

# Exploring Network Programmability with Python and YANG

Lisa Roach

# Who am I?

- Previously a Software Systems Engineer at Cisco Systems
- Currently a Production Engineer at Facebook in security

# What can we program?

- Configuration management
- Health monitoring and problem remediation
- Customized protocols

# How to program networking devices?

- APIs
- SSH
- YANG models + RPC

# Our Scenario



# Diagnosing without YANG or APIs

## SSH + Screenscraping + Regex

```
import paramiko
import re
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect('10.1.1.5', 22, 'vagrant', 'vagrant')

def check_box():
    command = 'show interface GigabitEthernet 0/0/0/0'
    stdin, stdout, stderr = ssh.exec_command(command)
    output = ''
    for line in stdout:
        output += line
    return bool(re.search('GigabitEthernet0/0/0/0 is up', output))
```

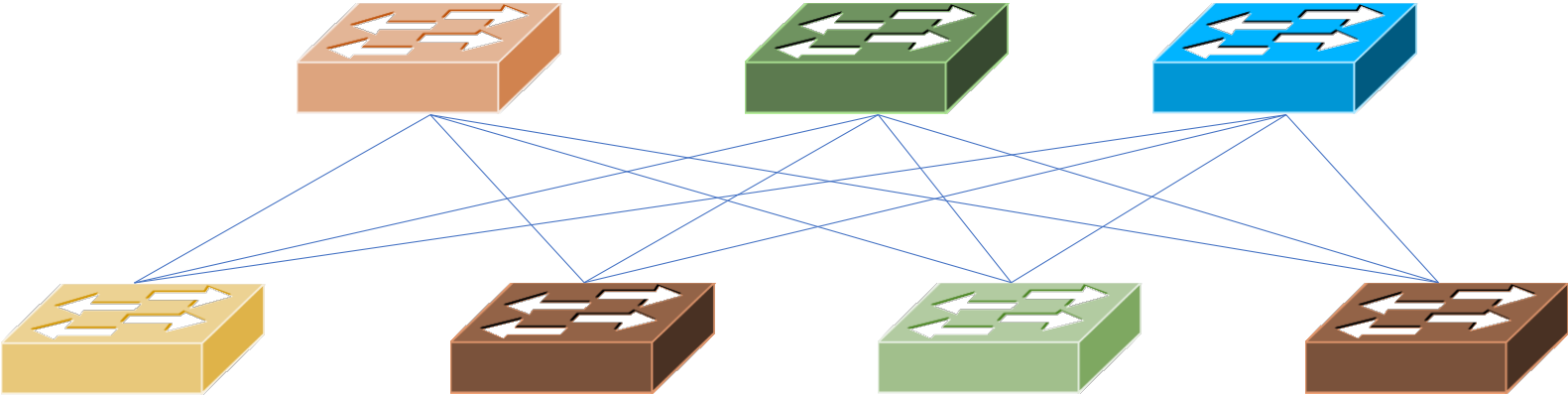
# Diagnosing without YANG or APIs

## SSH + Screenscraping + Regex

```
import paramiko
import re
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect('10.1.1.5', 22, 'vagrant', 'vagrant')

def check_box():
    command = 'show interface GigabitEthernet 0/0/0/0'
    stdin, stdout, stderr = ssh.exec_command(command)
    output = ''
    for line in stdout:
        output += line
    return bool(re.search('GigabitEthernet0/0/0/0 is up', output))
```

# Networks are Mixed





# The Emergence of YANG

“Yet Another Next Generation”

## **What is YANG?**

A data modeling language used to describe configuration and state data.

## **How does YANG help?**

Provides a consistent model to configure and monitor network devices in a software driven way, regardless of vendor.

