP is left to the

realization of the (extended) SFF operations towards the next (named)
hop. For this, we will first outline the general lifecycle of an SFC
packet in the following subsection, followed by two examples for
determining next hop information in Section 5.2.3, finalized by a
layered view on the realization of the nSFF in Section 5.2.4

4.5. High Level Architecture

```
+----------+
| SF1      |                    +--------+                    +------+
| instance |\                   |   NR   |                    | SF2  |
+----------+ \                  +--------+                    +------+
              \                      ||                           ||
+------------+ \ +-------+    +---------+   +---------+   +-------+
| Classifier |---| nSFF1 |---|Forwarder|---|Forwarder|---| nSFF2 |
+------------+   +-------+    +---------+   +---------+   +-------+
                                                            ||
                                                       +----------+
                                                       | Boundary |
                                                       |  node    |
                                                       +----------+
```
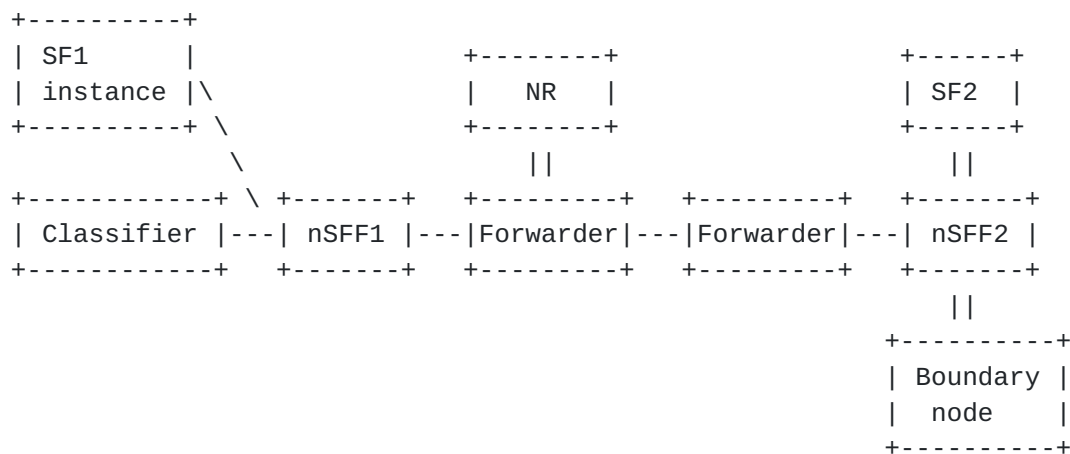
Figure 5: High-level architecture

The high-level architecture for name based operation shown in
Figure 5 is very similar to the SFC architecture, as described in
[RFC7665]. Two new functions are introduced, as shown in the above
diagram, namely the name-based Service Function Forwarder (nSFF) and
the Name Resolver (NR).

nSFF (name-based Service Function Forwarder) is an extension of the
existing SFF and is capable of processing SFC packets based on name-
based network locator map (nNLM) information, determining the next
SF, where the packet should be forwarded and the required transport
encapsulation. Like standard SFF operation, it adds transport
encapsulation to the SFC packet and forwards it.

The Name Resolver is a new functional component, capable of
identifying the execution end points, where a "named SF" is running,
triggered by suitable resolution requests sent by the nSFF. Though
this is similar to DNS function, but it is not same. It does not use
DNS protocols or data records. A new procedure to determine the
suitable routing/forwarding information towards the Nsff (Named SFF)

serving the next hop of the SFP (Serving Function Path) is used. The
details is described later.

The other functional components such as Classifier, SF are same as
described in SFC architecture [RFC7665], while the Forwarders shown
in the above diagram are traditional Layer 2 switches.

4.6. Operational Steps

In the proposed solution, the operations are realized by the name-
based SFF, called nSFF. We utilize the high-level architecture in
Figure 5 to describe the traversal between two service function
instances of an nSFP-based transactions in an example chain of :
192.168.x.x -> SF1 (www.foo.com) -> SF2 (www.foo2.com) -> SF3 -> ...

