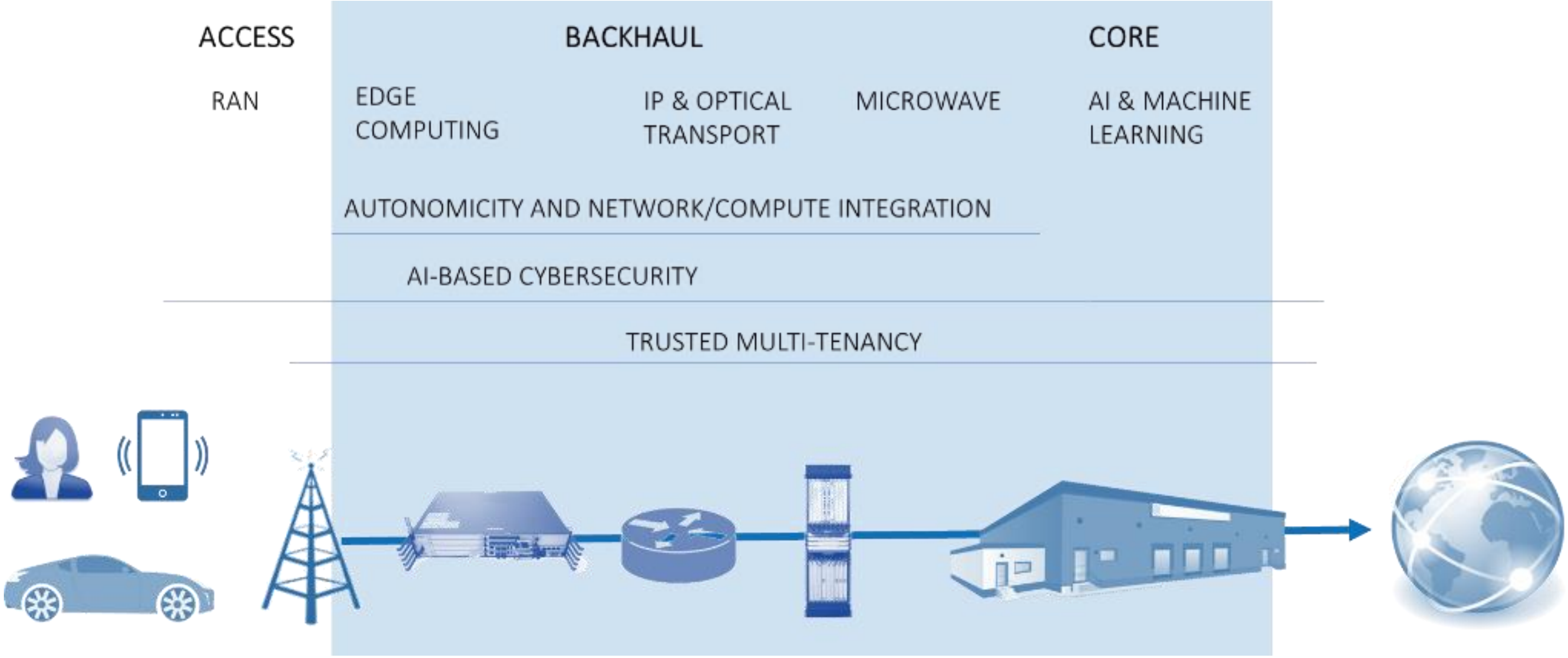
ETSI TeraFlowSDN

Networks have become more complex



ACCESS

BACKHAUL

CORE

RAN

EDGE COMPUTING

IP & OPTICAL TRANSPORT

MICROWAVE

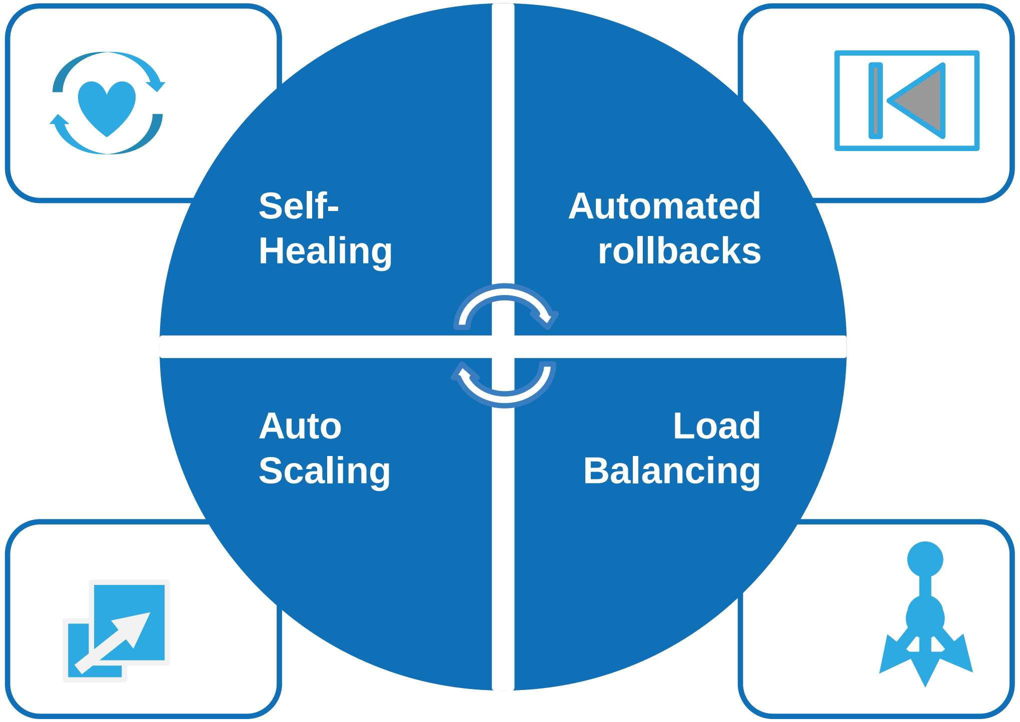
AI & MACHINE LEARNING

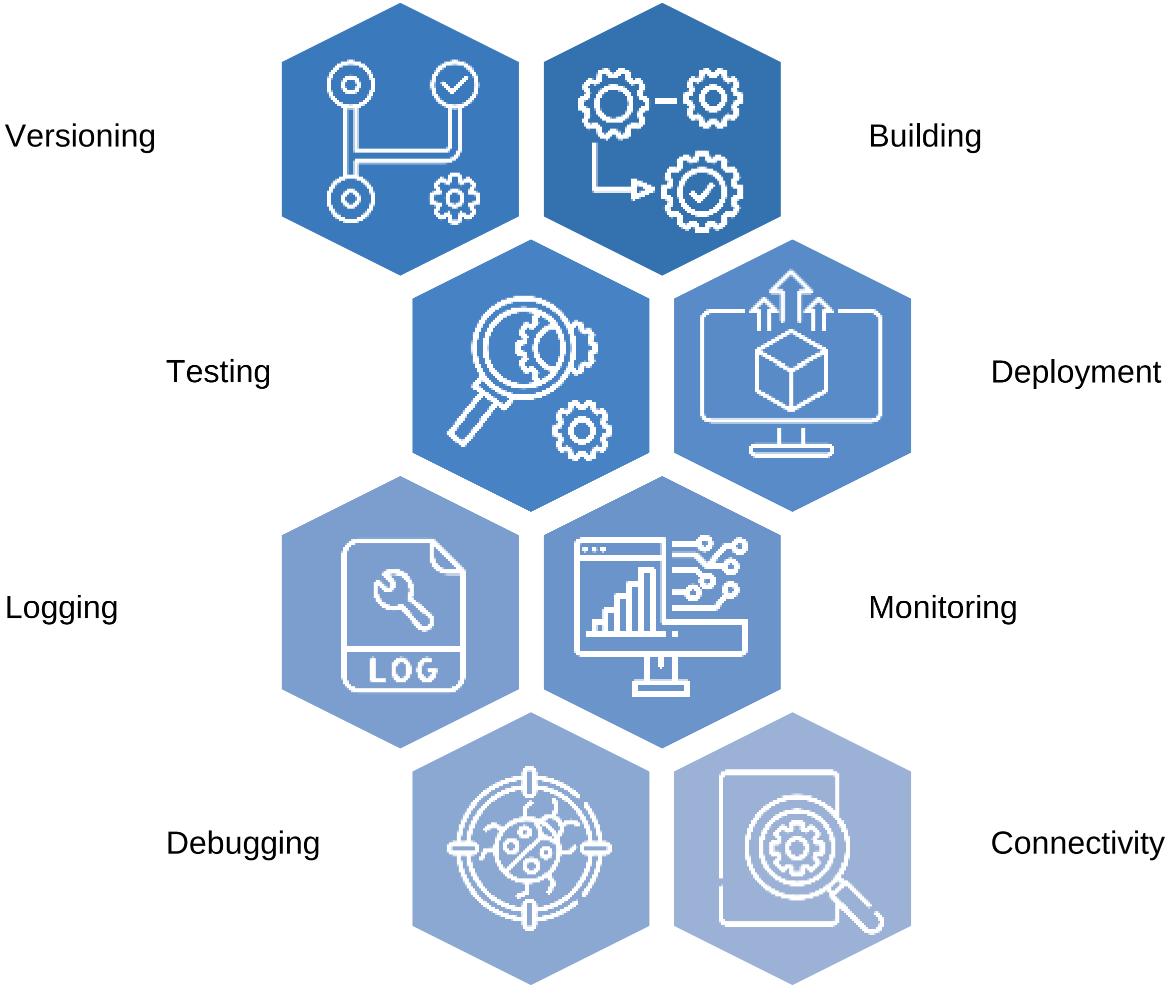
AUTONOMICITY AND NETWORK/COMPUTE INTEGRATION

AI-BASED CYBERSECURITY

TRUSTED MULTI-TENANCY

Benefits and Challenges of TFS cloud-native approach





ETSI TeraFlowSDN

- Open-source
- IP over DWDM
- Cloud-native
- Compliant with European Cyber-security requirements
- Community-driven
- Global community
- Focus on operator use cases
- Aligned with TIP MUST requirements

Who are we?

- Members

cttc Centre Tecnològic de Telecomunicacions de Catalunya

DEMOKRITOS

Fraunhofer HHI

POLITÉCNICA

SM siae microelettronica

Telefónica

telenor

- Participants (Non-ETSI members)

CHALMERS UNIVERSITY OF TECHNOLOGY

Globe

it instituto de telecomunicações

NTNU Norwegian University of Science and Technology

OLD DOG CONSULTING

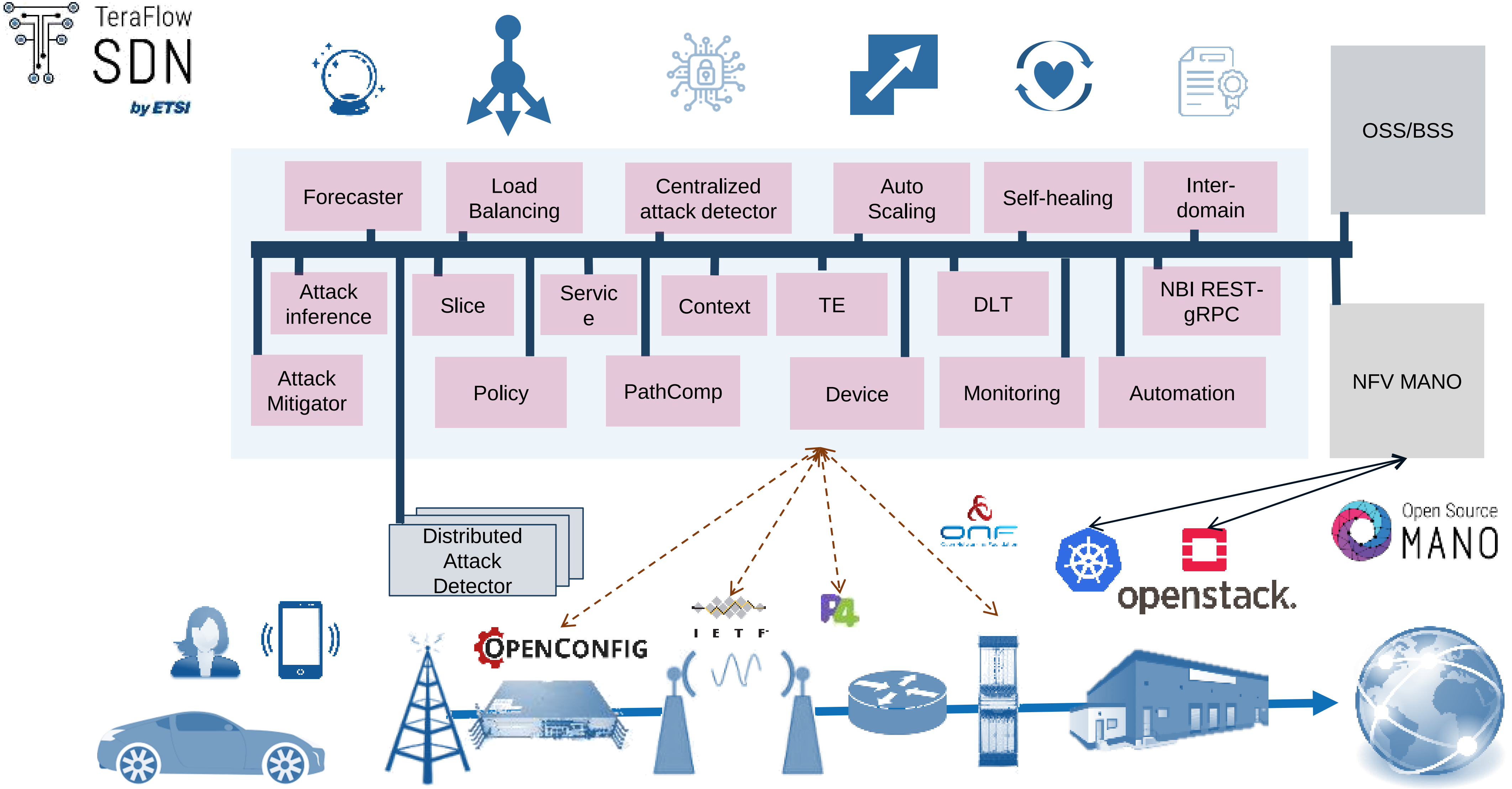
DIPL. PHYS. PEER STRITZINGER GMBH

UBITECH ubiquitous solutions

ETSI OSG TeraFlowSDN

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

TFS architecture for release 2



Deploy TeraFlowSDN controller

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

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

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

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

Deploy TeraFlowSDN controller

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

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

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

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

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

Deploy TeraFlowSDN controller

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

- Example: see “my_deploy.sh”

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

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