# Yang Models Need to be Standardized

HOW STANDARDS PROLIFERATE:  
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)



# What does YANG look like?

```
module openconfig-interfaces {
  yang-version "1";
  grouping interfaces-top {
    container interfaces {
      list interface {
        key "name";
        leaf name {
          type leafref {
            path "../config/name";
          }
        }
      }
      container config {
        uses interface-phys-config;
      }
    }
  }
  ...
}
```

# How can we use YANG?

- Configuration and state data is modeled on box to YANG specifications
- RPC calls can be used to grab the data off the box and return it as XML or JSON
- Python can be used to manipulate the XML or JSON values
- A RPC can be used to send the manipulated data back to the box

# Pyang

```
pyang -f tree pyang/public/release/models/interfaces/openconfig-interfaces.yang
```

```
module: openconfig-interfaces
  +--rw interfaces
    +--rw interface* [name]
      +--rw name          -> ../config/name
      +--rw config
        | +--rw type          identityref
        | +--rw mtu?         uint16
        | +--rw name?       string
        | +--rw description? string
        | +--rw enabled?    boolean
      +--ro state
        | +--ro type          identityref
```

```
...
```

# How can we use YANG?

```
RP/0/RP0/CPU0:rt1#show interface gigabitEthernet 0/0/0
Tue May 9 23:54:59.353 UTC
GigabitEthernet0/0/0 is up, line protocol is up
Interface state transitions: 1
Hardware is GigabitEthernet, address is 0800.2703.e97f (bia 0800.2703.e97f)
Description: to pop router
Internet address is 12.1.1.20/24 MTU 1514 bytes,
BW 1000000 Kbit (Max: 1000000 Kbit)
reliability 255/255, txload 0/255, rxload 0/255
Encapsulation ARPA,
Full-duplex, 1000Mb/s, unknown, link type is force-up
output flow control is off, input flow control is off
Carrier delay (up) is 10 msec loopback not set,
Last link flapped 00:01:12
ARP type ARPA, ARP timeout 04:00:00
Last input never, output 00:00:01
Last clearing of "show interface" counters never
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 1000 bits/sec, 0 packets/sec
0 packets input, 0 bytes, 0 total input drops
0 drops for unrecognized upper-level protocol
Received 0 broadcast packets, 0 multicast packets
0 runs, 0 giants, 0 throttles, 0 parity
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
6796 packets output, 9746584 bytes, 0 total output drops Output 0 broadcast packets, 0 multicast packets
0 output errors, 0 underruns, 0 applique, 0 resets
0 output buffer failures, 0 output buffers swapped out 13 carrier transitions
Last link flapped 00:01:12
ARP type ARPA, ARP timeout 04:00:00
Last input never, output 00:00:01
Last clearing of "show interface" counters never
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 1000 bits/sec, 0 packets/sec
0 packets input, 0 bytes, 0 total input drops
0 drops for unrecognized upper-level protocol
0 runs, 0 giants, 0 throttles, 0 parity
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
```

