TeraFlow SDN by ETSI

Controlling and Monitoring of Packet Optical Networks

Ricard Vilalta (CTTC, ETSI TFS Chair)

Lluís Gifre (CTTC, ETSI TFS TSC Chair)

20/10/2022

Short course Materials

For a perfect hands-on experience, a VirtualBox VM image is needed. Please download the course VM from the link below and make sure the VM is installed and loads/starts up on your PC before the Tutorial/Short Course:

- https://www.dropbox.com/s/gbqyybdv6nndufn/TFS-HF-VM.rar?dl=0
- https://cttcbarcelona-my.sharepoint.com/:u:/g/personal/rvilalta_cttc_es/EbRQLhIf9AFOisM0SHcxmiYBNzSPS_l1YqOX-UG2AAln4w?e=tKOtzg
- VM user/pass: tfs/tfs123

Inside the VM, open: /home/tfs/tfs-ctl/hackfest/commands.txt

to have all commands listed in this tutorial.

- Also available at:
  - https://labs.etsi.org/rep/tfs/controller/-/blob/feat/hackfest/hackfest/commands.txt
- Please go to github repository to get latest version:
  - https://labs.etsi.org/rep/tfs/controller/-/tree/develop
- Use proper environment:
  - pyenv activate 3.9.13/envs/tfs

Agenda

11:00 am - 11:10 am - Welcome & Logistics (10 min)

11:10 am - 11:20 am - Motivation (10 min)

11:20 am - 11:45 am - YANG Data Modelling Language (25 min)

11:45 am - 12:25 pm - Netconf (40 min)

12:25 am - 12:55 pm - ONF Transport API (30 min)

12:55 pm - 01:00 pm - Group Picture!! (5 min)

01:00 pm - 02:00 pm - Lunch Break (60 min)

02:00 pm - 02:10 pm - Introduction ETSI TeraFlowSDN (10 min)

02:10 pm - 02:30 pm - Deployment of TeraFlowSDN (20 min)

02:30 pm - 02:50 pm - Onboarding Network Devices (20 min)

02:50 pm - 03:30 pm - Programmable L3 routers (40 min)

03:30 pm - 04:00 pm - Afternoon Break (30 min)

04:00 pm - 04:30 pm - Monitoring with gRPC (30 min)

04:30 pm - 04:55 pm - Introduction to P4 and P4 demo (25 min)

04:55 pm - 05:00 pm - Conclusion (5 min)

ETSI TFS Hackfest Team

Lluis Gifre, CTTC
TFS TSC Chair

Ricard Vilalta, CTTC
TFS Chair

Georgios Katsikas, UBITECH
TFS TSC Member

Panagiotis Famelis, UBITECH
TFS Member

Silvia Almagia, ETSI
TFS Technical Officer

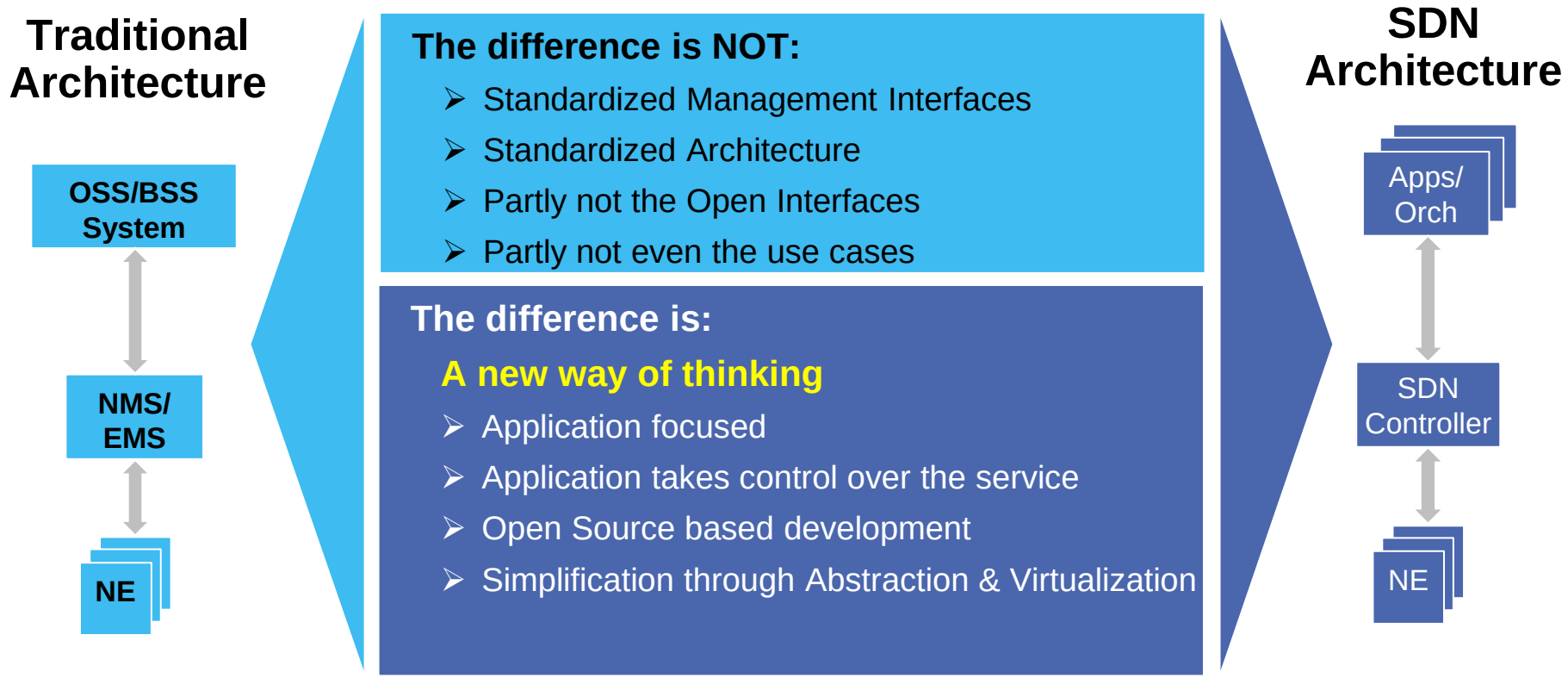
Aurelie Sfez, ETSI
Event Coordinator

Jean Luc Freisse, ETSI
IT Matters

TeraFlow SDN

Motivation

Why is SDN different from traditional Architectures?



Why do we need SDN in Transport?

TeraFlow SDN

Principles of SDN

Programmability:
- Programmable interfaces
- Applications focused architecture
- Abstraction & Virtualization
- Multi-Tenant capabilities

Openness:
- Open Standards & Interfaces
- Open Source SW

Integration focused:
- Multi-layer
- Multi-vendor

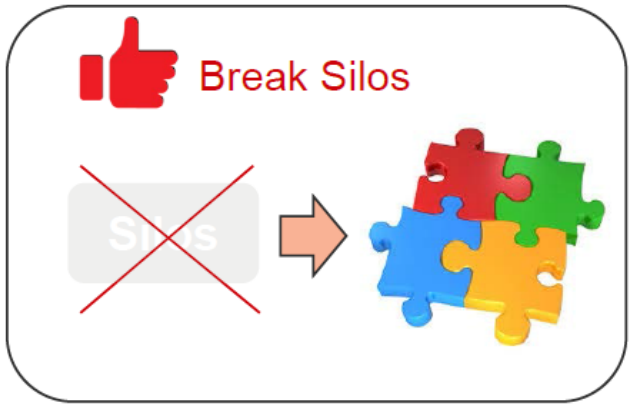
What it Enables in Transport Network

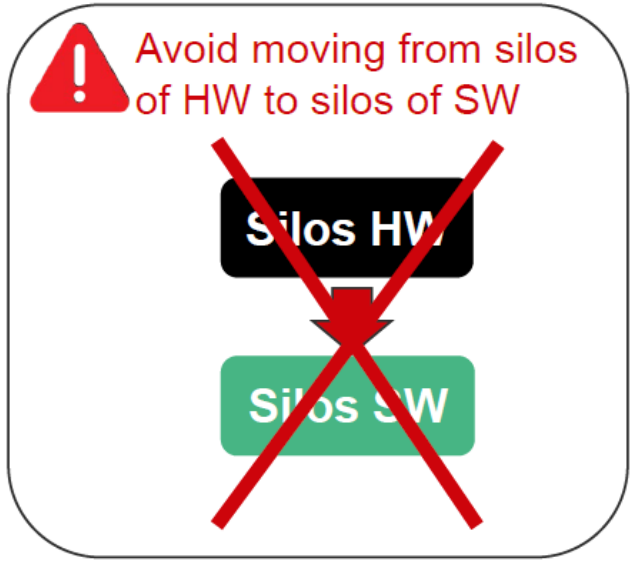
Innovation:
- Opens doors for new service models
- Service differentiation through new application

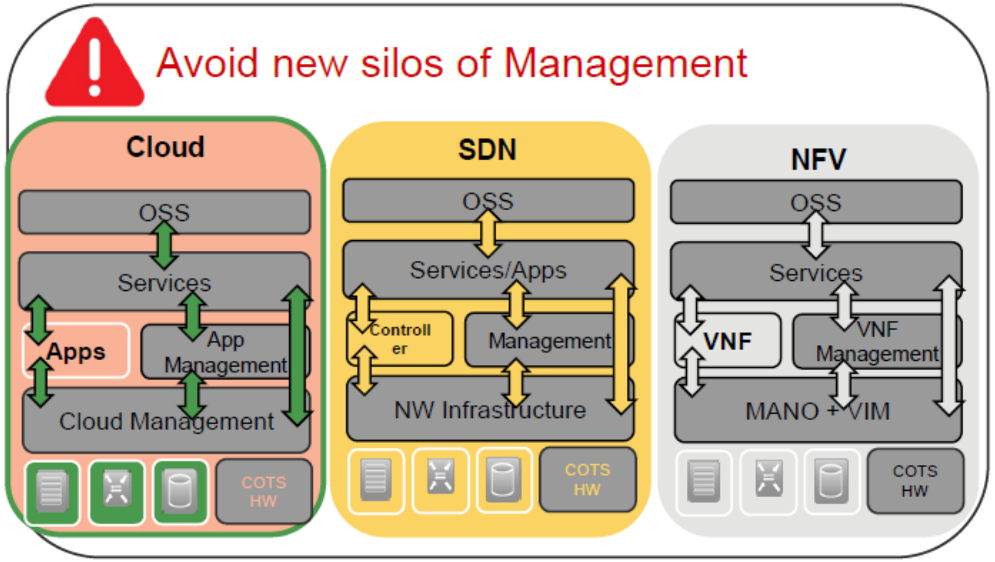
Simplified Architectures:
- Integrated E2E / Multi-layer service creation
- Automatic reaction on errors or any changes

Financial Benefits:
- Opex: efficient service setup
- Capex: fast ROI / hardware utilization
- New revenue opportunities

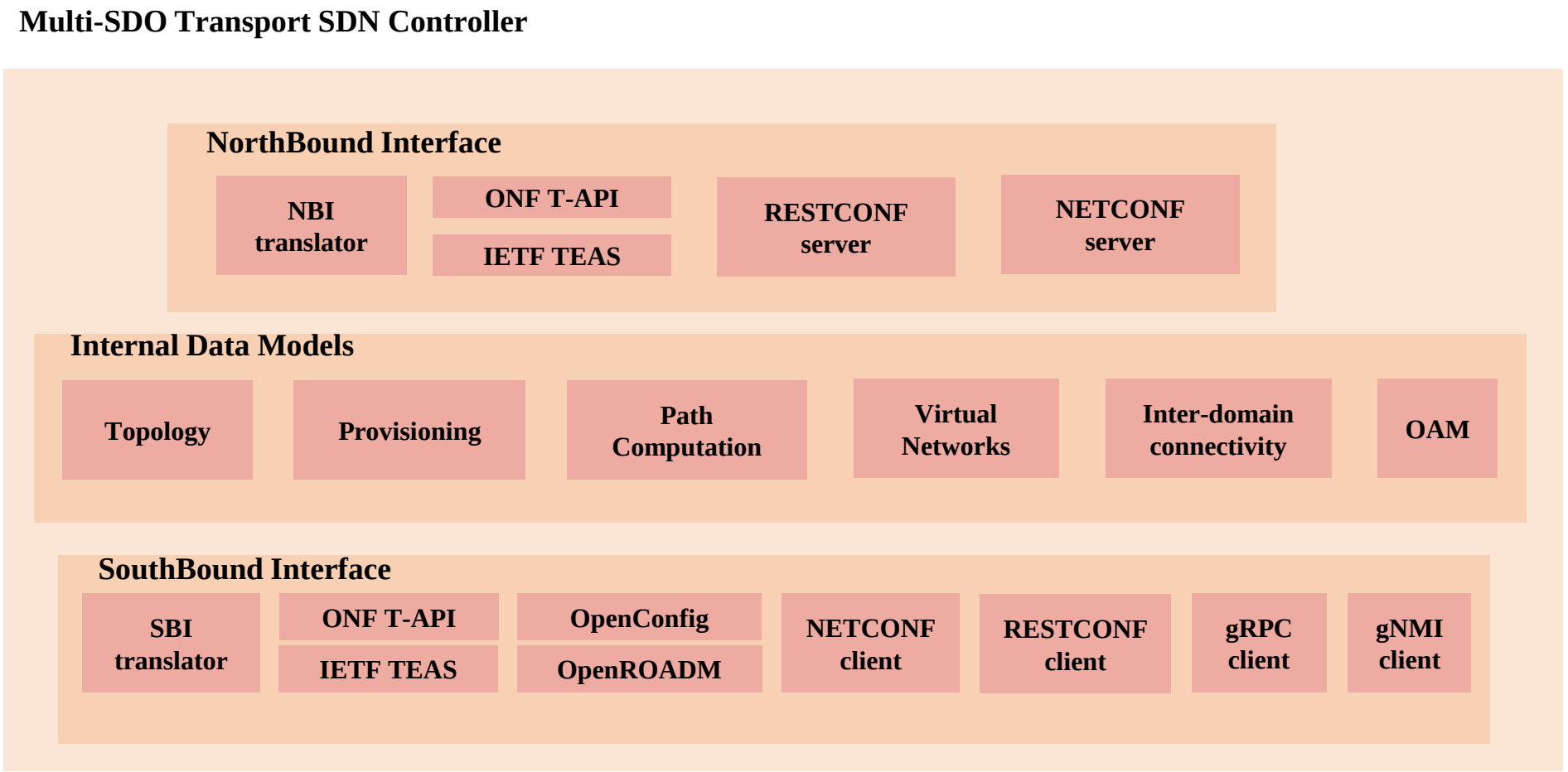
Keys to success





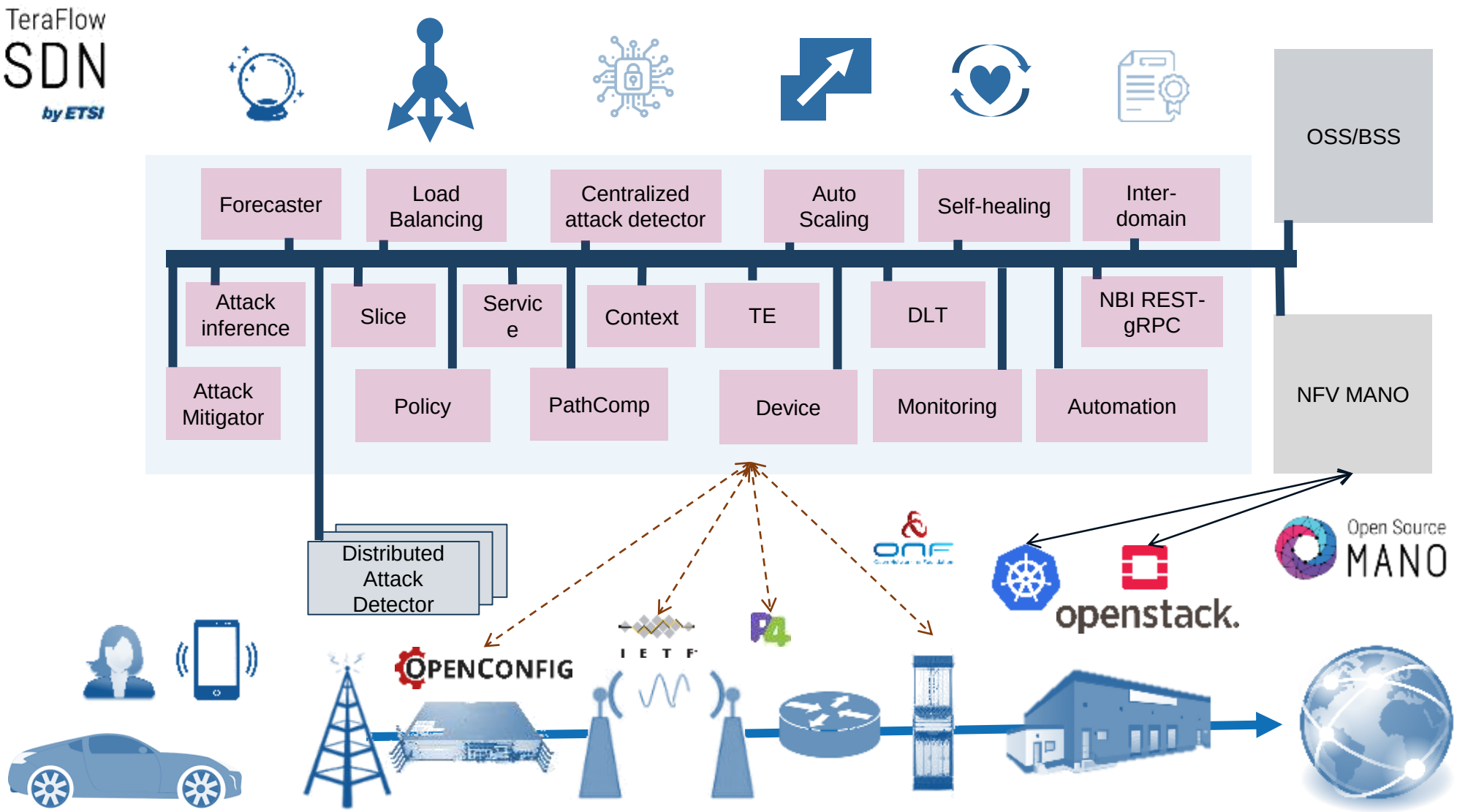


A multi-SDO SDN controller architecture



R. Vilalta et al., Experimental Evaluation of Control and Monitoring Protocols for Optical SDN Networks and Equipment [Invited Tutorial], JOCN 2021.

From theory to practice… Welcome ETSI TeraFlowSDN



Experimental evaluation

The purpose of these experiments is…

- To demonstrate how ETSI TeraFlowSDN controller can manage connectivity services
- Learn how to use different SBIs to configure the network equipment
- **Have fun with network automation!** ☺

TeraFlow SDN

YANG/NETCONF

Unified Information and Data Modeling (1)

TeraFlow SDN

Some deployments of optical transport networks are purely managed, without a dedicated control plane.

- The need of better management frameworks and protocols has long been established.

From the perspective of an operator, the configuration of a control plane (e.g., definition of routing policies, configuration of routing peers) remains a management task.

There is a need to have better configuration management, a clear separation of configuration and operational data, while enabling high level constructs more adapted to operators' workflows supporting network-wide transactions.

While such frameworks are initially focused on management tasks, it is reasonable to *adopt them holistically, covering most aspects related to device and network control*

- Increase of information and data modelling bound to the rise of network programmability.

In general, a device (or system)

- Information Model macroscopically describes the device capabilities, in terms of operations and configurable parameters, using high level abstractions without specific details on aspects such as a particular syntax or encoding.
- Data Model determines the structure, syntax and semantics of the data that is externally visible.

Unified Information and Data Modeling (2) : Goals

Unified information and data modeling language to describe a device capabilities, attributes, operations to be performed on a device or system and notifications

- A common language with associated tools
- Enabling complex models with complex semantics, flexible, supporting extensions and augmentations
- A “best-practice” and guidelines for model authors

An architecture for remote configuration and control

- Client / Server, supporting multiple clients, access lists, transactional semantics, roll-back

An **associated transport protocol** provides primitives to view and manipulate the data, providing a suitable encoding as defined by the data-model.

- Flexible, efficient
- *Ideally, data models should be protocol independent*

Standard, agreed upon models for devices

- Huge activity area
- Hard to reach consensus (controversial aspects)
- Some models do exist. Most stable ones cover mature aspects (interface configuration, RIB, BGP routing)

The YANG Language I

YANG is a data modeling language, initially conceived to model configuration and state data for network devices

- Models define the device configurations & notifications, capture semantic details and are easy to understand.
- Significant adoption as data modelling language, across frameworks and Open Source projects
- Ongoing notable effort across the SDOs to model constructs (e.g. topologies, protocols), including optical devices, such as transceivers, ROADMs,… Literally hundreds of emerging standards across SDOs.

A YANG model includes a header, imports and include statements, type definitions, configurations and operational data declarations as well as actions (RPC) and notifications.

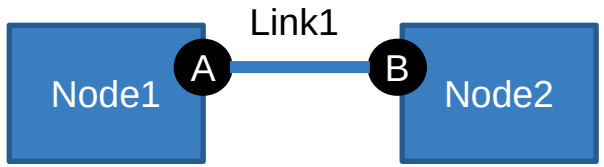
- The language is expressive enough to:
  - Structure data into data trees within the so called datastores, by means of encapsulation of containers and lists, and to define constrained data types (e.g. following a given textual pattern).
  - Condition the presence of specific data to the support of optional features.
  - Allow the refinement of models by extending and constraining existing models (by inheritance/augmentation), resulting in a hierarchy of models.
  - Define configuration and/or state data.

The YANG Language II

YANG has become the data modeling language of choice for multiple network control and management aspects

- Covering devices, networks, and services, even pre-existing protocols.
- Due in part, for its features and flexibility and the availability of tools.
- Examples:
  - An SDN controller may export the underlying optical topology in a format that is unambiguously determined by its associated YANG schema,
  - A high-level service may be described so that an SDN controller is responsible for mediating and associating high-level service operations to per-device configuration operations.

A YANG model for network topology



A network consists of:

- Nodes and Links

A node consists of:

- node-id and ports

A port consists of:

- port-id and type of port

A link consists of:

- link-id, reference to source node, reference to target node, reference to source port and reference to target port.

topology.yang

```
module topology {

  namespace "urn:topology";
  prefix "topology";
  organization
    "CTTC";
  contact
    "ricard.vilalta@cttc.es";
  description
    "Basic example of network
topology";

  revision "2018-08-24" {
    description "Basic
example of network
topology";
    reference "";
  }

typedef layer-protocol-name {
     type enumeration {
         enum "ETH";
         enum "OPTICAL";
     }
   }

...
```

```
…

grouping port {
    leaf port-id {
      type string;
    }
    leaf layer-protocol-name {
      type layer-protocol-
name;
    }

  }

  grouping node {
    leaf node-id {
      type string;
    }
    list port {
      key "port-id";
      uses port;
    }
  }
```

```
…

grouping link {
    leaf link-id {
      type string;
    }
    leaf source-node {
      type leafref {
        path "/topology/node/node-id";
      }
    }
    leaf target-node {
      type leafref {
        path "/topology/node/node-id";
      }
    }
    leaf source-port {
      type leafref {
        path "/topology/node/port/port-id";
      }
    }
    leaf target-port {
      type leafref {
        path "/topology/node/port/port-id";
      }
    }
  }
```

```
…

 grouping topology {
     list node {
       key "node-id";
       uses node;
     }
     list link {
       key "link-id";
       uses link;
     }
 }

 /**
  * Container/lists
  */
 container topology {
   uses topology;
 }
```

[Tool] pyang

TeraFlow SDN

An extensible YANG validator and converter in python https://github.com/mbj4668/pyang

- Check correctness, to transform YANG modules into other formats, and to generate code from the modules

```
# pyang -f tree topology.yang

module: topology
   +--rw topology
      +--rw node* [node-id]
      |  +--rw node-id    string
      |  +--rw port* [port-id]
      |     +--rw port-id                string
      |     +--rw layer-protocol-name?   layer-protocol-name
      +--rw link* [link-id]
         +--rw link-id        string
         +--rw source-node?   -> /topology/node/node-id
         +--rw target-node?   -> /topology/node/node-id
         +--rw source-port?   -> /topology/node/port/port-id
         +--rw target-port?   -> /topology/node/port/port-id
```

```
# pyang -f sample-xml-skeleton --sample-xml-skeleton-annotations
topology.yang

<?xml version='1.0' encoding='UTF-8'?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<topology xmlns="urn:topology">
  <node>
    <!-- # entries: 0.. -->
    <node-id><!-- type: string --></node-id>
    <port>
      <!-- # entries: 0.. -->
      <port-id><!-- type: string --></port-id>
      <layer-protocol-name><!-- type: layer-protocol-name --></layer-protocol-
name>
    </port>
  </node>
  <link>
    <!-- # entries: 0.. -->
    <link-id><!-- type: string --></link-id>
    <source-node><!-- type: leafref --></source-node>
    <target-node><!-- type: leafref --></target-node>
    <source-port><!-- type: leafref --></source-port>
    <target-port><!-- type: leafref --></target-port>
  </link>
</topology>
</data>
```

UML diagram

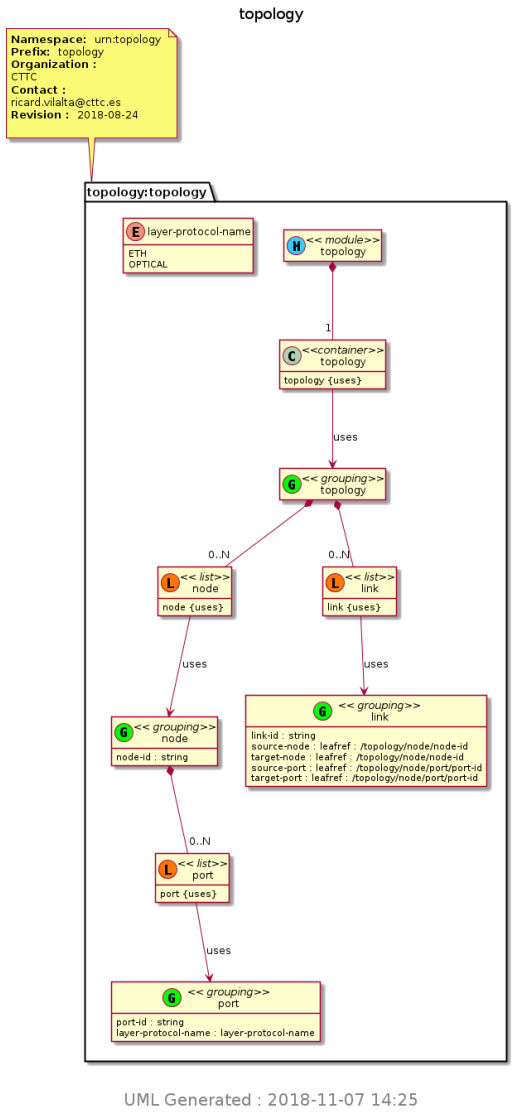
PlantUML is an opensource tool to create UML diagrams

Pyang is able to create an UML diagram of the desired yang module

Only a certain version of PlantUML is compatible with provided output:

http://sourceforge.net/projects/plantuml/files/plantuml.7997.jar/download

```
# pyang -f uml topology.yang -o topology.uml
# java -jar plantuml.jar topology.uml
```



From YANG to code: pyangbind

PyangBind is a plugin for Pyang that generates a Python class hierarchy from a YANG data model. The resulting classes can be directly interacted with in Python. Particularly, PyangBind will allow you to:

- Create new data instances - through setting values in the Python class hierarchy.
- Load data instances from external sources - taking input data from an external source and allowing it to be addressed through the Python classes.
- Serialise populated objects into formats that can be stored, or sent to another system (e.g., a network element).

Please install from sources. It includes new serialization to XML.

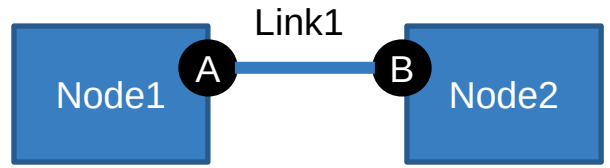
```
$ export PYBINDPLUGIN=`/usr/bin/env python -c \
'import pyangbind; import os; print ("{}/plugin".format(os.path.dirname(pyangbind.__file__)))'`
$ echo $PYBINDPLUGIN
$ pyang -f pybind topology.yang --plugindir $PYBINDPLUGIN -o binding_topology.py
```

Source: https://github.com/robshakir/pyangbind

How to Create a topology

Create an XML and a JSON that is compliant with topology.yang

Use the proposed simple network topology



Import the generated pyangbind bindings

Basic pyangbind tutorial:
https://github.com/robshakir/pyangbind#getting-started

Use pyangbind serializers

```
$ python3 topology.py
```

```
from binding_topology import topology
from pyangbind.lib.serialise import pybindIETFXMLEncoder
import pyangbind.lib.pybindJSON as pybindJSON

topo = topology()
node1=topo.topology.node.add("node1")
node1.port.add("node1portA")
node2=topo.topology.node.add("node2")
node2.port.add("node2portA")
link=topo.topology.link.add("link1")
link.source_node = "node1"
link.target_node = "node2"
link.source_port = "node1portA"
link.target_port = "node2portA"

print(pybindIETFXMLEncoder.serialise(topo))
print(pybindJSON.dumps(topo))
```

Topology XML

```
<topology xmlns="urn:topology">
  <topology>
    <node>
      <node-id>node1</node-id>
      <port>
        <port-id>node1portA</port-id>
      </port>
    </node>
    <node>
      <node-id>node2</node-id>
      <port>
        <port-id>node2portA</port-id>
      </port>
    </node>
    <link>
      <target-node>node2</target-node>
      <source-port>node1portA</source-port>
      <link-id>link1</link-id>
      <source-node>node1</source-node>
      <target-port>node2portA</target-port>
    </link>
  </topology>
</topology>
```

Topology JSON

TeraFlow SDN


```
{
   "topology": {
      "node": {
         "node1": {
            "node-id": "node1",
            "port": {
               "node1portA": {
                  "port-id": "node1portA"
               }
            }
         },
         "node2": {
            "node-id": "node2",
            "port": {
               "node2portA": {
                  "port-id": "node2portA"
               }
            }
         }
      },
      "link": {
         "link1": {
            "link-id": "link1",
            "source-port": "node1portA",
            "target-node": "node2",
            "target-port": "node2portA",
            "source-node": "node1"
         }
      }
   }
}
```

Exercise: Create a connection data model

TeraFlow SDN

Create a YANG data model for connection.

- Connection consists of:
  - connection-id (string)
  - source-node, source-port, destination-node, destination-port (leaf-ref)
  - bandwidth (uint32)
  - layer-protocol-name (from topology.yang)

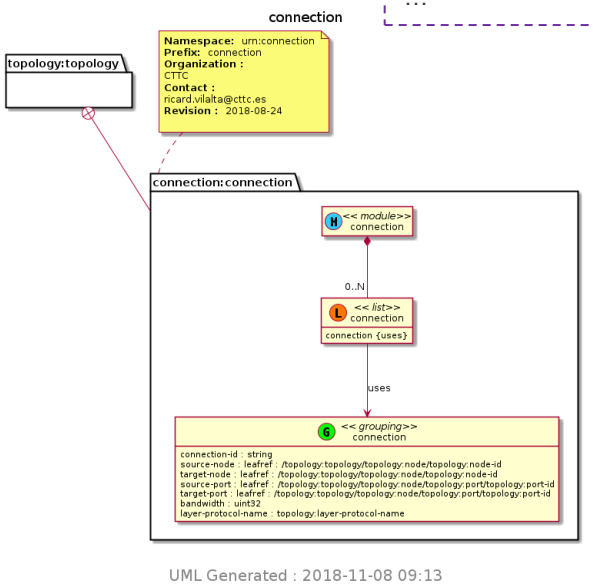
Validate model with pyang

Create pyangbind bindings

Create xml using bindings

Solution: connection.yang

```
module connection {
  namespace "urn:connection";
  prefix "connection";
  import topology {
    prefix "topology";
  }
  organization
    "CTTC";
  contact
    "ricard.vilalta@cttc.es";
  description
    "Basic example of network topology";
  revision "2018-08-24" {
    description "Basic example of network
topology";
    reference "";
  }
…
```




```
…
grouping connection {
    leaf connection-id {
      type string;
    }
    leaf source-node {
      type leafref {
        path "/topology:topology/topology:node/topology:node-id";
      }
    }
    leaf target-node {
      type leafref {
        path "/topology:topology/topology:node/topology:node-id";
      }
    }
    leaf source-port {
      type leafref {
        path "/topology:topology/topology:node/topology:port/topology:port-id";
      }
    }
    leaf target-port {
      type leafref {
        path "/topology:topology/topology:node/topology:port/topology:port-id";
      }
    }
    leaf bandwidth {
      type uint32;
    }
    leaf layer-protocol-name {
      type topology:layer-protocol-name;
    }
}
list connection {
    key "connection-id";
    uses connection;
  }
}
```

Solution: connection.py

TeraFlow SDN

```
$ python3 connection.py
```

```
from binding_connection import connection
from pyangbind.lib.serialise import pybindIETFXMLEncoder
import pyangbind.lib.pybindJSON as pybindJSON

con = connection()
con1=con.connection.add("con1")
con1.source_node = "node1"
con1.target_node = "node2"
con1.source_port = "node1portA"
con1.target_port = "node2portA"
con1.bandwidth = 1000
con1.layer_protocol_name = "OPTICAL"
print(pybindIETFXMLEncoder.serialise(con))
print(pybindJSON.dumps(con))
```

```
<connection xmlns="urn:connection">
  <connection>
    <connection-id>con1</connection-id>
    <source-node>node1</source-node>
    <target-node>node2</target-node>
    <source-port>node1portA</source-port>
    <target-port>node2portA</target-port>
    <bandwidth>1000</bandwidth>
    <layer-protocol-name>OPTICAL</layer-protocol-name>
  </connection>
</connection>
```

The NETCONF Protocol (1)

Offers primitives to view and manipulate data, providing a suitable encoding as defined by the data-model.

- Data is arranged into one or multiple *configuration datastores* (set of configuration information that is required to get a device from its initial default state into a desired operational state.)

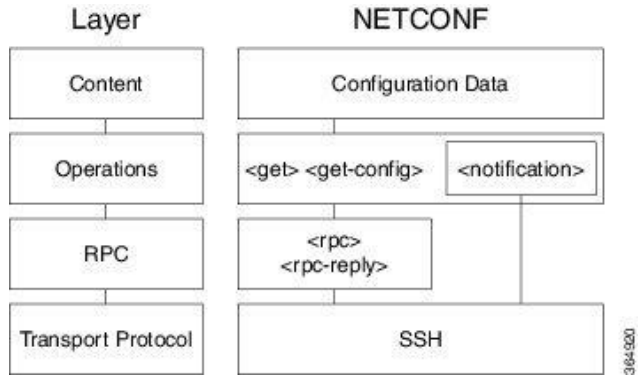
Enables remote access to a device, and provides the set of rules by which multiple clients may access and modify a datastore within a NETCONF server (e.g., device).

- NETCONF enabled devices *include a NETCONF server*,
- Management applications *include a NETCONF client* and device Command Line Interfaces (CLIs) can be a wrapped around a NETCONF client.

