SDN Protocols in Internet

BRKSPG-2516

Josef Ungerman, CCIE #6167
Agenda

• Need for Automation
• Overview of Protocols and API’s
• Openflow, OnePK, Yang
• Open Data Models – Yang, Netconf/Restconf
• IETF protocols SDN evolution – BGP, MPLS
• Summary
“Civilization advances by extending the number of important operations which we can perform without thinking of them.”

Alfred North Whitehead, 1911

(Mathematician and Philosopher, Author of “An Introduction to Mathematics” and “Principia Mathematica”)
Need for Automation and Programmability

**Reduce CAPEX**
“Move bits cheaper!”
Network De-layering, TDM → Packet, Multi-Layer Optimization

**Reduce OPEX and R&D**
“Stop networks getting more complicated!”
Automation, Simplification, Open Standard API’s

**Monetize the Network**
“Deliver quickly what your customers want!”
Agility with NfV – Faster time to revenue, Risk reduction

**SDN** (Software-Defined Networking) makes the network **programmable**.
Real-time automation via closed loop of network data collection, analytics, and policy programming.

New Services are tough – long time to revenue, high risk, high adoption (20-30%) needed to be profitable,…

Caution: requires API’s for control plane offload
Caution: requires new programmability concepts
Caution: requires open API’s between physical & virtual
Segmentation of Programmability Interfaces

Configure
Operate

Device
Extension

DevOps
Integration

Internet Routing &
Carrier Ethernet

NETCONF
YANG
IETF
IEEE

BGP-LS
PCEP
IETF

BGP
Flowspec
IETF

onePK

OpenFlow

OpFlex

NX-OS
Python API

REST API
REST/JSON

OPEN
DAYLIGHT
RESTCONF

Chef

puppet
labs

Python

openstack

© 2015 Cisco and/or its affiliates. All rights reserved.

Cisco Public
SDN – Software Defined Networking
Evolve the Control- and Management Plane Architecture

**Traditional Control Plane Architecture**

- Distributed Control Plane

**Evolved Control Plane Architecture**

- Centralized SDN → DC
- Hybrid SDN → SP

**SDN Definition (ONF):** The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices.

<table>
<thead>
<tr>
<th>Control/Network/Services-plane component(s)</th>
<th>ASIC's, Data-plane component(s)</th>
<th>Control &amp; RT-Automation Applications</th>
</tr>
</thead>
</table>

API’s

New Protocols

Innovated Protocols

Overlay Tunnels

Underlay (physical)
SDN Protocols – Focus on SP (Internet & Carrier Ethernet)
SDN Protocols: Agenda

Application Frameworks, Management Systems, Controllers, ...

Protocols
C, Java, Py
NETCONF
REST
OpenFlow
ACI Fabric
OpenStack
DevOps

Management
Orchestration
Network Services
Control
Forwarding
Device

BGP, IGP, TE, ...

I2RS
IPSE

OpFlex
Plug-in’s

API’s
(OnePK)

Data Models
(Yang)

Operating Systems – IOS / NX-OS / IOS-XR

where SDN started
Openflow Protocol Summary

Openflow v1.0
- Initial Standard – still prevalent in the market today

Openflow v1.1
- Added support for multiple flow tables
- Added support for MPLS
- Defines two operating modes – Hybrid | Pure Openflow

Openflow v1.2
- Adds support for IPv6

Openflow v1.3
- Adds support for Rate Limiting | IPv6 extension headers
- GRE – The version deemed *production ready*

Openflow v1.3.x
- Adds Negotiation TLV’s, bug fixes

Openflow v1.4
- Extensibility, bundles, tcp/6633→6653, improvements…

Openflow v1.5
- (Dec’14) Egress pipeline, TCP flags, packet types…
Openflow

Pipeline – this is how L2/L3 ASIC works inside
– series of TCAM lookups

- Symmetric Sync Messages (Hello, Echo, Vendor…)
- Async Messages (Port-Status, Flow-Removed, Error…)
- Forwarding Control & Stats Collection

OF 1.1+: New ASIC design required
OF 1.5+: Egress pipeline
### OpenFlow

**FLOW TABLE**

<table>
<thead>
<tr>
<th>HEADER FIELDS</th>
<th>COUNTERS</th>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Required Counters:**
- per Table, Flow, Queue, Port
- Bytes, Packets, Errors, Flow duration…
- Counter Disable (1.3)

**Required Matches (N-tuple):**
- Ingress Port
- Source MAC
- Dest MAC
- Ether Type
- VLAN ID
- VLAN Priority
- MPLS Label
- MPLS Traffic Class
- IP SRC
- IP DEST
- IP Protocol
- IP TOS
- TCP/UDP SRC
- TCP/UDP DEST
- ICMP Type
- ICMP Code

**Required Actions (1.0):**
1. Forward out all ports except input port
2. Redirect to Openflow Controller
3. Forward to local Forwarding Stack (CPU)
4. Perform action in flow table
5. Forward to input port
6. Forward to destination port
7. Drop Packet

**Optional Actions** – Modify-Field, Enqueue, Forward Normally…
OF: describe generalized forwarding primitives

Made for a pipelining L2/L3-switching ASIC (common in DC and LAN)

