University of Colorado **Boulder**

Network Management and Automation

Network Configuration (NETCONF), YANG, and OpenConfig

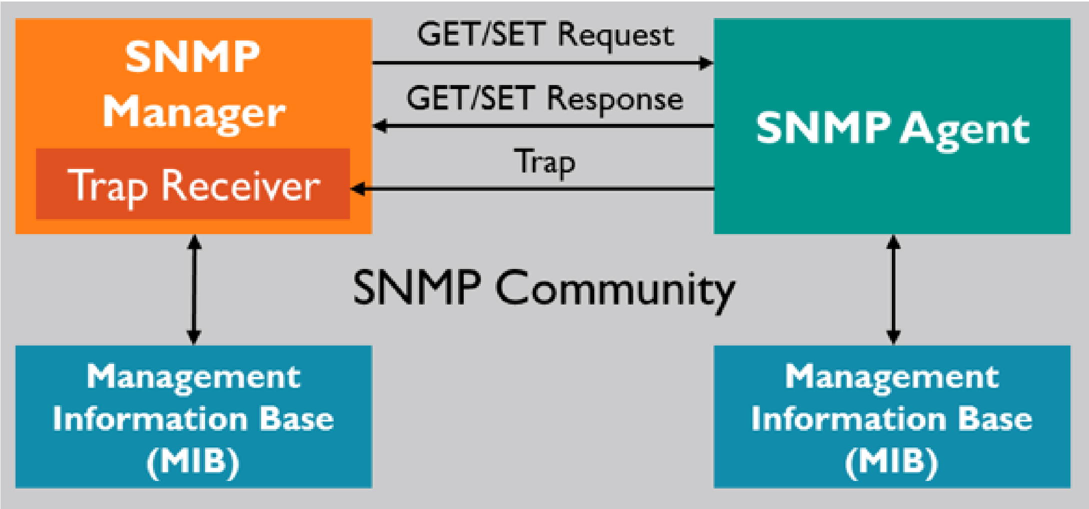
Levi Perigo, Ph.D.
University of Colorado Boulder
Department of Computer Science
Network Engineering

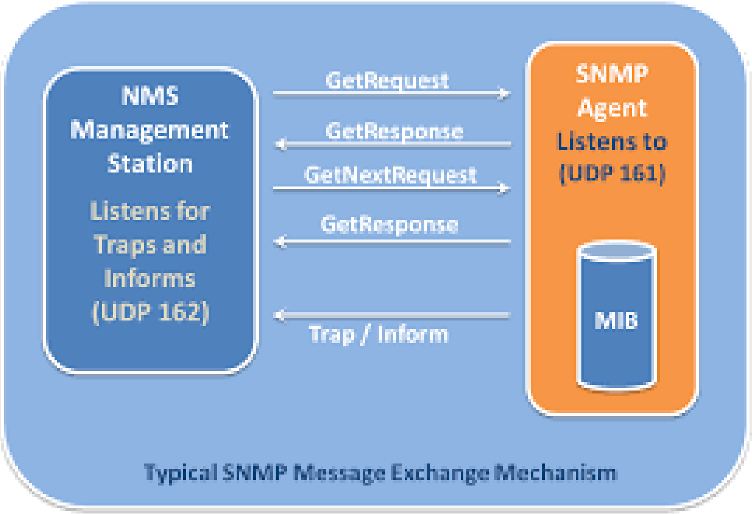
Review

- **Network Management**
- **Network Automation**
- **DevOps**
- **Abstraction**
- **Virtualization**
- **SDN / NFV**

SNMP

The first main-stream network management protocol





SNMP Has Failed

- **Typically used for network monitoring and fault handling**
  - – Not configuration management

- **Vendor specific CLIs used to configure each device/hardware ~70% vendor specific**
  - – Individual MIBs

- **RFC 3535 (2002)**

SNMP Protocol Limitations

- **Lack of support for:**
  - Atomic transactions
    - ***Providing a full config at boot time***
    - ***Providing backup and restore capabilities***
    - ***Validation of config data set prior to activation***
  - Connection-oriented management sessions
    - ***Limited to connectionless transport which can generate more traffic (e.g., inefficient for configuring complex devices)***
  - Multiple configuration data stores

- **Limited set of protocol operations (Get, Set, etc.)**

- **Difficult to scale**

- **Using SNMPv3 for secure connections is complex and difficult to deploy**

What would you want out of a network management protocol?

