TeraFlow SDN by ETSI

Hackfest #2: Integrating TeraFlowSDN with ContainerLab

Ricard Vilalta (CTTC)

Lluís Gifre (CTTC)

Hackfest Materials

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

- https://www.dropbox.com/s/662xlovamanzkx1/TFS-HF2.1-VM.rar?dl=0 (~9GB)
  - Download and unzip the RAR file.
- VM user/pass: tfs/tfs123
- VM Networking:
  - Network adapter: Attached to NAT
  - VM IP address: 10.0.2.10/24 / Gateway: 10.0.2.1 / DNS: 8.8.8.8, 8.8.4.4

Hackfest Materials

TeraFlow SDN

- Inside the VM, you have all commands used in files:
  - `/home/tfs/tfs-ctl/hackfest/commands.txt`
  - `/home/tfs/tfs-ctl/hackfest/containerlab/commands.txt`
- Also available at:
  - https://labs.etsi.org/rep/tfs/controller/-/blob/feat/hackfest-r2.1/hackfest/commands.txt
  - https://labs.etsi.org/rep/tfs/controller/-/blob/feat/hackfest-r2.1/hackfest/containerlab/commands.txt
- Please update latest version from ETSI GitLab repository:
  - `git checkout feat/hackfest-r2.1`
  - `git pull`
- Use proper environment:
  - `pyenv activate 3.9.16/envs/tfs`

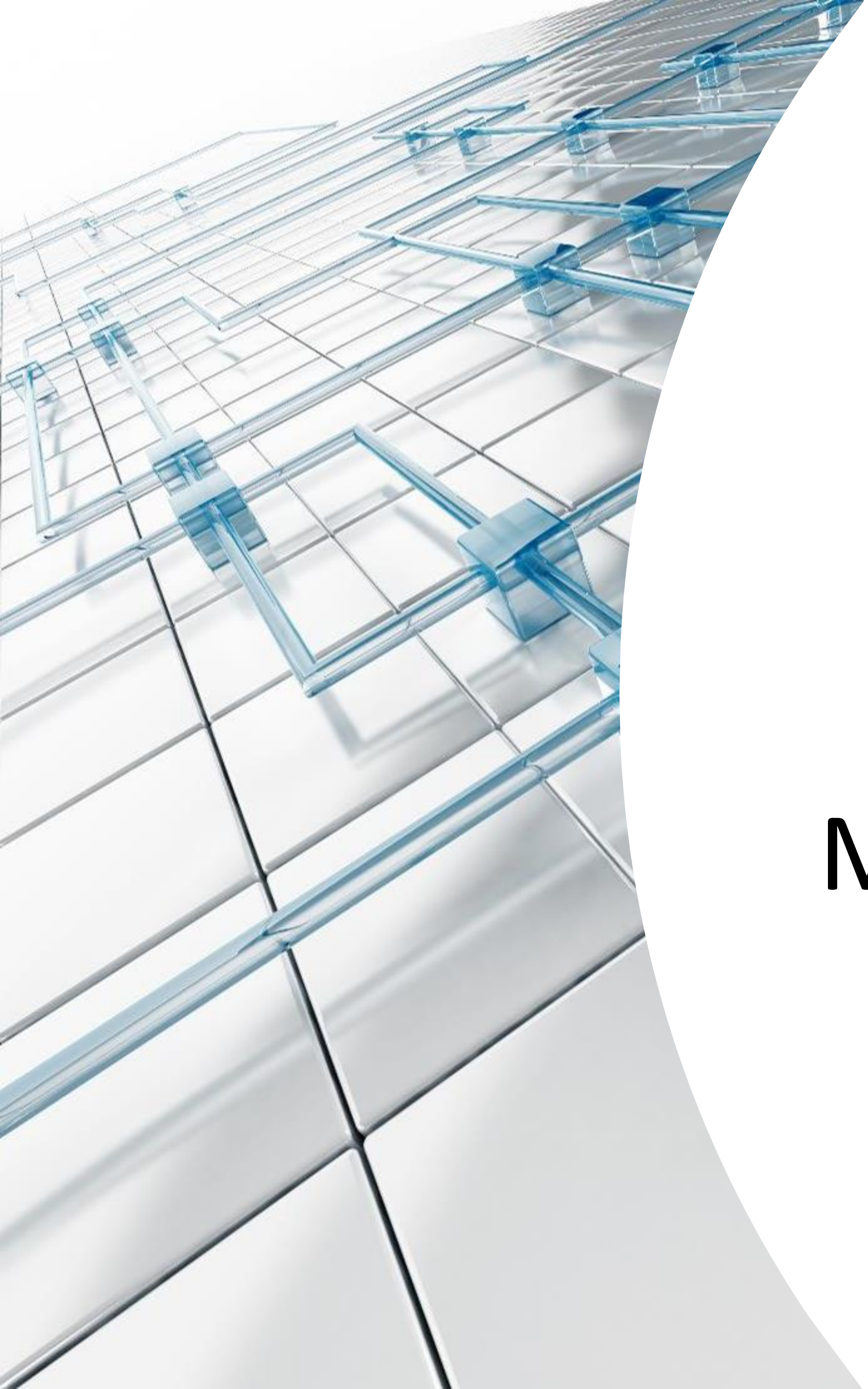
Agenda

Tuesday 20 June 2023

9:00	9:10	Welcome & Logistics (ETSI)
9:10	9:40	TeraFlowSDN 101 (TFS Chair)
9:40	10:00	Deploy and Basic use of TeraFlowSDN (TFS TSC Chair)
10:00	10:30	Introduction to ContainerLab (TFS TSC Chair)
10:30	11:00	Introduction to gNMI and OpenConfig (TFS TSC Chair)
11:00	11:30	Coffee break
11:30	11:55	Presentation of the challenges (TFS TSC Chair)
11:55	12:00	Group Picture
12:00	12:15	Form the teams
12:15	13:30	Team-Hacking Starts!
13:30	14:30	Lunch
14:30	16:30	Team-Hacking!
16:30	17:00	Coffee break
17:00	18:15	Team-Hacking!
18:15	18:30	Wrap-up day 1 (TFS TSC Chair)

Wednesday 21 June 2023

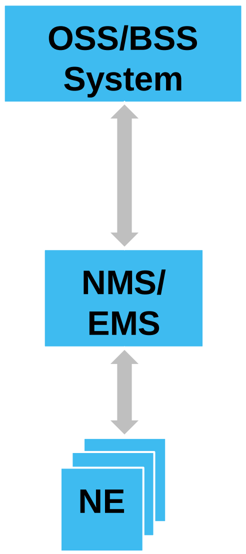
9:00	9:10	Welcome Day 2 (ETSI)
9:10	9:20	Brief discussion and progress checkpoint (TFS TSC Chair)
9:20	11:00	Team-Hacking!
11:00	11:30	Coffee break
11:30	13:30	Team-Hacking!
13:30	14:30	Lunch
14:30	16:30	Team-Hacking!
16:30	17:00	Coffee break
17:00	18:00	Final presentations: Teams will present their achievements - 7-10 teams x 5-10 min per team - Report: Progress, Working Experiment, Results, etc. - Feedback: Road blocks, Missing documentation, Report bugs, etc.
18:00	18:15	Deliberation & Winner announcement (TFS TSC Chair)
18:15	18:30	Wrap-up day 2 & end of Hackfest



Motivation

Why is SDN different from traditional Architectures?

Traditional Architecture



The difference is NOT:

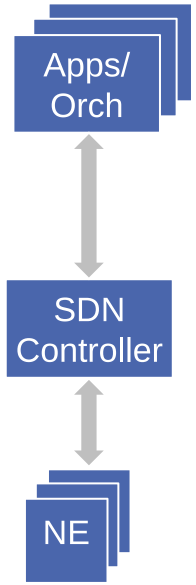
- Standardized Management Interfaces
- Standardized Architecture
- Partly not the Open Interfaces
- Partly not even the use cases

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



Why do we need SDN in Transport?

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:

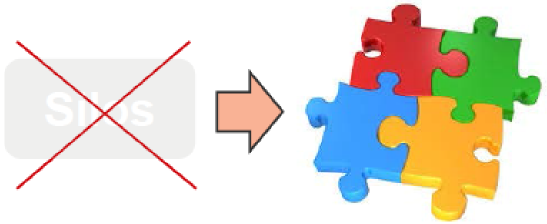
- Integrated E2E / Multi-layer service creation
- Automatic reaction on errors or any changes

Financial Benefits:

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

Keys to success

Break Silos

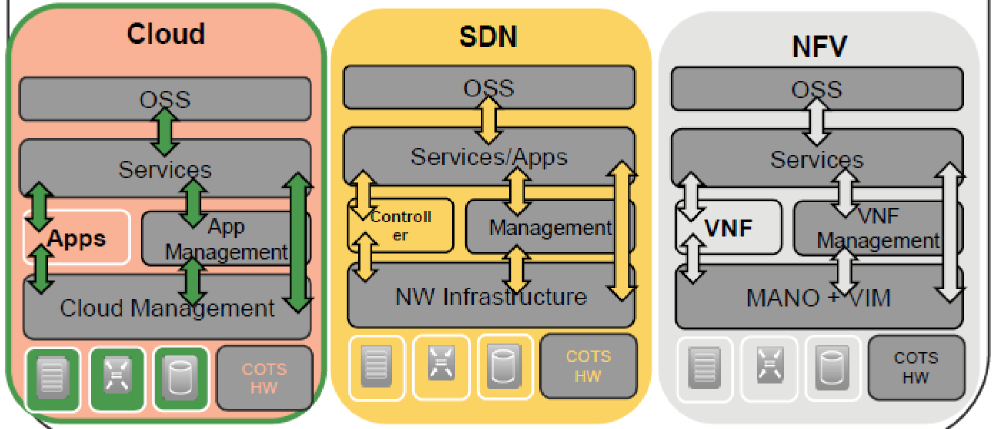


Metrics of Success

1) Multi-vendor Interoperability
2) Industry Adoption

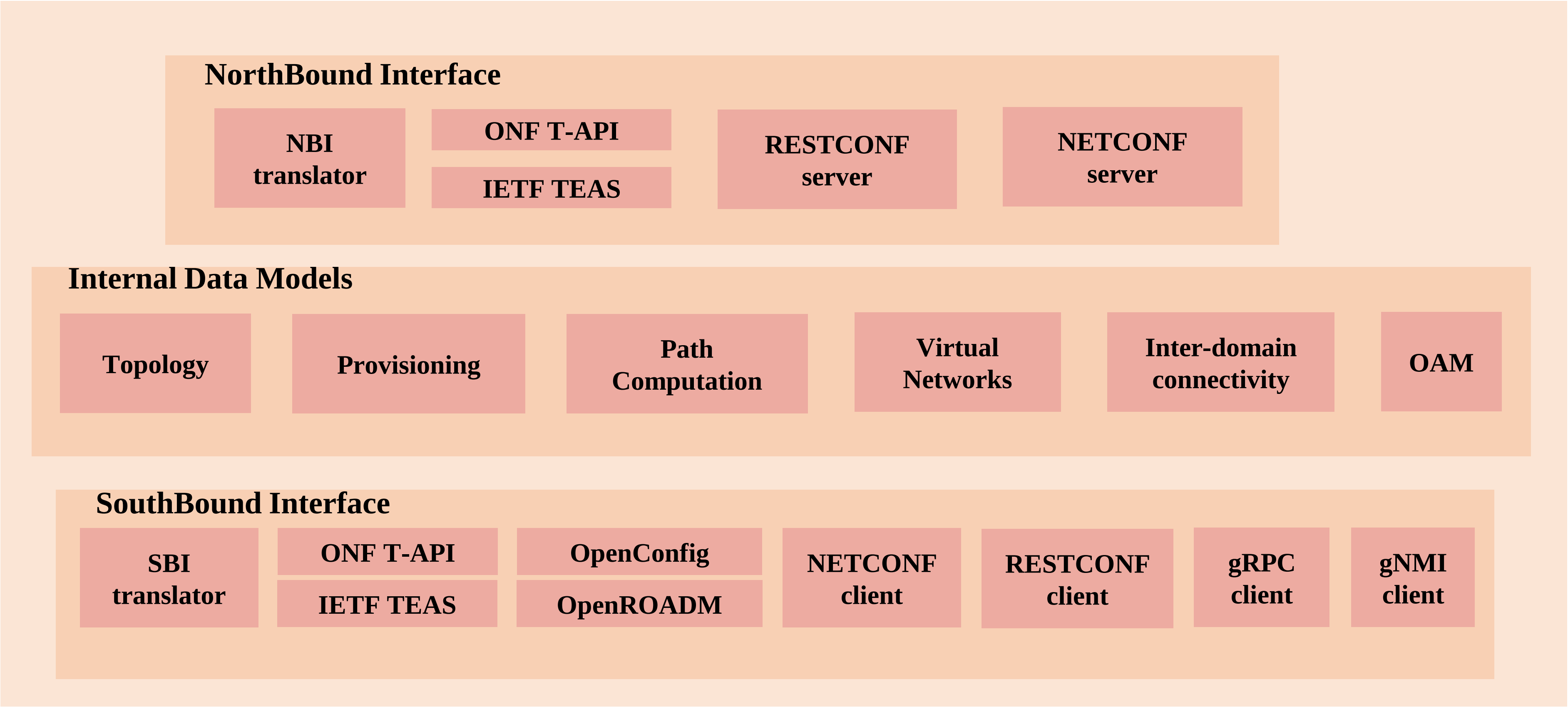
Avoid moving from silos of HW to silos of SW

Avoid new silos of Management



A multi-SDO SDN controller architecture

Multi-SDO Transport SDN Controller



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

TeraFlow SDN

ETSI TeraFlowSDN 101

Do we need YET another Transport SDN controller?

TeraFlow SDN

Cloud-native SDN controller for supporting future networks beyond 5G.

Hosted by ETSI and based on results of the European Union-funded TeraFlow 5G PPP research project.

Micro-services architecture provides key benefits:

Scalability, Self-healing, Integrity

'Toolbox' for ETSI groups working on network transformation.

Supports use cases such as autonomous networks, inter-domain, and cybersecurity.

Enables the alignment of multi-SDO goals and helping to accelerate standardization cycles.

ETSI TeraFlowSDN to serve as reference implementation for Telecom Infra Project

The source code of TeraFlowSDN is publicly available under the Apache Software Licence.

ETSI TeraFlowSDN: A growing community

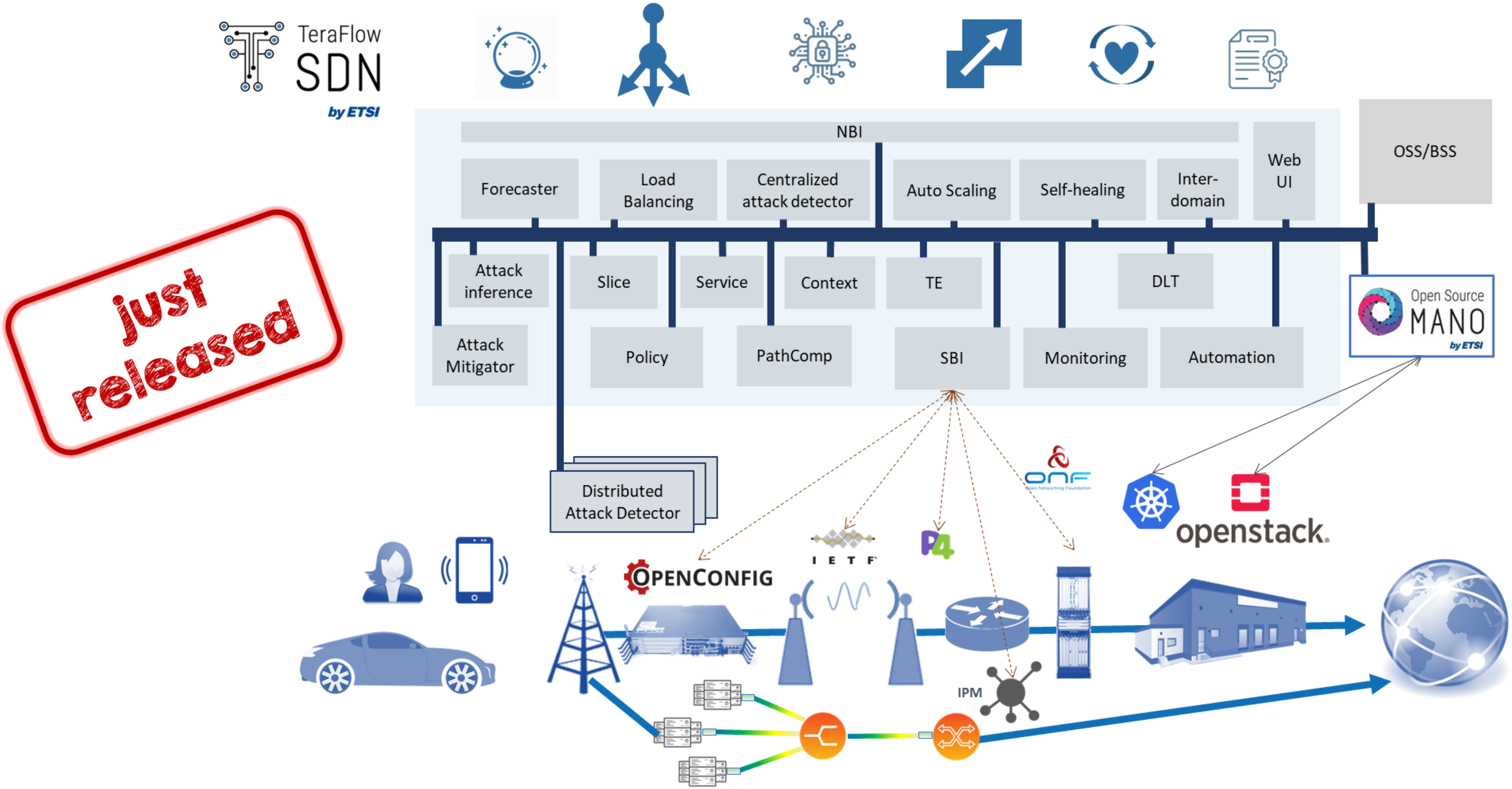
TeraFlow SDN

- Members

- Participants (Non-ETSI members)