<table>
<thead>
<tr>
<th>Match Fields</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>OXM_OF_IN_PORT</td>
<td>Output</td>
</tr>
<tr>
<td>OXM_OF_IN_PHY_PORT</td>
<td>Port</td>
</tr>
<tr>
<td>OXM_OF_METADATA</td>
<td>OFPP_IN_PORT</td>
</tr>
<tr>
<td>OXM_OF_ETH_DST</td>
<td>OFPP_NORMAL</td>
</tr>
<tr>
<td>OXM_OF_ETH_SRC</td>
<td>OFPP_FLOOD</td>
</tr>
<tr>
<td>OXM_OF_ETHETYPE</td>
<td>OFPP_ALL</td>
</tr>
<tr>
<td>OXM_OF_VLAN_VID</td>
<td>OFPP_CONTROLLER</td>
</tr>
<tr>
<td>OXM_OF_VLAN_PCP</td>
<td>OFPP_LOCAL</td>
</tr>
<tr>
<td>OXM_OF_IPV6_DST</td>
<td>Set-Queue</td>
</tr>
<tr>
<td>OXM_OF_IPV6_FLABEL</td>
<td>Drop</td>
</tr>
<tr>
<td>OXM_OF_ICMPV6_TYPE</td>
<td>Group</td>
</tr>
<tr>
<td>OXM_OF_IP_PROTO</td>
<td>Push-Tag/Pop-Tag</td>
</tr>
<tr>
<td>OXM_OF_IPV6_ND_TARGET</td>
<td>Push VLAN header</td>
</tr>
<tr>
<td>OXM_OF_IPV6_ND_SLL</td>
<td>Pop VLAN header</td>
</tr>
<tr>
<td>OXM_OF_IPV6_ND_TLL</td>
<td>Push MPLS header</td>
</tr>
<tr>
<td>OXM_OF_TCP_SRC</td>
<td>Pop MPLS header</td>
</tr>
<tr>
<td>OXM_OF_MPLS_LABEL</td>
<td>Push PBB header</td>
</tr>
<tr>
<td>OXM_OF_MPLS_TC</td>
<td>Pop PBB header</td>
</tr>
<tr>
<td>OXM_OF_MPLS_BOS</td>
<td>Change-TTL</td>
</tr>
<tr>
<td>OXM_OF_MPLS_PBB_ISID</td>
<td>Set MPLS TTL</td>
</tr>
<tr>
<td>OXM_OF_TUNNEL_ID</td>
<td>Decrement MPLS TTL</td>
</tr>
<tr>
<td>OXM_OF_IPV6_EXTHDR</td>
<td>Set IP TTL</td>
</tr>
<tr>
<td>OXM_OF_ICMPV4_TYPE</td>
<td>Decrement IP TTL</td>
</tr>
<tr>
<td>OXM_OF_ICMPV4_CODE</td>
<td>Copy TTL outwards</td>
</tr>
</tbody>
</table>

Very low-level and complex
• start from scratch

What if I want just something simple?
• look for something else

Example: Catalyst 3850, 6800
• UADP ASIC – OF1.3 h/w support
• OF: 17K entries (Flows)
• FIB: 80K entries (IP Routes)
Software can’t control what hardware can’t deliver.
Internet Router

Programmable NPU – not pipeline

Yes, it can emulate the OF pipeline
- NPU is fully programmable and has TCAM

Cisco ASR9000 – XR 5.1.2
- 8 OF switches
- 4 types: L2, L3, IPv4, dual-stack

Example: ASR 9000
- Typhoon NPU: OF1.3 h/w support
- OF: 30K entries per LC (Flows)
- FIB: 4M entries (IP Routes)
Internet Router – Specific API’s & Protocols

Router has different tables to control

• Higher-level abstraction

If I want something simple…
• add route, delete route
• add VRF, add ACL
• create new LSP/Tunnel…

Routing Protocols

- BGP-LS/FS, SR
- I2RS
- PCEP, I2RS
- IPSE, OVSDB
- Openflow focus

- RIB
- FIB
- Platform Specific Control Plane
  - Features (QoS, ACL, Netflow…)
  - Interface Mgmt.
- Drivers, HAL, SDK

SDN Controller

Router FORWARDING ENGINE (NPU)
OnePK (Open Networking Environment – Platform Kit)

Open API and SDK for C and Java (ev. Python)
- API’s (functions) grouped in Service Sets (libraries)
- Low-level programming

Router FORWARDING ENGINE (NPU)

Platforms Specific Control Plane

Routing Protocols
- BGP
- OSPF
- ISIS
- LDP
- RIB
- FIB
- Features (QoS, ACL, Netflow...)

- Topology SS (discovery)
- Routing SS (RIB read, write, notify)
- Policy SS (ACL, QoS,...)
- Element & Interface SS (interfaces, CPU,...)
- Data Path SS (match, set. punt...)

Drivers, HAL, SDK

Thrift
RPC message framework (communication library)
SDN Protocols: Agenda

Application Frameworks, Management Systems, Controllers, ...

Protocols

C, Java, Py

NETCONF

REST

OpenFlow

ACI Fabric

OpenStack

DevOps

Management

Orchestration

Network Services

Control

Forwarding

Device

BGP, IGP, TE,...

I2RS

IPSE

SSH

HTTP

API’s (OnePK)

Data Models (Yang)

Operating Systems – IOS / NX-OS / IOS-XR

Plug-in’s

Puppet/Chef

Neutron

OpFlex

OpenFlow

OpFlex

Data Models

Yang

API’s

OnePK

IPSE

I2RS

BGP, IGP, TE...