Network Operators Requirements (RFC3535)

1. **Easy to USE for the operator**
   - SNMP is easy to implement (not use/configuration)
   - Don’t care if it’s difficult to implement

2. **Clear distinction between configuration data and state/stats**

3. **Fetch and compare multi-vendor configs**

4. **Focus on managing the network, not individual devices**

5. **Network wide transactions**

Network Operators Requirements (Cont.)

6. **Config A to Config B**
   - Set of changes (transactional) (not sequence)
     - ***VPN config. – Missing command "validates" (i.e. missing line 7)***
7. **Standard for pulling/pushing/restoring configs**
8. **Validation of configuration**
9. **Use of text processing tools (diff, version mgmt)**
10. **Common database schema (for configs/commands)**

Automation - Service Providers & Hyperscale

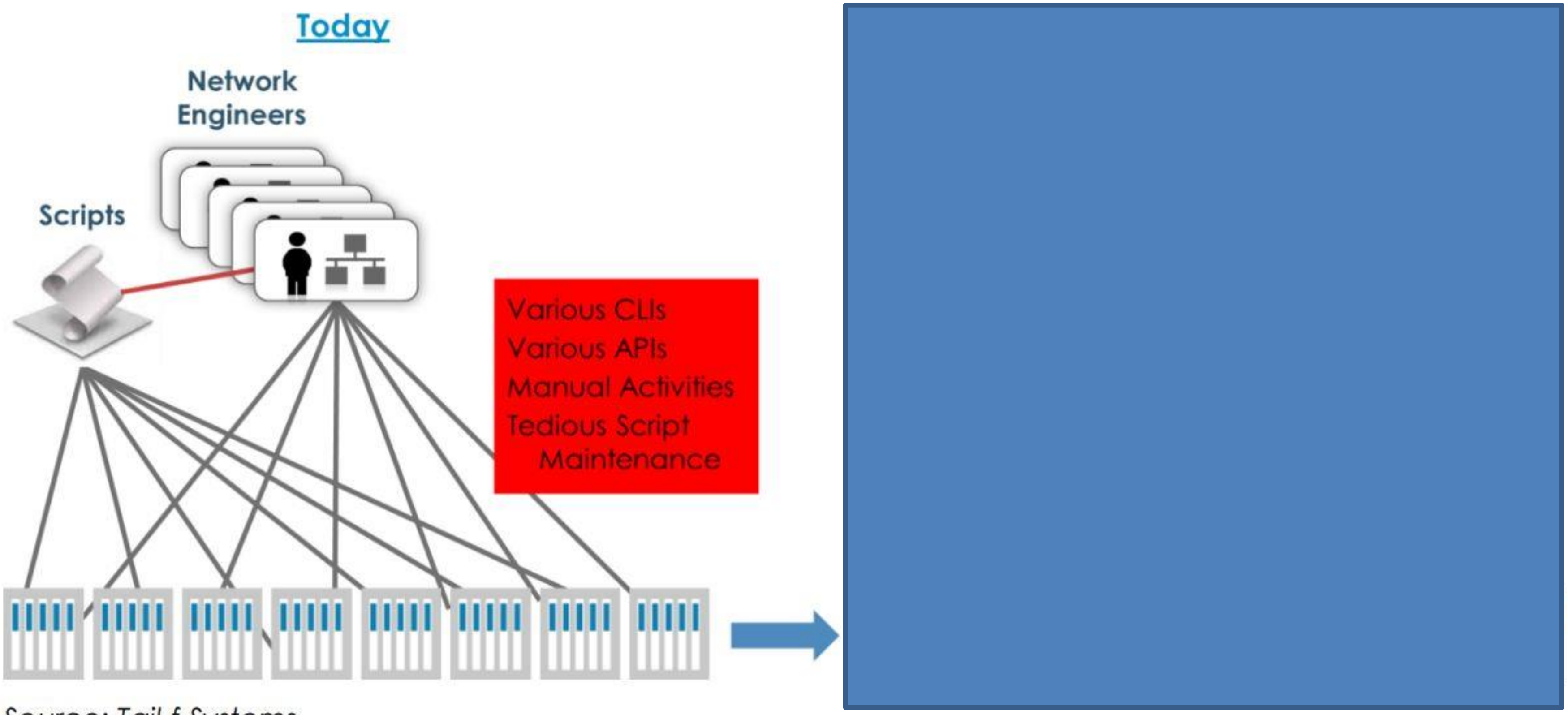
- **SP/Hs are leading the way for programmatic and standards-based network configuration & automation**
  - Writing configurations to any network device (from any vendor) instead of manual configuration of hundreds of devices
    - ***What can be used for this?***
  - Vendor proprietary days are numbered

