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

© ETSI CC-BY-4.0

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

© ETSI CC-BY-4.0

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 CC-BY-4.0

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

© ETSI CC-BY-4.0

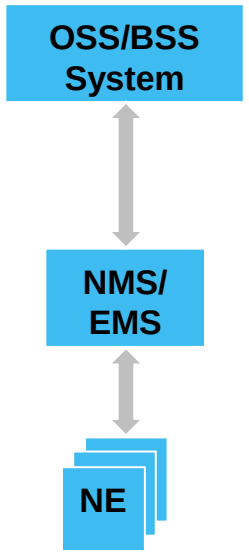
Motivation

© ETSI CC-BY-4.0

Why is SDN different from traditional Architectures?

TeraFlow SDN

Traditional Architecture



The difference is NOT:

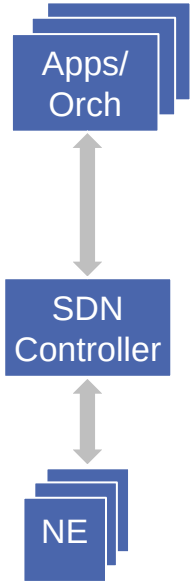
- Standardized Management Interfaces
- Standardized Architecture
- Partly not the Open Interfaces
- Partly not even the use cases

The difference is:

A new way of thinking

- Application focused
- Application takes control over the service
- Open Source based development
- Simplification through Abstraction & Virtualization

SDN Architecture



© ETSI CC-BY-4.0

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:

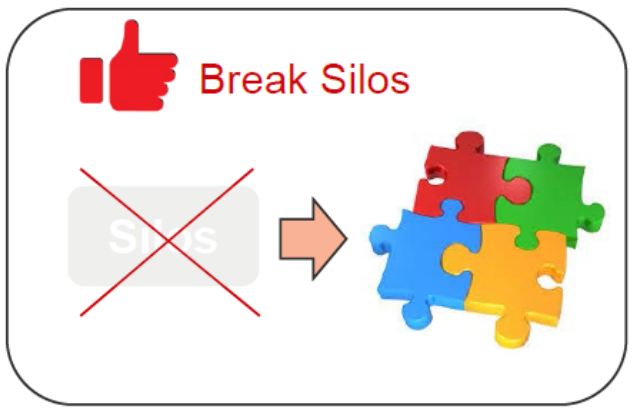
- Integrated E2E / Multi-layer service creation
- Automatic reaction on errors or any changes

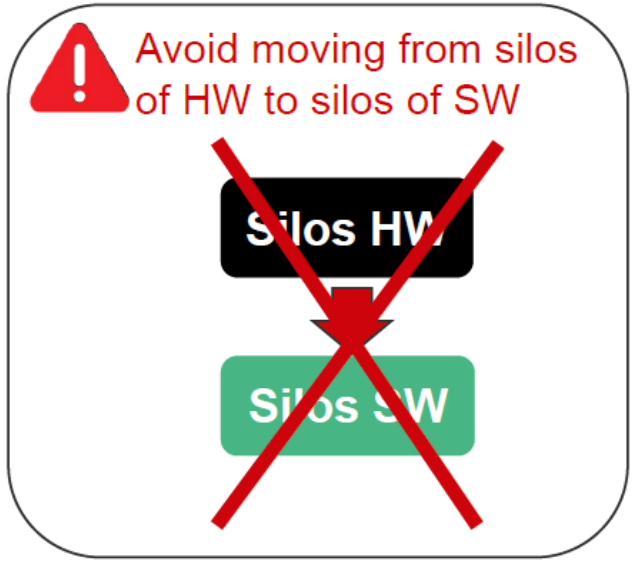
Financial Benefits:

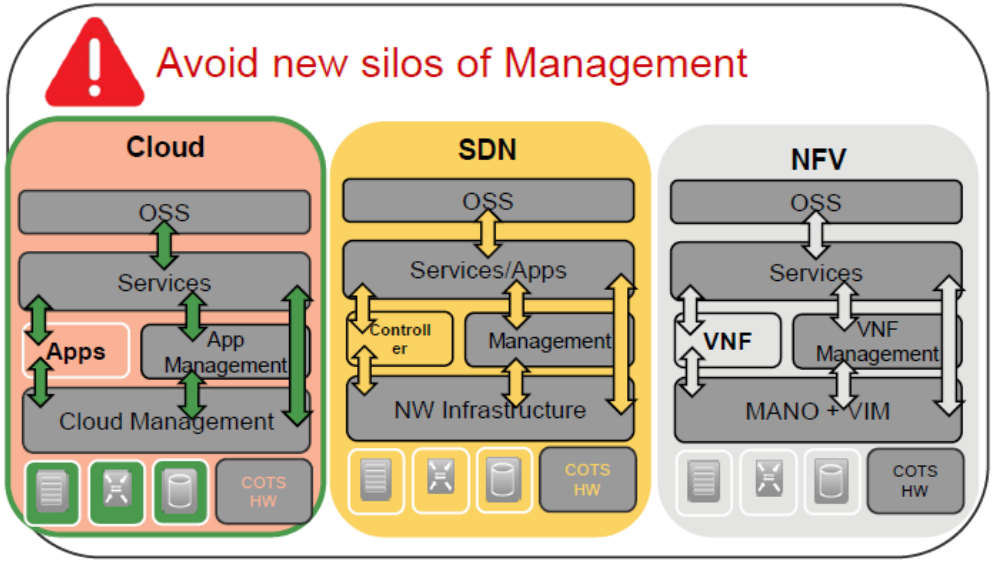
- Opex: efficient service setup
- Capex: fast ROI / hardware utilization
- New revenue opportunities