Control

Forwarding

Device

Management

Orchestration

Network Services

Service Provider

Enterprise
Programmable Interface to Router

IETF

• I2RS (Interface to the Routing System) – RIB & Co.
  – IETF (I2RS WG): draft-clemm-i2rs-yang-network-topo, etc.
  – Asynchronous, high throughput, multi-channel, full-duplex…
  – Data Models: converged to Yang (self-describing semantics)

• IPSE (Interface to Packet Switching Element) – FIB & Co.
  – IETF: draft-rfernando-ipse
  – Interface between CP (Control Plane) and DP (Data Plane)
    • implemented on Virtual PE (VTS), usable with ASIC’s as well
  – Data Models: Yang modules for FIB and other fwd’ing tables
    • interface-table
    • l2-table, l2tp-table, flow-table
    • ip-unicast-table, label-table, context-selector-table (VRF)
    • arp-table, arp-proxy-table
    • pse-oam
vRouter and Virtual Data Plane

NfV Solutions

• Virtual Data Plane
  – Networking Stack on top of Intel x86 (utilizing DPDK)
  – OF/OVSDB is not optimal to control this
  – IPSE – REST/Yang API between CP and DP

• Cisco VTS (Virtual Topology System)
  – Distributed virtual router (400 forwarders tested)
  – Today, optimized for NfV (managed services chaining)
  – Evolution towards features and scale
  – Integration with Physical Networking
    • 1.0: MPLS (in GRE) and MP-BGP (L3VPN)
    • 2.0: VXLAN and EVPN (L2VPN)
YANG

YANG (RFC6020 etc.) is a data modelling language
- “Yet Another Next Generation” – developed after the NG-SNMP fiasco
- Data modelling for NETCONF protocol, and more: XML conversion, visualization, code generation...
- NETCONF/YANG’s ambition is to provide the ultimate multi-vendor R/W element and service management, thus an open standard programmability architecture

YANG vs. SNMP

<table>
<thead>
<tr>
<th></th>
<th>YANG</th>
<th>SNMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework</td>
<td>YANG</td>
<td>SMI</td>
</tr>
<tr>
<td>(definition language)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content</td>
<td>YANG module</td>
<td>MIB</td>
</tr>
<tr>
<td>(information model)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td>XML (ASCII)</td>
<td>ASN.1 BER</td>
</tr>
<tr>
<td>(encoded data)</td>
<td></td>
<td>(binary)</td>
</tr>
<tr>
<td>Protocol</td>
<td>NETCONF,</td>
<td>SNMP</td>
</tr>
<tr>
<td>(remote access)</td>
<td>RESTCONF</td>
<td></td>
</tr>
</tbody>
</table>

For example, I can use YANG to describe:
- a data structure of a routing protocol
- RIB, FIB, ARP table, MAC table...
- Anything we use „show command“ or MIB for

Or also:
- a communication protocol (header fields, options...)

And also a Service:
- MEF E-Line – UNI attributes
- IP VPN service – contract, SLA, QoS...

IOS XR Yang Support: XR-XML (4.3), XR-YANG (5.3.x)
 iTf-interfaces.yang – YANG module example:

```yang
container interfaces {
    description "Interface configuration parameters.";

    list interface { key "name";
        leaf name { type string; }
        leaf description { type string; }
        leaf type {
            type identityref {
                base interface-type;
            } mandatory true;
        }
        leaf enabled {
            type boolean;
            default "true";
        }
        leaf link-up-down-trap-enable {
            if-feature if-mib;
            type enumeration {
                enum enabled { value 1; }
                enum disabled { value 2; }
            }
            mandatory true;
        }
    }
}
```

Tree data structure
Leaf carries data
iietf-ip.yang

---rw if:interfaces
   ---rw if:interface* [name]
   ...
   ---rw ipv4
      | ---rw enabled? boolean
      | ---rw forwarding? boolean
      | ---rw mtu? uint16
      | ---rw address* [ip]
      | | ---rw ip inet:ipv4-address-no-zone
      | | | ---:(prefix-length)
      | | | | ---rw ip:prefix-length? uint8
      | | | ---:(netmask)
      | | | | ---rw ip:netmask? yang:dotted-quad
      | ---rw neighbor* [ip]
      | ---rw ip inet:ipv4-address-no-zone
      | ---rw link-layer-address yang:phys-address
---rw ipv6!
   ---rw enabled? boolean
   ...

---rw if:interfaces
   ---rw if:interface* [name]
   ...
   ---rw ipv4
      | ---rw enabled? boolean
      | ---rw forwarding? boolean
      | ---rw mtu? uint16
      | ---rw address* [ip]
      | | ---rw ip inet:ipv4-address-no-zone
      | | | ---:(prefix-length)
      | | | | ---rw ip:prefix-length? uint8
      | | | ---:(netmask)
      | | | | ---rw ip:netmask? yang:dotted-quad
      | ---rw neighbor* [ip]
      | ---rw ip inet:ipv4-address-no-zone
      | ---rw link-layer-address yang:phys-address
---rw ipv6!
   ---rw enabled? boolean
   ...

---rw if:interfaces
   ---rw if:interface* [name]
   ...
   ---rw ipv4
      | ---rw enabled? boolean
      | ---rw forwarding? boolean
      | ---rw mtu? uint16
      | ---rw address* [ip]
      | | ---rw ip inet:ipv4-address-no-zone
      | | | ---:(prefix-length)
      | | | | ---rw ip:prefix-length? uint8
      | | | ---:(netmask)
      | | | | ---rw ip:netmask? yang:dotted-quad
      | ---rw neighbor* [ip]
      | ---rw ip inet:ipv4-address-no-zone
      | ---rw link-layer-address yang:phys-address
---rw ipv6!
   ---rw enabled? boolean
   ...