It is based on the exchange of XML-encoded RPC messages over a secure (commonly Secure Shell, SSH) connection.

NETCONF Layering :

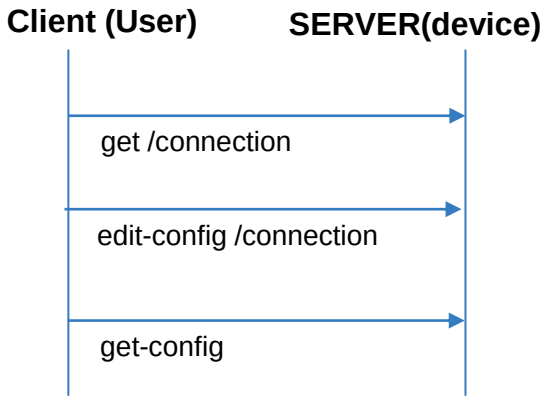
- Configuration or notification data (Content Layer) that is exchanged between a client and a server,
- Operations layer (e.g. <get-config>, <edit-config>)
- Message layer for RPC messages or notifications
- Secure Transport.

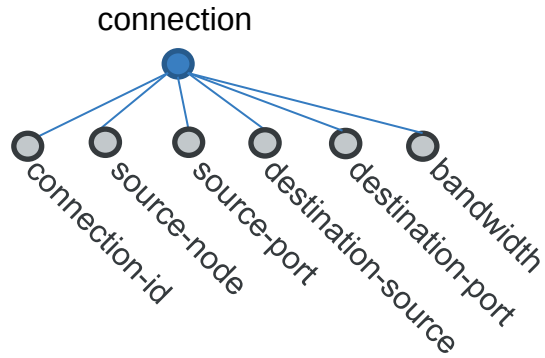


The NETCONF Protocol (2)

After establishing a session over a secure transport, both entities send a hello message to announce their protocol capabilities, the supported data models, and the server's session identifier.

When accessing configuration or state data, with NETCONF operations, subtree filter expressions can select subtrees.





Operation	Description
<get>	Retrieve running configuration and device state information
<get-config>	Retrieve all or part of a specified configuration datastore
<edit-config>	Edit a configuration datastore by creating, deleting, merging or replacing content
<copy-config>	Copy an entire configuration datastore to another configuration datastore
<delete-config>	Delete a configuration datastore
<lock>	Lock an entire configuration datastore of a device
<unlock>	Release a configuration datastore lock previously obtained with the <lock> operation
<close-session>	Request graceful termination of a NETCONF session

```
# pyang -f tree connection.yang

module: connection
   +--rw connection* [connection-id]
      +--rw connection-id          string
      +--rw source-node?            -> /topology:topology/node/node-id
      +--rw target-node?           -> /topology:topology/node/node-id
      +--rw source-port?           -> /topology:topology/node/port/port-id
      +--rw target-port?          -> /topology:topology/node/port/port-id
      +--rw bandwidth?             uint32
      +--rw layer-protocol-name?   topology:layer-protocol-name
```

NETCONF Basic server

Use Python library: Netconf http://netconf.readthedocs.io/

Simple server listening on port 830 that handles one RPC:

- Read and parse as data the file topology.xml
- Provide it when get-config is requested

Serve as capability:

- topology

Basic tutorial:
https://netconf.readthedocs.io/en/master/develop.html#netconf-server

NOTE: We use port 8300

instead of default privileged port 830.

Basic server (simplified)

```
import sys
import time
import logging
import os

from binding_topology import topology

from netconf import nsmap_add, NSMAP
from netconf import server, util
from lxml import etree

logging.basicConfig(level=logging.DEBUG)


nsmap_add("topology", "urn:topology")

class MyServer(object):
    def load_file(self):
        # create configuration
        xml_root = open('topology.xml', 'r').read()
        topo = pybindIETFXMLDecoder.decode(
                    xml_root, binding_topology, "topology")
        xml = pybindIETFXMLEncoder.serialise(topo)
        tree = etree.XML(xml)
        data = util.elm("nc:data")
        data.append(tree)
        self.node_topology = data
```

```
//(…)
def __init__(self, username, password, port):
        host_key_value = os.path.join(os.path.abspath(
                    os.path.dirname(__file__)), "server-key")
        auth = server.SSHUserPassController(
                    username=username, password=password)
        self.server = server.NetconfSSHServer(
                    server_ctl=auth, server_methods=self, port=port,
                    debug=False)
         self.load_file()

def nc_append_capabilities(self, capabilities):
        util.subelm(capabilities, "capability").text = \
                    "urn:ietf:params:netconf:capability:xpath:1.0"
        util.subelm(capabilities, "capability").text = NSMAP["topology"]

def rpc_get_config(self, session, rpc, source_elm, filter_or_none):
        return util.filter_results(rpc, self.node_topology, None)

def close(self):
        self.server.close()
```

```
def main(*margs):
    s = MyServer("admin","admin", 8300)

    if sys.stdout.isatty():
        logging.debug("^C to quit server")

    try:
        while True:
            time.sleep(1)

    except Exception:
        logging.debug("quitting server")

    s.close()
```

Basic client OSS client

Create a client to CRUD the topology

Python library: Netconf http://netconf.readthedocs.io/

Tutorial: https://netconf.readthedocs.io/en/master/develop.html#netconf-client

First, connect

Second, print capabilities

Third, get config

Fourth, edit basic config

Netconf client

```
from lxml import etree
from netconf.client import NetconfSSHSession

# connexion parameters
host = 'localhost'
port = 2022
username = "admin"
password = "admin"

# connexion to server
session = NetconfSSHSession(host, port, username,
password)

# server capabilities
c = session.capabilities
print(c)

# get config
print("---GET CONFIG---")
config = session.get_config()
xmlstr = etree.tostring(config, encoding='utf8',
xml_declaration=True)
print(xmlstr)
…
```

```
…

# edit config
new_config = '''
<config>
   <topology xmlns="urn:topology">
      <node operation="merge"> <!-- modify with delete -->
         <node-id>10.1.7.64</node-id>
         <port>
            <port-id>3</port-id>
         </port>
      </node>
   </topology>
</config>
'''
print("---EDIT CONFIG---")
config = session.edit_config(newconf=new_config)
xmlstr = etree.tostring(config, encoding='utf8',
xml_declaration=True)
print(xmlstr)

# close connexion
session.close()
```

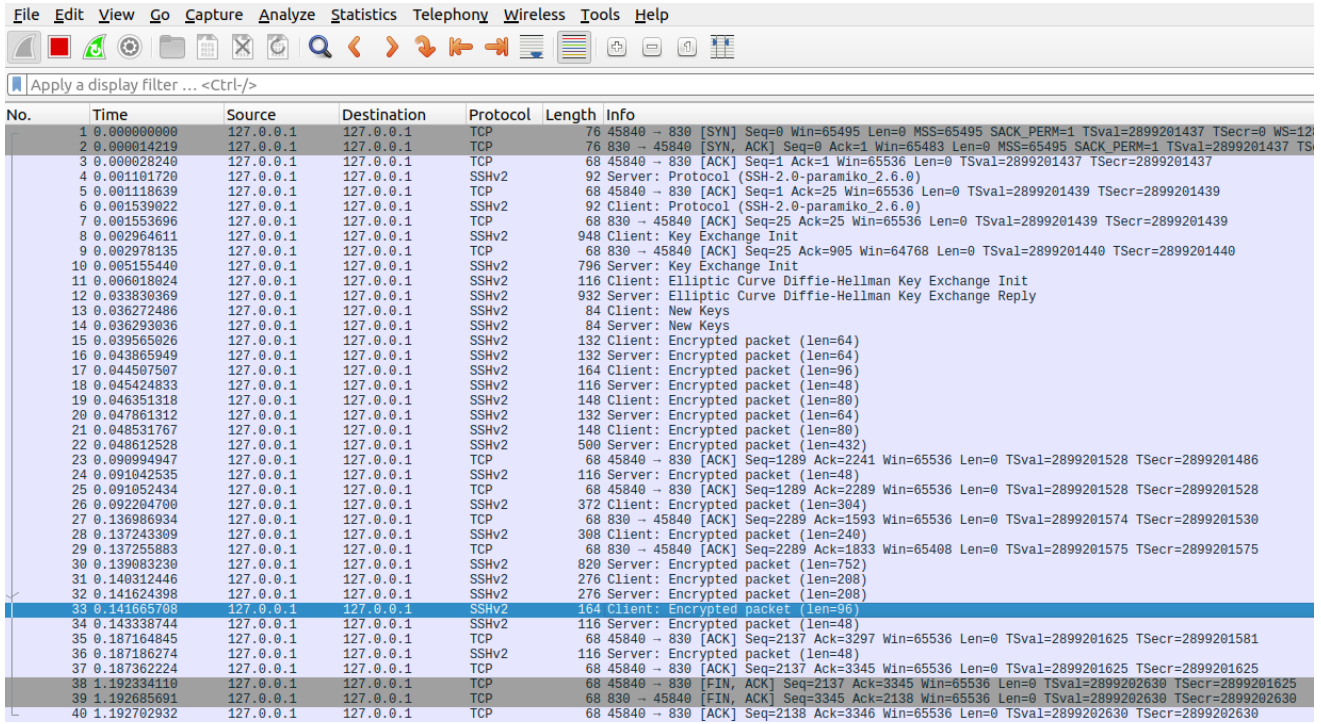
Run NETCONF example

Run server:
$ cd ~/tfs-ctrl/hackfest/netconf
$ python3 server_topology.py

Run client:
$ cd ~/tfs-ctrl/hackfest/netconf
$ python3 client_topology.py

Run Wireshark

- Configure Dissector for port 8300:
- Analyze > Decode As...
- Add row (+ button)
  - Field “TCP port”
  - Value “8300”
  - Type “Integer, base 10”
  - Default “(none)”
  - Current “SSH”
- Click “OK”



Exercise: NETCONF edit-config

TeraFlow SDN

Include connection.yang

Request to create a new connection (client and server).

Server adds new connection

Client list connection

Time: 15min

NETCONF server edit-config: server_topology_connection.py

TeraFlow SDN

```
def rpc_edit_config(self, session, rpc, target, new_config):
        logging.debug("--EDIT CONFIG--")
        logging.debug(session)

        data_list = new_config.findall(".//xmlns:connection", namespaces={'xmlns': 'urn:connection'})
        for connect in data_list:
            logging.debug("connect: " )
            logging.debug(etree.tostring(connect) )
            logging.debug("CURRENT CONNECTION")
            logging.debug(etree.tostring(self.data[1]) )
            self.data[1].append(connect)
            break
        return util.filter_results(rpc, self.data, None)
```

```
Run server:
$ cd ~/tfs-ctrl/hackfest/netconf/connection
$ python3 server_topology_connection.py
```

NETCONF client edit-config client_connection.py

```
# edit config
new_config = '''
<config>
        <connection xmlns="urn:connection" operation="merge">
            <connection-id>connection1</connection-id>
            <source-node>node1</source-node>
            <source-port>node1portA</source-port>
            <target-node>node2</target-node>
            <target-port>node2portA</target-port>
            <bandwidth>10</bandwidth>
            <layer-protocol-name>ETH</layer-protocol-name>
        </connection>
</config>
'''
print("---EDIT CONFIG---")
config = session.edit_config(newconf=new_config)
xmlstr = etree.tostring(config, encoding='utf8', xml_declaration=True)
print(xmlstr)
```

Run client:

```
$ cd ~/tfs-ctrl/hackfest/netconf/connection
$ python3 client_connection.py
```

TeraFlow SDN

CONFD Tutorial

Run a Netconf server

For this example, we will use confd as a netconf server.

Confd is not OpenSource, but follows a Freemium model, which allows testing and usage.

Is a powerful server, with lots of options, and it is useful for training purposes.

Later, we will introduce the development of a netconf server, using open source libraries.

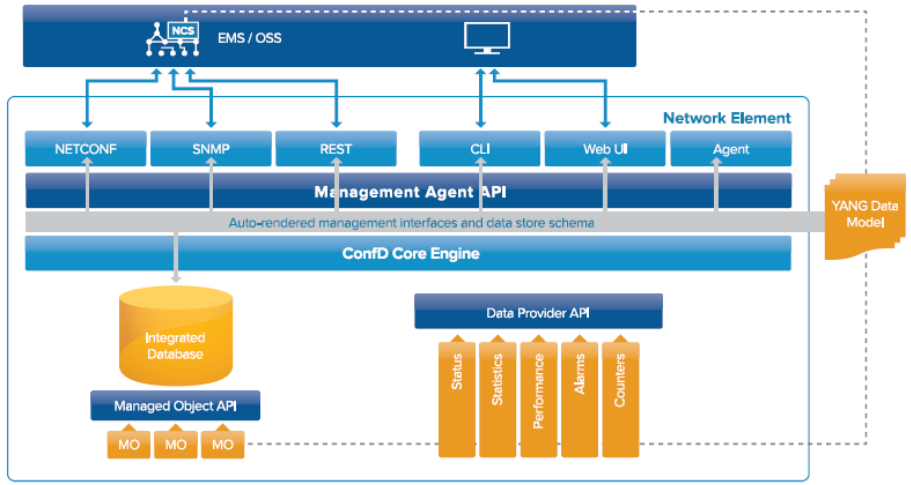


Figure 1: ConfD block diagram

Source:
http://www.tail-f.com/confd-basic/

Using Cisco (Tail-f) ConfD

TeraFlow SDN

Installation

```
$ cd ~/tfs-ctrl/hackfest/netconf
$ unzip confd-basic-6.4.linux.x86_64.zip
$ cd confd-basic-6.4.linux.x86_64/
$ ./confd-basic-6.4.linux.x86_64.installer.bin ~/confd/
```

```
$ cd /root/confd/bin/
$ ./confdc -c ~/tfs-ctrl/hackfest/yang/topology.yang
```

Data-Model Compilation

```
$ ./confd --foreground -v --addloadpath .
```

Start ConfD

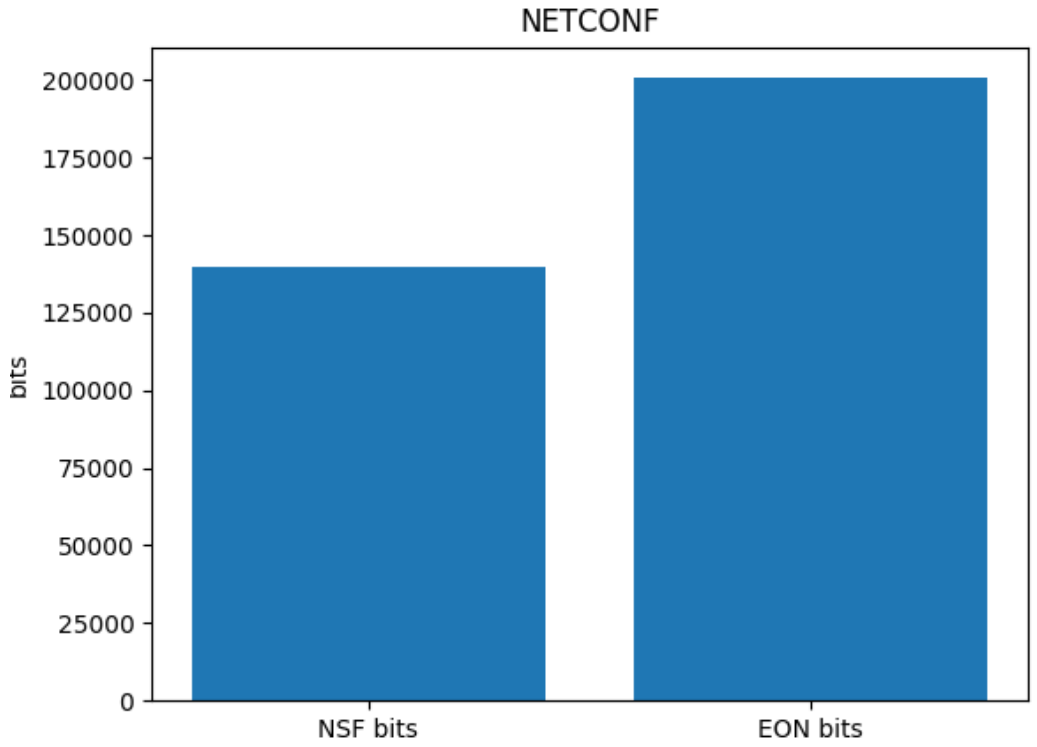
```
$ ./confd_cli
➢ conf
➢ topology node node1
➢ exit
➢ commit
➢ exit
➢ exit
```

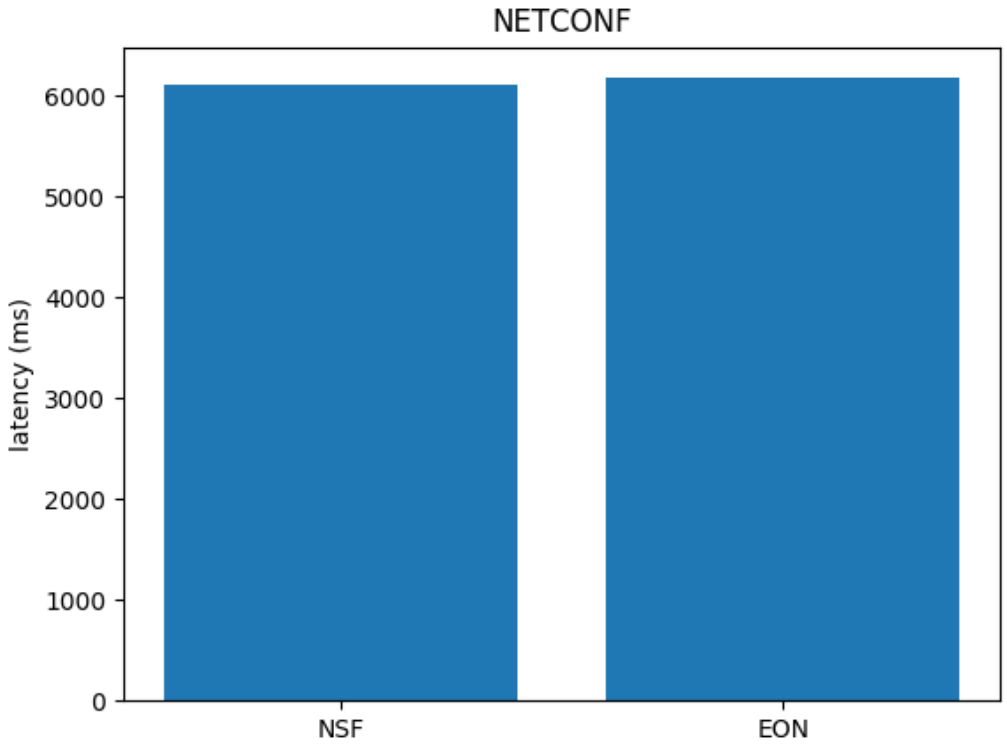
Use ConfD-client

```
$ ./confd_cli
➢ conf
➢ show full-configuration
➢ exit
➢ exit
```

Source:
http://www.tail-f.com/confd-basic/

Example NETCONF results





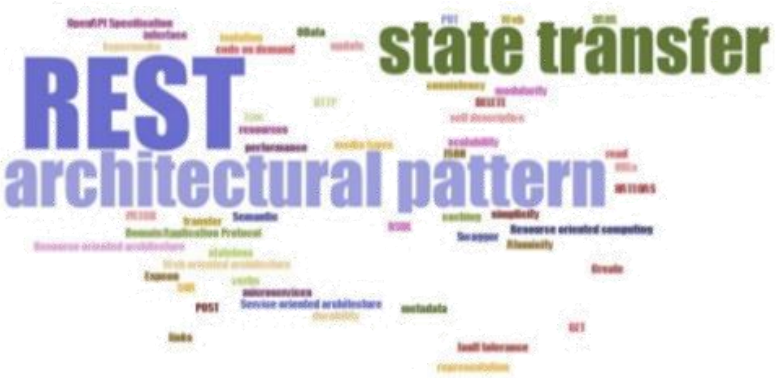
TeraFlow SDN

ONF Transport API 2.1

REST API

A RESTful application is an application that exposes its state and functionality as a set of resources that the clients can manipulate and conforms to a certain set of principles:

- All resources are uniquely addressable, usually through URIs; other addressing can also be used, though.
- All resources can be manipulated through a constrained set of well-known actions, usually CRUD (create, read, update, delete), represented most often through the HTTP's POST, GET, PUT and DELETE;
- The data for all resources is transferred through any of a constrained number of well-known representations, usually HTML, XML or JSON;
- The communication between the client and the application is performed over a stateless protocol.



REST vs non-REST API

RESTful API

GET /user/15

{
“name” : “John Doe”,
“email” : “john.doe@gmail.com”
…
}

Non-RESTful API

GET /last_search?page=2

{
“products” : […]
…
}

RESTCONF

RESTCONF

- RFC 8040
- RESTful protocol to access YANG defined data
- Representational State Transfer, i.e. server maintains no session state
- URIs reflect data hierarchy in a Netconf datastore
- HTTP as transport
- Data encaded with either XML or JSON
- Operations :

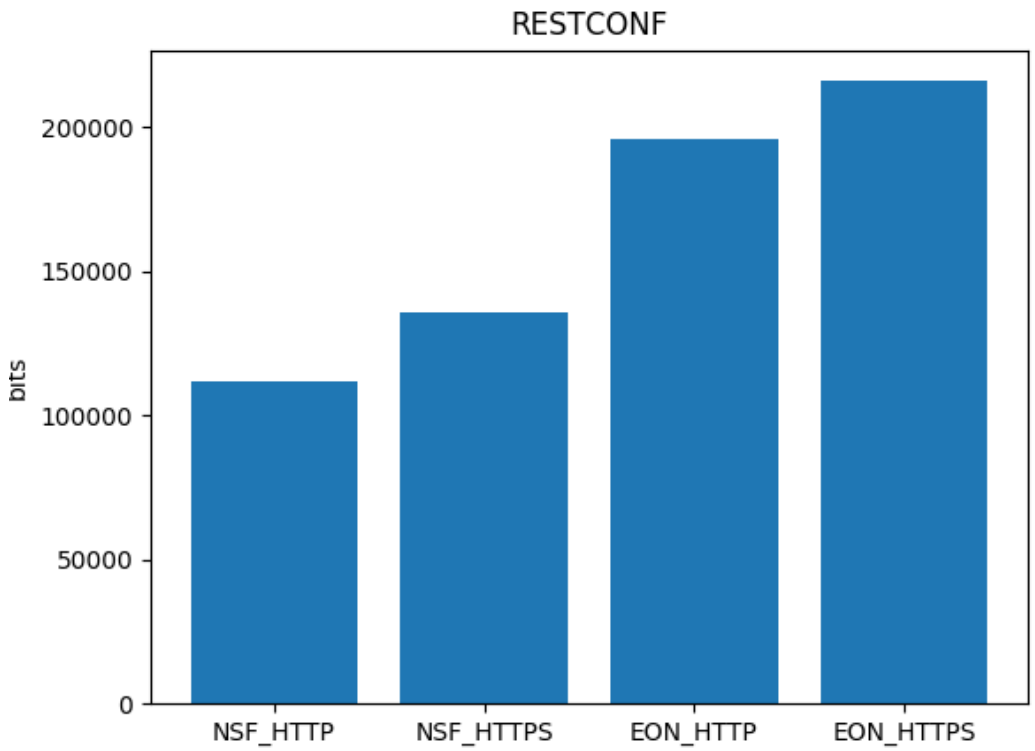
RESTCONF	Netconf
GET	<get-config>, <get>
POST	<edit-config> (“create”)
PUT	<edit-config> (“replace”)
PATCH	<edit-config> (“merge”)
DELETE	<edit-config> (“delete”)
OPTIONS	(discover supported operations)
HEAD	(get without body)

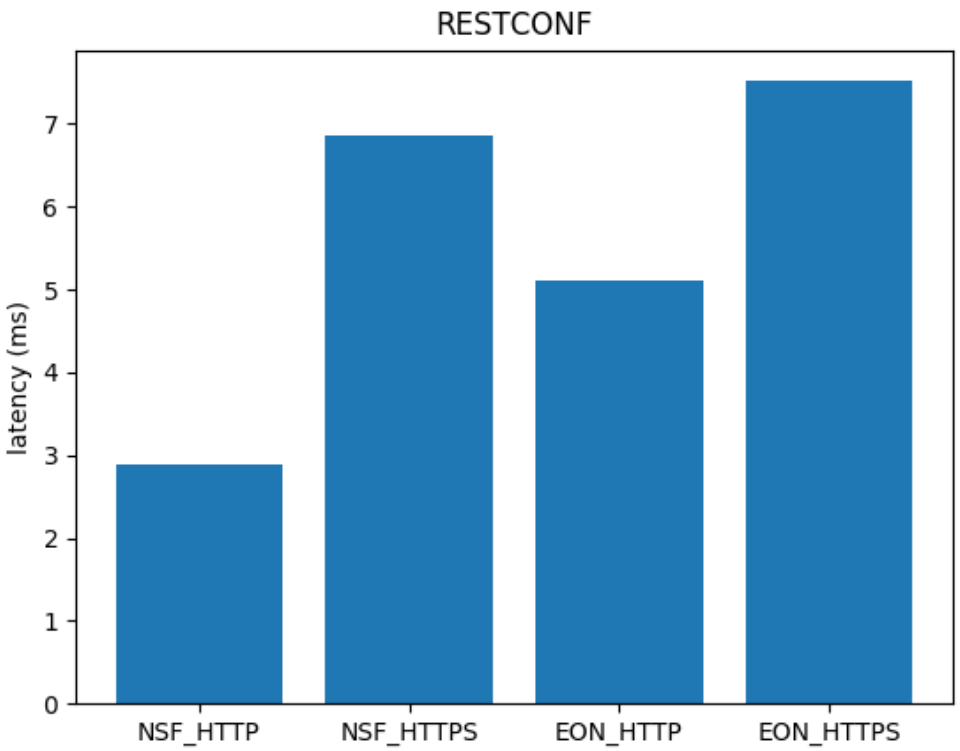
RESTCONF HTTP tree

RESTCONF is a REST-like protocol that provides a HTTP-based API to access the data, modeled by YANG. The REST-like operations are used to access the hierarchical data within a datastore. The information modeled in YANG is structured in the following tree:

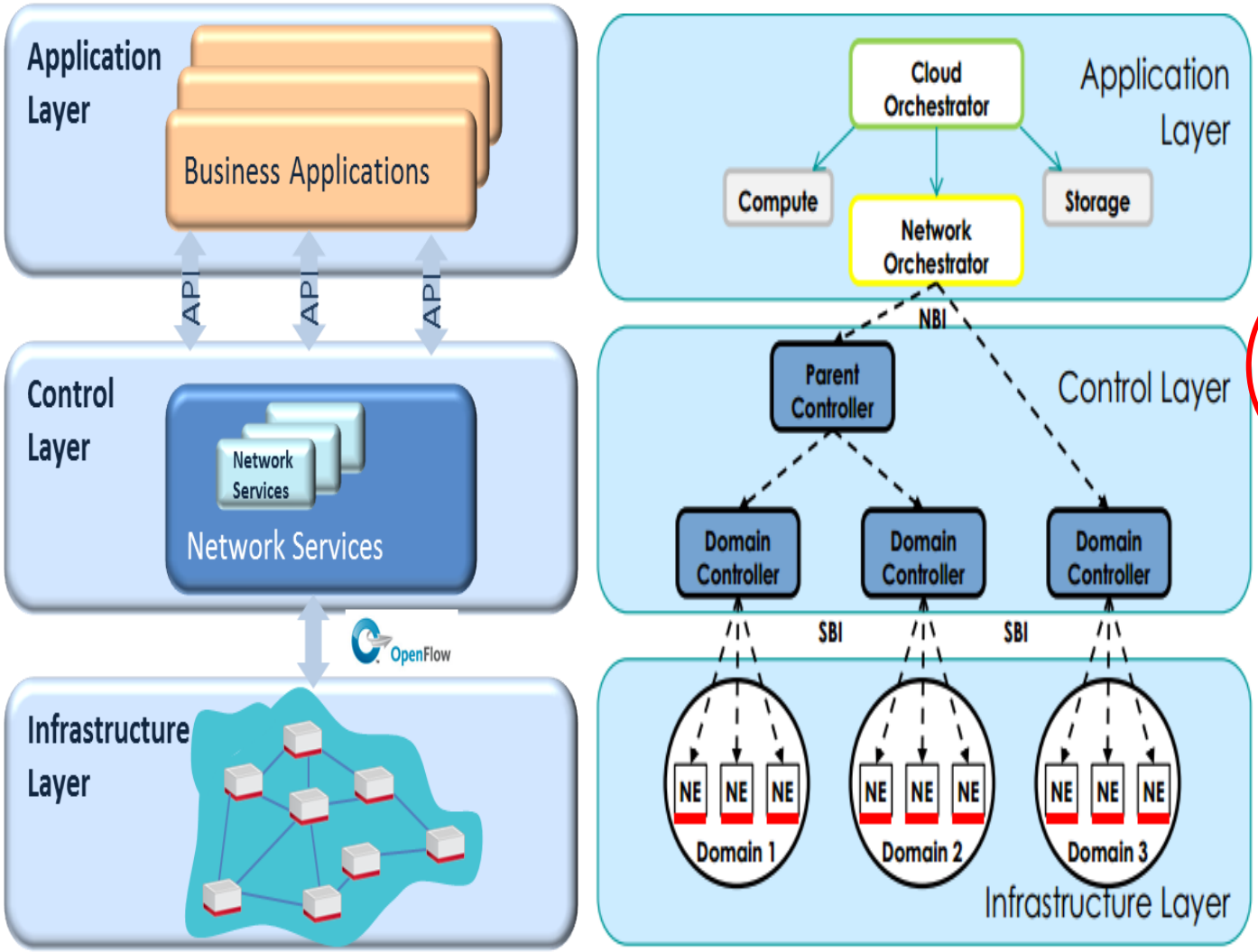
- /restconf/data : “Data (configuration/operational) accessible from the client”
- /restconf/modules : “Set of YANG models supported by the RESTCONF server”
- /restconf/operations : “Set of operations (**YANG-defined RPCs**) supported by the server”
- /restconf/streams: “Set of notifications supported by the server”

Example RESTCONF comparison





Transport SDN controller



TAPI

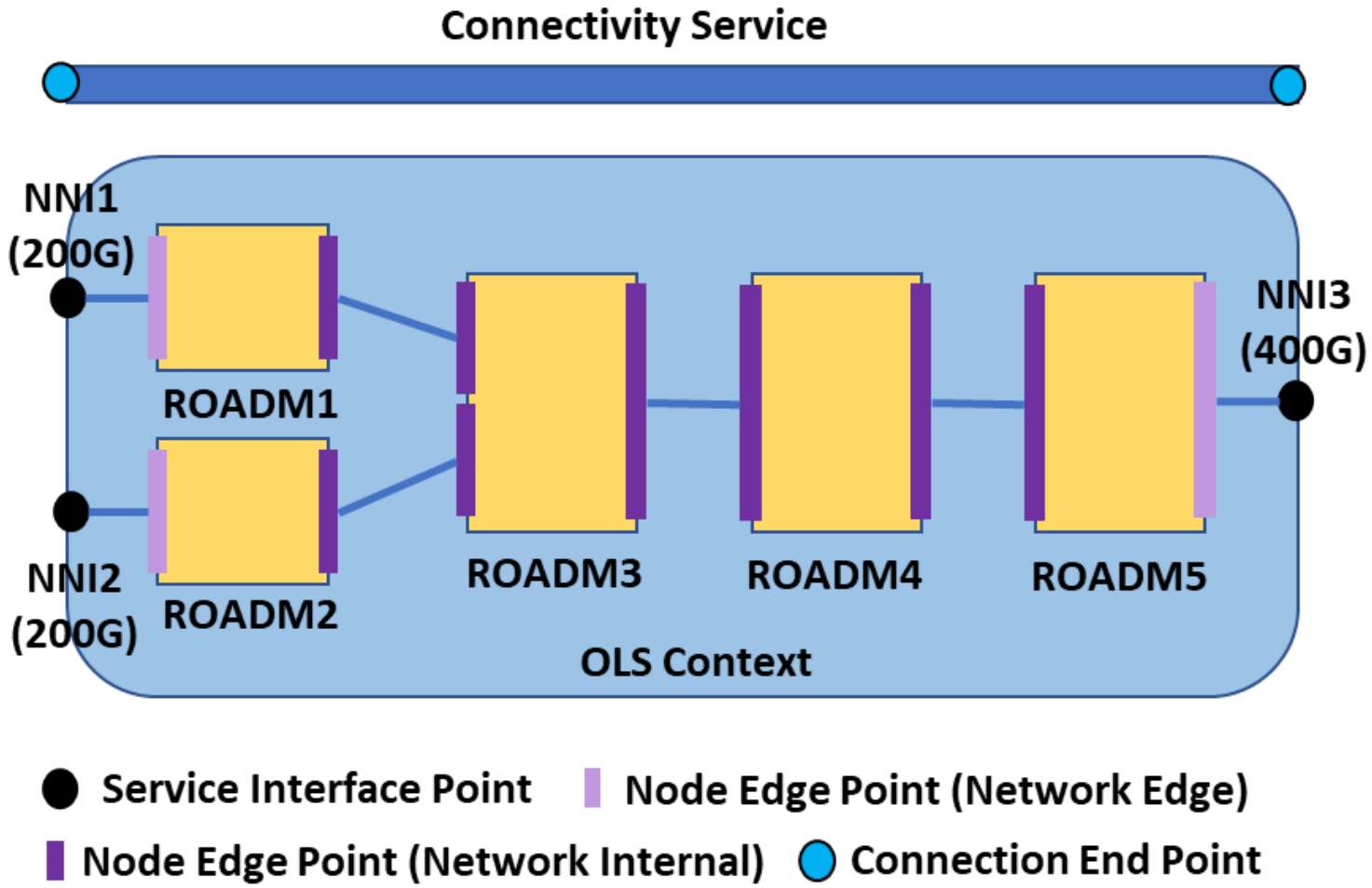
REST APIs

Service Request
Topology

OpenFlow

Control to Dataplane
Control to Virtual Network

TAPI Basic concepts



TAPI Context, Topology & Connectivity Overview

TeraFlow SDN

All TAPI interaction between an TAPI provider (SDN Controller) and an TAPI Client (Application, Orchestrator or parent SDN Controller) occur within a shared "*Context*"

TAPI *Context* is defined by a set of *ServiceInterfacePoints* (and some policy)

- *ServiceInterfacePoints* enable TAPI Client to request TAPI Services between them.

A TAPI provider may expose 1 or more abstract *Topology* within shared *Context*

- These topologies may or may-not map 1-to-1 to a provider's internal topology.

A *Topology* is expressed in terms of *Nodes* and *Links*.