# How can we use YANG?

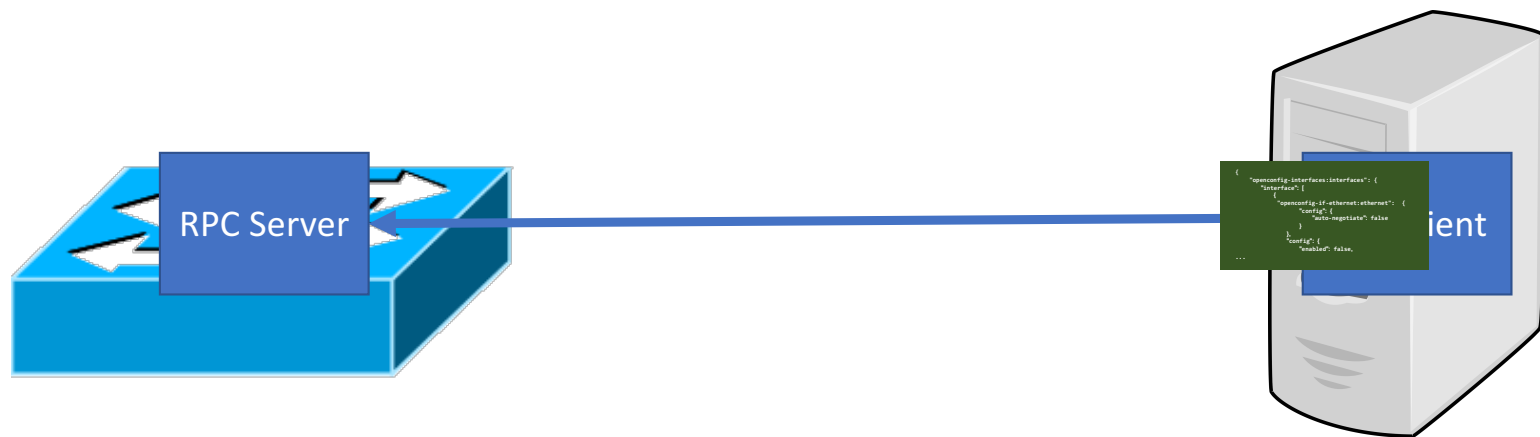
```
{
  "openconfig-interfaces:interfaces": {
    "interface": [
      {
        "name": "GigabitEthernet0/0/0/0",
        "config": {
          "name": "GigabitEthernet0/0/0/0",
          "type": "iana-if-type:ethernetCsmacd",
          "enabled": true,
          "description": "to DC router"
        },
        "openconfig-if-ethernet:ethernet": {
          "config": {
            "auto-negotiate": false
          }
        },
        "subinterfaces": {
          "subinterface": [
```

# How can we use YANG?





# How can we use YANG?



# Mapping YANG to JSON

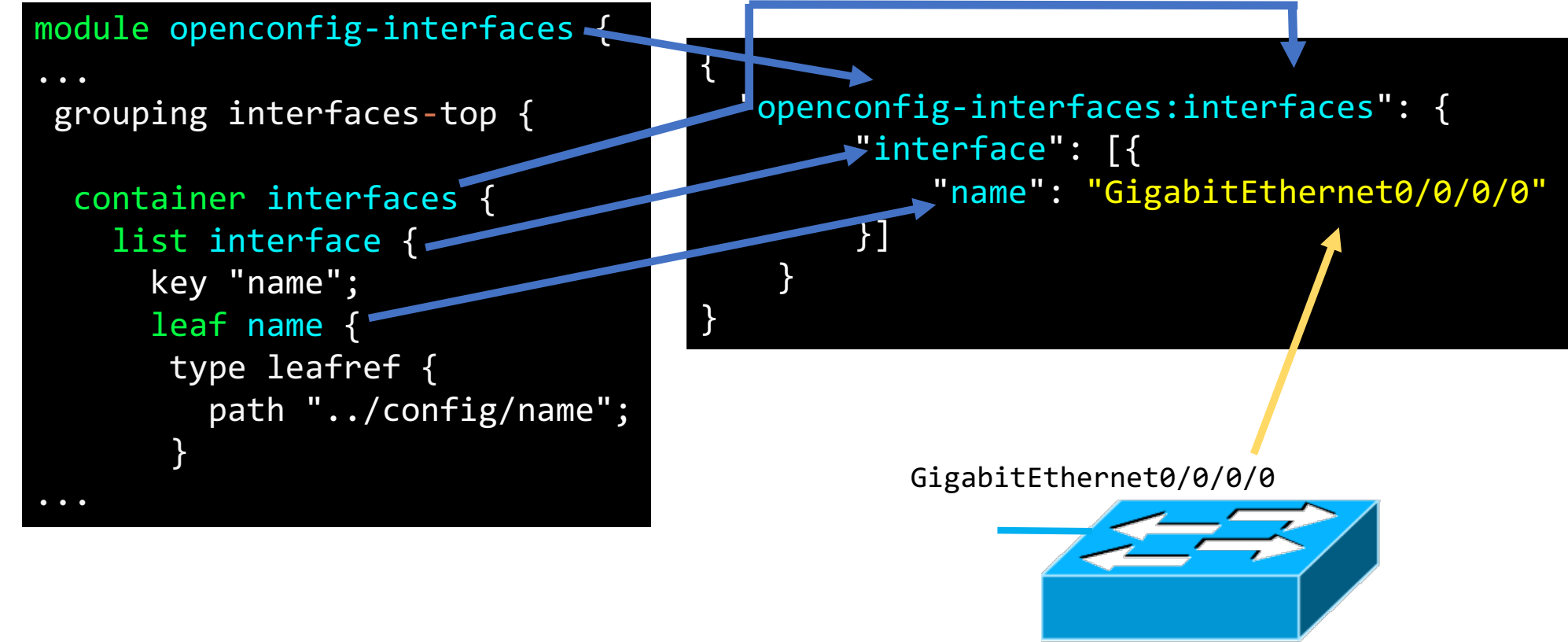
YANG

```
module openconfig-interfaces {  
  ...  
  grouping interfaces-top {  
    container interfaces {  
      list interface {  
        key "name";  
        leaf name {  
          type leafref {  
            path "../config/name";  
          }  
        }  
      }  
    }  
  }  
  ...  
}
```