- **Dynamic, self-provisioning & self-healing network services**
  - Troubleshooting

- **Vendors that support this have a competitive advantage, but in the future, it will be mandatory**

Shift to Automation

Figure 1: The Shift Toward Standards-Based Network Abstraction & Automation



Various CLIs
Various APIs
Manual Activities
Tedious Script Maintenance

Source: Tail-f Systems

Models, Encodings, Protocols and Transports

– UML is a general purpose modelling language to visualize (through diagrams) the design of a system

– YANG is a general purpose data modelling language to describe the structure of data

– Data described by a data model can be encoded in a number of different ways, such as XML and JSON

– Encoded data can be carried over a number of different network management protocols, such as NETCONF and RESTCONF

– Network management protocols provide the operations for acting on the data described by the data model, and carry the corresponding encoded data

– The protocol messages are carried over secure application layer protocols such as SSH, TLS and HTTPS

– Transport layer protocols including TCP and UDP provide the secure application layer protocols a transport mechanism

– See Appendix (NetO-App)

NETwork CONFiguration Protocol (NETCONF)

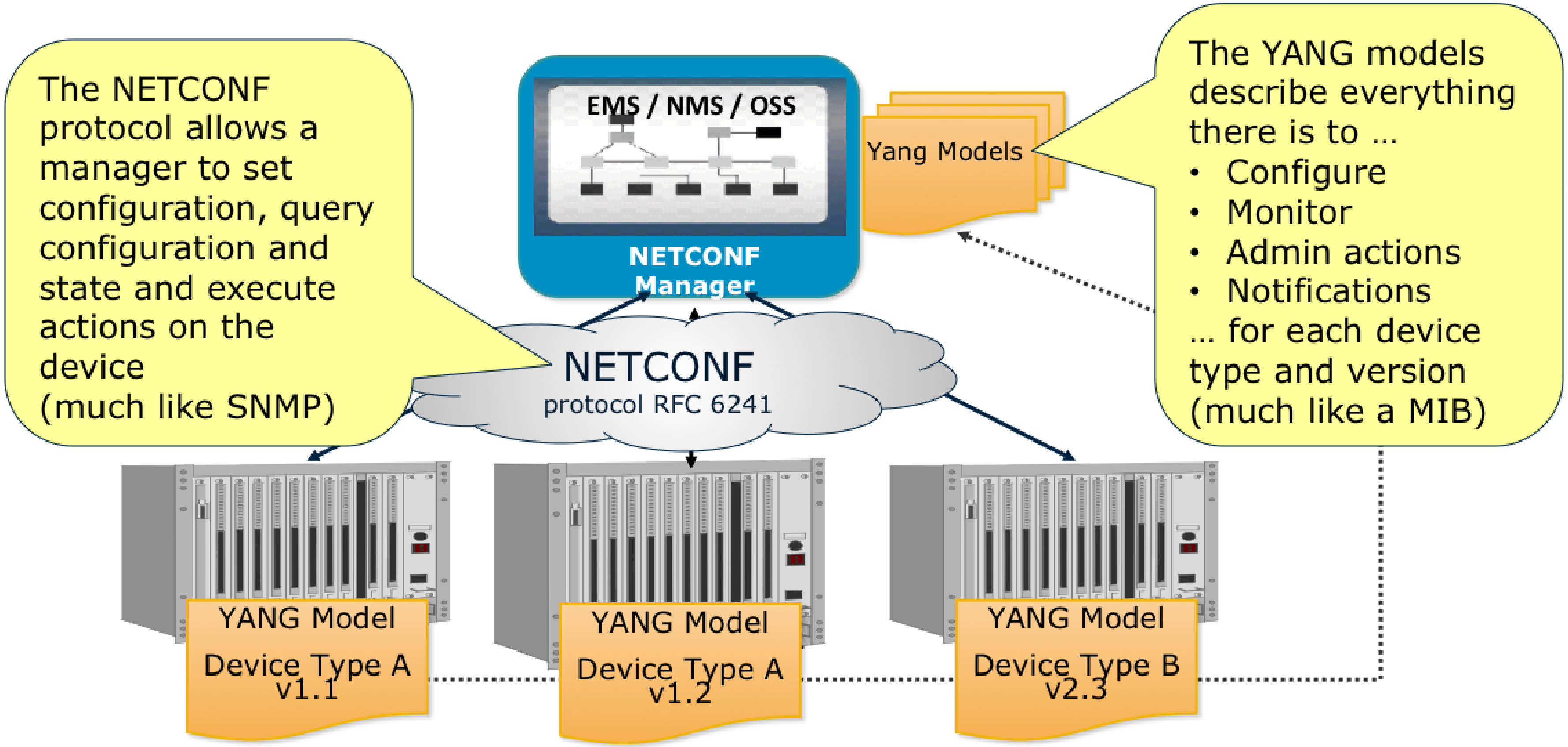
- **IETF (RFC 6241)- network management protocol designed to support configuration of network devices**