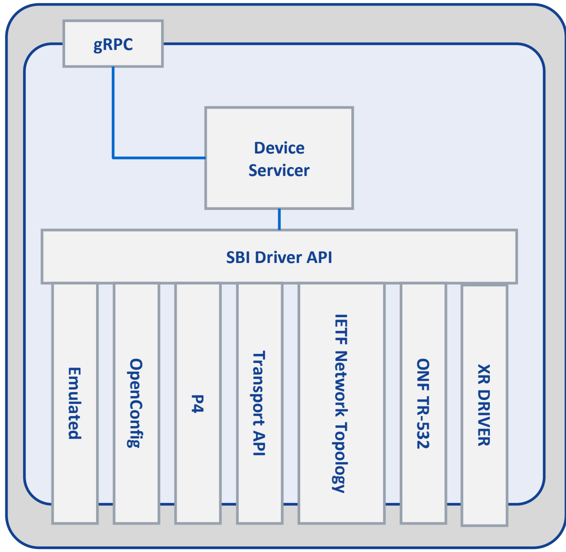
YOUR LOGO HERE

TFS Release 2 Architecture



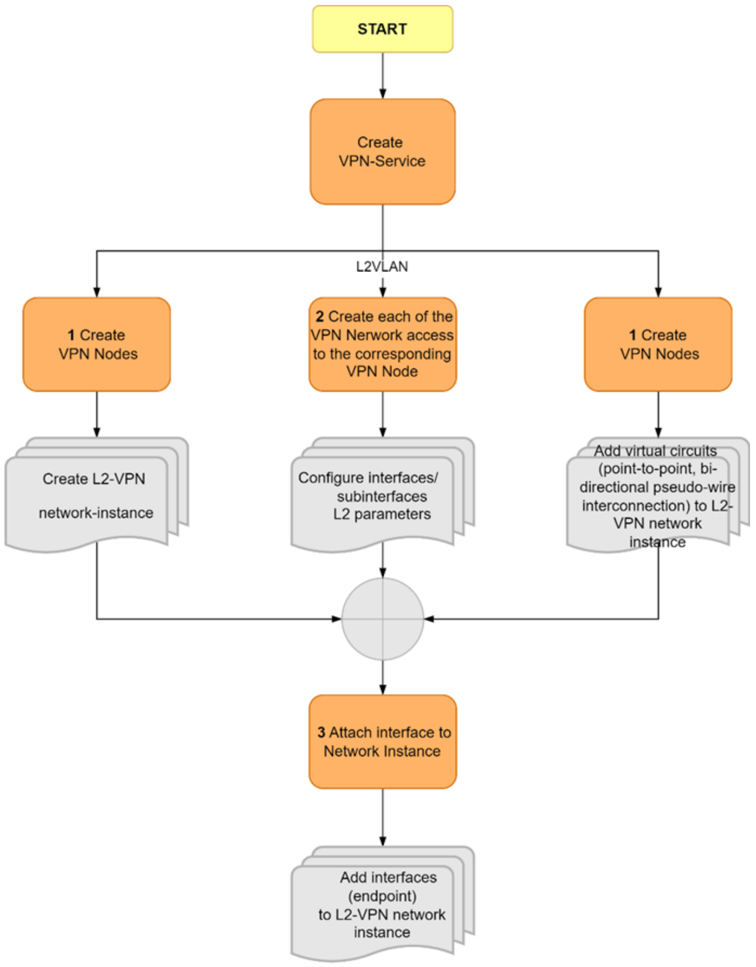
Controlled and managed network elements/domains

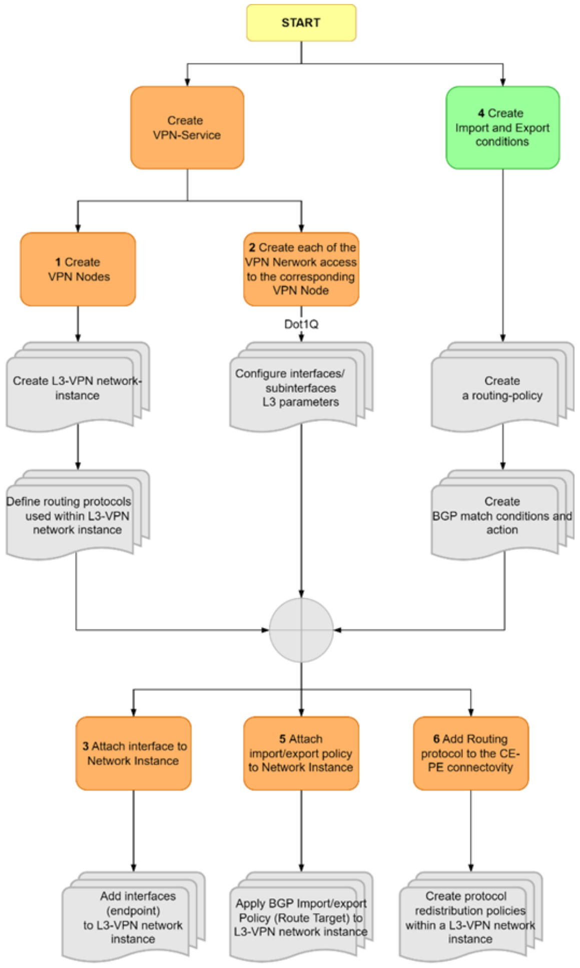
- The TeraFlowSDN controller uses its North-Bound Interface (NBI) component (previously known as Compute) to receive:
  - Layer 2 Virtual Private Network (L2VPN) requests and convert them to necessary connectivity services
  - Transport Network Slices via the Slice and Service components.
- The Service component is responsible for selecting, configuring, and deploying the requested connectivity service through the South-Bound Interface (SBI). To this end, the SBI component interacts with the network equipment through pluggable drivers. In addition, a Driver Application Programming Interface (API) has been defined to facilitate the addition of new network protocols and data models to the SBI component. TeraFlowSDN Release 2 provides extended and validated support for:
  - OpenConfig-based routers. Interaction with optical SDN controllers through the Open Networking Foundation (ONF) Transport API (TAPI).
  - Integration for microwave network elements (through the Internet Engineering Task Force - IETF - network topology YANG model).
  - Point-to-Multipoint integration of XR optical transceivers.
  - Support for P4 routers that includes loading a P4 pipeline on a given P4 switch; getting runtime information (i.e., flow tables) from the P4 switch; and pushing runtime entries into the P4 switch pipeline, thus allowing total usage of P4 switches.



Support for OpenConfig Whiteboxes

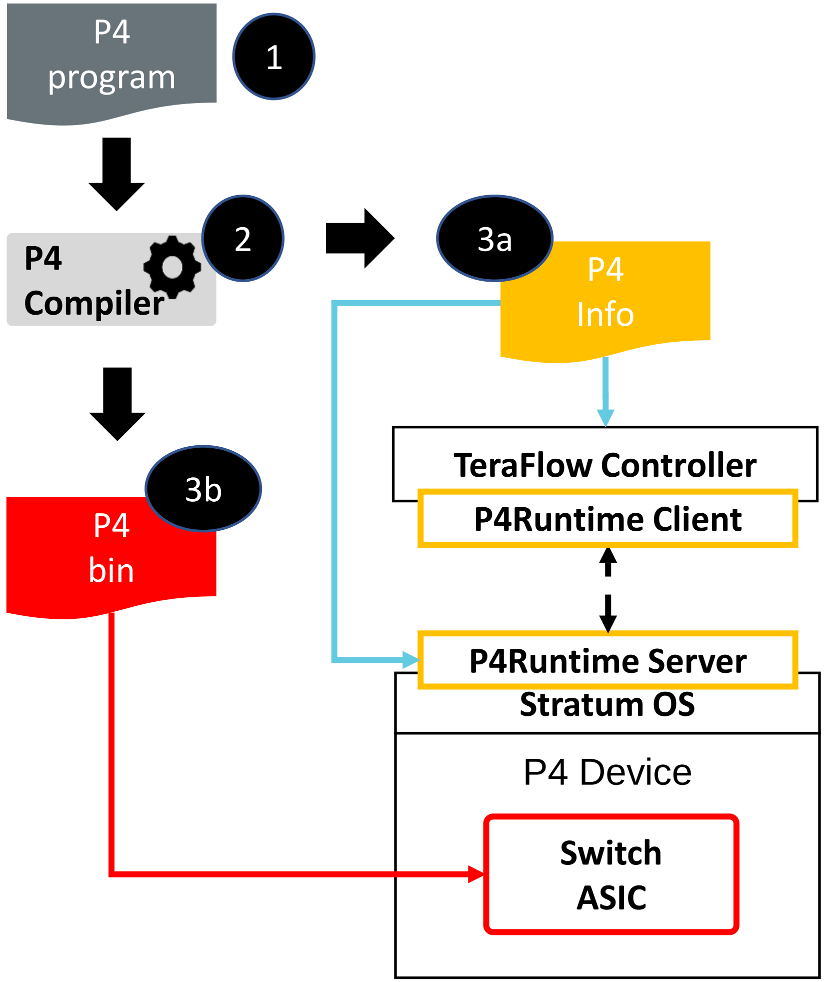
- Support for L2/L3 VPN Network Models
- Control of whiteboxes with NOS based on OpenConfig. Validated with:
  - Infinera
  - ADVA
  - Emulated
  - More in the pipeline…





Support for P4

- The desired P4 program needs to be written (step 1) by a network developer and compiled (step 2) by a P4 compiler.
- The P4 compiler generates two outputs:
  - A “P4 Info” file (step 3a) which describes the “schema” of the P4 pipeline for runtime control. This schema captures P4 program attributes such as tables, actions, parameters, etc, in a target-independent format (I.e., same P4Info for a software switch, ASIC, etc.);
  - A target-specific “P4 bin” binary (step 3b) used to realize a switch pipeline, such as a binary configuration for an application-specific integrated circuit (ASIC), a bitstream for a field-programmable gate array (FPGA), etc.
- At runtime the TeraFlowSDN controller uses a gRPC-based P4Runtime interface to manage the match-action pipelines specified in the P4 program.



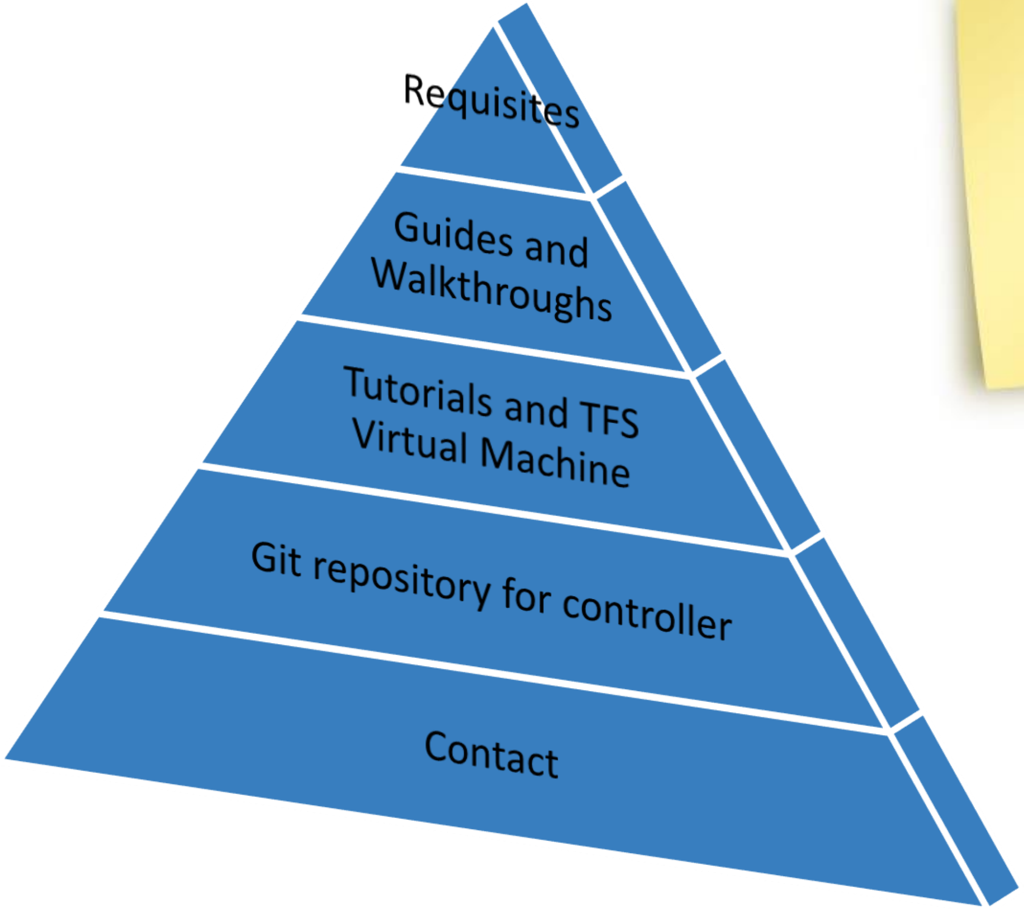
Configuration and Monitoring with gNMI

Released today!

NBI Extensions

- New NBI interfaces
  - Extend IETF Slice/L2VPN/L3VPN
  - IETF Topology
  - Device Inventory
  - ONF Transport API
  - MEC BWM API

Our single point of entry: https://tfs.etsi.org



Hackfest #2: 20-21 June 2023, Madrid (Spain). Collocated with IEEE NetSoft

Hackfest #3: 16-17 October, Castelldefels (Spain)

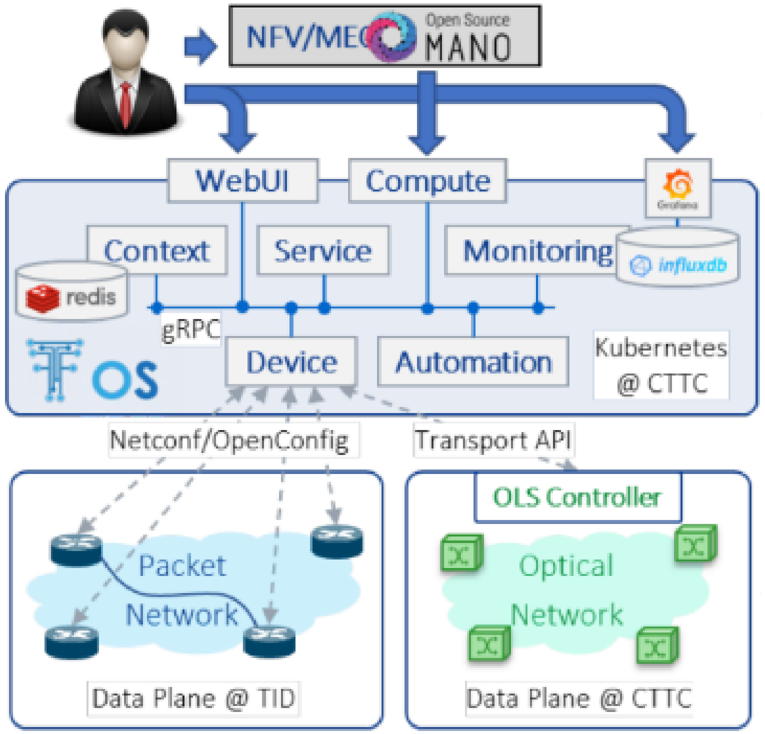
TFS Ecosystem day: 18 October, Castelldefels (Spain)

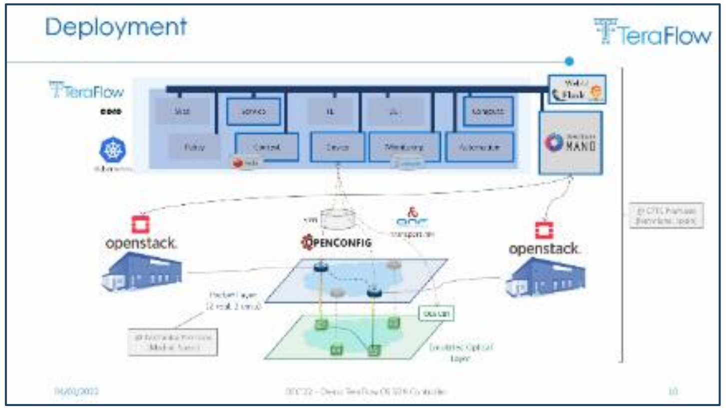
Bridges to Research – Building the TFS ecosystem