Augment example:

```
container ipv4 {
  leaf enabled { type boolean; }
  leaf forwarding {type boolean; }
  leaf mtu {
    type uint16 { range "68..max"; }
    units octets;
  }
}
```

List example:

```
list address {
  leaf ip {
    type inet:ipv4-address-no-zone;
  }
}
choice subnet {
  mandatory true;
  leaf prefix-length {
    type uint8 { range "0..32"; }
  }
  leaf netmask {
    type yang:dotted-quad;
  }
}
```

**iietf-ip.yang**

- **Nice & Easy extensibility**
- Choice (radio button)
  - unlike MIB or XSD
- **RW (config) and RO (operational) data all in one**
  - one definition is reusable by many operations like <get>, <get-config>, <edit-config>,... (unlike XSD)
Netconf example with ietf-interfaces.yang & ietf-ip.yang

<get> request – “sh run int eth0”

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
        <source>
            <running/>
        </source>
        <filter xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces"
                type="xpath"
                select="/if:interfaces/if:interface[if:name='eth0']"/>
    </get-config>
</rpc>
```

<get> reply – XML-encoded YANG data example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
    <data>
        <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
            <interface>
                <name>eth0</name>
                <type>ethernetCsmacd</type>
                ...
                <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                    <address>
                        <ip>192.0.2.1</ip>
                        <prefix-length>24</prefix-length>
                    </address>
                </ipv4>
            </interface>
        </interfaces>
    </data>
</rpc-reply>
```
**Netconf**

**NETCONF (Network Configuration Protocol)**

- IETF protocol for configuration data and operational state data management & notifications
- Addresses SNMP SMI short-comings like no transactions, no backup&restore, strange&legacy concepts…
- Based on XML (Yang to provide the data modeling)
- 830/tcp – runs over SSH (possibly also TLS, BEEP…)

**Example: ASR9000 – XR 5.3.0**

- ssh server netconf port 830
- netconf-yang agent ssh
- crypto key generate rsa

- show netconf-yang statistics
- show netconf-yang clients

```bash
linux$ ssh -p 830 cisco@172.16.1.51 -s netconf
Password:

<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
  </_capabilities>
  <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
</hello>
```

**NETCONF 1.1**

- [RFC 6242](https://tools.ietf.org/html/rfc6242) – NETCONF over SSH, port 830/tcp (1.0 transport defined in [RFC 4742](https://tools.ietf.org/html/rfc4742))
## Base Netconf Operations – RFC 6241

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>REQ. CAPABILITY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;get-config&gt;</td>
<td>:base</td>
<td>Retrieve data from the running configuration database</td>
</tr>
<tr>
<td>&lt;get&gt;</td>
<td>:base</td>
<td>Retrieve data from the running configuration database and/or device statistics</td>
</tr>
<tr>
<td>&lt;edit-config&gt;</td>
<td>:base</td>
<td>Modify a configuration database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- operation = merge (default), delete, create, replace, remove</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- test-option (:validate), error-option</td>
</tr>
<tr>
<td>&lt;copy-config&gt;</td>
<td>:base</td>
<td>Copy a configuration database</td>
</tr>
<tr>
<td>&lt;delete-config&gt;</td>
<td>:base</td>
<td>Delete a configuration database</td>
</tr>
<tr>
<td>&lt;discard-changes&gt;</td>
<td>:base :candidate</td>
<td>Clear all changes from the &lt;candidate/&gt; configuration db and make it match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the &lt;running/&gt; configuration db</td>
</tr>
<tr>
<td>&lt;commit&gt;</td>
<td>:base :candidate</td>
<td>Commit the contents of the &lt;candidate/&gt; configuration db to the &lt;running/&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>configuration db</td>
</tr>
<tr>
<td>&lt;cancel-commit&gt;</td>
<td>:base :candidate</td>
<td>Cancels an ongoing confirmed commit.</td>
</tr>
<tr>
<td>&lt;lock&gt;</td>
<td>:base</td>
<td>Lock a configuration database so only my session can write</td>
</tr>
<tr>
<td>&lt;unlock&gt;</td>
<td>:base</td>
<td>Unlock a configuration database so any session can write</td>
</tr>
<tr>
<td>&lt;close-session&gt;</td>
<td>:base</td>
<td>Terminate this session</td>
</tr>
<tr>
<td>&lt;kill-session&gt;</td>
<td>:base</td>
<td>Terminate another session</td>
</tr>
</tbody>
</table>
Netconf and Restconf

RPC (Remote Procedure Call) operations to access config and state data

- **NETCONF** (RFC6241) – for networking community (feels like a Protocol)
  - XML format, defined operations
  - Runs over SSH/TLS/...

- **RESTCONF** (draft-bierman-netconf-restconf) – for developers community (feels like an API)
  - JSON or XML format, uses standard HTTP and its methods: GET, POST, HEAD...
  - ReST (Representational State Transfer) based – a software architectural style used in Web services

<table>
<thead>
<tr>
<th>NETCONF (XML)</th>
<th>RESTCONF (HTTP)</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;get&gt;</td>
<td>HTTP GET</td>
<td>Get operational data (like “show” commands)</td>
</tr>
<tr>
<td>&lt;get-config&gt;</td>
<td>HTTP GET</td>
<td>Get configuration (like “show run”)</td>
</tr>
<tr>
<td>&lt;edit-config&gt;</td>
<td>HTTP PATCH</td>
<td>Edit configuration (like “conf t” and then “commit”)</td>
</tr>
<tr>
<td>&lt;edit-config&gt; operation=“delete”</td>
<td>HTTP DELETE</td>
<td>Delete configuration (eg. like “no int lol11”)</td>
</tr>
<tr>
<td>&lt;edit-config&gt; operation=“create”</td>
<td>HTTP POST</td>
<td>Create configuration (eg. like “int tunnel-te 100 …”)</td>
</tr>
<tr>
<td>&lt;edit-config&gt; operation=“replace”</td>
<td>HTTP PUT</td>
<td>Replace configuration</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Restconf GET reply example

