NETCONF

vs

gNMI

what can I do with them?

21 June 2022

Tomáš Kubina

Orange Business Services

tomas.kubina@orange.com

orange™

Agenda

1. NETCONF & gNMI quick overview
2. Configuration operations with NETCONF vs gNMI (read, create, update, delete)
3. Operational data retrieval with NETCONF vs gNMI
4. Payload size & response time
5. Operational commands
6. Summary

Idea

- **Test NETCONF and gNMI using Python3 and latest packages implementing these protocols with different NOSes**

- **Tested NOSes – IOSXR, IOSXE, JUNOS**

NETCONF & gNMI overview

NETCONF protocol

- since 2006
- expose API for network device management
- push & retrieve of configuration data (get, create, update, delete)
- retrieve of state (operational data), including filtering specified on client side for selective retrieve
- configuration changes done in transactions
- based on paradigm of using RPCs encoded in XML messages

NETCONF protocol

- protocol messages transported over secure SSH channel (TLS, …)
- standard RPC calls: <get>, <get-config>, <edit-config>, <copy-config>, <delete-config>, <lock>, <unlock>, <close-session>, <kill-session>
- standard operations should be supported by vendor, better to check each time

NETCONF messages examples

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:1d08782f-8818-4600-a5b5-7e5ae6a46757">
  <nc:get>
    <nc:filter type="subtree">
      <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
        <interface-configuration>
          <interface-name>GigabitEthernet0/2/0/12.201</interface-name>
        </interface-configuration>
      </interface-configurations>
    </nc:filter>
  </nc:get>
</nc:rpc>

+++++

<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:1d08782f-8818-4600-a5b5-7e5ae6a46757" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
      <interface-configuration>
        <active>act</active>
        <interface-name>GigabitEthernet0/2/0/12.201</interface-name>
<---------- output omitted ---------->
      </interface-configuration>
    </interface-configurations>
  </data>
</rpc-reply>
```

gNMI protocol

- since 2015/2016
- based on generic gRPC framework
- calls designed especially for networking world => gNMI calls defined by using gRPC
- expose API for network device management
- push & retrieve of configuration data (get, create, update, delete)
- retrieve of state (operational) data

gNMI protocol

- configuration changes done in transactions
- RPCs messages encoded in “protobuf” format – binary, smaller and more effective than XML (json, bytes formats possible)
- protocol messages transported over HTTP/2 (w/o TLS)
- only standard RPC calls: Get, Set, Subscribe, Capabilities
- possible extensions for operational commands (ping, reset)

gNMI messages examples

```
gNMI request:
-------------------------------------------------
prefix {
}
path {
  origin: "Cisco-IOS-XR-ifmgr-cfg"
  elem {
    name: "interface-configurations"
  }
  elem {
    name: "interface-configuration"
    key {
      key: "active"
      value: "act"
    }
    key {
      key: "interface-name"
      value: "GigabitEthernet0/2/0/12.201"
    }
  }
}
```

```
gNMI response:   (output omitted)
-------------------------------------------------
notification {
  timestamp: 1654601762401932841
  prefix {
  }
  update {
    path {
      elem {
        name: "interface-configurations"
      }
      elem {
        name: "interface-configuration"
        key {
          key: "active"
          value: "act"
        }
        key {
          key: "interface-name"
          value: "GigabitEthernet0/2/0/12.201"
        }
      }
    }
    val {
      json_ietf_val: "[{\"active\":\"act\",\"interface-
name\":\"GigabitEthernet0/2/0/12.201\",\"interface-mode-non-
physical\":\"default\",\"description\":\"\\\"Link to P2 via
C3\",\"bandwidth\":\"1000000}]"
    }
  }
}
error {
}
```

Configuration manipulation

NETCONF protocol

- **Python “ncclient” used for testing**

- **default script wrapping all ncclient calls**

```
from ncclient import manager

# device type ‘iosxr’/’iosxe’/’junos’

session = manager.connect(host='192.168.243.21',
                          port=830,
                          username=‘username',
                          password=‘password',
                          hostkey_verify=False,
                          device_params={'name': 'iosxr'})