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

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

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

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

Deploy TeraFlowSDN controller

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

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

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

The deployment might take few minutes…

- You should see the progress of the deployment.

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

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

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

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

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

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

Deploy TeraFlowSDN controller

The process concludes reporting the status of the microservices.

You can always retrieve this status as follows:

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

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

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

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

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

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

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

Exercise: Onboard Devices using TeraFlowSDN

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

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

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

Exercise: Onboard Devices using TeraFlowSDN

Connect TFS to an OLS controller supporting TAPI

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

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

Check “src/common/DeviceTypes.py”

Set IP Address of your VM & port of TAPI server

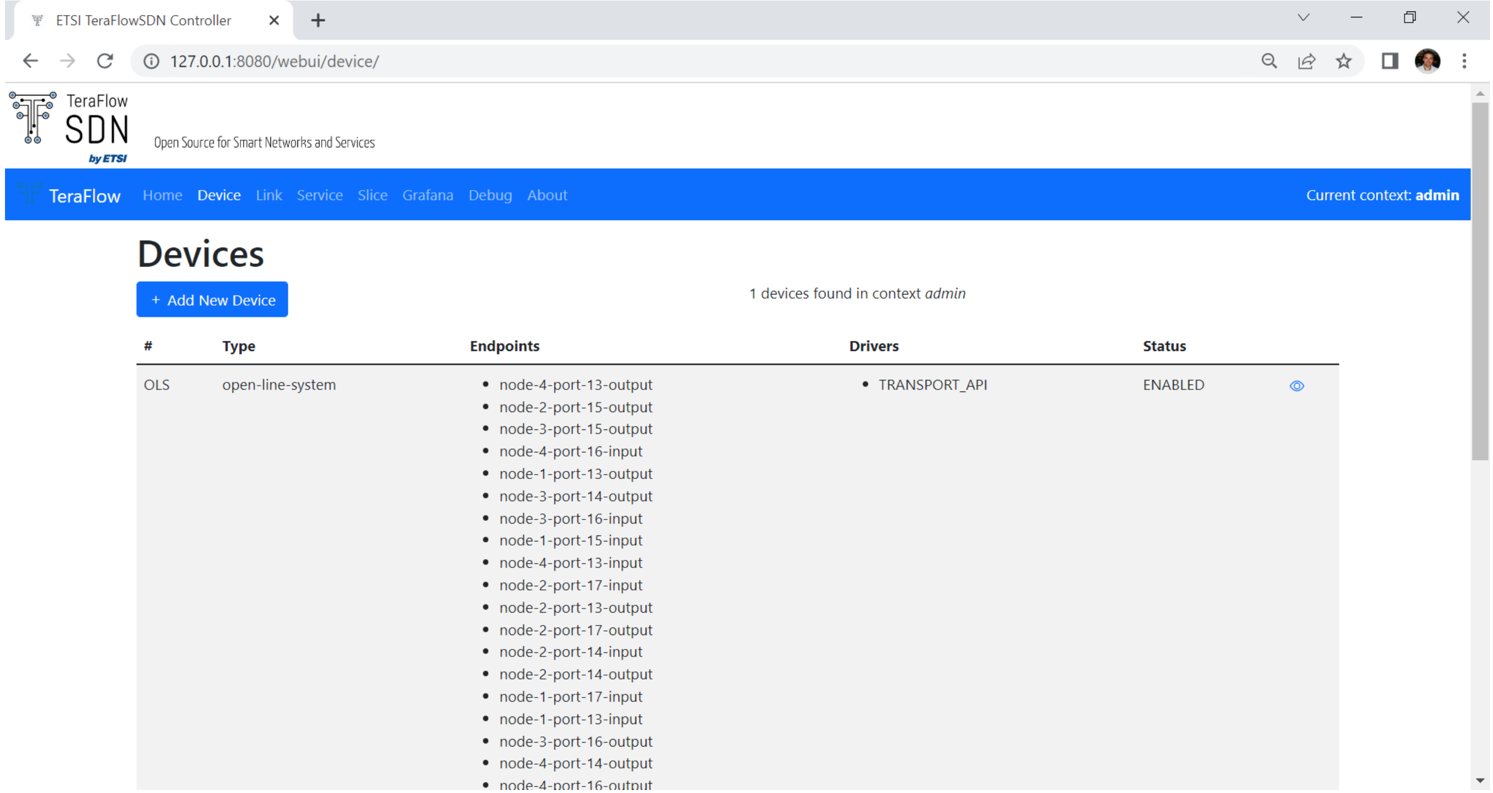
By default, DISABLED, Will be activated by Automation.
Check “proto/context.proto” “DeviceOperationalStatusEnum”

