Service Function Chaining: Programming Your Data and Service Planes

Joe Clarke – Distinguished Engineer, Services
Carlos Pignataro – Distinguished Engineer, Services
BRKSDN-2066
Abstract

Software Defined Networking (SDN) combined with Network Function Virtualization (NFV) provide unprecedented programmatic access to drive networks in new and exciting ways. These capabilities lead to dynamic service creation, with Service Function Chaining (SFC) completing the programmable picture, creating a "whole stack" view. Service chaining realizes network virtualization, allowing network services to exchange metadata and be seamlessly deployed into the packet path independent of their physical location. Service chaining, and its underlying standards such as Network Service Header (NSH) offer a innovative way to "software define" packet processing. Complementing these technologies is the Open Source Vector Packet Processing (VPP) that enables one to add new data path services with tremendous performance.

This session will show the value of service creation and the power that software-defined packet processing can afford. This session will describe a journey that starts by looking at how to program devices and virtual services using both Service Function Chaining and the Network Service Header. Further, this session will explore details of how policy and declarative intent drives the creation of service chains. Examples will be shown on how to inspect traffic, manipulate packets, and inject new flows. Integrating these software services with VPP in order to achieve accelerated performance will be discussed. This session will focus heavily on programming. Code examples will be provided for both the data path service set and Service Function Chaining to highlight the power both technologies can provide.

Attendees should have a background in programming prior to attending this talk.
Why Play With Your Traffic?

0110101010101

01001100110
01001100110

ART!
Why Play With Your Traffic

Software Defined Packet Control

Cisco live!
Agenda

- Introduction to Data Path Programming
- Intro to Service Chaining (SFC), Network Service Header (NSH) and Vector Packet Processing (VPP)
- Programming With SFC/NSH
- Example Use Cases
- Key Takeaways
What Is Data Path Programming?

- Device programmability has typically focused on the control plane of the device
  - CLI
  - SNMP
  - Embedded Event Manager
  - NETCONF

- Data path programmability opens application developers up to the packets flowing through the device

- Can come in various forms
  - Instructing hardware how to forward packets (e.g., OpenFlow)
  - Attaching applications to the packet path of the device
  - Operations, Administration, and Maintenance
Why Is Data Path Programming Interesting?

• Vendors do not always design for every use case and scenario

• Data path programming fills the gap by opening the network to all levels of customization

• Make the network more relevant to your business
  • Create specific optimizations
  • Deliver unique, customized traffic behavior
  • Build new network services

• Enable new network architectures using existing hardware
  • Cloud applications
  • Network Function Virtualization
  • Network visualizations and analytics
Who Does What?

- There are a lot of APIs and abstractions flying around out there

- There are a number of components and solutions that exist at various layers to enable applications
  - Cisco’s APIC controllers
  - OpenDaylight
  - Service Function Chaining
  - OpenFlow

- Who does what and with what technology depends on the use case

- Today, Cisco is helping out customers and partners through DevNet explore and become comfortable with these technologies

- And in turn, partners are also helping their customers realize the power of a more programmable network
Service Function Chaining (SFC) & Network Service Header (NSH)
What Is A Service?

• An application?
  • Web portal for managing your {internet, bank, wireless} account
  • A CRM system
  • A productivity suite

• A collection of resources?
  • Database
  • Compute
  • Storage
  • Bandwidth

• A transit packet processor?
  • Firewall
  • Load balancer
  • NAT gateway
  • Video Transcoder
Chaining Services Together

- A Service Chain is an ordered graph of specific network service functions through which packets must flow
- E.g.: Firewall → NAT → Load Balancer
Service Chaining Today

- Services are built using rudimentary service chaining techniques; accomplished via hop-by-hop switching/routing changes
  - **Dependencies:** Leading to a dependencies between the network & service topologies
  - **Very complex:** VLAN-stitching, Policy Based Routing (PBR), Routing tricks, etc.
  - **Static:** no dynamic, horizontal or vertical scaling, and requires network changes

- 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
SFC – Creating a Services-layer Ecosystem Explosion
SFC – Creating a Services-layer Ecosystem Explosion

Whole Stack for Services

POLICY VIEW
SERVICE VIEW
VIRTUAL TOPOLOGY
PHYSICAL TOPOLOGY
RESOURCE VIEW

Full-Stack
Multi-market industry, realizingNFV and SDN
Customer acceptance
Open standards (IETF, YANG, 3GPP)

Open source (ODL, OF, OVS, VPP)

Open ecosystem
The “Big Idea”

“With a common Service Plane, service policies and context are end-to-end, and can leverage any transport.”
The “Big Idea”

“With a common Service Plane, service policies and context are end-to-end, and can leverage any transport.”

Service Graphs

Service creation through graphs vs. linear Chains
- Support for all graph topologies
- Collection of SFs form a graph

Rich Metadata

Policy distribution b/w Service Functions through metadata
- Shares network and service context
- Ingress (e.g., VRF), classification/DPI, user/service context (e.g., Subscriber) consumed by services
- Includes application-ID

End-to-End Service Path and Policy

NSH
- Single FIB disposition based on Service Path Index (SPI) Field
- Tunnel encapsulation choices; (VXLAN-GPE, GRE, MPLS, etc.)
- NETCONF, YANG, GBP
- OAM and Service Innovation
Service Function Chaining Architecture
High-level Component Structure

- Architecture components
  - Service Chaining Orchestration
    - Define service chains & build service paths
  - Control / Policy Planes
    - Instantiate service chains adhering to policy
  - Data Plane
    - Traffic steering & Metadata
    - Data plane architecture accessible through open APIs
Service Function Chaining
Terminology

- **Service Classifier**
  - Determines which traffic requires service and forms the logical start of a service path.

- **Service Function (SF)**
  - Component used to provide some level of processing to received packets.

- **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 Path**
  - A service path is the actual forwarding path used to realize a service chain
  - Think of service chain as an abstract “intent for packets”; the service path is the actual instantiation of the chain in the network.
  - **Service Chain**: Firewall \(\rightarrow\) NAT \(\rightarrow\) Load Balancer; **Service Path**: cl-fw1 \(\rightarrow\) cl-nat44 \(\rightarrow\) cl-iosslb
Evolving Service Chaining
Example: Business Policy Drives Service Deployment

• A service is rendered based on a business policy like …

• To all traffic between the Internet & web front end servers apply:
  • De/Encryption with highest throughput / low latency and least $$ cost
  • Copy all “mobile” only transactions to a Big Data analytics system
  • Perform the copy at most optimal point ($$ cost & least latency impact)
  • Send all traffic through a load balancer/web application firewall and IDS

• Additionally, deploy this policy with other features like:
  • Service functions are both virtual and physical and vendor neutral
  • Compute & service elasticity; compute mobility

BUT HOW DO WE STEER THE TRAFFIC THROUGH THIS PATH?
Service Function Chaining
Network Service Header (NSH) 101

Network Service Header is a data-plane protocol that represents a service path in the network and provides a common service plane fully orchestrated top to bottom.