<code for individual rpcs>

session.close_session()
```

gNMI protocol

- **Python “pygnmi” used for testing**

- **default script wrapping all gNMI calls**

```
from pygnmi.client import gNMIclient
import json


host = ('192.168.243.69', '50001')

with gNMIclient(target=host, username=‘username', password=‘password', insecure=True, debug=0) as gc:

    <code for individual rpcs>

# no need to explicitly close session as done by python context manager
```

NETCONF protocol – GET full configuration

- **code:**

```
full_config = session.get_config(source='running').data_xml
print(full_config)
```

- **output: full configuration expressed by using all YANG models supported by current device model and OS version**

- **comments: - JUNOS full configuration available without any model, natively available**

gNMI protocol – GET full configuration

- **code:**

```
result = gc.get(path=[], encoding='json_ietf', datatype='config’)
print(json.dumps(result, indent=4))
```

- **output: full configuration expressed by using all YANG models supported by current device model and OS version**

- **comments: - IOSXE get full config not working**
  - JUNOS only ‘all’ datatype is working

NETCONF protocol – GET partial configuration

- **code:**

```
nc_request_config = """<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
                          <interface-configuration>
                            <interface-name>GigabitEthernet0/2/0/12.202</interface-name>
                          </interface-configuration>
                        </interface-configurations>"""

interface_config = session.get(('subtree', nc_request_config)).data_xml
print(interface_config)
```

- **output: full configuration of element in “filter” using specific YANG model supported by current device model and OS version**

- **comments: - working on all platforms**

gNMI protocol – GET partial configuration

- **code:**

```
result = gc.get(path=["/Cisco-IOS-XR-ifmgr-cfg:interface-configurations/interface-configuration[active=act][interface-name=GigabitEthernet0/0/0/1.789]"], encoding='json_ietf')
print(json.dumps(result, indent=4))
```

- **output: full configuration of element in “filter” using specific YANG model supported by current device model and OS version**

- **comments: - on JUNOS not working**

NETCONF protocol – CREATE new config

- **code:**

```
nc_set_config = """<config>
                   <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
                    <interface-configuration>
                    <active>act</active>
                     <interface-name>GigabitEthernet0/2/0/12.202</interface-name>
                     <interface-mode-non-physical>default</interface-mode-non-physical>
                     <description>"TEST LINK</description>
                     <bandwidth>1000000</bandwidth>
                     <ipv4-network xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
                      <addresses>
                       <primary>
                        <address>10.10.2.2</address>
                        <netmask>255.255.255.252</netmask>
                       </primary>
                      </addresses>
                     </ipv4-network>
                     <vlan-sub-configuration xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-l2-eth-infra-cfg">
                      <vlan-identifier>
                       <vlan-type>vlan-type-dot1q</vlan-type>
                       <first-tag>202</first-tag>
                      </vlan-identifier>
                     </vlan-sub-configuration>
                    </interface-configuration>
                   </interface-configurations>
                   </config>"""

response = session.edit_config(nc_set_config)
response = session.commit()
```

NETCONF protocol – CREATE new config

- **output:**

```
<?xml version="1.0"?>
<rpc-reply message-id="urn:uuid:5801c1ef-120b-4a11-890f-252d84e98fe0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <ok/>
</rpc-reply>
```

- **comments: - working on all platforms**
  - on JUNOS and IOSXR ‘commit’ needed!

gNMI protocol – CREATE new config

- **code:**

```
new_subinterface_path_oc = "interfaces/interface[name=Loopback222]"
new_subinterface_config_oc = {
                              "config": {
                                  "name": "Loopback222",
                                  "type": "iana-if-type:softwareLoopback",
                                  "enabled": True
                              },
                              "subinterfaces": {
                                  "subinterface": [
                                      {
                                          "index": 0,
                                          "config": {
                                              "index": 0,
                                              "enabled": True
                                          },
                                          "openconfig-if-ip:ipv4": {
                                              "addresses": {
                                                  "address": [
                                                      {
                                                          "ip": "8.8.8.8",
                                                          "config": {
                                                              "ip": "8.8.8.8",
                                                              "prefix-length": 32
                                                          }
                                                      }
                                                  ]
                                              },
                                              "proxy-arp": {
                                                  "config": {
                                                      "mode": "ALL"
                                                  }
                                              }
                                          }
                                      }
                                  ]
                              }
                             }
```

gNMI protocol – CREATE new config

- **code:**