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

Automatically discovered from TAPI server by Device component.

Exercise: Onboard Devices using TeraFlowSDN

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



Devices

1 devices found in context *admin*

#	Type	Endpoints	Drivers	Status
OLS	open-line-system	node-4-port-13-output node-2-port-15-output node-3-port-15-output node-4-port-16-input node-1-port-13-output node-3-port-14-output node-3-port-16-input node-1-port-15-input node-4-port-13-input node-2-port-17-input node-2-port-13-output node-2-port-17-output node-2-port-14-input node-2-port-14-output node-1-port-17-input node-1-port-13-input node-3-port-16-output node-4-port-14-output node-4-port-16-output	TRANSPORT_API	ENABLED

Exercise: Onboard Devices using TeraFlowSDN

Check the logs of the TeraFlowSDN components:

- Example: Device component

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

Programmable L3 routers with OpenConfig

OpenConfig Projects

Data models

Models for common configuration and operational state across platforms

Streaming telemetry

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

RPCs and tools

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

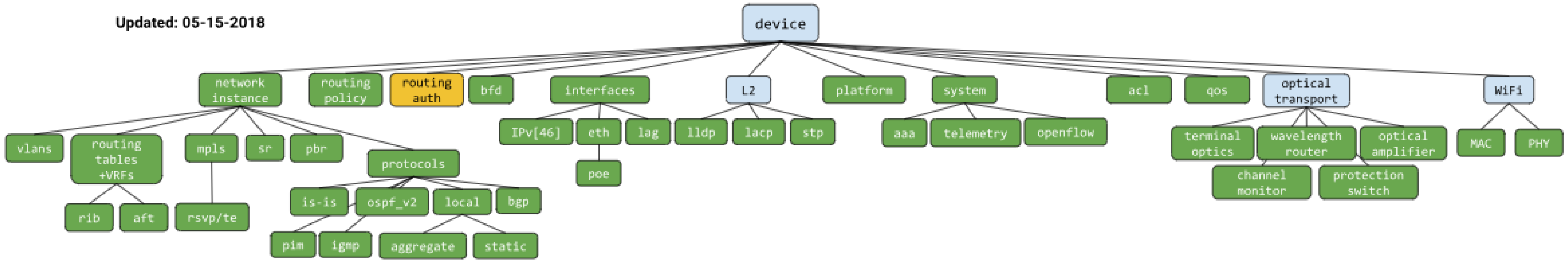
OpenConfig

Data models for configuration and operational state, written in YANG

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

Development priorities driven by operator requirements

Technical engagement with major vendors to deliver native implementations



Updated: 05-15-2018

OpenConfig Data Model Principles

Modular model definition

Model structure combines

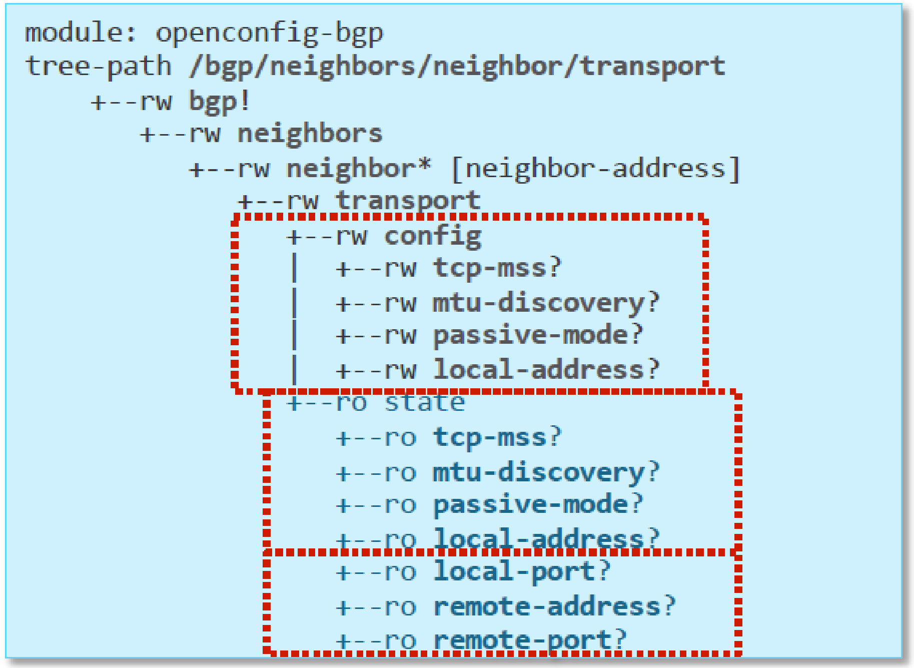
- Configuration (intended)
- Operational data (applied config and derived state)

Each module subtree declares config and state containers.

Model backward compatibility

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

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



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

OpenConfig L3 data models – Interfaces

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

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

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

OpenConfig L3 data models – Routing Policies

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

Examples (import & export BGP routing rules):