According to the SFC lifecycle [RFC7665] and based on our example
chain above, the traffic originates from a Classifier or another SFF
on the left. The traffic is processed by the incoming nSFF1 (on the
left side) through the following steps. The traffic exits at nSFF2.

o Step 1: At nSFF1 the following nNLM is assumed

```
+------+------+--------------------+------------------------+
| SPI  | SI   | Next Hop(s)        | Transport Encapsulation |
+------+------+--------------------+------------------------+
| 10   | 255  | 192.0.2.1          | VXLAN-gpe              |
|      |      |                    |                        |
| 10   | 254  | 198.51.100.10      | GRE                    |
|      |      |                    |                        |
| 10   | 253  | www.foo.com        | HTTP                   |
|      |      |                    |                        |
| 10   | 252  | www.foo2.com       | HTTP                   |
|      |      |                    |                        |
| 40   | 251  | 198.51.100.15      | GRE                    |
|      |      |                    |                        |
| 50   | 200  | 01:23:45:67:89:ab  | Ethernet               |
|      |      |                    |                        |
| 15   | 212  | Null (end of path) | None                   |
+------+------+--------------------+------------------------+
```

Figure 6: nNLM at nSFF1

o Step 2: nSFF1 removes the previous transport encapsulation (TE)
   for any traffic originating from another SFF or classifier
   (traffic from an SF instance does not carry any TE and is
   therefore directly processed at the nSFF).

o Step 3: nSFF1 then processes the NSH information to identify the
   next SF at the nSFP level by mapping the NSH information to the
   appropriate entry in its nNLM (see Figure 6) based on the provided
   SPI/SI information in the NSH (see Section 3) in order to
   determine the name-based identifier of the next hop SF. With such
   nNLM in mind, the nSFF searches the map for SPI = 10 and SI = 253.
   It identifies the next hop as = www.foo.com and HTTP as the
   protocol to be used. Given the next hop resides locally , the SFC
   packet is forwarded to the SF1 instance of www.foo.com. Note that
   the next hop could also be identified from the provided HTTP
   request, if the next hop information was identified as a generic
   HTTP service, as defined in Section 5.3.

o Step 4: The SF1 instance then processes the received SFC packet
   according to its service semantics and modifies the NSH by setting
   SPI = 10, SI = 252 for forwarding the packet along the SFP. It
   then forwards the SFC packet to its local nSFF, i.e., nSFF1.

o Step 5: nSSF1 processes the NSH of the SFC packet again, now with
   the NSH modified (SPI = 10, SI = 252) by the SF1 instance. It
   retrieves the next hop information from its nNLM in Figure 6, to
   be www.foo2.com. Due to this SF not being locally available , the
   nSFF consults any locally available information regarding routing/
   forwarding towards a suitable nSFF that can serve this next hop.

o Step 6: If such information exists, the Packet (plus the NSH
   information) is marked to be sent towards the nSFF serving the
   next hop based on such information in step 8.

o Step 7: If such information does not exist, nSFF1 consults the
   Name Resolver (NR) to determine the suitable routing/forwarding
   information towards the identified nSFF serving the next hop of
   the SFP. For future SFC packets towards this next hop, such
   resolved information may be locally cached , avoiding to contact
   the Name Resolver for every SFC packet forwarding. The packet is
   now marked to be sent via the network in step 8.

o Step 8: Utilizing the forwarding information determined in steps 6
   or 7, nSFF1 adds the suitable transport encapsulation (TE) for the
   SFC packet before forwarding via the forwarders in the network
   towards the next nSFF22.

o Step 9: When the Packet (+NSH+TE) arrives at the outgoing nSFF2,
   i.e., the nSFF serving the identified next hop of the SFP, removes
   the TE and processes the NSH to identify the next hop information.
   At nSFF2 the nNLM in Figure 7 is assumed. Based on this nNLM and
   NSH information where SPI = 10 and SI = 252, nSFF2 identifies the
   next SF as www.foo2.com.

```
+------+------+--------------------+-------------------------+
| SPI  | SI   | Next Hop(s)        | Transport Encapsulation |
+------+------+--------------------+-------------------------+
|      |      |                    |                         |
| 10   | 252  | www.foo2.com       | HTTP                    |
|      |      |                    |                         |
| 40   | 251  | 198.51.100.15      | GRE                     |
|      |      |                    |                         |
| 50   | 200  | 01:23:45:67:89:ab  | Ethernet                |
|      |      |                    |                         |
| 15   | 212  | Null (end of path) | None                    |
+------+------+--------------------+-------------------------+
```

Figure 7: nNLM at SFF2

o Step 10: If the next hop is locally registered at the nSFF, it
   forwards the packet (+NSH) to the service function instance, using
   suitable IP/MAC methods for doing so.

o Step 11: Otherwise, the outgoing nSFF adds a new TE information to
   the packet and forwards the packet (+NSH+TE) to the next SFF or
   boundary node, as shown in Figure 7.

