OPEN DAYLIGHT SUMMIT

ODL: Service Function Chaining

Reinaldo Penno (repenno@cisco.com)

Paul Quinn (paulq@cisco.com)

#ODSummit

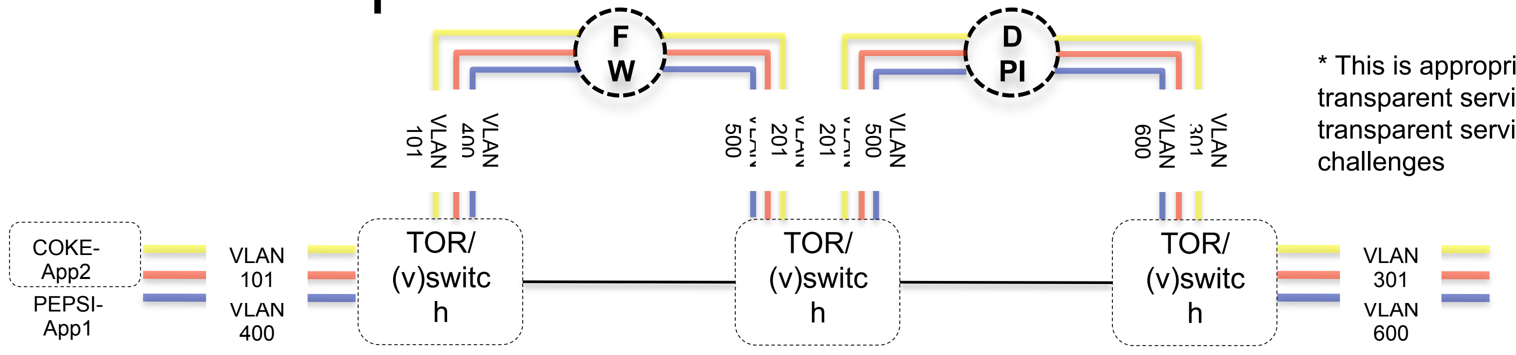
Agenda

- Why do we care about service function chaining?
- A modern architecture for service function chaining
- Service function chaining as a service
- Opendaylight Service Function Chaining Implementation
- Opendaylight SFC+GBP Integration
- Coming to a theater near you: SFC+GBP=NFV Whole Stack

Network Service Insertion (Today)
Off-box Service Chaining

- Services are built using rudimentary service chaining techniques; accomplished via hop-by-hop switching/routing changes or inline services
  - <u>Very complex:</u> VLAN-stitching, Policy Based Routing (PBR), Routing tricks, etc.
  - <u>Static:</u> no dynamic, horizontal or vertical scaling, and requires network changes
  - <u>Operationally disjoint:</u> no "whole stack" view or orchestration
- Service functions are deployed based on physical network topology and physical network elements
  - Changes to service functions or ordering within a service chain require network changes
    - Example: Firewalls typically require "in" & "out" layer-2 segments; adding new FW means adding new layer-2 segments to the network topology
  - Inhibits optimal use of service resources; limits scale, capacity & redundancy

Service Chaining (Today)
Generic Representative Use Case



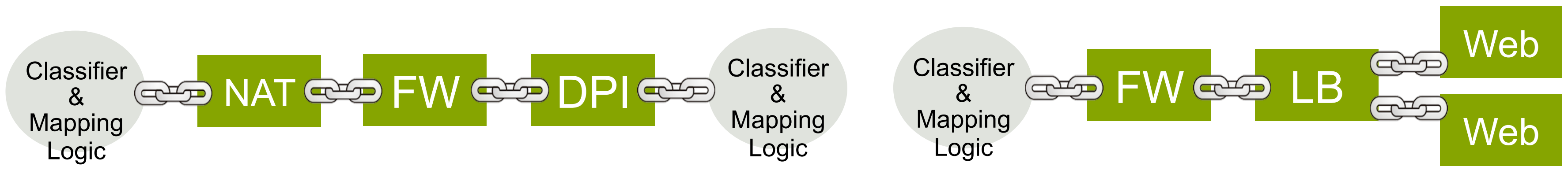
* This is appropriate for all transparent services. Non-transparent services present other challenges

- VLAN stitching overloaded for tenant separation, policy creation, and service chain construction
- Service chain ordering or addition of new services requires network topology changes
- **All** traffic flows through **all** services regardless of need
  - Increases load on services and increases configuration complexity

Network Service Insertion Primer

How are services built and what is Service Chaining?

- Linkage of service functions to realize a service is called Service Chaining
- A (logical) classification function selects traffic that needs to be chained
  - The policy can be as simple as match on VLAN or VRF or match flow rules
  - Or it could be complex policies including subscriber ID and application parameters


Classifier & Mapping Logic
NAT
FW
DPI
Classifier & Mapping Logic
Classifier & Mapping Logic
FW
LB
Web
Web

New Technology Trends
Changing Network Service Insertion Landscape

- Existing network service insertion techniques cannot remain static and must evolve
- SDN & NFV enabling control / data plane independence and virtualization of network elements

From:	To:
Physical appliances	Virtual & physical appliances
Static services	Dynamic services
Separate domains: physical vs. virtual	Seamless physical & virtual interoperability
Hop-by-hop service deployment	Chain of “service functions”
Underlay networks	Dynamic overlay networks
Topological dependent service insertion	Insertion based on resources & scheduling
No shared context	Rich metadata
Policy based on VLANs	Policy based on metadata