```
update_message = [(new_subinterface_path_oc, new_subinterface_config_oc)]

result = gc.set(update=update_message, replace=None, delete=None)
print(json.dumps(result, indent=4))
```

- **comments:** - on JUNOS not working
  - on IOSXE limits of openconfig implementation
  - on IOSXR not working

NETCONF protocol – UPDATE existing configuration

- **code:**

```
nc_set_config = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
                    <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
                     <interface-configuration>
                      <active>act</active>
                      <interface-name>GigabitEthernet0/2/0/12.202</interface-name>
                      <shutdown/>
                     </interface-configuration>
                    </interface-configurations>
                   </config>"""

response = session.edit_config(nc_set_config)
response = session.commit()
```

- **output: rpc-reply with OK status hopefully**

- **comments: - working on all platforms**
  - on JUNOS and IOSXR ‘commit’ needed!

gNMI protocol – UPDATE existing configuration

- **code:**

```
upd_subinterface_path_oc = "interfaces/interface[name=Loopback222]/subinterfaces/subinterface[index=0]"
upd_subinterface_config_oc = {"config": {
                                    "enabled": False
                                 }
                                }

update_message = [(upd_subinterface_path_oc, upd_subinterface_config_oc)]
result = gc.set(update=update_message, replace=None, delete=None)
print(json.dumps(result, indent=4))
```

- **output:**

- **comments: - on JUNOS not working**
  - on IOSXE limits of openconfig implementation
  - on IOSXR not working

NETCONF protocol – DELETE existing configuration

- **code:**

```
nc_set_config = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
                    <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
                     <interface-configuration xc:operation="delete">
                      <active>act</active>
                      <interface-name>GigabitEthernet0/2/0/12.202</interface-name>
                     </interface-configuration>
                    </interface-configurations>
                   </config>"""

response = session.edit_config(nc_set_config)
response = session.commit()
```

- **output: rpc-reply with OK status hopefully**

- **comments: - working on all platforms**
  - on JUNOS and IOSXR ‘commit’ needed!

gNMI protocol – DELETE existing configuration

- **code:**

```
delete_path = "interfaces/interface[name=Loopback222]/subinterfaces/subinterface[index=0]/ipv4/addresses“
result = gc.set(update=None, replace=None, delete=[delete_path])
print(json.dumps(result, indent=4))
```

- **output:**

- **comments:** - on JUNOS not working
  - on IOSXE ok, but limits of openconfig implementation
  - on IOSXR ok

Operational data retrieval

NETCONF protocol – GET operational state

- **code:**

```
nc_request_oper = """<interface-properties xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-oper">
                        <data-nodes>
                          <data-node>
                            <system-view>
                              <interfaces>
                                <interface>
                                  <interface-name>GigabitEthernet0/2/0/12.202</interface-name>
                                </interface>
                              </interfaces>
                             </system-view>
                           </data-node>
                         </data-nodes>
                      </interface-properties>"""

response = session.get(('subtree', nc_request_oper)).data_xml
print(response)
```

NETCONF protocol – GET operational state

- **output:**