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

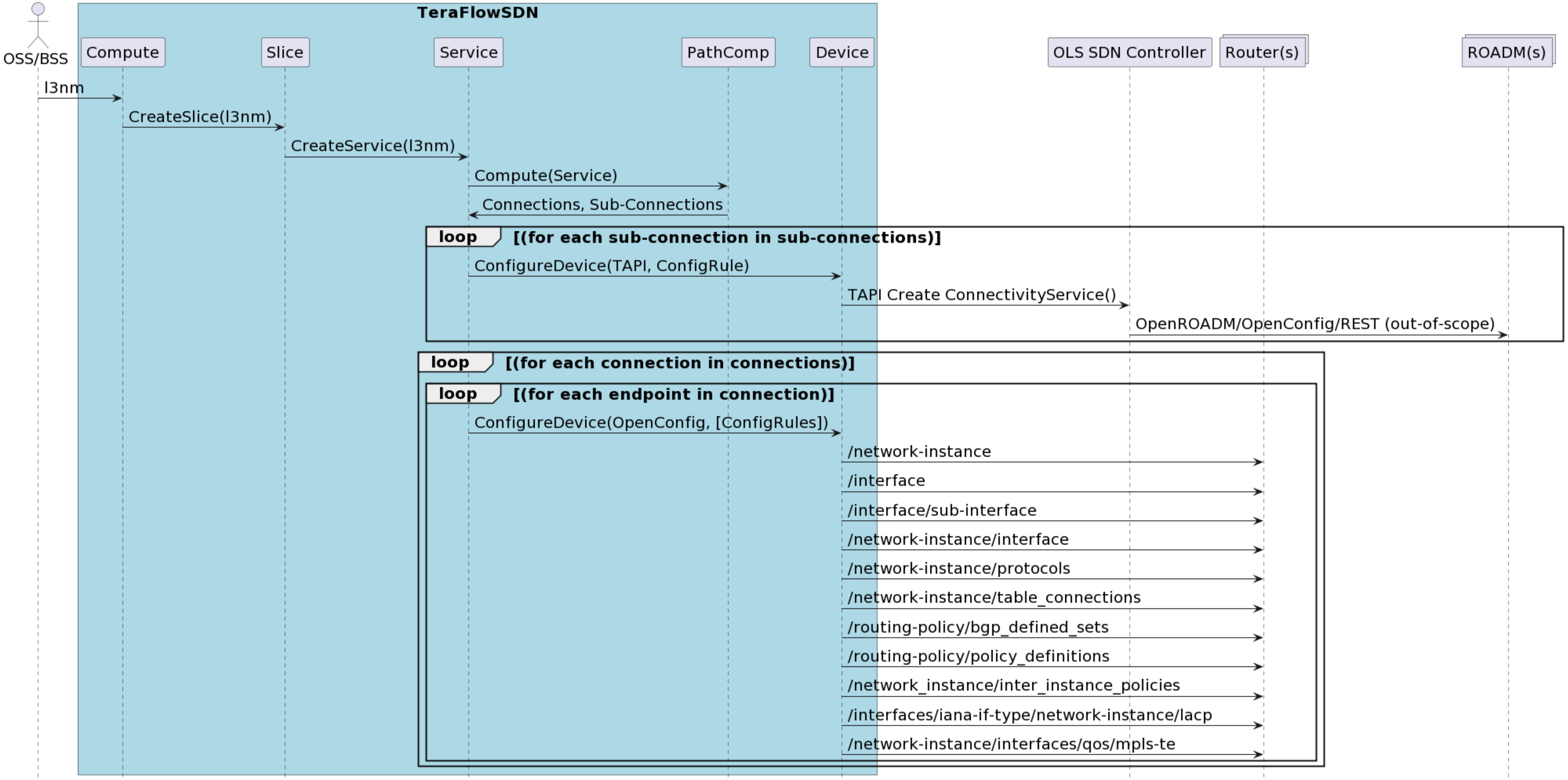
```
ext-community-set-name : my_net_inst_rt_export
ext-community-member   : route-target:65000:333
match-set-options      : oc-pol-types:ANY
policy-definition.name : my_net_inst_export
statement.name         : 3
install-protocol-eq    : oc-pol-types:DIRECTLY_CONNECTED
policy-result          : ACCEPT_ROUTE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

OpenConfig L3 data models – Network Instance

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

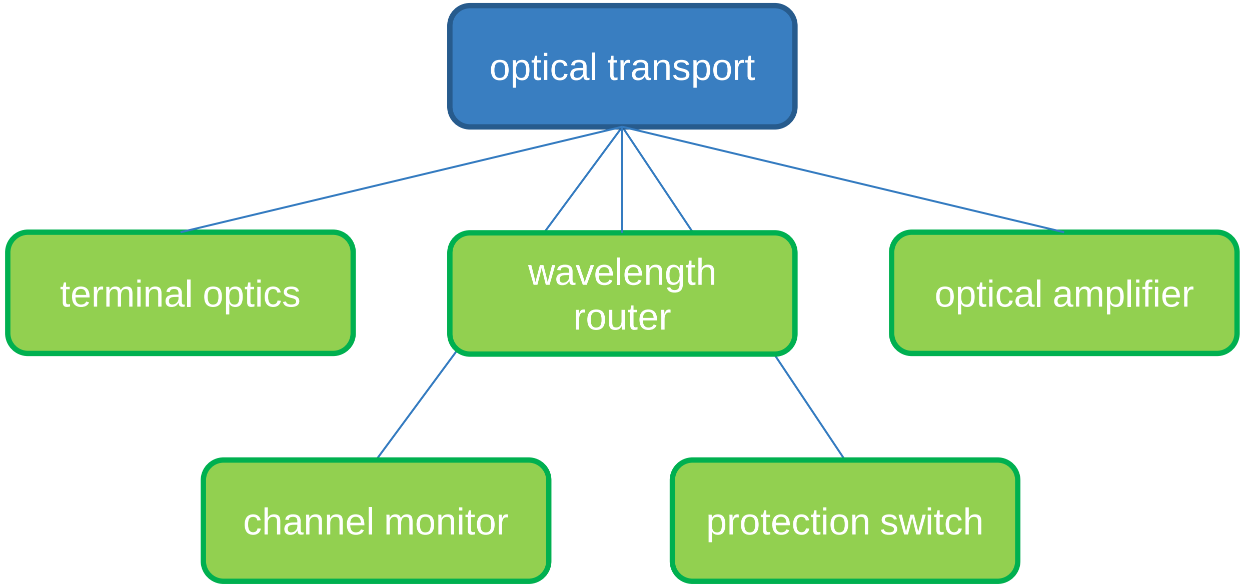
```
+--rw interfaces
|  +--rw interface* [id]
|     +--rw id        -> ../config/id
|     +--rw config
|     |  +--rw id?            string
|     |  +--rw interface?     -> /interfaces/interface/name
|     |  +--rw subinterface?  -> /interfaces/interface[…]/
|     ......                     subinterfaces/subinterface/index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

OpenConfig workflow MS2.2


TeraFlowSDN
OSS/BSS
Compute
Slice
Service
PathComp
Device
OLS SDN Controller
Router(s)
ROADM(s)
l3nm
CreateSlice(l3nm)
CreateService(l3nm)
Compute(Service)
Connections, Sub-Connections
loop [(for each sub-connection in sub-connections)]
ConfigureDevice(TAPI, ConfigRule)
TAPI Create ConnectivityService()
OpenROADM/OpenConfig/REST (out-of-scope)
loop [(for each connection in connections)]
loop [(for each endpoint in connection)]
ConfigureDevice(OpenConfig, [ConfigRules])
/network-instance
/interface
/interface/sub-interface
/network-instance/interface
/network-instance/protocols
/network-instance/table_connections
/routing-policy/bgp_defined_sets
/routing-policy/policy_definitions
/network_instance/inter_instance_policies
/interfaces/iana-if-type/network-instance/lacp
/network-instance/interfaces/qos/mpls-te

Optical-Transport

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



openconfig-terminal-device.yang

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

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

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

Vendor-supported operational modes. Example of possible info:

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

openconfig-terminal-device.yang (I)

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

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

openconfig-terminal-device.yang (II)

```
module: openconfig-terminal-device
    +--rw terminal-device
       +--rw config
       +--ro state
       +--rw logical-channels
       |  +--rw channel* [index]
       |     +--rw ethernet
       |     |  +--rw config
       |     |  +--ro state
       |     |     +--ro in-mac-control-frames?             oc-yang:counter64
       |     |     +--ro in-mac-pause-frames?               oc-yang:counter64
       |     |     +--ro in-oversize-frames?                oc-yang:counter64
       |     |     +--ro in-undersize-frames?               oc-yang:counter64
       |     |     +--ro in-jabber-frames?                  oc-yang:counter64
       |     |     +--ro in-fragment-frames?                oc-yang:counter64
       |     |     +--ro in-8021q-frames?                   oc-yang:counter64
       |     |     +--ro in-crc-errors?                     oc-yang:counter64
       |     |     +--ro in-block-errors?                   oc-yang:counter64
       |     |     +--ro out-mac-control-frames?            oc-yang:counter64
       |     |     +--ro out-mac-pause-frames?              oc-yang:counter64
       |     |     +--ro out-8021q-frames?                  oc-yang:counter64
       |     |     +--ro in-pcs-bip-errors?                 oc-yang:counter64
       |     |     +--ro in-pcs-errored-seconds?            oc-yang:counter64
       |     |     +--ro in-pcs-severely-errored-seconds?   oc-yang:counter64
       |     |     +--ro in-pcs-unavailable-seconds?        oc-yang:counter64
       |     |     +--ro out-pcs-bip-errors?                oc-yang:counter64
       |     |     +--ro out-crc-errors?                    oc-yang:counter64
       |     |     +--ro out-block-errors?                  oc-yang:counter64
```

```
     |     +--rw ingress
     |     |  +--rw config
     |     |  |  +--rw transceiver?        -> /oc-platform:components/component/name
     |     |  |  +--rw physical-channel*   -> /oc-platform:components/component/
     |     |  |                               oc-transceiver:transceiver/
     |     |  |                               physical-channels/channel/index
     |     |  +--ro state
     |     +--rw logical-channel-assignments
     |        +--rw assignment* [index]
     |           +--rw index     -> ../config/index
     |           +--rw config
     |           |  +--rw index?             uint32
     |           |  +--rw description?       string
     |           |  +--rw assignment-type?   enumeration
     |           |  +--rw logical-channel?   -> /terminal-device/logical-channels/
     |           |                              channel/index
     |           |  +--rw optical-channel?   -> /oc-platform:components/component/name
     |           |  +--rw allocation?        decimal64
     |           +--ro state (idem)
+--rw operational-modes
        +--ro mode* [mode-id]
           +--ro mode-id    -> ../state/mode-id
           +--ro config
           +--ro state
              +--ro mode-id?       uint16
              +--ro description?   string
              +--ro vendor-id?     string
```

openconfig-terminal-device.yang (III)

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

Pre-built Netconf/OpenConfig server

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

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

Start the server:

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

← Remember to set the listen port!