Evolving Service Chaining
Overlay-based Service Chaining

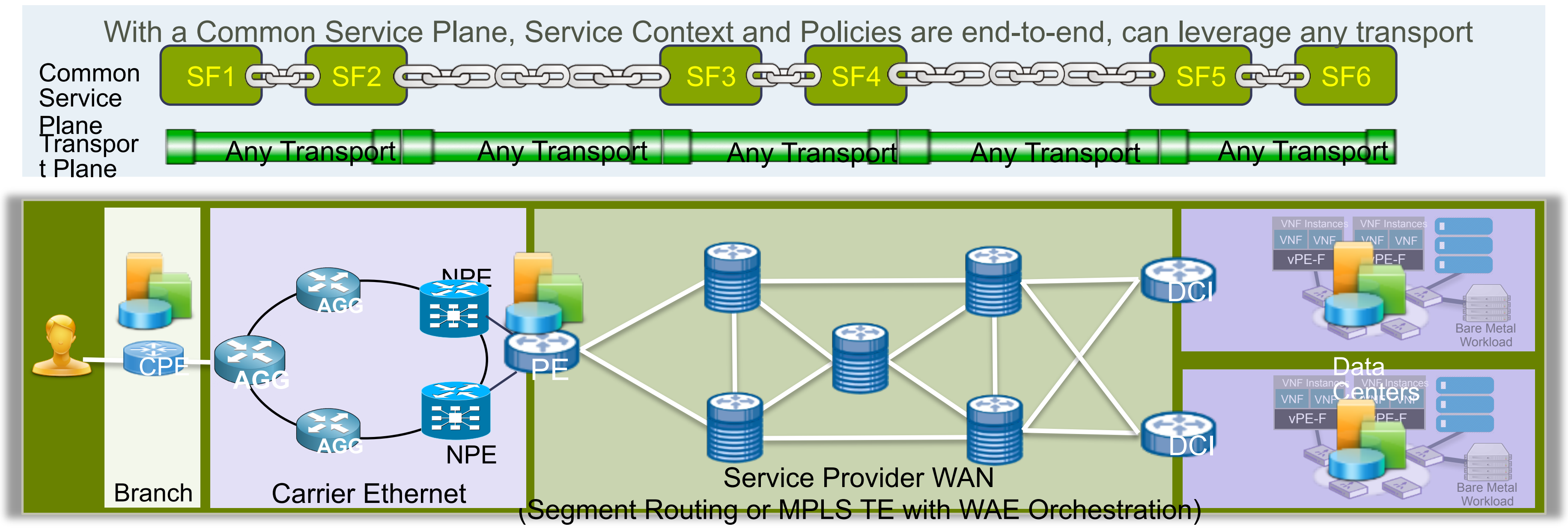
- So can't we just use an overlay?
- 1st step towards true chaining
- Suffers from a number of significant drawbacks:
  - Configuration complexity
  - Limited service chain construction capabilities; Rigid service ordering (e.g. chains but not graphs)
  - Must support many overlays, mapping between overlays
  - Tenant-ID is not granular enough
  - No way to pass "more" information; No common service context
  - Limited visibility and audit capabilities
  - Per service (re)-classification

Evolving Network Service Insertion

Architectural Requirements

- Service deployments will be driven by applications and application policy
- Services will be built using flexible service graphs rather than linear service chains
- The service elements used to build service chains will be both physical and virtualized
- Flexible placement of service elements in transport and topologically independent fashion
- Policy enforcement via metadata exchange

Why Common Service Plane is Important



Common Service Plane
Network Service Header (NSH) 101

- Network Service Header is a data-plane protocol that represents a service path in the network
- **IETF adopted protocol**
- Two major components: **path information** and **metadata**
  - Path information is akin to a subway map: it tells the packets where to go without requiring per flow configuration
  - Metadata is information about the packets, and can be used for policy
- NSH is added to packet via a **classifier**
- NSH is carried along the chain to services
  - Intermediate nodes do not need to be NSH aware
  - Non-NSH enabled services are supported

Network Service Header (NSH)
Core Driving Principles

- Support for all service graph topologies; move up the stack from linear service chains
- Provide a transport independent and topology agnostic service plane
- Remove the need for service functions to participate in a transport / fabric overlay
- Simplify service AND network provisioning
- Enable a broad range of classification types and sources
- Provide clear visibility and OAM to users
- Allow the network transport to “do its job” and evolve
- Centralized or distributed control plane
- Enable metadata conveyance to/from service functions and the network

Network Service Header (NSH) – MD-Type 1
Service Plane Encapsulation Format

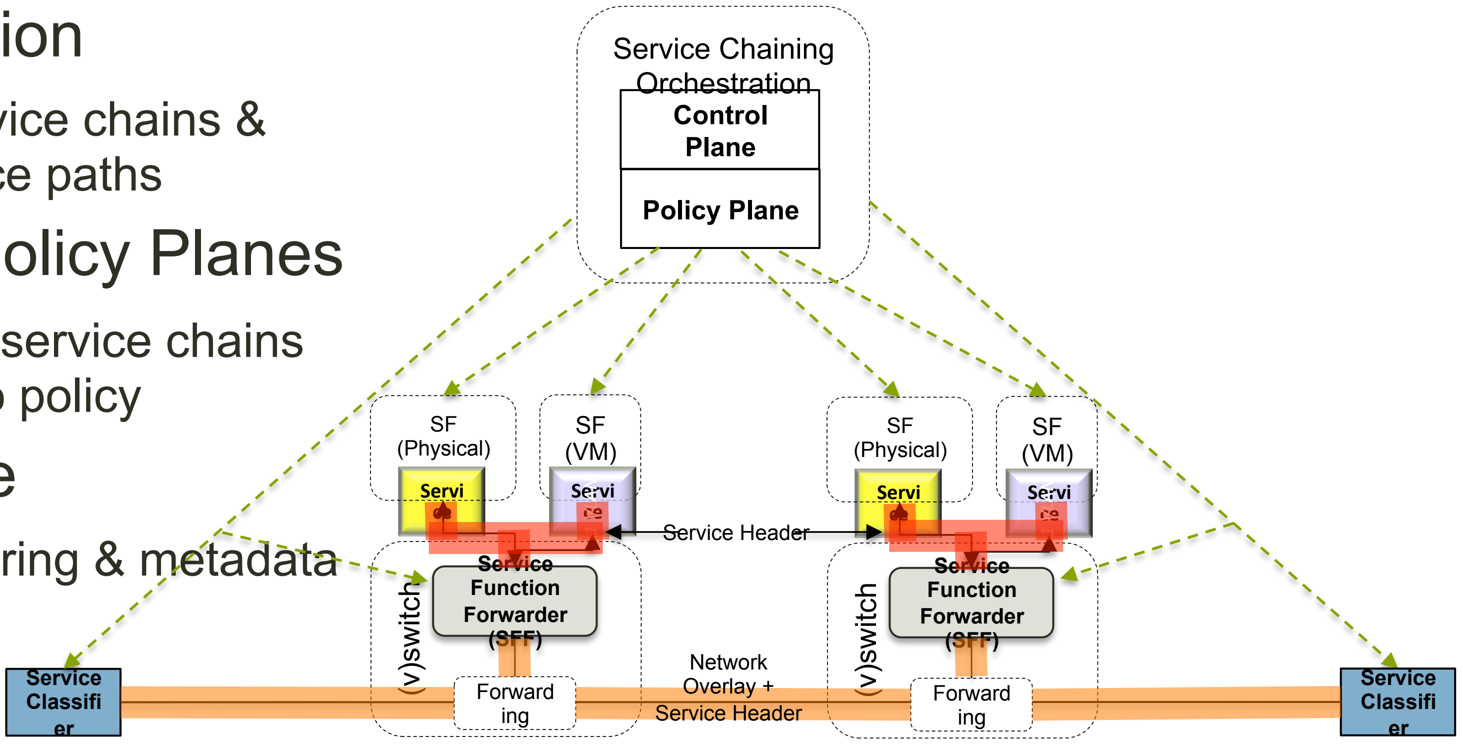
00	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Ver		O	C	R	R	R	R	R	R	Length (6)						MD Type (8)								Next Protocol (8)							
Service Path Identifier (24)																								Service Index (8)							
Mandatory Context Header (1) - Network Platform Context																															
Mandatory Context Header (2) - Network Shared Context																															
Mandatory Context Header (3) - Service Platform Context																															
Mandatory Context Header (4) - Service Shared Context																															
Original Packet Payload																															