- **Designed for configuration of network devices**
  - Install
  - Manipulate
  - Delete

- **Remote Procedure Call (RPC) mechanism**

- **Extensible Markup Language (XML)**
  - Data encoding for configuration data
  - Protocol messages
    - ***Secure transport protocol***

NETCONF & YANG in Context

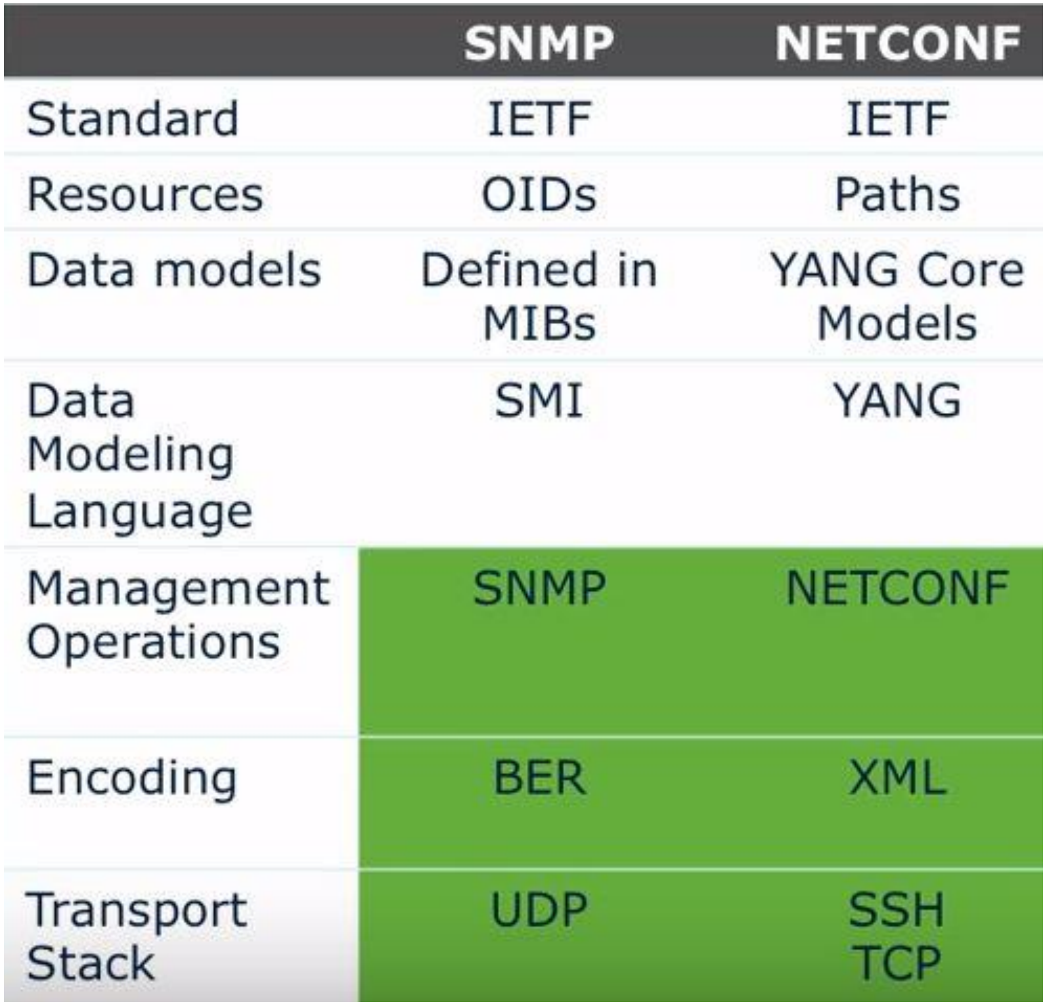


The NETCONF protocol allows a manager to set configuration, query configuration and state and execute actions on the device
(much like SNMP)