5. nSFF Forwarding Operations

This section outlines the realization of various nSFF forwarding
operations in Section 4.6. Although the operations in Section 4
utilize the notion of name-based transactions in general, we
exemplify the operations here in Section 5 specifically for HTTP-
based transactions to ground our description into a specific protocol
for such name-based transaction. We will refer to the various steps
in each of the following sub-sections.

5.1. nSFF Protocol Layers

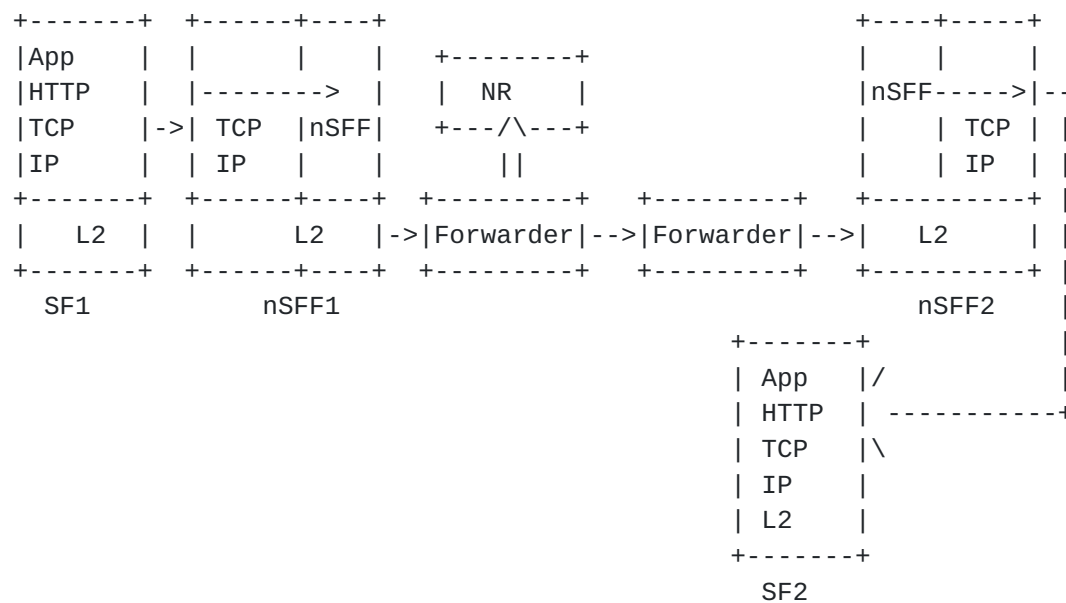Figure 8 shows the protocol layers, based on the high-level
architecture in Figure 5.

```
  +-------+  +------+----+                              +----+-----+
  |App    |  |      |    |   +--------+                 |    |     |
  |HTTP   |  |------->   |   |   NR   |                 |nSFF----->|--
  |TCP    |->|  TCP  |nSFF|   +---/\---+                 |    | TCP | |
  |IP     |  |  IP   |    |        ||                   |    | IP  | |
  +-------+  +------+----+   +---------+   +---------+   +----------+ |
  |  L2   |  |     L2    |->|Forwarder|-->|Forwarder|-->|    L2    | |
  +-------+  +------+----+   +---------+   +---------+   +----------+ |
    SF1            nSFF1                                    nSFF2     |
                                          +-------+                  |
                                          |  App  |/                 |
                                          |  HTTP | -----------+
                                          |  TCP  |\
                                          |  IP   |
                                          |  L2   |
                                          +-------+
                                            SF2
```

Figure 8: Protocol layers

The nSFF component here is shown as implementing a full incoming/
outgoing TCP/IP protocol stack towards the local service functions,
while implementing the nSFF-NR and nSFF-nSFF protocols based on the
descriptions in Section 5.2.3.

