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))
```

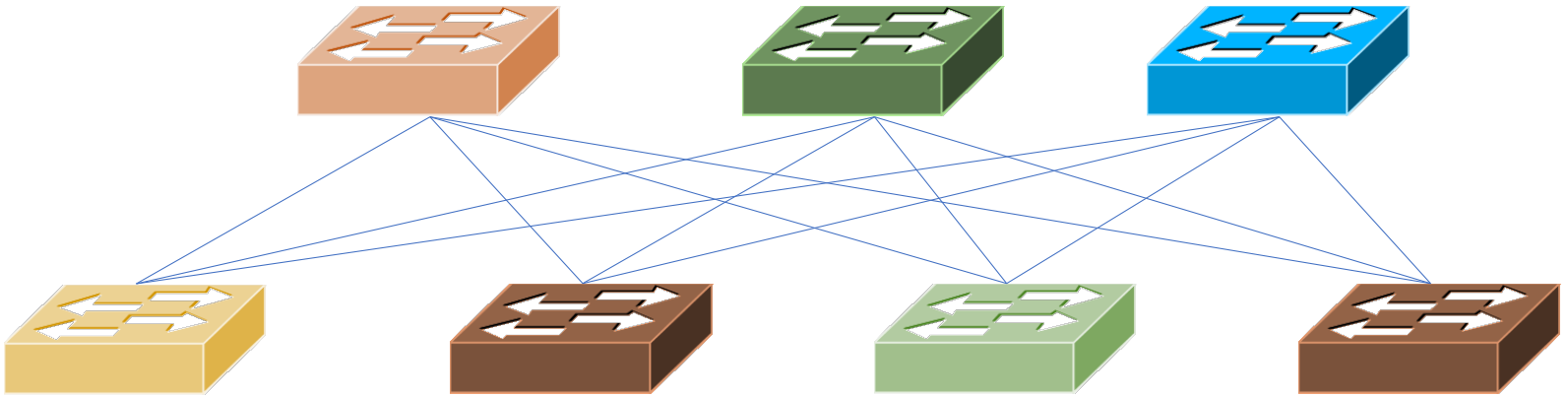
Diagnosing without YANG or APIs

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"