- *Nodes* aggregate *NodeEdgePoints, Links* connect 2 Nodes & terminate on *NodeEdgePoints*
- *NodeEdgePoints* may be mapped to 1 or more *ServiceInterfacePoints* at edge of Network

TAPI Client requests *ConnectivityService* between 2 or more *ServiceInterfacePoints*

TAPI Provider creates 1 or more *Connections* in response to *ConnectivityService*

- *ConnectionEndPoints* encapsulate information related to a *Connection* at the ingress/egress points of every *Node* that the *Connection* traverses in a *Topology*
- Every *ConnectionEndPoint* is supported by a specific "parent" *NodeEdgePoint*
- Thus with reference to *ConnectivityServices*, a *ServiceInterfacePoint* conceptually represents a pool of "potential" *ConnectionEndPoints* at the edge of the Network

Launch/Run TAPI Reference Implementation

TeraFlow SDN

Run in a terminal:

```
$ cd ~/tfs-ctrl/hackfest/tapi/server
$ python3 -m tapi_server 8080 database/mini-ols-context.json
```

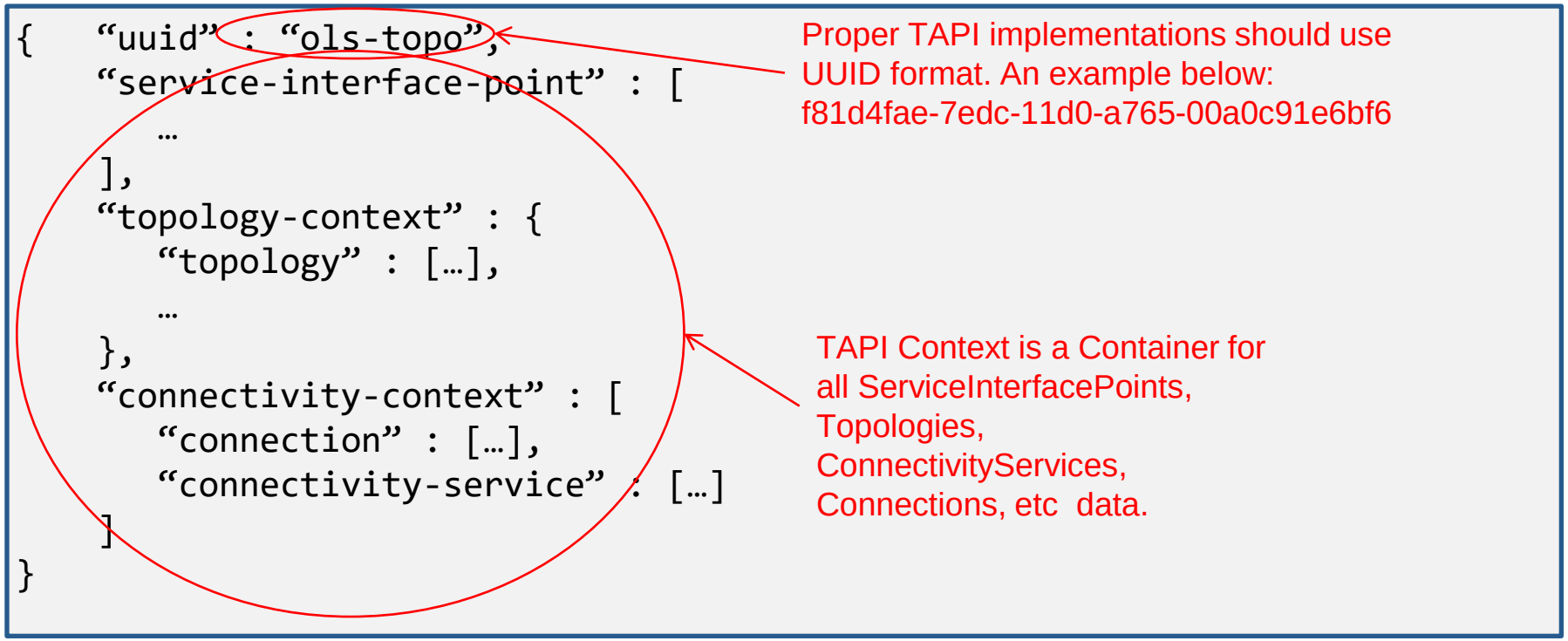
Run in a new terminal:

```
$ cd ~/tfs-ctrl/hackfest/tapi/client
$ curl -X GET -H "Content-Type: application/json" \
        http://127.0.0.1:8080/restconf/data/tapi-common:context/
```

TAPI: Retrieve Context

curl -X GET -H "Content-Type: application/json" http://127.0.0.1:8080/restconf/data/tapi-common:context/

Response:



TAPI: Retrieve Service Interface Point Details

TeraFlow SDN

curl -X GET -H "Content-Type: application/json" \

http://127.0.0.1:8080/restconf/data/tapi-common:context/service-interface-point=node-4-port-16-output

Response:


```
{
 "administrative-state": "UNLOCKED",
  "direction": "OUTPUT",
  "layer-protocol-name": "PHOTONIC_MEDIA",
  "operational-state": "ENABLED",
  "supported-layer-protocol-qualifier": [
    "tapi-photonic-media:PHOTONIC_LAYER_QUALIFIER_NMC"
  ],
  "uuid": "node-4-port-16-output"
}
```


Most TAPI objects have layer & state attributes

TAPI: Retrieve Topology

TeraFlow SDN

curl -X GET -H "Content-Type: application/json" \

http://127.0.0.1:8080/restconf/data/tapi-common:context/tapi-topology:topology-context/

Response:


```
{
  "nw-topology-service":
  {
      "topology": [
        {
          "topology-uuid": "ols-topo"
        }
      ],
      "uuid": "ols-ctx"
  },
  "topology": [ ...]
}
```

TAPI: Retrieve Topology Details

curl -X GET -H "Content-Type: application/json" \

http://127.0.0.1:8080/restconf/data/tapi-common:context/tapi-topology:topology-context/topology=ols-topo/

Response:


```
{
   “uuid”: “ols-topo”,
   “name”: [
      { “value-name”: “name”,
        “value”: “NETWORK_TOPOLOGY”
      }
      …
    ],
   “layer-protocol-name”: [ “PHOTONIC_MEDIA”, “DSR” ],
   “node” : [ {…}, … ],
   “link” : [ {…}, … ]
}
```


Every TAPI object has a name attribute that is defined as a list of name-value pairs.

Topology defines the network layers it covers.

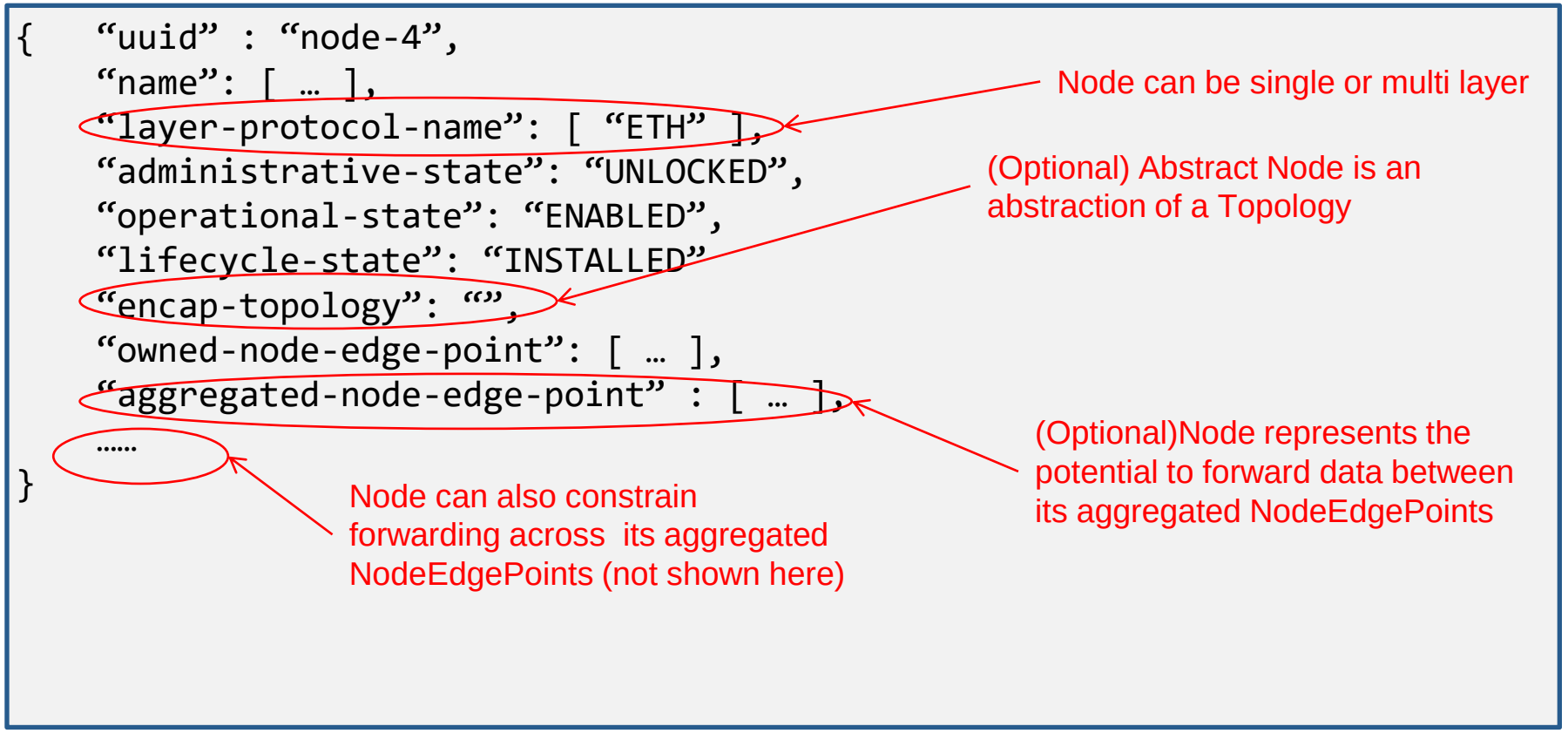
Topology contains Nodes & Links (by value).

TAPI: Retrieve Node Details - 1

curl -X GET -H "Content-Type: application/json" \

http://127.0.0.1:8080/restconf/data/tapi-common:context/tapi-topology:topology-context/topology=ols-topo/node=node-4/

Response:



TAPI: Retrieve Node Details - 2

curl -X GET -H "Content-Type: application/json" \

http://127.0.0.1:8080/restconf/data/tapi-common:context/tapi-topology:topology-context/topology=ols-topo/node=node-4/

Response:

```
{   “uuid” : “node-4”,
    “name”: [ … ],
    “layer-protocol-name”: [“ETH” ],
    “administrative-state”: “UNLOCKED”,
    “operational-state”: “ENABLED”,
    “lifecycle-state”: “INSTALLED”
    “encap-topology”: “”,
    “owned-node-edge-point”: [
        {…},
        …
    ]
}
```

Switch Node is typically single layer

Switch Node contains/owns a list of NodeEdgePoints

TAPI: NodeEdgePoint Details

curl -X GET -H "Content-Type: application/json" \

http://127.0.0.1:8080/restconf/data/tapi-common:context/tapi-topology:topology-context/topology=ols-topo/node=node-4/owned-node-edge-point=node-4-port-13/

Response:


```
{   “uuid” : “node-4-port-13”,
    “name”: [ … ],
    “layer-protocol-name”: “PHOTONIC_MEDIA”,
    “administrative-state”: “UNLOCKED”,
    “operational-state”: “ENABLED”,
    “lifecycle-state”: “INSTALLED”
    “termination-state”: “LP_CAN_NEVER_TERMINATE”,
    “termination-direction”: “BIDIRECTIONAL”,
    “link-port-direction”: “BIDIRECTIONAL”,
    “link-port-role”: “SYMMETRIC”,
    “mapped-service-interface-point” : [
      {"service-interface-point-uuid": "node-4-port-13-output"},
      {"service-interface-point-uuid": "node-4-port-13-input"}
    ]
}
```


NodeEdgePoint is single layer

NodeEdgePoint can be mapped to (1 or more) ServiceInterfacePoint to function as a network interface. This attribute is empty for “internal” NodeEdgePoints

TAPI: Retrieve Link Details

curl -X GET -H "Content-Type: application/json" \

http://127.0.0.1:8080/restconf/data/tapi-common:context/tapi-topology:topology-context/topology=ols-topo/link=link-4:2-2:4/

Response:


```
{
  "cost-characteristic": [{"cost-name": "te-metric","cost-value": "1"}],
  "direction": "UNIDIRECTIONAL",
  "layer-protocol-name": [“PHOTONIC_MEDIA"],
  "node-edge-point": [
    {
      "node-edge-point-uuid": "node-4-port-2",
      "node-uuid": "node-4"
    },
    {
      "node-edge-point-uuid": "node-2-port-4",
      "node-uuid": "node-2"
    }
  ],
  "uuid": "link-4:2-2:4"
}
```

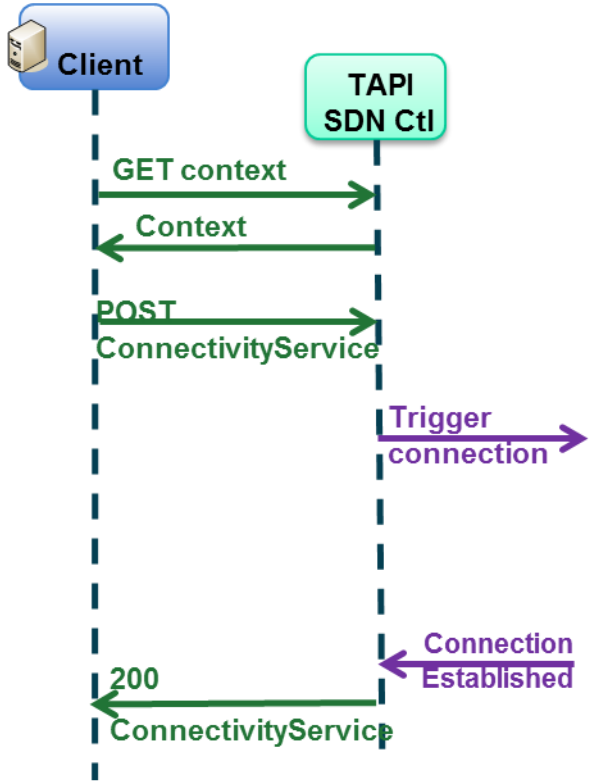

Link conveys “cost-characteristic” information

Link represents adjacency information between 2 NodeEdgePoints

TAPI: Connectivity Service workflow

TeraFlow SDN

```
$ cd ~/tfs-ctrl/hackfest/tapi/client
```



TAPI: Establish Connectivity Service

curl -X POST -H "Content-Type: application/json" \

http://127.0.0.1:8080/restconf/data/tapi-common:context/tapi-connectivity:connectivity-context/connectivity-service/ -d @cs1.json

cs1.json:

```
{
  "tapi-connectivity:connectivity-service": [
    {
      "uuid" : "cs1",
      "service-type":"POINT_TO_POINT_CONNECTIVITY",
      "connectivity-constraint": {
        "requested-capacity": {"total-size": { "value": "50", "unit": "GHz" }},
        "connectivity-direction": "UNIDIRECTIONAL"
      },
      "end-point": [
        {
          "local-id": "csep-1",
          "layer-protocol-name": "PHOTONIC_MEDIA",
          "service-interface-point": {
            "service-interface-point-uuid": "node-1-port-13-input"
          }
        },
        {
          "local-id":"csep-2",
          "layer-protocol-name":"PHOTONIC_MEDIA",
          "service-interface-point": {
            "service-interface-point-uuid": "node-2-port-14-output"
          }}]}]}
```

ConnectivityService endpoint information has to specify the ServiceInterfacePoint

TAPI: Retrieve Connection Details

TeraFlow SDN

curl -X GET -H "Content-Type: application/json" \

http://127.0.0.1:8080/restconf/data/tapi-common:context/tapi-connectivity:connectivity-context/connection=cs1/

Response:

```
{
  "uuid": "cs1",
  "connection-end-point": [
    {
      "connection-end-point-uuid": "cep13",
      "node-edge-point-uuid": "node-1-port-3"
    },
    {
      "connection-end-point-uuid": "cep32",
      "node-edge-point-uuid": "node-3-port-2"
    }
  ]
}
```

ConnectivityService has triggered the establishment of a Connection

Node Edge Point is augmented with a list of Connection End Points

Other TAPI models

We have learned tapi-topology and tapi-connectivity, but there are other significant models:

- Notifications
- Path Computation
- Virtual Network
- OAM
- Technological augments:
  - Eth
  - ODU
  - PHOTONIC_MEDIA

TAPI Optical Augments: node-edge-point

TeraFlow SDN

```
augment /tapi-common:context/tapi-topology:topology-context/tapi-topology:topology/
                tapi-topology:node/tapi-topology:owned-node-edge-point:
+--ro media-channel-node-edge-point-spec
   +--ro mc-pool
      +--ro supportable-spectrum* [upper-frequency lower-frequency]
      |  +--ro upper-frequency         uint64
      |  +--ro lower-frequency         uint64
      |  +--ro frequency-constraint
      |     +--ro adjustment-granularity?   adjustment-granularity
      |     +--ro grid-type?                grid-type
      +--ro available-spectrum* [upper-frequency lower-frequency]
      |  +--ro upper-frequency         uint64
      |  +--ro lower-frequency         uint64
      |  +--ro frequency-constraint
      |     +--ro adjustment-granularity?   adjustment-granularity
      |     +--ro grid-type?                grid-type
      +--ro occupied-spectrum* [upper-frequency lower-frequency]
         +--ro upper-frequency         uint64
         +--ro lower-frequency         uint64
         +--ro frequency-constraint
         +--ro adjustment-granularity?   adjustment-granularity
         +--ro grid-type?                grid-type
```

TAPI Optical Augments: connection-end-point

TeraFlow SDN

```
module: tapi-photonic-media
augment /tapi-common:context/tapi-topology:topology-context/tapi-topology:topology/tapi-topology:node/
                    tapi-topology:owned-node-edge-point/tapi-connectivity:cep-list/tapi-connectivity:connection-end-point:
+--ro otsi-connection-end-point-spec
   +--ro otsi-termination
      +--ro selected-central-frequency
      |  +--ro frequency-constraint
      |  |  +--ro adjustment-granularity?   adjustment-granularity
      |  |  +--ro grid-type?                grid-type
      |  +--ro central-frequency?      uint64
      +--ro selected-application-identifier
      |  +--ro application-identifier-type?   application-identifier-type
      |  +--ro application-code?              string
      +--ro selected-modulation?               modulation-technique
      +--ro selected-spectrum
      |  +--ro upper-frequency?        uint64
      |  +--ro lower-frequency?        uint64
      |  +--ro frequency-constraint
      |     +--ro adjustment-granularity?   adjustment-granularity
      |     +--ro grid-type?                grid-type
      +--ro transmited-power
      |  +--ro total-power?              decimal64
      |  +--ro power-spectral-density?   decimal64
      +--ro received-power
      |  +--ro total-power?              decimal64
      |  +--ro power-spectral-density?   decimal64
      +--ro laser-properties
         +--ro laser-status?            laser-control-status-type
         +--ro laser-application-type?   laser-type
         +--ro laser-bias-current?       decimal64
         +--ro laser-temperature?        decimal64
```

Exercise: Writing a TAPI Topology client

TeraFlow SDN

Objective:

- Retrieve and draw Network Topology using TAPI

Steps:

- Run TAPI-RI
- Load topological information
- Start coding using the following libraries:
  - NetworkX
  - Matplotlib / PyDot
  - Requests
  - Json

Time: 10min

TAPI_APP

TeraFlow SDN

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import json, requests, networkx as nx
from requests.auth import HTTPBasicAuth

IP = '127.0.0.1'
PORT = 8080
TOPO_UUID = 'ols-topo'
TOPO_URL = 'http://{:s}:{:d}/restconf/data/tapi-common:context/'\
    'tapi-topology:topology-context/topology={:s}/'

def retrieve_topology(ip, port, topo_uuid, user='', passwd=''):
    print ("Reading network-topology")
    topo_url = TOPO_URL.format(ip, port, topo_uuid)
    response = requests.get(topo_url, auth=HTTPBasicAuth(user, passwd))
    topology = response.json()
    print ("Retrieved Topology: " + json.dumps(topology, indent=4))
    return topology

def to_png_matplotlib(nwk_graph, topo_uuid):
    import matplotlib.pyplot as plt
    nx.draw(nwk_graph, pos=nx.spring_layout(nwk_graph, scale=500))
    plt.show(block=False)
    plt.savefig('{:s}.png'.format(topo_uuid), format='PNG')
    plt.close()

def to_png_pydot(nwk_graph, topo_uuid):
    from networkx.drawing.nx_pydot import write_dot, to_pydot
    write_dot(nwk_graph, '{:s}.dot'.format(topo_uuid))
    dot_graph = to_pydot(nwk_graph)
    with open('{:s}.png'.format(topo_uuid), 'wb') as f:
        f.write(dot_graph.create(format='png'))

def draw_topology(topology):
    nwk_graph = nx.DiGraph()

    for node in topology['node']:
        if node['owned-node-edge-point']:
            nwk_graph.add_node(node['uuid'])

    for link in topology['link']:
        node1 = link['node-edge-point'][0]['node-uuid']
        node2 = link['node-edge-point'][1]['node-uuid']
        nwk_graph.add_edge(node1, node2)

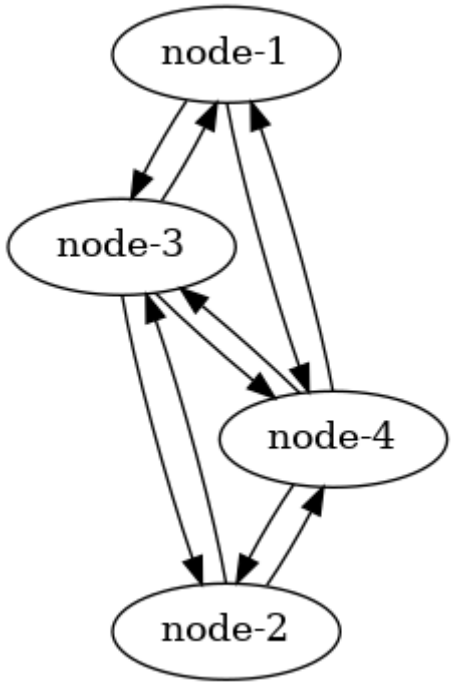
    #to_png_matplotlib(nwk_graph, topology['uuid'])
    to_png_pydot(nwk_graph, topology['uuid'])

if __name__ == "__main__":
    draw_topology(retrieve_topology(IP, PORT, TOPO_UUID))
```

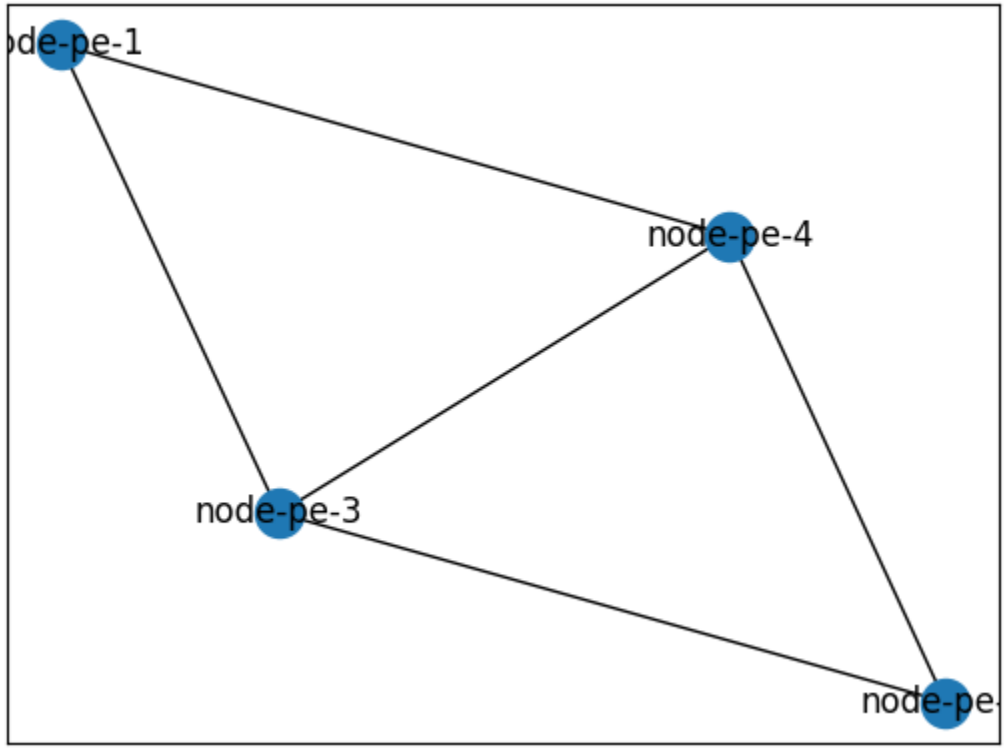
Run TAPI Application Client

Run in a terminal:

```
$ cd ~/tfs-ctrl/hackfest/tapi/tapi_app
$ python3 tapi_app.py
```



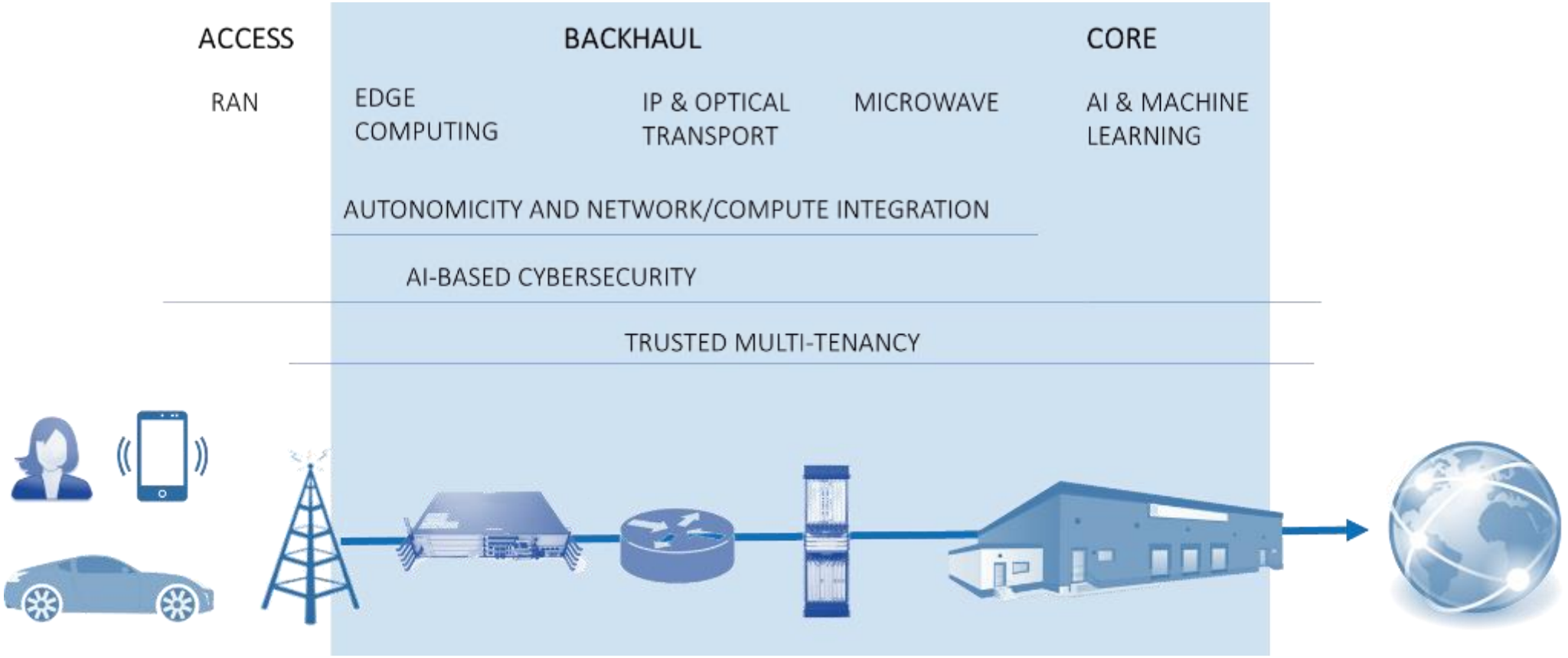
PyDot



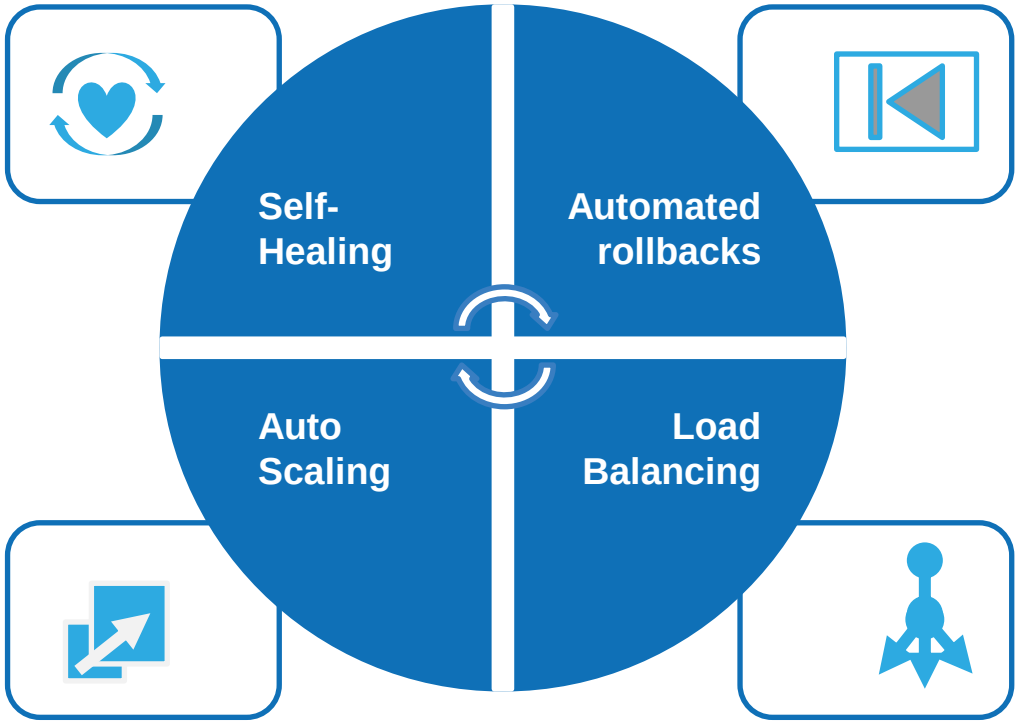
MatPlotLib

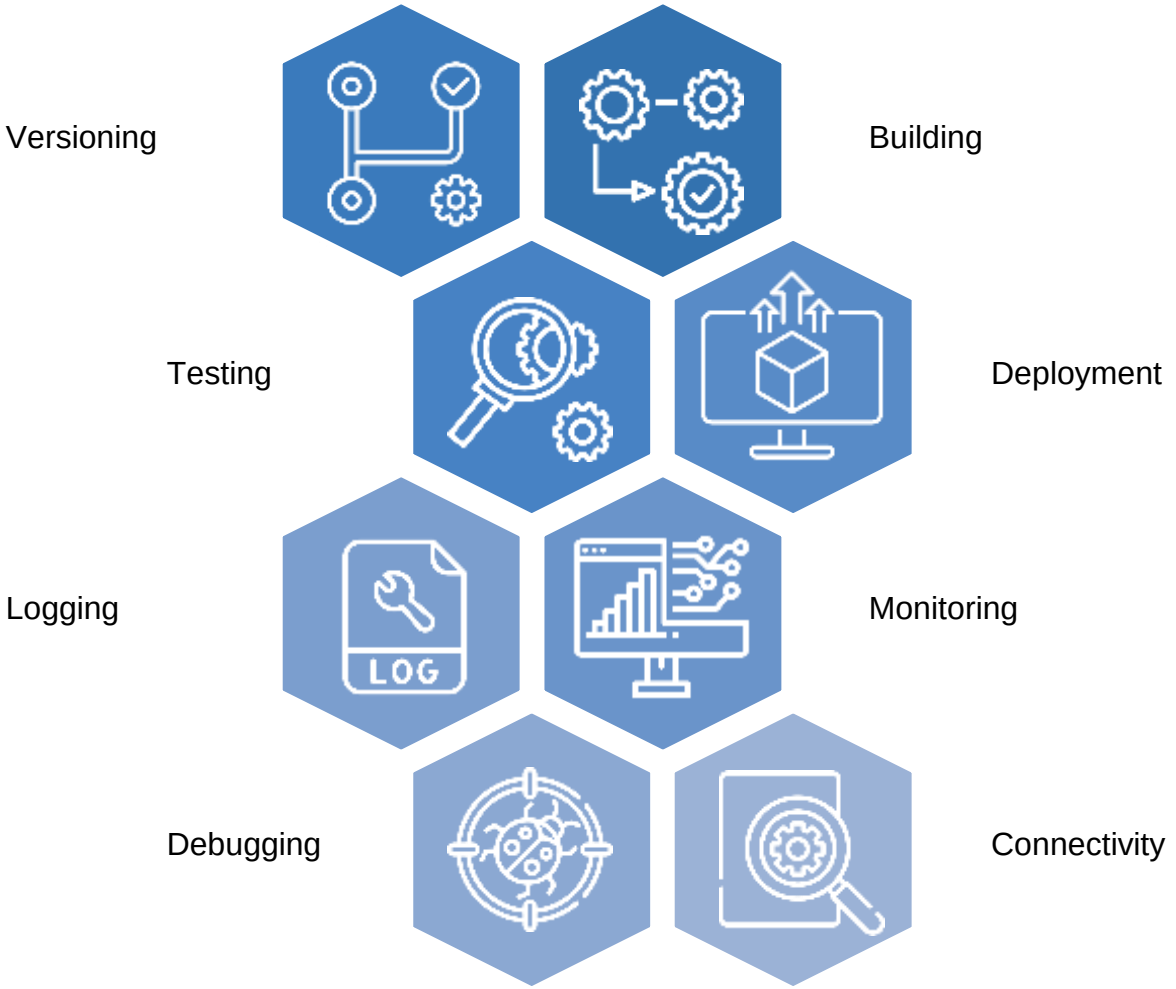
ETSI TeraFlowSDN

Networks have become more complex



Benefits and Challenges of TFS cloud-native approach





ETSI TeraFlowSDN

TeraFlow SDN

Open-source

IP over DWDM

Cloud-native

Compliant with European Cyber-security requirements

Community-driven

Global community

Focus on operator use cases

Aligned with TIP MUST requirements

Who are we?

TeraFlow SDN

- Members

ttc Centre Tecnològic de Telecomunicacions de Catalunya

DEMOKRITOS

Fraunhofer HHI

POLITÉCNICA

SME siae microelettronica

Telefónica

telenor

- Participants (Non-ETSI members)

CHALMERS UNIVERSITY OF TECHNOLOGY

Globe

it instituto de telecomunicações

NTNU Norwegian University of Science and Technology

OLD DOG CONSULTING

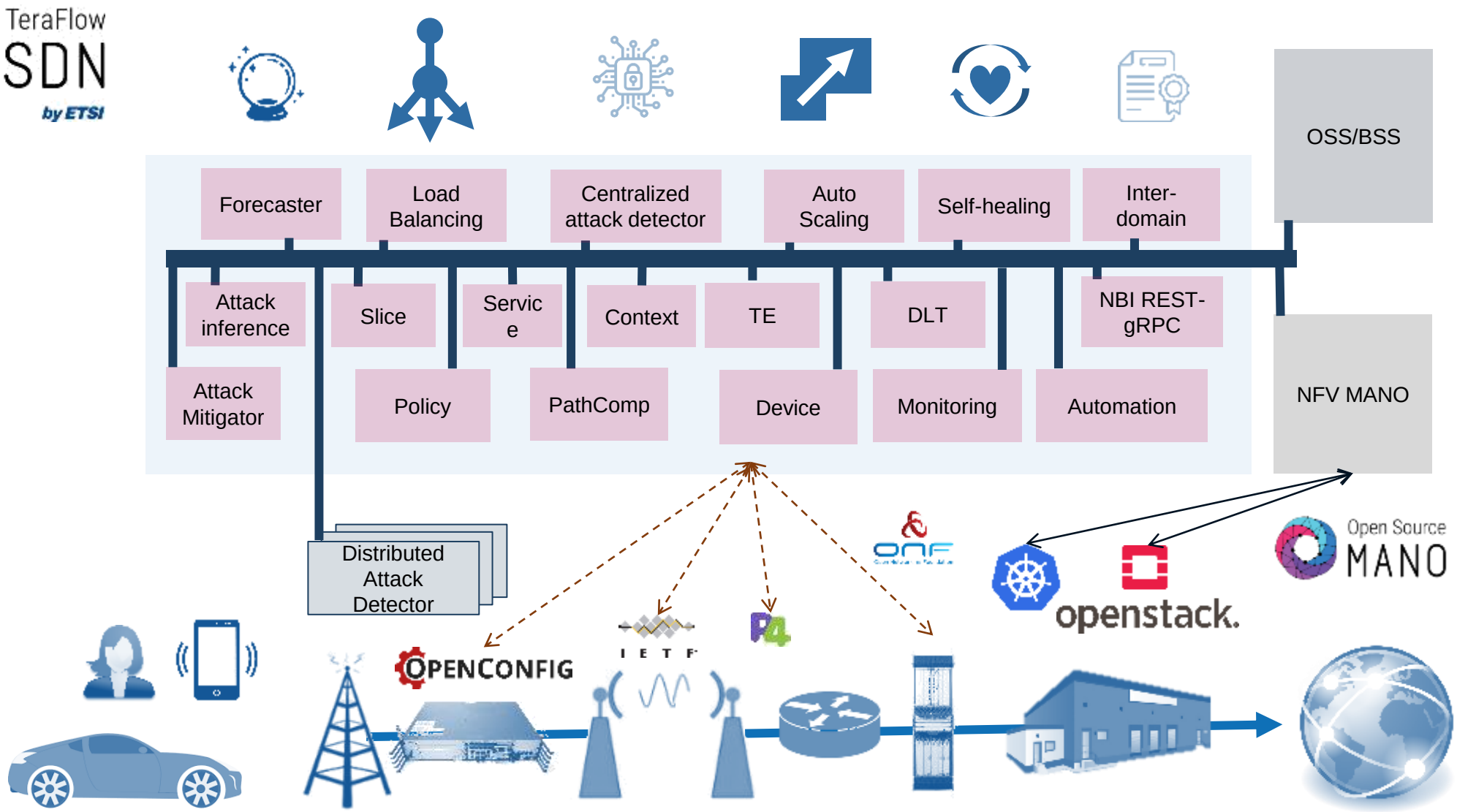
DIPL. PHYS. PEER STRITZINGER GMBH

UBITECH ubiquitous solutions

ETSI OSG TeraFlowSDN

- Provide a toolbox for rapid prototyping and experimentation with innovative network technologies and use cases.
- TeraFlowSDN aims to ease the adoption of SDN by telco operators.
- We want to bring innovation to the ecosystem and contribute to network programmability for current 5G and beyond deployments.
- TeraFlowSDN will ease proof-of-concept demonstration for many ETSI ISGs to demonstrate the proposed standard solutions faster.

TFS architecture for release 2



Deploy TeraFlowSDN controller

TeraFlowSDN controller runs a number of microservices on top of a Kubernetes-based environment.

- The minimal requirements are:
  - Ubuntu 20.04 LTS operating system (server or desktop)
  - 4 vCPUs @ 100% execution capacity
  - 8 GB of RAM
  - 40 GB of storage disk

For the sake of simplicity, we provide a pre-installed Ubuntu 20.04 VM with MicroK8s installed.

- To perform your own installation, follow the steps described in the Tutorial:
  - https://labs.etsi.org/rep/tfs/controller/-/blob/develop/tutorial/1-1-create-vm.md
  - https://labs.etsi.org/rep/tfs/controller/-/blob/develop/tutorial/1-2-install-microk8s.md

Deploy TeraFlowSDN controller

Before continuing, check the status of your MicroK8s environment as described in:

- Check status of Kubernetes (https://labs.etsi.org/rep/tfs/controller/-/blob/develop/tutorial/1-2-install-microk8s.md#126-check-status-of-kubernetes)