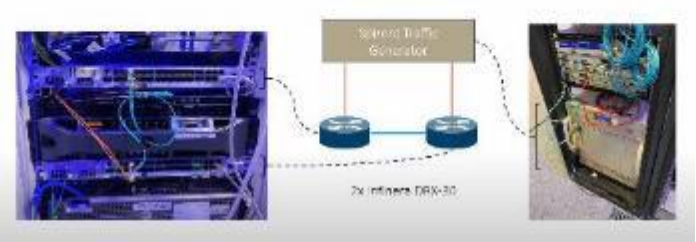
- TeraFlow

TeraFlowSDN
Demos and Use cases

ETSI OpenSourceMANO and ETSI TeraFlowSDN integration

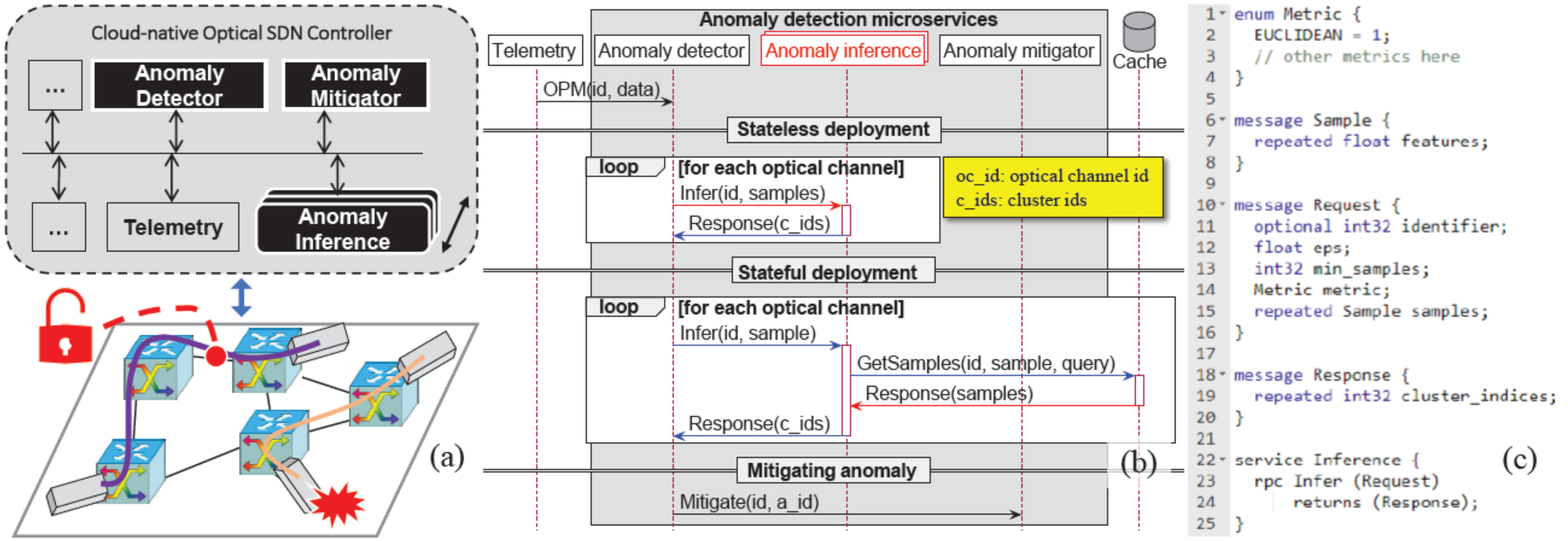






Demonstration of Zero-touch Device and L3-VPN Service Management using the TeraFlow Cloud-native SDN Controller, Ll. Gifre, C. Natalino, S. Gonzalez-Diaz, F. Soldatos, S. Barguil, C. Aslanoglou, F. J. Moreno-Muro, A. N. Quispe Cornelio, L. Cepeda, R. Martinez, C. Manso, V. Apostolopoulos, S. Petteri Valiviita, O. Gonzalez de Dios, J. Rodriguez, R. Casellas, P. Monti, G. P. Katsikas, R. Muñoz, and R. Vilalta

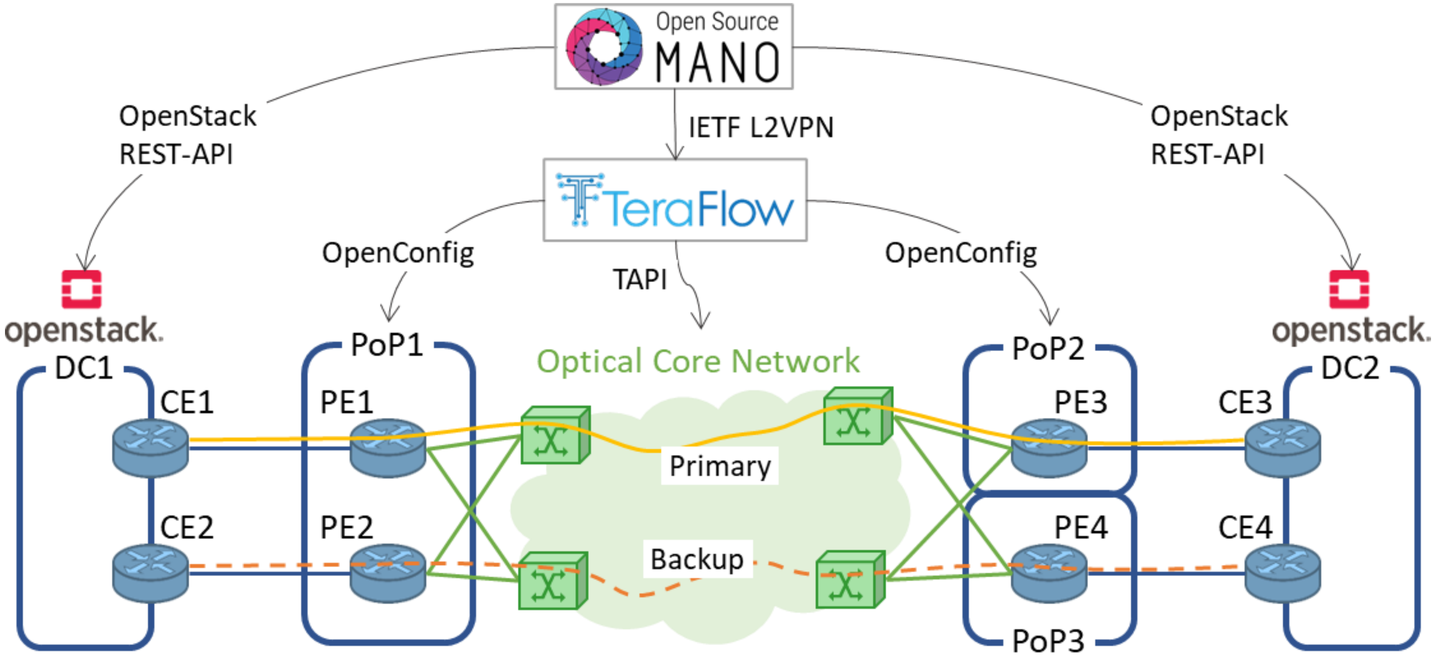
TeraFlowSDN release 1 and cybersecurity



Microservice-Based Unsupervised Anomaly Detection Loop for Optical Networks, Carlos Natalino, Carlos Manso, Lluis Gifre, Raul Muñoz, Ricard Vilalta, Marija Furdek, Paolo Monti

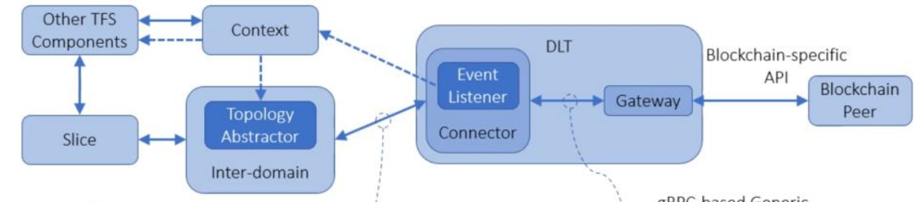
Transport Network Slicing with SLA Using the TeraFlowSDN Controller

This demo presents the TeraFlowSDN controller as a solution to provide dedicated transport network slices with SLAs. To this end, the demo details how the interface between an NFV orchestrator and the SDN controller can provide transport network slices using protected disjoint paths.



Experimental Demonstration of Transport Network Slicing with SLA Using the TeraFlowSDN Controller
Ll. Gifre, D. King, A. Farrel, R. Casellas, R. Martinez, J.-P. Fernández-Palacios, O. González-de-Dios, J.-J. Pedreno-Manresa, A. Autenrieth, R. Muñoz, R. Vilalta

DLT-based End-to-end Inter-domain Transport Network Slice with SLA Management Using Cloud-based SDN Controllers

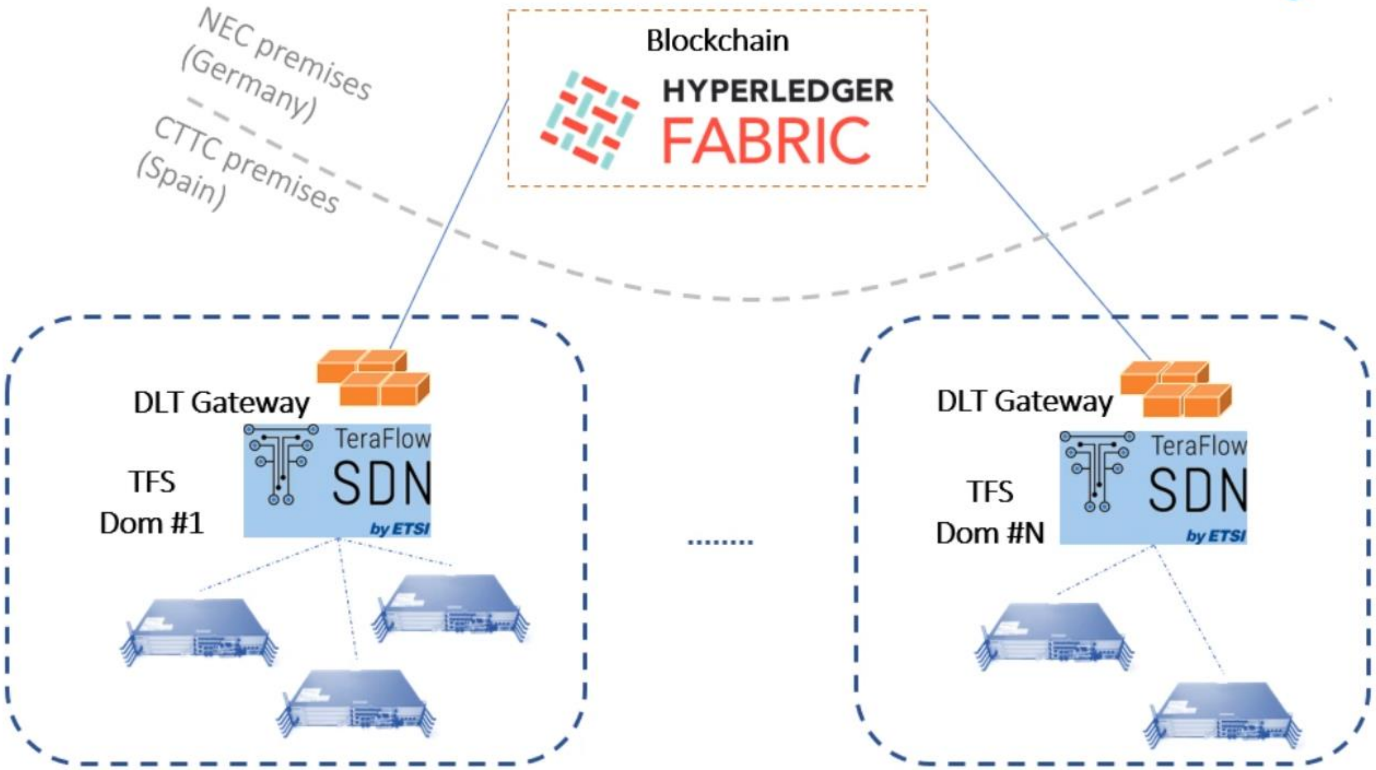


gRPC-based DLT API

Method	Parameters	Response
RecordAllDevices RecordAllLinks RecordAllSlices	-	-
RecordDevice	DeviceId	-
RecordLink	LinkId	-
RecordSlice	SliceId	-

gRPC-based Generic Gateway API

Method	Parameters	Response
RecordToDlt	Record	RecordType, RecordId, Status, ErrorMsg
GetFromDlt	RecordId, RecordType	Record
SubscribeToDlt	List[RecordType], List[Operation]	Stream [<Operation, RecordType, RecordId>]



IEEE NFV-SDN

cttc Centre Tecnològic de Telecomunicacions de Catalunya

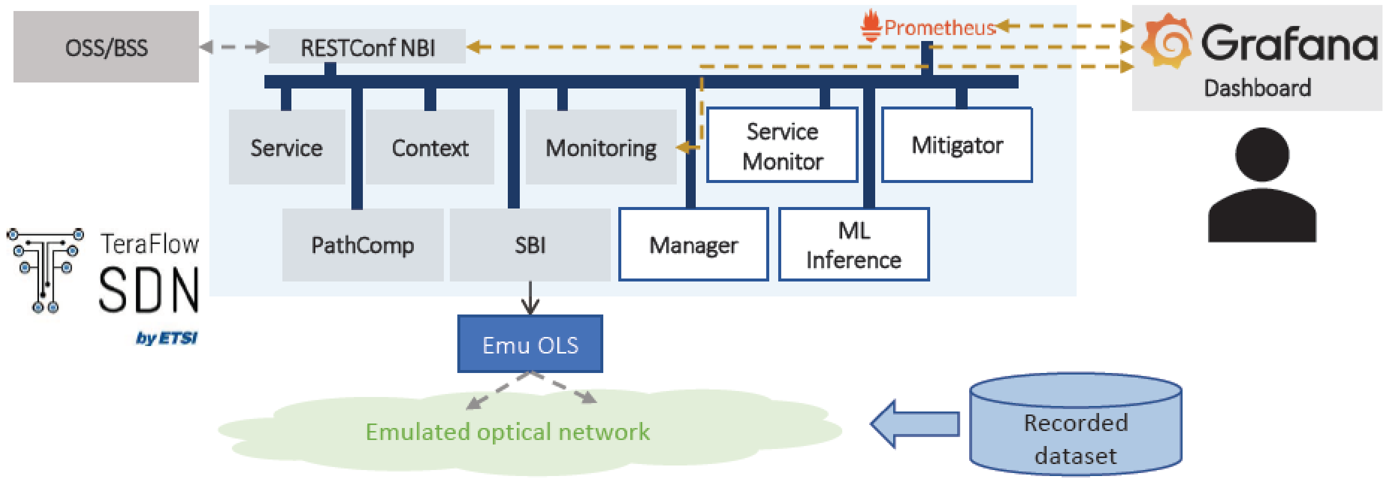
NEC

telenor

NTNU Norwegian University of Science and Technology

Network Security

We demonstrate a scalable processing of OPM data using ML to detect anomalies in optical services at run time. A dashboard will show operational SDN controller metrics, raw OPM data, and the ML assessment results



Carlos Natalino, Lluis Gifre, Raul Muñoz, Ricard Vilalta, Marija Furdek, Paolo Monti, "Scalable and Efficient Pipeline for ML-based Optical Network Monitoring", Demo Zone OFC 2023

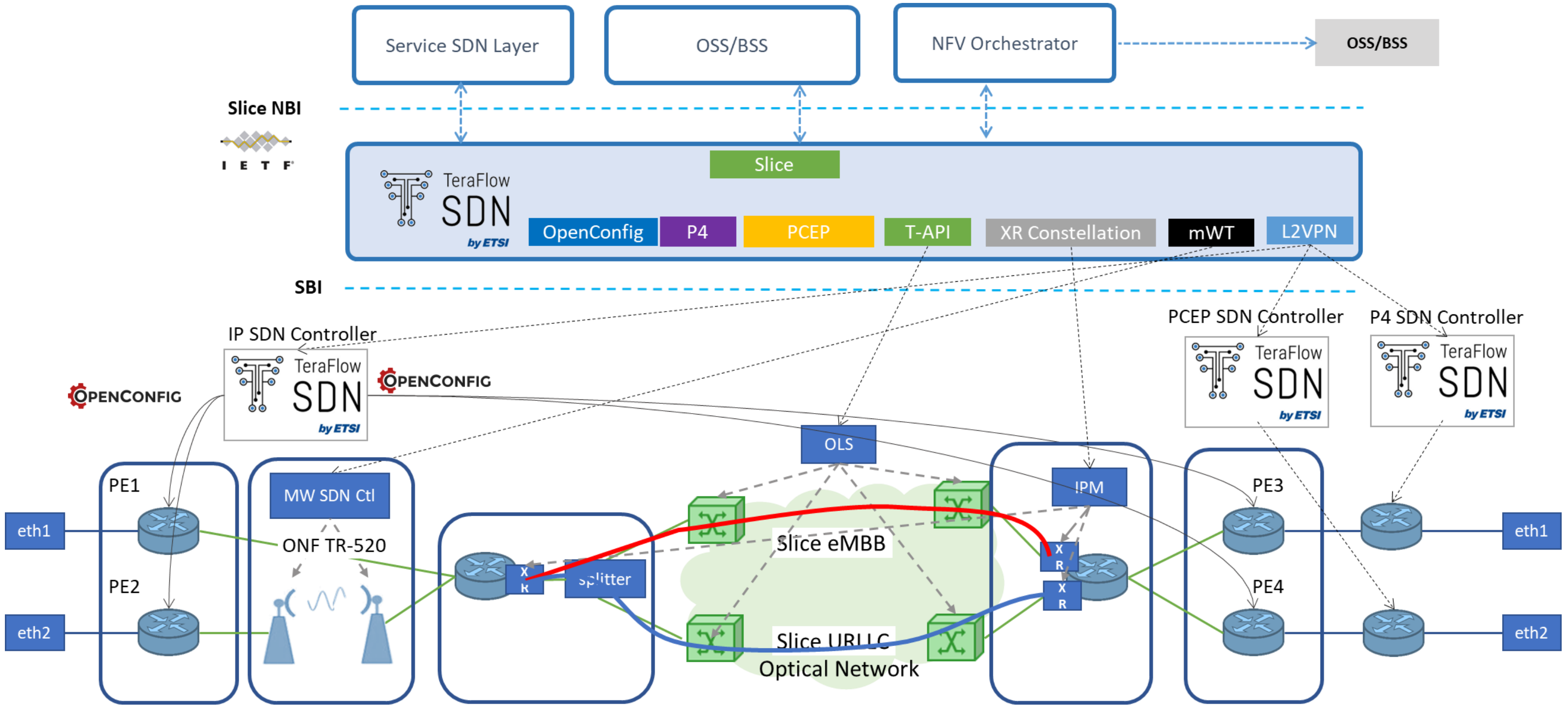
OFC 2023

Bringing network automation in transport networks

TeraFlow SDN

This demonstration showcases how TeraFlowSDN provides support for hierarchical control of multiple heterogeneous SDN domains (through IP, microwave and optical technologies). Different transport slices are offered with multiple SLAs and grouped to optimize resources

OFC 2023



Ll. Gifre, R. Vilalta, J.C. Caja-Díaz, O. Gonzalez de Dios, J.P. Fernández-Palacios, J.-J. Pedreno-Manresa, A. Autenrieth, M. Silvola, N. Carapellese, M. Milano, A. Farrel, D. King, R. Martinez, R. Casellas, and R. Muñoz, "Slice Grouping for Transport Network Slices Using Hierarchical Multi-domain SDN Controllers", Demo zone OFC 2023.