The YANG models describe everything there is to …
- Configure
- Monitor
- Admin actions
- Notifications

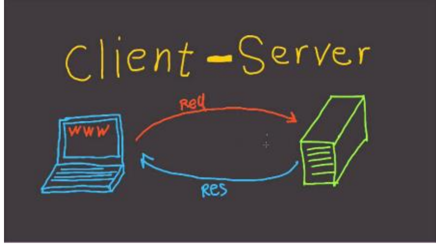
… for each device type and version
(much like a MIB)

NETCONF vs. SNMP



	SNMP	NETCONF
Standard	IETF	IETF
Resources	OIDs	Paths
Data models	Defined in MIBs	YANG Core Models
Data Modeling Language	SMI	YANG
Management Operations	SNMP	NETCONF
Encoding	BER	XML
Transport Stack	UDP	SSH TCP

Client / Server Model

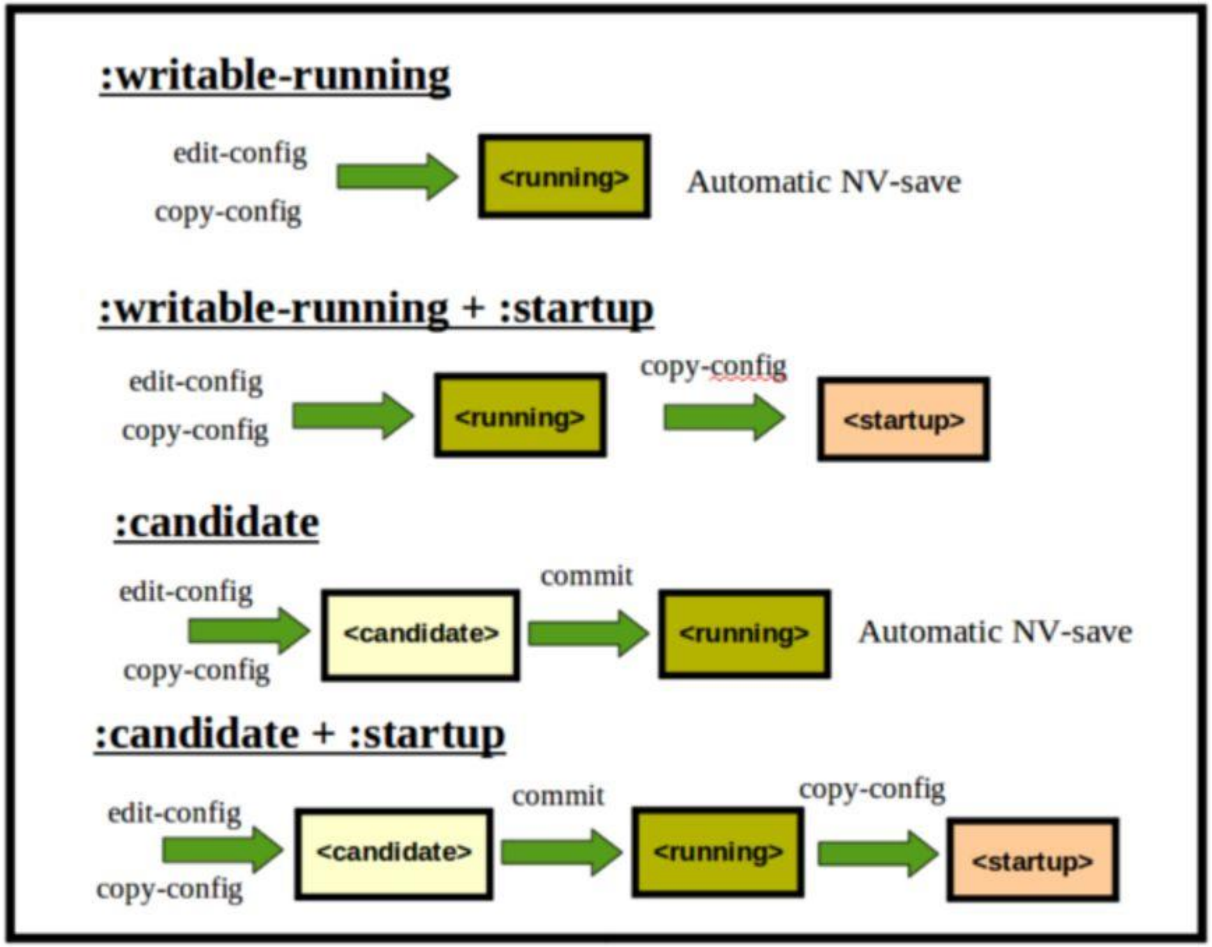


- **For network engineering, the server is the hardware “device” (router, switch, firewall, etc.)**
  - Server exposes an API to client
- **Configuration databases and “capabilities”**
  - “API contract” between server and client
- **Conceptual configuration databases**
  - Running
  - Candidate
  - Startup