© ETSI CC-BY-4.0

Keys to success

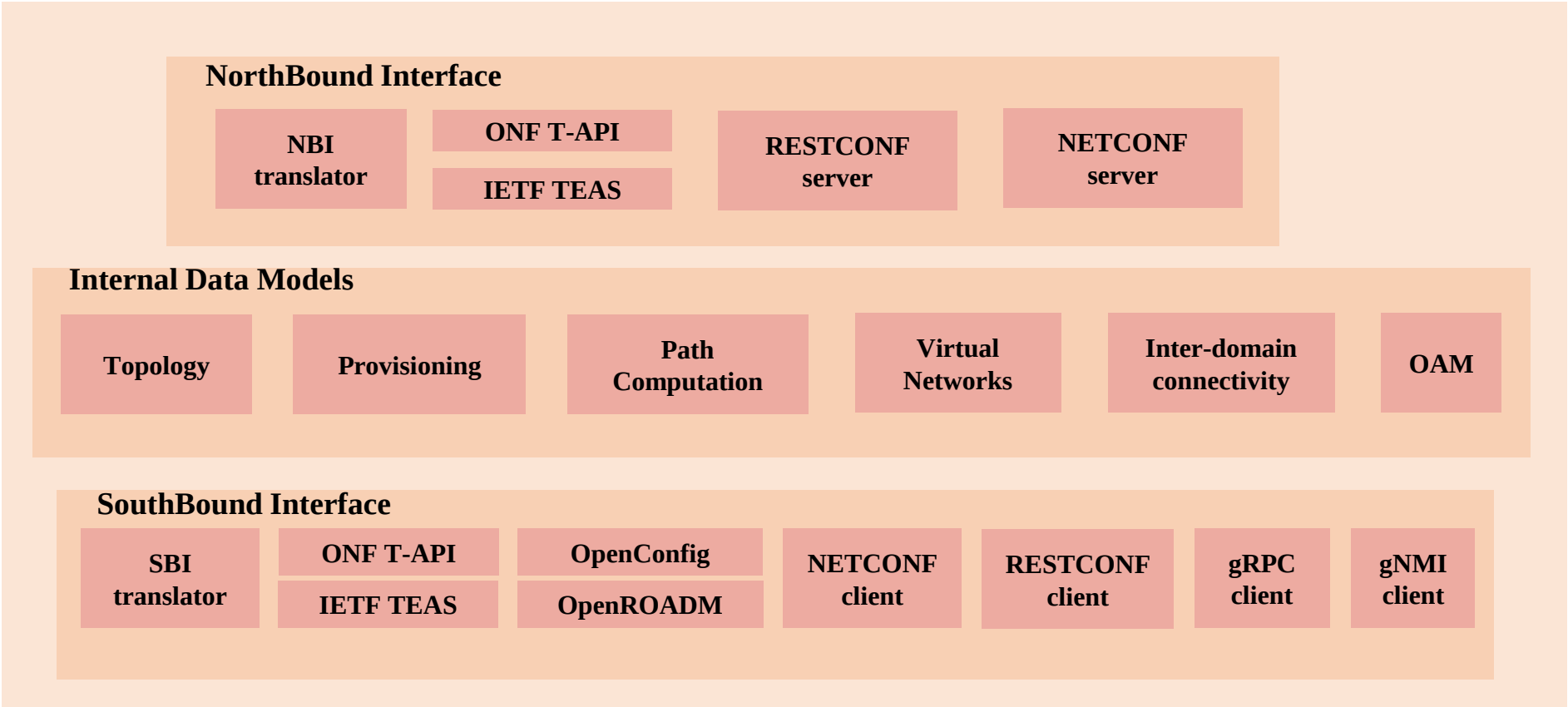






© ETSI CC-BY-4.0

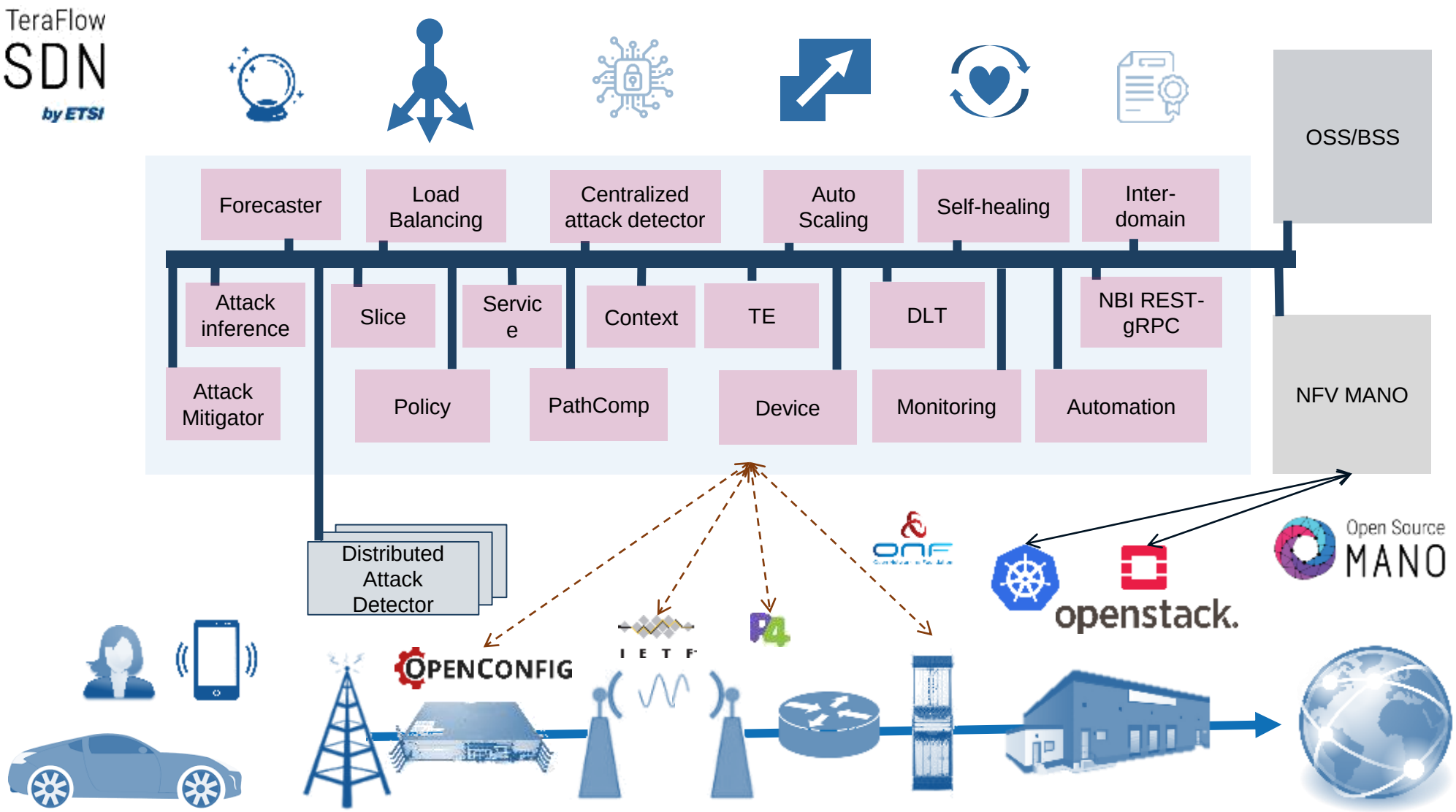
A multi-SDO SDN controller architecture



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

© ETSI CC-BY-4.0

From theory to practice… Welcome ETSI TeraFlowSDN



© ETSI CC-BY-4.0

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! ☺**

© ETSI CC-BY-4.0

YANG/NETCONF

© ETSI CC-BY-4.0

Unified Information and Data Modeling (1)

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.

© ETSI CC-BY-4.0

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)

© ETSI CC-BY-4.0

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.

© ETSI CC-BY-4.0

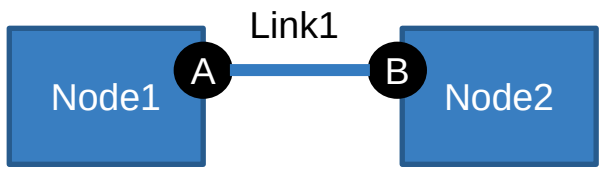
The YANG Language II

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.

© ETSI CC-BY-4.0

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.