TeraFlowSDN Evolution

Need for TeraFlowSDN evolution

TeraFlow SDN

Edge – cloud continuum using Intent Based Networking	• Intelligent connectivity across a huge number of heterogeneous domains, resources with unlimited number of application requirements and conflict resolution mechanisms for incompatible requirements. • IT tools and practices extending to network (NetOps)
Accountable and Sustainable Networks	• Need to measure impact and deploy networks and services that minimize carbon footprint.
Disaggregated HW and SW evolution	• Need for operational simplicity. • Need for accelerated innovation.
Zero Trust Networks	• System integrity and self-preservation • Digital Twin Networks for Protected modes
Avoid industry fragmentation	• Competing standards addressing same areas and use cases.

Proposed TeraFlowSDN evolution paths

End-to-End Sustainable Data plane evolution

End-to-End orchestration between SMO and Transport

In-band processing, mission-critical and high priority traffic flows

Support for sustainable networks

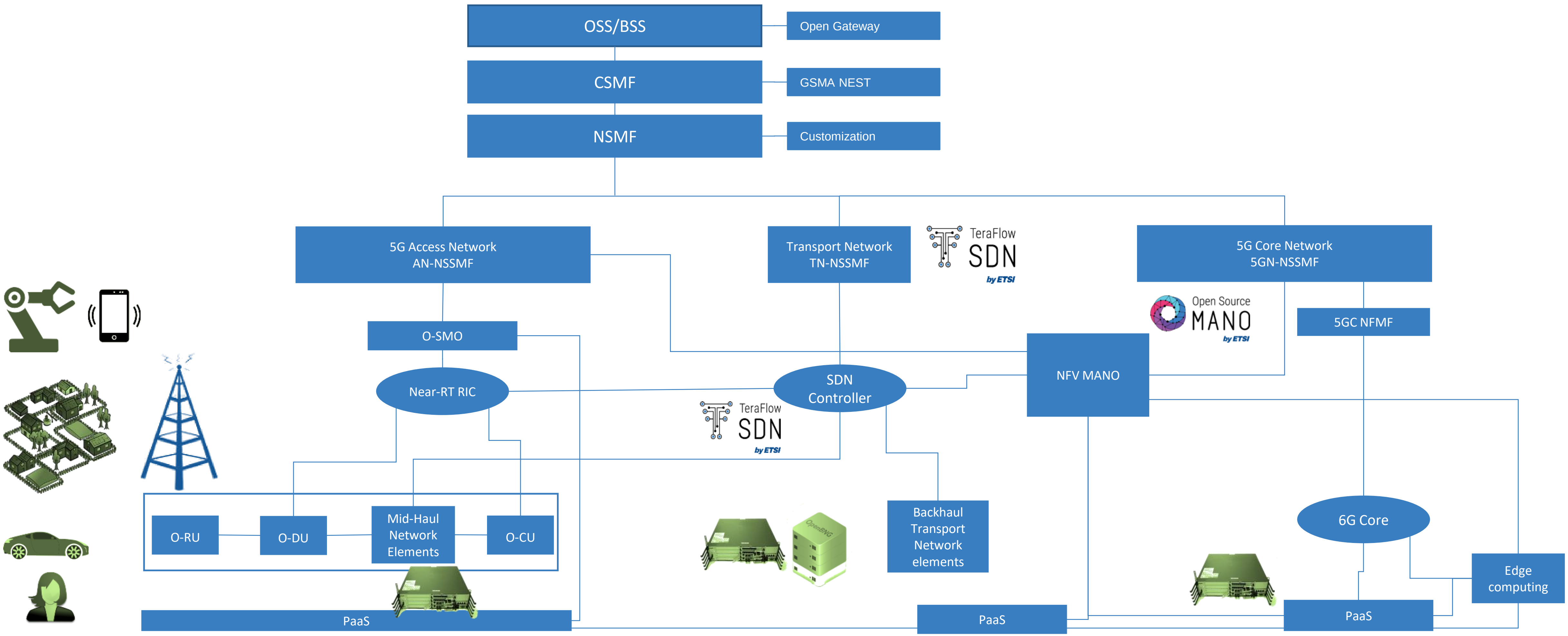
Accountable Edge-cloud continuum

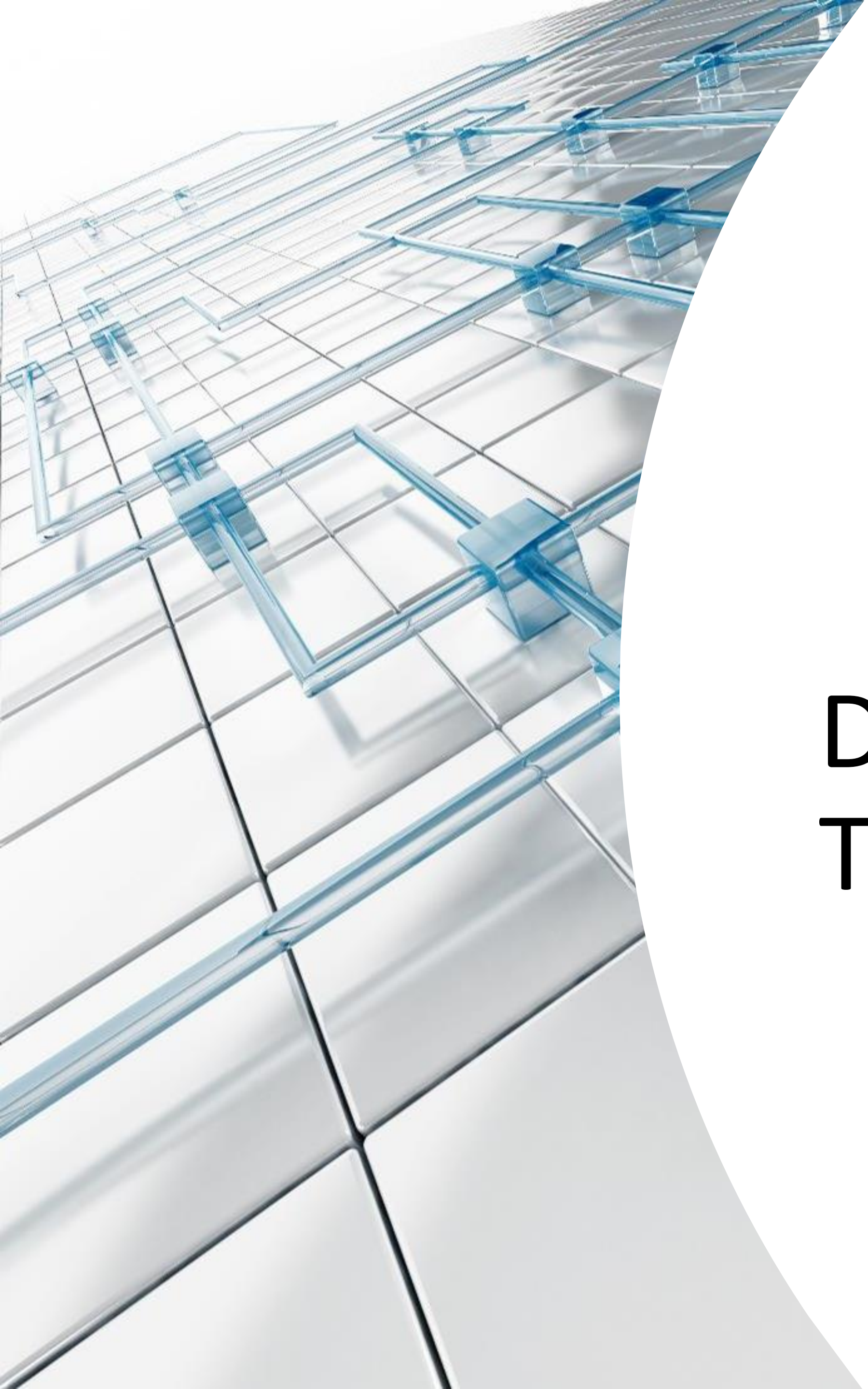
System integrity, self-preservation and accountability

Efficient Network and Service Resource Management in dynamic multi-tenant environments

Frictionless inter-domain resource management

Innovations of TeraFlowSDN in 6G networks





TeraFlow SDN

Deploy and Basic use of TeraFlowSDN

Deploy TeraFlowSDN controller

TeraFlowSDN controller runs a number of microservices on top of a Kubernetes-based environment. For development and demonstration purposes, we use MicroK8s v1.24.

- The minimum requirements are:
  - Ubuntu 20.04 or 22.04 LTS operating system (server or desktop)
  - 4 vCPUs @ 100% execution capacity
  - 8 GB of RAM
  - 40 GB of storage disk (recommended 60 GB if used for development)

For the sake of simplicity, we provide a pre-installed Ubuntu 22.04 VM with MicroK8s v1.24 installed.

- To perform your own installation, follow the steps described in the Wiki pages:
  - https://labs.etsi.org/rep/tfs/controller/-/wikis/1.-Deployment-Guide/1.2.-Create-Virtual-Machine/1.2.1.-Introduction
  - https://labs.etsi.org/rep/tfs/controller/-/wikis/1.-Deployment-Guide/1.3.-Install-MicroK8s

Deploy TeraFlowSDN controller

TeraFlow SDN

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

https://labs.etsi.org/rep/tfs/controller/-/wikis/1.-Deployment-Guide/1.3.-Install-MicroK8s

- Check status of MicroK8s:

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

- If the command reports “microk8s is not running”, start MicroK8s:

```
microk8s.start
```

Deploy TeraFlowSDN controller

TeraFlow SDN

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

https://labs.etsi.org/rep/tfs/controller/-/wikis/1.-Deployment-Guide/1.3.-Install-MicroK8s

- Report status of MicroK8s every second. Wait till addons “dns, helm3, hostpath-storage, ingress, registry” are enabled, then terminate command with Ctrl+C.

```
watch -n 1 microk8s.status --wait-ready
```

- Report status of TFS components every second. Wait till all pods are Running and Available, then terminate command with Ctrl+C.

```
watch -n 1 kubectl get all --all-namespaces
```

Deploy TeraFlowSDN controller

Specifications to deploy TeraFlowSDN are defined in a bash script as a set of environment variables. Complete details available in:

https://labs.etsi.org/rep/tfs/controller/-/wikis/1.-Deployment-Guide/1.4.-Deploy-TeraFlowSDN

- Organized in 4 sections:
  - TeraFlowSDN: variables related to the deployment of TeraFlowSDN controller
  - CockroachDB: variables related to the deployment of CockroachDB distributed database (used by Context)
  - NATS: variables related to the deployment of NATS message broker (used by Context)
  - QuestDB: variables related to the deployment of QuestDB time-series database (used by Monitoring)

- Changing the default values for CockroachDB, NATS, and QuestDB is for advanced setups.
  - Not covered in this session.

Deploy TeraFlowSDN controller

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

- Example TeraFlowSDN section (check “my_deploy.sh” for the complete list of settings)

```
# Set the URL of the internal MicroK8s Docker registry where the images will be uploaded to.
export TFS_REGISTRY_IMAGES="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 TFS to.
export TFS_K8S_NAMESPACE="tfs"

# Set additional manifest files to be applied after the deployment (example, NGINX ingress controller)
export TFS_EXTRA_MANIFESTS="manifests/nginx_ingress_http.yaml"

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

# Enable skip-build flag to prevent rebuilding the Docker images (images are pre-built, only for demo purposes).
export TFS_SKIP_BUILD="YES"
```

Deploy TeraFlowSDN controller

If you want to tweak the deployment specifications, create a copy of “my_deploy.sh” script and adjust parameters at will.

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

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

The script deploys CockroachDB, NATS and QuestDB, and then procedes with TFS deployment.

- The deployment might take few minutes the first time…
  - VM is configured with pre-built components to speed-up the deployment.
- You should see the progress of the deployment.

Deploy TeraFlowSDN controller

```
Drop database if exists
Error from server (NotFound): namespaces "crdb" not found

CockroachDB
…
CockroachDB (single-node)
CockroachDB Port Mapping
…


NATS
…
Install NATS (single-node)
NATS Port Mapping
…


QuestDB
…
QuestDB
QuestDB Port Mapping
…
```

```
Deleting and Creating a new namespace...
…

Create secret with CockroachDB data
Create secret with NATS data
Create secret with QuestDB data

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

Configuring WebUI DataStores and Dashboards...
…
```

Deploy TeraFlowSDN controller

TeraFlow SDN

The process concludes reporting the status of the microservices.

You can always retrieve this status as follows:

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

```
Deployment Resources:
NAME                                        READY   STATUS    RESTARTS        AGE
pod/contextservice-55f7f77f-dqfsc           1/1     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
…

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

Onboard Devices using TeraFlowSDN

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

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


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

Onboard Devices using TeraFlowSDN

TeraFlow SDN

Onboard an emulated device

- Example (see ~/tfs-ctrl/hackfest/tfs-descriptors/emulated-topology.json)

```
{
    "devices": [
        {
            "device_id": {"device_uuid": {"uuid": "R1"}},
            "device_type": "emu-packet-router",
            "device_drivers": [0],
            "device_endpoints": [],
            "device_operational_status": 1,
            "device_config": {"config_rules": [
                {"action": 1, "custom": {"resource_key": "_connect/address", "resource_value": "127.0.0.1"}},
                {"action": 1, "custom": {"resource_key": "_connect/port", "resource_value": "0"}},
                {"action": 1, "custom": {"resource_key": "_connect/settings", "resource_value": {"endpoints": [
                    {"uuid": "1/1", "type": "copper", "sample_types": []},
                    {"uuid": "1/2", "type": "copper", "sample_types": []},
                    ...
                ]}}}
            ]}
        },
        ...
    ]
}
```

Check “src/common/DeviceTypes.py”

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

Onboard Devices using TeraFlowSDN

TeraFlow SDN

Onboard an emulated device

- Example (see ~/tfs-ctrl/hackfest/tfs-descriptors/emulated-topology.json)

Set Mgmt IP address and port of the target device/controller.
(ignored by Emulated driver)

```
{
    "devices": [
        {
            "device_id": {"device_uuid": {"uuid": "R1"}},
            "device_type": "emu-packet-router",
            "device_drivers": [0],
            "device_endpoints": [],
            "device_operational_status": 1,
            "device_config": {"config_rules": [
                {"action": 1, "custom": {"resource_key": "_connect/address", "resource_value": "127.0.0.1"}},
                {"action": 1, "custom": {"resource_key": "_connect/port", "resource_value": "0"}},
                {"action": 1, "custom": {"resource_key": "_connect/settings", "resource_value": {"endpoints": [
                    {"uuid": "1/1", "type": "copper", "sample_types": []},
                    {"uuid": "1/2", "type": "copper", "sample_types": []},
                    ...
                ]}}}
            ]}
        },
        ...
    ]
}
```

Onboard Devices using TeraFlowSDN

TeraFlow SDN

Onboard an emulated device

- Example (see ~/tfs-ctrl/hackfest/tfs-descriptors/emulated-topology.json)

```
{
    "devices": [
        {
            "device_id": {"device_uuid": {"uuid": "R1"}},
            "device_type": "emu-packet-router",
            "device_drivers": [0],
            "device_endpoints": [],
            "device_operational_status": 1,
            "device_config": {"config_rules": [
                {"action": 1, "custom": {"resource_key": "_connect/address", "resource_value": "127.0.0.1"}},
                {"action": 1, "custom": {"resource_key": "_connect/port", "resource_value": "0"}},
                {"action": 1, "custom": {"resource_key": "_connect/settings", "resource_value": {"endpoints": [
                    {"uuid": "1/1", "type": "copper", "sample_types": []},
                    {"uuid": "1/2", "type": "copper", "sample_types": []},
                    ...
                ]}}}
            ]}
        },
        ...
    ]
}
```

EndPoints automatically discovered from the device
(except for emulated that we provide them in driver settings)

Onboard Devices using TeraFlowSDN

TeraFlow SDN

Onboard an emulated device

- Example (see ~/tfs-ctrl/hackfest/tfs-descriptors/emulated-topology.json)

```
{
    "devices": [
        {
            "device_id": {"device_uuid": {"uuid": "R1"}},
            "device_type": "emu-packet-router",
            "device_drivers": [0],
            "device_endpoints": [],
            "device_operational_status": 1,
            "device_config": {"config_rules": [
                {"action": 1, "custom": {"resource_key": "_connect/address", "resource_value": "127.0.0.1"}},
                {"action": 1, "custom": {"resource_key": "_connect/port", "resource_value": "0"}},
                {"action": 1, "custom": {"resource_key": "_connect/settings", "resource_value": {"endpoints": [
                    {"uuid": "1/1", "type": "copper", "sample_types": []},
                    {"uuid": "1/2", "type": "copper", "sample_types": []},
                    ...
                ]}}}
            ]}
        },
        ...
    ]
}
```

By default, DISABLED, will be activated during onboarding.
Check “proto/context.proto” “DeviceOperationalStatusEnum”

Onboard Devices using TeraFlowSDN

TeraFlow SDN

Onboard an emulated device

- Example (see ~/tfs-ctrl/hackfest/tfs-descriptors/emulated-topology.json)

```
{
    "devices": [
        {
            "device_id": {"device_uuid": {"uuid": "R1"}},
            "device_type": "emu-packet-router",
            "device_drivers": [0],
            "device_endpoints": [],
            "device_operational_status": 1,
            "device_config": {"config_rules": [
                {"action": 1, "custom": {"resource_key": "_connect/address", "resource_value": "127.0.0.1"}},
                {"action": 1, "custom": {"resource_key": "_connect/port", "resource_value": "0"}},
                {"action": 1, "custom": {"resource_key": "_connect/settings", "resource_value": {"endpoints": [
                    {"uuid": "1/1", "type": "copper", "sample_types": []},
                    {"uuid": "1/2", "type": "copper", "sample_types": []},
                    ...
                ]}}}
            ]}
        },
        ...
    ]
}
```

Drivers for real devices/controllers usually contain other settings, such as username, password, timeout, etc.
(we will see it later)

Onboard Links

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

- Example (see ~/tfs-ctrl/hackfest/tfs-descriptors/emulated-topology.json)
- Link Template:

Link UUID (if you provide a plain string, a UUID will be generated automatically)

Specify Device and Endpoint UUIDs (you can provide them as the device and endpoint name, the UUID will be located automatically)

```
{
    "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": "…"}}
            ]
        }
    ]
}
```

Create Services

Service requests can be uploaded using JSON-based descriptors as well.

- Example (see ~/tfs-ctrl/hackfest/tfs-descriptors/emulated-topology.json)
- 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": [ … ]}
}]}
```

Service UUID (if you provide a plain string, a UUID will be generated automatically)

Set Service Type.
Check “proto/context.proto” “ServiceTypeEnum”

Create Services

TeraFlow SDN

Service requests can be uploaded using JSON-based descriptors as well.