```
$ microk8s.status --wait-ready
```

- Check all resources in Kubernetes (https://labs.etsi.org/rep/tfs/controller/-/blob/develop/tutorial/1-2-install-microk8s.md#127-check-all-resources-in-kubernetes)

```
$ microk8s.kubectl get all --all-namespaces
```

Deploy TeraFlowSDN controller

Specifications to deploy TeraFlowSDN are defined in a bash script as a set of enviroment variables.

- Example: see “my_deploy.sh”

```
# Set the URL of your local Docker registry where the images will be uploaded to.
export TFS_REGISTRY_IMAGE="http://localhost:32000/tfs/"

# Set the list of components, separated by spaces, you want to build images for, and deploy.
export TFS_COMPONENTS="context device automation monitoring pathcomp service slice compute webui"

# Set the tag you want to use for your images.
export TFS_IMAGE_TAG="dev"

# Set the name of the Kubernetes namespace to deploy to.
export TFS_K8S_NAMESPACE="tfs"

# Set additional manifest files to be applied after the deployment
export TFS_EXTRA_MANIFESTS="manifests/nginx_ingress_http.yaml"

# Set the neew Grafana admin password
export TFS_GRAFANA_PASSWORD="admin123+"
```

Deploy TeraFlowSDN controller

If you want to tweak the deployment specifications, create a copy of “my_deploy.sh” script.

When you are fine with your specifications, perform the deployment as follows:

```
$ cd ~/tfs-ctrl
$ source my_deploy.sh
$ ./deploy.sh
```

The deployment might take few minutes…

- You should see the progress of the deployment.

```
Deleting and Creating a new namespace...
namespace "tfs" deleted
namespace/tfs created

Deploying components and collecting environment variables...
Processing 'context' component...
  Building Docker image...
  Pushing Docker image to 'http://localhost:32000/tfs/'...
  Adapting 'context' manifest file...
  Deploying 'context' component to Kubernetes...
  Collecting env-vars for 'context' component...
…

Deploying extra manifests...
Processing manifest 'manifests/nginx_ingress_http.yaml'...
ingress.networking.k8s.io/tfs-ingress created

Waiting for 'MonitoringDB' component...
statefulset.apps/monitoringdb condition met

Waiting for 'context' component...
deployment.apps/contextservice condition met
…

Configuring WebUI DataStores and Dashboards...
Connecting to grafana at URL: http://admin:admin@127.0.0.1:80/grafana...
…
Creating a datasource...
…
```

Deploy TeraFlowSDN controller

The process concludes reporting the status of the microservices.

You can always retrieve this status as follows:

```
$ cd ~/tfs-ctrl
$ source my_deploy.sh
$ ./show_deploy.sh
```

```
Deployment Resources:
NAME                                  READY   STATUS    RESTARTS        AGE
pod/contextservice-55f7f77f-dqfsc     2/2     Running   0               5m12s
pod/deviceservice-67fb99b9dd-cjcsk    1/1     Running   0               5m1s
…

NAME                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
service/contextservice      ClusterIP   10.152.183.225   <none>        1010/TCP,8080/TCP
service/deviceservice       ClusterIP   10.152.183.194   <none>        2020/TCP
…

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/contextservice      1/1     1            1           5m12s
deployment.apps/deviceservice       1/1     1            1           5m1s
…

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/contextservice-55f7f77f       1         1         1       5m12s
replicaset.apps/deviceservice-67fb99b9dd      1         1         1       5m1s
…

NAME                              READY   AGE
statefulset.apps/monitoringdb     1/1     4m44s

Deployment Ingress:
NAME          CLASS    HOSTS   ADDRESS     PORTS   AGE
tfs-ingress   public   *       127.0.0.1   80      3m15s
```

Exercise: Onboard Devices using TeraFlowSDN

TeraFlowSDN enables to create entities through JSON-based descriptor files.

- Example (context & topology, see ~/tfs-ctrl/hackfest/tfs-descriptors/context-topology.json)

```
{
  "contexts": [
    {
      "context_id": {"context_uuid": {"uuid": "admin"}},
        "topology_ids": [],
        "service_ids": []
    }
  ],
  "topologies": [
    {
      "topology_id": {
        "context_id": {"context_uuid": {"uuid": "admin"},
        "topology_uuid": {"uuid": "admin"}
      }},
      "device_ids": [],
      "link_ids": []
    }
  ]
}
```

Exercise: Onboard Devices using TeraFlowSDN

Connect TFS to an OLS controller supporting TAPI

- Example (see ~/tfs-ctrl/hackfest/tfs-descriptors/device-tapi-ols.json)

```
{
    "devices": [
        {
            "device_id": {"device_uuid": {"uuid": "OLS"}},
            "device_type": "open-line-system",
            "device_config": {"config_rules": [
                {"action": 1, "custom": {"resource_key": "_connect/address", "resource_value": "10.0.2.10"}},
                {"action": 1, "custom": {"resource_key": "_connect/port", "resource_value": "8080"}},
                {"action": 1, "custom": {"resource_key": "_connect/settings", "resource_value": {"timeout": 120}}}
            ]},
            "device_operational_status": 1,
            "device_drivers": [2],
            "device_endpoints": []
        }
    ]
}
```

Check “src/common/DeviceTypes.py”

Set IP Address of your VM & port of TAPI server

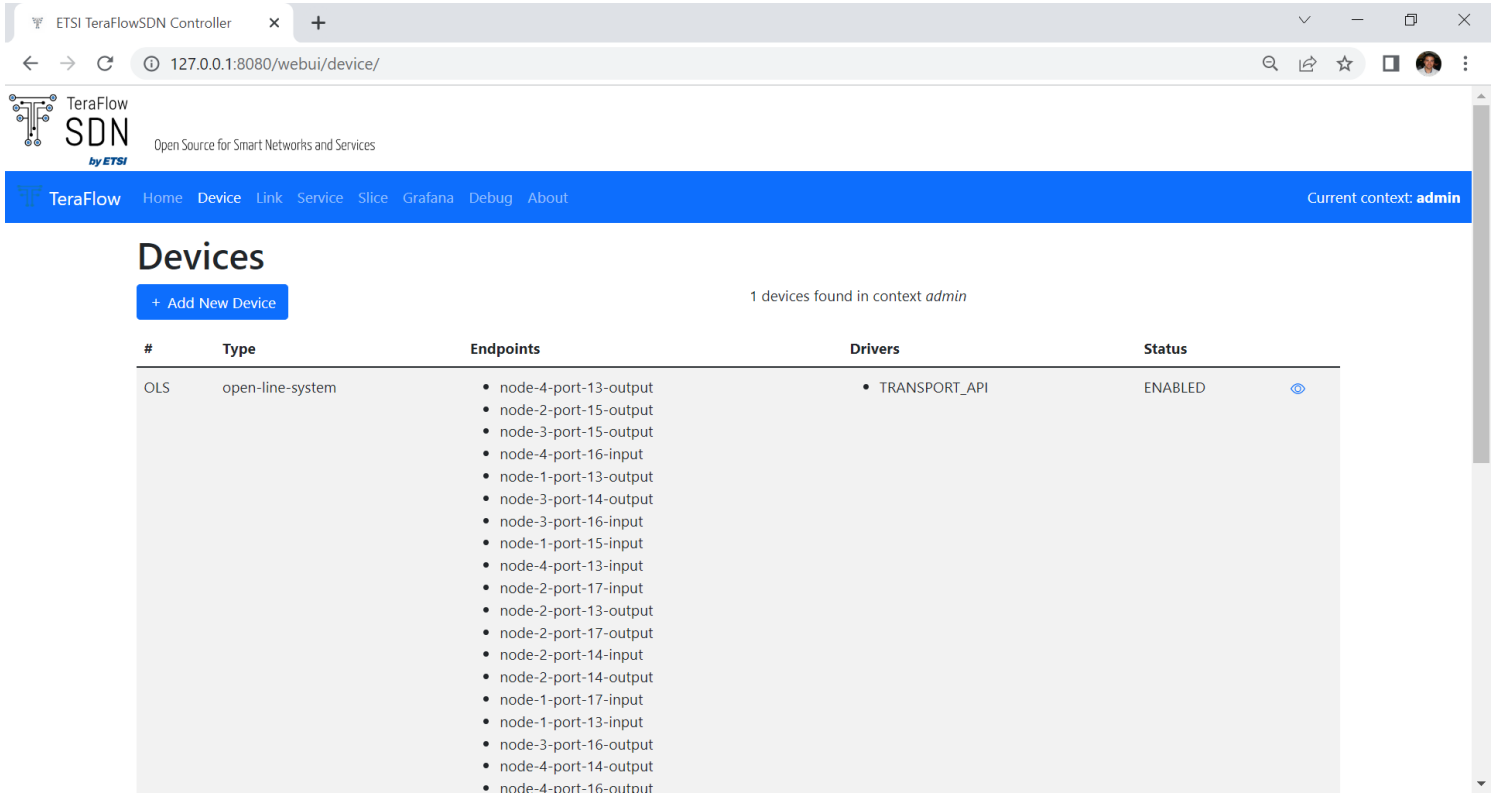
By default, DISABLED, Will be activated by Automation.
Check “proto/context.proto” “DeviceOperationalStatusEnum”

Use TAPI driver for this device descriptor.
Check “proto/context.proto” “DeviceDriverEnum”

Automatically discovered from TAPI server by Device component.

Exercise: Onboard Devices using TeraFlowSDN

You should see the device in WebUI with the endpoints discovered (from SIPs)



Exercise: Onboard Devices using TeraFlowSDN

Check the logs of the TeraFlowSDN components:

- Example: Device component

```
$ cd ~/tfs-ctrl
$ source my_deploy.sh
$ scripts/show_logs_device.sh
```

TeraFlow SDN

Programmable L3 routers with OpenConfig

OpenConfig Projects

Data models

Models for common configuration and operational state across platforms

Streaming telemetry

Scalable, secure, real-time monitoring with modern streaming protocols

RPCs and tools

Management RPC specs and implementations
Tooling to build config and monitoring stacks

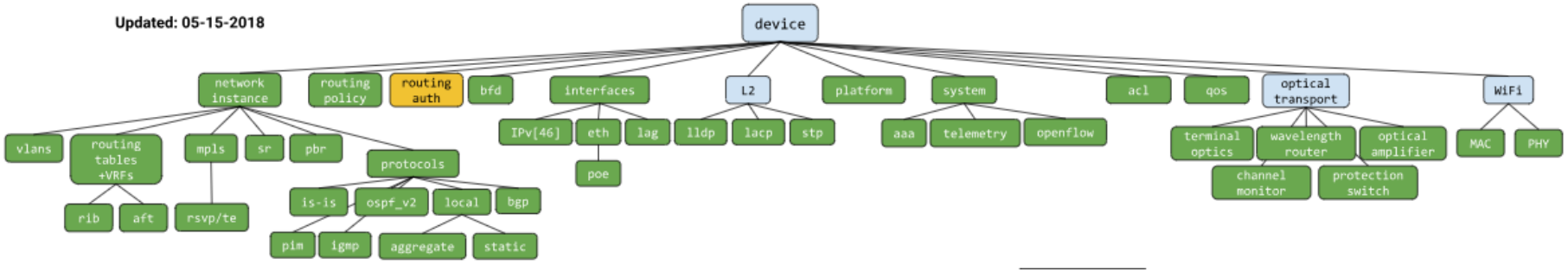
OpenConfig

Data models for configuration and operational state, written in YANG

Initial focus: device data for switching, routing, and transport

Development priorities driven by operator requirements

Technical engagement with major vendors to deliver native implementations



OpenConfig Data Model Principles

Modular model definition

Model structure combines

- Configuration (intended)
- Operational data (applied config and derived state)

Each module subtree declares config and state containers.

Model backward compatibility

- Driven by use of semantic versioning (xx.yy.zz)
- Diverges from IETF YANG guidelines (full compatibility)

String patterns (regex) follow POSIX notation (instead of W3C as defined by IETF)

```
module: openconfig-bgp
tree-path /bgp/neighbors/neighbor/transport
    +--rw bgp!
       +--rw neighbors
          +--rw neighbor* [neighbor-address]
             +--rw transport
                +--rw config
                |  +--rw tcp-mss?
                |  +--rw mtu-discovery?
                |  +--rw passive-mode?
                |  +--rw local-address?
                +--ro state
                   +--ro tcp-mss?
                   +--ro mtu-discovery?
                   +--ro passive-mode?
                   +--ro local-address?
                   +--ro local-port?
                   +--ro remote-address?
                   +--ro remote-port?
```

OpenConfig L3 data models – Interfaces

```
module: openconfig-interfaces
  +--rw interfaces
     +--rw interface* [name]
        +--rw name              -> ../config/name
        +--rw config
        |  +--rw name?            string
        |  +--rw type             identityref
        |  +--rw mtu?             uint16
        |  +--rw loopback-mode?   boolean
        |  +--rw description?     string
        |  +--rw enabled?         boolean
        +--ro state
        |  +--ro name?            string
        |  +--ro type             identityref
        |  +--ro admin-status     enumeration
        |  +--ro oper-status      enumeration
        |  ...
        |  +--ro counters
        |     +--ro in-octets?            oc-yang:counter64
        |     +--ro in-pkts?              oc-yang:counter64
        |     +--ro out-octets?           oc-yang:counter64
        |     +--ro out-pkts?             oc-yang:counter64
        |     ...
        |  ...
        |
        +--rw subinterfaces
           +--rw subinterface* [index]
           +--rw index     -> ../config/index
           +--rw config
           |  +--rw index?         uint32
           |  +--rw description?   string
           |  +--rw enabled?       boolean
           +--ro state
              ...
```

```
module: openconfig-vlan
  augment /oc-if:interfaces/oc-if:interface/oc-if:subinterfaces/oc-if:subinterface:
    +--rw vlan
       +--rw config
       |  x--rw vlan-id?   union
       +--ro state
       |  x--ro vlan-id?   union
       +--rw match
       |  +--rw single-tagged
       |  |  +--rw config
       |  |  |  +--rw vlan-id?   oc-vlan-types:vlan-id
       |  |  +--ro state
       |  |     +--ro vlan-id?   oc-vlan-types:vlan-id
       |  ...
       ...
```

```
module: openconfig-if-ip
  augment /oc-if:interfaces/oc-if:interface/oc-if:subinterfaces/oc-if:subinterface:
    +--rw ipv4
       +--rw addresses
       |  +--rw address* [ip]
       |     +--rw ip         -> ../config/ip
       |     +--rw config
       |     |  +--rw ip?              oc-inet:ipv4-address
       |     |  +--rw prefix-length?   uint8
       |     +--ro state
       |     |  +--ro ip?              oc-inet:ipv4-address
       |     |  +--ro prefix-length?   uint8
       |     |  +--ro origin?          ip-address-origin
       |     ...
       ...
```

OpenConfig L3 data models – Routing Policies

TeraFlow SDN

```
module: openconfig-routing-policy + openconfig-bgp-policy
  +--rw routing-policy
     +--rw defined-sets
     |  +--rw bgp-defined-sets  (augment from openconfig-bgp)
     |  |  +--rw ext-community-sets
     |  |  |  +--rw ext-community-set* [ext-community-set-name]
     |  |  |     +--rw ext-community-set-name    -> ../config/ext-community-set-name
     |  |  |     +--rw config
     |  |  |     |  +--rw ext-community-set-name?   string
     |  |  |     |  +--rw ext-community-member*     union
     |  |  |     |  +--rw match-set-options?     oc-pol-types:match-set-options-type
     |  .............
     +--rw policy-definitions
        +--rw policy-definition* [name]
           +--rw name          -> ../config/name
           +--rw config
           |  +--rw name?   string
           +--rw statements
           |   +--rw statement* [name]
           |      +--rw name          -> ../config/name
           |      +--rw config
           |      |  +--rw name?   string
           |      +--rw conditions
           |      |  +--rw config
           |      |  |  +--rw install-protocol-eq?   identityref
           |      |  +--rw bgp-conditions  (augment from openconfig-bgp)
           |      |  |  +--rw config
           |      |  |  |  +--rw ext-community-set?   -> /…/bgp-defined-sets/
           |      |  ......             ext-community-sets/ext-community-set/
           |      |                     ext-community-set-name
           |      +--rw actions
           |         +--rw config
           |         |  +--rw policy-result?   policy-result-type
           ...       ...
```

Examples (import & export BGP routing rules):

```
ext-community-set-name : my_net_inst_rt_import
ext-community-member   : route-target:65000:333
match-set-options      : oc-pol-types:ANY
policy-definition.name : my_net_inst_import
statement.name         : 3
install-protocol-eq    : oc-pol-types:DIRECTLY_CONNECTED
policy-result          : ACCEPT_ROUTE
```

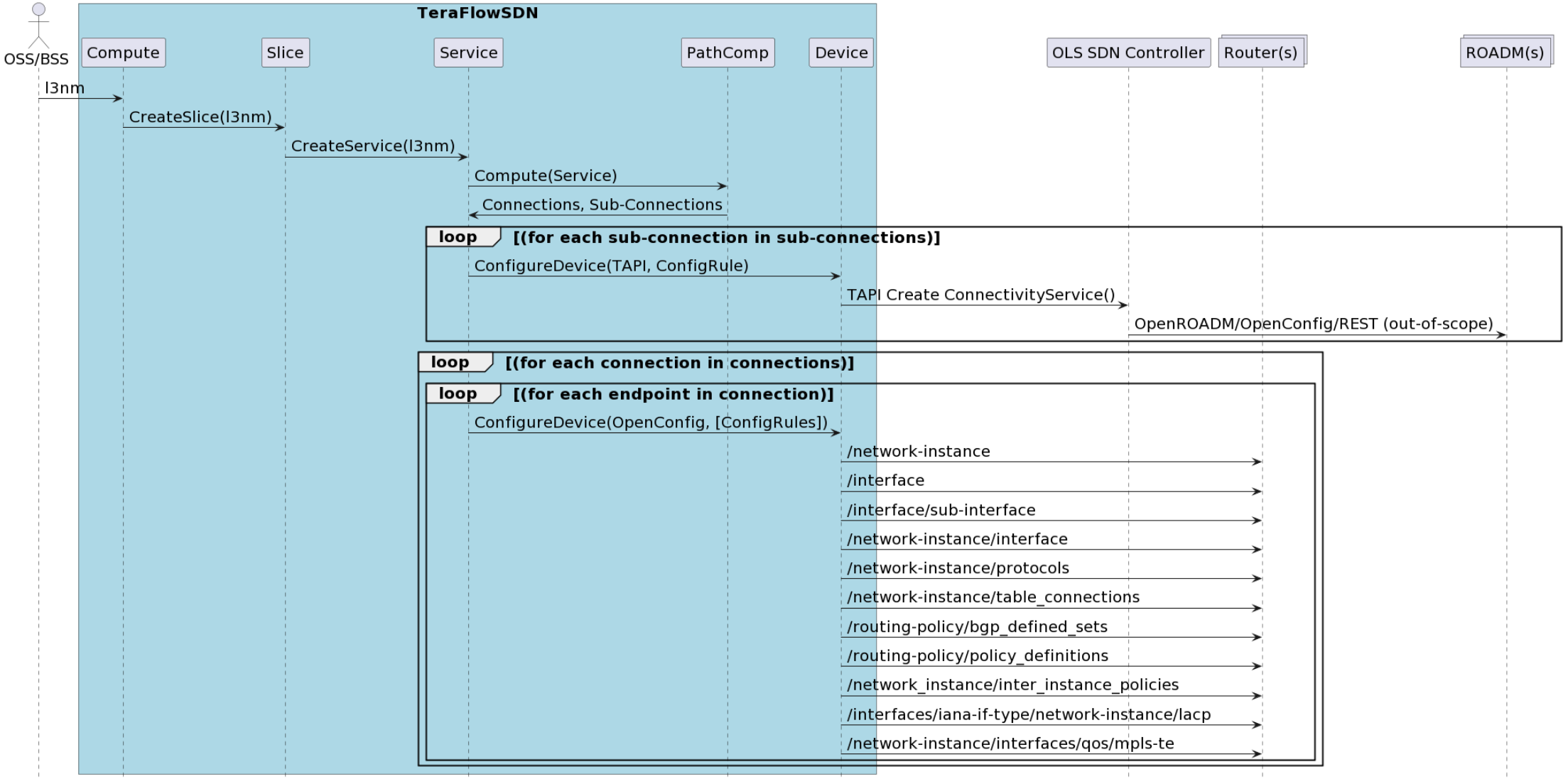
```
ext-community-set-name : my_net_inst_rt_export
ext-community-member   : route-target:65000:333
match-set-options      : oc-pol-types:ANY
policy-definition.name : my_net_inst_export
statement.name         : 3
install-protocol-eq    : oc-pol-types:DIRECTLY_CONNECTED
policy-result          : ACCEPT_ROUTE
```

OpenConfig L3 data models – Network Instance

```
module: openconfig-network-instance
  +--rw network-instances
     +--rw network-instance* [name]
        +--rw name                          -> ../config/name
        +--rw config
        |  +--rw name?                          string
        |  +--rw type?                          identityref
        |  +--rw enabled?                       boolean
        |  +--rw router-id?                     yang:dotted-quad
        |  +--rw route-distinguisher?           oc-ni-types:route-distinguisher
        |  ...
        +--ro state ...
        +--rw encapsulation
        |  +--rw config
        |  |  +--rw encapsulation-type?      identityref
        |  |  +--rw label-allocation-mode?   identityref
        |  ......
        +--rw inter-instance-policies
        |  +--rw apply-policy
        |  |  +--rw config
        |  |  |  +--rw import-policy*           -> …/policy-definition/name
        |  |  |  +--rw export-policy*           -> …/policy-definition/name
        |  .........
        +--rw table-connections
        |  +--rw table-connection* [src-protocol dst-protocol address-family]
        |     +--rw src-protocol      -> ../config/src-protocol
        |     +--rw dst-protocol      -> ../config/dst-protocol
        |     +--rw address-family    -> ../config/address-family
        |     +--rw config
        |     |  +--rw src-protocol?     -> …/table/config/protocol
        |     |  +--rw address-family?   -> …/table[…]/config/address-family
        |     |  +--rw dst-protocol?     -> …/table/config/protocol
        |     ......
…
```

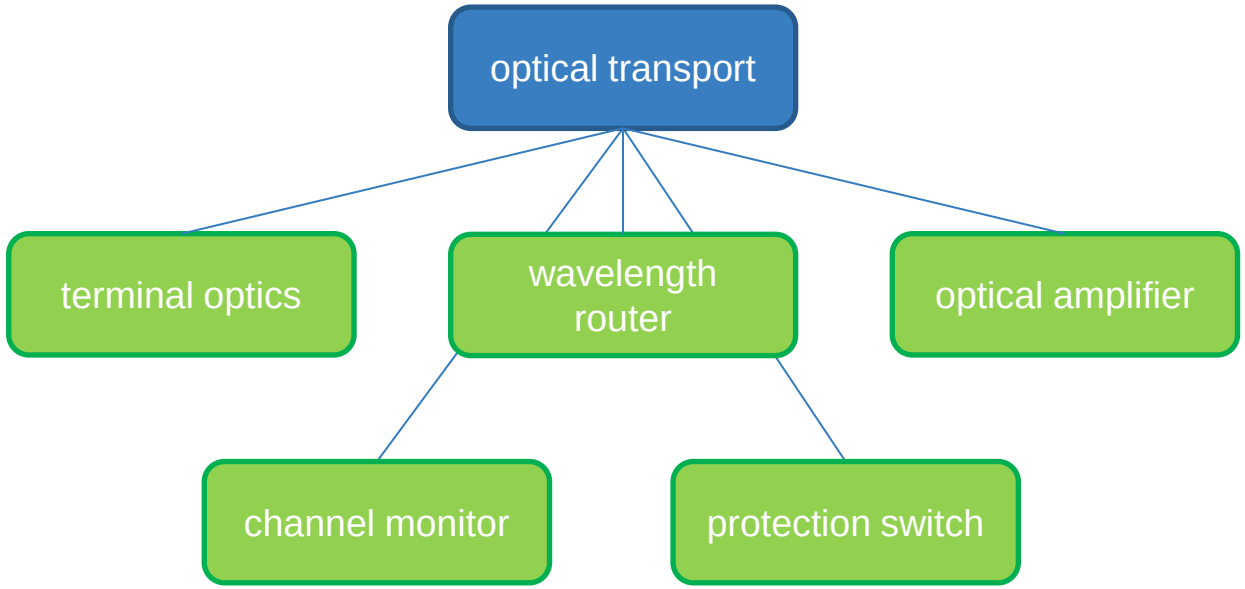
```
  +--rw interfaces
  |  +--rw interface* [id]
  |     +--rw id        -> ../config/id
  |     +--rw config
  |     |  +--rw id?            string
  |     |  +--rw interface?     -> /interfaces/interface/name
  |     |  +--rw subinterface?  -> /interfaces/interface[…]/
  |     ......                     subinterfaces/subinterface/index
  +--rw tables
  |  +--rw table* [protocol address-family]
  |     +--rw protocol          -> ../config/protocol
  |     +--rw address-family    -> ../config/address-family
  |     +--rw config
  |     |  +--rw protocol?         -> …/protocol/config/identifier
  |     |  +--rw address-family?   identityref
  |     +--ro state ...
  +--rw protocols
     +--rw protocol* [identifier name]
        +--rw identifier          -> ../config/identifier
        +--rw name                -> ../config/name
        +--rw config
        |  +--rw identifier?       identityref
        |  +--rw name?             string
        |  ...
        +--rw static-routes ...
        +--rw bgp
        |  +--rw global
        |  |  +--rw config
        |  |  |  +--rw as           oc-inet:as-number
        |  |  |  +--rw router-id?   oc-yang:dotted-quad
        |  ......
        +--rw ospfv2 ...
        +--rw isis ...
        ...
```

OpenConfig workflow MS2.2



Optical-Transport

Provides a configuration and state model for terminal optical devices within a DWDM system, including both client- and line-side parameters.



openconfig-terminal-device.yang

Terminal optics device model for managing the terminal systems (client and line side) Elements of the model:

- physical port: corresponds to a physical, pluggable client port on the terminal device. Examples includes 10G, 40G, 100G and 400G/1T in the future.
- physical channel: a physical lane or channel in the physical client port. Each physical client port has 1 or more channels. An example is 100GBASE-LR4 client physical port having 4x25G channels.
- **logical channel**: a logical grouping of logical grooming elements that may be assigned to subsequent grooming stages for multiplexing / de-multiplexing, or to an optical channel for line side transmission. The logical channels can represent, for example, an ODU/OTU logical packing of the client data onto the line side.
- **optical channel**: corresponds to an optical carrier and is assigned a wavelength/frequency. Optical channels have PMs such as power, BER, and operational mode.

Directionality: To maintain simplicity in the model, the configuration is described from client-to-line direction. The assumption is that equivalent reverse configuration is implicit, resulting in the same line-to-client configuration.

Vendor-supported operational modes. Example of possible info:

- Symbol rate (32G, 40G, 43G, 64G, etc.), Modulation (QPSK, 8-QAM, 16-QAM, etc.)
- Differential encoding (on, off/pilot symbol, etc), FEC mode (SD, HD, % OH)
- State of polarization tracking mode (default, med. high-speed, etc.), Pulse shaping (RRC, RC, roll-off factor)

openconfig-terminal-device.yang (I)

```
module: openconfig-terminal-device
    +--rw terminal-device
       +--rw config
       +--ro state
       +--rw logical-channels
       |  +--rw channel* [index]
       |     +--rw index                          -> ../config/index
       |     +--rw config
       |     |  +--rw index?                 uint32
       |     |  +--rw description?           string
       |     |  +--rw admin-state?           oc-opt-types:admin-state-type
       |     |  +--rw rate-class?            identityref
       |     |  +--rw trib-protocol?         identityref
       |     |  +--rw logical-channel-type?  identityref
       |     |  +--rw loopback-mode?         oc-opt-types:loopback-mode-type
       |     |  +--rw test-signal?           boolean
       |     +--ro state (idem)
```

```
       |     +--rw otn
       |     |  +--rw config
       |     |  +--ro state
       |     |     +--ro tti-msg-transmit?           string
       |     |     +--ro tti-msg-expected?           string
       |     |     +--ro tti-msg-auto?               boolean
       |     |     +--ro tti-msg-recv?               string
       |     |     +--ro rdi-msg?                    string
       |     |     +--ro errored-seconds?            yang:counter64
       |     |     +--ro severely-errored-seconds?   yang:counter64
       |     |     +--ro unavailable-seconds?        yang:counter64
       |     |     +--ro code-violations?            yang:counter64
       |     |     +--ro errored-blocks?             yang:counter64
       |     |     +--ro fec-uncorrectable-blocks?   yang:counter64
       |     |     +--ro fec-uncorrectable-words?    yang:counter64
       |     |     +--ro fec-corrected-bytes?        yang:counter64
       |     |     +--ro fec-corrected-bits?         yang:counter64
       |     |     +--ro background-block-errors?    yang:counter64
       |     |     +--ro pre-fec-ber
       |     |     |  +--ro instant?    decimal64
       |     |     |  +--ro avg?        decimal64
       |     |     |  +--ro min?        decimal64
       |     |     |  +--ro max?        decimal64
       |     |     |  +--ro interval?   oc-types:stat-interval
       |     |     |  +--ro min-time?   oc-types:timeticks64
       |     |     |  +--ro max-time?   oc-types:timeticks64
       |     |     +--ro post-fec-ber (idem pre-fec-ber)
       |     |     +--ro q-value (idem pre-fec-ber)
       |     |     +--ro esnr (idem pre-fec-ber)
```

openconfig-terminal-device.yang (II)

```
module: openconfig-terminal-device
    +--rw terminal-device
       +--rw config
       +--ro state
       +--rw logical-channels
       |  +--rw channel* [index]
       |     +--rw ethernet
       |     |  +--rw config
       |     |  +--ro state
       |     |     +--ro in-mac-control-frames?            oc-yang:counter64
       |     |     +--ro in-mac-pause-frames?              oc-yang:counter64
       |     |     +--ro in-oversize-frames?               oc-yang:counter64
       |     |     +--ro in-undersize-frames?              oc-yang:counter64
       |     |     +--ro in-jabber-frames?                 oc-yang:counter64
       |     |     +--ro in-fragment-frames?               oc-yang:counter64
       |     |     +--ro in-8021q-frames?                  oc-yang:counter64
       |     |     +--ro in-crc-errors?                    oc-yang:counter64
       |     |     +--ro in-block-errors?                  oc-yang:counter64
       |     |     +--ro out-mac-control-frames?           oc-yang:counter64
       |     |     +--ro out-mac-pause-frames?             oc-yang:counter64
       |     |     +--ro out-8021q-frames?                 oc-yang:counter64
       |     |     +--ro in-pcs-bip-errors?                oc-yang:counter64
       |     |     +--ro in-pcs-errored-seconds?           oc-yang:counter64
       |     |     +--ro in-pcs-severely-errored-seconds?  oc-yang:counter64
       |     |     +--ro in-pcs-unavailable-seconds?       oc-yang:counter64
       |     |     +--ro out-pcs-bip-errors?               oc-yang:counter64
       |     |     +--ro out-crc-errors?                   oc-yang:counter64
       |     |     +--ro out-block-errors?                 oc-yang:counter64
       |     +--rw ingress
       |     |  +--rw config
       |     |  |  +--rw transceiver?        -> /oc-platform:components/component/name
       |     |  |  +--rw physical-channel*   -> /oc-platform:components/component/
       |     |  |                               oc-transceiver:transceiver/
       |     |  |                               physical-channels/channel/index
       |     |  +--ro state
       |     +--rw logical-channel-assignments
       |        +--rw assignment* [index]
       |           +--rw index     -> ../config/index
       |           +--rw config
       |           |  +--rw index?             uint32
       |           |  +--rw description?       string
       |           |  +--rw assignment-type?   enumeration
       |           |  +--rw logical-channel?   -> /terminal-device/logical-channels/
       |           |                              channel/index
       |           |  +--rw optical-channel?   -> /oc-platform:components/component/name
       |           |  +--rw allocation?        decimal64
       |           +--ro state (idem)
       +--rw operational-modes
          +--ro mode* [mode-id]
             +--ro mode-id    -> ../state/mode-id
             +--ro config
             +--ro state
                +--ro mode-id?       uint16
                +--ro description?   string
                +--ro vendor-id?     string
```

openconfig-terminal-device.yang (III)