© ETSI CC-BY-4.0

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;
  }
```

© ETSI CC-BY-4.0

[Tool] pyang

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>
```

© ETSI CC-BY-4.0

UML diagram

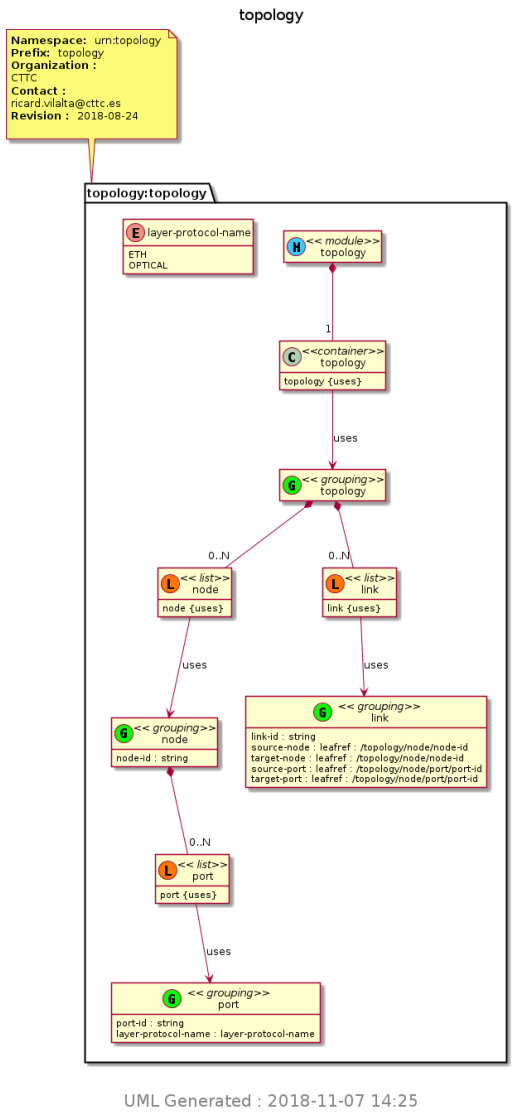
PlantUML is an opensource tool to create UML diagrams

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
```



© ETSI CC-BY-4.0

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
```

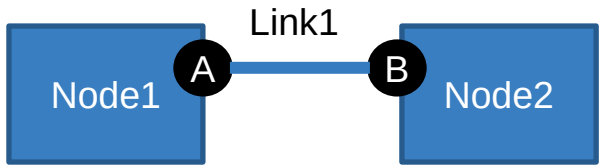
Source: https://github.com/robshakir/pyangbind

© ETSI CC-BY-4.0

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))
```

© ETSI CC-BY-4.0

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>
```

© ETSI CC-BY-4.0

Topology JSON


```
{
    "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"
            }
        }
    }
}
```


© ETSI CC-BY-4.0

Exercise: Create a connection data model

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

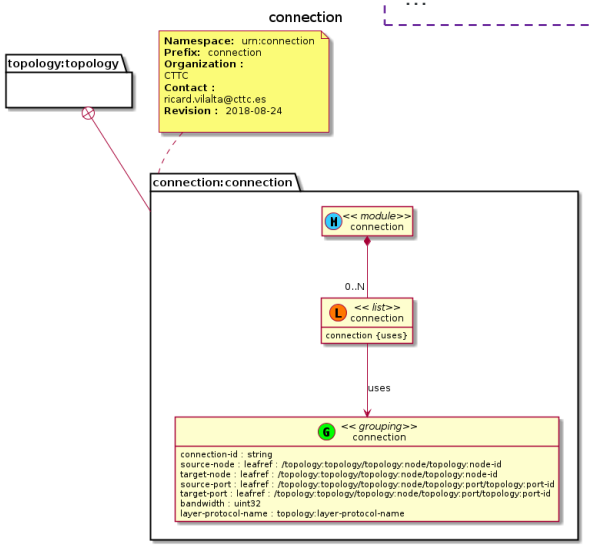
Create pyangbind bindings

Create xml using bindings

© ETSI CC-BY-4.0

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 "";
  }
…
```



UML Generated : 2018-11-08 09:13

```
…
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;
  }
}
```

© ETSI CC-BY-4.0

Solution: connection.py

```
$ 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>
```

© ETSI CC-BY-4.0

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.)

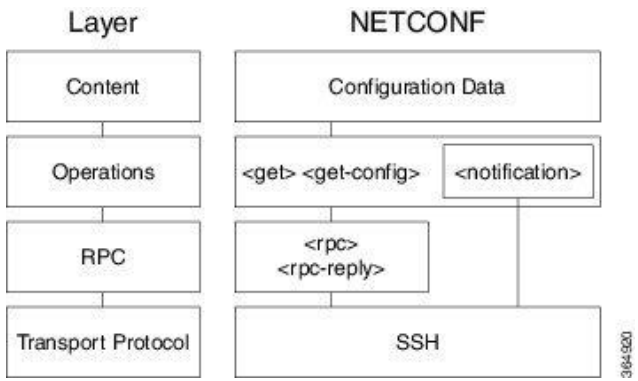
Enables remote access to a device, and provides the set of rules by which multiple clients may access and modify a datastore within a NETCONF server (e.g., device).

- 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.

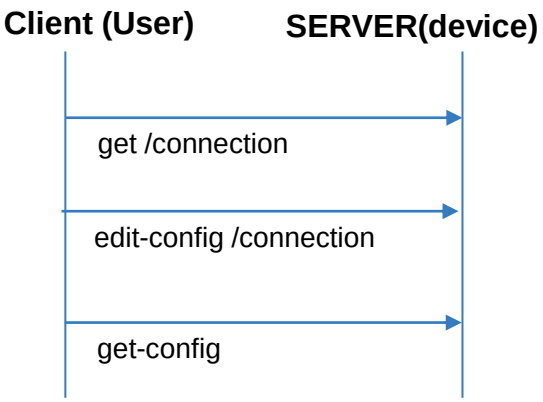


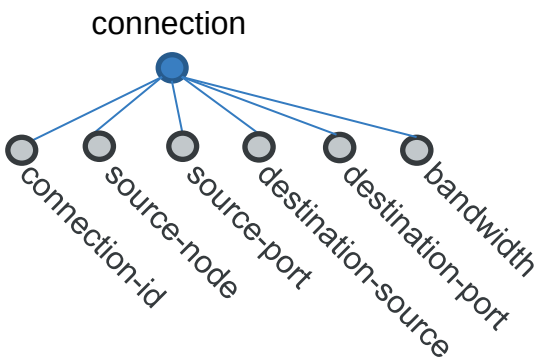
© ETSI CC-BY-4.0

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
```

© ETSI CC-BY-4.0

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.

© ETSI CC-BY-4.0

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()
```

© ETSI CC-BY-4.0

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

© ETSI CC-BY-4.0

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()
```