HTTP/1.1 200 OK
Content-Type: application/json
{
    "ietf-interfaces:interfaces": {
        "interface": [
            {
                "name": "eth0",
                "type": "ethernetCsmacd",
                "location": "0",
                "enabled": true,
                "if-index": 2
            },
            {
                "name": "eth1",
                "type": "ethernetCsmacd",
                "location": "1",
                "enabled": false,
                "if-index": 2
            }
        ]
    }
}

HTTP/1.1 200 OK
Content-Type: application/xml

<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
        <name>eth0</name>
        <type>ethernetCsmacd</type>
        <location>0</location>
        <enabled>true</enabled>
        <if-index>2</if-index>
    </interface>
    <interface>
        <name>eth1</name>
        <type>ethernetCsmacd</type>
        <location>1</location>
        <enabled>false</enabled>
        <if-index>7</if-index>
    </interface>
</interfaces>

JSON ~214B
XML ~347B
Restconf/Yang example – OpenDaylight controller

PCEP topology – add LSP (create TE tunnel)*

HTTP PUT (XML) – use RESTclient, or POSTman, or Python (Requests library)...

```
Content-Type: application/xml

<input>
  <node>pcc://172.16.1.51</node>
  <name>Test-Tunnel1</name>
  <arguments>
    <endpoints-obj>
      <ipv4>
        <source-ipv4-address>192.168.0.9</source-ipv4-address>
        <destination-ipv4-address>192.168.0.11</destination-ipv4-address>
      </ipv4>
    </endpoints-obj>
  </arguments>
    /topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]
  </network-topology-ref>
</input>
```

Yang models for this REST API:

https://wiki.opendaylight.org/view/BGP_LS_PCEP:Models#PCEP_models

- This is ODL Yang – completely different from IOS Netconf (with XR XML or XR Yang)
How open is “Open”?

• **Open Standard** is not a buzzword
  – collaborative and consensus-driven process, publicly available

• **Open != Open Source**
  – it may contain commercial parts as well as open source

• **Open can be mean an “open invitation to vendor lock-in” 😊**
  – many “open” projects **dominated by a single vendor**
    – eg. Open Contrail, Open vSwitch, proprietary OVSDB extensions…
  – look for a **foundational governance**, diverse community with major industry players
    – eg. OpenStack, OpenDaylight…
SDN Protocols: Agenda

IETF Open Standards

Application Frameworks, Management Systems, Controllers, ...

- Protocols
  - C, Java, Py
  - NETCONF
  - REST
- OpenFlow
- ACI Fabric
- OpenStack
- DevOps
- Management
- Orchestration
- Network Services
  - BGP, MPLS,...
- Control
- Forwarding
- Device

Operating Systems – IOS / NX-OS / IOS-XR

API’s (OnePK)
Data Models (Yang)

SSH
HTTP
Plug-in’s

OpFlex
Neutron
Puppet/Chef

IPSE
I2RS

NETCONF
RESTful

C, Java, Py

OpenFlow

ACI Fabric
OpenStack
DevOps
MPLS/CarrierE Network Programmability
The Existing Solution and Programmability Enhancements

**Motivation: Real-time Network Optimization**
- utilize unused links, find non-SPF resources

- **PCE (Path Computation Element)**
  - SDN extensions: LSP control delegation and state synchronization
  - Stateful PCEP (PCE Protocol)

- **BGP-LS (Link State)**
  - SDN Controller gets an accurate topology information & instant updates
  - New BGP AF: redistribute ISIS/OSPF topology and SR labels

- **BGP-FS (FlowSpec)**
  - SDN-driven innovations: iBGP, Set Next-Hop/TE tunnel, IPv6 support…
  - Granular (ACL based) traffic selection and action set

- **Segment Routing**
  - SDN-driven evolution of MPLS – ISIS/OSPF enhancements
  - Simplification (no RSVP-TE, no LDP), State is moved to a controller

- **Application**
  - Cisco WAE
  - The optimal path is not the shortest path!
  - More granular Traffic selection (BGP-FS, OF, …)
  - Netflow, MIB’s
  - REST

- **SDN Controller**
  - OpenDaylight
  - Paths DB
  - Topology DB

- **Cisco live!**

BRKSPG-2516 © 2015 Cisco and/or its affiliates. All rights reserved. Cisco Public 31
How to get the topology? (and SR labels)

**BGP-LS**

- BGP Link-State (BGP-LS) – *draft-ietf-idr-ls-distribution*
- Redistribute IGP LSDB into per-domain BGP speaker
- Advantages:
  - Single upstream topology feed (BGP)
  - IGP isolated from external entities
  - Leverage well-known BGP security, transport and policy knobs
  - Enables operator control, responsibility border

- IGP LSDB can contain more information than just cost
  - Delay, Jitter, Loss, Residual bw, Available bw
  - Extensions to ISIS/OSPF (new TLVs)
    - *draft-previdi-isis-te-ospf-metric-extensions*
    - *draft-ietf-ospf-te-metric-extensions*
BGP speakers express their BGP-LS support in capabilities
- LSDB carried in MP_REACH_NLRI
  - Link State
    - LS NLRI: link, node or prefix (IPv4/IPv6)
    - LS Attribute: Describes a topology element

Prefix string format (sh bgp link link)

```
[NLRI-Type][Area][Protocol-ID][Local node descriptor][Remote node descriptor][Attributes]/prefix-length
```
How to program the explicit path?