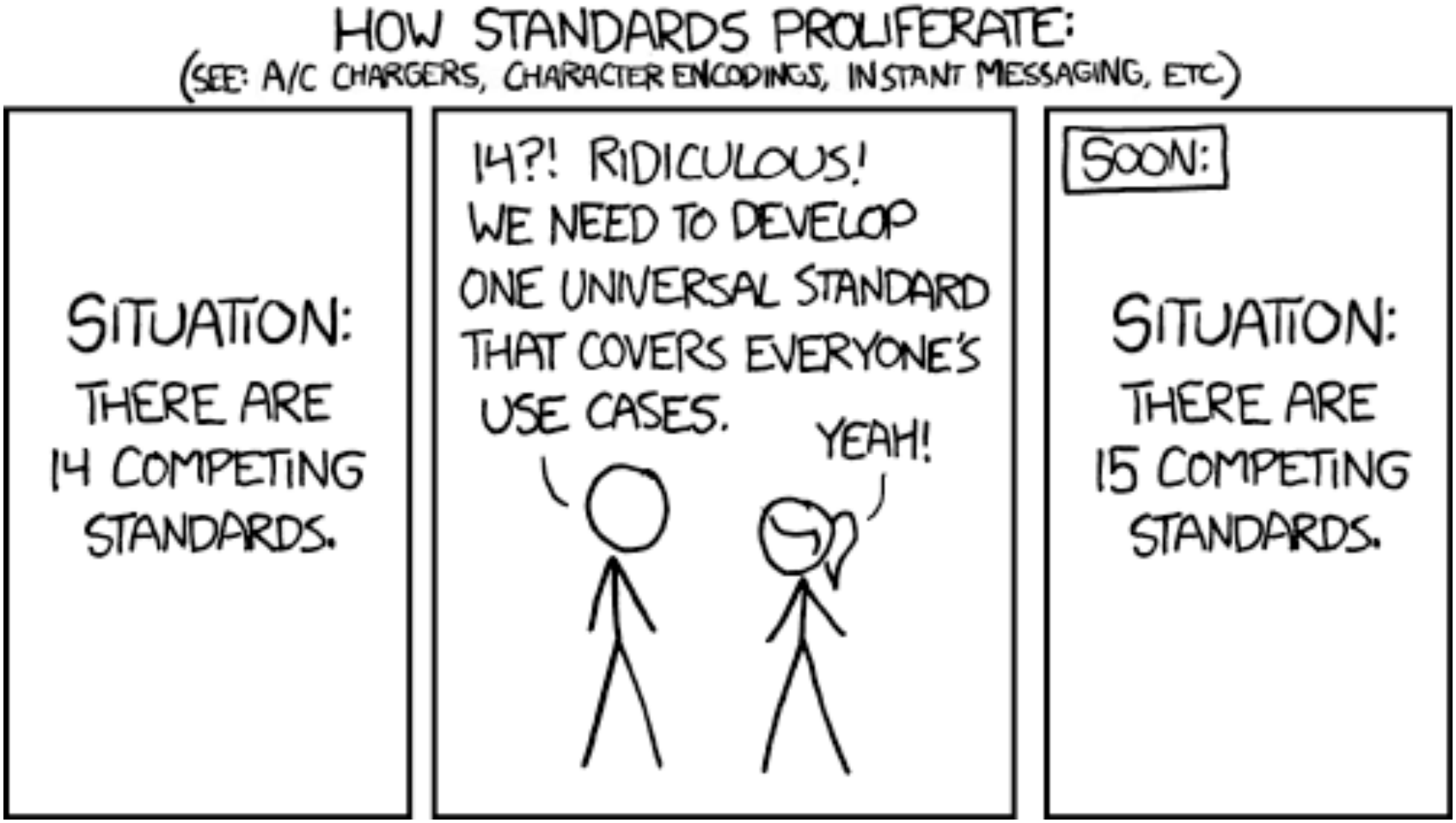
What is YANG?

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



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;
        }
...
```