© ETSI CC-BY-4.0

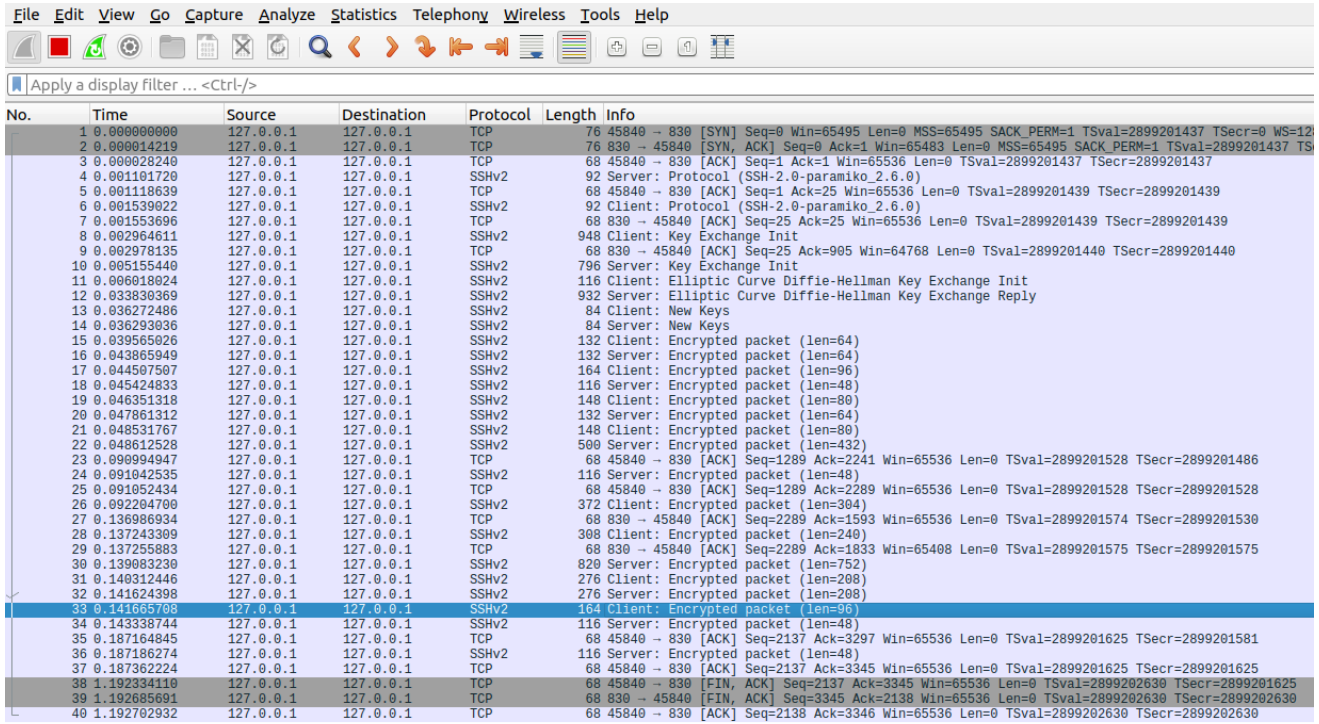
Run NETCONF example

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”



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	76	45840 → 830 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2899201437 TSecr=0 WS=12
2	0.000014219	127.0.0.1	127.0.0.1	TCP	76	830 → 45840 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=2899201437 TS
3	0.000028240	127.0.0.1	127.0.0.1	TCP	68	45840 → 830 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2899201437 TSecr=2899201437
4	0.001101720	127.0.0.1	127.0.0.1	SSHv2	92	Server: Protocol (SSH-2.0-paramiko_2.6.0)
5	0.001118639	127.0.0.1	127.0.0.1	TCP	68	45840 → 830 [ACK] Seq=1 Ack=25 Win=65536 Len=0 TSval=2899201439 TSecr=2899201439
6	0.001539022	127.0.0.1	127.0.0.1	SSHv2	92	Client: Protocol (SSH-2.0-paramiko_2.6.0)
7	0.001553696	127.0.0.1	127.0.0.1	TCP	68	830 → 45840 [ACK] Seq=25 Ack=25 Win=65536 Len=0 TSval=2899201439 TSecr=2899201439
8	0.002964611	127.0.0.1	127.0.0.1	SSHv2	948	Client: Key Exchange Init
9	0.002978135	127.0.0.1	127.0.0.1	TCP	68	830 → 45840 [ACK] Seq=25 Ack=905 Win=64768 Len=0 TSval=2899201440 TSecr=2899201440
10	0.005155440	127.0.0.1	127.0.0.1	SSHv2	796	Server: Key Exchange Init
11	0.006018024	127.0.0.1	127.0.0.1	SSHv2	116	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
12	0.033830369	127.0.0.1	127.0.0.1	SSHv2	932	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply
13	0.036272486	127.0.0.1	127.0.0.1	SSHv2	84	Client: New Keys
14	0.036293036	127.0.0.1	127.0.0.1	SSHv2	84	Server: New Keys
15	0.039565026	127.0.0.1	127.0.0.1	SSHv2	132	Client: Encrypted packet (len=64)
16	0.043865949	127.0.0.1	127.0.0.1	SSHv2	132	Server: Encrypted packet (len=64)
17	0.044507507	127.0.0.1	127.0.0.1	SSHv2	164	Client: Encrypted packet (len=96)
18	0.045424833	127.0.0.1	127.0.0.1	SSHv2	116	Server: Encrypted packet (len=48)
19	0.046351318	127.0.0.1	127.0.0.1	SSHv2	148	Client: Encrypted packet (len=80)
20	0.047861312	127.0.0.1	127.0.0.1	SSHv2	132	Server: Encrypted packet (len=64)
21	0.048531767	127.0.0.1	127.0.0.1	SSHv2	148	Client: Encrypted packet (len=80)
22	0.048612528	127.0.0.1	127.0.0.1	SSHv2	500	Server: Encrypted packet (len=432)
23	0.090994947	127.0.0.1	127.0.0.1	TCP	68	45840 → 830 [ACK] Seq=1289 Ack=2241 Win=65536 Len=0 TSval=2899201528 TSecr=2899201486
24	0.091042535	127.0.0.1	127.0.0.1	SSHv2	116	Server: Encrypted packet (len=48)
25	0.091052434	127.0.0.1	127.0.0.1	TCP	68	45840 → 830 [ACK] Seq=1289 Ack=2289 Win=65536 Len=0 TSval=2899201528 TSecr=2899201528
26	0.092204700	127.0.0.1	127.0.0.1	SSHv2	372	Client: Encrypted packet (len=304)
27	0.136986934	127.0.0.1	127.0.0.1	TCP	68	830 → 45840 [ACK] Seq=2289 Ack=1593 Win=65536 Len=0 TSval=2899201574 TSecr=2899201530
28	0.137243309	127.0.0.1	127.0.0.1	SSHv2	308	Client: Encrypted packet (len=240)
29	0.137255883	127.0.0.1	127.0.0.1	TCP	68	830 → 45840 [ACK] Seq=2289 Ack=1833 Win=65408 Len=0 TSval=2899201575 TSecr=2899201575
30	0.139083230	127.0.0.1	127.0.0.1	SSHv2	820	Server: Encrypted packet (len=752)
31	0.140312446	127.0.0.1	127.0.0.1	SSHv2	276	Client: Encrypted packet (len=208)
32	0.141624398	127.0.0.1	127.0.0.1	SSHv2	276	Server: Encrypted packet (len=208)
33	0.141665708	127.0.0.1	127.0.0.1	SSHv2	164	Client: Encrypted packet (len=96)
34	0.143338744	127.0.0.1	127.0.0.1	SSHv2	116	Server: Encrypted packet (len=48)
35	0.187164845	127.0.0.1	127.0.0.1	TCP	68	45840 → 830 [ACK] Seq=2137 Ack=3297 Win=65536 Len=0 TSval=2899201625 TSecr=2899201581
36	0.187186274	127.0.0.1	127.0.0.1	SSHv2	116	Server: Encrypted packet (len=48)
37	0.187362224	127.0.0.1	127.0.0.1	TCP	68	45840 → 830 [ACK] Seq=2137 Ack=3345 Win=65536 Len=0 TSval=2899201625 TSecr=2899201625
38	1.192334110	127.0.0.1	127.0.0.1	TCP	68	45840 → 830 [FIN, ACK] Seq=2137 Ack=3345 Win=65536 Len=0 TSval=2899202630 TSecr=2899201625
39	1.192685691	127.0.0.1	127.0.0.1	TCP	68	830 → 45840 [FIN, ACK] Seq=3345 Ack=2138 Win=65536 Len=0 TSval=2899202630 TSecr=2899202630
40	1.192702932	127.0.0.1	127.0.0.1	TCP	68	45840 → 830 [ACK] Seq=2138 Ack=3346 Win=65536 Len=0 TSval=2899202630 TSecr=2899202630