- 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)  
The Missing Link To SFC

• A Network Service Header (NSH) contains metadata and service path information that is added to a packet or frame and used to create a service plane. The packets and the NSH are then encapsulated in an outer header for transport.

• More specifically NSH is composed of a 4-byte base header, a 4-byte service path header, and four mandatory 4-byte context headers.
  • **Base header**: provides information about the service header and the payload
  • **Service path header**: provides path identification and location within a path
  • **Context headers**: provide **metadata** that can be shared between network and service nodes
The NSH From A CSR1Kv

**Base Header**

**GRE Encapsulation**
The NSH From A CSR1Kv
The NSH From A CSR1Kv

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>04-23 17:20:36.806570</td>
<td>192.168.1.4</td>
<td>192.168.2.1</td>
<td>TCP</td>
<td>02</td>
<td>Source port: 5787. Destination port: 5787</td>
</tr>
</tbody>
</table>

Context Headers (Metadata)

- NSHv4: Version: 0, c-bit: 0
- Next Protocol Type: IPv4 (1)
- Service Path: 8x800001
- Context Header 1: 0x00000000
- Context Header 2: 0x00000000
- Context Header 3: 0x00000000
- Context Header 4: 0x00000000
The NSH From A CSR1Kv

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>2015-04-23</td>
<td>17:20:37.839314</td>
<td>192.168.1.48</td>
<td>UDP</td>
<td>2066</td>
<td>2066 Destination port: 5787</td>
</tr>
<tr>
<td>29</td>
<td>2015-04-23</td>
<td>17:20:36.039070</td>
<td>192.168.1.48</td>
<td>UDP</td>
<td>2066</td>
<td>2066 Destination port: 5787</td>
</tr>
<tr>
<td>30</td>
<td>2015-04-23</td>
<td>17:20:36.239786</td>
<td>192.168.1.48</td>
<td>UDP</td>
<td>2066</td>
<td>2066 Destination port: 5787</td>
</tr>
<tr>
<td>31</td>
<td>2015-04-23</td>
<td>17:20:36.239006</td>
<td>192.168.1.48</td>
<td>UDP</td>
<td>2066</td>
<td>2066 Destination port: 5787</td>
</tr>
<tr>
<td>32</td>
<td>2015-04-23</td>
<td>17:20:36.439815</td>
<td>192.168.1.48</td>
<td>UDP</td>
<td>2066</td>
<td>2066 Destination port: 5787</td>
</tr>
<tr>
<td>33</td>
<td>2015-04-23</td>
<td>17:20:36.039082</td>
<td>192.168.1.48</td>
<td>UDP</td>
<td>2066</td>
<td>2066 Destination port: 5787</td>
</tr>
<tr>
<td>34</td>
<td>2015-04-23</td>
<td>17:20:36.389595</td>
<td>192.168.1.48</td>
<td>UDP</td>
<td>2066</td>
<td>2066 Destination port: 5787</td>
</tr>
<tr>
<td>35</td>
<td>2015-04-23</td>
<td>17:20:36.808670</td>
<td>192.168.1.48</td>
<td>UDP</td>
<td>2066</td>
<td>2066 Destination port: 5787</td>
</tr>
</tbody>
</table>

Original Packet
But Sometimes More Is More

- The four 32-bit context headers really require one to think about what is absolutely needed to share between nodes and services
- But sometimes you need more
  - Mobility use cases want to transport IMEI and user info
  - Data Center use cases want to transport tenant and application information
- NSH allows for this with Type 2 variable length headers
**Network Service Header (NSH)**

**MD-Type 2 Format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ver</td>
<td>Version</td>
</tr>
<tr>
<td>O</td>
<td>Control</td>
</tr>
<tr>
<td>C</td>
<td>Reserved</td>
</tr>
<tr>
<td>R</td>
<td>Reserved</td>
</tr>
<tr>
<td>R</td>
<td>Reserved</td>
</tr>
<tr>
<td>R</td>
<td>Reserved</td>
</tr>
<tr>
<td>R</td>
<td>Reserved</td>
</tr>
<tr>
<td>R</td>
<td>Reserved</td>
</tr>
<tr>
<td>R</td>
<td>Reserved</td>
</tr>
<tr>
<td>Length (6)</td>
<td>Length of the NSH</td>
</tr>
<tr>
<td>Ver 0</td>
<td>Version</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>MD Type 2</td>
<td>MD Type 2</td>
</tr>
<tr>
<td>Next Protocol (8)</td>
<td>Next Protocol</td>
</tr>
<tr>
<td>Service Path Identifier (24)</td>
<td>Service Path Identifier</td>
</tr>
<tr>
<td>Service Index (8)</td>
<td>Service Index</td>
</tr>
<tr>
<td>Optional Variable Length Context Headers</td>
<td>Optional Variable Length Context Headers</td>
</tr>
<tr>
<td>Original Packet Payload</td>
<td>Original Packet Payload</td>
</tr>
</tbody>
</table>

**TLV Class**

<table>
<thead>
<tr>
<th>Class</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>C</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>C</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>C</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>C</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

**Variable Length Metadata**

As many and as much as your use case requires
Service Function Chaining – Ecosystem

• Dedicated Service Function Chaining (SFC) Working Group in the Internet Engineering Task Force (IETF)

• Several vendors support this idea of Service Function Chaining with Network Service Header
  • Close to becoming a standard

• Hear ye, hear ye! Read all about it!
  • https://tools.ietf.org/html/rfc7498 (Problem Statement)
  • https://tools.ietf.org/html/rfc7665 (SFC Architecture)
Service Function Chaining – Ecosystem

Internet Engineering Task Force (IETF)  J. Halpern, Ed.
Request for Comments: 7665  Ericsson
Category: Informational  C. Pignataro, Ed.
ISSN: 2070-1721  Cisco

October 2015

Service Function Chaining (SFC) Architecture

Abstract

This document describes an architecture for the specification, creation, and ongoing maintenance of Service Function Chains (SFCs) in a network. It includes architectural concepts, principles, and components used in the construction of composite services through deployment of SFCs, with a focus on those to be standardized in the IETF. This document does not propose solutions, protocols, or extensions to existing protocols.
NSH Implementation in Open Source

• **Vibrant open source community**

• Data Plane in **Linux** kernel
  • NSH (along with VXLAN-GPE) support for Lightweight Tunneling

• Data Plane in **OVS**
  • Classifier and SFF controlled via OVSDB and Openflow protocols
  • Decoupled transport and NSH encap/decap

• Data Plane in **FD.io** (in progress)
  • NSH encap/decap in VPP
  • NSH-aware SF
  • Control plane agent Honeycomb
  • Classifier, SFF and Proxy to be supported in new NSH SFC (https://wiki.fd.io/view/Project_Proposals/NSH_SFC) sub-project
Implementation Update: Open Source

• Control Plane in **OpenDaylight**
  • Vibrant project, with new features ever release
  • Integration with ODL Group Based Policy (as a classifier controller) and OVSDB NetVirt (as a classifier controller)
  • Standalone classifier control for Telco use case
  • Pipeline Coexistence: allowing GBP, Netvirt, and SFC to all Coexist on the same OpenFlow switch
  • Refactor of Openflow renderer and YANG models for better stability

• Control Plane in **Openstack**
  • Networking-sfc in Mitaka release, with backend driver directly control NSH based traffic steering of OvS

• Integration with **OPNFV**
  • Service Function Abstract Data Types, allowing for better integration with OPNFV.
  • ODL Beryllium (inc. SFC) in Brahmaputra release
The Fast Data Project (FD.io)

Relentlessly focused on data IO speed and efficiency for more flexible and scalable networks and storage
Vector Packet Processing (VPP)

• At the heart of fd.io is Vector Packet Processing (VPP) technology.