SFC Data Plane Components

- Service Classifier
  - Determines which traffic requires service and forms the logical start of a service path
- Service Path
  - A service path is the actual forwarding path used to realize a service chain
  - Think of service chain as the “intent”; service path the actual instantiation of the chain in the network
- Service Function Forwarder (SFF)
  - Responsible for delivering traffic received from the network to one or more connected service functions according to information carried in the network service header as well as handling traffic coming back from the SF
- Service Function Proxy
  - Component used to process network service headers on-behalf of an attached SF

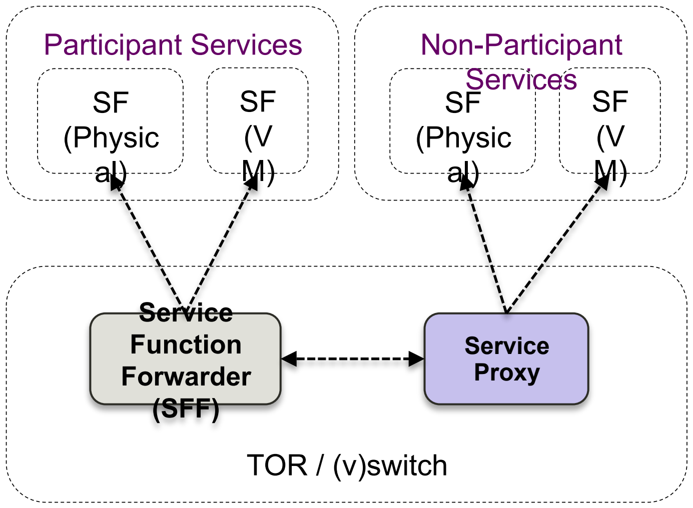
Services Function Chaining Primer
High-level Component Structure

- Orchestration
  - Define service chains & build service paths
- Control / Policy Planes
  - Instantiate service chains adhering to policy
- Data Plane
  - Traffic steering & metadata



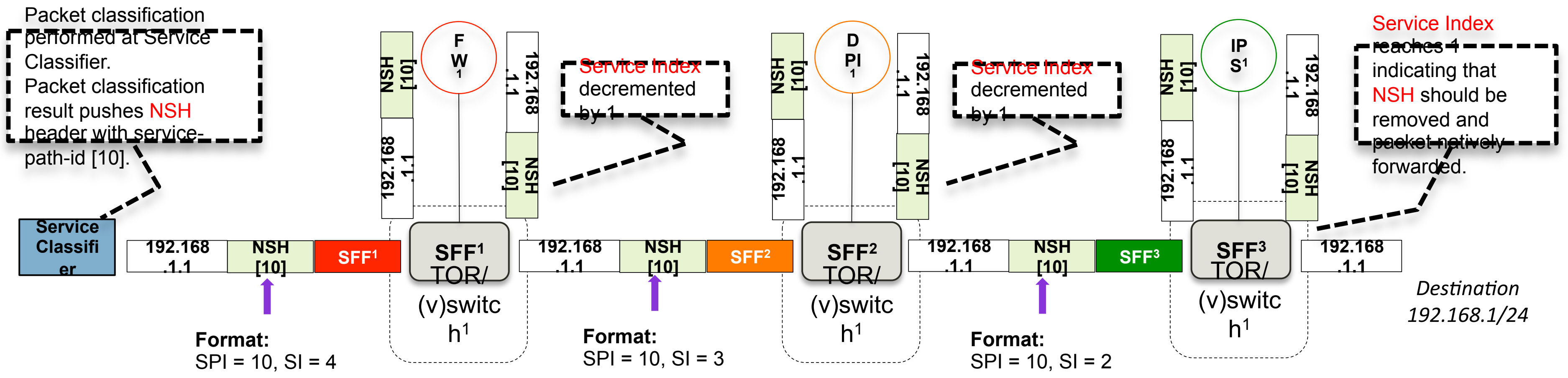
Service Function Proxy
Support for Participant / Non-Participant Services

- Legacy service functions may not have the capability to process packets encapsulated with a network service header
- The network service header architecture introduces the concept of a “Service Proxy” that is responsible for processing of the network service headers and mapping to/from service functions
- Allows for participant and non-participant services to co-exist and belong to the same service chain



Unidirectional Service Chain

Packet Forwarding – Participant Service Functions

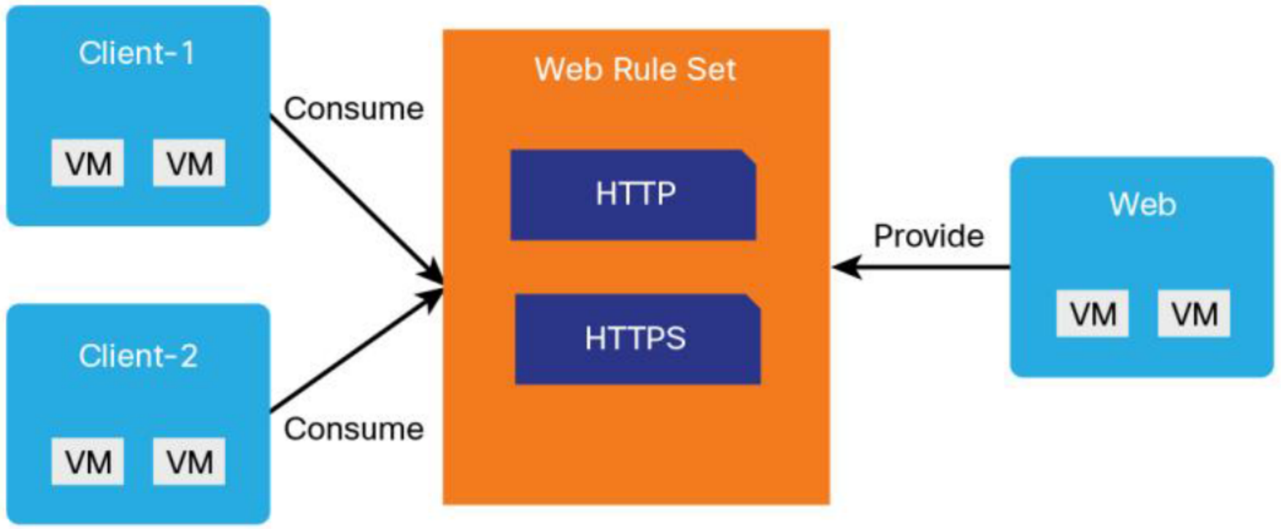


Packet classification performed at Service Classifier. Packet classification result pushes NSH header with service-path-id [10].
Service Classifier
192.168.1.1
NSH [10]
SFF¹
Format: SPI = 10, SI = 4
FW¹
SFF¹ TOR/(v)switch¹
Service Index decremented by 1
Format: SPI = 10, SI = 3
SFF²
DPI¹
SFF² TOR/(v)switch¹
Service Index decremented by 1
Format: SPI = 10, SI = 2
SFF³
IPS¹
SFF³ TOR/(v)switch¹
Service Index reaches 1 indicating that NSH should be removed and packet natively forwarded.
Destination 192.168.1/24