JSON Mapping

```
{  
  "openconfig-interfaces:interfaces": {  
    "interface": [{  
      "name": "GigabitEthernet0/0/0/0"  
    }]  
  }  
}
```

GigabitEthernet0/0/0/0



# Let's Fix Our Switch



## The SSH + Screenscraping Way

```
...
if not check_box():
    ssh.connect('10.1.1.5', 22, 'vagrant', 'vagrant')
    shell = ssh.invoke_shell()
    while not shell.recv_ready():
        sleep(0.1)
    shell.send('conf t\n')
    shell.send('interface GigabitEthernet 0/0/0/0\n')
    shell.send('no shutdown\n')
    shell.send('commit\n')
    sleep(1)
...
```

## The SSH + Screenscraping Way

```
...
if not check_box():
    ssh.connect('10.1.1.5', 22, 'vagrant', 'vagrant')
    shell = ssh.invoke_shell()
    while not shell.recv_ready():
        sleep(0.1)
    shell.send('conf t\n')
    shell.send('interface GigabitEthernet 0/0/0/0\n')
    shell.send('no shutdown\n')
    shell.send('commit\n')
    sleep(1)
...
```

## The SSH + Screenscraping Way

```
...
if not check_box():
    ssh.connect('10.1.1.5', 22, 'vagrant', 'vagrant')
    shell = ssh.invoke_shell()
    while not shell.recv_ready():
        sleep(0.1)
    shell.send('conf t\n')
    shell.send('interface GigabitEthernet 0/0/0/0\n')
    shell.send('no shutdown\n')
    shell.send('commit\n')
    sleep(1)
...
```

## The SSH + Screenscraping Way

```
...
if not check_box():
    ssh.connect('10.1.1.5', 22, 'vagrant', 'vagrant')
    shell = ssh.invoke_shell()
    while not shell.recv_ready():
        sleep(0.1)
    shell.send('conf t\n')
    shell.send('interface GigabitEthernet 0/0/0/0\n')
    shell.send('no shutdown\n')
    shell.send('commit\n')
    sleep(1)
...
```

## The SSH + Screenscraping Way

```
...
if not check_box():
    ssh.connect('10.1.1.5', 22, 'vagrant', 'vagrant')
    shell = ssh.invoke_shell()
    while not shell.recv_ready():
        sleep(0.1)
    shell.send('conf t\n')
    shell.send('interface GigabitEthernet 0/0/0/0\n')
    shell.send('no shutdown\n')
    shell.send('commit\n')
    sleep(1)
...
```



The YANG way

## Formulate Get Request

```
from Tools.grpc_client import GRPCClient
client = GRPCClient('10.1.1.5', 57777, 10, 'vagrant', 'vagrant')
{
  "openconfig-interfaces:interfaces": {
    "interface": [{
      "name": "GigabitEthernet0/0/0/0"
    }]
  }
}
config = client.getconfig(model)
```