```
augment /oc-platform:components/oc-platform:component:
   +--rw optical-channel
      +--rw config
      |  +--rw frequency?             oc-opt-types:frequency-type
      |  +--rw target-output-power?   decimal64
      |  +--rw operational-mode?      uint16
      |  +--rw line-port?             -> /oc-platform:components/component/name
      +--ro state
         +--ro frequency?                                  oc-opt-types:frequency-type
         +--ro target-output-power?                        decimal64
         +--ro operational-mode?                           uint16
         +--ro line-port?                                  -> /oc-platform:components/component/name
         +--ro group-id?                                   uint32
         +--ro output-power
         |  +--ro instant?    decimal64
         |  +--ro avg?        decimal64
         |  +--ro min?        decimal64
         |  +--ro max?        decimal64
         |  +--ro interval?   oc-types:stat-interval
         |  +--ro min-time?   oc-types:timeticks64
         |  +--ro max-time?   oc-types:timeticks64
         +--ro input-power
         +--ro laser-bias-current
         +--ro chromatic-dispersion
         +--ro polarization-mode-dispersion
         +--ro second-order-polarization-mode-dispersion
         +--ro polarization-dependent-loss
```

Pre-built Netconf/OpenConfig server

A basic Netconf/OpenConfig server supporting basic OpenConfig data model entries can be found at

- ~/tfs-ctrl/hackfest/netconf-oc
- If you're curious, check build steps in file "build-instructions.txt"

Start the server:

```
$ cd ~/tfs-ctrl/hackfest/netconf-oc
$ python3 server_openconfig.py 8300
```

Remember to set the listen port!

Exercise: Onboard Devices using TeraFlowSDN

Connect TFS to Netconf/OpenConfig devices using descriptor files

```
{
    "devices": [
        {
            "device_id": {"device_uuid": {"uuid": "R1"}},
            "device_type": "packet-router",
            "device_config": {"config_rules": [
                {"action": 1, "custom": {"resource_key": "_connect/address", "resource_value": "10.0.2.10"}},
                {"action": 1, "custom": {"resource_key": "_connect/port", "resource_value": "8300"}},
                {"action": 1, "custom": {"resource_key": "_connect/settings", "resource_value": {
                    "username": "admin", "password": "admin", "force_running": true, "hostkey_verify": false,
                    "look_for_keys": false, "allow_agent": true, "delete_rule": false,
                    "device_params"  : {"name": "default"}, "manager_params" : {"timeout": 15}
                }}}
            ]},
            "device_operational_status": 1,
            "device_drivers": [1],
            "device_endpoints": []
        }
    ]
}
```

Check “src/common/DeviceTypes.py”

Set IP Address of your VM & port of Netconf server

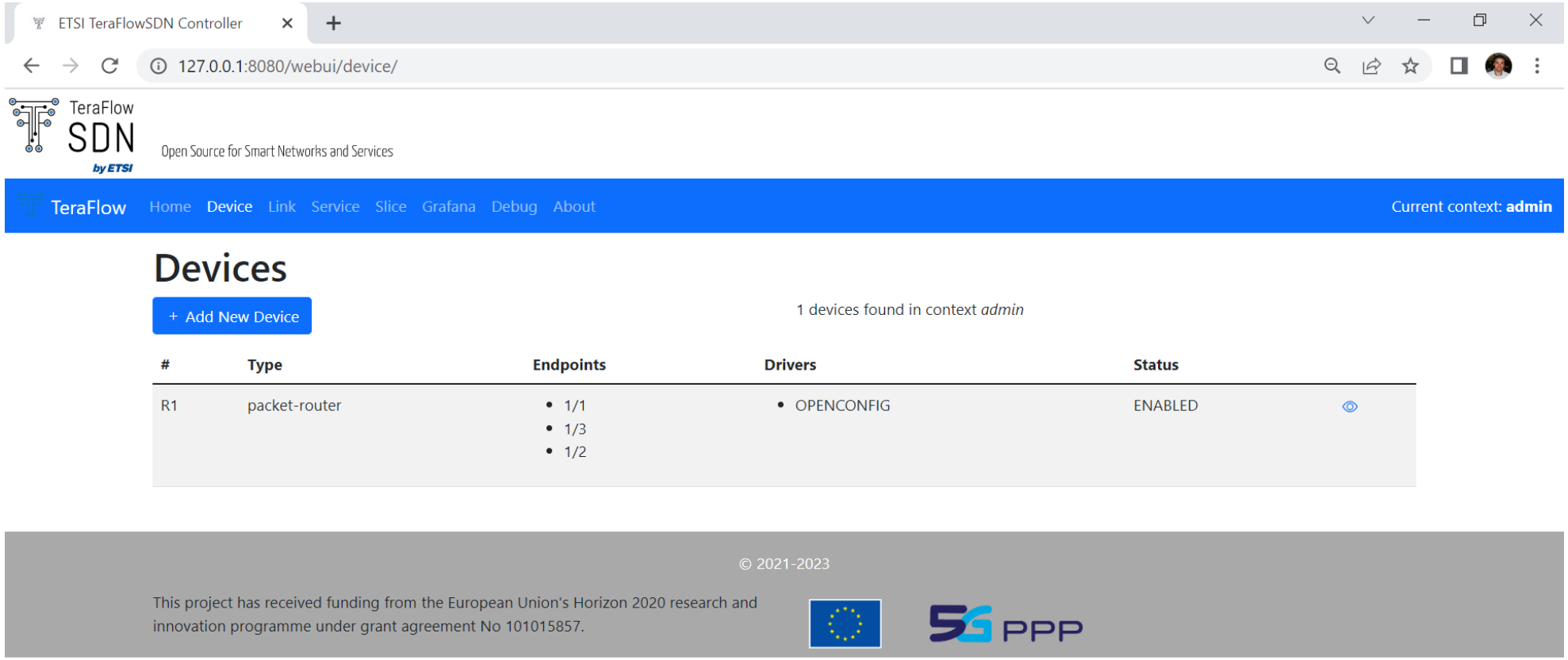
By default, DISABLED, Will be activated by Automation.
Check “proto/context.proto” “DeviceOperationalStatusEnum”

Use OpenConfig driver for this device descriptor.
Check “proto/context.proto” “DeviceDriverEnum”

Automatically discovered from TAPI server by Device component.

Exercise: Onboard Devices using TeraFlowSDN

You should see the device in WebUI with the endpoints discovered



Exercise: Onboard Devices using TeraFlowSDN

Check the logs of the TeraFlowSDN components:

- Example: Device component

```
$ cd ~/tfs-ctrl
$ source my_deploy.sh
$ scripts/show_logs_device.sh
```

Upload links and services

Similarly, Links and Service requests can be uploaded using JSON-based descriptors.

- Link Template:

```
{
    "links": [
        {
            "link_id": {"link_uuid": {"uuid": "…"}},
            "link_endpoint_ids": [
                {"device_id": {"device_uuid": {"uuid": "…"}}, "endpoint_uuid": {"uuid": "…"}},
                {"device_id": {"device_uuid": {"uuid": "…"}}, "endpoint_uuid": {"uuid": "…"}}
            ]
        }
    ]
}
```

Upload links and services

Similarly, Links and Service requests can be uploaded using JSON-based descriptors.

- Service Template:

```
{
    "services": [
        {
            "service_id": {
                "context_id": {"context_uuid": {"uuid": "…"}},
                "service_uuid": {"uuid": "…"}
            },
            "service_type": 1,
            "service_status": {"service_status": 1},
            "service_endpoint_ids": [
                {"device_id": {"device_uuid": {"uuid": "…"}}, "endpoint_uuid": {"uuid": "…"}},
                {"device_id": {"device_uuid": {"uuid": "…"}}, "endpoint_uuid": {"uuid": "…"}}
            ],
            "service_constraints": [ … ],
            "service_config": {"config_rules": [ … ]}
        }
    ]
}
```

Exercise: Establishing L3VPN service using TeraFlowSDN

Start 4 Netconf/OpenConfig servers

Start 1 TAPI OLS controller

Create & load descriptor files:

- all the devices
- links between endpoints as listed in

Issue service request

- service-l3vpn.json

Delete service through WebUI

Time: 20 min

Links:

Device	Endpoint	Device	EndPoint
R1	1/1	OLS	node-1-port-15-input
OLS	node-1-port-15-output	R1	1/1
R2	1/1	OLS	node-2-port-15-input
OLS	node-2-port-15-output	R2	1/1
R3	1/1	OLS	node-3-port-15-input
OLS	node-3-port-15-output	R3	1/1
R4	1/1	OLS	node-4-port-15-input
OLS	node-4-port-15-output	R4	1/1

Exercise: Establishing L3VPN service using TeraFlowSDN

Solution:

- Command to deploy servers

```
# (in a new terminal) Start OLS TAPI server:
cd ~/tfs-ctrl/hackfest/tapi/server
python3 -m tapi_server 8080 database/mini-ols-context.json

# (in a new terminal) Start the Netconf/OpenConfig server for R1
cd ~/tfs-ctrl/hackfest/netconf-oc
python3 server_openconfig.py 8301

# (in a new terminal) Start the Netconf/OpenConfig server for R2
cd ~/tfs-ctrl/hackfest/netconf-oc
python3 server_openconfig.py 8302

# (in a new terminal) Start the Netconf/OpenConfig server for R3
cd ~/tfs-ctrl/hackfest/netconf-oc
python3 server_openconfig.py 8303

# (in a new terminal) Start the Netconf/OpenConfig server for R4
cd ~/tfs-ctrl/hackfest/netconf-oc
python3 server_openconfig.py 8304
```

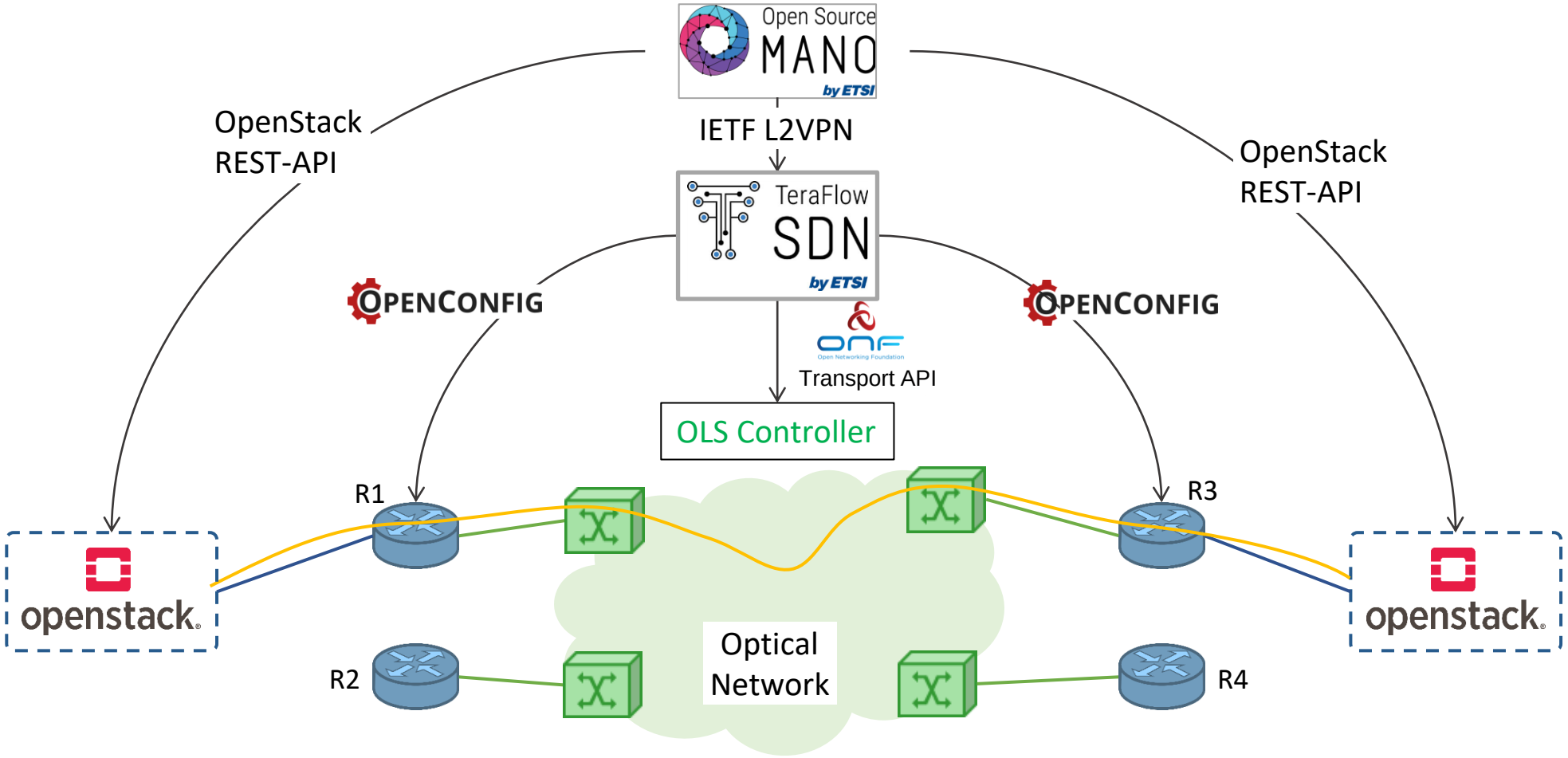
- Descriptor files can be found in

  ~/tfs-ctrl/hackfest/tfs-descriptors

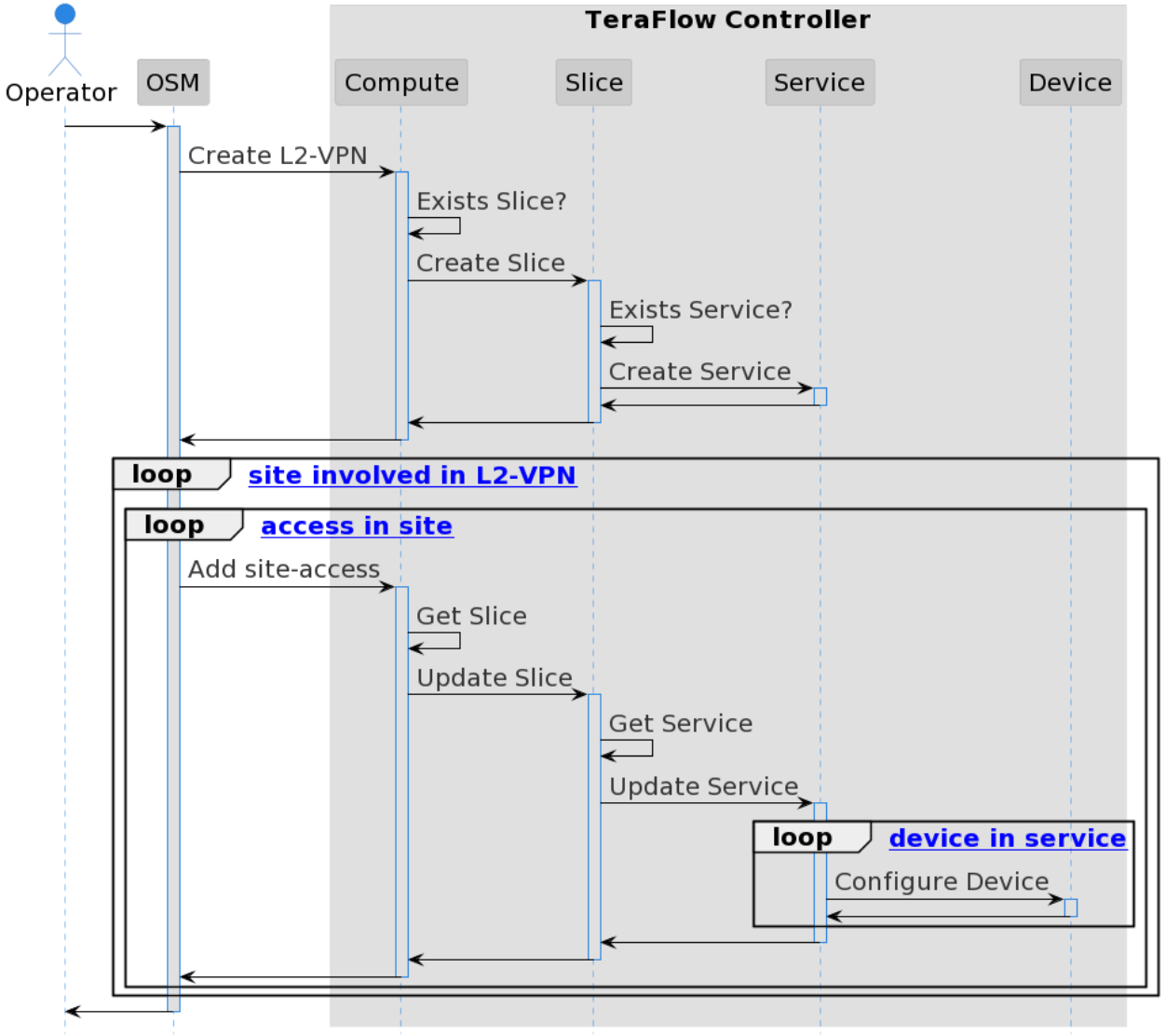
  - context-topology.json
  - device-all.json
  - links.json
  - service-l3vpn.json

Exercise: Set-up L2VPN using emulated IP devices and TAPI OLS

This exercise emulates the requests that ETSI OpenSourceMANO issues to underlying WAN Infrastructure Managers to how OSM



Exercise: Set-up L2VPN using emulated IP devices and TAPI OLS



IETF L2VPN request from OSM (add site access)

```
Hypertext Transfer Protocol
JavaScript Object Notation: application/json
▼ "ietf-l2vpn-svc:site-network-access":
  ▼ 0:
    ▼ "connection":
        "encapsulation-type": "dot1q-vlan-tagged"
      ▼ "tagged-interface":
        ▼ "dot1q-vlan-tagged":
            "cvlan-id": 300
    ▼ "vpn-attachment":
        "vpn-id": "b93d90c0-6b35-4bf5-8593-ac78f09f16a4"
        "site-role": "any-to-any-role"
      "network-access-id": "0a00b6a0-8911-4abd-9b89-f1a318d95456"
    ▼ "bearer":
        "bearer-reference": "CE1-PE1"
```

Exercise: Set-up L2VPN slice using emulated IP devices and TAPI OLS

Assume same topology as in previous exercise, but do not load the L3 service descriptor file.

Open Wireshark to see the request messages in detail.

Run Mock OSM to create/inspect/delete an L2VPN slice

- Tip: try “help” command.

```
$ cd ~/tfs-ctrl/hackfest/
$ python -m mock_osm
```

Time: 10 min

Exercise: Set-up L2VPN slice using emulated IP devices and TAPI OLS

TeraFlow SDN

Solution:

```
$ cd ~/tfs-ctrl/hackfest/
$ python -m mock_osm
Welcome to the MockOSM shell.
Type help or ? to list commands.

(mock-osm) create
Service b8c99e2c-39d8-424d-9833-554634269555 created
# Service should be created in TFS. Check "Slice" and "Service" pages on WebUI.
# Check configuration rules in "Devices"

(mock-osm) status
response.status_code=200
Status of Service b8c99e2c-39d8-424d-9833-554634269555 is {'sdn_status': 'ACTIVE'}

(mock-osm) delete
Service b8c99e2c-39d8-424d-9833-554634269555 deleted
# Service should be removed from TFS. Check "Slice" and "Service" pages on WebUI.
# Check configuration rules in "Devices"

(mock-osm) exit
Bye!
```

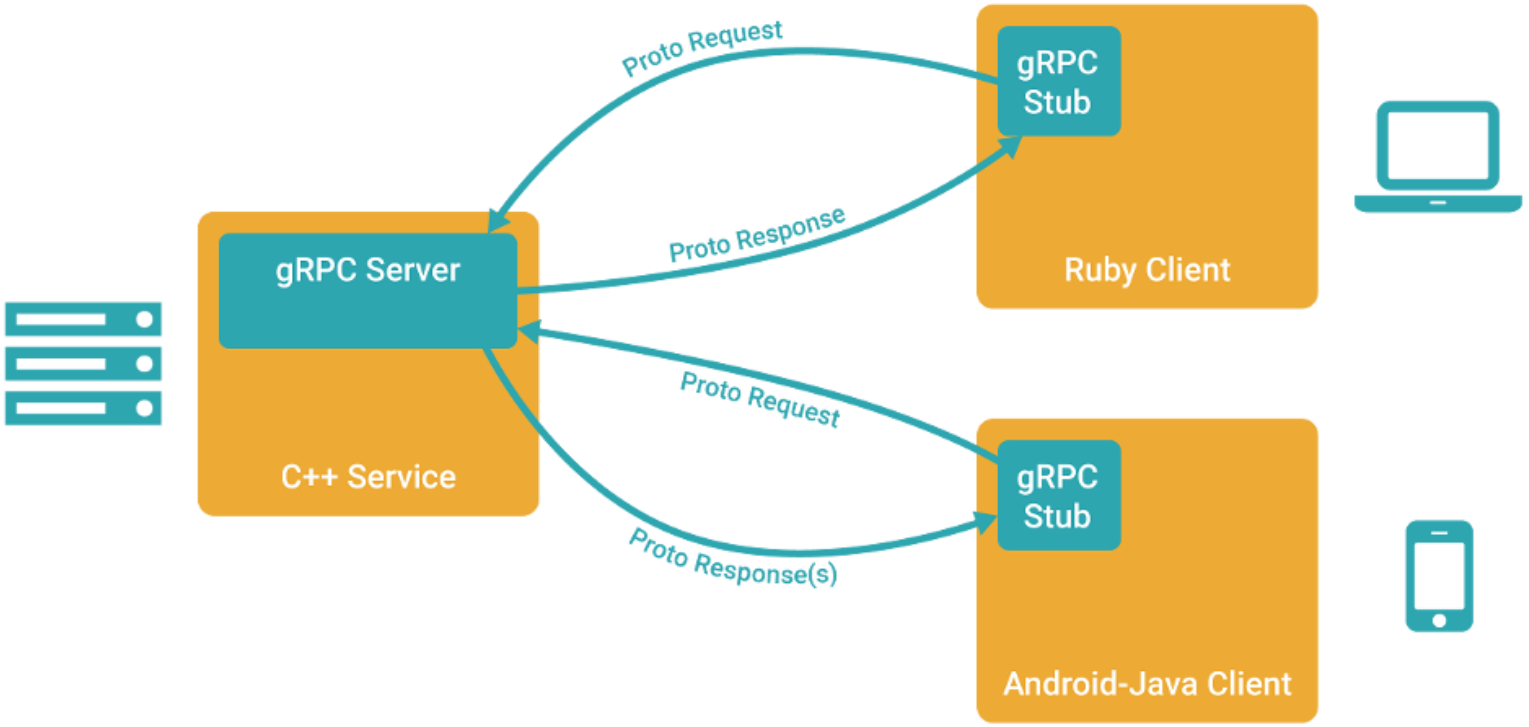
TeraFlow SDN

Monitoring

What is gRPC

- gRPC stands for gRPC Remote Procedure Calls
- A high performance, general purpose, feature-rich RPC framework
- Part of Cloud Native Computing Foundation
- HTTP/2 and mobile first
- Open sourced version of Stubby RPC used in Google

gRPC architecture



Source:
https://grpc.io/

Protocol Buffers

TeraFlow SDN

Interface Definition Language (IDL)

- Describe once and generate interfaces for any language.

Data Model

- Structure of the request and response.

Describes Wire format

- Binary format for network transmission.
- No more parsing text!
- Compression
- Streaming

Compilation:

```
$ protoc -I=. --python_out=out_dir/ example.proto
```

```
syntax = "proto3";
option java_multiple_files = true;
option java_package = "com.grpc.search";
option java_outer_classname = "SearchProto";
option objc_class_prefix = "GGL";
package search;

service Google {
  // Search returns a Search Engine result for the query.
  rpc Search(Request) returns (Result) {}
}
message Request {
  string query = 1;
}
message Result {
  string title = 1;
  string url = 2;
  string snippet = 3;
}
```

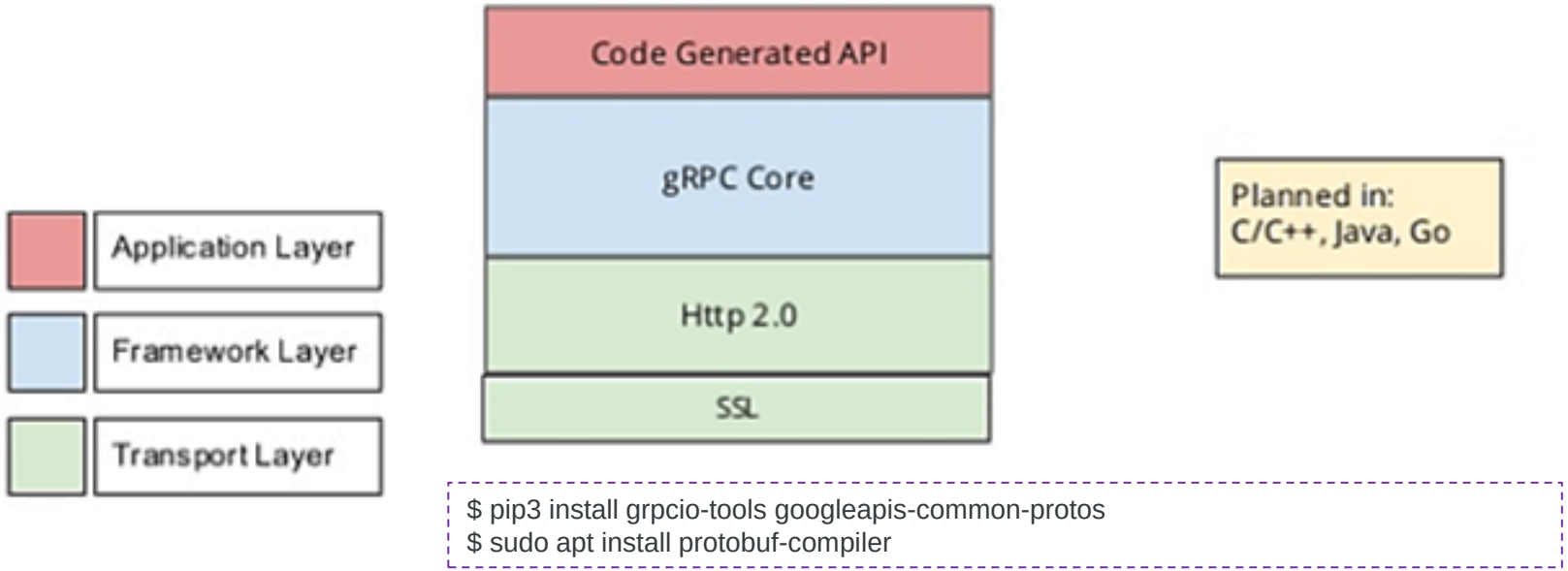
gRPC Main Use Cases and architecture

Efficiently connecting polyglot services in microservices style architecture

Connecting mobile devices, browser clients to backend services

Generating efficient client libraries

Low latency, highly scalable, distributed systems.




```
$ pip3 install grpcio-tools googleapis-common-protos
$ sudo apt install protobuf-compiler
```

Usage of protobufs

TeraFlow SDN

Translate connection.yang to protobuf

Create a script that writes new connections to a file

Create a script that lists all stored connections from a file

You can use the following tutorial

https://developers.google.com/protocol-buffers/docs/pythontutorial

Warning: Be “careful” with hyphens!

connection.proto

```
//Example of connection
syntax = "proto3";
package connection;

message Connection {
  string connectionId = 1;
  string sourceNode = 2;
  string targetNode = 3;
  string sourcePort = 4;
  string targetPort = 5;
  uint32 bandwidth = 6;

  enum LayerProtocolName {
    ETH = 0;
    OPTICAL = 1;
  }

  LayerProtocolName layerProtocolName = 7;

}

message ConnectionList {
  repeated Connection connection = 1;
}
```

```
$ cd ~/tfs-ctrl/hackfest/grpc
$ python -m grpc_tools.protoc -I=. --python_out=connection/ connection.proto
```

Create Connection

```
#! /usr/bin/env python3
import connection_pb2
import sys

def PromptForConnection(connection):
  connection.connectionId = raw_input("Enter connectionID:
")
  connection.sourceNode = raw_input("Enter sourceNode: ")
  connection.targetNode = raw_input("Enter targetNode: ")
  connection.sourcePort = raw_input("Enter sourcePort: ")
  connection.targetPort = raw_input("Enter targetPort: ")
  connection.bandwidth = int( raw_input("Enter bandwidth: ")
)
  type = raw_input("Is this a eth or optical connection? ")
  if type == "eth":
    connection.layerProtocolName =
connection_pb2.Connection.ETH
  elif type == "optical":
    connection.layerProtocolName =
connection_pb2.Connection.OPTICAL
  else:
    print("Unknown layerProtocolName type; leaving as
default value.")

…
```

```
…
if __name__ == '__main__':
  if len(sys.argv) != 2:
    print("Usage:", sys.argv[0], "CONNECTION_FILE")
    sys.exit(-1)

  connectionList = connection_pb2.ConnectionList()

  # Read the existing address book.
  try:
    with open(sys.argv[1], "rb") as f:
      connectionList.ParseFromString(f.read())
  except IOError:
    print(sys.argv[1] + ": File not found.  Creating a new file.")