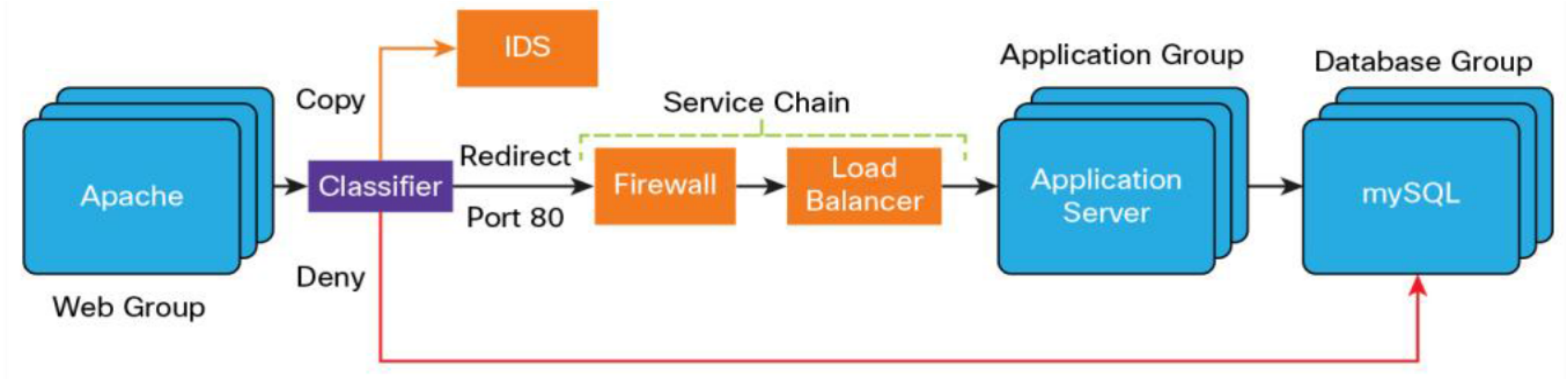
- Transport encapsulation directed to SFF
  - SFF depicted as resident in TOR/(v)Switch but could also be implemented within SF
  - If SFF resident in SF then transport encapsulation addresses SF directly

Group Based Policy (GBP)

- **Policy is a top-level abstraction** which ‘nodes’ are groups of endpoints and edges are the directional policy between them.
- **Policy specifies the semantic ‘what’** we want for network flows



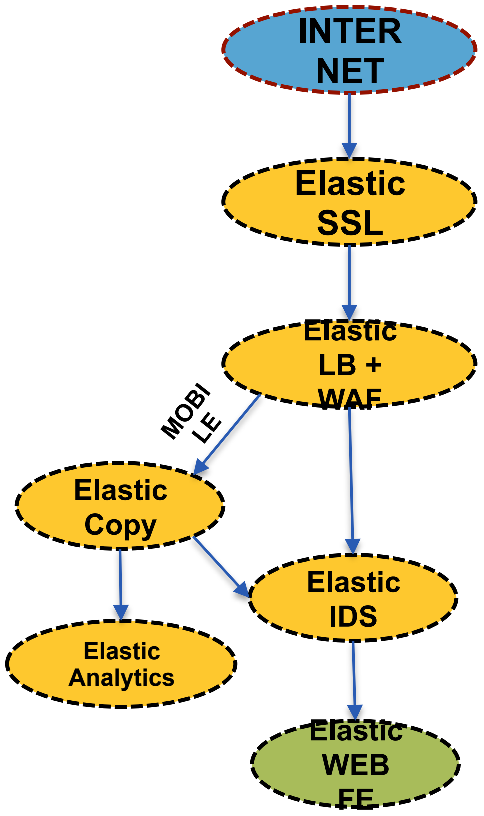
SFC (Service) Abstraction



- **The service-layer abstraction** provides the semantic **how** for service graph traversal (enabled by IETF SFC/NSH).
- Nodes are **network functions** (physical or virtual) and edges indicate the **direction, order and sequence** of the flow of traffic through those chains.
- Avoids repurposing traditional networking constructs by reducing the need to ape “logical linearity” with them by churning the underlying topology to create dynamic services.

The Power of SFC NSH + GBP

- Service path is **decoupled from transport header**
- The encapsulation **carries metadata**
  - Imputed (measurement), VRF context, User context (User ID), intermediate result, etc.
  - Important tool to ***move us past "NETWORK function virtualization***"
- GBP rendered SFC can **reflect business policy**
  - All traffic between the Internet & web front end servers apply:
    - Multi-tenancy, Openstack/Neutron integration
    - De/Encryption with highest throughput / low latency and least cost
    - Copy all "mobile" (metadata ID) only transactions to a Big Data analytics system at most optimal point (cost & least latency impact)
    - Send all traffic through a SLB+WAF and IDS



