



# Automated network OS testing

Tomáš Kubina / Orange Business Services  
tomas.kubina@orange.com



# Automated network OS testing



## Network OS testing at OBS

- Orange Business Services (OBS) – worldwide telecommunications operator and service provider for enterprise market
- main features of network: L3VPN, L2VPN, BGP, ISIS, MPLS, MPLS/TE, QoS, LAC, IPv6, LDP, etc...
- 3 labs with real and virtual devices simulating our network
- 3 locations and around 35 people working on tests and development
- usual test of network feature: prepare and check testbed, put configuration of feature, check state of device and feature, do some bad things if needed, check stability and finish
- 70% of checks and 30% of configuration

# Automated network OS testing



## Testing challenges

- increase of complexity of network and network OSes in order to provide more and more features to customers
- many recurring tests (small/mid complexity)
- isolation of human error / overlook
- people need to focus on development of new features rather than testing of old ones
- reduce time to test network OS before deployment on live network

=> WE SHOULD AUTOMATE

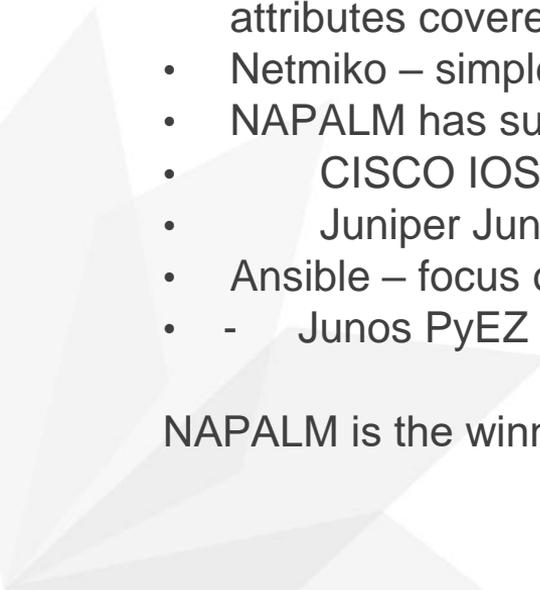
# Automated network OS testing



## Key factors of automation in OBS

- verification of operational state is key
- YANG models for operational state are still heavily evolving => not all monitored attributes covered, available only in newer OS releases / router platforms
- Netmiko – simple management of device via ssh, push command / get output
- NAPALM has support for configuration and operational state :
  - CISCO IOS/IOS-XE - screen scrapping and parsing
  - Juniper JunOS – processing of xml reply received over NETCONF
- Ansible – focus on provisioning and controlling
- - Junos PyEZ – configuration + operational state for JunOS only

NAPALM is the winner, because...



# Automated network OS testing



## NAPALM

- open source project
- implements a set of functions using a unified API to different router vendor network devices (ios, ios-xr, junos, nx-os,...)
- replacing/merging/comparing of configuration
- rollback of configuration
- retrieving operational data (interfaces' ip/state, BGP neighbors details, cpu, memory,...)
- can be integrated in ansible, SaltStack

# Automated network OS testing



## OBS NAPALM extension

- functions to provide operational data for IS-IS, LDP, BFD, BGP, PIM, MPLS TE, ...
- created for ios-xe and junos
- tested on CISCO ASR1K, CISCO 76xx, Juniper MX480
- some functions are specific to one vendor or are platform specific
- unification of various output
- new code will be published as official napalm code or standalone extension, point to discuss with NAPALM maintainer

# Automated network OS testing



functions added for Juniper platform:

**get\_redundancy\_info**

**get\_card\_type\_info**

**get\_mx\_re\_memory\_info**

**get\_mx\_active\_re\_memory\_info**

**get\_isis\_neighbors\_detail**

**get\_isis\_topology**

**get\_bfd\_neighbors\_detail**

**get\_ldp\_igp\_sync**

**get\_ldp\_neighbors**

**get\_pim\_neighbors(\_vrf)**

**get\_pim\_joins(\_vrf)**

**get\_multicast\_route(\_vrf)**

**get\_mvpn\_info**

**get\_l2ckt\_connections**

**get\_vpls\_connections**

**get\_interfaces\_rates**

**get\_route\_summary(\_vrf)**

**get\_fib\_table(\_vrf)**

# Automated network OS testing



functions added for CISCO platform:

**get\_redundancy\_info**

**get\_card\_type\_info**

**get\_asr1k\_slot\_memory\_info**

**get\_isis\_neighbors\_detail**

**get\_isis\_topology**

**get\_bfd\_neighbors\_detail**

**get\_ldp\_igp\_sync**

**get\_ldp\_neighbors**

**get\_mpls\_l2transport\_summary**

**get\_mpls\_te\_tunnels\_brief**

**get\_ip\_route\_summary(\_vrf)**

**get\_ip\_cef(\_vrf)**

**get\_interfaces\_rates**

**get\_xconnect\_all**

**get\_bgp\_all\_summary**

**get\_pim\_neighbors(\_vrf)**

**get\_ip\_mroute(\_vrf)**

**get\_ip\_mfib (\_vrf)**

**get\_ip\_mfib\_active(\_vrf)**

**get\_vrf**

**get\_bgp\_vpnv4\_vrf**

# Automated network OS testing



## NAPALM example

- Test name: Reload Test
- Test description: Connect to router, get global operational state of router, execute full reload, connect after 30minutes, get global operational state again and compare with outputs with state gathered before reload
- Pass criteria: Operational state before and after reload of device must be same, with some tolerance (memory, dynamic virtual interfaces name, routing protocol peers uptime,...)

# Automated network OS testing



## CISCO CLI:

```
Bnet-A1#sh isis neighbors detail
```

System Id	Type	Interface	IP Address	State	Holdtime	Circuit Id
Bnet-P1	L2	Gi3/0/17	192.168.244.189	UP	29	00

```
Area Address(es): 65.0421
```

```
SNPA: e4c7.2259.96c5
```

```
State Changed: 1d01h
```

```
Format: Phase V
```

```
Remote TID: 0
```

```
Local TID: 0
```

```
Interface name: GigabitEthernet3/0/17
```

```
Neighbor Circuit Id: 78
```

```
BFD enabled: (MTID:0, ipv4)
```

Bnet-P1	L2	Gi3/0/8.1308	192.168.244.109	UP	23	00
---------	----	--------------	-----------------	----	----	----

```
Area Address(es): 65.0421
```

```
SNPA: e4c7.2259.96b4
```

```
State Changed: 1d01h
```

```
Format: Phase V
```

```
Remote TID: 0
```

```
Local TID: 0
```

```
Interface name: GigabitEthernet3/0/8.1308
```

```
Neighbor Circuit Id: 116
```

```
Remote BFD Support: (MTID:0, IPV4)
```

System Id	Type	Interface	IP Address	State	Holdtime	Circuit Id
-----------	------	-----------	------------	-------	----------	------------

```
BFD enabled: (MTID:0, ipv4)
```

```
Bnet-A1#
```

# Automated network OS testing



- output from get\_isis\_neighbors from cisco platform:

```
'Bnet-P1': {u'Te3/1/0.1622': {u'area': u'65.0421',
                             u'bfd_enabled': u'(MTID:0, ipv4)',
                             u'circuit_id': u'00',
                             u'format': u'Phase V',
                             u'holdtime': u'24', ←===== can be different after reload
                             u'ip_address': u'192.168.244.250',
                             u'level': u'L2',
                             u'local_tid': u'0',
                             u'remote_bfd_support': u'(MTID:0, IPV4)',
                             u'remote_tid': u'0',
                             u'snpa': u'e4c7.225b.8d2f',
                             u'state': u'UP',
                             u'state_changed': 80892}, ←===== can be different after reload
  u'Te3/1/1': {u'area': u'65.0421',
              u'bfd_enabled': u'(MTID:0, ipv4)',
              u'circuit_id': u'00',
              u'format': u'Phase V',
              u'holdtime': u'28', ←===== can be different after reload
              u'ip_address': u'192.168.244.50',
              u'level': u'L2',
              u'local_tid': u'0',
              u'remote_bfd_support': u'(MTID:0, IPV4)',
              u'remote_tid': u'0',
              u'snpa': u'70e4.2219.9f8f',
              u'state': u'UP',
              u'state_changed': 80895}} ←===== can be different after reload
```

# Automated network OS testing



- output from `get_asr1k_slot_memory_info` from cisco platform:

```
'0': {'state': 'ok', 'type': 'ASR1000-SIP40'},
'0/0': {'state': 'ok', 'type': 'SPA-5X1GE-V2'},
'0/1': {'state': 'ok', 'type': 'SPA-5X1GE-V2'},
'0/2': {'state': 'ok', 'type': 'SPA-4XT3/E3-V2'},
'0/3': {'state': 'ok', 'type': 'SPA-4X1FE-TX-V2'}
```

- output from `get_environment` from cisco platform:

```
u'cpu': {0: {'usage': 11.0}},
u'fans': {'invalid': {'status': True}},
u'memory': {'available_ram': 8619809248, u'used_ram': 1632733416}, ←===== +/- 5% tolerance
u'power': {'invalid': {'capacity': -1.0, u'output': -1.0, u'status': True}},
u'temperature': {'invalid': {'is_alert': False,
                             u'is_critical': False,
                             u'temperature': -1.0}}
```

# Automated network OS testing



The screenshot shows a web browser window with the URL `10.253.60.123:8004/task_scheduler`. The page title is "Test automation framework | F...". The browser's address bar and tabs are visible at the top. The application interface has a dark blue sidebar on the left with the following menu items: "Task scheduling" (selected), "Tasks", "Scheduler", "Devices", "IP-v4 Classic", and "IP-v4 Custom". The main content area is titled "Test automation" and contains a "Welcome," message with a user profile icon. Below this, there are two sections: "Tests" and "Devices".