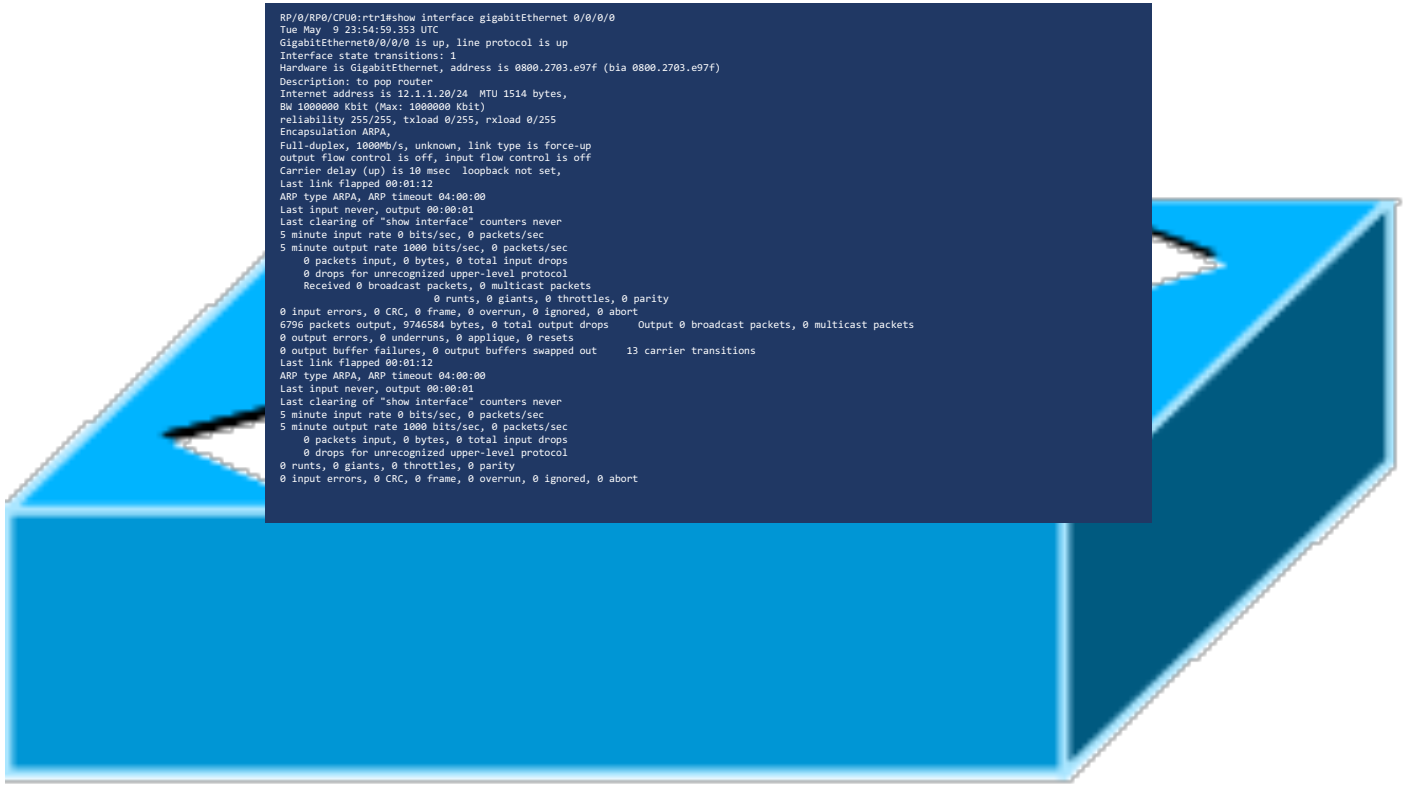
How can we use YANG?

- 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
...
```

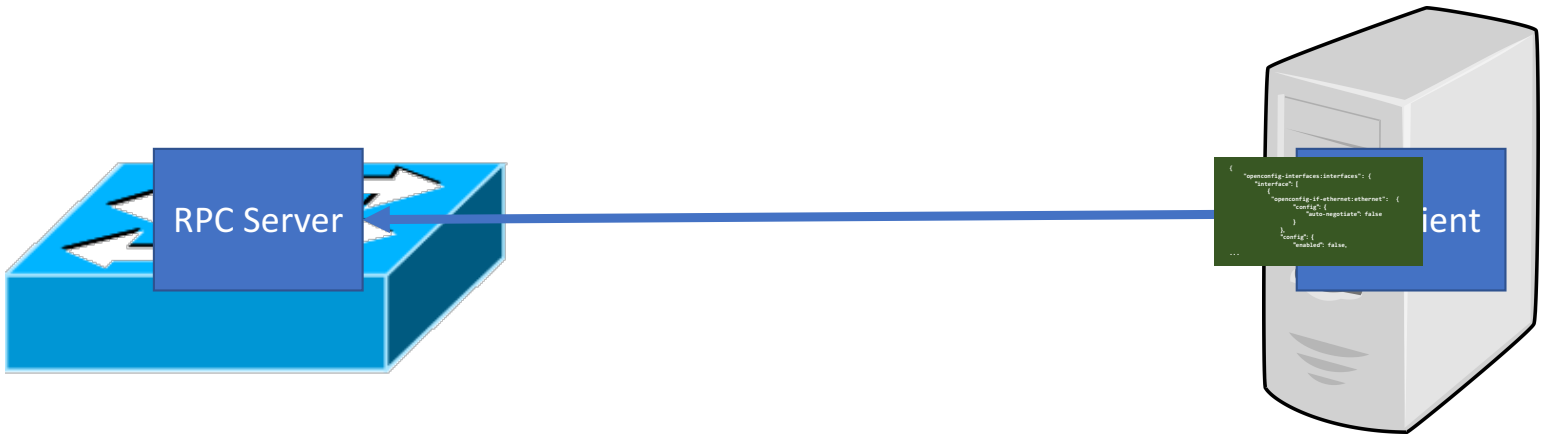
How can we use YANG?



How can we use YANG?