Opendaylight Service Function Chaining Implementation

Opendaylight SFC Main Components

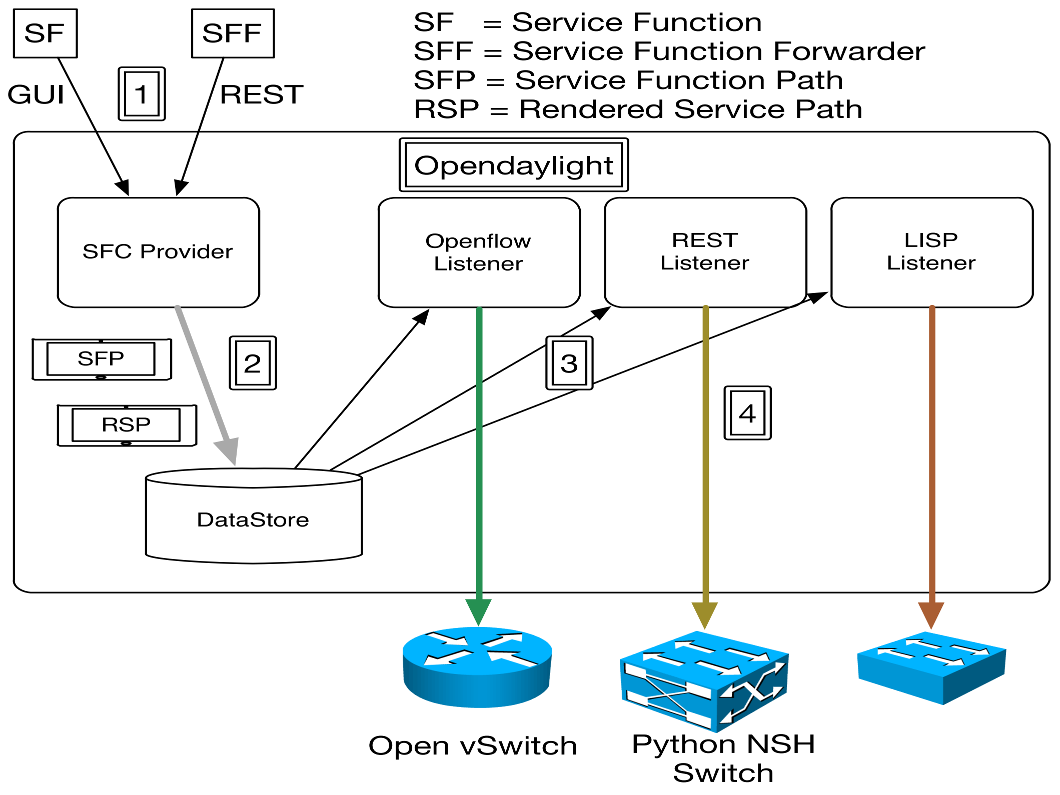
- ODL SFC implementation components
  - Provider
  - YANG Models
  - UI
  - Data Plane
  - Data store Listeners and Renderers
    - REST
    - Openflow
    - OVS

https://wiki.opendaylight.org/view/Service_Function_Chaining:Main

Big Picture

- ODL SFC in essence a point to multipoint architecture
- SFC Provider manages all configuration information provided by orchestration system or admin.
- SFC Provider writes constructed Service Function Paths and Rendered Service Path to the datastore
- Protocol datastore listeners are notified of service objects creation
- These listeners will process RSP information and communicate to their controlled southbound devices

Opendaylight SFC Architecture



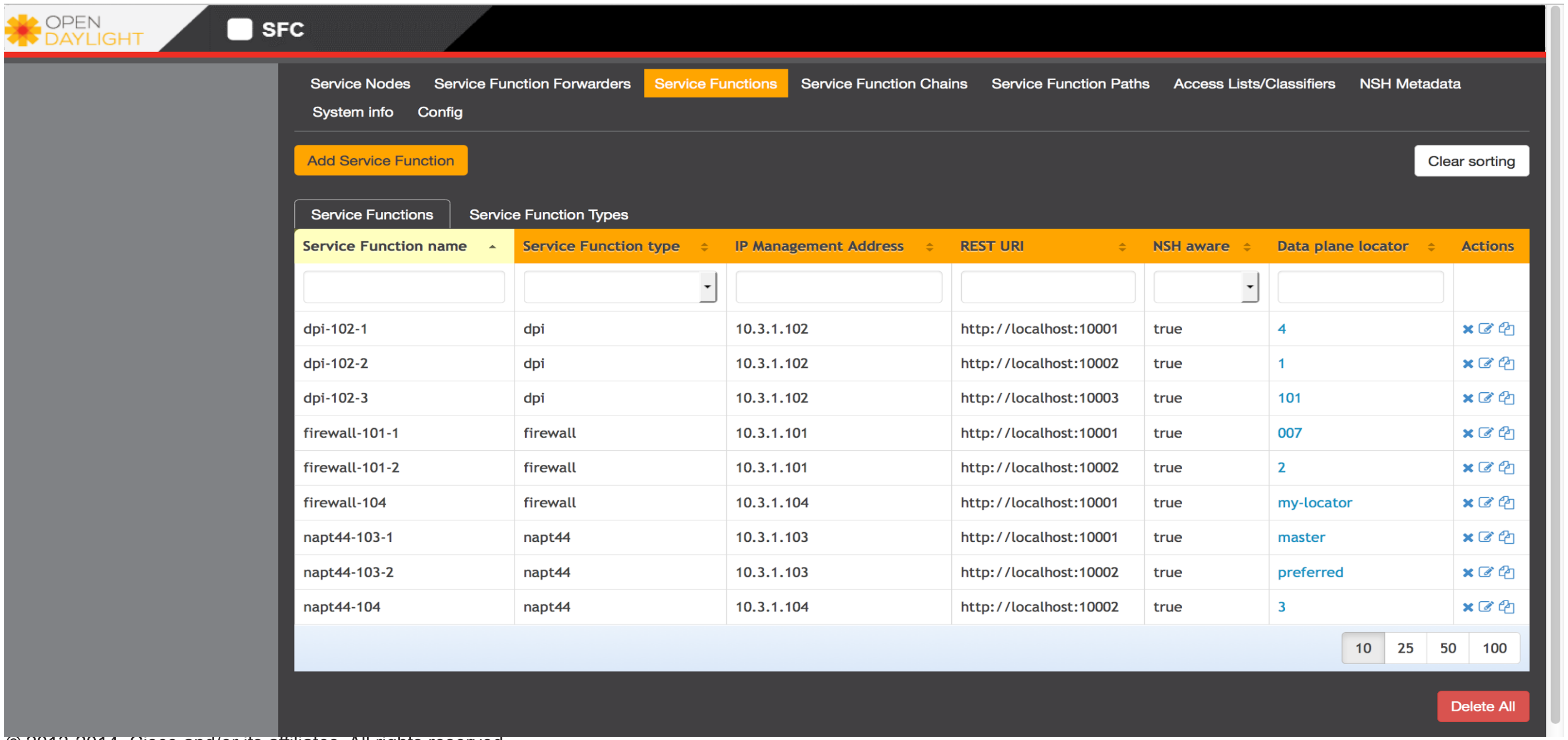
Yang Models

- Cornerstone of SFC implementation
- Data model driven development
- 20+ Yang models
- They cover data plane provisioning, service function selection, statistics, classification, OVS and OF augmentations and monitoring
- Persistence and Clustering
- Yang Models + Datastore = Anonymous Point to Multipoint messaging system

SFC-UI

- One stop shop for everything SFC
- Provides graphical view and configuration of Rendered Service Paths, Service Chains, Service Functions, etc
- Extremely easy to use
- Makes configuration and repetitive tasks easy: uses templates, allows copy & replicating configuration, bulk edits, amongst others
- UI has built-in diagnostics to tell if SFC components are running, state, pull logs from ODL, amongst others.