**Tests**

- VRF creation
- VPN address overlapping
- Unnumbered addresses
- Management routes aggregate
- Management routes
- Customer route aggregation
- BGP Route Map in/out
- BGP Route Map match/set commands
- BGP Prefix-list in/out
- BGP Prefix-list filtering
- BGP MD5 authentication
- BGP maximum prefix: over maximum
- BGP maximum prefix: under maximum
- BGP Hold timer
- BGP Allow as-in

**Devices**

- bnet-a1
- bnet-a2
- bnet-a3
- bnet-g2
- bnet-g1
- bnet-p1
- bnet-e4
- bnet-e5
- bnet-e6
- bnet-e7
- bnet-e1
- bnet-e2
- bnet-e3
- bnet-i1
- bnet-i2

**Form fields:**

- Name:**
- Waiting time between steps:**
- Start date:**
- End date:**
- Frequency:**
- Arguments:**

**Buttons:**

-

# Automated network OS testing



## Evolution?

- Robot Framework = generic open source automation framework for acceptance testing, acceptance test driven development and process automation
- high-level “keywords” instructs what to do, usually have input arguments and output value
- built- in keywords
- custom keywords directly in test suite or in external library (Python or Java)
- final test report generated in html
- easy integration into CI/CD

# Automated network OS testing



## Robot Framework test example - file *csnog-demo.robot*

\*\*\* Settings \*\*\*

Documentation    Demo script for CSNOG 2019:  
...                - Requirementns: NAPALM (pip install napalm)  
...                - Please update SUT parameters

Library          OBSRobotNapalm.OBSRobotNapalm  
Test setup      Connect device

\*\*\* Variables \*\*\*

\${SUT}=          192.168.243.69  
\${LOGIN}=        auto\_test\_user  
\${PASSWORD}=    auto\_test\_user\_password

\*\*\* Keywords \*\*\*

Connect device

    \${napalm-connection-SUT}=    Napalm Connect    \${SUT}    \${LOGIN}    \${PASSWORD}  
    Set Suite Variable    \${napalm-connection-SUT}

< built-in keywords    custom keywords from OBSNapalmRobot library >

# Automated network OS testing



## Robot Framework test description example

\*\*\* Test cases \*\*\*

TEST ISIS PEER FLAP

[Documentation] Check if ISIS peer is UP, clear ISIS process, check again

`${isis_peer_ip}= Set Variable 192.168.245.89`

`${command}= Set Variable clear isis *`

`${peer_state}= Get ISIS Peer State ${napalm-connection-SUT} ${isis_peer_ip}`  
`Should Be Equal UP ${peer_state}`

`Execute Operational Command ${napalm-connection-SUT} ${command}`

`Sleep 100`

`${peer_state}= Get ISIS Peer State ${napalm-connection-SUT} ${isis_peer_ip}`  
`Should Be Equal UP ${peer_state}`

TEST BGP PEER FLAP

<another test case>

< built-in keywords    custom keywords from OBSNapalmRobot library >

# Automated network OS testing



## Robot Framework CLI output

```
(venv) tkubina@nap-dev:~/python/obs-eval-library$ robot csnog-demo.robot
=====
Csnog-Demo :: Demo script for CSNOG 2019:
=====
TEST ISIS PEER FLAP :: Check if ISIS peer is UP, clear ISIS proces... | PASS |
-----
Csnog-Demo :: Demo script for CSNOG 2019: | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
=====
Output: /home/tkubina/python/obs-eval-library/output.xml
Log:    /home/tkubina/python/obs-eval-library/log.html
Report: /home/tkubina/python/obs-eval-library/report.html
(venv) tkubina@nap-dev:~/python/obs-eval-library$
```

# Automated network OS testing



## Robot Framework html report

The screenshot shows a web browser window displaying a Robot Framework HTML report. The browser's address bar shows the file path: file:///Z:/jobs-eval-library/report.html#. The report title is 'Csnog-Demo Report', generated on 20190517 14:23:08 UTC-02:00, 13 seconds ago. A 'LOG' button is visible in the top right corner of the report area.