- Example (see ~/tfs-ctrl/hackfest/tfs-descriptors/l3-service.json)
- 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": [ … ]}
}]}
```

Set Service Status. During provisioning: PLANNED. Check “proto/context.proto” “ServiceStatusEnum”

Specify Device and Endpoint UUIDs (you can provide them as the device and endpoint name, the UUID will be located automatically)

Specify Service-specifuc Constraints (SLAs, Capacity, Latency, etc) and Config Rules (IP addresses to use, VLAN tags, etc.).

Inspect elements created

The details of the managed entities can be shown using the “eye” button.

Devices

+ Add New Device

1 devices found in context *admin*

Click to Show Details

UUID	Name	Type	Endpoints	Drivers	Status	Config Rules	
6ab8fa38-ec20-5c32-8d9b-4fd86fce2555	OLS	open-line-system	38	• TRANSPORT_API	ENABLED	41	

Error checking, if something went wrong…

Check the logs of the TeraFlowSDN components:

- Example: Device component

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

Exercise: Onboard Emulated Topology & Create Service

- Onboard Emulated Topology (~/tfs-ctrl/hackfest/tfs-descriptors/emulated-topology.json)
  - Use Upload form in “Home” tab
  - Select the created “Context/Topology” in “Home” tab
- Check devices in the “Device” tab
- Check links in the “Link” tab
- Create L3 Service (~/tfs-ctrl/hackfest/tfs-descriptors/l3-service.json)
- Check Service in the “Service” tab

10 minutes

Introduction to ContainerLab

Network Emulation

https://containerlab.dev/

https://www.gns3.com/

… and many more: https://www.brianlinkletter.com/2023/02/network-emulators-and-network-simulators-2023/

ContainerLab

CONTAINERlab

https://containerlab.dev/

- Many Network Operating Systems
  - Some containerized, others require VMs.
- Experts need to run NOSes on demand in user-defined topologies
  - Experimentation, testing, development, etc.
- Container orchestration tools (e.g., docker-compose) does not fit well with this purpose.
  - Unable to create connections defining the topology.

ContainerLab

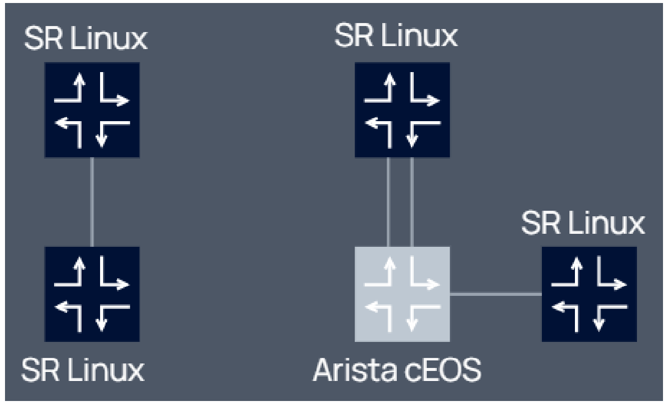
https://containerlab.dev/

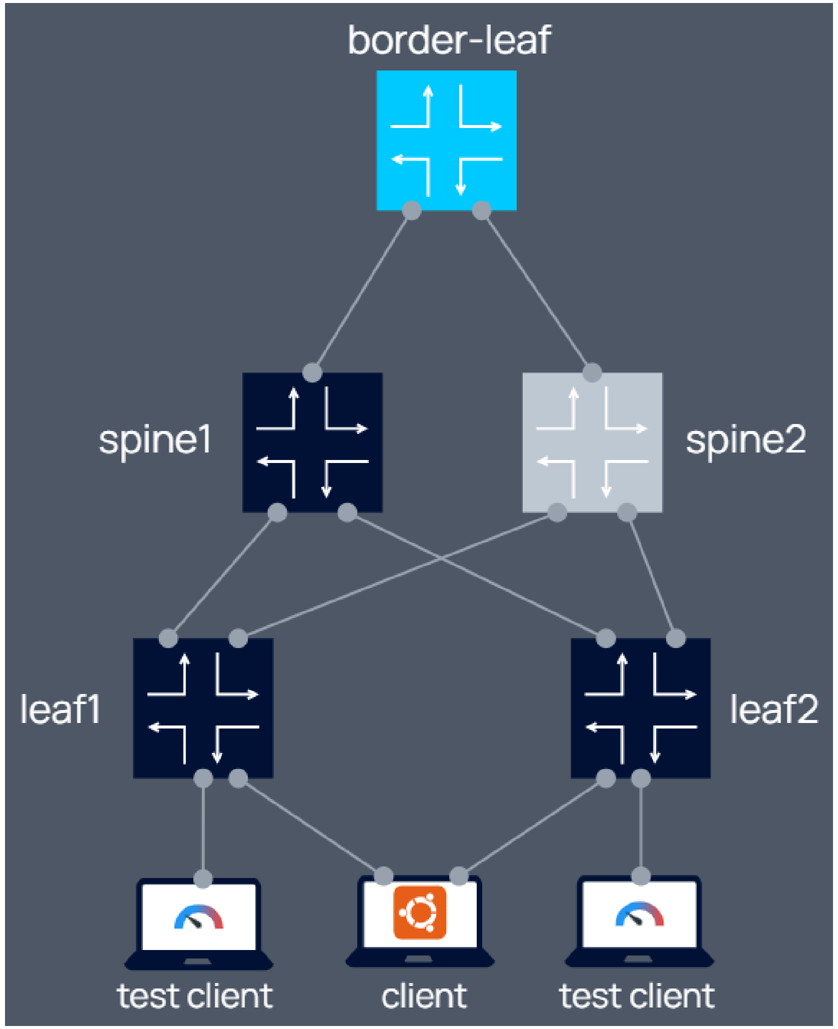
ContainerLab:

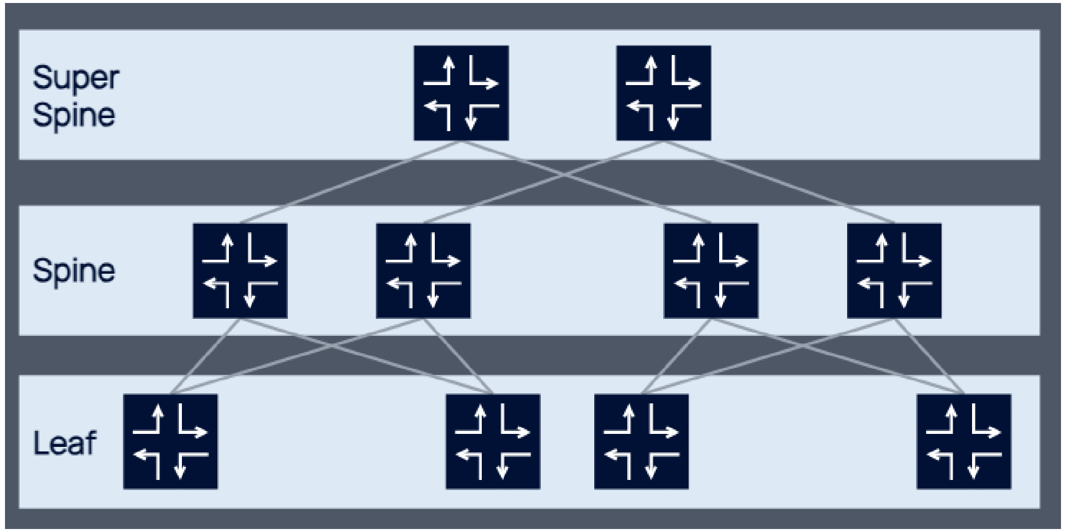
- CLI for orchestration and managing container-based networking labs
- Starts containers, builds virtual wiring between them.
- Manage labs lifecycle.
- Support for many network device kinds (https://containerlab.dev/manual/kinds/)
- Many examples (https://containerlab.dev/lab-examples/lab-examples/)

ContainerLab - Examples

https://containerlab.dev/







Quick Start (I)

CONTAINERlab

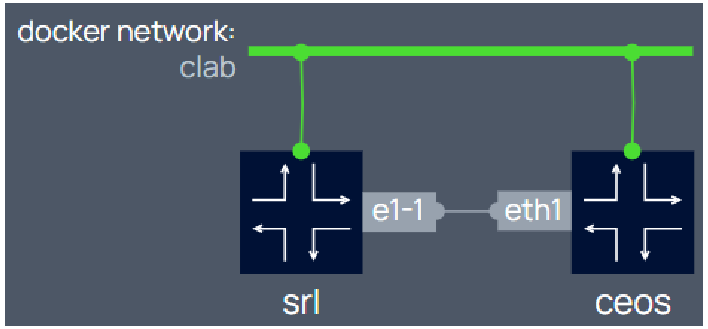
https://containerlab.dev/

Download and install the latest release (may require sudo)

```
bash -c "$(curl -sL https://get.containerlab.dev)"
```

NOTE: for this example you need to get a license for CEOS;
We Will only use SRL that can be used without a license.

Topology definition




```
name: srlceos01
topology:
  nodes:
    srl:
      kind: srl
      image: ghcr.io/nokia/srlinux
    ceos:
      kind: ceos
      image: ceos:4.25.0F
  links:
    - endpoints: ["srl:e1-1", "ceos:eth1"]
```

Additional Details:
https://containerlab.dev/quickstart/

Quick Start (II)

Check that container images are available

```
$ docker images | grep -E "srlinux|ceos"
REPOSITORY             TAG      IMAGE ID      CREATED        SIZE
ghcr.io/nokia/srlinux  latest   79019d14cfc7  3 months ago   1.32GB
ceos                   4.25.0F  15a5f97fe8e8  3 months ago   1.76GB
```

Start the lab deployment

```
$ mkdir ~/clab-quickstart
$ cd ~/clab-quickstart
$ cp -a /etc/containerlab/lab-examples/srlceos01/* .
$ containerlab deploy --topo srlceos01.clab.yml
…
```

CONTAINERlab

https://containerlab.dev/

```
+---+---------------------+--------------+-----------------------+------+-------+---------+----------------+----------------------+
| # |        Name         | Container ID |         Image         | Kind | Group |  State  |  IPv4 Address  |     IPv6 Address     |
+---+---------------------+--------------+-----------------------+------+-------+---------+----------------+----------------------+
| 1 | clab-srlceos01-ceos | 2e2e04a42cea | ceos:4.25.0F          | ceos |       | running | 172.20.20.3/24 | 2001:172:20:20::3/80 |
| 2 | clab-srlceos01-srl  | 1b9568fcdb01 | ghcr.io/nokia/srlinux | srl  |       | running | 172.20.20.4/24 | 2001:172:20:20::4/80 |
+---+---------------------+--------------+-----------------------+------+-------+---------+----------------+----------------------+
```

Additional Details:
https://containerlab.dev/quickstart/

Quick Start (III)

CONTAINERlab

https://containerlab.dev/

Connecting to the nodes

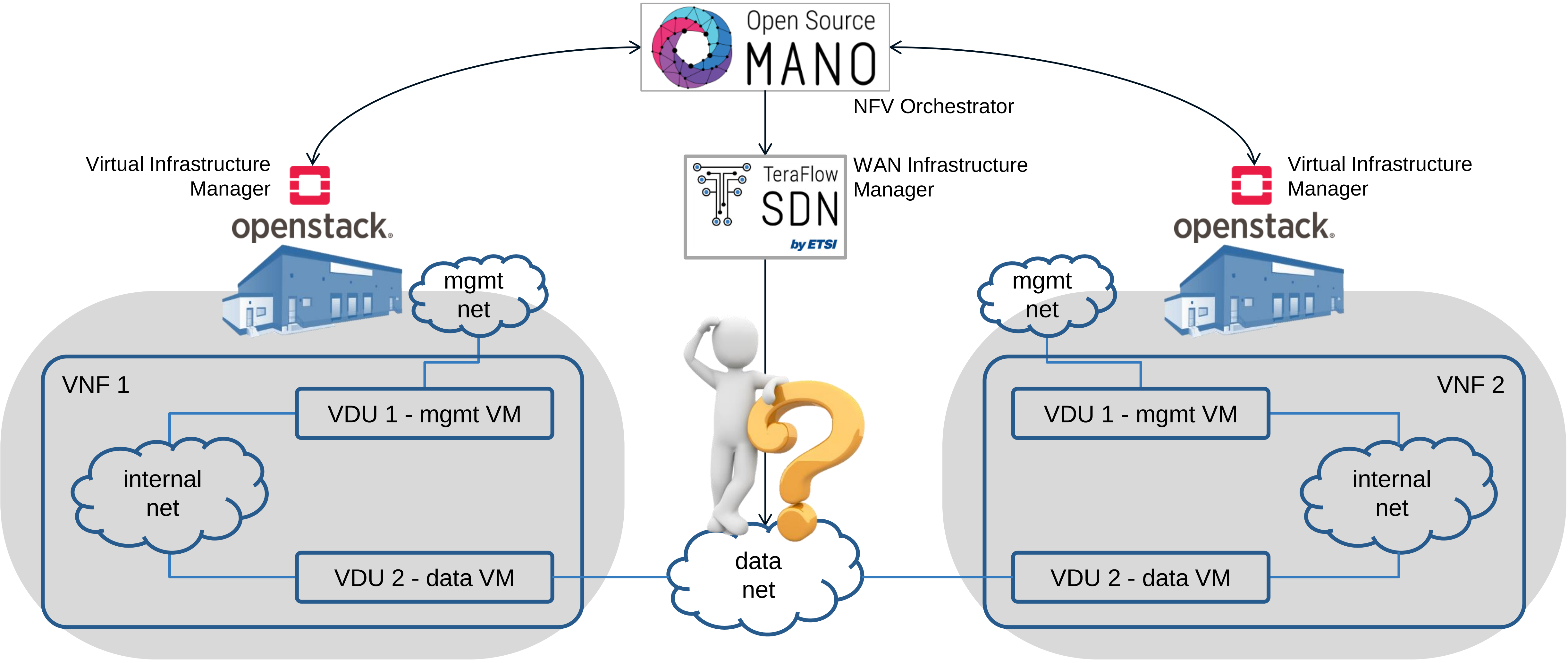
```
$ docker exec -it clab-srlceos01-srl1 sr_cli
$ docker exec -it clab-srlceos01-srl1 bash

$ ssh admin@172.20.20.3
admin@172.20.20.3's password:
Using configuration file(s): []
Welcome to the srlinux CLI.
Type 'help' (and press <ENTER>) if you need any help using this.
--{ running }--[  ]--
A:srl1#