SFC Front End



Service Function name	Service Function type	IP Management Address	REST URI	NSH aware	Data plane locator	Actions
dpi-102-1	dpi	10.3.1.102	http://localhost:10001	true	4	
dpi-102-2	dpi	10.3.1.102	http://localhost:10002	true	1	
dpi-102-3	dpi	10.3.1.102	http://localhost:10003	true	101	
firewall-101-1	firewall	10.3.1.101	http://localhost:10001	true	007	
firewall-101-2	firewall	10.3.1.101	http://localhost:10002	true	2	
firewall-104	firewall	10.3.1.104	http://localhost:10001	true	my-locator	
napt44-103-1	napt44	10.3.1.103	http://localhost:10001	true	master	
napt44-103-2	napt44	10.3.1.103	http://localhost:10002	true	preferred	
napt44-104	napt44	10.3.1.104	http://localhost:10002	true	3	

10 25 50 100

SFC JSON Data

```
"service-function": [
  {
    "name": "SF5",
    "sf-data-plane-locator": [
      {
        "name": "vxlan",
        "ip": "10.0.1.43",
        "port": 40001,
        "transport": "service-
locator:vxlan-gpe",
        "service-function-forwarder":
"SFF4"
      }
    ],
    "nsh-aware": true,
    "rest-uri": "http://
10.0.1.43:5000",
    "ip-mgmt-address": "10.0.1.43",
    "type": "service-function-
type:napt44"
  }
```

```
"service-function-forwarder": [
  {
    "name": "SFF4",
    "sff-data-plane-locator": [
      {
        "name": "eth0",
        "data-plane-locator": {
          "port": 4789,
          "ip": "10.0.1.44",
          "transport": "service-
locator:vxlan-gpe"
        }
      }
    ],
    "rest-uri": "http://10.0.1.44:5000",
    "service-function-dictionary": [
      {
        "name": "SF5",
        "type": "service-function-
type:napt44",
        "sff-sf-data-plane-locator": {
          "port": 40001,
          "ip": "10.0.1.43",
          "transport": "service-
locator:vxlan-gpe"
        }
      }
    ],
    "ip-mgmt-address": "10.0.1.43",
  }
```

SFC JSON Data

```
"service-function-chain": [
  {
    "name": "SFC2",
    "sfc-service-function": [
      {
        "name": "firewall-
abstract2",
        "type": "service-function-
type:firewall",
        "order": 0
      },
      {
        "name": "napt44-
abstract2",
        "type": "service-function-
type:napt44",
        "order": 1
      }
```

```
"service-function-path": [
  {
    "name": "Path-2-SFC2",
    "service-chain-name":
"SFC2",
    "symmetric": true
  }
```

Operational Data

```
"rendered-service-path": [
  {
    "name": "Path-2-SFC2",
    "parent-service-function-path": "Path-2-SFC2",
    "path-id": 9,
    "service-chain-name": "SFC2",
    "starting-index": 255,
    "rendered-service-path-hop": [
      {
        "hop-number": 0,
        "service-function-name": "SF4",
        "service-function-forwarder": "SFF3",
        "service_index": 255
      },
      {
        "hop-number": 1,
        "service-function-name": "SF5",
        "service-function-forwarder": "SFF4",
        "service_index": 254
      }
    ]
  }
```

Opendaylight SFC+GBP Integration

#ODSummit

Summary

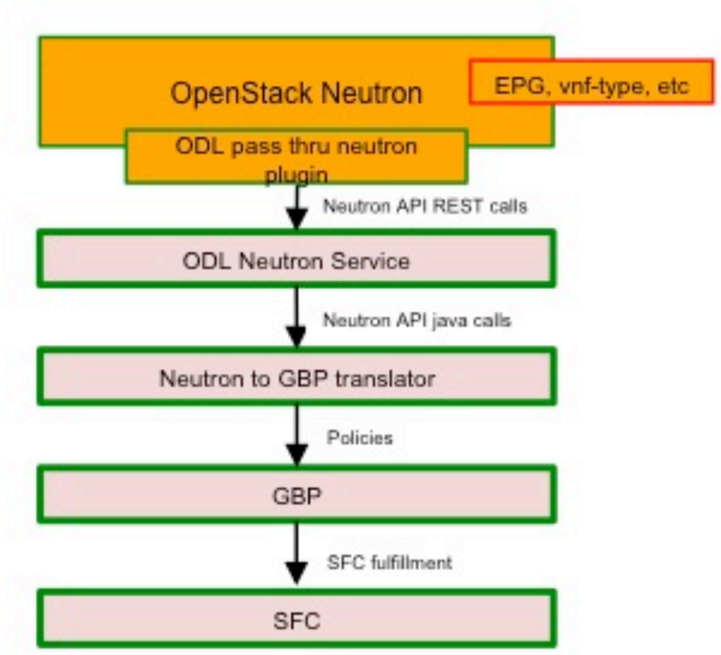
- What is it?

  Integration of two important and popular Opendaylight projects.

  - GBP: Policy abstraction. Allows users to express network configuration in a declarative versus imperative way.
  - SFC: The WG Session you are attending now…hopefully this one is clear

- Why should you care?
  - Running code (open-source and multi-vendor)
  - Cornerstone of NFV “whole stack” solution
  - Opendaylight Neutron Integration (GBP)