• In development since 2002, VPP is production code currently running in shipping products.

• It runs in user space on multiple architectures including x86, ARM, and Power architectures on both x86 servers and embedded devices. The design of VPP is hardware, kernel, and deployment (bare metal, VM, container) agnostic.

• It runs completely in userspace.
VPP – 1 of 3
Read Vector of Packets

- VPP reads the largest available vector of packets from the network IO layer.
VPP – 2 of 3
Process Vector of Packets

• VPP processes the vector of packets through a Packet Processing graph.
• Rather than processing the first packet through the whole graph, and then the second packet through the whole graph, VPP instead processes the entire vector of packets through a graph node before moving on to the next graph node.
VPP – 3 of 3
Process Vector of Packets

• The graph node architecture of VPP also makes for easy extensibility.

• You can build an independent binary plugin for VPP from a separate source code base (you need only the headers).

• Plugins are loaded from the plugin directory.

• A plugin for VPP can rearrange the packet graph and introduce new graph nodes.
Putting it all together

• VPP is a rapid packet processing development platform for highly performing network applications.

• It runs on commodity CPUs and leverages DPDK

• It creates a vector of packet indices and processes them using a directed graph of nodes – resulting in a highly performant solution.

• Runs as a Linux user-space application

• Ships as part of both embedded & server products, in volume; Active development since 2002

• See also: FD.IO (The Fast Data Project)
The OpenDaylight Project

- An open source project formed by industry leaders and others under the Linux Foundation with the mutual goal of furthering the adoption and innovation of Software Defined Networking (SDN) through the creation of a common vendor supported framework.

- Focus: Customers with some programming resources that desire a free, community-supported SDN controller

- Downloads available from http://www.opendaylight.org

Platinum: Cisco, Citrix, Microsoft, Brocade, IBM, Ericsson, Dell, Intel, Red Hat

Gold: NEC, PLUMgrid, Juniper Networks, VMware, Arista

Silver: Nuage Networks, Fujitsu
OpenDaylight SDN Platform
Linux Foundation
Open Source
Software Defined Networking
Innovation
Collaboration
Network Function Virtualization
SFC+NSH
OpenDaylight SFC Implementation

- Released in Lithium, planning Beryllium
- Renderers: OVS, OF and REST
- Hackathon at IETF-92, 93, and 94
- GBP Integration.
- Application Identification, SFC Traceroute, Reference Data plane, NSH client test tool, IPTables classifier.
sff_client.py --remote-sff-ip 10.0.1.41 --remote-sff-port 4789 --sfp-id 22 --sfp-index 255 --trace-req --num-trace-hops 3

Sending Trace packet to Service Path and Service Index: (22, 255)
Trace response...
Service-hop: 0. Service Type: dpi, Service Name: SF1, Address of Reporting SFF: ('10.0.1.41', 4789)
Service-hop: 1. Service Type: firewall, Service Name: SF4, Address of Reporting SFF: ('10.0.1.42', 4789)
Service-hop: 2. Service Type: napt44, Service Name: SF5, Address of Reporting SFF: ('10.0.1.43', 4789)
Trace end
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-GBP 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)
The ODL SFC Application

- ODL contains an SFC application with:
  - REST API
  - GUI
  - YANG models
- Includes a reference implementation built on Python and using VXLAN-GPE for transport
Playing With ODL And SFC

- Download the DevNet VM from https://communities.cisco.com/community/developer/dev-vm

- Follow the SFC wiki at https://wiki.opendaylight.org/view/Service Function Chaining:Main to pull down the SFC module

- Play with the SFC REST API and GUI
Where Is SFC+NSH Supported?

• Almost all major Cisco platforms have a roadmap
  • ASR1K, ISR4400, CSR1Kv, ASR9K, Nexus 7K, Nexus 9K
  • Expect to see code releases this year that support SFC+NSH

• Services coming along, including 3rd parties
  • NBAR classification
  • Server Load Balancing
  • Firewalls

• Orchestration integration
  • APIC, OpenDaylight, Cisco Open SDN Controller
Why Is This Cool?

• SFC and NSH enable a more software-defined landscape by allowing service deployment to be fluid and independent of the network topology

• This is key in order to move from a purely physical network into network function virtualization (NfV) and a Full Stack view

<table>
<thead>
<tr>
<th>Today</th>
<th>Tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical appliances</td>
<td>Virtual &amp; physical appliances</td>
</tr>
<tr>
<td>Static services</td>
<td>Dynamic services</td>
</tr>
<tr>
<td>Separate domains: physical vs. virtual</td>
<td>Seamless physical &amp; virtual interoperability</td>
</tr>
<tr>
<td>Hop-by-hop service deployment</td>
<td>Chain of “service functions”</td>
</tr>
<tr>
<td>Underlay networks</td>
<td>Dynamic overlay networks</td>
</tr>
<tr>
<td>Topological dependent service insertion</td>
<td>Insertion based on resources &amp; scheduling</td>
</tr>
<tr>
<td>No shared context</td>
<td>Rich metadata</td>
</tr>
<tr>
<td>Policy based on VLANs</td>
<td>Policy based on metadata</td>
</tr>
</tbody>
</table>
How Does Data Path Programming Fit?

• The Data Path programming capabilities are being ported to use SFC and NSH for transport and metadata exchange.