For the exchange of HTTP-based service function transactions, the
nSFF terminates incoming TCP connections from as well as outgoing TCP
connections to local SFs, e.g., the TCP connection from SF1
terminates at nSFF1, and nSFF1 may store the connection information,
such as socket information. It also maintains the mapping
information for the HTTP request such as originating SF, destination
SF and socket ID. nSFF1 may implement sending keep-alive messages
over the socket to maintain the connection to SF1. Upon arrival of
an HTTP request from SF1, nSFF1 extracts the HTTP Request and
forwards it towards the next node, as outlined in Section 5.2. Any
returning response is mapped onto the suitable open socket (for the
original request) and send towards SF1.

At the outgoing nSFF2, the destination SF2/Host is identified from
the HTTP request message. If no TCP connection exists to the SF2, a
new TCP connection is opened towards the destination SF2 and the HTTP
request is sent over said TCP connection. The nSFF2 may also save
the TCP connection information (such as socket information) and
maintain the mapping of the socket information to the destination
SF2. When an HTTP response is received from SF2 over the TCP
connection, nSFF2 extracts the HTTP response, which is forwarded to

the next node. nSFF2 may maintain the TCP connection through keep-
alive messages.

5.2. nSFF Operations

In this section, we present three key aspects of operations for the
realization of the steps in Section 4.6, namely (i) the registration
of local SFs (for step 3 in Section 4.6), (ii) the forwarding of SFC
packets to and from local SFs (for step 3 and 4 as well as 10 in
Section 4.6), (iii) the forwarding to a remote SF (for steps 5, 6.,
and 7 in Section 4.6) and to the NR as well as (iv) for the lookup of
a suitable remote SF (for step 7 in Section 4.6). We also cover
aspects of maintaining local lookup information for reducing lookup
latency and others issues.

5.2.1. Forwarding between nSFFs and nSFF-NR

Forwarding between the distributed nSFFs as well as between nSFF and
NR is realized over the operator network via a path-based approach.
A path-based approach utilizes path information provided by the
source of the packet for forwarding said packet in the network. This
is similar to segment routing albeit differing in the type of
information provided for such source-based forwarding, as described
in this section. In this approach, the forwarding information to a
remote nSFF or the NR is defined as a 'path identifier' (pathID) of a
defined length where said length indicate the overall pathID length
as the 2 to the power of 'length', i.e., maximum 2^16 bits as path
information. The payload of the packet is defined by the various
operations outlined in the following sub-sections, resulting in an
overall packet being transmitted. With this, the generic forwarding
format (GFF) for transport over the operator network is defined in
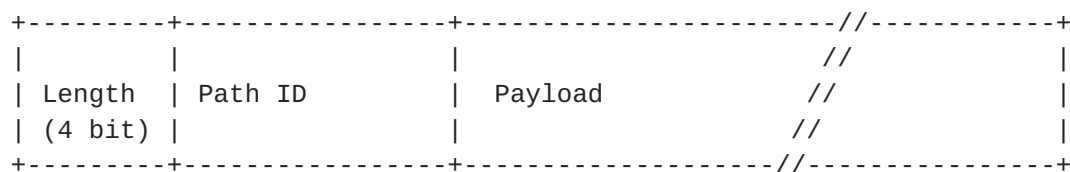Figure 9 with the length field defining the length of the pathID
provided.

```
+---------+----------------+----------------------//------------+
|         |                |                      //            |
| Length  | Path ID        | Payload              //            |
| (4 bit) |                |                      //            |
+---------+----------------+-------------------//---------------+
```

Figure 9: Generic Forwarding Format(GFF)

For the pathID information, solutions such as those in [Reed2016] can
be used. Here, the IPv6 source and destination addresses are used to
realize a so-called path-based forwarding from the incoming to the