# Response

```
>>> response = json.loads(config, object_pairs_hook=OrderedDict)
```

# Response

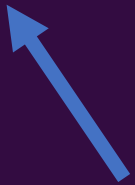
```
>>> response = json.loads(config, object_pairs_hook=OrderedDict)
>>> print(json.dumps(response, indent=4))
```

# Response

```
>>> response = json.loads(config, object_pairs_hook=OrderedDict)
>>> print(json.dumps(response, indent=4))
{
  "openconfig-interfaces:interfaces": {
    "interface": [{
      "name": "GigabitEthernet0/0/0/0",
      "config": {
        "name": "GigabitEthernet0/0/0/0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false,
        ...
      }
    }
  ]
}
```

# Response

```
>>> response = json.loads(config, object_pairs_hook=OrderedDict)
>>> print(json.dumps(response, indent=4))
{
  "openconfig-interfaces:interfaces": {
    "interface": [{
      "name": "GigabitEthernet0/0/0/0",
      "config": {
        "name": "GigabitEthernet0/0/0/0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false,
        ...
      }
    }
  ]
}
```



# Response

```
>>> response = json.loads(config, object_pairs_hook=OrderedDict)
>>> print(json.dumps(response, indent=4))
{
  "openconfig-interfaces:interfaces": {
    "interface": [{
      "name": "GigabitEthernet0/0/0/0",
      "config": {
        "name": "GigabitEthernet0/0/0/0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false,
        ...
      }
    }
  ]
}
```

## Manipulate the data

```
response = json.loads(config, object_pairs_hook=OrderedDict)
interface = response['openconfig-interfaces:interfaces']['interface'][0]
```

```
>>> print(json.dumps(interface, indent=4))
{
  "name": "GigabitEthernet0/0/0/0",
  "config": {
    "name": "GigabitEthernet0/0/0/0",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": false,
    ...
  }
}
```



## Manipulate the data

```
response = json.loads(config, object_pairs_hook=OrderedDict)
interface = response['openconfig-interfaces:interfaces']['interface'][0]
interface['config']['enabled'] = True
```

```
>>> print(json.dumps(response, indent=4))
{
  "openconfig-interfaces:interfaces": {
    "interface": [{
      "name": "GigabitEthernet0/0/0/0",
      "config": {
        "name": "GigabitEthernet0/0/0/0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
```

## RPC Put/Patch/Merge

```
response = json.loads(config, object_pairs_hook=OrderedDict)
interface = response['openconfig-interfaces:interfaces']['interface'][0]
interface['config']['enabled'] = True
updated_config = json.dumps(response)
err = client.mergeconfig(updated_config)
```

# Check Change is Successful

```
config = client.getconfig(model)
response = json.loads(config, object_pairs_hook=OrderedDict)
```

```
>>> print(json.dumps(response, indent=4))
{
  "openconfig-interfaces:interfaces": {
    "interface": [
      {
        "name": "GigabitEthernet0/0/0/0",
        "config": {
          "name": "GigabitEthernet0/0/0/0",
          "type": "iana-if-type:ethernetCsmacd",
          "enabled": true,
          ...
        }
      }
    ]
  }
}
```

# Let's Fix Our Switch



# Timing

Type	\$ time python yang_change.py	\$ time python screenscape.py	Difference
real	0.551s	2.800s	<b>2.249s</b>
user	0.161s	0.453s	<b>0.292s</b>
sys	0.029s	0.053s	<b>0.024s</b>

# Drawbacks of YANG

- Not 100% stable
- Not largely available (yet)
- Devices can be blocking
- Few tools and documentation for support

# Helpful Resources

- RFC: <https://tools.ietf.org/html/rfc6020>

## Github

- Yang: <https://github.com/YangModels/yang>
- Pipedown: <https://github.com/cisco-ie/Pipedown>
- Solenoid: <https://github.com/lisroach/Solenoid>
- GRPC Client: <https://github.com/cisco-ie/ios-xr-grpc-python>
- Napalm (for ssh and YANG!): <https://github.com/napalm-automation/napalm>