• This allows one to create a Service Function into which you can plug your applications.
Programming With SFC+NSH
Tapping Into The Data Path Through The Service Path

CSR1Kv

GRE

Service Function
Classifier Configuration

class-map match-all http
    match access-group 101

class-map match-all all-ip
    match access-group 103

class-map match-all ssh
    match access-group 102
!
policy-map type service-chain sfc-pol
    class http
        forward service-path 1 service-index 3
!
service-chain service-function service1
    description Service-Function-1
    ip address 192.168.1.4
    encapsulation gre
!
service-chain service-path 1
    description Service-Chain-1
    service-index 3 service-function service1
    service-index 2 terminate

- Match all HTTP in both directions
- Insert matched traffic into service chain 1 and index 3
- Our service chain consists of one service at 192.168.1.4
Setting Up The Service Topology…Programmatically

- Given that SFC+NSH is industry-standard, there must be an industry-standard way to set up the service topology

- YANG+RESTCONF provides a way to configure devices using well-defined data models

- Instead of passing raw CLI, the modeled attributes can be sent using traditional REST calls

- YANG as a data modeling language is defined in RFC 6020

- RESTCONF is still being standardized in the IETF in draft-ietf-netconf-restconf
Using RESTCONF To Program Devices

YANG interface model

TLS

Configuring The Service Path With RESTCONF

Class Maps

```python
import requests

text = "https://10.1.1.1/api/running/native/class-map"

headers = {
    'authorization': 'Basic YWRtaW46Y2lzY28=',
    'content-type': 'application/vnd.yang.data+json',
    'accept': 'application/vnd.yang.collection+json',
}

response = requests.request("PATCH", text, data=payload, headers=headers)
```
import requests

url = "https://10.1.1.1/api/running/native/class-map"

payload = '{
    "ned:class-map": [
        {
            "name": "http",
            "prematch": "match-all",
            "match": {
                "access-group": { "index": 101 }
            }
        },
        {
            "name": "all-ip",
            "prematch": "match-all",
            "match": {
                "access-group": { "index": 103 }
            }
        },
        {
            "name": "ssh",
            "prematch": "match-all",
            "match": {
                "access-group": { "index": 102 }
            }
        }
    ]
}

headers = {
    'authorization': 'Basic YWRtaW46Y2lzY28=',
    'content-type': 'application/vnd.yang.data+json",
    'accept': 'application/vnd.yang.collection+json",
}

response = requests.request("PATCH", url, data=payload, headers=headers)
Configuring The Service Path With RESTCONF

Policy Map

```python
import requests

url = "https://10.1.1.1/api/running/native/policy-map"

payload = "{\r\n"ned:policy-map": {\r\n"name": "sfc-pol", \r\n"type": "service-chain", \r\n{Name": "http", \r\n{\r\n"action-list": {
{\r\n"forward": {\r\n"forward": {\r\n"service-path": {\r\n"service-path-id": 1, \r\n3\r\n}"action-type": "forward": {\r\n"service-index": 3\r\n}}\r\n}}
headers = {
'authorization': "Basic YWRtaW46Y2lzY28=",
'content-type': "application/vnd.yang.data+json",
'accept': "application/vnd.yang.collection+json",
}

response = requests.request("PATCH", url, data=payload, headers=headers)
```
import requests

url = "https://10.1.1.1/api/running/native/policy-map"

payload = "{"ned:policy-map": ["pol", "type": "service-chain", "name": "http", "forward": "forward", "service-path": ["service-path-id": 1, "service-index": 3], "action-list": ["action-type": "forward", "service-path": ["service-path-id": 1, "service-index": 3]]}

headers = {
    'authorization': 'Basic YWRtaW46Y2lzY28=',
    'content-type': 'application/vnd.yang.data+json",
    'accept': 'application/vnd.yang.collection+json",
}

response = requests.request("PATCH", url, data=payload, headers=headers)
Configuring The Service Path With RESTCONF

Service Function

```python
import requests

url = "https://10.1.1.1/api/running/native/service-chain"

payload = "{
    "service-chain": {
        "service-function": {
            "name": "service1",
            "config-service-chain-sf-mode": {
                "description": "Service-Function-1",
                "encapsulation": {
                    "gre": {},
                    "ip": {
                        "address": "192.168.1.4"
                    }
                }
            }
        }
    }
}

headers = {
    'authorization': "Basic YWRtaW46Y2lzY28=",
    'content-type': "application/vnd.yang.data+json",
    'accept': "application/vnd.yang.data+json",
}

response = requests.request("PATCH", url, data=payload, headers=headers)
```
Configuring The Service Path With RESTCONF

Service Function

```python
import requests

url = "https://10.1.1.1/api/running/native/service-chain"

payload = "{ "service-chain": { "service-function": [ { "name": "service1", "config-service-chain-sf-mode": { "description": "Service-Function-1", "encapsulation": { "gre": { "enhanced": "divert" } }, "ip": { "address": "192.168.1.4" } } ] } }

headers = { 'authorization': "Basic YWRtaW46Y2lzY28=",
            'content-type': "application/vnd.yang.data+json",
            'accept': "application/vnd.yang.data+json",
        }

response = requests.request("PATCH", url, data=payload, headers=headers)
```
Configuring The Service Path With RESTCONF

Service Function Path

```python
import requests

url = "https://10.1.1.1/api/running/native/service-chain"

payload = "{"service-chain": {
    "service-path": [ {
        "service-path-id": "1",
        "config-service-chain-path-mode": {
            "description": "Service-Chain-1",
            "service-index": {
                "services": {
                    "service-index-id": "3",
                    "service-function": "service1"
                }
            }
        }
    }, {
        "service-index-id": "2",
        "services": {
            "terminate": {
                "service-index-id": "2"
            }
        }
    }
},
"services": {

headers = {
    'authorization': "Basic YWRtaW46Y2lzY28=",
    'content-type': "application/vnd.yang.data+json",
    'accept': "application/vnd.yang.data+json",
}

response = requests.request("PATCH", url, data=payload, headers=headers)
```
Configuring The Service Path With RESTCONF

Service Function Path

```python
import requests

url = "https://10.1.1.1/api/running/native/service-chain"

payload = "{ "service-chain": { "service-path": { "service-path-id": "1", "config-service-chain": { "description": "Service-Chain-1", "service": [ { "services": [ { "service-function": "service1" } ] }, { "service-index-id": "3", "terminate": {} } ] } }, { "service-index-id": "2", "terminate": { } } ] }

headers = {
    'authorization': 'Basic YWRtaW46Y2lzY28=',
    'content-type': 'application/vnd.yang.data+json",
    'accept': 'application/vnd.yang.data+json",
}

response = requests.request("PATCH", url, data=payload, headers=headers)
```
Our Service Function

• We are essentially building our own SF right now

• We handle terminating the GRE and processing the encapsulated traffic
  • We’re free to get creative in our language choice 😊

• Service function is a Python script that runs on Linux

• We use the Python `pcapy` module from

• GRE packets from the CSR are read using `libpcap`
def main(argv):
    if len(argv) < 1:
        print "Please specify a device on which to capture."
        sys.exit(1)

    dev = argv[1]

    print "Capturing on dev " + dev
    myip = commands.getoutput("/sbin/ifconfig " + dev).split("\n")[1].split()[1][5:]

    tap = pcapy.open_live(dev, 65536, 0, 100)
    tap.setfilter("src not " + myip)

    while (1):
        try:
            (hdr, pkt) = tap.next()
        except socket.timeout:
            continue
        else:
            parse_incoming(pkt)