Stateful PCEP

Stateless PCE for Inter-Area MPLS TE

- ABRs act as stateless PCEs
- ABRs implement backward recursive PCE-Based Computation
- Introduced in IOS XR 3.5.2
- IOS XR 5.1.1 introduces PCEP RFC-compliance

- Out-of-network, stateless PCE server
- PCC initiates LSPs
- Introduced in IOS XR 3.5.2
- IOS XR 5.1.1 introduces PCEP RFC-compliance

SDN Controller
- ODL, WAE

LSP DB

Stateful PCE Server

TED

PCEP

PCC Agent

PCE-initiated LSP

• Out-of-network, stateful PCE server
  Introduced in IOS XR 5.1.1
How to program the explicit path?

Stateful PCEP

- PCE (Path Computation Element) Architecture
  - Centralized Computation Model for MPLS TE (2006)
  - Cisco innovation, originally for Inter-AS TE (stateless)

- PCE Architecture Elements
  - Controller = PCE Server (PCS)
  - Agent = Path Computation Client (PCC) on the router
  - PCE Protocol (PCEP) = protocol between PCC & PCS (RFC5440)
  - Traffic Engineering Database (TED) – contains topology and resource information (LSDB etc.)

- Stateful PCE (RFC4655) – introduces PCEP extensions for:
  - LSP state synchronization between PCCs and PCEs
  - PCC delegation of LSP control to PCE (active PCE)
PCEP (Path Computation Element Protocol)

Protocol Mechanics

- **Session establishment (capabilities), keep and closure**
  - PCE or PCC initiated (Open, Close, Keepalive messages)

- **Path computation**
  - PCC sends requests, PCE sends replies (Request, Reply messages)

- **Event notification (e.g. request cancelation, congestion)**
  - PCE or PCC originated (Notification messages)

- **Error announcement (e.g. protocol error)**
  - PCE or PCC originated (Error message)

- **LSP state synchronization and delegation**
  - PCC originated (Report message)

- **LSP update (delegated LSPs only)**
  - PCE originated (Update message)

- **LSP creation / initiation**
  - PCE originated (Create / Initiate message)

- **Protocol operates over TCP (port 4189)**

---

**IETF PCE WG**: PCE Extensions for SDN
- draft-ietf-pce-stateful-pce
- draft-crabbe-pce-pce-initiated-lsp
- draft-ali-pce-remote-initiated-gmpls-lsp

---

Stateless and stateful PCE

Stateful PCE only
PCE (Path Computation Element) – SDN style

Config Example – IOS-XR 5.1.2

```
! ipv4 unnumbered mpls traffic-eng Loopback0
!
mpls traffic-eng
  pce
    peer ipv4 172.16.255.3
    stateful-client
    instantiation
  !
!
  auto-tunnel pcc
    tunnel-id min 1000 max 5000
  !
!
```

Verification & Troubleshooting
- show mpls traffic-eng auto-tunnel pcc [...]
- show mpls traffic-eng pce peer [ipv4, node-id,…]
- show mpls traffic-eng pce tunnels <id>
- show mpls traffic-eng pce trace […]

Other options – delegation of LSP to Controller (future)

```
interface tunnel-te 101
  pce delegation

interface tunnel-te 102
  path-option 10 dynamic segment-routing PCE addr ipv4 1.1.1.1
```
How to steer traffic to the new path?

Multiple Options

- Per-destination granularity (XR 5.1.1): Use standard routing tools
  - Autoroute Announce or FA (granularity per egress router)
  - Static route or Autoroute Destination (granularity per prefix)
  - PBTS (set forward-class) can add CoS awareness
  - l2vpn preferred path (per pseudowire)
  - ABF (Access-list Based Forwarding)
  - Enhanced Policy Based Routing – PBF, PBTS, Flow-tag (5.2.2)

- Per-app granularity: Controller can use BGP Flowspec to program classifiers on the Edge
  - SET Next-Hop (can resolve over Tunnel, like PBF)
  - MATCH Application (L3/L4 header fields)
BGP Flowspec (RFC5575)
Dissemination of Flow Specification Rules

New NLRI defined (AFI=1, SAFI=133) – components are optional, order dependent

1. Destination IP Address (1 component)
2. Source IP Address (1 component)
3. IP Protocol (+1 component)
4. Port (+1 component)
5. Destination port (+1 component)
6. Source Port (+1 component)
7. ICMP Type
8. ICMP Code
9. TCP Flags
10. Packet length
11. DSCP
12. Fragment

Traffic Actions
• Rate-limit or Drop, Redirect VRF, Mark DSCP…

The MP_REACH_NLRI – RFC4760
BGP Flowspec improvements towards SDN

Original BGP-FS was for DDoS protection
- only policing or set VRF actions

SDN-driven enhancements were needed:
- Redirect IP extension: draft-simpson-idr-flowspec-redirect
- IBGP extension: draft-ietf-idr-bgp-flowspec-oid
- Persistence Support: draft-uttaro-idr-bgp-persistence
- IPv6 extensions: draft-ietf-idr-flow-spec-v6
- HA/NSR Support

Redirect IP details:
- The redirect nexthop can be explicitly configured and can possibly resolve over IP/MPLS/tunnel
- Redirect nexthop is encoded as the MP_REACH nexthop in the BGP flowspec NLRI along with associated extended community.
- The MP_REACH nexthop can be preserved through the use of the “unchanged” knob.
BGP Flowspec Configuration Example