# Creates /etc/hosts entries so you can use names
$ ssh admin@clab-srlceos01-srl
```

Destroying a lab

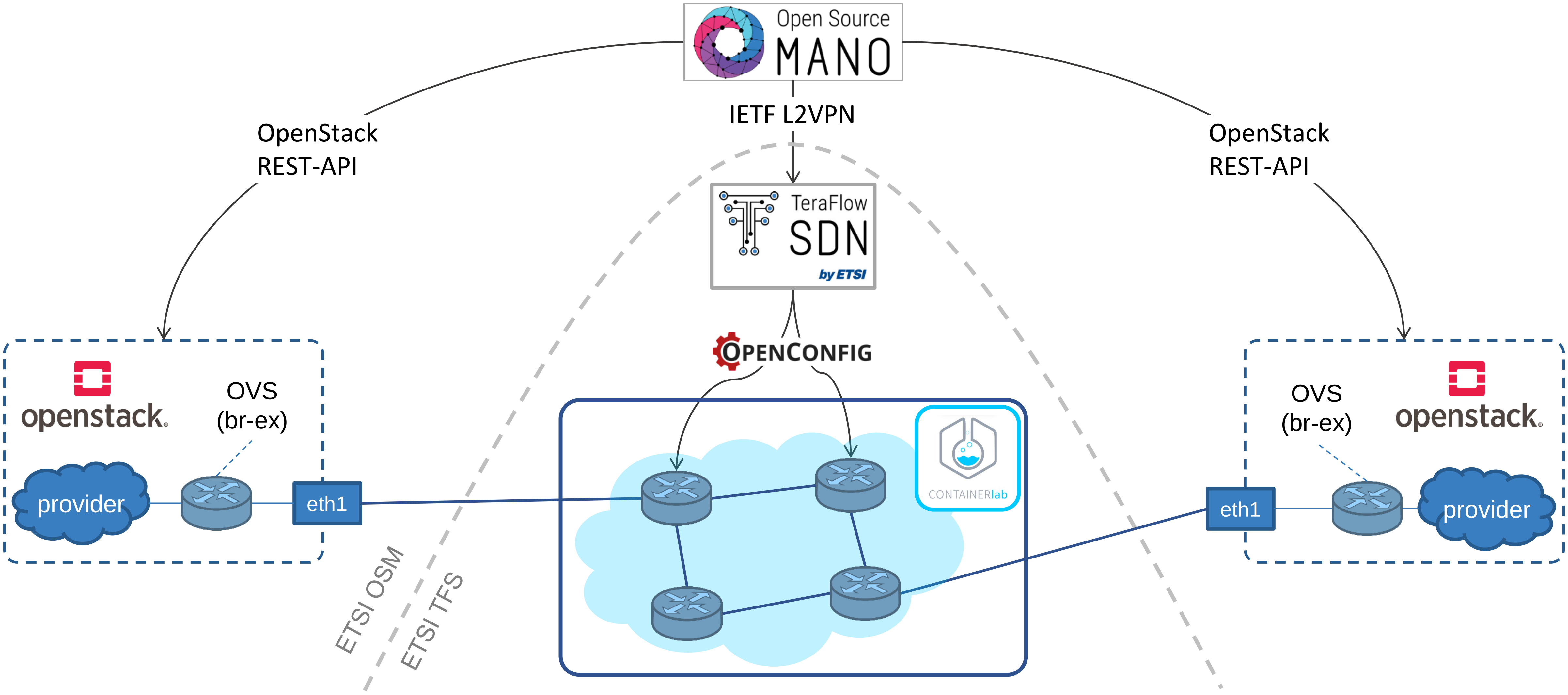
```
$ containerlab destroy --topo srlceos01.clab.yml
```

Additional Details:
https://containerlab.dev/quickstart/

Why we need ContainerLab?



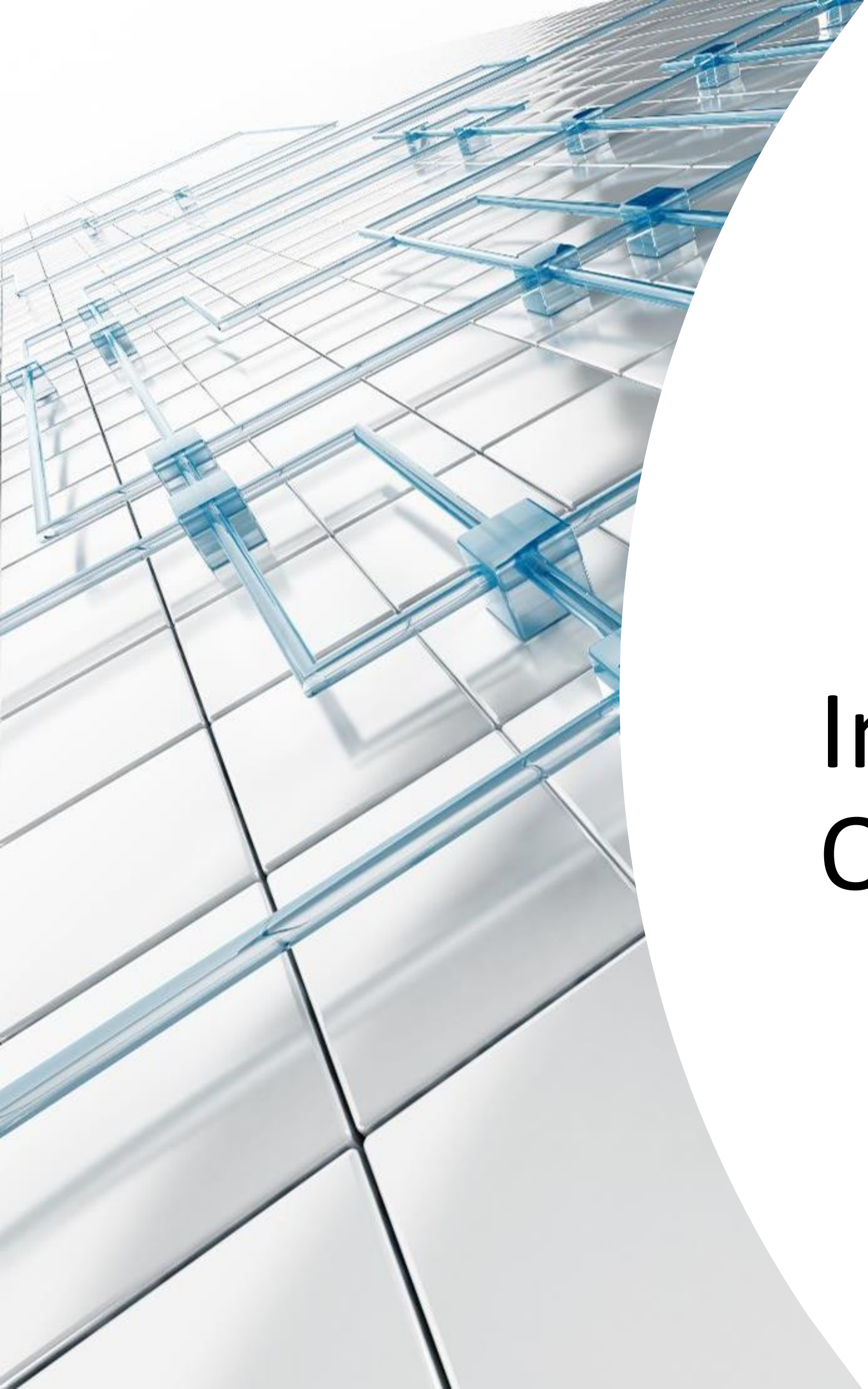
ETSI OSM-TFS Long-Term Testbed Proposal



Your participation is very valuable!

You can help us in creating the new
ETSI OSM-TFS Long-Term Testbed!

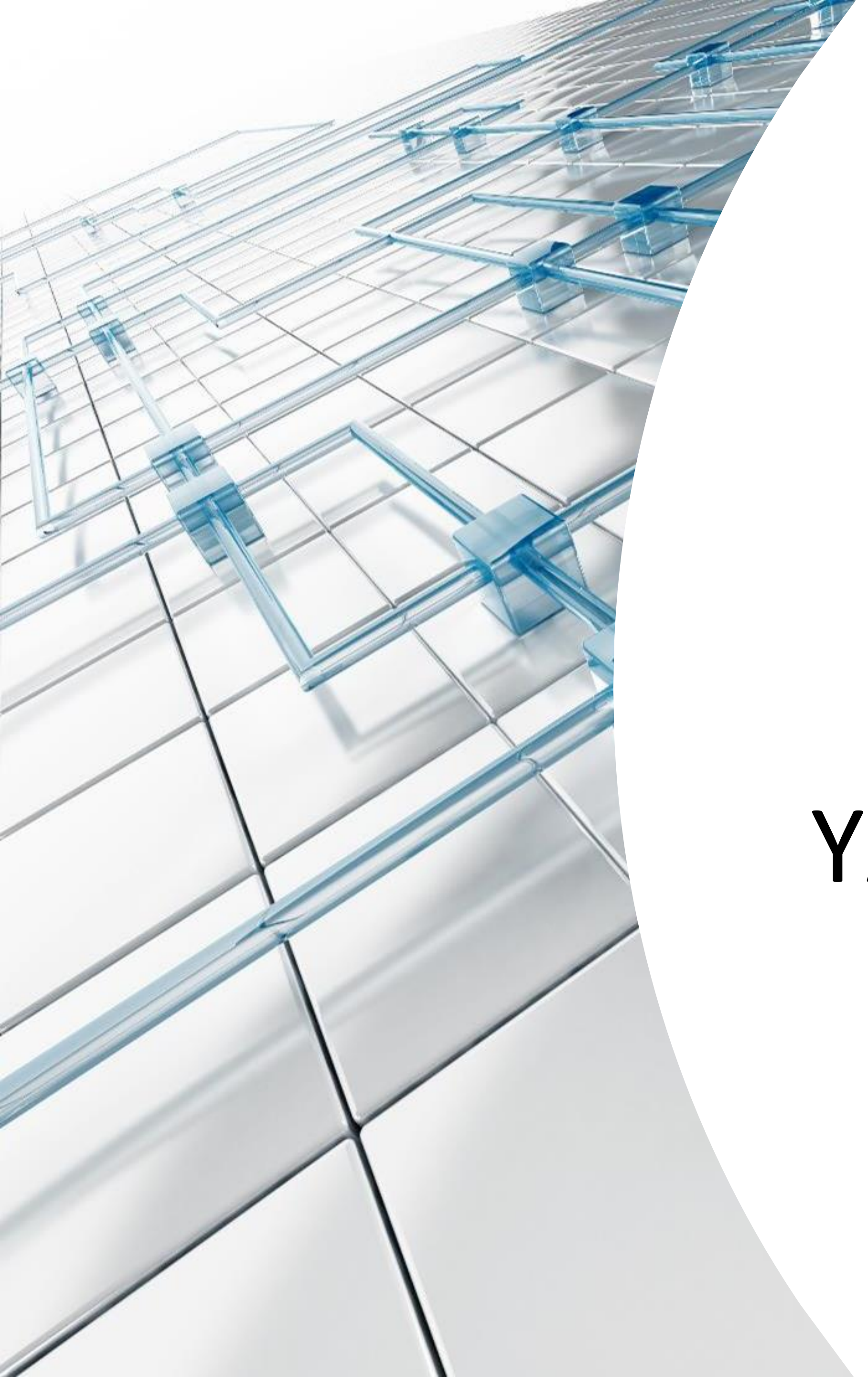
Your feedback and ideas are welcome!



Introduction to gNMI and OpenConfig

gNMI and OpenConfig

- gNMI: transport protocol based on gRPC used to exchange configuration messages and monitoring data.
- OpenConfig: a data model defined using the YANG language. It is used to encode the data sent through gNMI.
- Topics in this section:
  - YANG
  - OpenConfig
  - gRPC
  - gNMI



TeraFlow SDN

YANG

Unified Information and Data Modeling

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 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. ➔ 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)

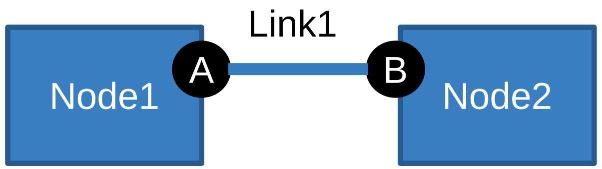
The YANG Language I

- YANG has become the data modeling language of choice for multiple network control and management aspects
  - Covering devices, networks, and services, even pre-existing protocols.
  - YANG models configuration and state data.
  - Significantly adopted, 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.

The YANG Language II

- Models define the device configurations & notifications, capture semantic details and are easy to understand.
- Ongoing notable effort across the SDOs to model constructs (e.g. topologies, protocols)
- 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.

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

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

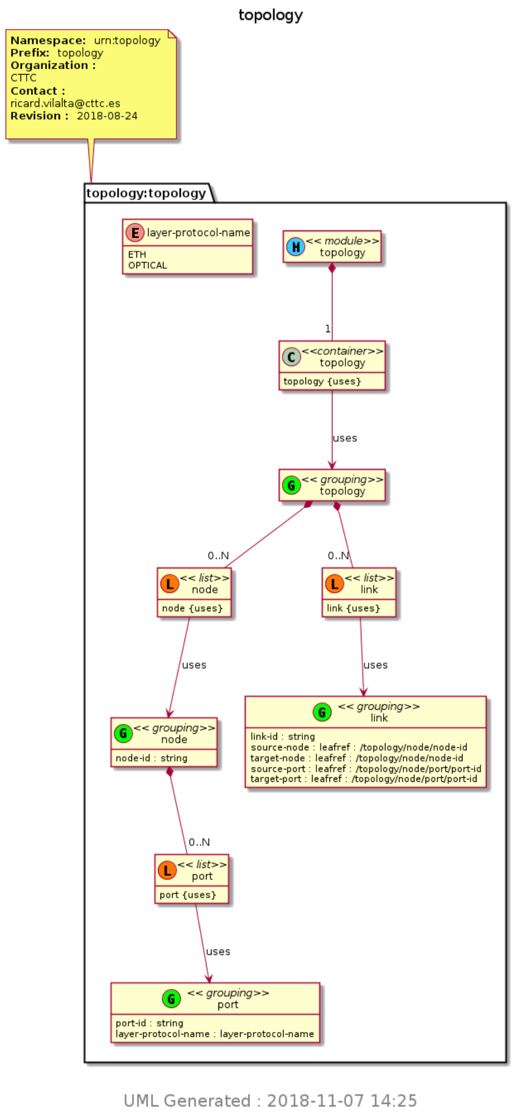
PlantUML is an opensource tool to create UML diagrams

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.

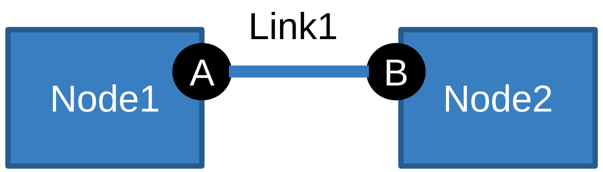
```
$ export PYBINDPLUGIN=`/usr/bin/env python -c \
'import pyangbind; import os; print ("{}/plugin".format(os.path.dirname(pyangbind.__file__)))'`
$ echo $PYBINDPLUGIN
$ pyang -f pybind topology.yang --plugindir $PYBINDPLUGIN -o binding_topology.py
```

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

```python
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))
```

Use pyangbind serializers

```
$ python3 topology.py
```

Topology XML vs JSON

```
<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": {
        "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"
            }
        }
    }
}
```

TeraFlow SDN

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

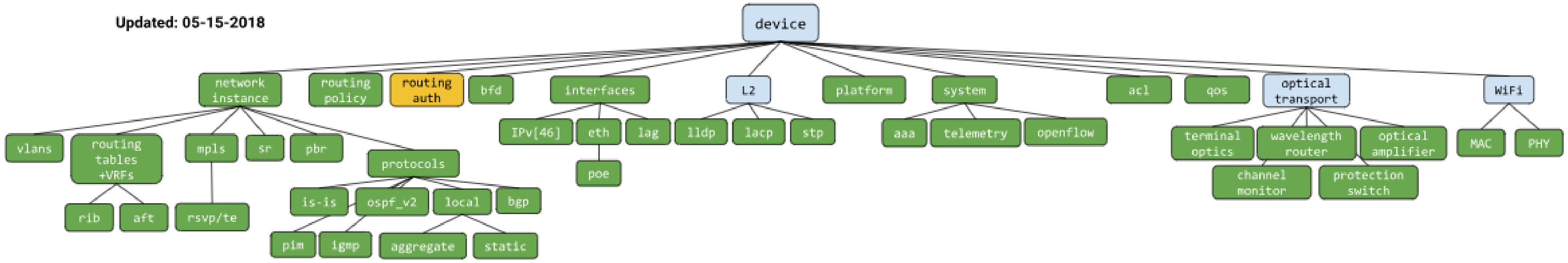
OpenConfig

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
              ...
```