Open a pcap live capture session, filtering on incoming packets.
if iproto == socket.IPPROTO_GRE:
    # GRE header:
    # Flags : 1 byte
    # Reserved Version : 1 byte
    # Protocol : 2 bytes
    #
    # NSH:
    # Version, OC bit, length : 2 bytes
    # MD Type : 1 byte
    # Next Protocol : 1 byte
    # Path ID : 3 bytes
    # Hop Index : 1 byte
    # Context Headers : 4 * 4 bytes = 16 bytes (for Type 1)
gend = glen + ipend
gpkt = pkt[ipend:gend]
ghdr = unpack("BBH", gpkt)
gproto = socket.ntohs(ghdr[2])
if frag_off == 0 and gproto == GREPROTO_NS:
    print "Received GRE-encapsulated NSH packet from " + src_addr + " to " + dst_addr

    nshend = nshlen + gend
    nshpkt = pkt[gend:nshend]

    nsh = unpack("=HBBBBBBLL", nshpkt)

    hop = nsh[6]

    print " NSH Header Type           : " + str(nsh[1])
    print " Encapsulated Protocol     : " + str(nsh[2])
    print " NSH Path ID               : " + str(pathid)
    print " NSH Hop Index             : " + str(hop)

    if nsh[1] == 1:
        print " NSH Context Header 0     : " + str(socket.ntohl(nsh[7]))
        print " NSH Context Header 1     : " + str(socket.ntohl(nsh[8]))
        print " NSH Context Header 2     : " + str(socket.ntohl(nsh[9]))
        print " NSH Context Header 3     : " + str(socket.ntohl(nsh[10]))

    if nsh[2] == NSH_IPV4:
        print "=== Parsing inner packet ==="
        parse_incoming(pkt[:elen] + pkt[nshend:], True)
Service Function Code (cont.)

Decrement the NSH hop index

```
... hop = hop - 1

new_nsh = pack("=HBBBBBLLLL", nsh[0], nsh[1], nsh[2], nsh[3], nsh[4], nsh[5], hop, nsh[7], nsh[8], nsh[9], nsh[10])
pkt = pkt[:gend] + new_nsh + pkt[nshend:]
```

We have to return the packet for the original flow to continue.

Create a new NSH header and stick it into the packet; then send it on its way

```
return_packet(iphdr, pkt)
...```

def return_packet(iph, pkt):
    new_iph = pack("BBHHHBBH4s4s", iph[0], iph[1], iph[2], iph[3], iph[4],
                    iph[5], iph[6], 0, iph[9], iph[8])

    pkt = new_iph + pkt[ipend:]

    s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_RAW)
    s.sendto(pkt, (socket.inet_ntoa(iph[8]), 0))
elif ip_proto == socket.IPPROTO_TCP:
    if do_print:
        print "TCP packet from " + src_addr + " to " + dst_addr
        tcp_end = ip_end + tcp_len
        tcp_pkt = pkt[ip_end:tcp_end]
        tcp_hdr = unpack("=HHLLBBHHH", tcp_pkt)

        if do_print:
            print " Src Port: " + str(socket.ntohs(tcp_hdr[0]))
            print " Dst Port: " + str(socket.ntohs(tcp_hdr[1]))

        offset = tcp_hdr[4] >> 4
        d_start = elen + ip_len + (offset * 4)

        if len(pkt) > d_start:
            if do_print:
                print " Packet data: "
                print "   " + pkt[d_start:] + ""
Making Things A Bit Easier

- Cisco is releasing a library to help developers create data path applications
- The Enhanced Service Function (ESF) library takes the place of the Service Function and allows one to focus on application development
- Easy to migrate existing onePK applications to ESF
Programming With ESF

• The Enhanced Service Function is a Linux C library that facilitates the encap and decap of SFC packets

• Supports a callback mechanism (upon packet receipt) or a file descriptor interface

• Supports NSH over GRE for packet encapsulation

• Provides a NETCONF interface to program the device

• Uses NSH MD-Type 2 metadata for control of flows (without the need for an additional process to manage flows)

• Allows for inspection of packets, inline modification, and injection of new flows
Classifying Traffic with ESF

• Let’s say we want to mark HTTP traffic with the DSCP value of Assured Forwarding 41

• Set up a class-map to match the traffic

• Define our Service Function with the enhanced GRE encapsulation and divert packets to it

• Build an ESF app to take the packets and perform the modification
Some NETCONF To Program The Classifier Device

```xml
<edit-config>
  <target>
    <running/>
  </target>
  <config>
    ...
    <service-function>
      <name>dscp-service</name>
      <config-service-chain-sf-mode>
        <description>SF to modify DSCP</description>
        <encapsulation>
          <gre>
            <enhanced>divert</enhanced>
          </gre>
        </encapsulation>
        <ip>
          <address>192.168.1.64</address>
        </ip>
      </config-service-chain-sf-mode>
    </service-function>
  </config>
</edit-config>
```

The config for the SF can be programmed from the ESF app itself using NETCONF.
Send the NETCONF Config

```c
rc = datapath_netconf_init(&datapath_ne, nc_addr,
                         830, username, password, NULL,
                         NULL, 0, &err);

if (rc != DATAPATH_OK) {
    printf("netconf_init returned %s, err=%ld\n",
            datapath_strerror(rc), err);
    datapath_strerror(rc), err);
    exit(1);
}