IOS-XR 5.2.0

- Unique capabilities
  - ipv4/vpnv4 and ipv6/vpnv6
  - Redirect IP

**Internet Flowspec**

```plaintext
router bgp 100
  address-family ipv4 flowspec
  address-family ipv6 flowspec
  neighbor 1.1.1.1
    remote-as 100
    address-family ipv4 flowspec
      address-family ipv6 flowspec
    !
    vrf foo
      address-family ipv4 flowspec
      address-family ipv6 flowspec
    !
```

**VPN Flowspec**

```plaintext
router bgp 100
  address-family vpnv4 flowspec
  address-family vpnv6 flowspec
  neighbor 1.1.1.1
    remote-as 100
    address-family vpnv4 flowspec
      address-family vpnv6 flowspec
    !
  vrf foo
    address-family ipv4 flowspec
    address-family ipv6 flowspec
    !
```

**Local Policies**

```plaintext
flowspec
  [local-install interface-all]
  address-family ipv4|ipv6
    service-policy type pbr <policy-name>
      [{local | remote}]
    service-policy type pbr <policy-name>
      [{local | remote}]
    !
  vrf <vrf-name>
    address-family ipv4|ipv6
    service-policy type pbr <policy-name>
      [{local | remote}]
    service-policy type pbr <policy-name>
      [{local | remote}]
    !
  interface <interface-name>
    {ipv4|ipv6} flowspec disable
```
SDN Protocols: Show me the money!

Application Frameworks, Management Systems, Controllers, ...

- Protocols
- C, Java, Py
- NETCONF
- REST
- OpenFlow
- ACI Fabric
- OpenStack
- DevOps

Management
Orchestration
Network Services
Control
Forwarding
Device

Management Systems

- BGP, IGP, TE,...

Orchestration

- I2RS
- IPSE

Network Services

- SSH
- HTTP

Forwarding

- API’s (OnePK)
- Data Models (Yang)

Device

- Data Models
- Operating Systems – IOS / NX-OS / IOS-XR

DevOps

Puppet/Chef

Neutron

OpFlex

Plug-in’s
Modern Architecture Analogy

Functionalism (leading architectural style in 1920-1970)

- Functionalism was introduced as new form that is able to move away from a pomp and ornamental aesthetics of the 19th century. Garishness and unnecessary complexity was elegantly replaced by pure geometry.
- It’s typical for the functionalistic architecture to use simple shapes. It uses new technology materials – scarlet bricks, iron, concrete.

Motto: Form follows function.
- Dominant functionality drives the design.

Villa Müller (Prague)

Villa Tugendhat (Brno)
Typical Architecture of National Backbones

Legacy Architecture vs. Modernism

# of Transit Nodes: 8 → 5 → 3
**Typical Architecture of National Backbones**

**Legacy Architecture vs. Modernism**

- **BRAS PE**
- **U-PE**
- **N-PE**
- **IP RAN**
- **Cell Site**
- **PAN**

**IP/MPLS Backbone**

- **ISP**
- **IGW**

**Motto:** Network follows traffic.

- Dominant traffic drives the design.

**New Complexity and Cost Center!**

- against all known Best Practices
- few dare today...

# of Transit Nodes: 8 → 5 → 3

Touch-less DWDM
Typical Architecture of National Backbones
Legacy Architecture vs. Modernism

Segment Routing – IPv4, IPv6, MPLS
- no RSVP, no LDP, no dual-stack
- massive state removed from the network

# of Transit Nodes: 8 → 5 → 3

Motto: Network follows traffic.
- Dominant traffic drives the design.
Typical Architecture of National Backbones

Legacy Architecture vs. Modernism

Segment Routing – IPv4, IPv6, MPLS
- no RSVP, no LDP, no dual-stack
- massive state removed from the network

Touch-less DWDM

# of Transit Nodes: 8 → 5 → 3

Motto: Network follows traffic.
- Dominant traffic drives the design.

Application (BRAS) specific API
- PPPoE – we can’t use OF
- option 1: OnePK
- option 2: Netconf/Yang

Carrier Ethernet

IP RAN

Cell Site

PAN

BRAS PE

U-PE

N-PE

P

ISP

Cloud Data Center

Session Start

Service State Offload
- VPN labels, PW labels

Control Plane Offload
- RR, TE, Security...

SDN Controller

vBRAS

vBNG Controller

vRR

MEF 2.0

Cloud VPN

CPE

PCE

BRAS

Cloud Ethernet

PAN

Touchless DWDM

Cisco Public
Virtualized Chaos is still Chaos.

RFC1925: It is easier to move a problem around than it is to solve it.

- Virtualization alone won’t help.
- Need to Modernize and Simplify (network, IT, OSS)!

SDN is like Art Deco:
Simplified architecture with features applied in a new and original way.
Segment Routing – an evolution of MPLS
Simple extension to IGP

- Simple extension to IS-IS or OSPF, automatically builds and maintains Segments
  - Prefix Segment – A Shortest path to the related prefix
  - Adjacency Segment – One hop through the related adjacency

- In MPLS environment: Segment == Label
  - Normal forwarding behavior (push, pop, swap), ECMP, PHP, etc.
  - No need for LDP, RSVP, etc.
Combining Segments to Engineer Path
“MPLSDN” – MPLS evolution towards SDN

- **Software Controls, Network Delivers**
  The Controller decides which path is the best
  Shortest Path and ECMP are the default

- **The state is in the packet, not in the network**
  Great scale – no midpoints, no tailends
  Headend can be a virtual router

- **Technology agnostic**
  Carries MPLS (service labels), IPv4, IPv6
  Transport over MPLS (stack) or IPv6 (SRH)
  Easy migration - no need for LDPv6, RSVPv6…
Segment Routing – IGP Enhancements
IOS XR 5.2.2