outgoing nSFF or the NR. The forwarders in Figure 8 are realized via
SDN (software-defined networking) switches, implementing an AND/CMP
operation based on arbitrary wildcard matching over the IPv6 source
and destination addresses, as outlined in [Reed2016]. Note that in
the case of using IPv6 address information for path-based forwarding,
the step of removing the transport encapsulation at the outgoing nSFF
in Figure 8 is realized by utilizing the provided (existing) IP
header (which was used for the purpose of the path-based forwarding
in [Reed2016]) for the purpose of next hop forwarding, such as that
of IP-based routing. As described in step 8 of the extended nSFF
operations, this forwarding information is used as traffic
encapsulation. With the forwarding information utilizing existing
IPv6 information, IP headers are utilized as TE in this case. The
next hop nSFF (see Figure 8) will restore the IP header of the packet
with the relevant IP information used to forward the SFC packet to
SF2 or it will create a suitable TE (Transport Encapsulation)
information to forward the information to another nSFF or boundary
node. Forwarding operations at the intermediary forwarders, i.e.,
SDN switches, examine the pathID information through a flow matching
rule in which a specific switch-local output port is represented
through the specific assigned bit position in the pathID. Upon a
positive match in said rule, the packet is forwarded on said output
port.

Alternatively, the solution in [I-D.purkayastha-bier-multicast-http]
suggests using a so-called BIER (Binary Indexed Explicit Replication)
underlay. Here, the nSFF would be realized at the ingress to the
BIER underlay, injecting the SFC packet (plus the NSH) header with
BIER-based traffic encapsulation into the BIER underlay with each of
the forwarders in Figure 8 being realized as a so-called Bit-
Forwarding Router (BFR) [RFC8279].

5.2.1.1. Transport Protocol Considerations

Given that the proposed solution operates at the 'named transaction'
level, particularly for HTTP transactions, forwarding between nSFFs
and/or NR SHOULD be implemented via a transport protocol between
nSFFs and/or NR in order to provide reliability, segmentation of
large GFF packets, and flow control. The details of this protocol
will be outlined at a later stage, with the GFF in Figure 9 being the
basic forwarding format for this.

5.2.2. SF Registration

As outlined in step 3 and 10 of Section 4.6, the nSFF needs to
determine if the SF derived from the nNLM is locally reachable or
whether the packet needs forwarding to a remote SFF. For this, a

registration mechanism is provided for such local SF with the local
nSFF. Two mechanisms can be used for this:

1. SF-initiated: We assume that the SF registers its FQDN to the
local nSFF. As local mechanisms, we foresee that either a REST-based
interface over the link-local link or configuration of the nSFF
(through configuration files or management consoles) can be utilized.
Such local registration event leads to the nSFF to register the given
FQDN with the NR in combination with a system-unique nSFF identifier
that is being used for path computation purposes in the NR. For the
registration, the packet format in Figure 10 is used (inserted as the
payload in the GFF of Figure 9 with the pathID towards the NR), with
a hash over the FQDN being conveyed in the registration and the R/D
bit set to 0 (for registration). We assume that the pathID towards
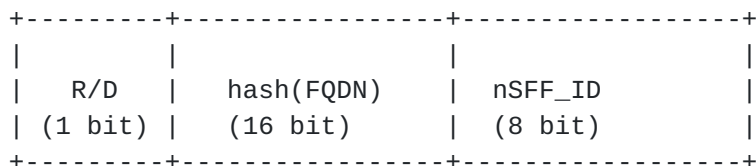the NR is known to the nSFF through configuration means.

```
        +---------+-----------------+------------------+
        |         |                 |                  |
        |   R/D   |   hash(FQDN)    |   nSFF_ID        |
        | (1 bit) |    (16 bit)     |   (8 bit)        |
        +---------+-----------------+------------------+
```

Figure 10: Registration packet format

The NR maintains an internal table that associates the hash(FQDN),
the nSFF_id information as well as the pathID information being used
for communication between nSFF and NR. The nSFF locally maintains a
mapping of registered FQDNs to IP addresses, for the latter using
link-local private IP addresses.

2. Orchestration-based: in this mechanism, we assume that SFC to be
orchestrated and the chain being provided through an orchestration
template with FQDN information associated to a compute/storage
resource that is being deployed by the orchestrator. We also assume
knowledge at the orchestrator of the resource topology. Based on
this, the orchestrator can now use the same REST-based protocol
defined in option 1 to instruct the NR to register the given FQDN, as
provided in the template, at the nSFF it has identified as being the
locally servicing nSFF, provided as the system-unique nSFF
identifier.

5.2.3. Local SF Forwarding

There are two cases of local SF forwarding, namely the SF sending an
SFC packet to the local nSFF (incoming requests) or the nSFF sending
a packet to the SF (outgoing requests) as part of steps 3 and 10 in

Section 4.6. In the following, we outline the operation for HTTP as
an example named transaction.