© ETSI CC-BY-4.0

Exercise: NETCONF edit-config

Include connection.yang

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

Server adds new connection

Client list connection

Time: 15min

© ETSI CC-BY-4.0

NETCONF server edit-config: server_topology_connection.py

```
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
```

© ETSI CC-BY-4.0

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
```

© ETSI CC-BY-4.0

CONFD Tutorial

© ETSI CC-BY-4.0

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.

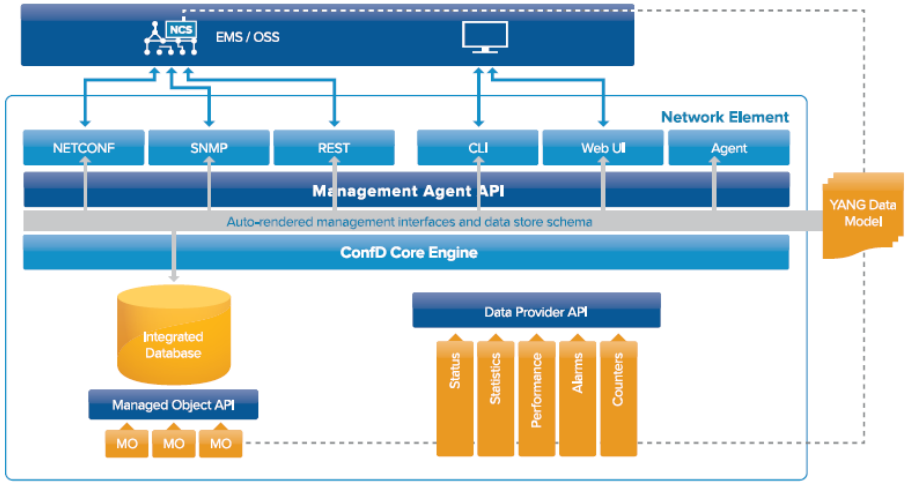


Figure 1: ConfD block diagram

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

© ETSI CC-BY-4.0

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/
```

Data-Model Compilation

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

Start ConfD

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

Use ConfD-client

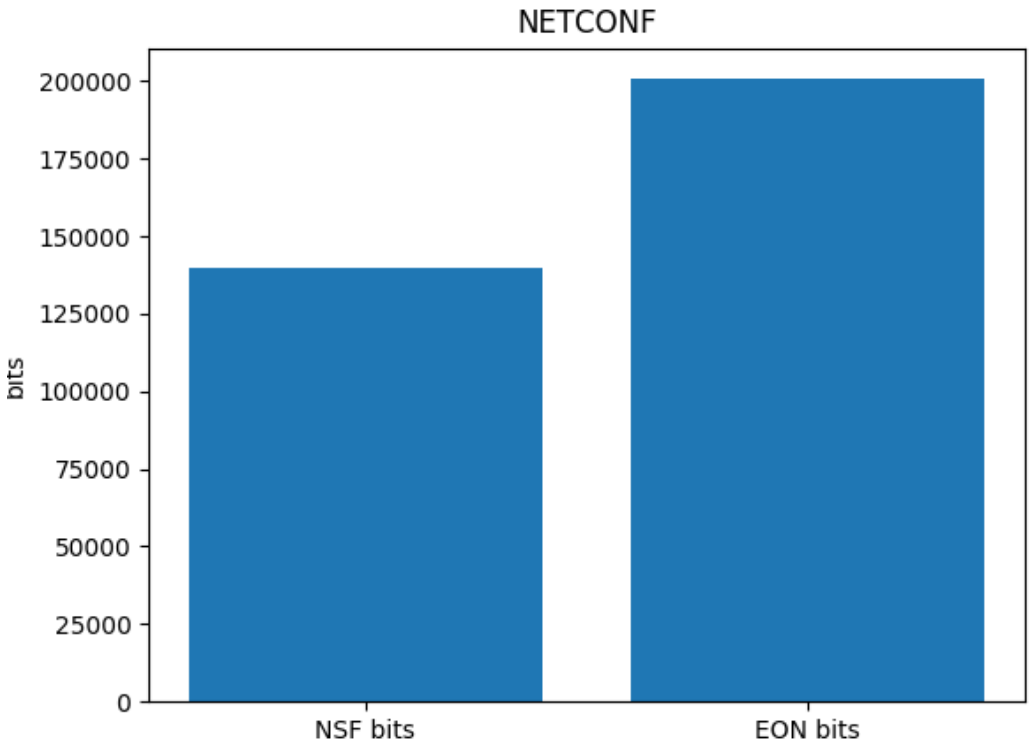
```
$ ./confd_cli
➢ conf
➢ topology node node1
➢ exit
➢ commit
➢ exit
➢ exit
```

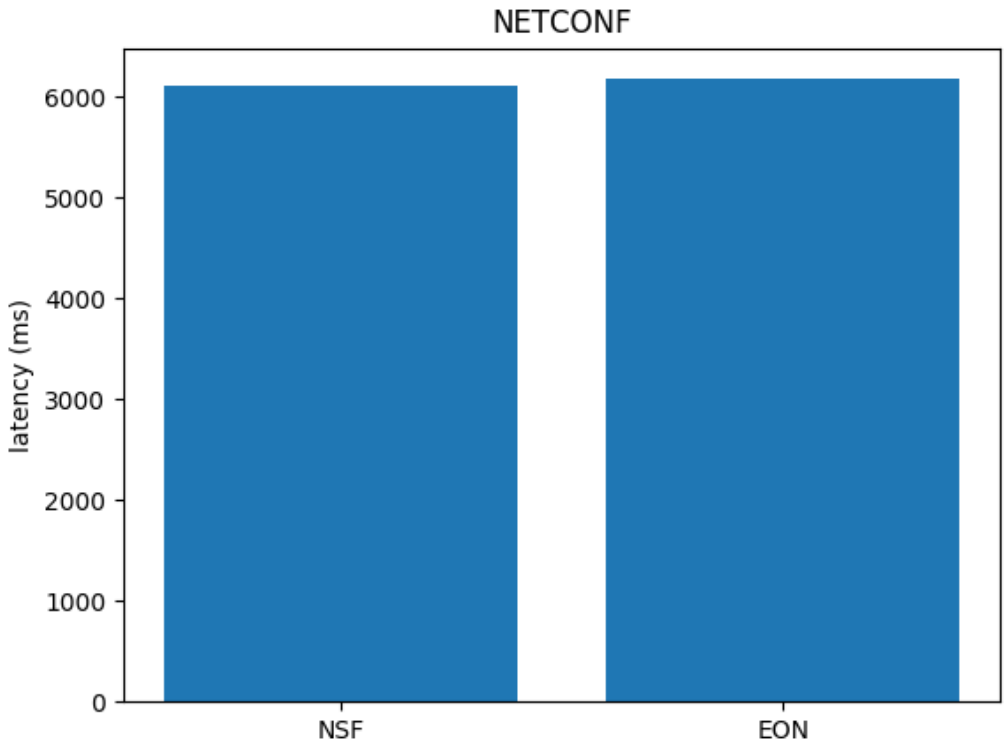
```
$ ./confd_cli
➢ conf
➢ show full-configuration
➢ exit
➢ exit
```