rc = datapath_netconf_write(&datapath_ne, SFC_CFG, strlen(SFC_CFG),
                            &err);
```

Send the NETCONF XML to configure the Service Function and related classifier parameters.
ESF Application Code

static inline void
modify_dscp_cb(struct datapath_paktype_ *pak, void * user_ctx) {
    datapath_error_t rc;
    uint8_t dscp;

    dscp = IPTOS_DSCP_AF41;

    rc = datapath_modify_packet(pak, DATAPATH_LAYER_3, 1, 1, (uint8_t *) &dscp, sizeof(uint8_t_t));
    if (rc != DATAPATH_OK) {
        fprintf(stderr, "Failed to modify the DSCP value of the packet '%s'
(%d)\n", datapath_strerror(rc), rc);
    }
    ...
datapath_set_service_path_index_handler(path_id, index, modify_dscp_cb, NULL);

Modify the ToS byte inline with the value of the “dscp” variable (in this case, AF41).

Register our callback with the desired SFC path and SF index
Modifying DSCP: The Results

Without the app running…
Modifying DSCP: The Results (cont.)

### Table: Wireshark Packet Capture

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>351</td>
<td>101.0.0.1.1</td>
<td>20.1.1.1.1</td>
<td>224.0.0.10.1</td>
<td>EIGRP</td>
<td>69</td>
<td>Hello</td>
</tr>
<tr>
<td>352</td>
<td>102.0.3.2004</td>
<td>20.1.1.1.1</td>
<td>224.0.0.10.1</td>
<td>EIGRP</td>
<td>69</td>
<td>Hello</td>
</tr>
<tr>
<td>353</td>
<td>102.5.32005</td>
<td>20.1.1.1.1</td>
<td>224.0.0.10.1</td>
<td>EIGRP</td>
<td>69</td>
<td>Hello</td>
</tr>
<tr>
<td>354</td>
<td>162.0.65002</td>
<td>20.1.1.1.1</td>
<td>224.0.0.10.1</td>
<td>EIGRP</td>
<td>69</td>
<td>Hello</td>
</tr>
<tr>
<td>355</td>
<td>163.6.160003</td>
<td>20.1.1.1.1</td>
<td>224.0.0.10.1</td>
<td>EIGRP</td>
<td>69</td>
<td>Hello</td>
</tr>
<tr>
<td>356</td>
<td>163.6.20606</td>
<td>192.168.1.1.1</td>
<td>192.168.2.1.1</td>
<td>TCP</td>
<td>60</td>
<td>49888 &gt; http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 SACK PERM=1 TS=764761386922618820478600</td>
</tr>
<tr>
<td>357</td>
<td>163.3.04004</td>
<td>192.168.1.1.1</td>
<td>192.168.2.1.1</td>
<td>TCP</td>
<td>60</td>
<td>49888 &gt; http [ACK] Seq=1 Ack=1 Win=4194240 Len=0</td>
</tr>
<tr>
<td>358</td>
<td>163.3.05999</td>
<td>28.1.1.1.1</td>
<td>224.0.0.10.1</td>
<td>EIGRP</td>
<td>69</td>
<td>Hello</td>
</tr>
<tr>
<td>359</td>
<td>163.3.06003</td>
<td>192.168.1.1.1</td>
<td>192.168.2.1.1</td>
<td>TCP</td>
<td>56</td>
<td>[TCP segment of a reassembled PDU]</td>
</tr>
<tr>
<td>360</td>
<td>164.0.61000</td>
<td>192.168.1.1.1</td>
<td>192.168.2.1.1</td>
<td>TCP</td>
<td>60</td>
<td>49888 &gt; http [ACK] Seq=183 Ack=183 Win=4194240 Len=0</td>
</tr>
<tr>
<td>361</td>
<td>164.0.22006</td>
<td>192.168.1.1.1</td>
<td>192.168.2.1.1</td>
<td>TCP</td>
<td>60</td>
<td>49888 &gt; http [FIN, ACK] Seq=183 Ack=183 Win=4194240 Len=0</td>
</tr>
<tr>
<td>362</td>
<td>164.0.22006</td>
<td>192.168.1.1.1</td>
<td>192.168.2.1.1</td>
<td>TCP</td>
<td>60</td>
<td>49888 &gt; http [ACK] Seq=183 Ack=183 Win=4194240 Len=0</td>
</tr>
<tr>
<td>363</td>
<td>164.3.480004</td>
<td>20.1.1.1.1</td>
<td>224.0.0.10.1</td>
<td>EIGRP</td>
<td>69</td>
<td>Hello</td>
</tr>
</tbody>
</table>

### Diagram: Packet Analysis

- **Layer 4: Transport Layer**
  - **TCP**: Establishing a connection with SYN and ACK
  - **HTTP**: Requesting a page with GET method

- **Layer 3: Network Layer**
  - **IP**: routing packets between networks

- **Layer 2: Data Link Layer**
  - **Ethernet**: Frame transmission over the network

---

...Now with the app running.
Additional ESF Actions

Packet Actions

• Inspect
• Divert and Modify
• Inject New

Flow Actions

• Drop
• Bypass
• L4 (IP and port) Redirect
• Set New Next Hop
• Mark DSCP
What About Performance?

- The ESF functionality (NSH over GRE) can be combined with VPP/FD.io to deliver high performance
- Leverage VPP’s plugin interface to introduce a new data path application
  - Plugins can be built independently of VPP source tree
  - Plugins can be added at runtime (drop into plugin directory)
- ESF’s MD-Type 2 TLVs are provided as part of the documentation
FD.io Plugins

- Plugins for VPP are written in C, and designed to be fast
- Plugins insert a node in the VPP graph, can add CLI and API elements for control, and tracing for debugging
- Sample MACSWAP is included with the VPP distribution as well as an NSHoGRE node
- FD.io offers a YouTube channel http://cs.co/9002Bak3e that provides excellent details
  - In particular http://cs.co/9008Bak3Q and http://cs.co/9007BakO3 walk through the MACSWAP plugin
FD.io MACSWAP Plugin

```c
vlib_get_next_frame (vm, node, next_index, 
                   to_next, n_left_to_next);

while (n_left_from >= 4 && n_left_to_next >= 2) 
{
    u32 next0 = SAMPLE_NEXT_INTERFACE_OUTPUT;
    u32 next1 = SAMPLE_NEXT_INTERFACE_OUTPUT;
    u32 sw_if_index0, sw_if_index1;
    u8 tmp0[6], tmp1[6];
    ethernet_header_t *en0, *en1;
    u32 b0, b1;
    vlib_buffer_t *b0, * b1;

    /* Prefetch next iteration. */
    { vlib_buffer_t * p2, * p3;

        p2 = vlib_get_buffer (vm, from[2]);
        p3 = vlib_get_buffer (vm, from[3]);

        vlib_prefetch_buffer_header (p2, LOAD);
        vlib_prefetch_buffer_header (p3, LOAD);

        CLIB_PREFETCH (p2->data, CLIB_CACHE_LINE_BYTES, STORE);
        CLIB_PREFETCH (p3->data, CLIB_CACHE_LINE_BYTES, STORE);
    }

    /* speculatively enqueue b0 and b1 to the current next frame */
    to_next[0] = b0: = from[0];
    to_next[1] = b1: = from[1];
    from += 2;
    to_next += 2;
    n_left_from -= 2;
    n_left_to_next -= 2;

    b0 = vlib_get_buffer (vm, b0);
    b1 = vlib_get_buffer (vm, b1);
}```

Dual-loop while reading packets from the vector

Optimized on the assumption that there will be more than one packet to read and packet 1 will go to the same next node as packet 2
FD.io MACSWAP Plugin (cont.)

```
node.c

en0 = vlib_buffer_get_current (b0);
en1 = vlib_buffer_get_current (b1);

/* This is not the fastest way to swap src + dst mac addresses */
#define _ (a) tmp0[a] = en0->src_address[a];
    foreach_mac_address_offset;
#undef _
#define _ (a) en0->src_address[a] = en0->dst_address[a];
    foreach_mac_address_offset;
#undef _
#define _ (a) en0->dst_address[a] = tmp0[a];
    foreach_mac_address_offset;
#undef _
#define _ (a) tmp1[a] = en1->src_address[a];
    foreach_mac_address_offset;
#undef _
#define _ (a) en1->src_address[a] = en1->dst_address[a];
    foreach_mac_address_offset;
#undef _
#define _ (a) en1->dst_address[a] = tmp1[a];
    foreach_mac_address_offset;
#undef _
```

Swap the src and dst MAC addresses.