Exercise: Onboard Devices using TeraFlowSDN

TeraFlow SDN

Connect TFS to Netconf/OpenConfig devices using descriptor files

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

Check “src/common/DeviceTypes.py”

Set IP Address of your VM & port of Netconf server

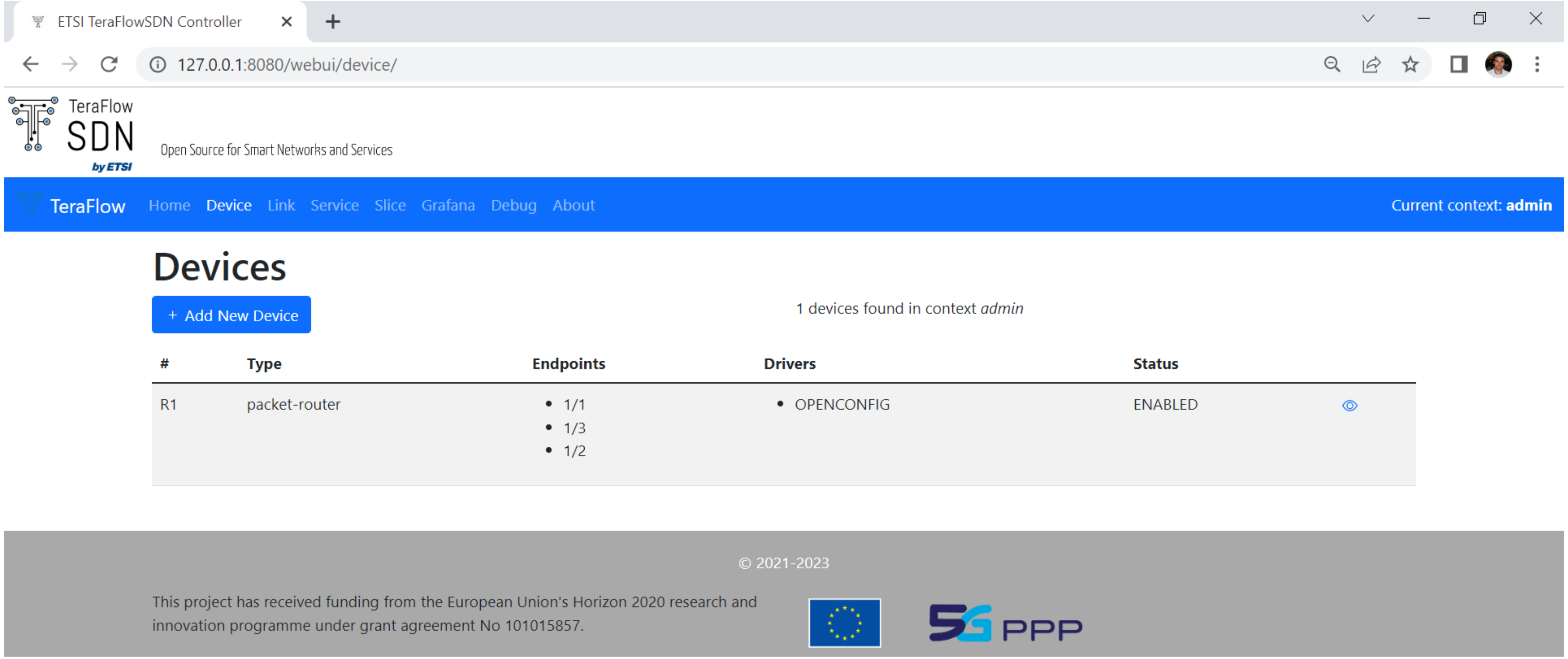
By default, DISABLED, Will be activated by Automation.
Check “proto/context.proto” “DeviceOperationalStatusEnum”

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

Automatically discovered from TAPI server by Device component.

Exercise: Onboard Devices using TeraFlowSDN

You should see the device in WebUI with the endpoints discovered



Devices

1 devices found in context *admin*

#	Type	Endpoints	Drivers	Status
R1	packet-router	• 1/1 • 1/3 • 1/2	• OPENCONFIG	ENABLED

© 2021-2023

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101015857.

Exercise: Onboard Devices using TeraFlowSDN

Check the logs of the TeraFlowSDN components:

- Example: Device component

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

Upload links and services

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

- Link Template:

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

Upload links and services

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

- Service Template:

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

Exercise: Establishing L3VPN service using TeraFlowSDN

Start 4 Netconf/OpenConfig servers

Start 1 TAPI OLS controller

Create & load descriptor files:

- all the devices
- links between endpoints as listed in ⇨

Issue service request

- service-l3vpn.json

Delete service through WebUI

Links:

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

Time: 20 min

Exercise: Establishing L3VPN service using TeraFlowSDN

Solution:

- Command to deploy servers

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

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

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

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

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

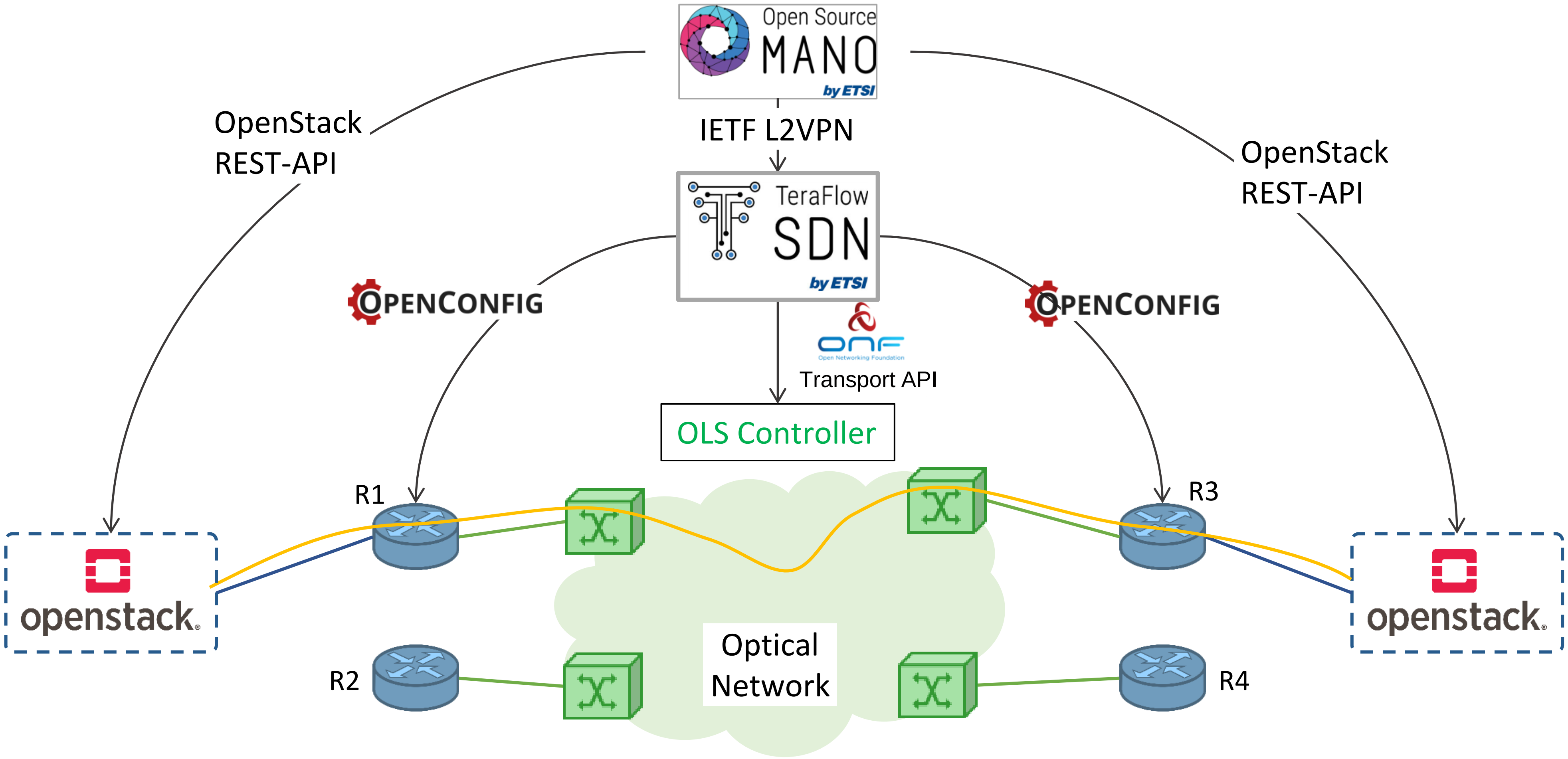
- Descriptor files can be found in

  ~/tfs-ctrl/hackfest/tfs-descriptors

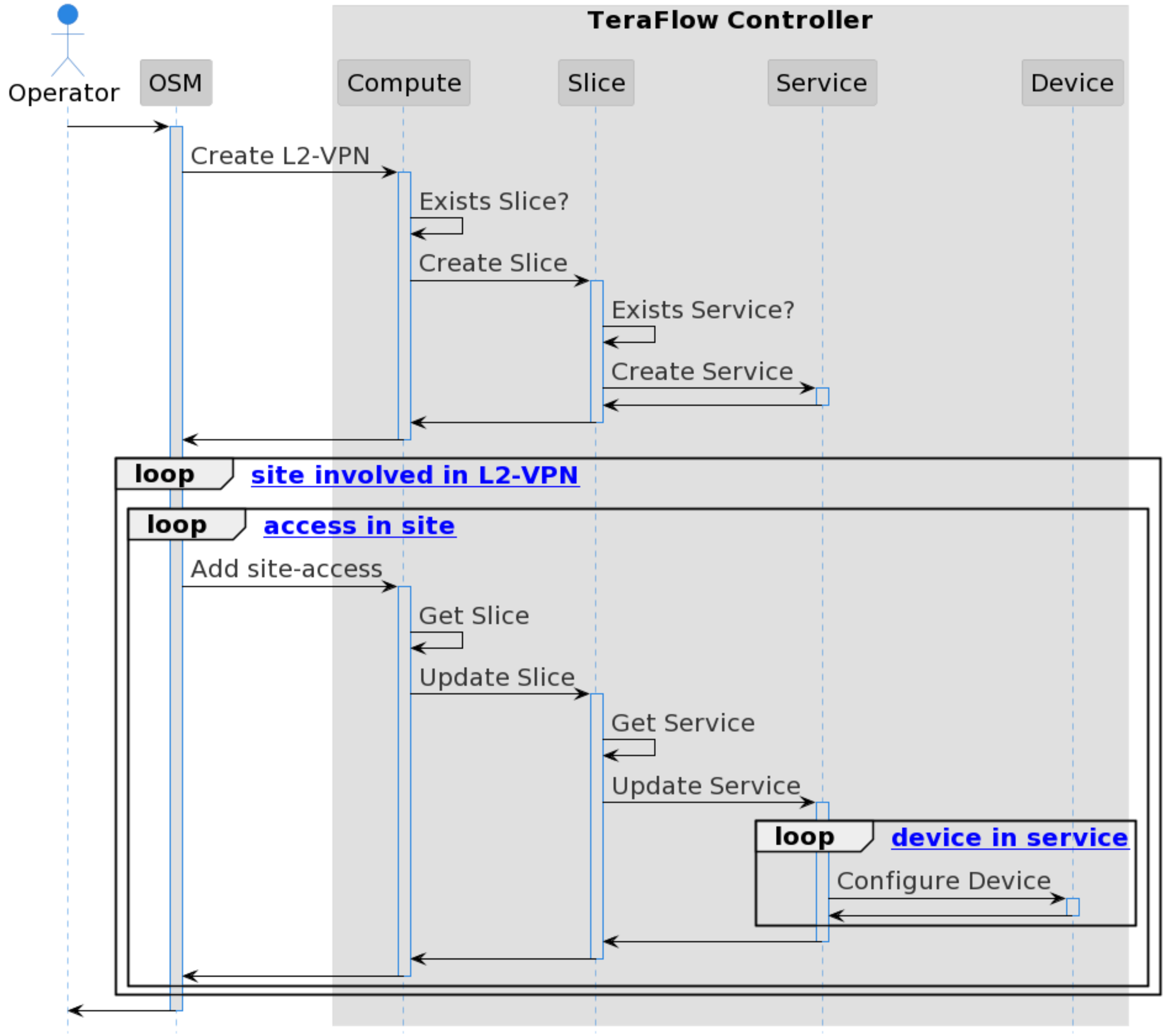
  - context-topology.json
  - device-all.json
  - links.json
  - service-l3vpn.json

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

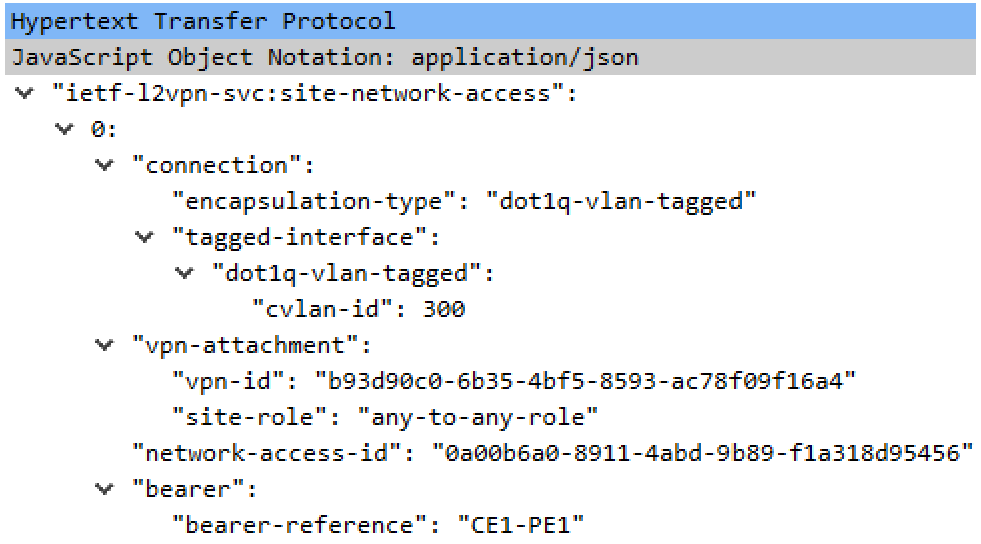
This exercise emulates the requests that ETSI OpenSourceMANO issues to underlying WAN Infrastructure Managers to how OSM



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



IETF L2VPN request from OSM (add site access)



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

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

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

Open Wireshark to see the request messages in detail.

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

- Tip: try “help” command.

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

Time: 10 min

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

Solution:

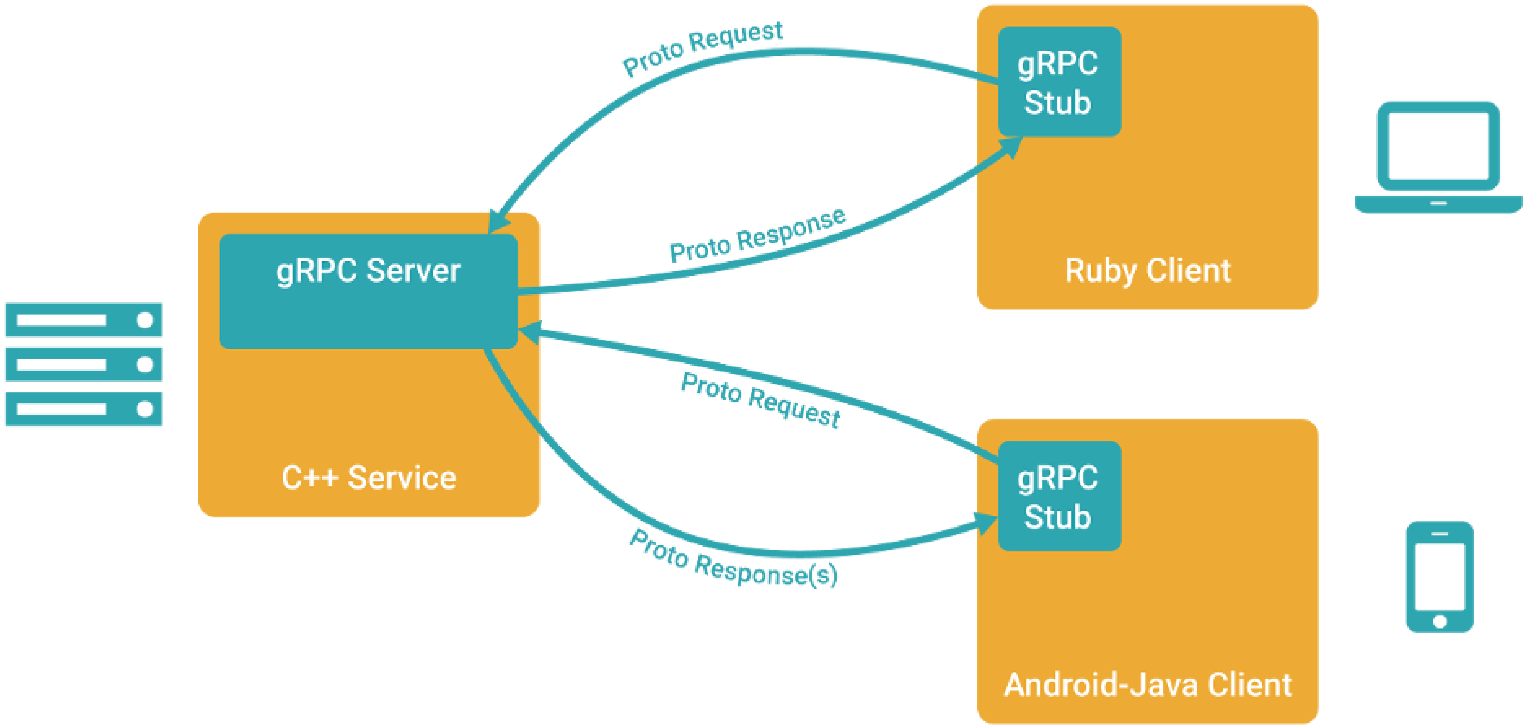
```
$ cd ~/tfs-ctrl/hackfest/
$ python -m mock_osm
Welcome to the MockOSM shell.
Type help or ? to list commands.

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

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

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

(mock-osm) exit
Bye!
```