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

© ETSI CC-BY-4.0

Example NETCONF results





© ETSI CC-BY-4.0

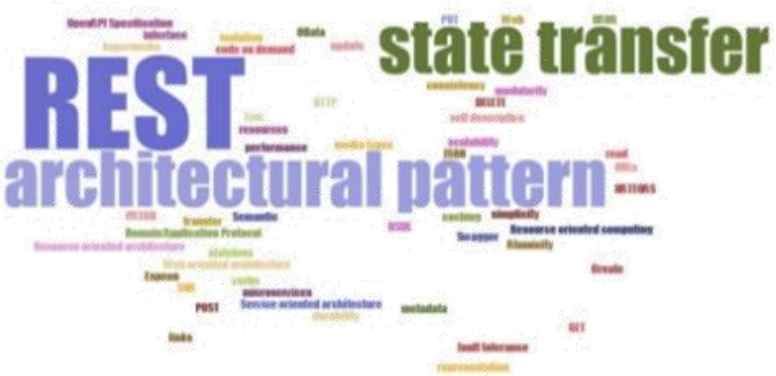
ONF Transport API 2.1

© ETSI CC-BY-4.0

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.



© ETSI CC-BY-4.0

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” : [ … ]
…
}
```

© ETSI CC-BY-4.0

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)

© ETSI CC-BY-4.0

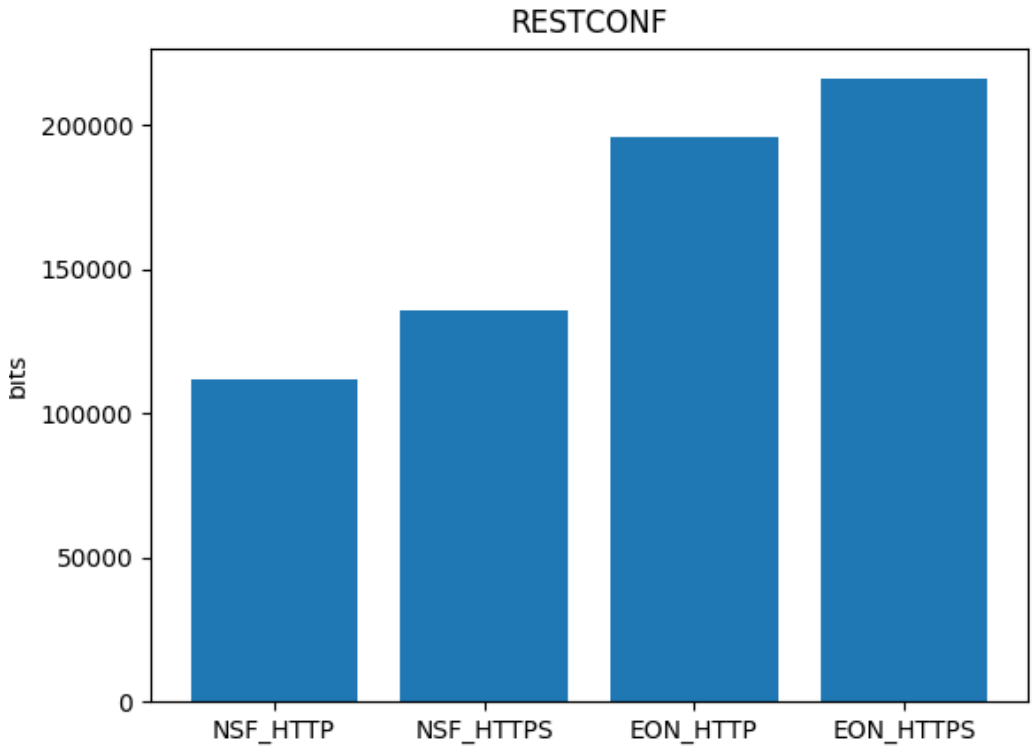
RESTCONF HTTP tree

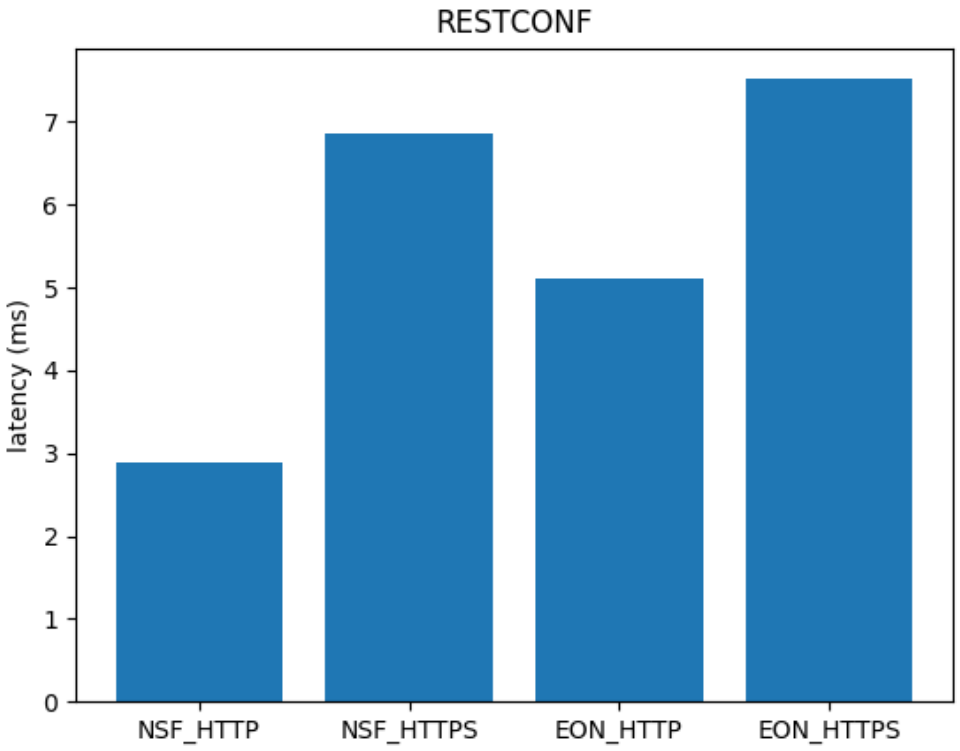
RESTCONF is a REST-like protocol that provides a HTTP-based API to access the data, modeled by YANG. The REST-like operations are used to access the hierarchical data within a datastore. The information modeled in YANG is structured in the following tree:

- /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”

© ETSI CC-BY-4.0

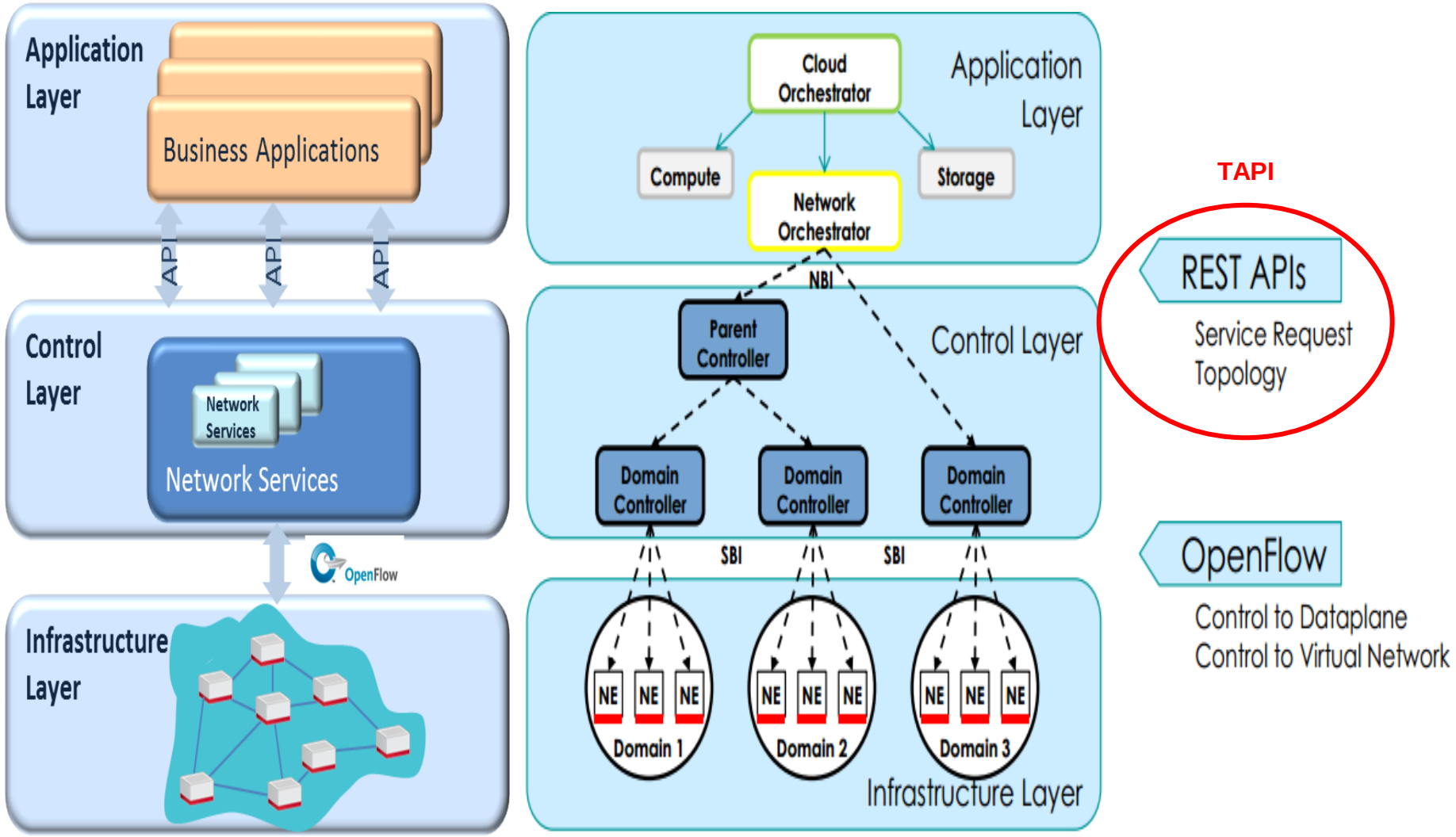
Example RESTCONF comparison