The packets will then be hair pinned back on the input interface.
FD.io MACSWAP Plugin (cont.)

node.c

```c
/* packet trace format function */
static u8 * format_sample_trace (u8 * s, va_list * args)
{
    CLIB_UNUSED (vlib_main_t * vm) = va_arg (*args, vlib_main_t *);
    CLIB_UNUSED (vlib_node_t * node) = va_arg (*args, vlib_node_t *);
    sample_trace_t * t = va_arg (*args, sample_trace_t *);

    s = format (s, "SAMPLE: sw_if_index %d, next index %d",
                t->sw_if_index, t->next_index);
    return s;
}
```

Add custom tracing for troubleshooting packets through the graph.

sample.c

```c
#include "sample_macswap.h"

VLIB_CLI_COMMAND (sr_content_command, static) = {
    .path = "sample macswap",
    .short_help = "sample macswap <interface-name> [disable]",
    .function = macswap_enable_disable_command_fn,
};
```

Provide CLI for controlling the plugin.
FD.io MACSWAP Plugin (cont.)

Custom CLI

Flexible tracing
ESF MD-Type 2 Metadata

Generic TLV

Class = Cisco (0x9)

Tag Type: 0: no tag type
1: VLAN ID
2: VXLAN ID
3: Security Group Tag (SGT)
4: Endpoint Group (EPG)
5: Segment
6: Zone
7: VRF ID (number)
8: Security context ID

S bit: Indicates that source or ingress tag is present.
D bit: Indicates that destination or egress tag is present.
# ESF MD-Type 2 Metadata

## Packet Metadata

<table>
<thead>
<tr>
<th>Class = Cisco (0x9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
</tr>
<tr>
<td>Type = Packet MD (0x50)</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>Length = 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action Type</th>
<th>Location ID</th>
<th>L2 / L3 Flags</th>
<th>Meta Flags (implementation specific)</th>
<th>Interface ID</th>
<th>Rx Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Action Type**:
- 1 = PUNT
- 2 = COPY
- 3 = DIVERT

**Location ID**:
- 1 = Hardware defined INPUT
- 2 = Hardware defined OUTPUT
- 3 = Hardware defined PRE_ROUTING
- 4 = Hardware defined POST_ROUTING

**L2 / L3 Flags**:
- 0x1 = IPv4
- 0x2 = IPv6
- 0x4 = MPLS

**Interface ID**: Interface from which packet was diverted

**D bit**: Packet direction (existing flow direction or reverse direction)

**X bit**: Set if it is a control frame (may miss some headers)
## ESF MD-Type 2 Metadata
### Flow ID

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Type = Flow ID (0x2A)</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Length = 1 / 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Class = Cisco (0x9)**
- **Flow ID**
- **Rest of Flow ID (if 64-bit)**
## ESF MD-Type 2 Metadata

### Injection Info

| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |

<table>
<thead>
<tr>
<th>Class = Cisco (0x9)</th>
<th>Type = Injection Info (0x51)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3 Start</td>
<td>L4 Start</td>
</tr>
</tbody>
</table>

### Physical Interface ID

### Logical Interface ID

<table>
<thead>
<tr>
<th>Location ID</th>
<th>Reserved</th>
</tr>
</thead>
</table>

L3 Start is for L2 injected packets, otherwise 0

Location ID:
1 = Hardware defined INPUT
2 = Hardware defined OUTPUT
3 = Hardware defined PRE_ROUTING
4 = Hardware defined POST_ROUTING
# ESF MD-Type 2 Metadata

## Flow Action – Drop, Purge, or Bypass Flow

<table>
<thead>
<tr>
<th>OT</th>
<th>C</th>
<th>T</th>
<th>S</th>
<th>D</th>
<th>R</th>
<th>A</th>
<th>H</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Class = Cisco (0x9)  
Type = Flow Action: DPB (0x54)

| Length = 2 |
| Bypass Mode |

### OT : Offload Type:
- 1 = Pass Through
- 2 = Drop

### C = Security Checks
- T = Termination (for Bypass flow)
- S = Reset Source (for Drop)
- D = Reset Destination (for Drop)
- R = Reset Subsequent (for Drop)
- A = ACK Subsequent (for Purge)
- H = Deny Source
- U = Deny Destination

### Bypass Mode values:
- 1 = bypass until packet number
- 2 = bypass until number of bytes reached
- 3 = bypass until time reached

### Bypass Until Param is dependent upon Bypass Mode:
- number of packets (packet mode),
- number of application layer (L7) bytes (bytes mode) or
- number of milliseconds (time mode)
## ESF MD-Type 2 Metadata

### Flow Action – DSCP and Precedence

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

**Class = Cisco (0x9)**

<table>
<thead>
<tr>
<th>QoS Marking</th>
<th>Q</th>
<th>Reserved</th>
</tr>
</thead>
</table>

**Q bit:** 0 for DSCP, 1 for Precedence
## ESF MD-Type 2 Metadata

### Flow Action – Next Hop IPv4

<table>
<thead>
<tr>
<th>Class = Cisco (0x9)</th>
<th>Type = Flow Action: NHv4 (0x55)</th>
<th>Next Hop IPv4 Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>R R R</td>
<td>Length = 1</td>
</tr>
</tbody>
</table>

For Your Reference
ESF MD-Type 2 Metadata

Flow Action – Next Hop IPv6

Class = Cisco (0x9)  

Type = Flow Action: NHv6 (0x56)  

Length = 4

Next Hop IPv6 Address
Example Use Cases
SFC+NSH Dump Packets For Wireshark

• Take the encapsulated packets obtained from ESF (GRE encapsulated) stream and write them to a PCAP file

• Run wireshark in a way so that it streams the contents of the file
  • `tail -f -c +0b FILENAME.pcap | wireshark -k -i -` (Linux)
  • `tail -f -c +0 FILENAME.pcap | wireshark -k -i -` (FreeBSD)

• This is an easier way of tying SFC+NSH to wireshark without modifying the wireshark code

• Other applications could be used to perform deep packet inspection or analytics on the traffic
The PCAP Code

```c
pd = pcap_open_dead(DLT_RAW, 65535);
pdumper = pcap_dump_open(pd, pcap_file);
datapath_set_service_path_index_handler(path_id, index,
                                        pcap_packet_cb, NULL);
```

Open a PCAP file in raw mode to write raw IP frames (i.e., no L2 header)
The PCAP Code (cont.)

```c
static inline void pcap_packet_cb (struct datapath_paktype_ *pak, void *user_ctx) {
    datapath_error_t rc;
    uint8_t *l3_start;

    rc = datapath_pkt_get_l3_start(pak, &l3_start);
    if (rc == DATAPATH_OK) {
        struct pcap_pkthdr phdr;
        uint32_t l3_len;
        struct ip *ih;

        ih = (struct ip *) l3_start;
        l3_len = ntohs(ih->ip_len);

        gettimeofday(&phdr.ts, NULL);
        phdr.caplen = l3_len;
        phdr.len = l3_len;

        pcap_dump((u_char*) pdumper, &phdr, (const u_char *) l3_start);
        pcap_dump_flush(pdumper);
    }
}
```