Monitoring

What is gRPC

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

gRPC architecture



Source: https://grpc.io/

Protocol Buffers

Interface Definition Language (IDL)

- Describe once and generate interfaces for any language.

Data Model

- Structure of the request and response.

Describes Wire format

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

Compilation:

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

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

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

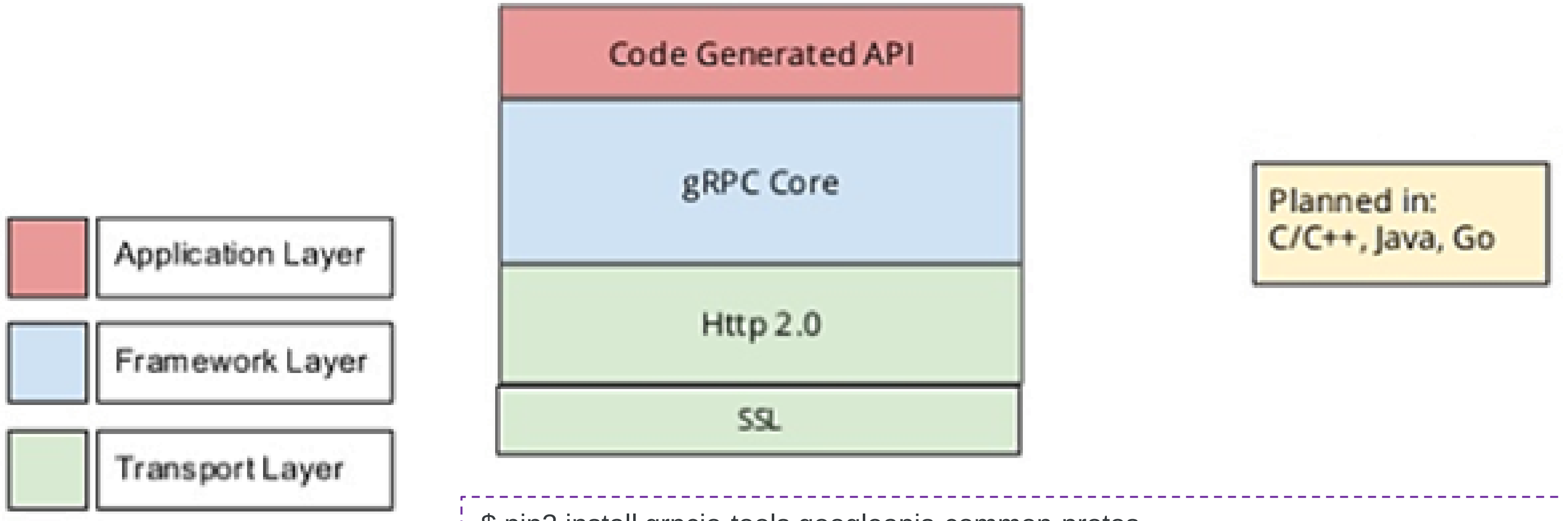
gRPC Main Use Cases and architecture

Efficiently connecting polyglot services in microservices style architecture

Connecting mobile devices, browser clients to backend services

Generating efficient client libraries

Low latency, highly scalable, distributed systems.



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

Usage of protobufs

Translate connection.yang to protobuf

Create a script that writes new connections to a file

Create a script that lists all stored connections from a file

You can use the following tutorial

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

Warning: Be “careful” with hyphens!

connection.proto

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

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

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

  LayerProtocolName layerProtocolName = 7;

}

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

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

Create Connection

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

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

…
```

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

  connectionList = connection_pb2.ConnectionList()

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

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

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

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

List Connection

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


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

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

  connectionList = connection_pb2.ConnectionList()

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

  ListConnections(connectionList)
```