  # Add an address.
  PromptForConnection(connectionList.connection.add())

  # Write the new address book back to disk.
  with open(sys.argv[1], "wb") as f:
    f.write(connectionList.SerializeToString())
```

```
$ cd ~/tfs-ctrl/hackfest/grpc/connection
$ python3 create.py connection.txt
```

List Connection

```
#! /usr/bin/env python3
from __future__ import print_function
import connection_pb2
import sys


# Iterates though all connections in the ConnectionList and
prints info about them.
def ListConnections(connectionList):
  for connection in connectionList.connection:
    print("connectionID:", connection.connectionId)
    print("  sourceNode:", connection.sourceNode)
    print("  targetNode:", connection.targetNode)
    print("  sourcePort:", connection.sourcePort)
    print("  targetPort:", connection.targetPort)
    print("  bandwidth:", connection.bandwidth)
    if connection.layerProtocolName ==
connection_pb2.Connection.ETH:
      print("  layerProtocolName:ETH")
    elif connection.layerProtocolName ==
connection_pb2.Connection.OPTICAL:
      print("  layerProtocolName:OPTICAL")
…
```

```
…
if __name__ == '__main__':
  if len(sys.argv) != 2:
    print("Usage:", sys.argv[0], "CONNECTION_FILE")
    sys.exit(-1)

  connectionList = connection_pb2.ConnectionList()

  # Read the existing address book.
  with open(sys.argv[1], "rb") as f:
    connectionList.ParseFromString(f.read())

  ListConnections(connectionList)
```

```
$ cd ~/tfs-ctrl/hackfest/grpc/connection
$ python3 list.py connection.txt
```

Create a gRPC client/server

TeraFlow SDN

Example tutorial

https://grpc.io/docs/tutorials/basic/python.html

Extend connection.proto to connectionService.proto with following service:

```
service ConnectionService {
  rpc CreateConnection (Connection) returns (google.protobuf.Empty) {}
  rpc ListConnection (google.protobuf.Empty) returns (ConnectionList) {}
}
```

```
$ cd ~/tfs-ctrl/hackfest/grpc
$ python -m grpc_tools.protoc -I=. --python_out=connectionService/ --
grpc_python_out=connectionService/ connectionService.proto
```

connectionService_server.py

TeraFlow SDN

```
from concurrent import futures
import time
import logging
import grpc

import connectionService_pb2
import connectionService_pb2_grpc
from google.protobuf import empty_pb2 as google_dot_protobuf_dot_empty__pb2

_ONE_DAY_IN_SECONDS = 60 * 60 * 24

class connectionService(connectionService_pb2_grpc.ConnectionServiceServicer):
    def __init__(self):
        self.connectionList = connectionService_pb2.ConnectionList()

    def CreateConnection(self, request, context):
        logging.debug("Received Connection " + request.connectionId)
        self.connectionList.connection.extend([request])
        return google_dot_protobuf_dot_empty__pb2.Empty()

    def ListConnection(self, request, context):
        logging.debug("List Connections")
        return self.connectionList

def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    connectionService_pb2_grpc.add_ConnectionServiceServicer_to_server(connectionService(), server)
    server.add_insecure_port('[::]:50051')
    logging.debug("Starting server")
    server.start()
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        server.stop(0)

if __name__ == '__main__':
    logging.basicConfig(level=logging.DEBUG)
    serve()
```

connectionService_client.py

TeraFlow SDN

```
from __future__ import print_function
import grpc

import connectionService_pb2
import connectionService_pb2_grpc
from google.protobuf import empty_pb2 as google_dot_protobuf_dot_empty__pb2

def createConnection():
    with grpc.insecure_channel('localhost:50051') as channel:
        connection=connectionService_pb2.Connection()
        connection.connectionId = raw_input("Enter connectionID: ")
        connection.sourceNode = raw_input("Enter sourceNode: ")
        connection.targetNode = raw_input("Enter targetNode: ")
        connection.sourcePort = raw_input("Enter sourcePort: ")
        connection.targetPort = raw_input("Enter targetPort: ")
        connection.bandwidth = int( raw_input("Enter bandwidth: ") )
        stub = connectionService_pb2_grpc.ConnectionServiceStub(channel)
        response = stub.CreateConnection(connection)
    print("ConnectionService client received: " + str(response) )

def listConnection():
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = connectionService_pb2_grpc.ConnectionServiceStub(channel)
        response = stub.ListConnection(google_dot_protobuf_dot_empty__pb2.Empty())
    print("ConnectionService client received: " + str(response) )

if __name__ == '__main__':
    createConnection()
    listConnection()
```

Run example

TeraFlow SDN

Run Server

```
$ cd ~/tfs-ctrl/hackfest/grpc/connectionService
$ python3 connectionService_server.py
```

Run client

```
$ cd ~/tfs-ctrl/hackfest/grpc/connectionService
$ python3 connectionService_client.py
```

Exercise: gRPC streams

TeraFlow SDN

Create a new function in our Service to return the BER of a connection every 5 seconds.

Use:

```
rpc GetBer(Connection) returns (stream Ber) {}
```

```
$ cd ~/tfs-ctrl/hackfest/grpc
$ python -m grpc_tools.protoc -I=. --python_out=connectionServiceWithNotif/ --
grpc_python_out=connectionServiceWithNotif/ connectionServiceWithNotif.proto
```

Time: 10min

Solution

TeraFlow SDN

Server

```
def GetBer (self, request, context):
        logging.debug("Get Ber")
        while True:
            time.sleep(5)
            ber=connectionServiceWithNotif_pb2.Ber(value=10)
            yield ber
```

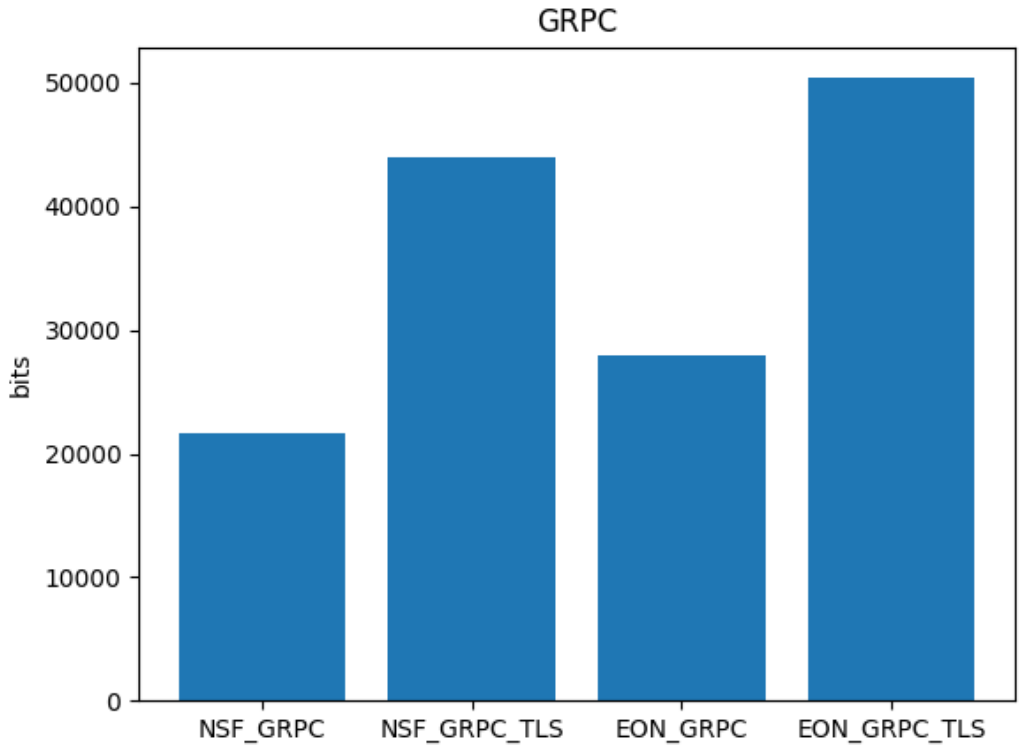
```
RUN SERVER
$ cd ~/tfs-ctrl/hackfest/grpc /connectionServiceWithNotif
$ python3 connectionServiceWithNotif_server.py
```

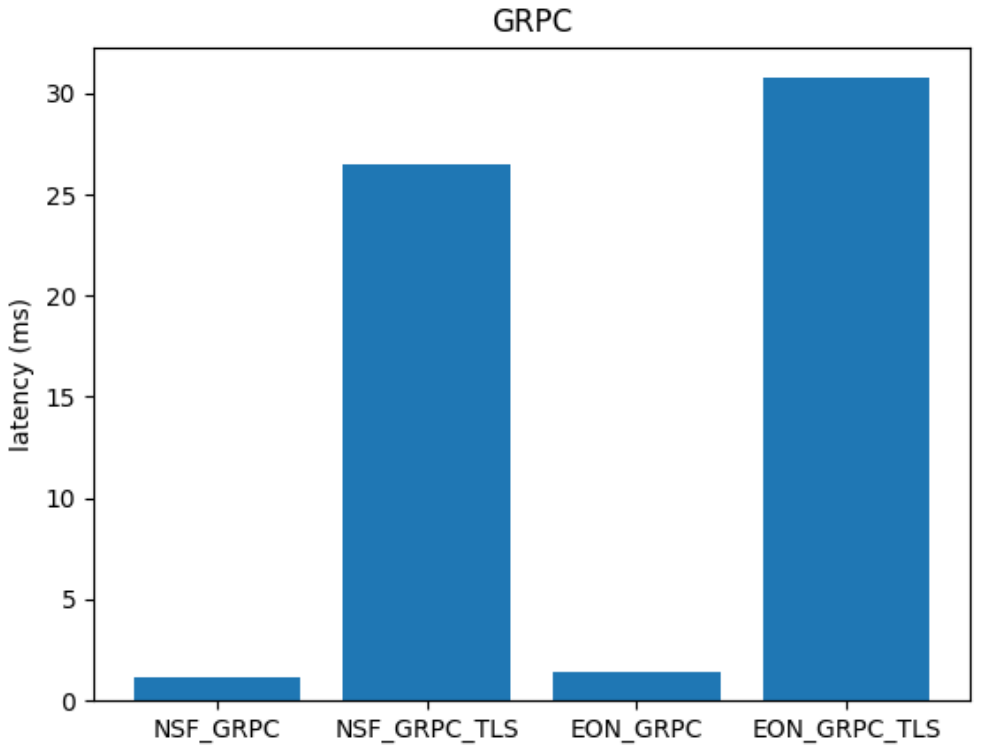
Client

```
def getBer(stub):
    responses = stub.GetBer(connectionServiceWithNotif_pb2.Connection(connectionId="conn1"))
    for response in responses:
        print("Received Ber %s" % (response.value) )
```

```
RUN CLIENT (in another window)
$ cd ~/tfs-ctrl/hackfest/grpc/connectionServiceWithNotif
$ python3 connectionServiceWithNotif_client.py
```

Example gRPC results



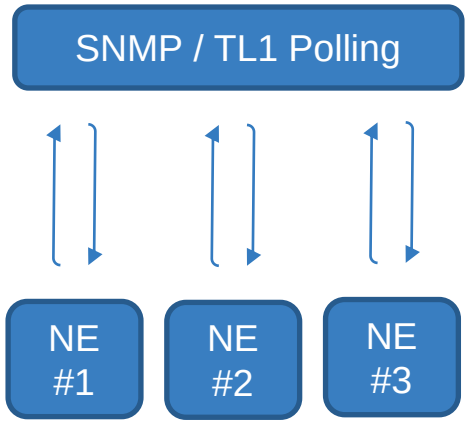


Better visibility with streaming telemetry

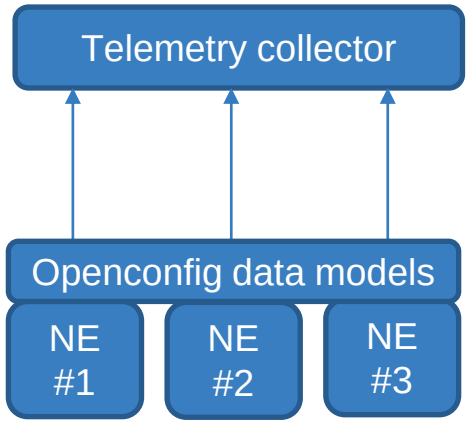
Operational state monitoring is crucial for network health and traffic management.

Examples:

- Counters, power levels, protocol stats, up/down events, inventory, alarms



- O(min) polling
- Resource drain on devices
- Legacy implementation
- Inflexible structure



- Subscribe to desired data based on models
- Streamed directly from devices
- Time-series or event-driven data
- Modern, secure transport

RPCs and gNMI

- gNMI is a protocol for the modification and retrieval of configuration from a target device, as well as the control and generation of telemetry streams from a target device to a data collection system.

https://github.com/openconfig/gnmi

- This gNMI is described using Protobuf:

https://github.com/openconfig/gnmi/blob/master/proto/gnmi/gnmi.proto

- The data can be either encoded in JSON or in Protobuf (Currently in JSON).

Why gNMI?

provides a single service for state management (streaming telemetry and configuration)

built on a modern standard, secure transport and open RPC framework with many language bindings

supports very efficient serialization and data access

- 3x-10x smaller than XML

offers an implemented alternative to NETCONF, RESTCONF, …

- early-release implementations on multiple router and transport platforms
- reference tools published by OpenConfig

https://datatracker.ietf.org/meeting/98/materials/slides-98-rtgwg-gnmi-intro-draft-openconfig-rtgwg-gnmi-spec-00

gNMI Terminology

- *Telemetry* - refers to streaming data relating to underlying characteristics of the device - either operational state or configuration.
- *Configuration* - elements within the data schema which are read/write and can be manipulated by the client.
- *Target* - the device within the protocol which acts as the owner of the data that is being manipulated or reported on. Typically this will be a network device.
- *Client* - the device or system using the protocol described in this document to query/modify data on the target, or act as a collector for streamed data. Typically this will be a network management system.

gNMI protocol buffer

```
service gNMI {
  rpc Capabilities(CapabilityRequest) returns (CapabilityResponse);
  rpc Get(GetRequest) returns (GetResponse);
  rpc Set(SetRequest) returns (SetResponse);
  rpc Subscribe(stream SubscribeRequest) returns (stream SubscribeResponse);
}
```

```
message GetRequest {
  Path prefix = 1;
  repeated Path path = 2;
  enum DataType {
    ALL = 0;
    CONFIG = 1;
    STATE = 2;
    OPERATIONAL = 3;
  }
  DataType type = 3;
  Encoding encoding = 5;
  repeated ModelData use_models = 6;
  repeated gnmi_ext.Extension extension = 7;
}

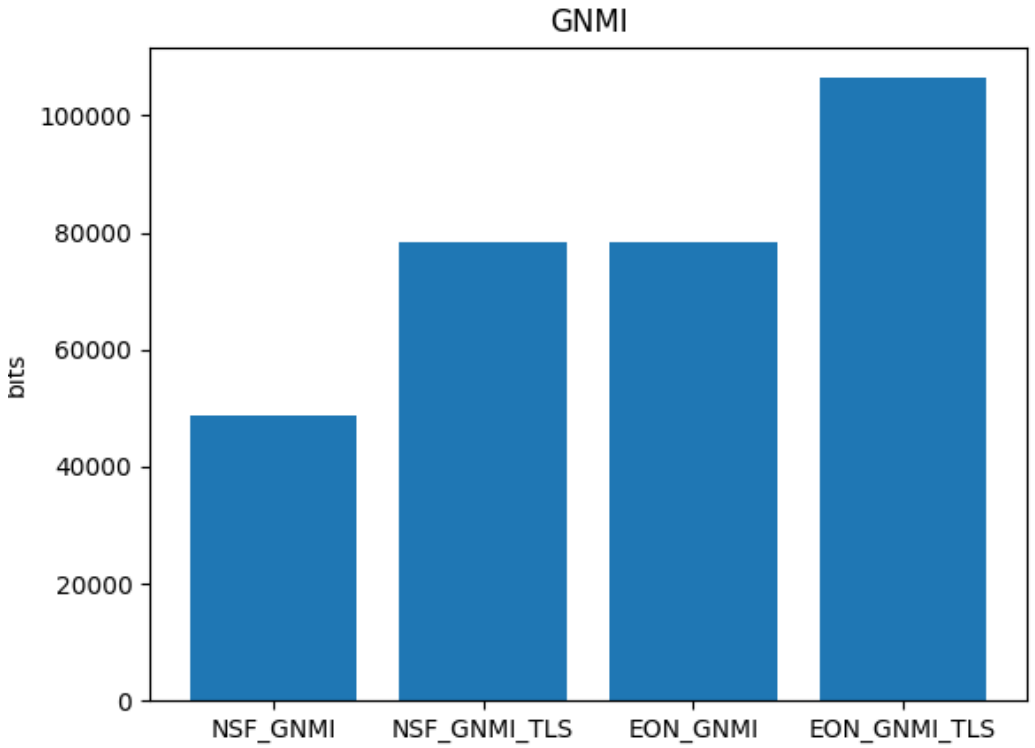
message GetResponse {
  repeated Notification notification = 1;
  Error error = 2 [deprecated=true];
  repeated gnmi_ext.Extension extension = 3;
}
```

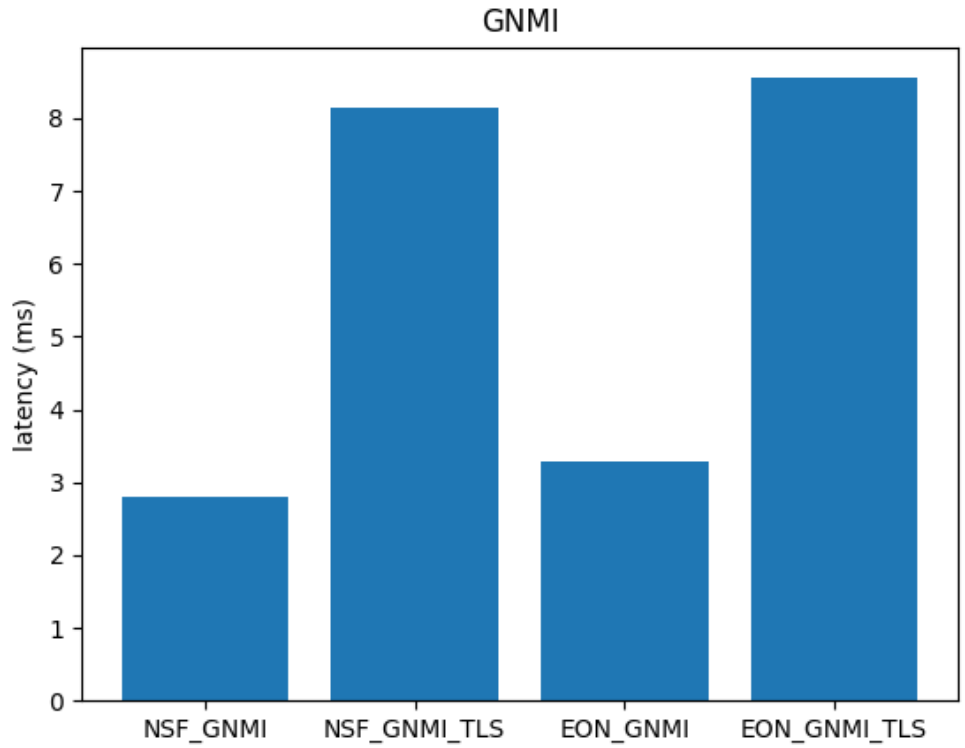
```
message CapabilityRequest {
  repeated gnmi_ext.Extension extension = 1;
}

message CapabilityResponse {
  repeated ModelData supported_models = 1;
  repeated Encoding supported_encodings = 2;
  string gNMI_version = 3;
  repeated gnmi_ext.Extension extension = 4;
}

message ModelData {
  string name = 1;
  string organization = 2;
  string version = 3;
}
```

Example gNMI results





TeraFlow SDN

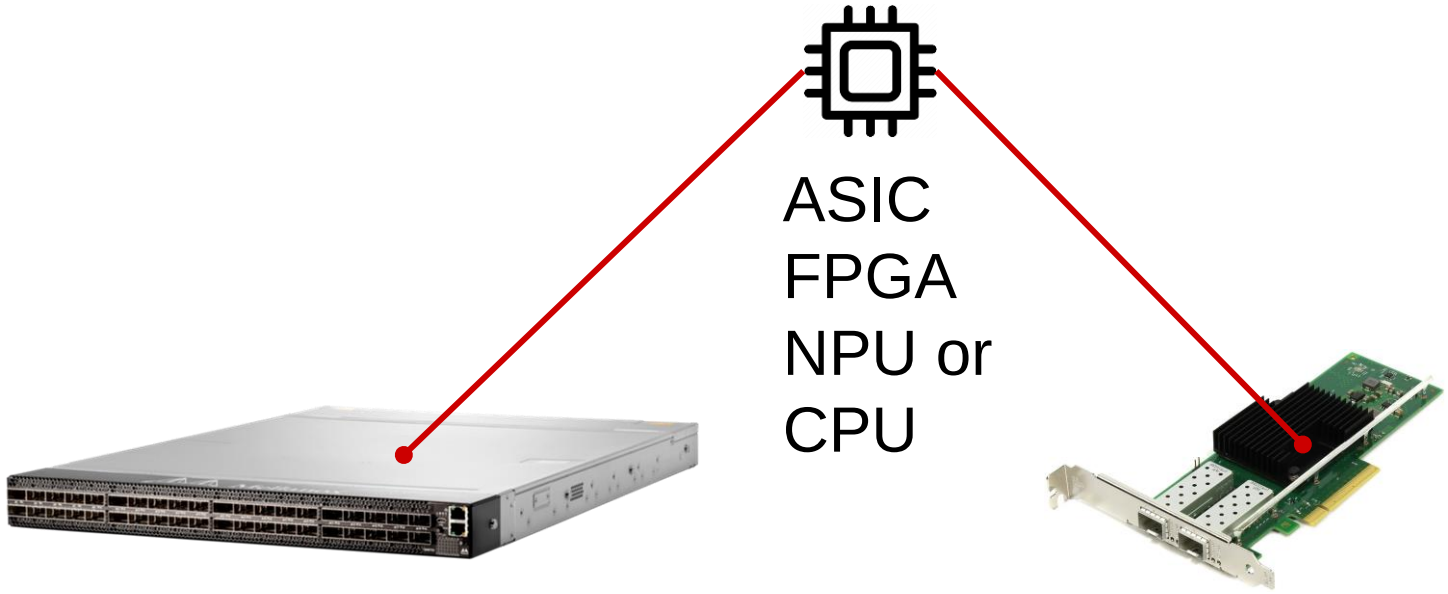
Introduction to P4

P4 Demonstration - Preparation

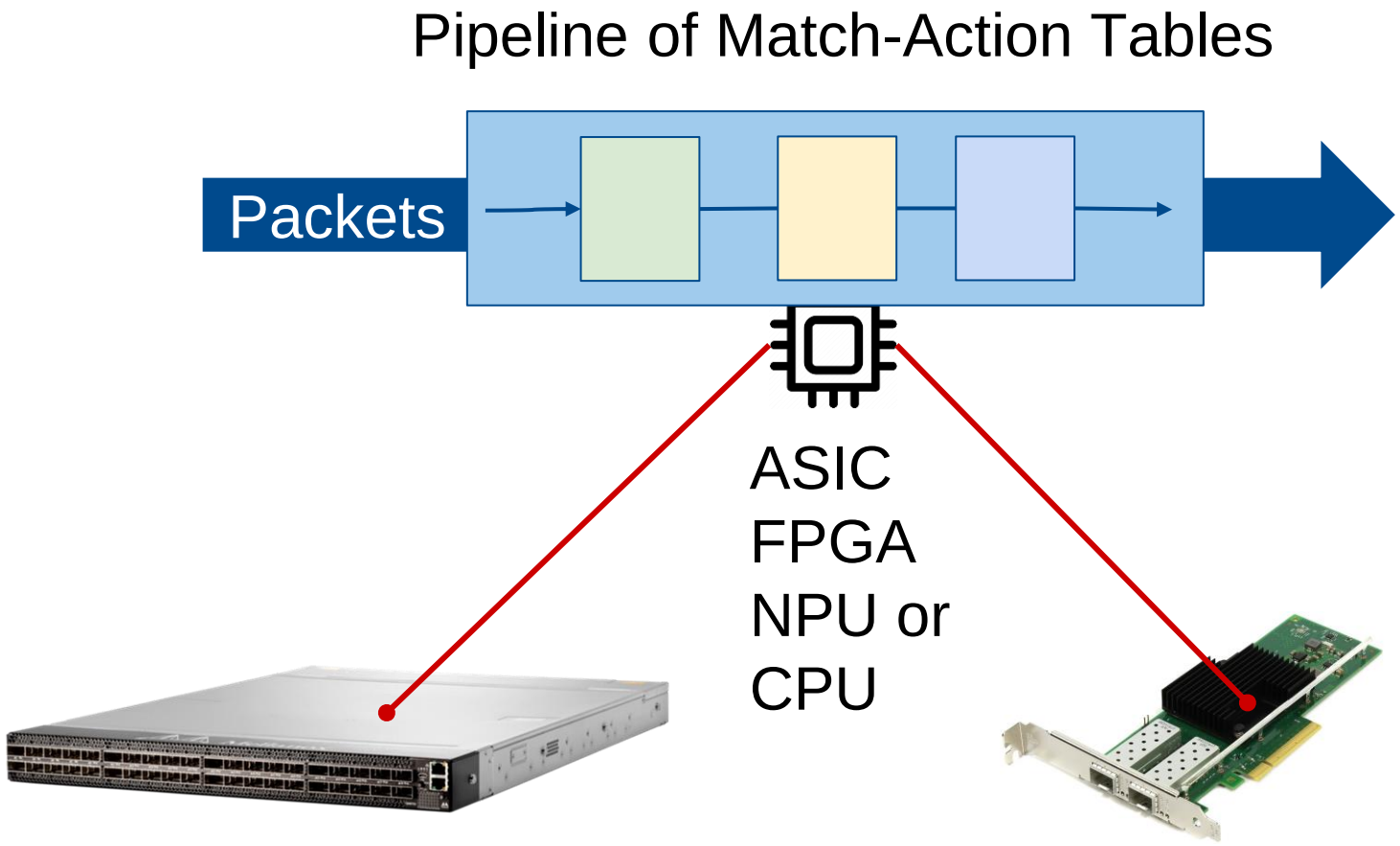
If you want to run this demo on your VM, please follow the next steps:

1. Install docker-compose:
   - `sudo apt-get install docker-compose`
2. Clone ONF's next-generation SDN Tutorial:
   - `cd ~`
   - `git clone -b advanced https://github.com/opennetworkinglab/ngsdn-tutorial`
3. Install dependencies:
   - `cd ngsdn-tutorial`
   - `make deps`
4. Insert the following in the Makefile (`~/ngsdn-tutorial/Makefile`):
   - ```
     start-simple: NGSDN_TOPO_PY := topo-simple.py
     start-simple: _start
     ```
5. Copy this tutorial's simple topology file into the ngsdn-tutorial repo:
   - `cp ~/tfs-ctrl/src/tests/netx22-p4/mininet/topo-simple.py ~/ngsdn-tutorial/mininet/`
     ```

What is a packet processing pipeline?



What is a packet processing pipeline?



The need for a different SDN

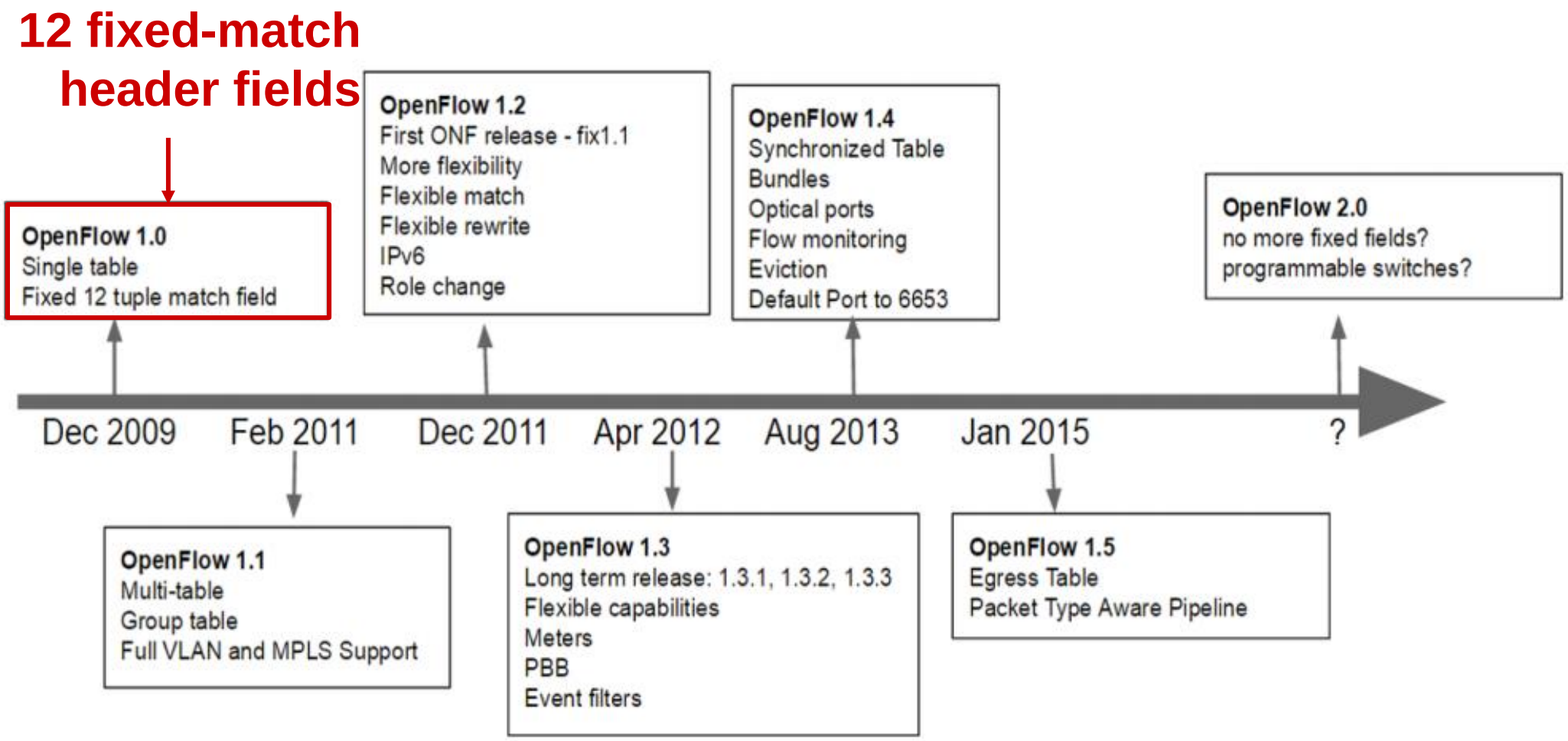


***Figure source:** https://kspviswa.github.io/OpenFlow_Version_Roadmap.html

The need for a different SDN

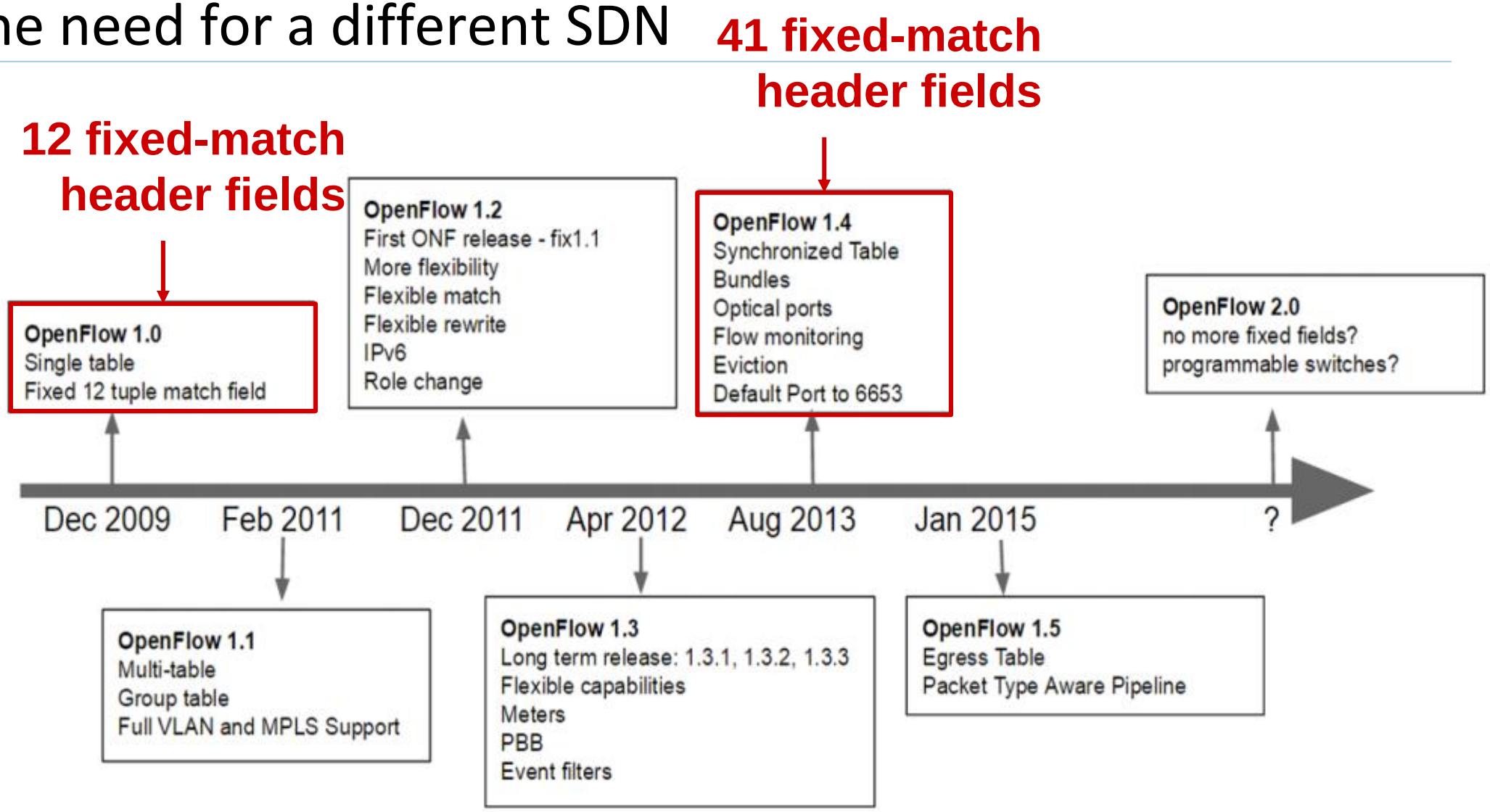


***Figure source:** https://kspviswa.github.io/OpenFlow_Version_Roadmap.html

The need for a different SDN

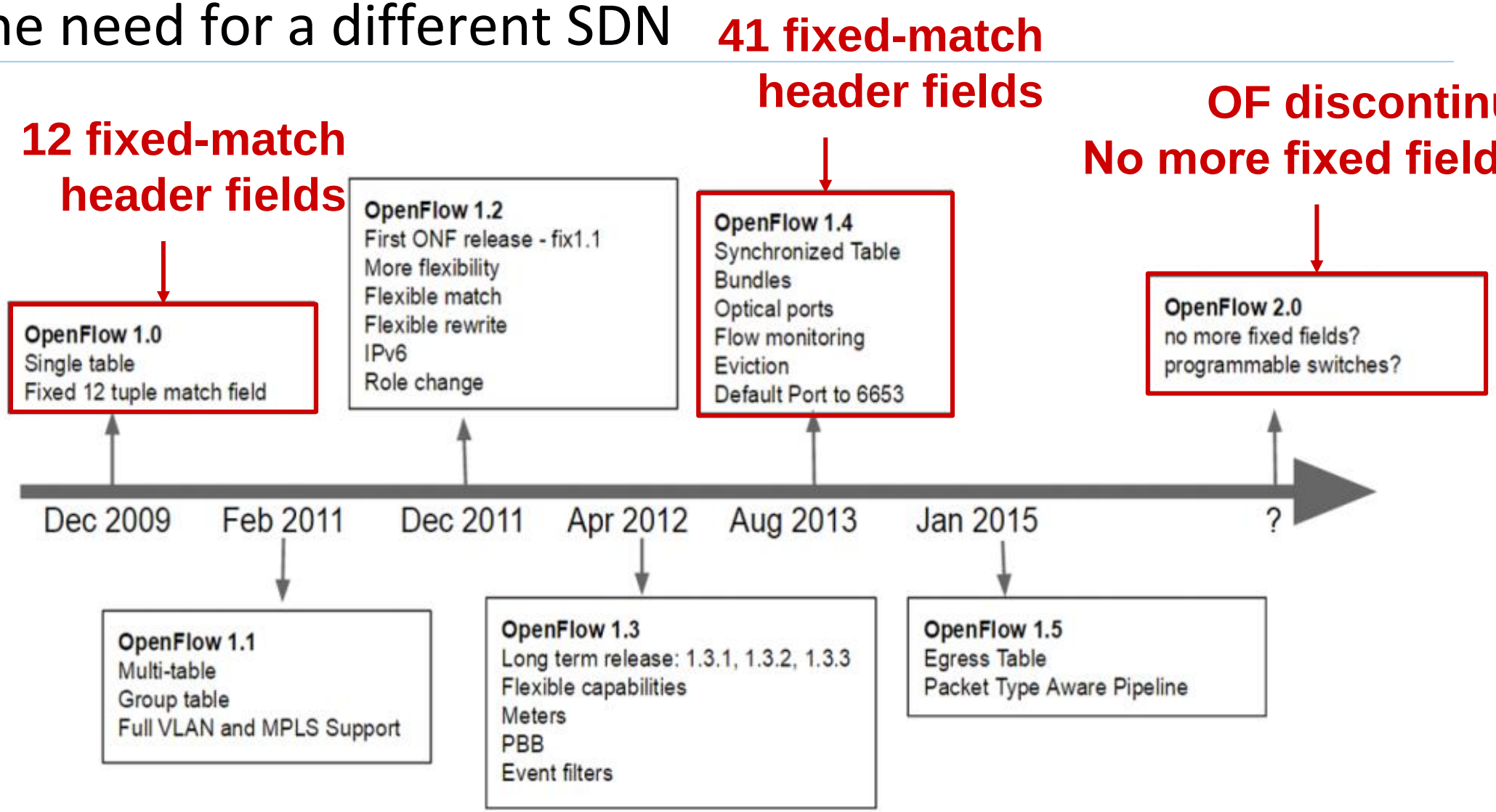


***Figure source:** https://kspviswa.github.io/OpenFlow_Version_Roadmap.html

The need for a different SDN

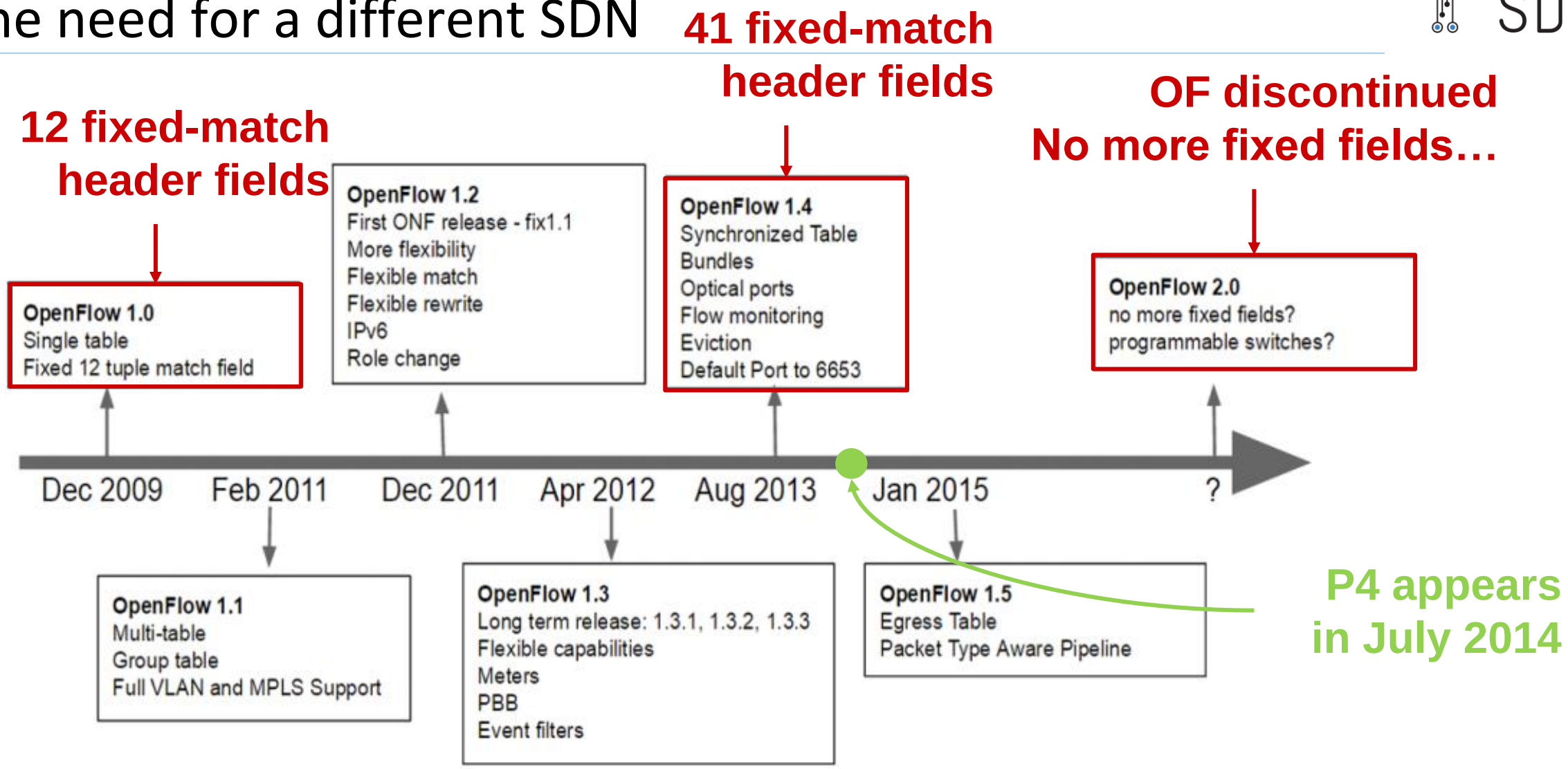


***Figure source:** https://kspviswa.github.io/OpenFlow_Version_Roadmap.html

The sad reality about OpenFlow

TeraFlow SDN

Specification

v1.4

41 fixed-match header fields

17 action types

The sad reality about OpenFlow

TeraFlow SDN

Specification	 v1.4	41 fixed-match header fields	17 action types

In reality

Most hardware switches only support a limited match/action set due to ASIC limitations

Hardware re-design requires long development cycles and increases cost

Why P4?