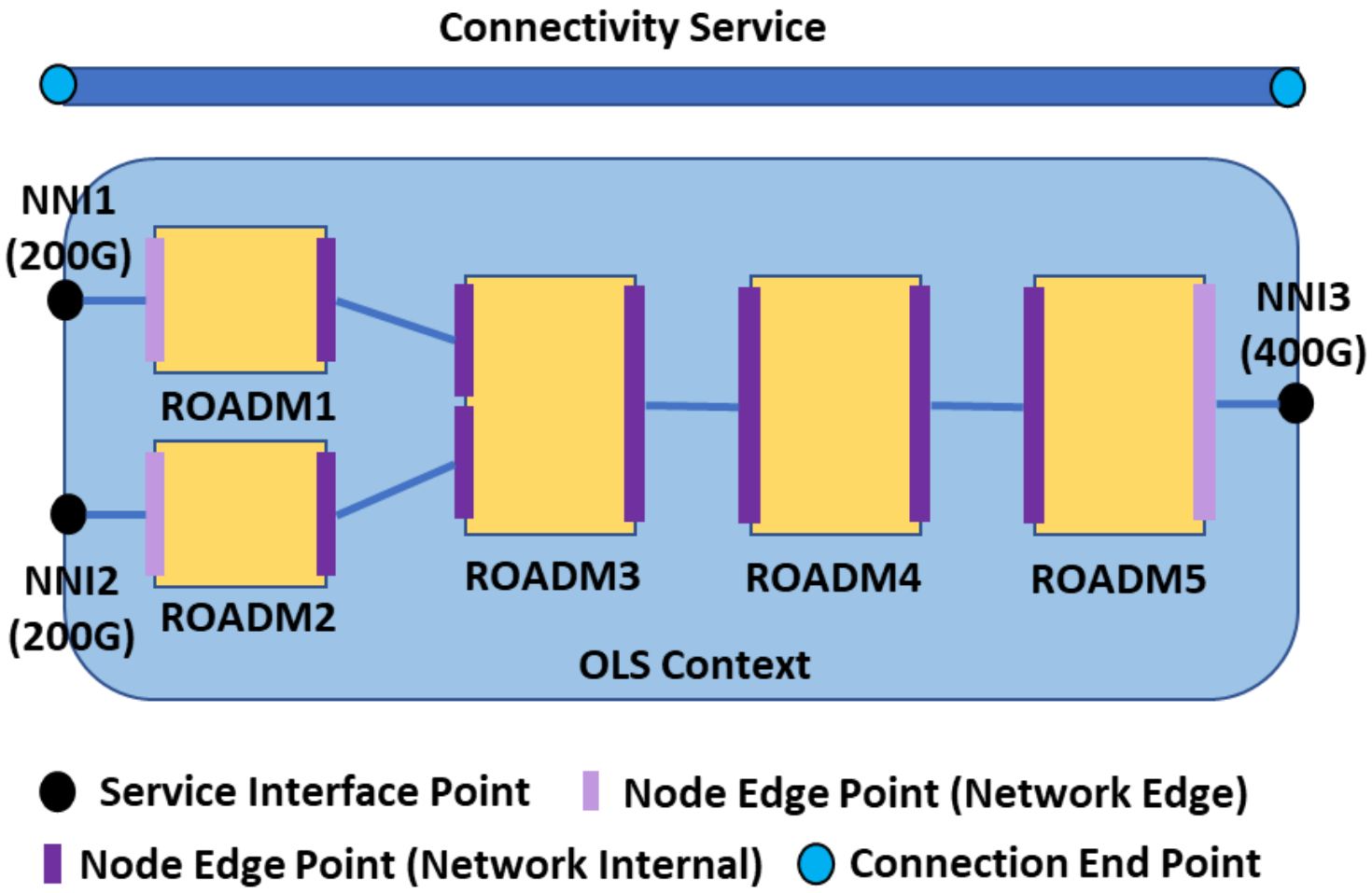
© ETSI CC-BY-4.0

Transport SDN controller



© ETSI CC-BY-4.0

TAPI Basic concepts



© ETSI CC-BY-4.0

TAPI Context, Topology & Connectivity Overview

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

© ETSI CC-BY-4.0

Launch/Run TAPI Reference Implementation

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/
```

© ETSI CC-BY-4.0

TAPI: Retrieve Context

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

Response:

```
{   "uuid" : "ols-topo",
    "service-interface-point" : [
        …
    ],
    "topology-context" : {
        "topology" : […],
        …
    },
    "connectivity-context" : [
        "connection" : […],
        "connectivity-service" : […]
    ]
}
```

Proper TAPI implementations should use UUID format. An example below: f81d4fae-7edc-11d0-a765-00a0c91e6bf6

TAPI Context is a Container for all ServiceInterfacePoints, Topologies, ConnectivityServices, Connections, etc data.