As shown in Figure 8, incoming HTTP requests from SFs are extracted
by terminating the incoming TCP connection at their local nSFFs at
the TCP level. The nSFF MUST maintain a mapping of open TCP sockets
to HTTP requests (utilizing the URI of the request) for HTTP response
association.

For outgoing HTTP requests, the nSFF utilizes the maintained mapping
of locally registered FQDNs to link-local IP addresses (see
Section 5.2.2 option 1). Hence, upon receiving an SFC packet from a
remote nSFF (in step 9 of Section 4.6), the nSFF determines the local
existence of the SF through the registration mechanisms in
Section 5.2.2. If said SF does exist locally, the HTTP (+NSH)
packet, after stripping the TE, is sent to the local SF as step 10 in
Section 4.6 via a TCP-level connection. Outgoing nSFF SHOULD keep
TCP connections open to local SFs for improving SFC packet delivery
in subsequent transactions.

5.2.4. Handling of HTTP responses

When executing step 3 and 10 in Section 4.6, the SFC packet will be
delivered to the locally registered next hop. As part of the HTTP
protocol, responses to the HTTP request will need to be delivered on
the return path to the originating nSFF (i.e. the previous hop). For
this, the nSFF maintains a list of link-local connection information,
e.g., sockets to the local SF and the pathID on which the request was
received. Once receiving the response, nSFF consults the table to
determine the pathID of the original request, forming a suitable GFF-
based packet to be returned to the previous nSFF.

When receiving the HTTP response at the previous nSFF, the nSFF
consults the table of (locally) open sockets to determine the
suitable local SF connection, mapping the received HTTP response URI
to the stored request URI. Utilizing the found socket, the HTTP
response is forwarded to the locally registered SF.

To be added later: Handling of semi-synchronous responses from
different chains to the same SF (with multicast capability)
[I-D.purkayastha-bier-multicast-http].

5.2.5. Remote SF Forwarding

In steps 5, 6, 7, and 8 of Section 4.6, an SFC packet is forwarded to
a remote nSFF based on the nNLM information for the next hop of the
nSFP. Section 5.2.4.1 handles the case of suitable forwarding
information to the remote nSFF not existing, therefore consulting the

NR to obtain suitable information, while Section 5.2.4.2 describes
the maintenance of forwarding information at the local nSFF, while
Section 5.2.4.3 describes the update of stale forwarding information.
Note that the forwarding described in Section 5.2.1 is used for the
actual forwarding to the various nSFF components. Ultimately,
Section 5.2.4.4 describes the forwarding to the remote nSFF via the
forwarder network

5.2.5.1. Remote SF Discovery

The nSFF communicates with the NR for two purposes, namely the
registration and discovery of FQDNs. The packet format for the
former was shown in Figure 10 in Section 5.2.2, while Figure 11
outlines the packet format for the discovery request. A path to a
specific FQDN is requested by sending a hash of the FQDN to the NR
together with its nSFF_id, receiving as a response a pathID with a
length identifier. The NR should maintain a table of discovery
requests that map discovered (hash of) FQDN to the nSFF_id that
requested it and the pathID that is being calculated as a result of
the discovery request.

The discovery request for an FQDN that has not previously been served
at the nSFF (or for an FQDN whose pathID information has been flushed
as a result of the update operations in Section 5.2.4.3), results in
an initial latency incurred by this discovery through the NR, while
any SFC packet sent over the same SFP in a subsequent transaction
will utilize the nSFF local mapping table. Such initial latency can
be avoided by pre-populating the FQDN-pathID mapping proactively as
part of the overall orchestration procedure, e.g., alongside the
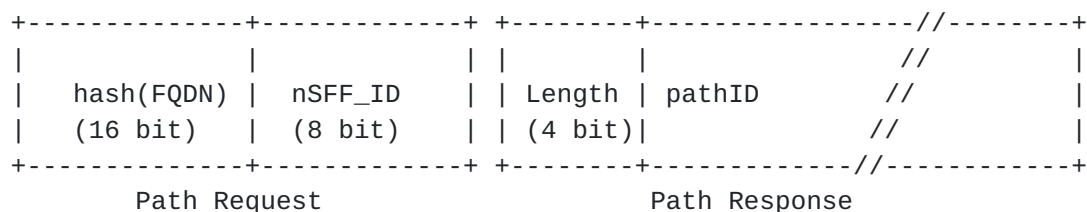distribution of the nNLM information to the nSFF.

```
+--------------+-------------+ +--------+----------------//--------+
|              |             | |        |                //        |
|   hash(FQDN) |   nSFF_ID   | | Length | pathID       //          |
|   (16 bit)   |   (8 bit)   | | (4 bit)|              //          |
+--------------+-------------+ +--------+------------//-----------+
       Path Request                   Path Response
```