Take the L3 (IP) packet, and write it to the open PCAP file.
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.001189</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>TCP</td>
<td>52</td>
<td>55600 [ACK] Seq=56000 Ack=56000</td>
</tr>
<tr>
<td>3</td>
<td>0.028859</td>
<td>102.168.1.4</td>
<td>10.10.10.10</td>
<td>TCP</td>
<td>52</td>
<td>55600 [ACK] Seq=56000 Ack=56000</td>
</tr>
<tr>
<td>5</td>
<td>0.028593</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>HTTP</td>
<td>52</td>
<td>GET /favicon.ico HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>6</td>
<td>0.028593</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>HTTP</td>
<td>52</td>
<td>GET /favicon.ico HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>7</td>
<td>0.028593</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>HTTP</td>
<td>52</td>
<td>GET /favicon.ico HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>8</td>
<td>0.028593</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>HTTP</td>
<td>52</td>
<td>GET /favicon.ico HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>9</td>
<td>0.028593</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>HTTP</td>
<td>52</td>
<td>GET /favicon.ico HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>10</td>
<td>0.028593</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>HTTP</td>
<td>52</td>
<td>GET /favicon.ico HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>11</td>
<td>0.028593</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>HTTP</td>
<td>52</td>
<td>GET /favicon.ico HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>12</td>
<td>0.028593</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>HTTP</td>
<td>52</td>
<td>GET /favicon.ico HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>13</td>
<td>0.028593</td>
<td>10.10.10.10</td>
<td>192.168.1.4</td>
<td>HTTP</td>
<td>52</td>
<td>GET /favicon.ico HTTP/1.1 200 OK</td>
</tr>
</tbody>
</table>

ESF Is Awesome!
Load Balancing

- Allow packets or flows to be sent along different paths based on dynamic policy
  - Simple per-IP
  - More complex such as determining source user

- Two approaches can be used
  - Modify packet headers (think NAT)
  - Modify the next hop IP for the flow

- ESF supports both solutions
Load Balancing Topology

- CSR1Kv
- Video Servers
- Video Clients
- Service Function

GRE
Modify Packet Headers

- In Solution 1, we create a virtual IP on the router to which clients will send requests.
- We classify based on packets to that IP.
- Use two Service Paths for client-to-server and for server-to-client traffic.
- Modify the source and destination IPs accordingly.

VIP: 192.168.2.132

192.168.1.5

192.168.1.4

192.168.2.235

192.168.2.236
Service Function Configuration

```plaintext
service-chain service-function load-balance-service
  ip address 192.168.1.64
  encapsulation gre enhanced divert
!
service-chain service-path 1
  service-index 2 service-function load-balance-service
service-chain service-path 2
  service-index 2 service-function load-balance-service
!
class-map match-all client-to-server
  match access-group 104
class-map match-all server-to-client
  match access-group 105
!
access-list 104 permit ip 192.168.1.0 0.0.0.255 192.168.2.0 0.0.0.255
access-list 105 permit ip 192.168.2.0 0.0.0.255 192.168.1.0 0.0.0.255
```

static inline void
transform_cb (struct datapath_pkttype_ *pak, void * user_ctx) {
struct in_addr xform_addr;
        const char *xform_ip;
        int direction, path_id, index, offset;
        uint8_t *l3_start;
        struct ip *ih;

datapath_pkt_nsh_get_pathid_index(pak, &path_id, &index);

datapath_pkt_get_l3_start(pak, &l3_start);
        ih = (struct ip *) l3_start;

        if (path_id == DIRECTION_CLIENT_TO_SERVER) {
                xform_ip = real_ips[ntohl((ih->ip_src).s_addr) % sizeof(real_ips) / (sizeof(char *))];
                offset = 16;
        } else {
                xform_ip = VIP_IP;
                offset = 12;
        }

        inet_pton(AF_INET, xform_ip, &xform_addr);
        datapath_modify_packet(pak, DATAPATH_LAYER_3, offset, sizeof(in_addr_t), (uint8_t *)
                &xform_addr.s_addr), sizeof(in_addr_t);}

Load balance based on source IP and adjust the header based on direction
Modify Flow Next Hop

- In Solution 2, each video server has a loopback with the same IP
- The IP next hop is switched based on client IP
- Only one Service Path is needed, and no virtual IP
Service Function Configuration

```plaintext
gre enhanced divert

! service-chain service-path 1
  service-index 2 service-function load-balance-service
!
class-map match-all video
  match access-group 104
!
access-list 104 permit ip 192.168.1.0 0.0.0.255 host 10.10.10.10
```
static inline void 
next_hop_cb (struct datapath_paktype_ *pak, void * user_ctx) { 
  uint8_t *l3_start;
  struct sockaddr_in sin;
  struct ip *ih;

  datapath_pkt_get_l3_start(pak, &l3_start);
  ih = (struct ip *) l3_start;

  sin.sin_family = AF_INET;
  sin.sin_port = 0;

  inet_pton(AF_INET, nh_ips[nthol((ih->ip_src).s_addr) %
      (sizeof(nh_ips) / sizeof(char *))], &sin.sin_addr);
  datapath_assign_next_hop(pak, (struct sockaddr *) &sin);
}
Now It’s Your Turn!
Where To Go Now

• Go to the DevNet Zone to see more about device and network programmability.

• Read the SFC wiki at https://wiki.opendaylight.org/view/Service_Function_Chaining:Main to get started with SFC+NSH within ODL

• Learn more about Vector Packet Processing and the Fast Data Project from https://fd.io and look for VPP modules coming to dCloud

• Start to prototype applications that take advantage of software-defined packet control

• Connect with developers and ask device programmability questions in the DNA community of interest at https://developer.cisco.com/site/dna-community/
Complete Your Online Session Evaluation

• Give us your feedback to be entered into a Daily Survey Drawing. A daily winner will receive a $750 Amazon gift card.

• Complete your session surveys through the Cisco Live mobile app or from the Session Catalog on CiscoLive.com/us.

Don’t forget: Cisco Live sessions will be available for viewing on-demand after the event at CiscoLive.com/Online
Please join us for the Service Provider Innovation Talk featuring:

Yvette Kanouff | Senior Vice President and General Manager, SP Business
Joe Cozzolino | Senior Vice President, Cisco Services

Thursday, July 14th, 2016
11:30 am - 12:30pm, In the Oceanside A room

What to expect from this innovation talk

• Insights on market trends and forecasts
• Preview of key technologies and capabilities
• Innovative demonstrations of the latest and greatest products
• Better understanding of how Cisco can help you succeed

Register to attend the session live now or watch the broadcast on cisco.com
Continue Your Education

• Demos in the Cisco campus
• Walk-in Self-Paced Labs
• Lunch & Learn
• Meet the Engineer 1:1 meetings
• Related sessions
  • DevNet Zone
What Solutions Will You Build With
Thank you