© ETSI CC-BY-4.0

TAPI: Retrieve Service Interface Point Details

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

© ETSI CC-BY-4.0

TAPI: Retrieve Topology

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": [ ...]
}
```

© ETSI CC-BY-4.0

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).

© ETSI CC-BY-4.0

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:

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

Node can be single or multi layer

(Optional) Abstract Node is an abstraction of a Topology

(Optional)Node represents the potential to forward data between its aggregated NodeEdgePoints

Node can also constrain forwarding across its aggregated NodeEdgePoints (not shown here)

© ETSI CC-BY-4.0

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

© ETSI CC-BY-4.0

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

© ETSI CC-BY-4.0

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"
}
```

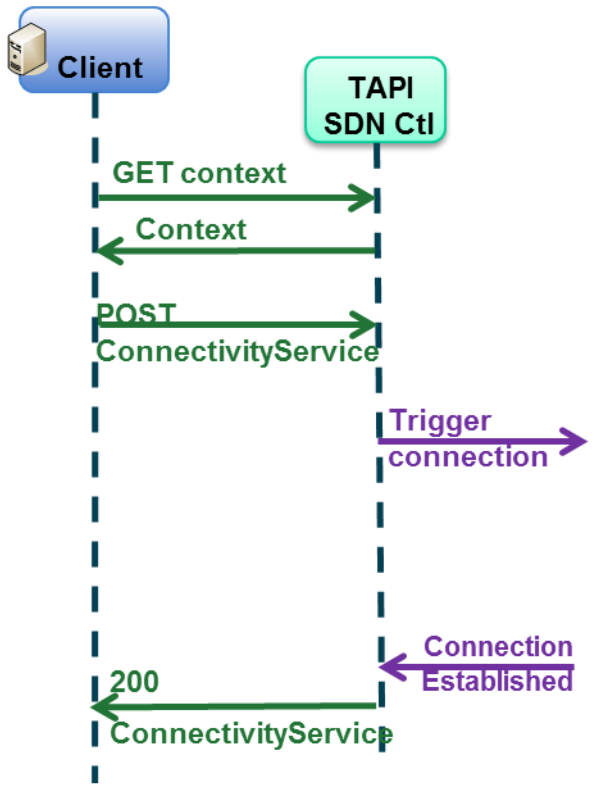

Link conveys “cost-characteristic” information

Link represents adjacency information between 2 NodeEdgePoints

© ETSI CC-BY-4.0

TAPI: Connectivity Service workflow

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



© ETSI CC-BY-4.0

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

© ETSI CC-BY-4.0

TAPI: Retrieve Connection Details

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

© ETSI CC-BY-4.0

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

© ETSI CC-BY-4.0

TAPI Optical Augments: node-edge-point

```
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
```

© ETSI CC-BY-4.0

TAPI Optical Augments: connection-end-point

```
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
```

© ETSI CC-BY-4.0

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

© ETSI CC-BY-4.0

TAPI_APP

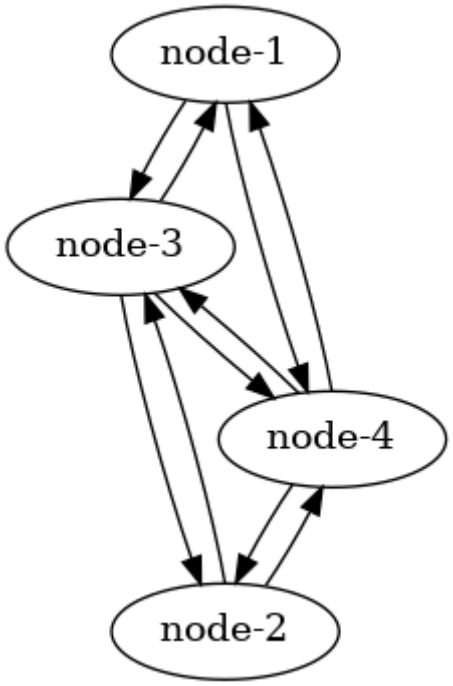
```
#!/usr/bin/python
# -*- coding: utf-8 -*-

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))
```

© ETSI CC-BY-4.0

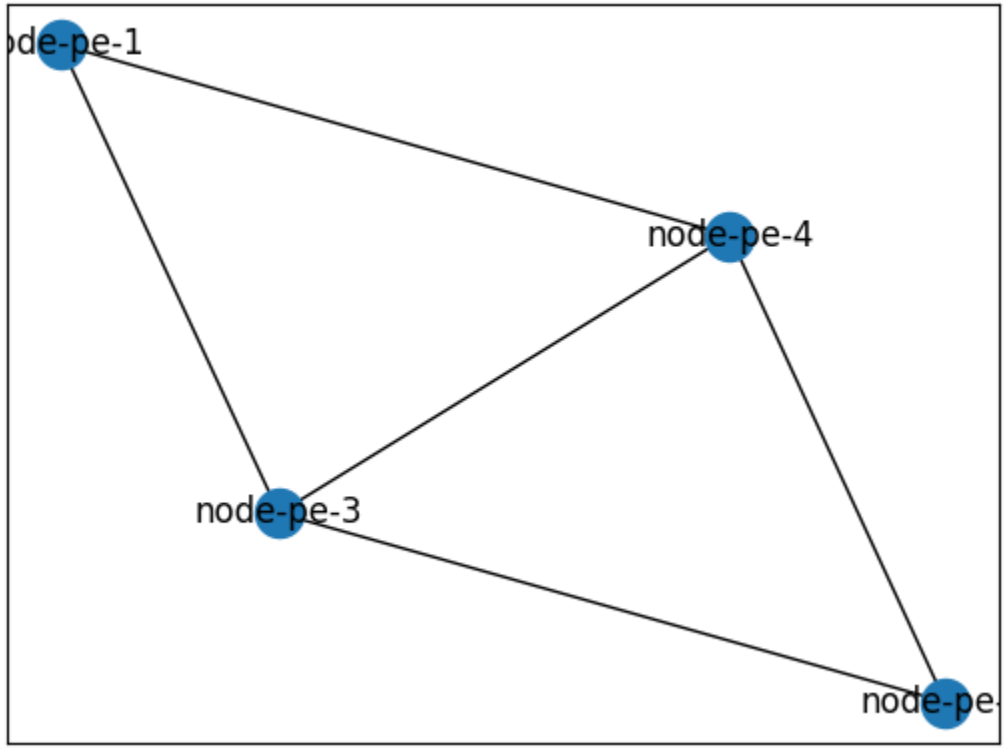
Run TAPI Application Client

Run in a terminal:

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



PyDot



MatPlotLib

© ETSI CC-BY-4.0

TeraFlow SDN by ETSI

Thank You!

© ETSI CC-BY-4.0