Figure 11: Discovery packet format

5.2.5.2. Maintaining Forwarding Information at Local nSFF

Each nSFF MUST maintain an internal table that maps the (hash of the)
FQDN information to a suitable pathID information. As outlined in
step 7 of Section 4.6, if a suitable entry does not exist for a given
FQDN, the pathID information is requested with the operations in

Section 5.2.4.1 and the suitable entry is locally created upon
receiving a reply with the forwarding operation being executed as
described in Section 5.2.1.

If such entry does exist (i.e., step 6 of Section 4.6) the pathID is
locally retrieved and used for the forwarding operation in
Section 5.2.1.

5.2.5.3. Updating Forwarding Information at nSFF

The forwarding information maintained at each nSFF (see
Section 5.2.4.2) might need to be updated for three reasons:

o An existing SF is no longer reachable: In this case, the nSFF with
   which the SF is locally registered, deregisters the SF explicitly
   at the NR by sending the packet in Figure 10 with the hashed FQDN
   and the R/D bit set to 1 (for deregister).

o Another SF instance has become reachable in the network (and
   therefore might provide a better alternative to the existing SF):
   in this case, the NR has received another packet with format
   defined in Figure 11 but a different nSFF_id value.

o Links along paths might no longer be reachable: the NR might use
   suitable southbound interface to transport networks to detect link
   failures, which it associates to the appropriate pathID bit
   position

For this purpose, the packet format in Figure 12 is sent from the NR
to all affected nSFFs, using the generic format in Figure 9. The
pathID to the affected nSFFs is computed as the binary OR over all
pathIDs to those nSFF_ids affected where the pathID information to
the affected nSFF_id values is determined from the NR-local table
maintained in the registration/deregistration operation of
Section 5.2.2.

```
        +---------+----------------+--------------//----+
        |         |                |              //    |
        |  Type   |     #IDs       |  IDs     //        |
        | (1 bit) |    (8 bit)     |          //        |
        +---------+----------------+----------//--------+
```
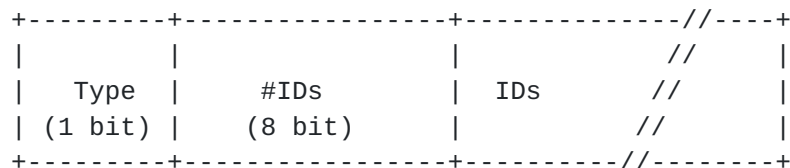
Figure 12: Path update format

The pathID may include the type of information being updated (e.g.,
node identifiers of leaf nodes or link identifiers for removed

links). The node identifier itself may be a special identifier to
signal "ALL NODES" as being affected. The node identifier may signal
changes to the network that are substantial (e.g., parallel link
failures). The node identifier may trigger (e.g., recommend) purging
of the entire path table (e.g., rather than the selective removal of
a few nodes only).

It will include the information according to the type. The included
information may also be related to the type and length information
for the number of identifiers being provided.

In case 1 and 2, the Type bit is set to 1 (type nSFF_id) and the
affected nSFFs are determined by those nSFFs that have previously
sent SF discovery requests, utilizing the optional table mapping
previously registered FQDNs to nSFF_id values. If no table mapping
the (hash of) FQDN to nSFF_id is maintained, the update is sent to
all nSFFs. Upon receiving the path update at the affected nSFF, all
appropriate nSFF-local mapping entries to pathIDs for the hash(FQDN)
identifiers provided will be removed, leading to a new NR discovery
request at the next remote nSFF forwarding to the appropriate FQDN.