### Summary Information

**Status:** All tests passed  
**Documentation:** Demo script for CSNOG 2019:

- Requirements: NAPALM (pip install napalm)
- Please update SUT parameters

**Start Time:** 20190517 14:21:18.896  
**End Time:** 20190517 14:23:08.429  
**Elapsed Time:** 00:01:49.531  
**Log File:** [log.html](#)

### Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:01:49	<span style="color: green;">██████████</span>
All Tests	1	1	0	00:01:49	<span style="color: green;">██████████</span>

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Csnog-Demo	1	1	0	00:01:50	<span style="color: green;">██████████</span>

### Test Details

Totals Tags Suites Search

Type:  Critical Tests  All Tests

# Automated network OS testing



## Robot Framework html report

The screenshot shows a web browser window displaying a Robot Framework HTML report. The browser address bar shows the file path: file:///Z:/obs-eval-library/log.html. The report title is 'Csnog-Demo Log' and it was generated on 20190517 14:23:08 UTC+02:00, 12 minutes and 29 seconds ago. A 'REPORT' button is visible in the top right corner.

### Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:01:49	<div style="width: 100%; height: 10px; background-color: green;"></div>
All Tests	1	1	0	00:01:49	<div style="width: 100%; height: 10px; background-color: green;"></div>

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Csnog-Demo	1	1	0	00:01:50	<div style="width: 100%; height: 10px; background-color: green;"></div>

### Test Execution Log

**SUITE: Csnog-Demo** 00:01:49.531

Full Name: Csnog-Demo  
Documentation: Demo script for CSNOG 2019:

- Requirements: NAPALM (pip install napalm)
- Please update SUT parameters

Source: /home/kubana/python/obs-eval-library/csnog-demo.robot  
Start / End / Elapsed: 20190517 14:21:18.898 / 20190517 14:23:08.429 / 00:01:49.531  
Status: 1 critical test, 1 passed, 0 failed  
1 test total, 1 passed, 0 failed

---

**TEST: TEST ISIS PEER FLAP** 00:01:49.367

Full Name: Csnog-Demo.TEST ISIS PEER FLAP  
Documentation: Check if ISIS peer is UP, clear ISIS process, check again  
Start / End / Elapsed: 20190517 14:21:19.061 / 20190517 14:23:08.429 / 00:01:49.367  
Status: **PASS** (critical)

- **SETUP** Connect device 00:00:04.850
- **KEYWORD** \$[sis\_peer\_ip] = **Run**. Set Variable 192.168.245.89 00:00:00.000
- **KEYWORD** \$[command] = **Run**. Set Variable clear isis \* 00:00:00.001
- **KEYWORD** \$[peer\_state] = **Run**. Get ISIS Peer State \$[napalm-connection-SUT]. \$[sis\_peer\_ip] 00:00:00.804
- **KEYWORD** **Run**. Should Be Equal UP. \$[peer\_state] 00:00:00.000
- **KEYWORD** **Run**. Execute Operational Command \$[napalm-connection-SUT]. \$[command] 00:00:00.002
- **KEYWORD** **Run**. Sleep 100 00:01:40.001
- **KEYWORD** \$[peer\_state] = **Run**. Get ISIS Peer State \$[napalm-connection-SUT]. \$[sis\_peer\_ip] 00:00:02.705
- **KEYWORD** **Run**. Should Be Equal UP. \$[peer\_state] 00:00:00.000

# Automated network OS testing



## Robot Framework OBSRobotNapalm.py keyword library

```
from napalm import get_network_driver
from robot.api.deco import keyword
```

```
@keyword('Napalm Connect')
```

```
def napalm_connect(device, login, password):
```

```
    driver = get_network_driver('ios')
    device = driver(device, login, password)
    device.open()
    return device
```

```
@keyword('Get ISIS Peer State')
```

```
def get_isis_peer_state(device, peer_ip):
```

```
    isis_peers = device.get_isis_neighbors_detail()
    for peer_name in isis_peers:
        for interface in isis_peers[peer_name]:
            if isis_peers[peer_name][interface]['ip_address'] == \
                peer_ip:
                return isis_peers[peer_name][interface]['state']
```

```
    return 'None'
```

```
@keyword('Execute Operational Command')
```

```
def execute_op_command(device, command):
```

```
    return device._send_command(command)
```

# Automated network OS testing



**Thank you!**



# Automated network OS testing



## Resources:

NAPALM <https://napalm-automation.net/>

Netmiko <https://github.com/ktbyers/netmiko>

Ansible <https://github.com/ansible>

YDK <http://ydk.cisco.com/py/docs/#>

Junos PyEZ <https://github.com/Juniper/py-junos-eznc>

NETCONF <https://tools.ietf.org/html/rfc6241>

YANG <https://napalm-automation.net/yang-for-dummies/>

Robot Framework <http://robotframework.org>

updated NAPALM repo:

[https://github.com/tomaskubina/napalm/tree/obs\\_nie\\_experiment](https://github.com/tomaskubina/napalm/tree/obs_nie_experiment)