P4 motivation: Instead of repeatedly extending the OpenFlow standards, let's define a whole new abstraction for programming the data plane

Why P4?

P4 motivation: Instead of repeatedly extending the OpenFlow standards, let's define a whole new abstraction for programming the data plane

P4 principles:

- a domain-specific programming language to formally define the data plane pipeline
  - Describes proto headers (existing+new), lookup tables, actions, counters, etc.
  - Describes both fast (ASIC, FPGA) and slow (e.g., soft. switch) pipelines

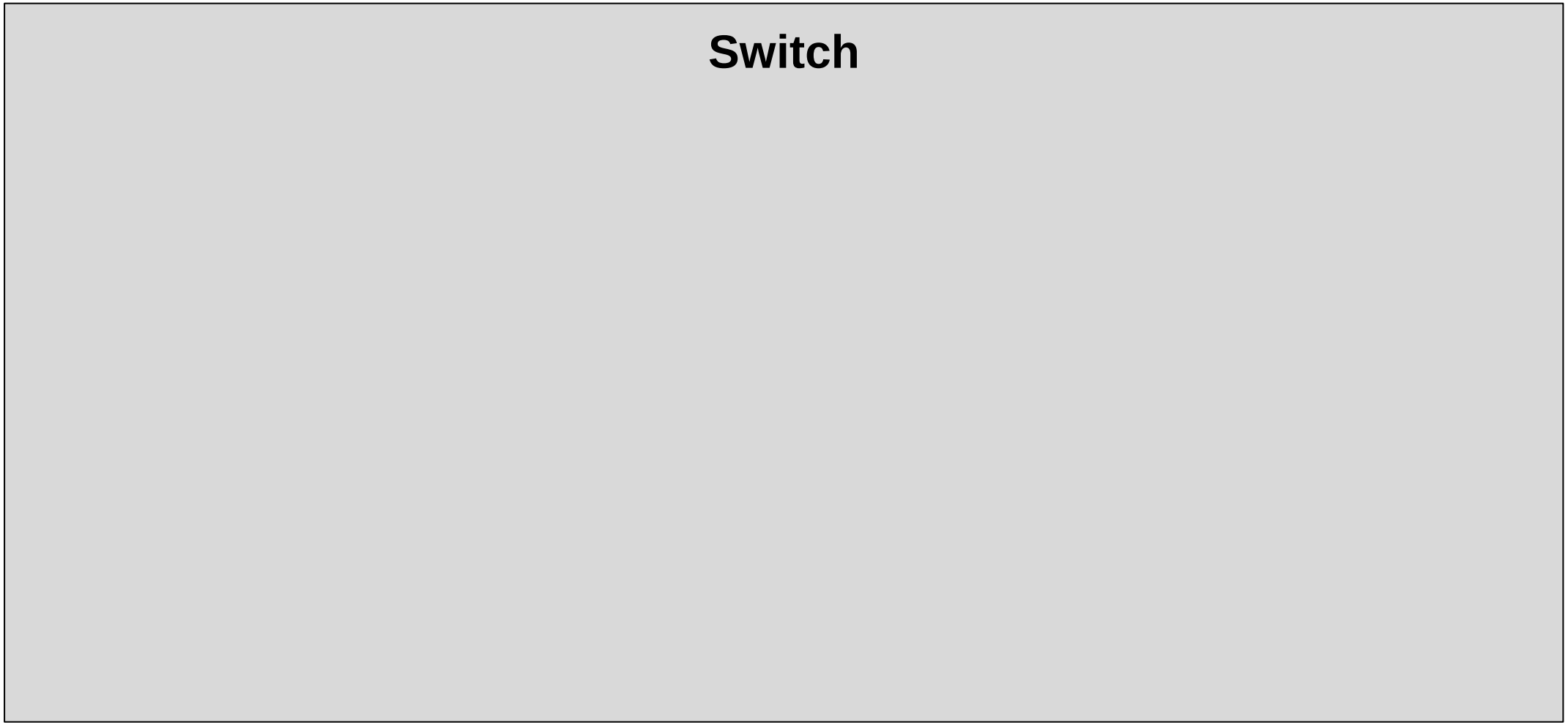
Why P4?

P4 motivation: Instead of repeatedly extending the OpenFlow standards, let's define a whole new abstraction for programming the data plane

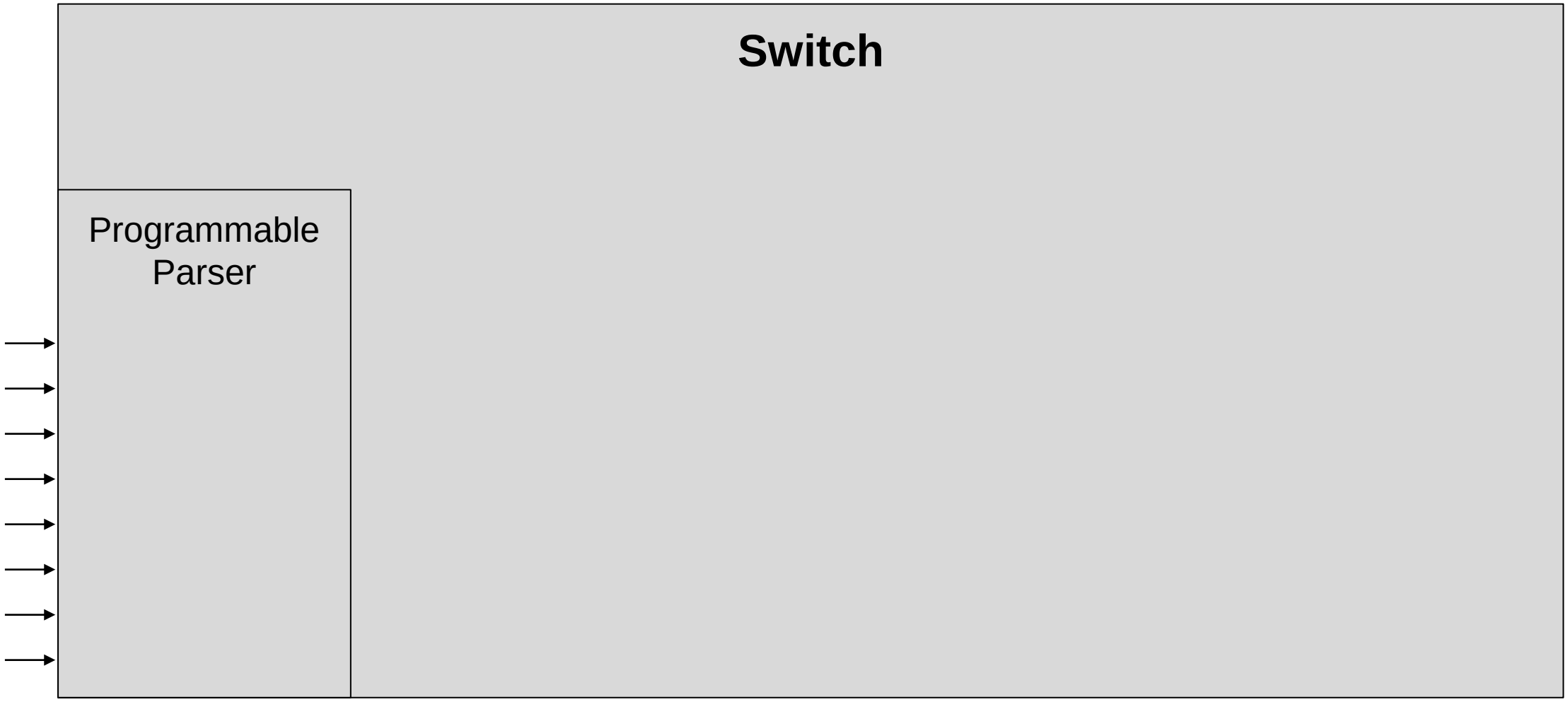
P4 principles:

- a domain-specific programming language to formally define the data plane pipeline
  - Describes proto headers (existing+new), lookup tables, actions, counters, etc.
  - Describes both fast (ASIC, FPGA) and slow (e.g., soft. switch) pipelines
- a common interface to parse packets and match (arbitrary) header fields
  - Defines a "contract" between the control and data plane

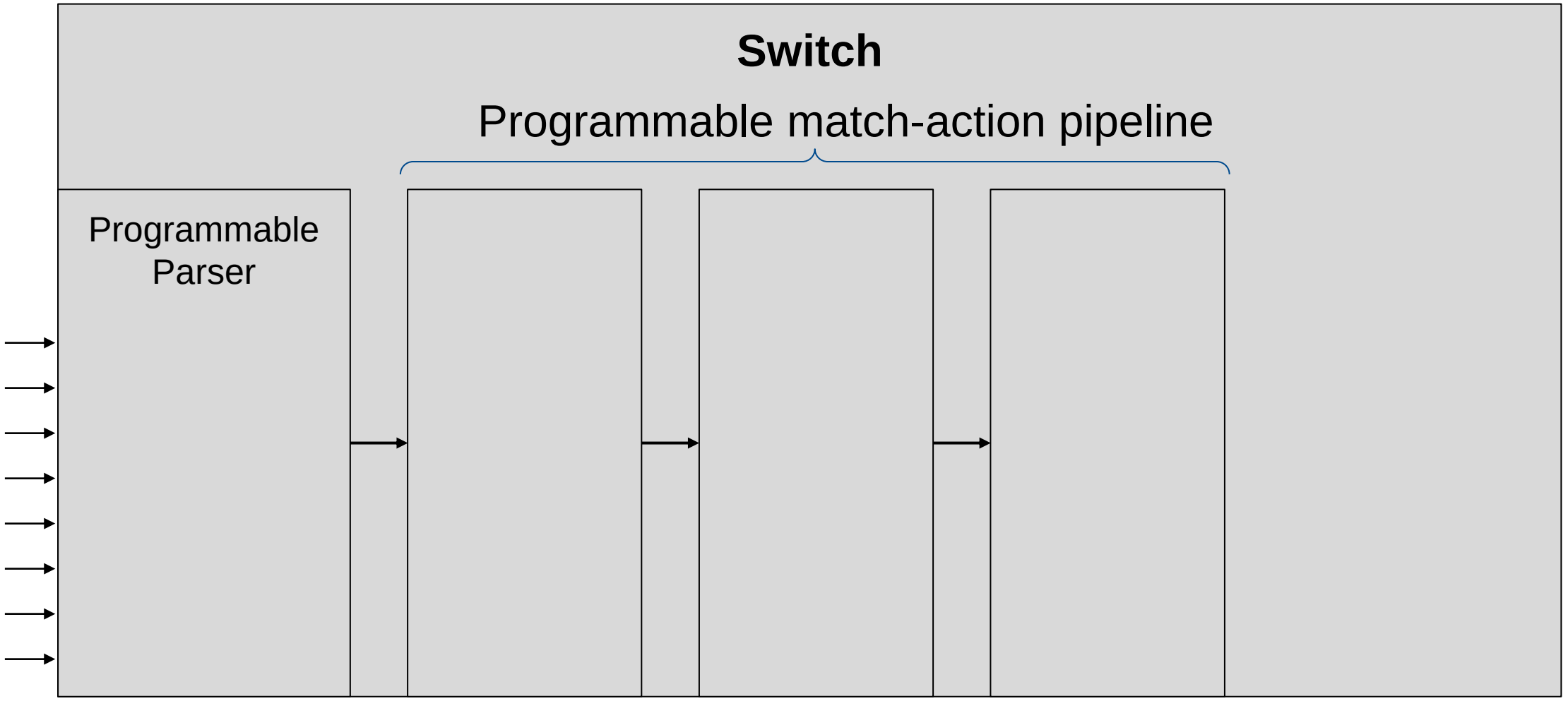
PISA: Protocol-Independent Switch Architecture



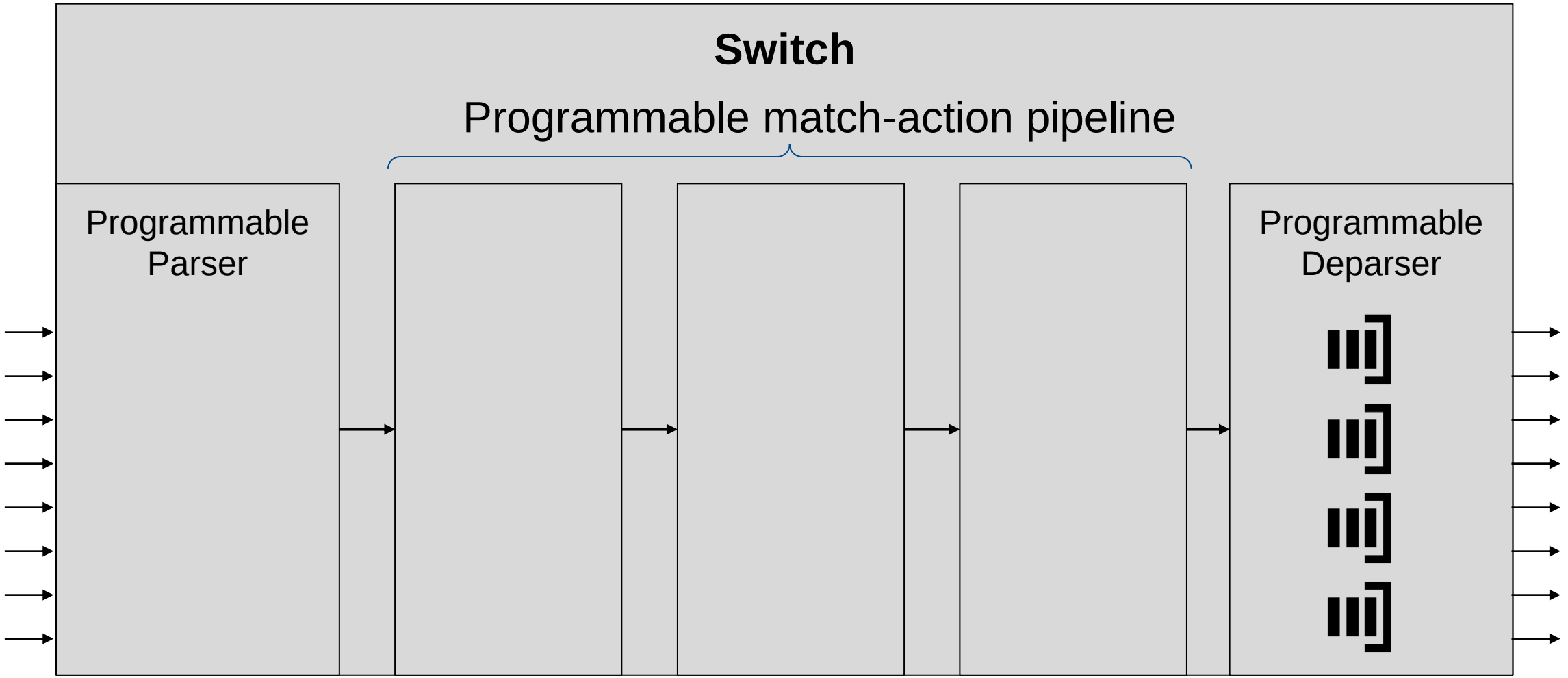
PISA: Protocol-Independent Switch Architecture



PISA: Protocol-Independent Switch Architecture



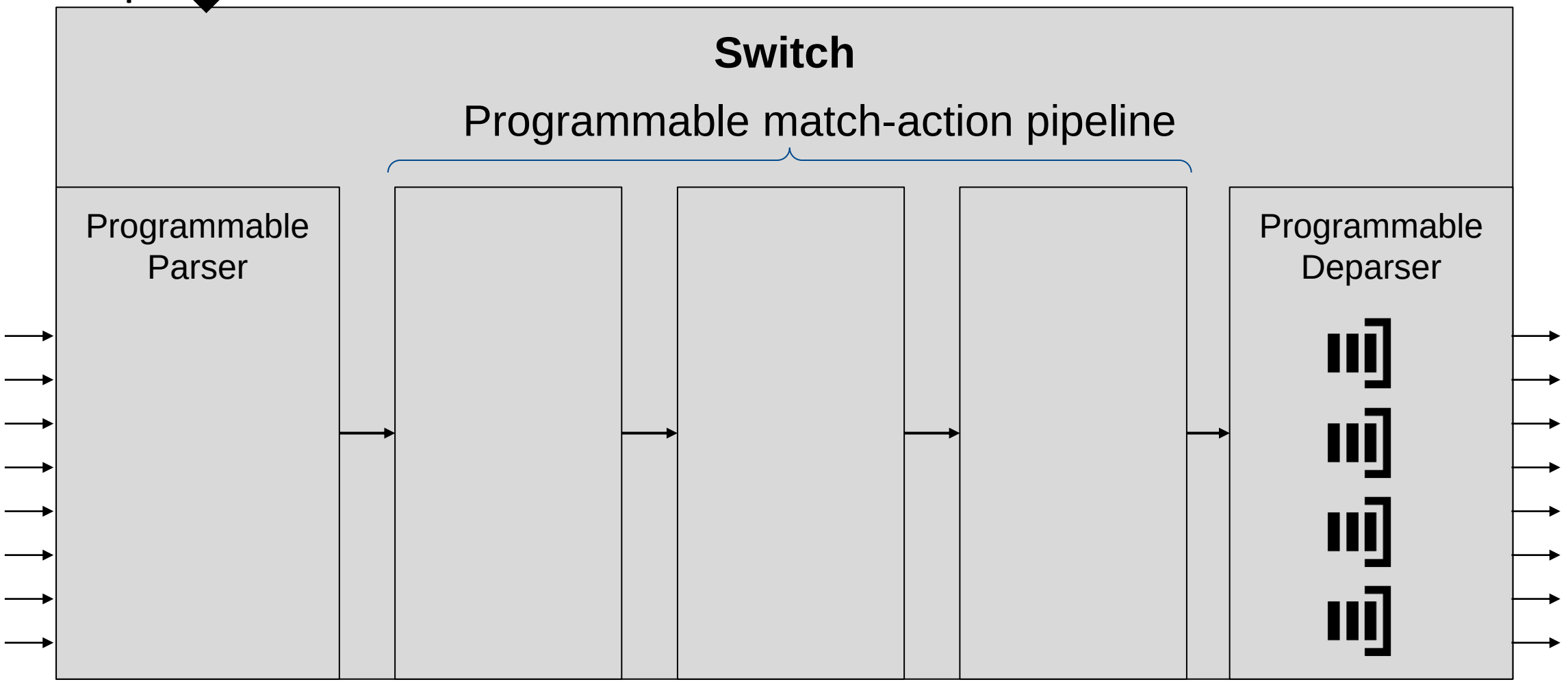
PISA: Protocol-Independent Switch Architecture



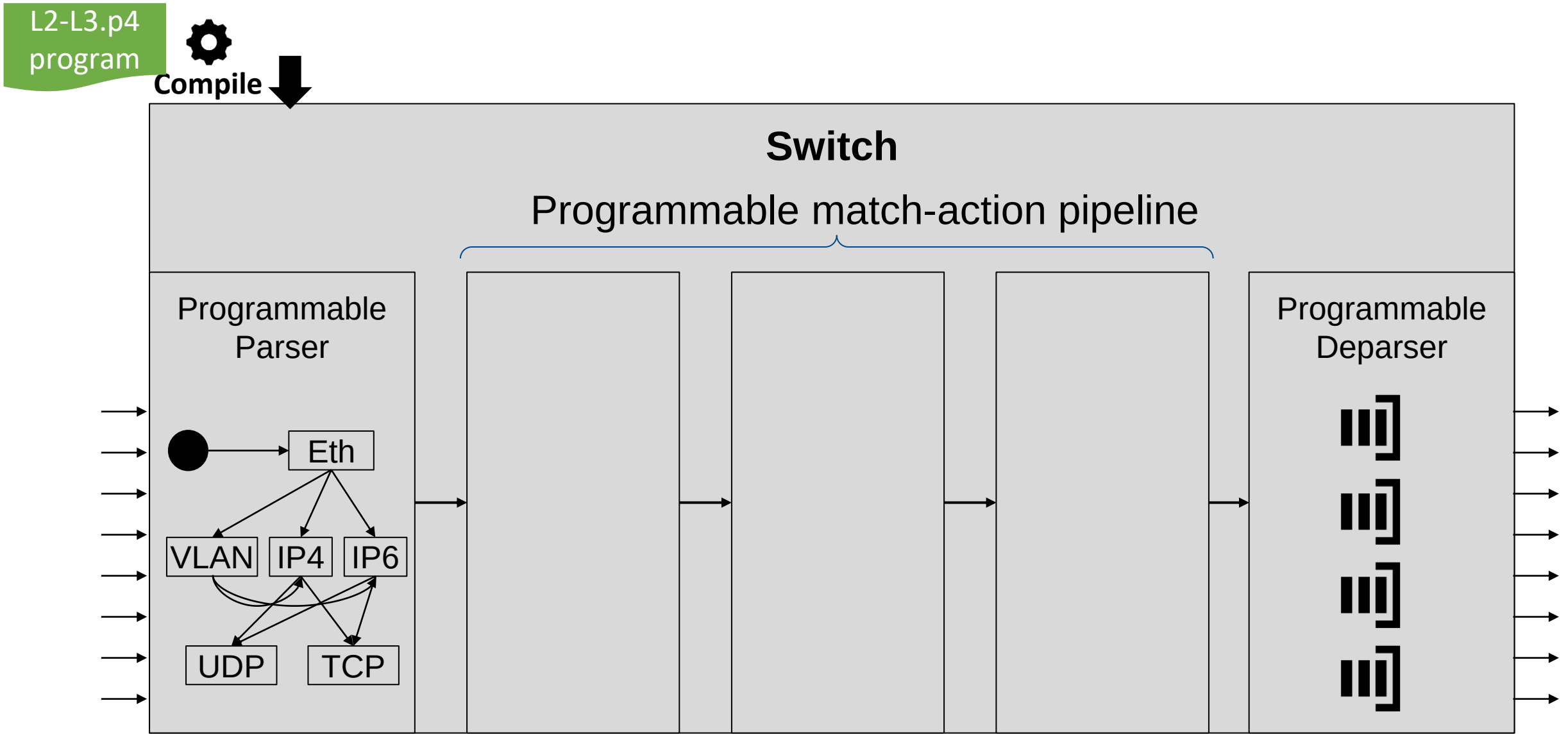
PISA: Protocol-Independent Switch Architecture

L2-L3.p4 program

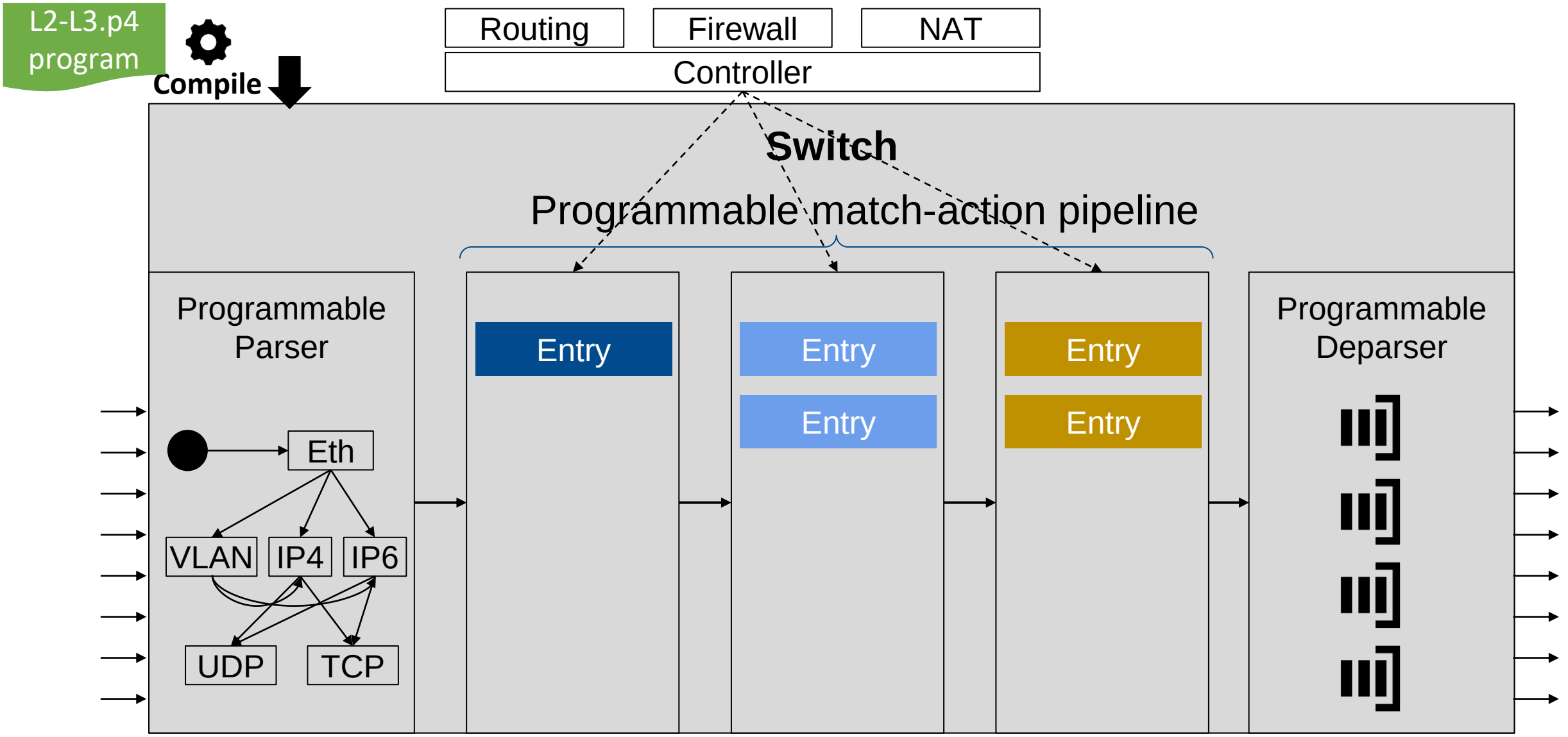
Compile



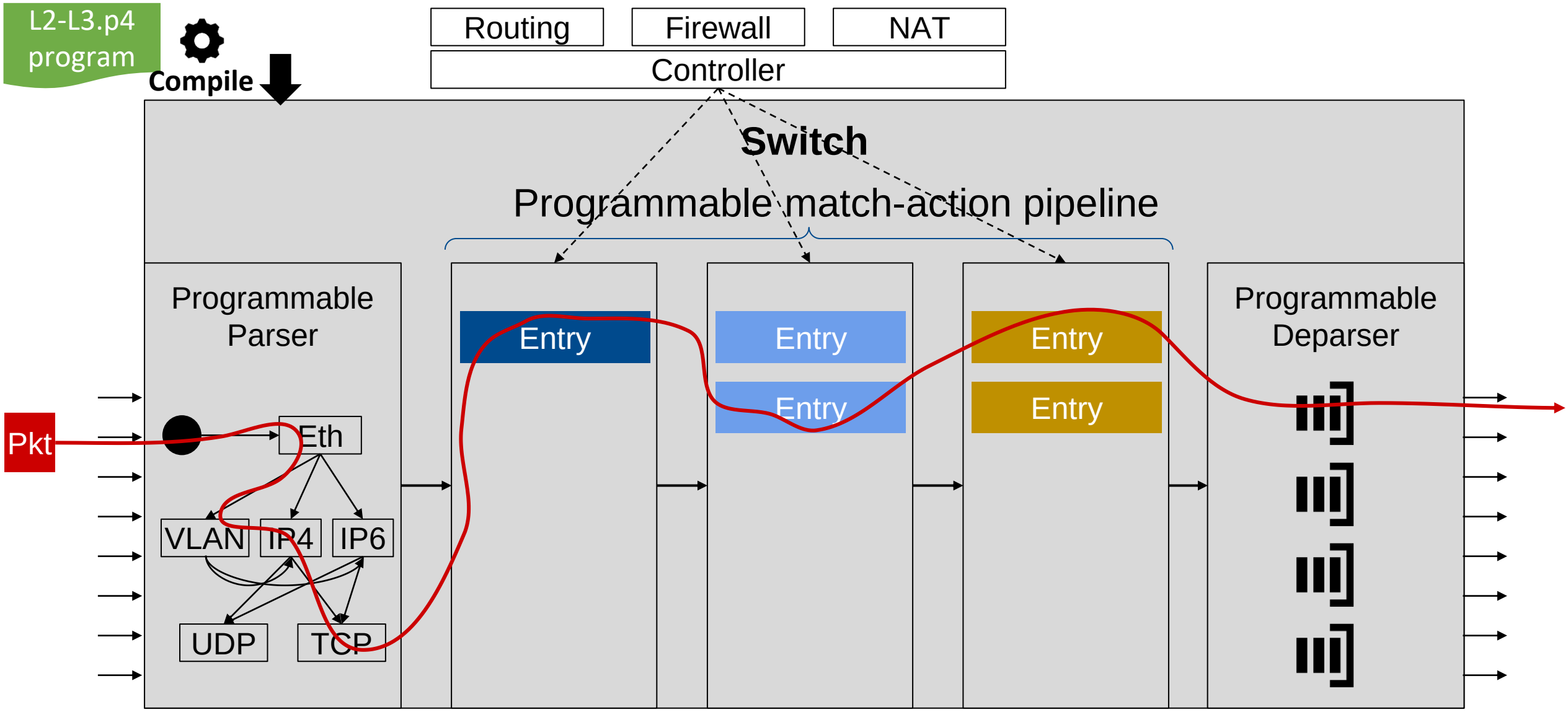
PISA: Protocol-Independent Switch Architecture



PISA: Protocol-Independent Switch Architecture



PISA: Protocol-Independent Switch Architecture



But switches still have ASICs…

Correct, but…

But switches still have ASICs…

Correct, but…

- New custom ASICs offer decent flexibility even at several Tera bits/sec

But switches still have ASICs…

Correct, but…

- New custom ASICs offer decent flexibility even at several Tera bits/sec
- Some switches offer higher programmability than others:
  - FPGAs (e.g., Intel, Xilinx)
  - NPUs (e.g., Netronome, EZchip)
  - Software-based (e.g., OVS)

P4Runtime: Not just yet another P4 API?

P4Runtime is a runtime control API for P4-defined data planes

P4Runtime: Not just yet another P4 API?

P4Runtime is a runtime control API for P4-defined data planes

API	Target independent Same API works with different switches from different vendors
OpenFlow	
P4Runtime	

P4Runtime: Not just yet another P4 API?

TeraFlow SDN

P4Runtime is a runtime control API for P4-defined data planes

API	Target independent Same API works with different switches from different vendors	Pipeline independent Same API allows control of many arbitrary pipelines
OpenFlow	✓	✓ With table type patterns (TTP)
P4Runtime	✓	✓ With P4

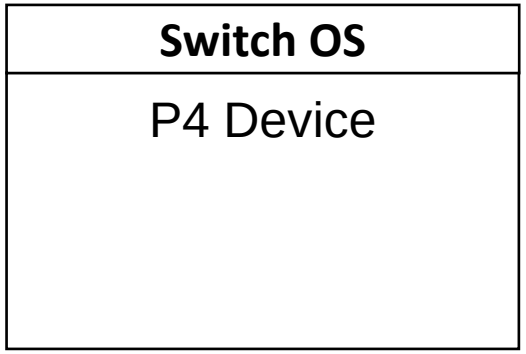
P4Runtime: Not just yet another P4 API?