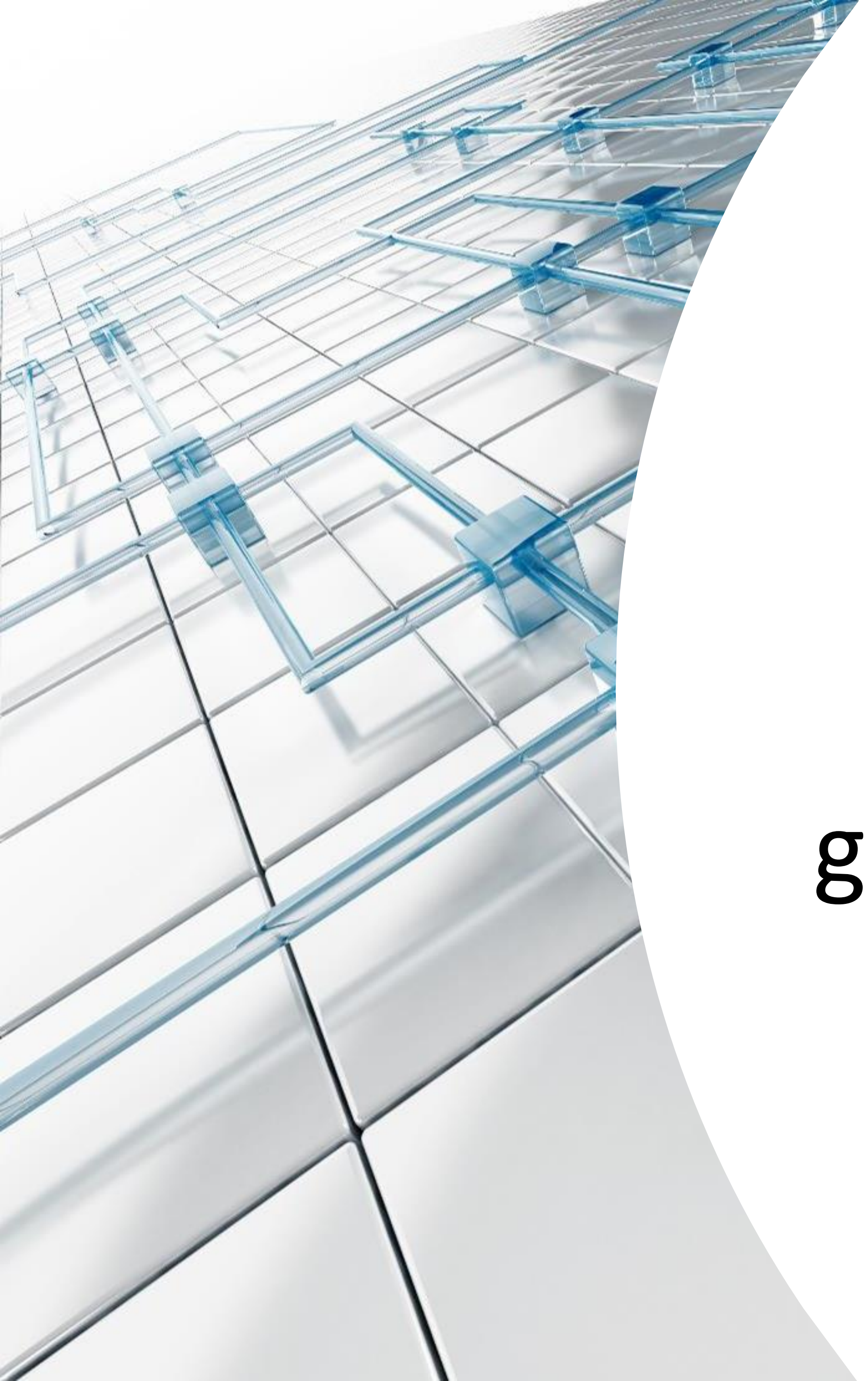
```
module: openconfig-vlan
  augment /oc-if:interfaces/oc-if:interface/oc-if:subinterfaces/oc-if:subinterface:
    +--rw vlan
       +--rw config
       |  x--rw vlan-id?   union
       +--ro state
       |  x--ro vlan-id?   union
       +--rw match
       |  +--rw single-tagged
       |  |  +--rw config
       |  |  |  +--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 – 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 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 static* [prefix]
       |     +--rw prefix     -> ../config/prefix
       |     +--rw config
       |     |  +--rw prefix?     inet:ip-prefix
       |     |  +--rw next-hop*   union
       ...   ...
       +--rw bgp ...
       +--rw ospfv2 ...
       +--rw isis ...
       ...
```

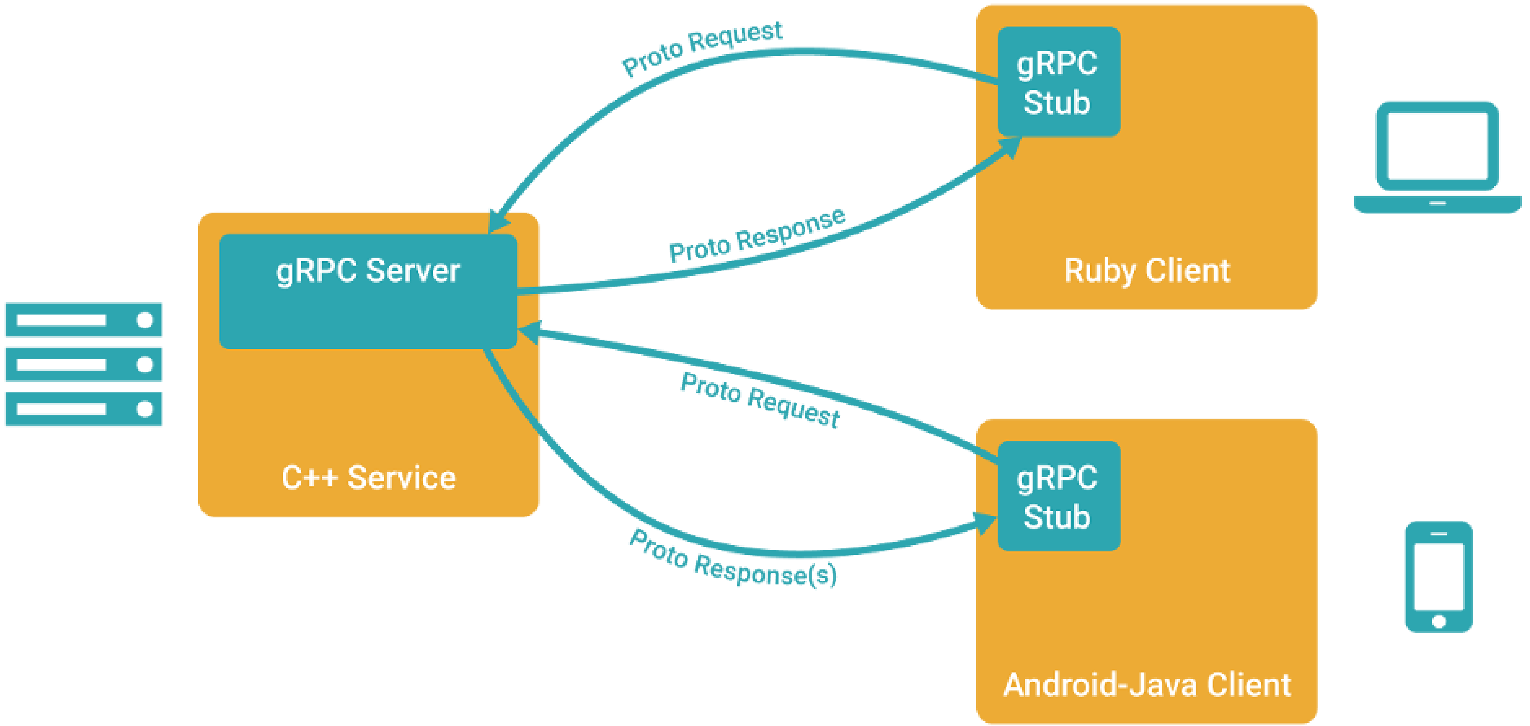


gRPC

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

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;
}
```

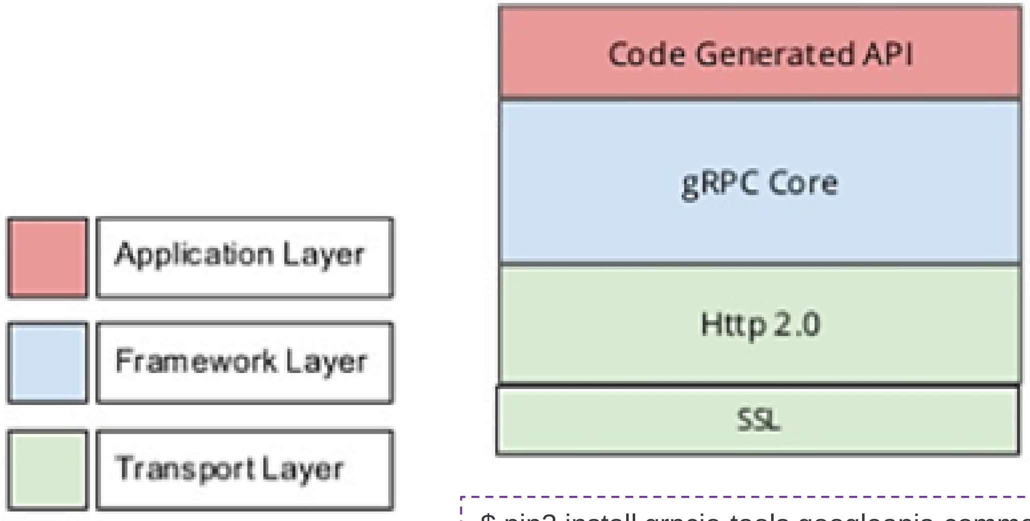
gRPC Main Use Cases and architecture

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.



Supported Languages (https://grpc.io/docs/languages/):

- C# / .NET
- C++
- Dart
- Go
- Java
- Kotlin
- Node
- Objective-C
- PHP
- Python
- Ruby

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

Usage of protobufs

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.")

…
```

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

```
…
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())
```

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

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

```
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

```
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()
```

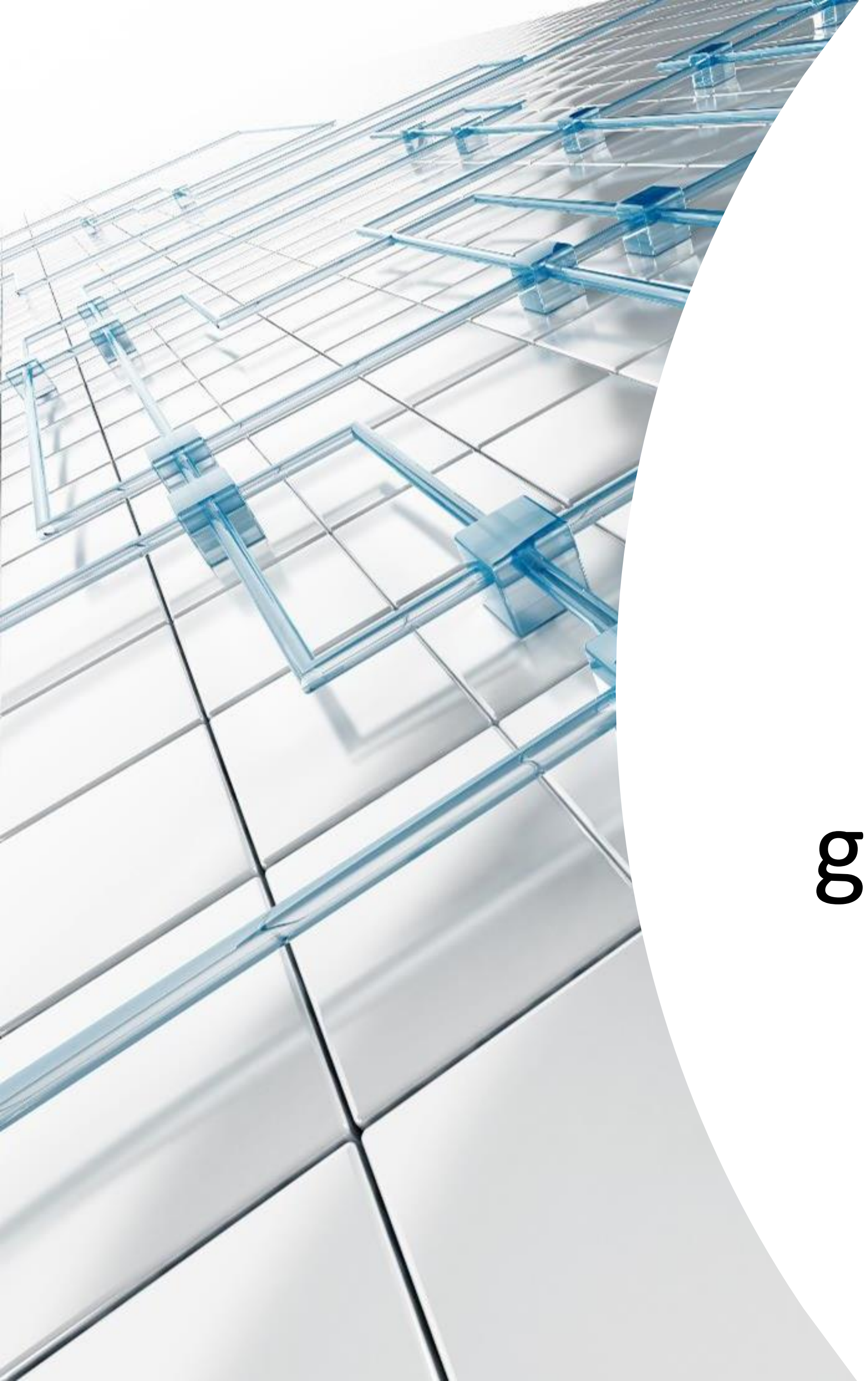
Run example

Run Server

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

Run client

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



gNMI

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);
}
```

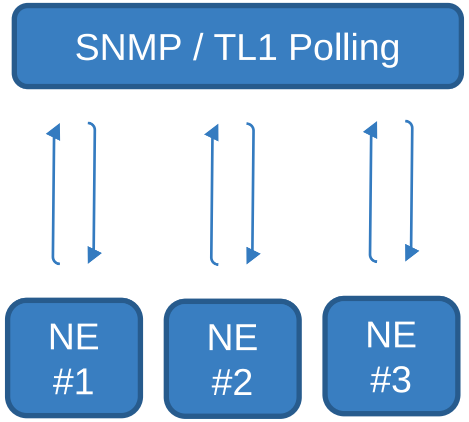
```
message GetRequest {
  Path prefix = 1;
  repeated Path path = 2;
  enum DataType {
    ALL = 0;
    CONFIG = 1;
    STATE = 2;
    OPERATIONAL = 3;
  }
  DataType type = 3;
  Encoding encoding = 5;
  repeated ModelData use_models = 6;
  repeated gnmi_ext.Extension extension = 7;
}

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;
}
```

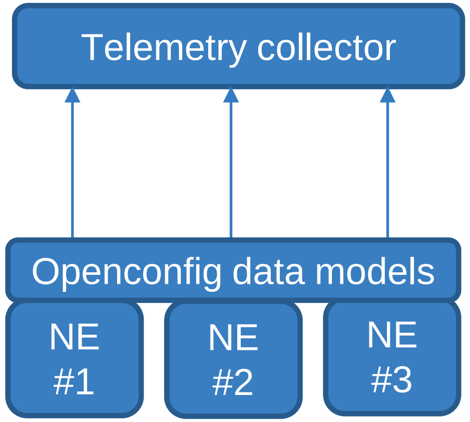
Better visibility with streaming telemetry

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

Using gNMIc

- Installation

```
sudo bash -c "$(curl -sL https://get-gnmic.kmrd.dev)"
```

- Capabilities request

```
gnmic -a clab-srlinux-srl1 -u admin -p NokiaSrl1! --skip-verify capabilities
```

- Get request

```
gnmic -a clab-srlinux-srl1 -u admin -p NokiaSrl1! --skip-verify -e json_ietf get --path
/interface[name=mgmt0]
```

Using gNMIc

- Get request

```
gnmic -a clab-srlinux-srl1 -u admin -p NokiaSrl1! --skip-verify -e json_ietf get --path
/system/name/host-name
```

- Set request

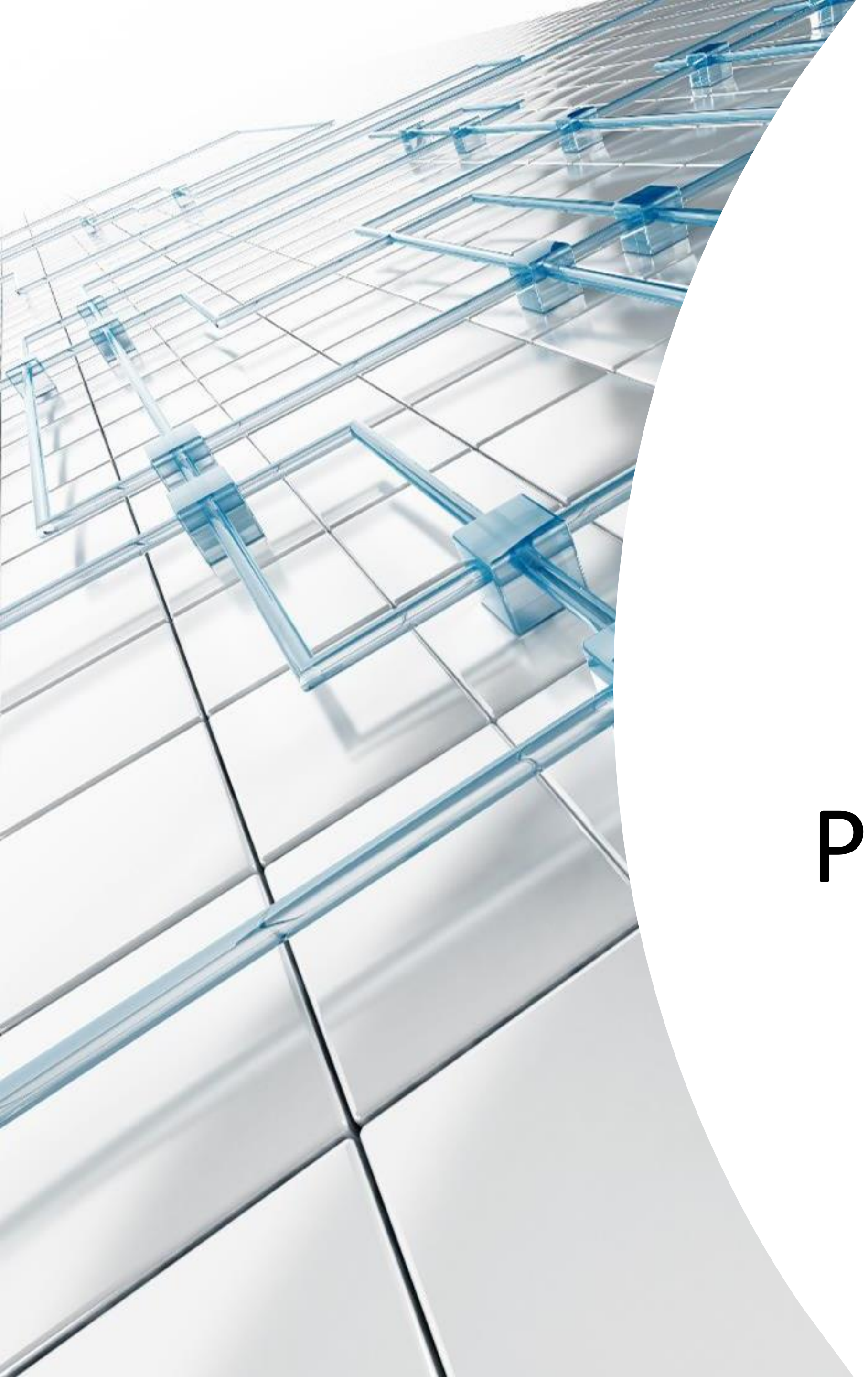
```
gnmic -a clab-srlinux-srl1 -u admin -p NokiaSrl1! --skip-verify -e json_ietf set --update-path
  /system/name/host-name --update-value slr11
```