Top-Down View

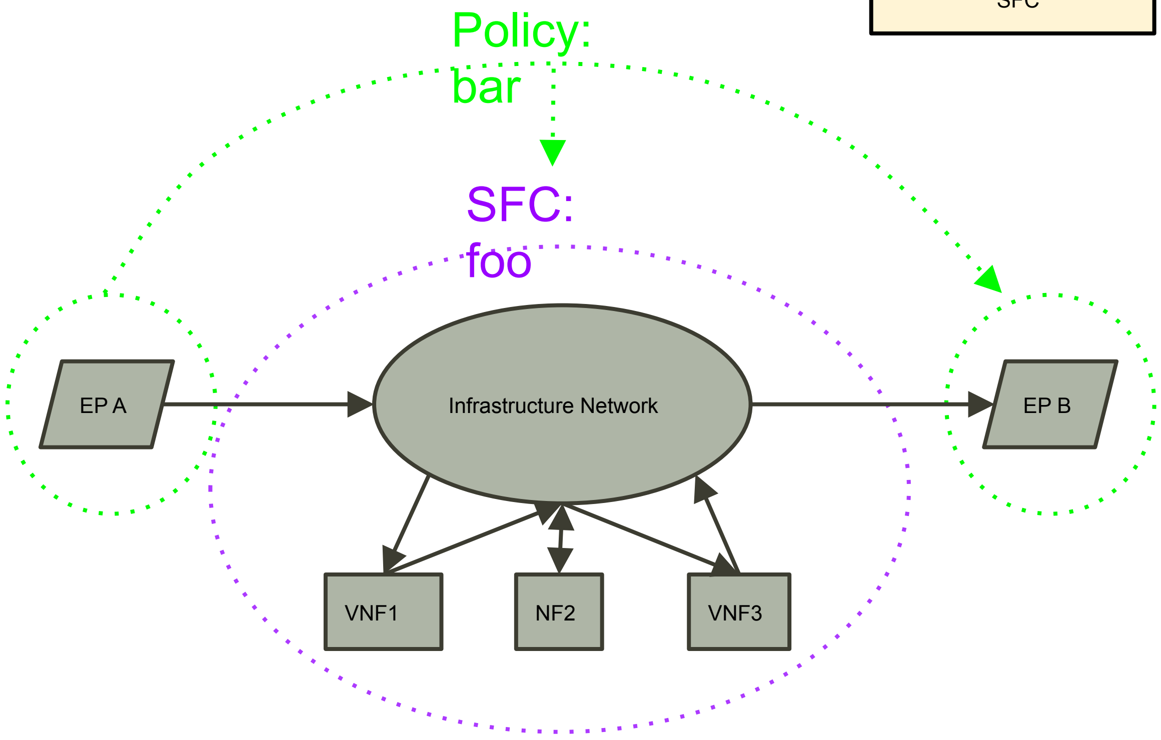


Network Integration Primer

Glossary

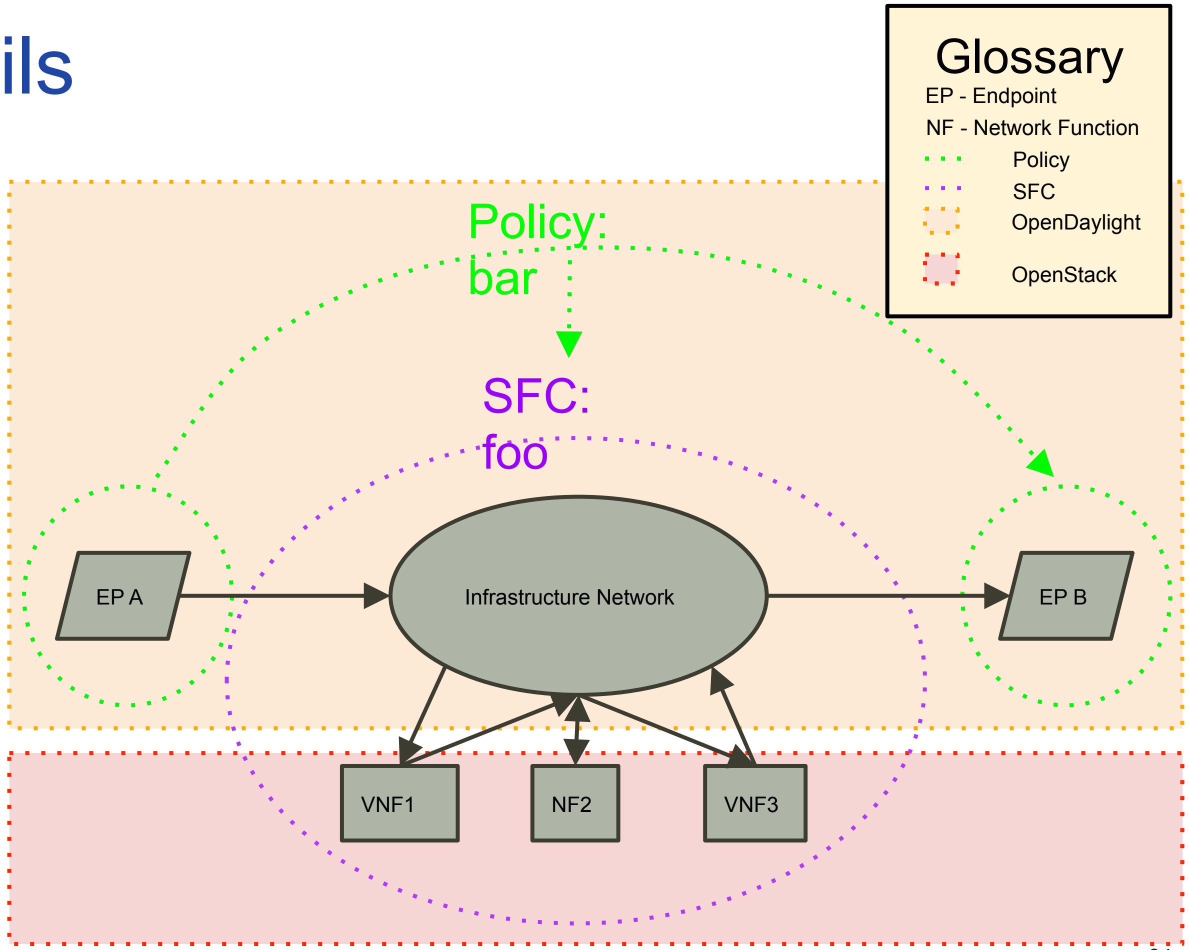
EP - Endpoint	
VNF - Network Function	
· · · (green)	Policy
· · · (purple)	SFC

1. **SFF and classifier are OpenvSwitches**
2. **Encapsulation is VXLAN +NSH+ Ethernet**
3. **Dummy Service Functions**



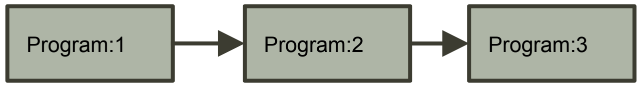
Integration Details

1. **GBP defines policy:bar and actions. One of the actions is "chain:foo"**
2. **GBP calls into SFC and asks for chain:foo**
3. **SFC creates the needed topology: OVS bridges and forwarding rules**
4. **SFC returns path-ID, starting index, first hop IP:port, and encapsulation to GBP**
5. **GBP creates the necessary classifier rules to direct packets to the path and attach context headers (metadata)**



Why this design? Unix philosophy

- “Write programs that do one thing and do it well” - Doug McIlroy
- “Write programs that work well together” - Doug McIlroy
- “Write programs to handle text streams, because that is the universal interface” - Doug McIlroy
    - Rethink this as “Text names for things you don’t understand or need to understand”