```
<data-node>
     <data-node-name>0/RSP0/CPU0</data-node-name>
     <system-view>
      <interfaces>
       <interface>
        <interface-name>GigabitEthernet0/2/0/12.202</interface-name>
        <interface>GigabitEthernet0/2/0/12.202</interface>
        <parent-interface>GigabitEthernet0/2/0/12</parent-interface>
        <type>IFT_VLAN_SUBIF</type>
        <state>im-state-admin-down</state>
        <actual-state>im-state-admin-down</actual-state>
        <line-state>im-state-admin-down</line-state>
        <actual-line-state>im-state-admin-down</actual-line-state>
        <encapsulation>dot1q</encapsulation>
        <encapsulation-type-string>802.1Q</encapsulation-type-string>
        <mtu>1618</mtu>
        <sub-interface-mtu-overhead>0</sub-interface-mtu-overhead>
        <l2-transport>false</l2-transport>
        <bandwidth>1000000</bandwidth>
        <bandwidth64-bit>1000000</bandwidth64-bit>
       </interface>
      </interfaces>
     </system-view>
    </data-node>
```

- **comments: - on JUNOS use openconfig or custom RPC for each show command**
  - on IOSXE or IOSXR use openconfig or native oper models

•

gNMI protocol – GET operational state

- **code:**

```
result = gc.get(path=['interfaces/interface[name=GigabitEthernet2/0/2]/subinterfaces/subinterface[index=0]'],
encoding='json_ietf', datatype='all’)

print(json.dumps(result, indent=4))
```

- **comments: - on JUNOS only via get of all data**
  - on IOSXE or IOSXR use openconfig or native oper models

gNMI protocol – GET operational state

- **output:**


```
{
    "path": "interfaces/interface[name=GigabitEthernet2/0/2]/subinterfaces/subinterface[index=0]",
    "val": {
        "index": 0,
        "config": {
            "index": 0,
            "description": “R1_link_R2",
            "enabled": false
        },
        "state": {
            "description": “R1_link_R2",
            "enabled": false,
            "name": "GigabitEthernet2/0/2",
            "ifindex": 23,
            "admin-status": "DOWN",
            "oper-status": "DOWN",
            "last-change": "1654274225687000000",
            "counters": {
                "in-octets": "54492",
                "in-unicast-pkts": "89",
                "in-broadcast-pkts": "1",
                "in-multicast-pkts": "67",
                "in-discards": "0",
                "in-errors": "0",
                "in-unknown-protos": "0",
                "in-fcs-errors": "0",
                "out-octets": "49470",
                "out-unicast-pkts": "54",
                "out-broadcast-pkts": "1",
                "out-multicast-pkts": "51",
                "out-discards": "0",
                "out-errors": "0",
                "last-clear": "1654085274000000000"
            }
        }
```

Payload size & response time

Payload size & response time

- gNMI data 1.5-4x smaller then same data in NETCONF
  (tested payload - one interface / subinterface config config and op)

- bigger payload in gNMI > more efficient

- response time of gNMI server side 1.5-3x faster then NETCONF

Operational commands

Operational commands

- custom RPC calls in NETCONF – JunOS show commands, IOSXR/XE – clear <>, reload, reset <>, …

- custom RPC calls in gNMI are complicated – need for update of specs and proto file distribution by vendor

Summary

Summary

	NETCONF	gNMI
Get configuration	✓	✓
Filtering	✓	!
Create/Update/Delete	✓	✗
Get operational data	✓	✓
Response time	!	✓
BW requirements	!	✓
Non standard RPCs	✓	✗

Summary

- ncclient 0.6.9
- pygnmi 0.6.9
- IOSXR 7.4.2
- IOSXE 17.6.2
- JUNOS 21.2R3.8
- Examples and slides available at https://github.com/tomaskubina/csnog2022

Thank you

Resources & tips for study

- https://github.com/ncclient/ncclient
- https://github.com/akarneliuk/pygnmi
- https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md
- https://datatracker.ietf.org/doc/html/rfc6241
- https://pubhub.devnetcloud.com/media/netdevops-live/site/files/s01t03.pdf

Resources & tips for study

- https://yangcatalog.org/home.html
- https://yangcatalog.org/yang-search/module_details - MUST USE FOR MODEL BROWSING!
- https://github.com/cisco-ie/cisco-gnmi-python
- https://gnmic.kmrd.dev/
- https://events19.linuxfoundation.org/wp-content/uploads/2017/12/Beyond-the-Command-Line-Programming-Network-Devices-with-gRPC-and-OpenConfig-Nicolas-Leiva-Cisco.pdf