(check with previous Get Request)

- Subscribe request

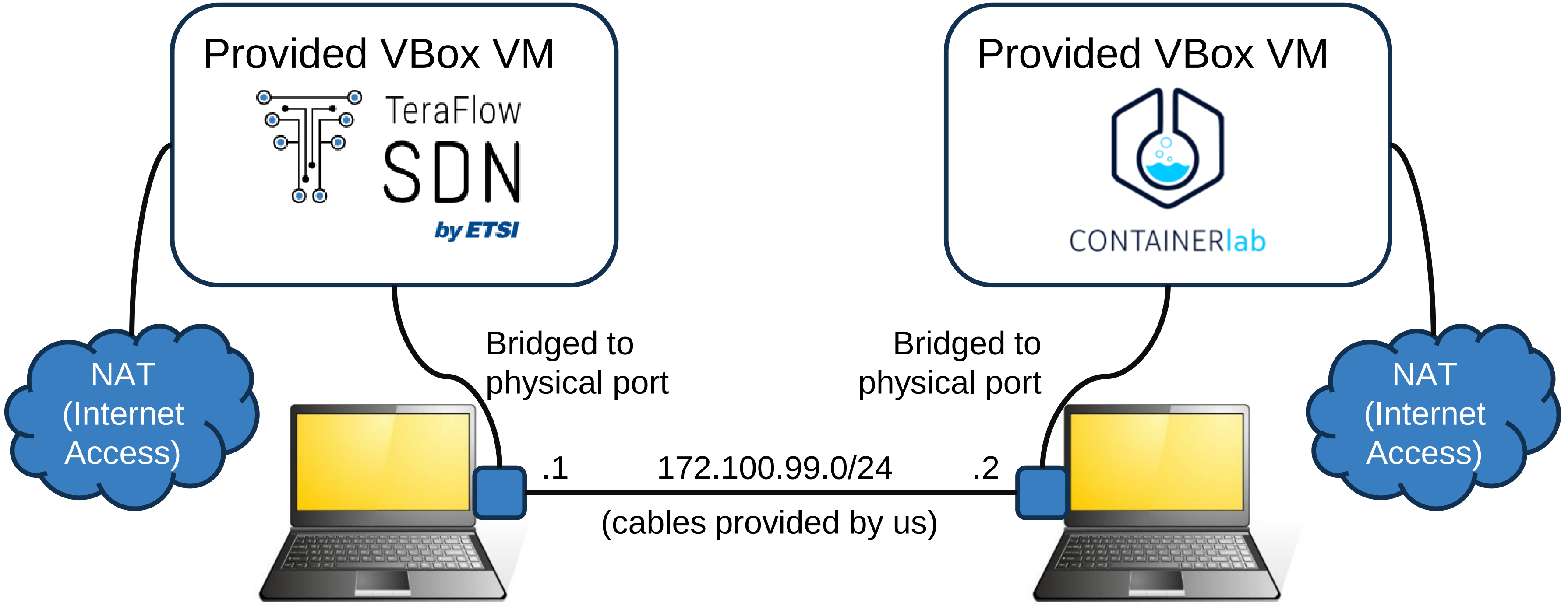
```
gnmic -a clab-srlinux-srl1 -u admin -p NokiaSrl1! --skip-verify -e json_ietf subscribe --path
/interface[name=mgmt0]/statistics
```

(In another terminal, you can generate traffic) `ssh admin@clab-srlinux-srl1`



Presentation of the challenges

Proposed mini-testbed per team



VM requirements:
- VirtualBox 6.1.40 or newer
- 4 vCPU
- 8 GB RAM
- 60 GB disk

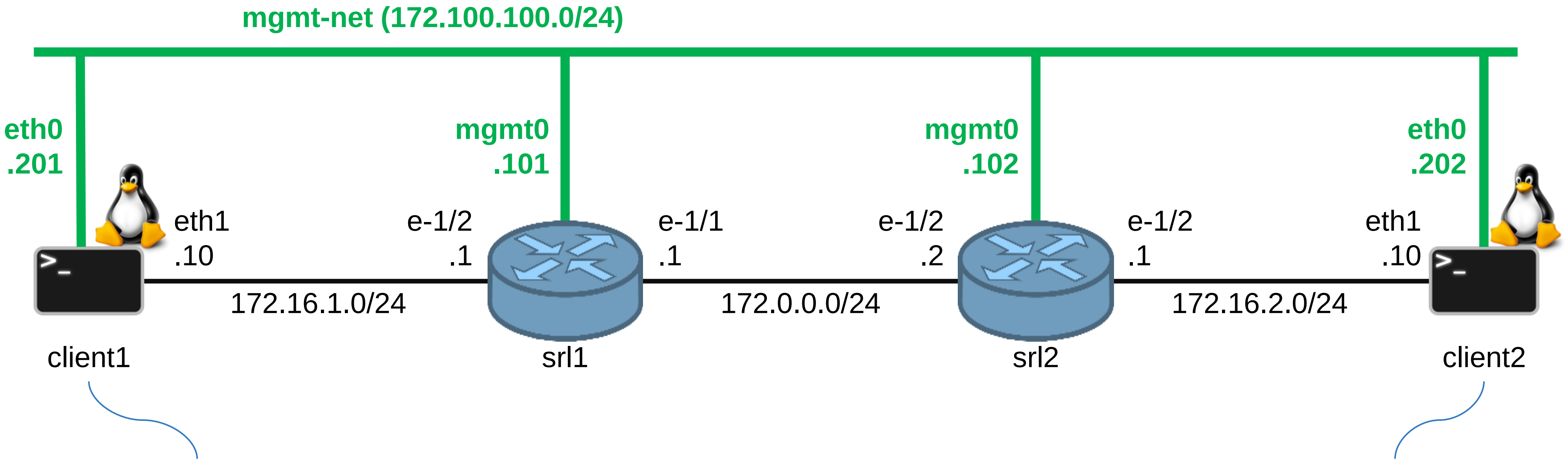
Also install on the host:
- VSCode
- Remote devel Ext. for VSCode
- MobaXterm or other SSH client

If you have a powerful laptop, you can try to run both VMs, but expect fan noise ☺.

Networking settings:
- Configure static IP/mask (e.g., 172.100.99.{1,2}/24)
- On TFS VM, configure route to ContainerLab VM, specifically, to its internal mgmt network:

    ip route add net 172.100.100.0/24 via 172.100.99.2

Target ContainerLab Scenario




```
ip address add 172.16.1.10/24 dev eth1
ip route add 172.16.2.0/24 via 172.16.1.1
```

```
ip address add 172.16.2.10/24 dev eth1
ip route add 172.16.1.0/24 via 172.16.2.1
```

Challenges

- **Have fun and learn in the meanwhile! ☺**
- Learn a bit on TeraFlowSDN and ContainerLab
    - Create your local testbed
        - Deploy TFS
        - Deploy Scenario in ContainerLab
    - Onboard the Devices from the ContainerLab Scenario in TFS
        - Some descriptors are provided as reference (~/tfs-ctrl/hackfest/containerlab)
    - Configure a Service with Static Routing in the devices
        - Some descriptors are provided as reference (~/tfs-ctrl/hackfest/containerlab)
    - Monitor the traffic in the device ports through Grafana

Useful notes

Many details are provided in file: ~/tfs-ctrl/hackfest/containerlab/commands.txt

- **IMPORTANT**: for Nokia SR Linux, use kind "srl" and type "ixr6“
    - Other types of hardware do not support OpenConfig or require a License
- **IMPORTANT**: Nokia SR Linux has **OpenConfig disabled by default**, to enable it, log into the SR CLI and enable it (see next slides).
- ContainerLab, gNMIc tool, TFS, etc. are already installed. In case of trouble, you might need to destroy and redeploy ContainerLab or TFS.
- Use gNMIc to test connectivity from TFS VM to Clab VM, and to inspect the data retrieved by SR Linux devices.

Manage ContainerLab Scenarios

```
# Deploy
cd ~/tfs-ctrl/hackfest/containerlab
$ sudo containerlab deploy --topo tfs-scenario.clab.yml

## Access SR Bash
$ docker exec -it clab-tfs-scenario-srl1 bash

## Access SR CLI (enables to get and set configs)
$ docker exec -it clab-tfs-scenario-srl1 sr_cli

# Destroy
$ sudo containerlab destroy --topo tfs-scenario.clab.yml
```

Warning:

User: admin
Pass: NokiaSrl1!

Activate OpenConfig in SR Linux

```
## Enable OpenConfig data models and set as default data model:

$ docker exec -it clab-tfs-scenario-srl1 sr_cli
  # enter candidate
  # system management openconfig admin-state enable
  # system gnmi-server network-instance mgmt yang-models openconfig
  # commit stay
  # quit
```

Retrieve Configurations of SR Linux with gNMIc

```
$ gnmic -a 172.100.100.101 -u admin -p NokiaSrl1! --skip-verify -e json_ietf get /
      --path '/network-instances' > srl1-nis.json
$ gnmic -a 172.100.100.101 -u admin -p NokiaSrl1! --skip-verify -e json_ietf get /
      --path '/interfaces' > srl1-ifs.json

$ gnmic -a 172.100.100.102 -u admin -p NokiaSrl1! --skip-verify -e json_ietf get /
      --path '/network-instances' > srl2-nis.json
$ gnmic -a 172.100.100.102 -u admin -p NokiaSrl1! --skip-verify -e json_ietf get /
      --path '/interfaces' > srl2-ifs.json
```

Configure ContainerLab clients

```
$ docker exec -it clab-tfs-scenario-client{1,2} bash
    ip address add 172.16.{1,2}.10/24 dev eth1
    ip route add 172.16.{2,1}.0/24 via 172.16.{1,2}.1

    ping 172.16.{2,1}.1  # test connectivity against remote router
    ping 172.16.{2,1}.10 # test connectivity against remote client
```

Too easy for you? Do you want more?

No problem! Here we go! ☺

- Extend the basic 2-router scenario with additional routers and clients
- Measure performance of ContainerLab using iperf
- Establish and monitor multiple parallel services
- Implement support in gNMI Driver for VLAN tags

Form the Teams

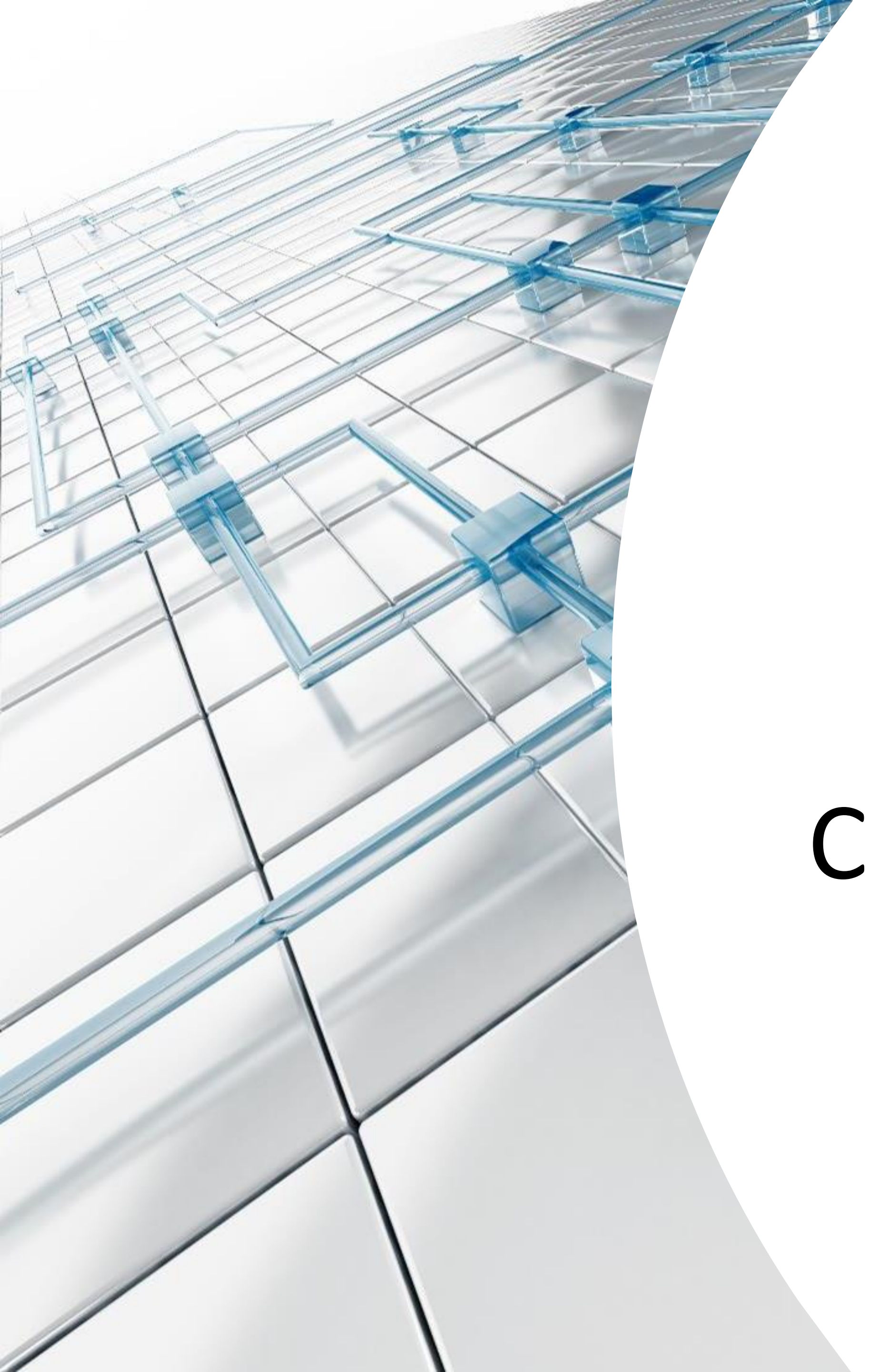
Form the Teams

Advises:

- Form 2-3 member teams
  - Try to have (at least) 1 member with previous experience with TeraFlowSDN.
- Find an awesome name for your team
- Come to the front desk to add your team to the table

Teams

Team #	Team Name	Member 1	Member 2	Member 3	Member 4
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					



Conclusion

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

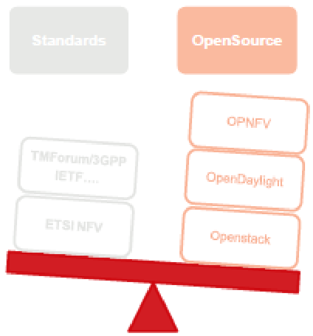
Standards summary

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

Standards Outcome/Benefits

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



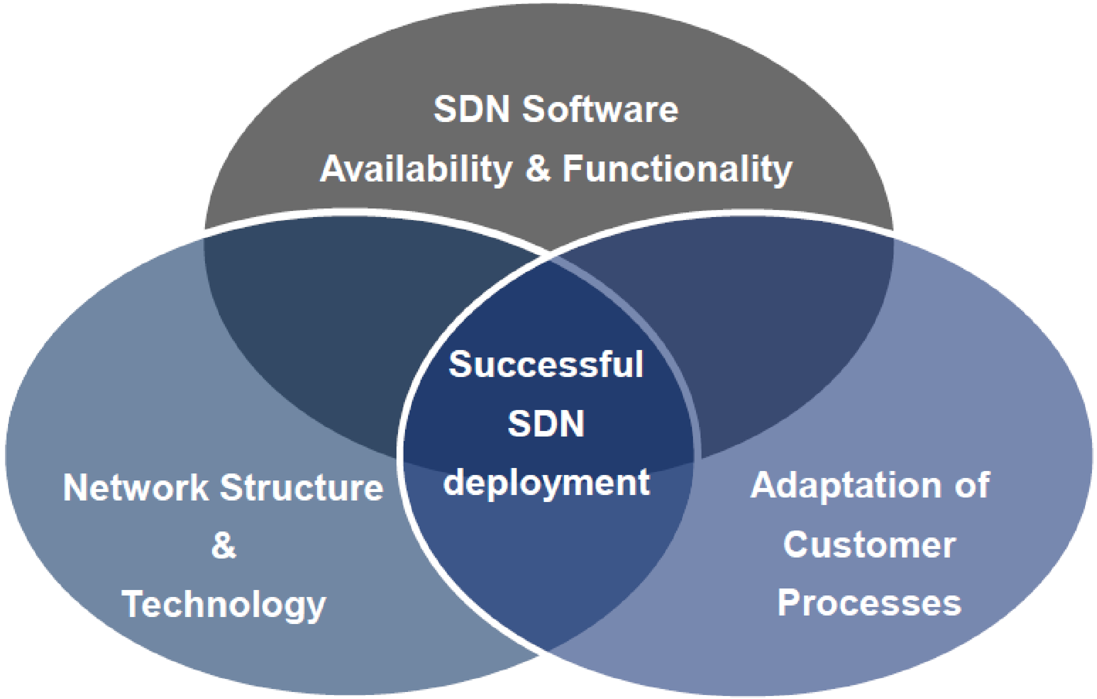
Open Source Outcome/Benefit

- 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…

TeraFlow SDN

- 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 Hackfest 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).