TeraFlow SDN

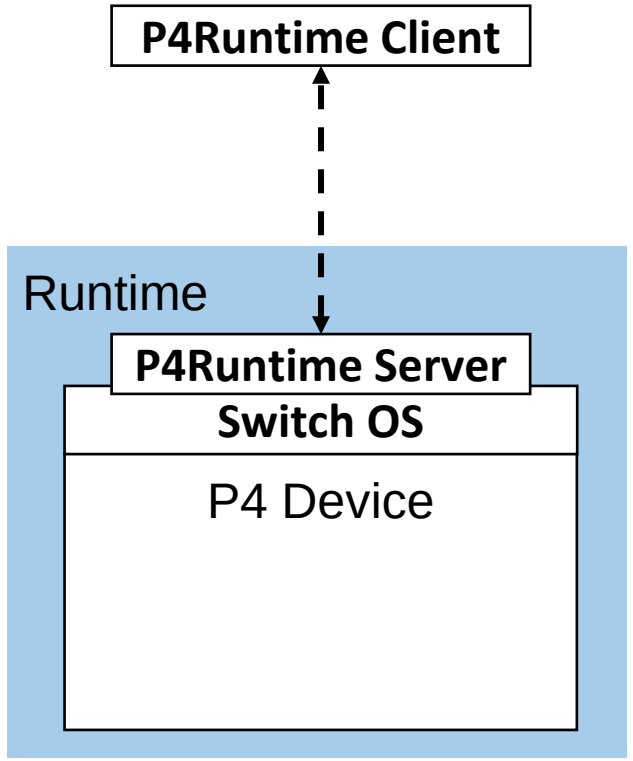
P4Runtime is a runtime control API for P4-defined data planes

API	Target independent Same API works with different switches from different vendors	Pipeline independent Same API allows control of many arbitrary pipelines	Protocol independent Same API allows control of any data plane proto (standard/custom)
OpenFlow	✓	✓ With table type patterns (TTP)	✗ Proto headers and actions hardcoded in the spec
P4Runtime	✓	✓ With P4	✓

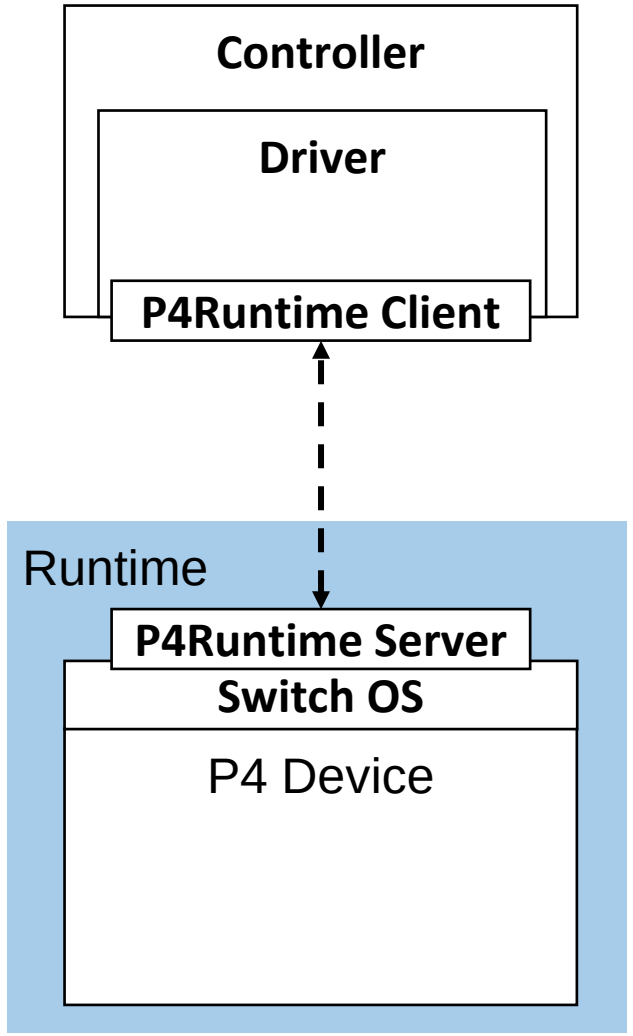
P4 workflow



P4 workflow



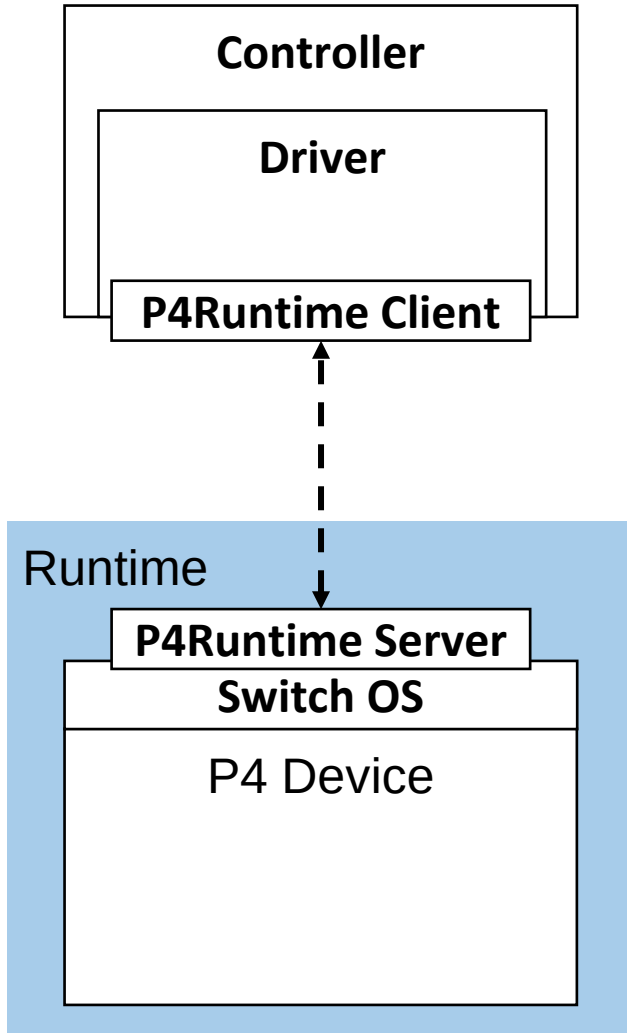
P4 workflow



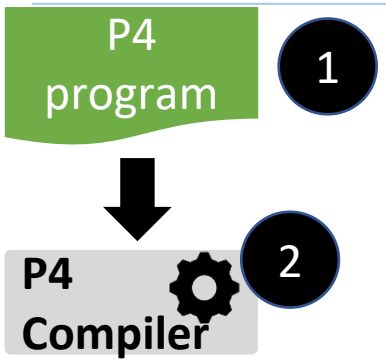
P4 workflow

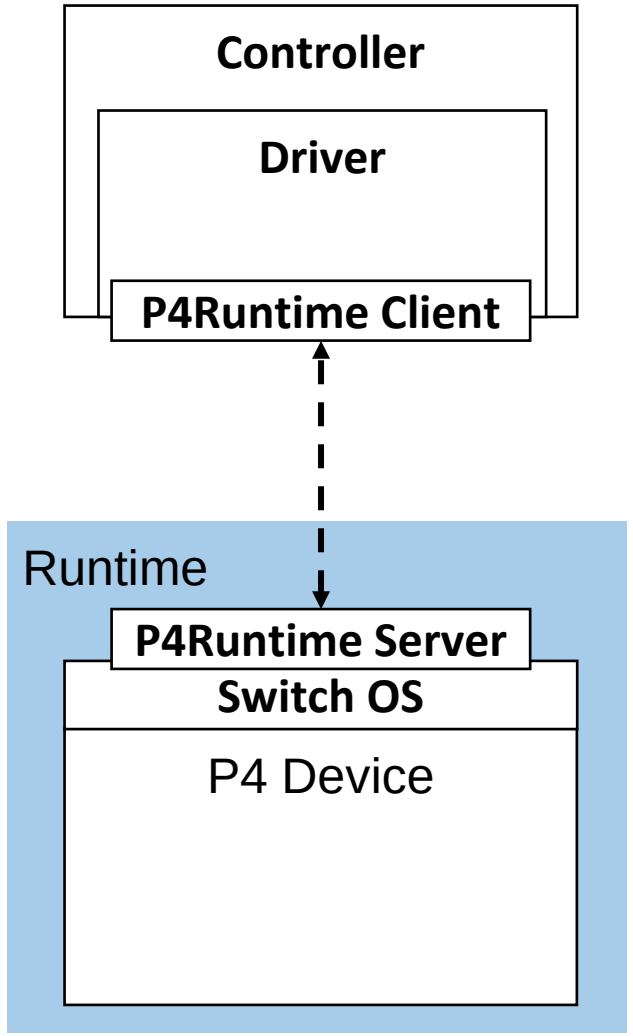
P4 program

1

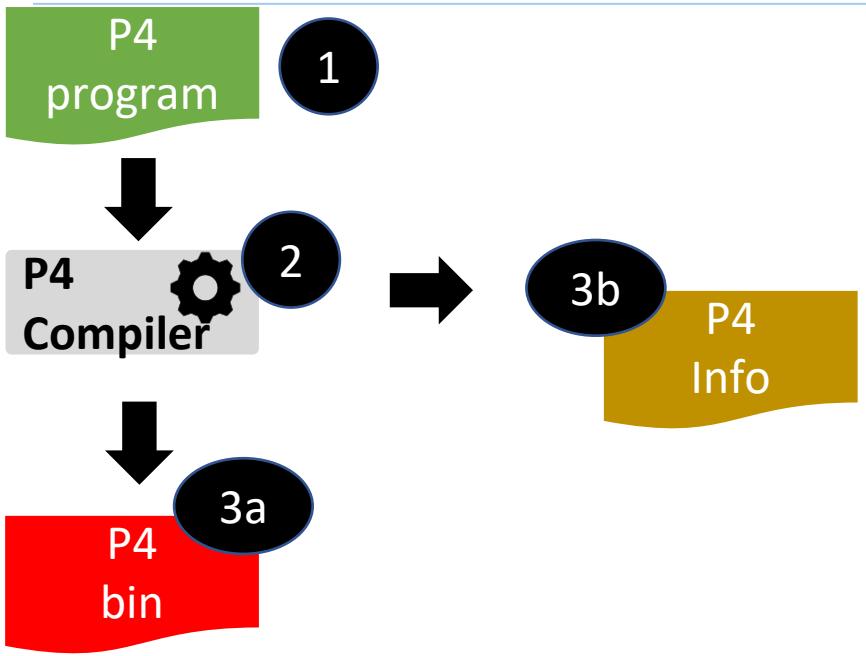


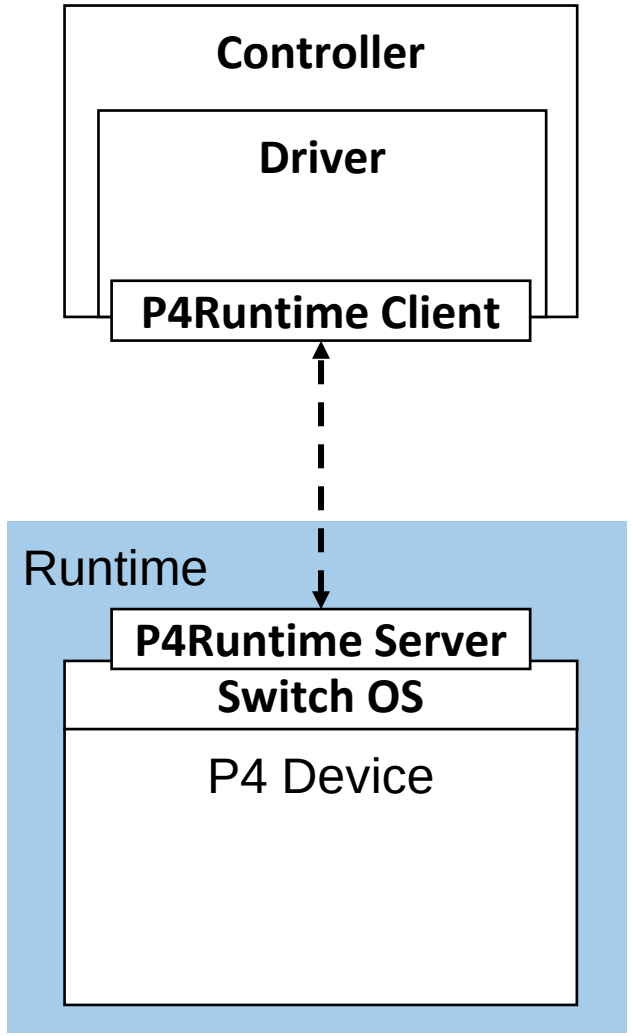
P4 workflow



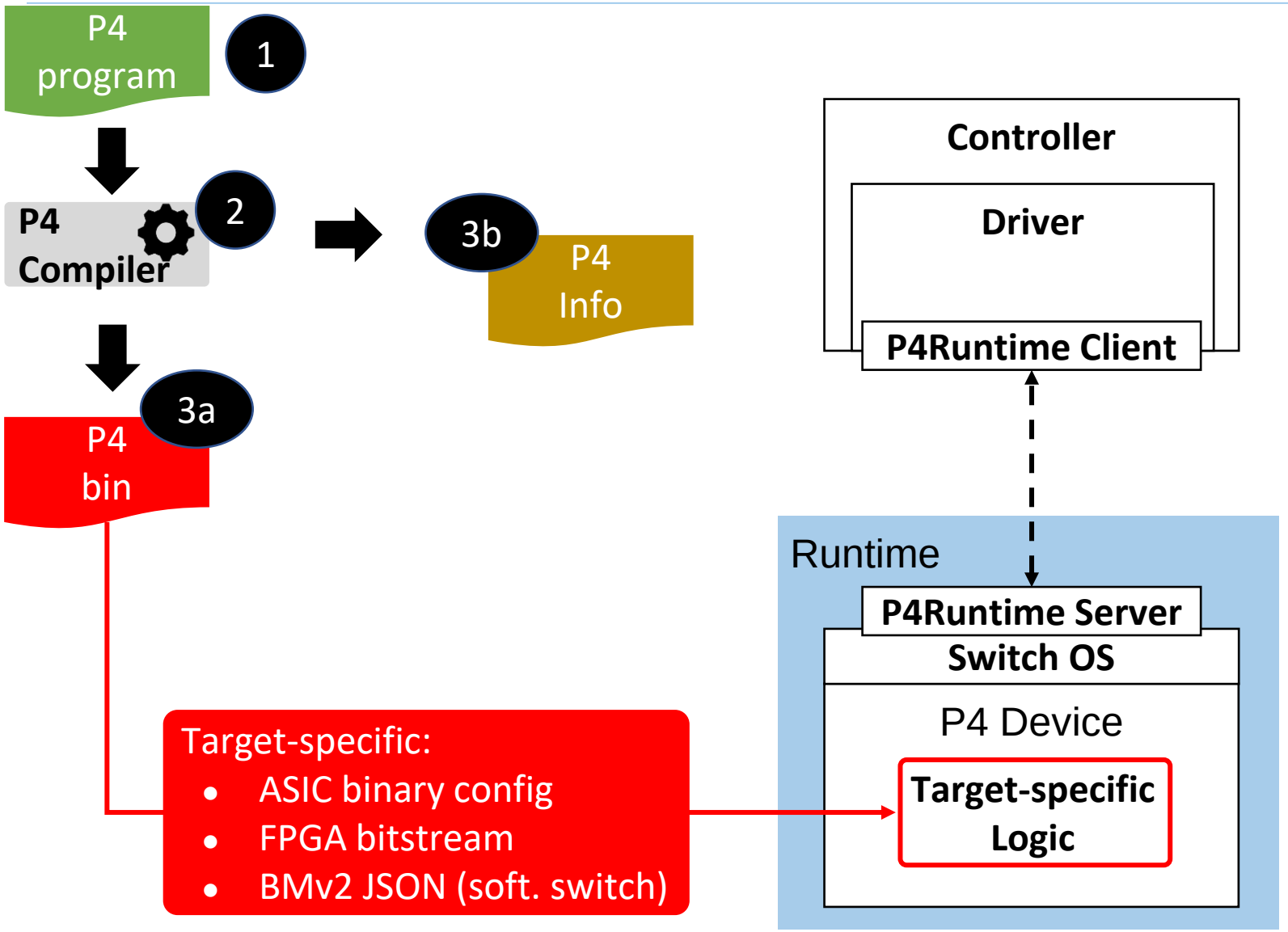


P4 workflow

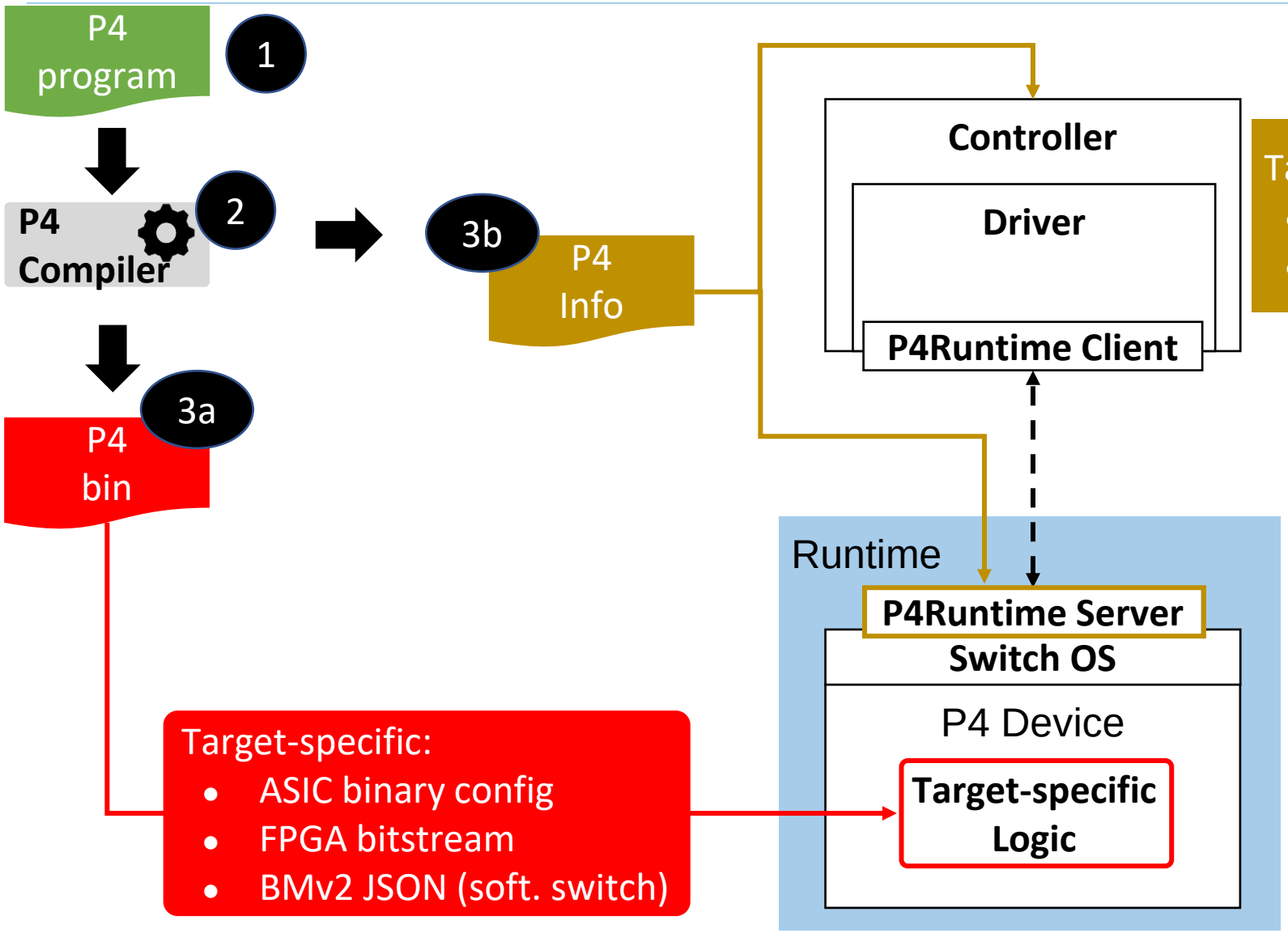




P4 workflow



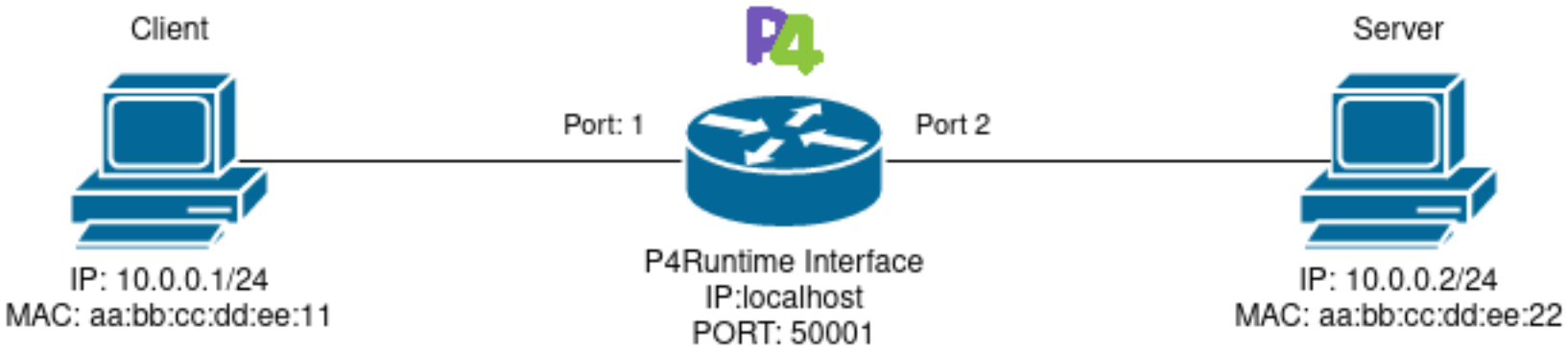
P4 workflow



Mini P4 Demonstration

P4 Demonstration - Setup

This demonstration consists of a simple client/server setup, with two hosts interconnected via a software-based P4 switch



P4 Demonstration - Scenario

Objective: Realize a simple l2 forwarding program in p4

P4 Demonstration - Scenario

Objective: Realize a simple l2 forwarding program in p4

Approach: Bi-directional ICMP connectivity from client to server

- Use mininet to emulate the network
- Static ARP entries in client and server (for the needs of this simple demo)
- Client sends an ICMP echo-request to the server
- Server replies to the client with an ICMP echo-reply

P4 Demonstration - L2 forwarding program

Types definitions:

```
typedef bit<9>   port_num_t;
typedef bit<48>  mac_addr_t;
```

l2 headers (ethernet):

```
header ethernet_t {
       mac_addr_t   dst_addr;
       mac_addr_t   src_addr;
       bit<16>
       ether_type;
}
```

Parser:

```
state start {
       transition parse_ethernet;
}

state parse_ethernet {
       packet.extract(hdr.ethernet);
       transition accept;
}
```

P4 Demonstration - L2 forwarding program

Actions:

```
action drop() {
  mark_to_drop(standard_metadata);
}

action set_egress_port(port_num_t
port_num) {
  standard_metadata.egress_spec =

      port_num;
}
```

Table:

```
table l2_exact_table {
  key = {
    hdr.ethernet.dst_addr: exact;
  }
    actions = {
      set_egress_port;
      @defaultonly drop;
    }
      const default_action = drop;
}
```

P4 Demonstration - L2 forwarding program

Actions:

```
action drop() {
  mark_to_drop(standard_metadata);
}

action set_egress_port(port_num_t
port_num) {
  standard_metadata.egress_spec =

      port_num;
}
```

Table:

```
table l2_exact_table {
  key = {
    hdr.ethernet.dst_addr: exact;
  }
  actions = {
    set_egress_port;
    @defaultonly drop;
  }
    const default_action = drop;
}
```

P4 Demonstration - Device details

The TeraFlowSDN controller connects to a device as follows:

```
DEVICE_SW1_CONNECT_RULES = json_device_connect_rules(
        DEVICE_SW1_IP_ADDR,                             # localhost
        DEVICE_SW1_PORT,                                # 50001
        {
                    'id': DEVICE_SW1_DPID,              # 1
                  'name': DEVICE_SW1_NAME,              # SW1
                'vendor': DEVICE_SW1_VENDOR,            # ONF
                'hw_ver': DEVICE_SW1_HW_VER,            # BMv2 simple_switch
                'sw_ver': DEVICE_SW1_SW_VER,            # Stratum
               'timeout': DEVICE_SW1_TIMEOUT,           # 60
                 'p4bin':      DEVICE_SW1_BIN_PATH,     # P4 binary in the device pod
                'p4info': DEVICE_SW1_INFO_PATH          # P4 info in the device pod
        }
)
```

P4 Demonstration - Rules

P4 Rules are composed based on the p4info.txt:

```
'table-name': 'IngressPipeImpl.l2_exact_table',
  'match-fields': [
      {
            'match-field': 'hdr.ethernet.dst_addr',
            'match-value': 'aa:bb:cc:dd:ee:11'
      }
  ],
  'action-name': 'IngressPipeImpl.set_egress_port',
  'action-params': [
      {
            'action-param': 'port_num',
            'action-value': '1'
      }
  ]
}
```

P4 Demonstration - Rules

P4 Rules are composed based on the p4info.txt:

```
'table-name': 'IngressPipeImpl.l2_exact_table',
  'match-fields': [
      {
            'match-field': 'hdr.ethernet.dst_addr',
            'match-value': 'aa:bb:cc:dd:ee:11'
      }
  ],
  'action-name': 'IngressPipeImpl.set_egress_port',
  'action-params': [
      {
            'action-param': 'port_num',
            'action-value': '1'
      }
  ]
}
```

Table to insert rule

P4 Demonstration - Rules

P4 Rules are composed based on the p4info.txt:

```
'table-name': 'IngressPipeImpl.l2_exact_table',
  'match-fields': [
      {
          'match-field': 'hdr.ethernet.dst_addr',
          'match-value': 'aa:bb:cc:dd:ee:11'
      }
  ],
  'action-name': 'IngressPipeImpl.set_egress_port',
  'action-params': [
      {
          'action-param': 'port_num',
          'action-value': '1'
      }
  ]
}
```

Table to insert rule

Match client's destination MAC address (Table key)

P4 Demonstration - Rules

P4 Rules are composed based on the p4info.txt:

```
'table-name': 'IngressPipeImpl.l2_exact_table',
  'match-fields': [
      {
          'match-field': 'hdr.ethernet.dst_addr',
          'match-value': 'aa:bb:cc:dd:ee:11'
      }
  ],
  'action-name': 'IngressPipeImpl.set_egress_port',
  'action-params': [
      {
          'action-param': 'port_num',
          'action-value': '1'
      }
  ]
}
```

Table to insert rule

Match client's destination MAC address (Table key)

Action to call

Parameter value for the action

P4 Demonstration - Step 1

- Run mininet:

```
cd ~/ngsdn-tutorial
make start-simple
make mn-cli
```

Expected Results:

- mininet starts with the correct topology
  - Run nodes to check

P4 Demonstration - Step 1

- Run mininet:

```
cd ~/ngsdn-tutorial
make start-simple
make mn-cli
```

- Try to ping using the mininet CLI:

```
client ping server
```

- Let it run

Expected Results:

- mininet starts with the correct topology
  - Run nodes to check
- We do not get any reply on client

P4 Demonstration - Step 2

On another terminal:

- Run setup script: `src/tests/netx22-p4/setup.sh`

Expected Results:

- The p4 artifacts are copied to the device pod

P4 Demonstration - Step 2

On another terminal:

- Run setup script: `src/tests/netx22-p4/setup.sh`

- Run bootstrap script `src/tests/netx22-p4/run_test_01_bootstrap.sh`

Expected Results:

- The p4 artifacts are copied to the device pod

- The device should be registered on the TeraFlowSDN UI

P4 Demonstration - Step 3

- Run create service script:
  `src/tests/netx22-p4/run_test_02_create_service.sh`

Expected Results:

- Server should reply
  - Check mininet terminal

P4 Demonstration - Step 3

- Run create service script:
  `src/tests/netx22-p4/run_test_02_create_service.sh`

Expected Results:

- Server should reply
  - Check mininet terminal

- Run delete service script
  `src/tests/netx22-p4/run_test_03_delete_service.sh`

- Server should stop replying
  - Check mininet terminal

P4 Demonstration - Step 4

- Retry the previous step

Expected Results:

- Server should begin replying
  - Check that the `icmp_seq` has advanced

P4 Demonstration - Step 4

- Retry the previous step

Expected Results:

- Server should begin replying
  - Check that the `icmp_seq` has advanced

- Run cleanup script `src/tests/netx22-p4/run_test_04_cleanup.sh`

- Device is no longer registered to the TeraFlowSDN UI

P4 Demonstration - Discussion

- For the needs of this demo ARP entries are hardcoded

P4 Demonstration - Discussion

- For the needs of this demo ARP entries are hardcoded
- How would we implement ARP in P4?
  - Insert separate entries for each port?
    - Really bad idea, does not scale!

P4 Demonstration - Discussion

- For the needs of this demo ARP entries are hardcoded
- How would we implement ARP in P4?
  - Insert separate entries for each port?
    - Really bad idea, does not scale!
  - **Use native P4 multicast group IDs**

P4 Demonstration - ARP

Actions:

```
action set_multicast_group(
                mcast_group_id_t gid)
{
  standard_metadata.mcast_grp =  gid;
}
```

Table:

```
table l2_exact_table {
  key = {
    hdr.ethernet.dst_addr: exact;
  }
    actions = {
      set_egress_port;
      set_multicast_group;
      @defaultonly drop;
    }
    const default_action = drop;
}
```

P4 Demonstration - ARP

P4Runtime Rule:

```
multicast_group_entry {
      multicast_group_id: 1
      replicas {
            egress_port: 1
      }
      replicas {
            egress_port: 2
      }
}
```

TeraFlow SDN

Conclusion

Time to spread the acquired knowledge!

Conclusion: Protocol summary

	NETCONF	RESTconf	gRPC	gNMI
Data Modelling Language	YANG	YANG	Protocol Buffers	YANG / Protocol Buffers
Transport	SSH, TLS, BEEP/TLS, SOAP/HTTP/TLS	HTTP	HTTP/2	gRPC
Encoding	XML	XML/JSON	byte	JSON/byte
Capability exchange	During Session establishment	Retrieval of Yang modules and capability URIs	NO	Yes
Multiple datastores	YES	NO	NO	YES (Config/State/ Operational)
Datastore Locking	YES	NO	NO	NO
Security	SSH	TLS	TLS	TLS

Standards summary

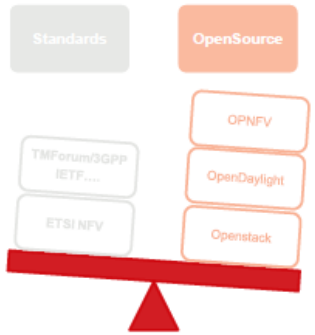
Standards	T-API	IETF TEAS	OpenROADM	OpenConfig	gNMI
Focus	NBI Transport SDN Controller	NBI Transport SDN Controller	Dissagregated ROADM	Router and line card configuration	Operations and notification of network elements
Data Model	YANG	YANG	YANG	YANG	Protobuf
Complexity	+	++	++	++	+
SDO	ONF, OIF	IETF	MSA	MSA	-

Standards and Open Source

TeraFlow SDN

Standards Outcome/Benefits

- Reference architectures
- Functional requirements
- Interface requirements
- Information models
- Data Models
- Protocols



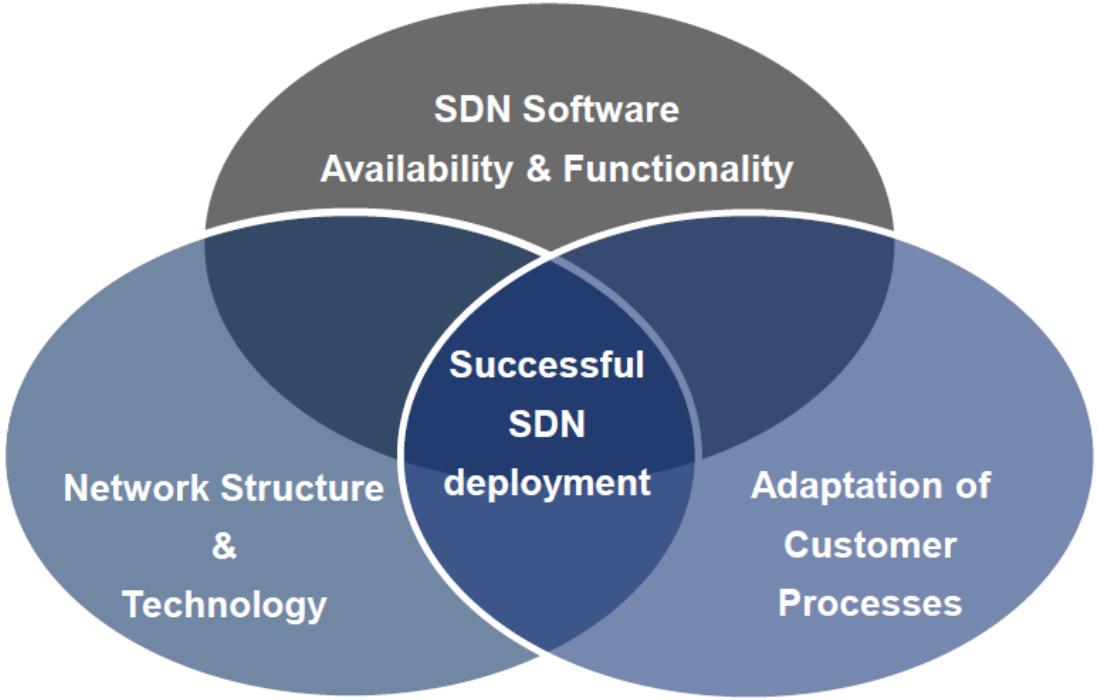
Open Source Outcome/Benefit

- Source code
- Reference implementation
- De-Facto API's
- Inter-operability testing
- Explore new features
- It's all about the community
- Upstream vs Integration

PoC Outcome/Benefits

- Technology Exploration
- Feasibility Check
- Create knowhow
- PR

Transport SDN Benefits and Challenges



- **Benefit**: Completely automated, programmable, integrated and flexible network – leveraging the installed base in an optimized manner.
- **Technical Challenges**:
  - agree on standardized architectures and abstraction/ virtualization models
  - performance of centralized systems & OF
- **Commercialization Challenges**:
  - Open Source business models
  - New business models leveraging SDN
- **Organizational Challenges**:
  - Adapt deep rooted processes across traditional silos & boundaries to leverage SDN flexibility
- **Deployment Challenges**:
  - Carrier grade SDN systems for field deployments
  - Maturity of SDN network technologies for green field deployments as well as integration of legacy networks

At the end of the day…

- A **satisfaction survey** will be circulated
  - Please take 2 minutes to reply and leave us comments
  - Your feedback is precious!
- **Certificates of participation** will be granted
  - Make sure you are properly registered, and we know where to send yours!
- And .. if you liked the **TeraFlowSDN** experience..
  - **Join us!** Participation is open to ETSI members, non-members, individual contributors and users… Learn how to join.

TeraFlow SDN by ETSI

Thank You!

References

RFC6020, YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF), https://tools.ietf.org/html/rfc6020

RFC6241, Network Configuration Protocol (NETCONF), https://tools.ietf.org/html/rfc6241

Open ROADM Overview, https://0201.nccdn.net/4_2/000/000/05e/0e7/Open-ROADM-whitepaper-v2_2.pdf

RFC8040, RESTCONF Protocol, https://tools.ietf.org/html/rfc8040

Transport API (TAPI) 2.0 Overview, https://wiki.opennetworking.org/display/OTCC/TAPI+Overview

gRPC Basics – Python, https://grpc.io/docs/tutorials/basic/python.html

OpenConfig FAQ for operators, http://www.openconfig.net/docs/faq-for-operators/

ONF's P4 Language Tutorial, https://opennetworking.org/wp-content/uploads/2020/12/P4_D2_East_2018_01_basics.pdf

ONF's Next generation SDN tutorial, https://github.com/opennetworkinglab/ngsdn-tutorial

This SC contains slides from previous OFC 2018 SC449: Hands-on: An introduction to Writing Transport SDN Applications by Ricard Vilalta (CTTC) and Karthik Sethuraman/Yuta Higuchi (NEC) and OFC 2018 SC448: Software Defined Networking for Optical Networks: a Practical Introduction by Ramon Casellas (CTTC).