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

Create a gRPC client/server

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
```

Exercise: gRPC streams

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

Use:

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

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

Time: 10min

Solution

Server

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

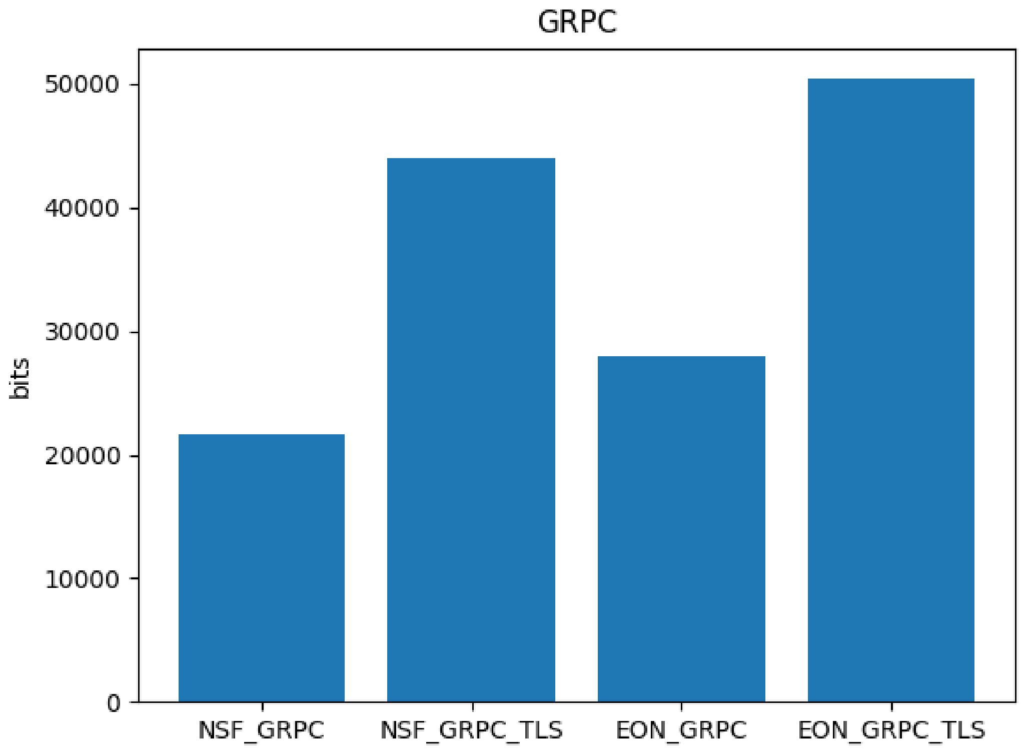
```
RUN SERVER
$ cd ~/tfs-ctrl/hackfest/grpc /connectionServiceWithNotif
$ python3 connectionServiceWithNotif_server.py
```

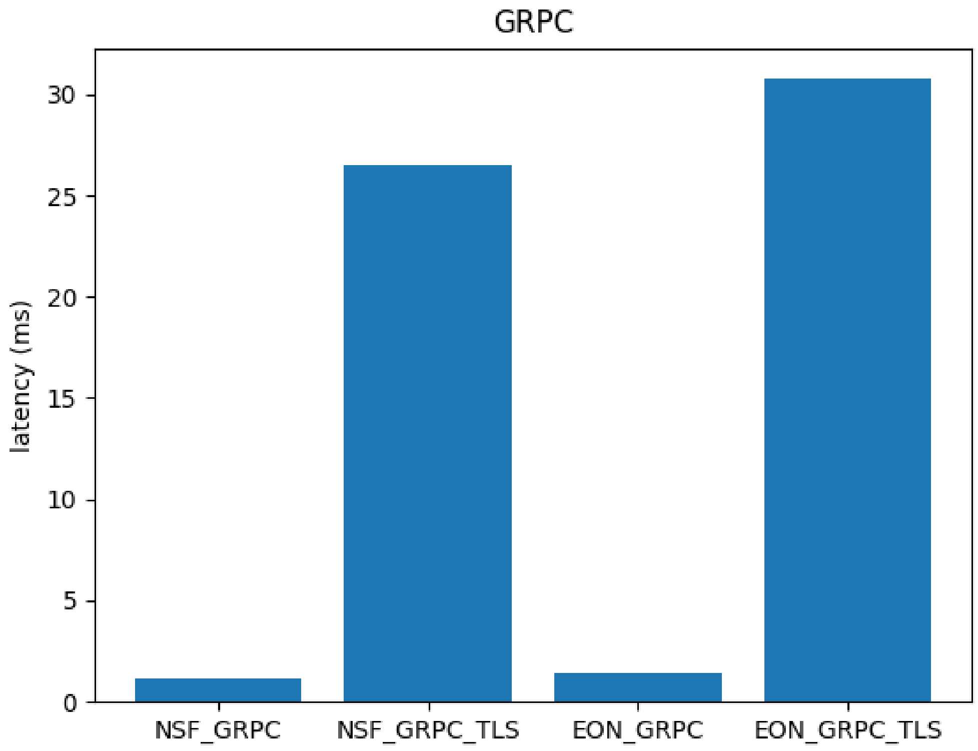
Client

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

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

Example gRPC results

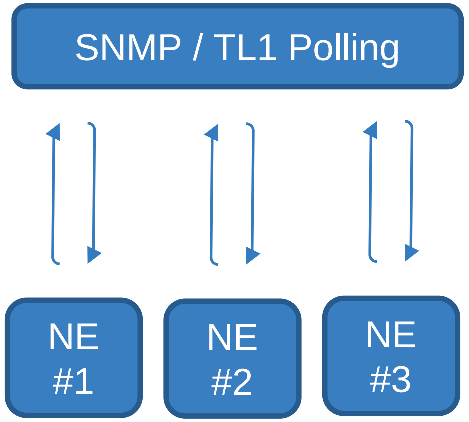




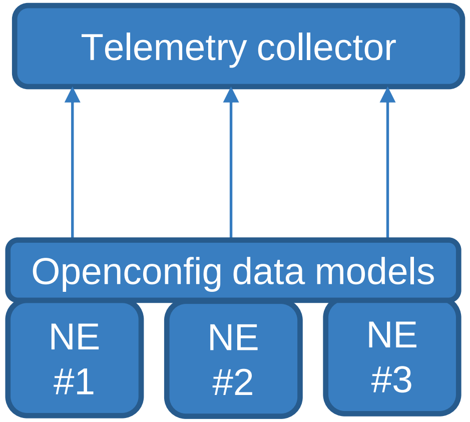
Better visibility with streaming telemetry

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

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



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



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

RPCs and gNMI

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

https://github.com/openconfig/gnmi

- This gNMI is described using Protobuf:

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

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

Why gNMI?

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

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

supports very efficient serialization and data access

- 3x-10x smaller than XML

offers an implemented alternative to NETCONF, RESTCONF, …

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

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

gNMI Terminology

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

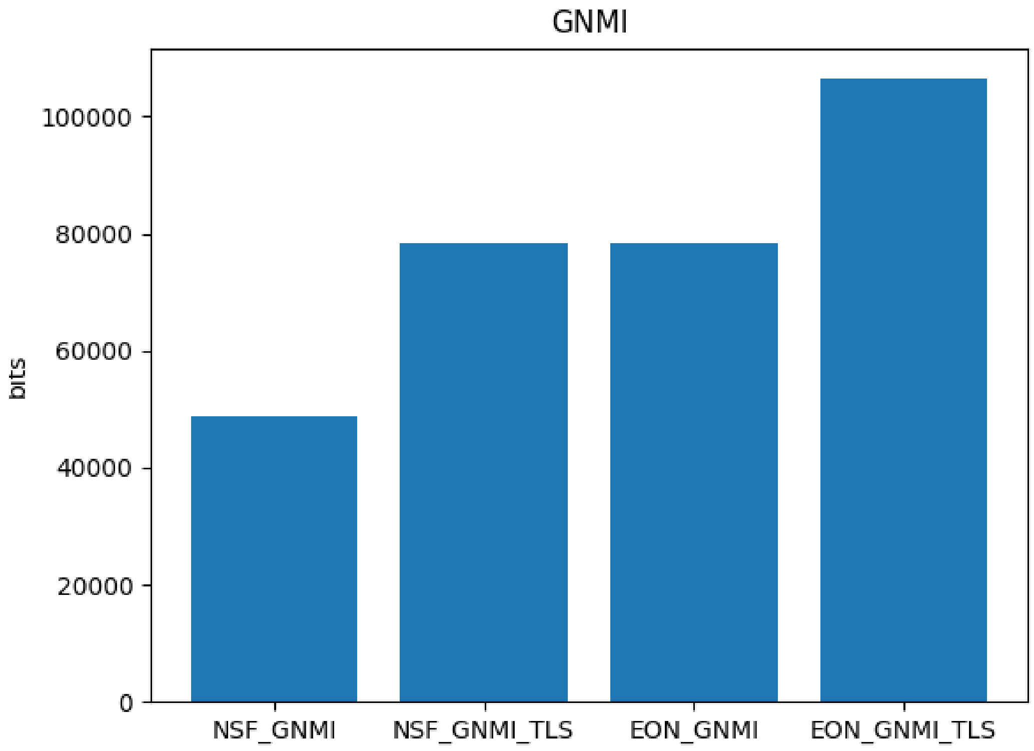
gNMI protocol buffer

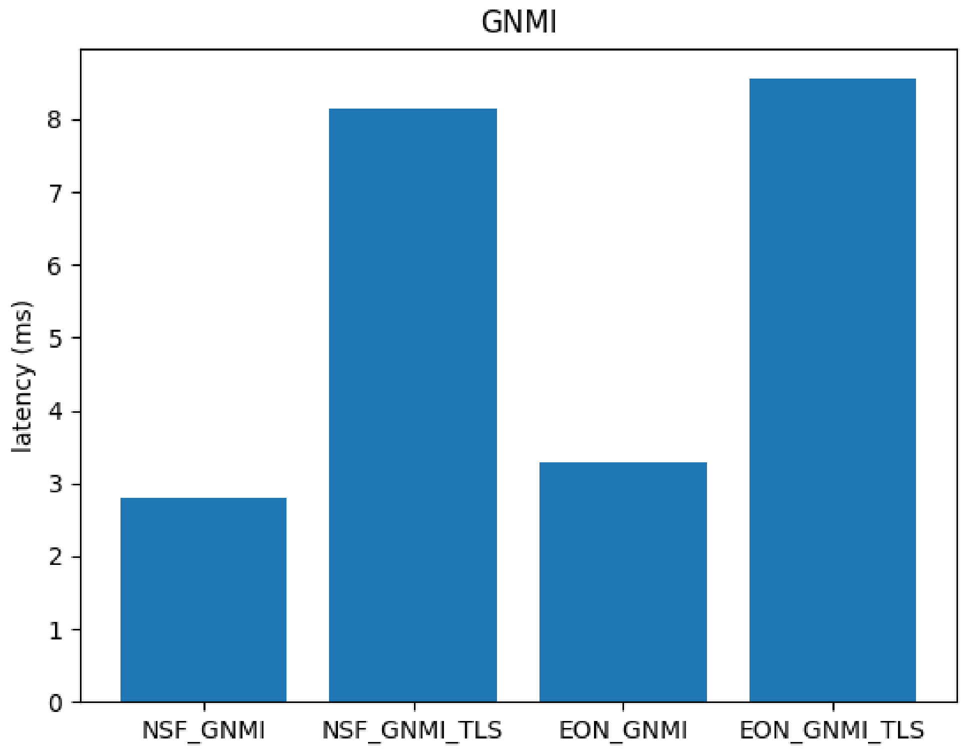
```
service gNMI {
  rpc Capabilities(CapabilityRequest) returns (CapabilityResponse);
  rpc Get(GetRequest) returns (GetResponse);
  rpc Set(SetRequest) returns (SetResponse);
  rpc Subscribe(stream SubscribeRequest) returns (stream SubscribeResponse);
}
```

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

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

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

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

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

Example gNMI results





TeraFlow SDN

by ETSI

Thank You!