What we need to advertise
- SR Capabilities
- SID (label) – Prefix SID, Adjacency SID
- SRGB (Segment Routing Global-label Block) – label indexing

ISIS: draft-ietf-isis-segment-routing-extensions
- IS-IS Router Capability TLV (242) – SR Capability sub-TLV (2) [SRGB]
- Extended IP reachability TLV (135) – Prefix-SID sub-TLV (3)
- SID/Label Binding TLV (149) – Prefix-SID sub-TLV (3)
- Extended IS Reachability TLV (22) – Adjacency-SID sub-TLV (31), LAN-Adjacency-SID sub-TLV (32)

OSPF: draft-ietf-ospf-prefix-link-attr-01, draft-ietf-ospf-segment-routing-extensions-02
- Router Information Opaque LSA (type 4) – SR-Algorithm TLV (8), SID/Label Range TLV (9) [SRGB]
- OSPFv2 Extended Prefix Opaque LSA (type 7) – OSPFv2 Extended Prefix TLV (1), Prefix SID Sub-TLV (2)
Segment Routing – CLI examples (XR 5.2.2)

```
router isis 1
  is-type level-2-only
  net 49.0000.1720.1625.5001.00
  address-family ipv4 unicast
  metric-style wide
  segment-routing mpls

interface Loopback0
  passive
  address-family ipv4 unicast
  prefix-sid {absolute|index} <value> [explicit-null]
```

Enable SR on all IPv4 interfaces in this IS-IS instance

```
Ipv4 Prefix-SID value for loopback0
```

```
router ospf 1
  router-id 100.0.0.1
  segment-routing mpls
  segment-routing forwarding mpls
  area 0
  interface Loopback0
  passive enable
  prefix-sid index <value> [explicit-null]
```

Enable SR on all areas (can be disabled per area)

```
Enable SR forwarding on all interfaces
```

```
Prefix-SID index for loopback0
```
Other benefits of Segment Routing
Scale, Simplicity, Programmability

- **TI-LFA: automatic <50ms convergence**
  Topology Independent Loop-Free Alternate
  100%-coverage <50ms link and node protection
  Automatic, Simple to operate and understand
  Uses the post-convergence path – no transient congestion and suboptimal routing
  Incremental deployment, protects all traffic

- **Anycast: automatic node redundancy**
  Multiple nodes advertise the same Segment Identifier (anycast segment) in addition to their Prefix-SID
  Traffic is forwarded to the closest node based on IGP best path (automatic rerouting)
  No need for PW Redundancy, VPLS full-mesh, etc.
MPLS/CarrierE Network Programmability
The Future: Autonomic Programmable Carrier Ethernet

**Motivation:** Solution for the SDN & NfV Era
- Simple & Programmable, De-Layered, Guaranteed SLA
- All the complexity moves to the Controller and Cloud
- Minimal but “Sufficient” network intelligence

**Software Controls:** YANG
- Network API using standard Network and Service Models, the OSS programs services and paths in real-time

**Network Delivers:** Autonomic Segment Routing
- Devices automatically form an “unbreakable” secure channel (IPv6 RPL) and self-configure for Cisco Validated Design (SDN & Segment Routing)
SDN Protocols: Summary

Application Frameworks, Management Systems, Controllers, ...

- Protocols
- C, Java, Py
- NETCONF
- REST
- OpenFlow
- ACI Fabric
- OpenStack
- DevOps

Management
Orchestration
Network Services
Control
Forwarding
Device

API’s (OnePK)
Data Models (Yang)
SSH
HTTP
Plug-in’s

Operating Systems – IOS / NX-OS / IOS-XR

BGP, IGP, TE, ...

I2RS
IPSE
OpenFlow
OpFlex
Plug-in’s

Puppet/Chef
OpenStack
Neutron

DevOps
ACI Fabric
OpenFlow
OpFlex
Plug-in’s

Operating Systems – IOS / NX-OS / IOS-XR

BGP, IGP, TE, ...

I2RS
IPSE
OpenFlow
OpFlex
Plug-in’s

Puppet/Chef
OpenStack
Neutron

DevOps
ACI Fabric
OpenFlow
OpFlex
Plug-in’s

Operating Systems – IOS / NX-OS / IOS-XR
Call to Action

• Visit the World of Solutions for
  – Related Sessions
    BRKSPG-2722 (SDN deployment in ASR9000), BRKSDN-1119 (Device APIs), BRKRST-3122 (Segment Routing), BRKSPG-2515 (SDN-enabled CarrierE), BRKCLD-2555 (Orchestration),…
  – Related Labs
    LABSPG-2442 (Yang & Tail-f), LABSDN-1335 (OpenDaylight), LABRST-2332 (Segment Routing),…
  – Technical Solution Clinics

• Meet the Engineer (I’m here till Thu)

• Lunch time Table Topics

• DevNet zone related labs and sessions

• Recommended Reading: for reading material and further resources for this session, please visit www.pearson-books.com/CLMilan2015

SSID: IPV6ONLYEXP
PASS: iknowbesteffort
SLAAC + SL-DHCPv6 with SF-NAT64 @ Go6
Support: @ayourtch, #IPV6ONLYEXP
Complete Your **Online Session Evaluation**

- Please complete your online session evaluations after each session. Complete 4 session evaluations & the Overall Conference Evaluation (available from Thursday) to receive your Cisco Live T-shirt.

- All surveys can be completed via the Cisco Live Mobile App or the Communication Stations.
Acknowledgements:
Alexander Clemm, Bertrand Duvivier, Bruno Klauser, Bryan Osoro, Carl Solder, Clarence Filsfils, Santiago Alvarez, and others.