In case 3, the Type bit is set to 0 (type linkID) and the affected
nSFFs are determined by those nSFFs whose discovery requests have
previously resulted in pathIDs which include the affected link,
utilizing the optional table mapping previously registered FQDNs to
pathID values (see Section 5.2.4.1). Upon receiving the node
identifier information in the path update, the affected nSFF will
check its internal table that maps FQDNs to pathIDs to determine
those pathIDs affected by the link problems and remove path
information that includes the received node identifier(s). For this,
the pathID entries of said table are checked against the linkID
values provided in the ID entry of the path update through a binary
AND/CMP operation to check the inclusion of the link in the pathIDs
to the FQDNs. If any pathID is affected, the FQDN-pathID entry is
removed, leading to a new NR discovery request at the next remote
nSFF forwarding to the appropriate FQDN.

5.2.5.4. Forwarding to remote nSFF

Once step 5, 6, and 7 in Section 4.6 are being executed, step 8
finally sends the SFC packet to the remote nSFF, utilizing the pathID
returned in the discovery request (Section 5.2.4.1) or retrieved from
the local pathID mapping table. The SFC packet is placed in the
payload of the generic forwarding format in Figure 14 together with
the pathID and the nSFF eventually executes the forwarding operations
in Section 5.2.1.

6. IANA Considerations

This document requests no IANA actions.

7. Security Considerations

The operations in Section 4 and 5 describes the forwarding of SFC packets between named SFs based on URIs exchanged in HTTP messages. For security considerations, TLS is sufficient between originating node and Nsff, Nsff to Nsff, Nsff to destination. TLS handshake allows to determine the FQDN, which in turn is enough for the service routing decision. Supporting TLS also allows the possibility of HTTPS based transactions.

8. Informative References

[_3GPP_SBA]
          3GPP, "Technical Realization of Service Based
          Architecture", 3GPP TS 29.500 0.4.0, January 2018,
          <http://www.3gpp.org/ftp/Specs/html-info/29500.htm>.

[_3GPP_SBA_ENHANCEMENT]
          3GPP, "New SID for Enhancements to the Service-Based 5G
          System Architecture", 3GPP S2-182904 , February 2018, <htt
          p://www.3gpp.org/ftp/tsg_sa/WG2_Arch/TSGS2_126_Montreal/
          Docs/S2-182904.zip>.

[I-D.purkayastha-bier-multicast-http]
          Purkayastha, D., Rahman, A., and D. Trossen, "Multicast
          HTTP using BIER", draft-purkayastha-bier-multicast-http-00
          (work in progress), March 2018.

[I-D.purkayastha-sfc-service-indirection]
          Purkayastha, D., Rahman, A., Trossen, D., Despotovic, Z.,
          and R. Khalili, "Alternative Handling of Dynamic Chaining
          and Service Indirection", draft-purkayastha-sfc-service-
          indirection-02 (work in progress), March 2018.

[Reed2016]
          Reed, M., Al-Naday, M., Thomas, N., Trossen, D., and S.
          Spirou, "Stateless multicast switching in software defined
          networks", ICC 2016, 2016.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function
          Chaining (SFC) Architecture", RFC 7665,
          DOI 10.17487/RFC7665, October 2015,
          <https://www.rfc-editor.org/info/rfc7665>.

   [RFC8279] Wijnands, IJ., Ed., Rosen, E., Ed., Dolganow, A.,
              Przygienda, T., and S. Aldrin, "Multicast Using Bit Index
              Explicit Replication (BIER)", RFC 8279,
              DOI 10.17487/RFC8279, November 2017,
              <https://www.rfc-editor.org/info/rfc8279>.

   [RFC8300] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed.,
              "Network Service Header (NSH)", RFC 8300,
              DOI 10.17487/RFC8300, January 2018,
              <https://www.rfc-editor.org/info/rfc8300>.

Authors' Addresses

   Dirk Trossen
   InterDigital Europe, Ltd
   64 Great Eastern Street, 1st Floor
   London EC2A 3QR
   United Kingdom

   Email: Dirk.Trossen@InterDigital.com

   Debashish Purkayastha
   InterDigital Communications, LLC
   1001 E Hector St
   Conshohocken
   USA

   Email: Debashish.Purkayastha@InterDigital.com

   Akbar Rahman
   InterDigital Communications, LLC
   1000 Sherbrooke Street West
   Montreal
   Canada

   Email: Akbar.Rahman@InterDigital.com