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

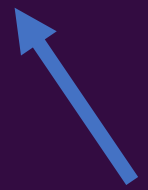
```
>>> response = json.loads(config, object_pairs_hook=OrderedDict)
>>> print(json.dumps(response, indent=4))
```

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

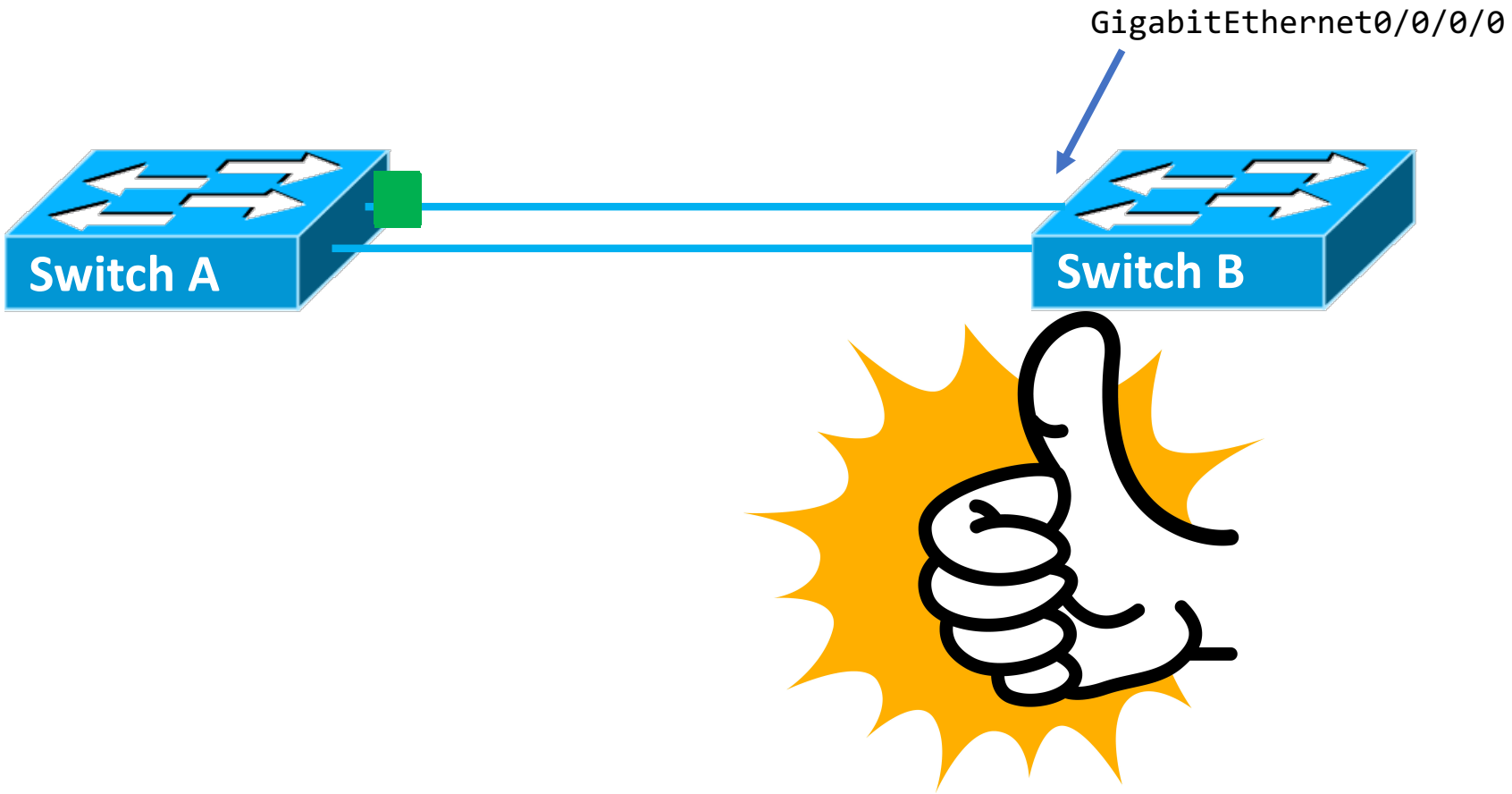
```
response = json.loads(config, object_pairs_hook=OrderedDict)
interface = response['openconfig-interfaces:interfaces']['interface'][0]
interface['config']['enabled'] = True
updated_config = json.dumps(response)
err = client.mergeconfig(updated_config)
```

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 screenscrape.py	Difference
real	0.551s	2.800s	**2.249s**
user	0.161s	0.453s	**0.292s**
sys	0.029s	0.053s	**0.024s**

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