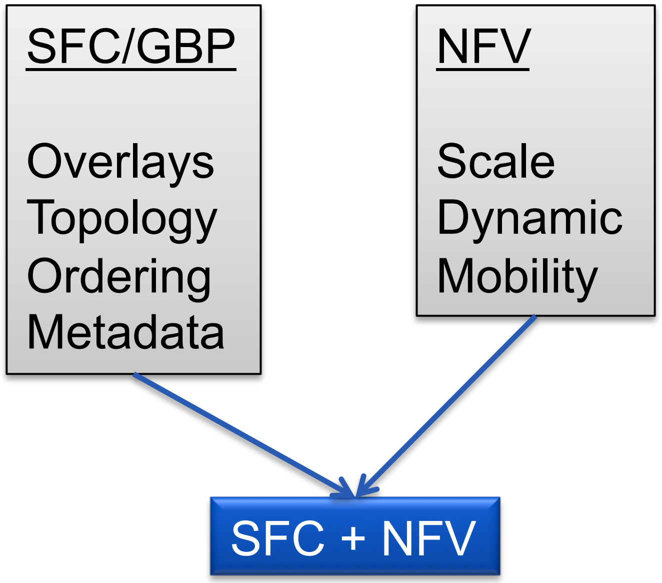
SFC+GBP Metadata Usage

- Context header 1 = Original destination IP address. Imagine if SFC did not exist
- Context header 2 = VNID

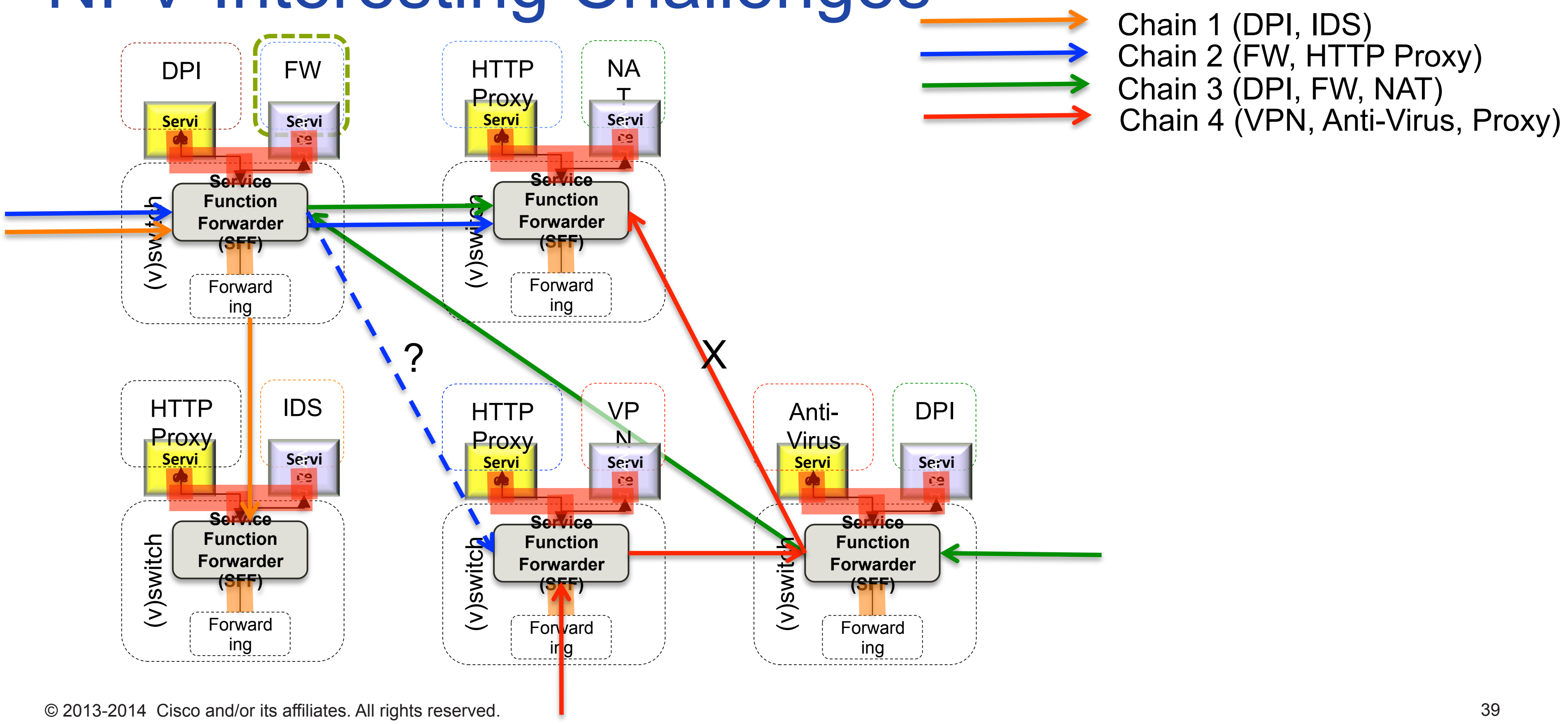
- These two context headers allow for multi-tenancy and overlapping IP addresses

Coming to a theater near you: SFC+GBP=NFV Whole Stack

SFC/GBP and NFV Challenges



NFV Interesting Challenges



NFV Interesting Challenges

- Topology Matters
  - Explosion of tunnels
  - Explosion of traffic (Hotspots)
  - Reachability
- Symmetry and Ordering Matters
  - Stateful services
- Operations and Management
  - Testing paths
  - Testing Service Functions

Topology Matters

- Connected Graph of SFFs, a.k.a. switches/routers
- Reuse tunnels
- Create and place VMs (Services) intelligently
- Smart and fast fail-over
  - Fail to SFFs that are reachable *and* have the necessary Services attached

Symmetry and Ordering Matters

- All Stateful services in a chain need to process packet in strict sequence
- Packet for the same session flowing in the reverse direction need to traverse the exact same Service Functions.
- Errors packets such as ICMP Errors, TCP Resets and others also need to traverse the chain in the exact the same (or reverse) order of the data packets that caused the error

Operations and Management

- Testing Service Paths as opposed to just routing paths
- Testing Service Functions as opposed to VM reachability
- Testing Symmetric paths

Beryllium Features under Consideration

#ODSummit

Some Beryllium Features under Consideration

- Automate all possible tests, integrate with Robot
- NSH proxy support
- Reclassification support mid-path
- Multi-tenancy (NSH and overlay)
- Full netconf integration
- MD-SAL network topology-based shortest path algorithm
- Stats, monitoring.
- Network Topology

References

- SFC Wiki: https://wiki.opendaylight.org/view/Service_Function_Chaining:Main
- GBPSFC demo: https://github.com/alagalah/gbpsfc-env
- Hackathon Document: SFC hackathon Document
- NFV Whole Stack: https://www.sdxcentral.com/articles/contributed/adopt-a-whole-stack-view-to-nfv-david-ward/2015/05/

Thank you.