Conceptual Configuration Databases

- **<running/>**
  - – Active configuration
  - – <get> or <get-config>
- **<candidate/>**
  - – Similar to “running” but does not take effect right away
  - – Used with “lock” feature
- **<startup/>**
  - – Changes are written to the startup configuration

Configuration Databases



:writable-running

edit-config

copy-config

<running>

Automatic NV-save

:writable-running + :startup

edit-config

copy-config

<running>

copy-config

<startup>

:candidate

edit-config

copy-config

<candidate>

commit

<running>

Automatic NV-save

:candidate + :startup

edit-config

copy-config

<candidate>

commit

<running>

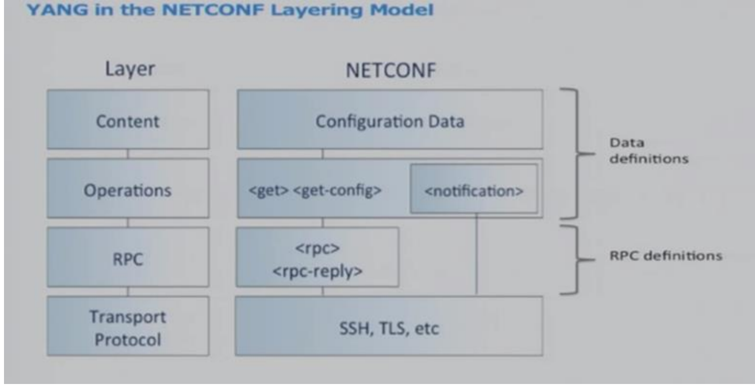
copy-config

<startup>

Sample NETCONF (XML)

```
•   <rpc message-id="101"
•        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
•       <edit-config>
•        <target>
•         <running/>
•        </target>
•        <config>
•         <top xmlns="http://example.com/schema/1.2/config">
•          <interface>
•           <name>Ethernet0/0</name>
•           <mtu>1500</mtu>
•          </interface>
•         </top>
•        </config>
•       </edit-config>
•      </rpc>

•   Acknowledge the operation:
•
•      <rpc-reply message-id="101"
•        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
•       <ok/>
•      </rpc-reply>
```

Four Layers



- **Content**
  - Configuration and notification data
    - ***Model of what can be done***

- **Operations**
  - Set of base protocol operations to retrieve/edit configuration data
    - ***Implementing the models***

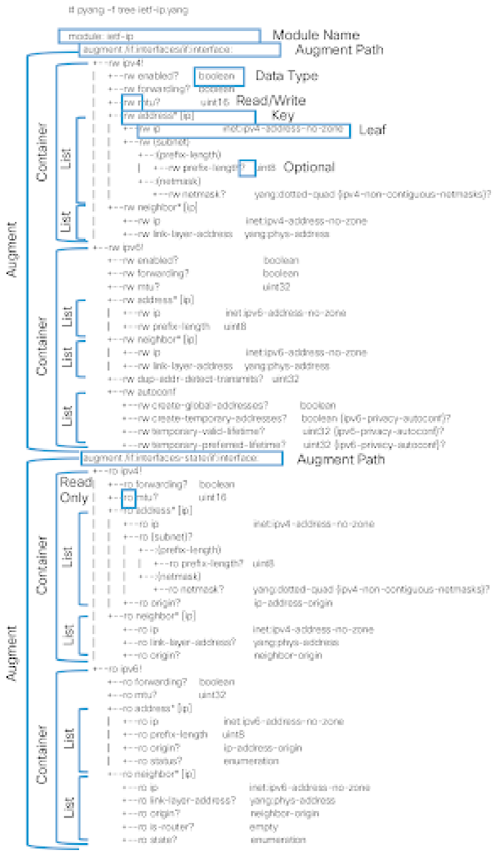
- **Messages**
  - Encoding RPCs and notifications

- **Secure Transport**
  - Secure and reliable transport between client/server

Content Layer

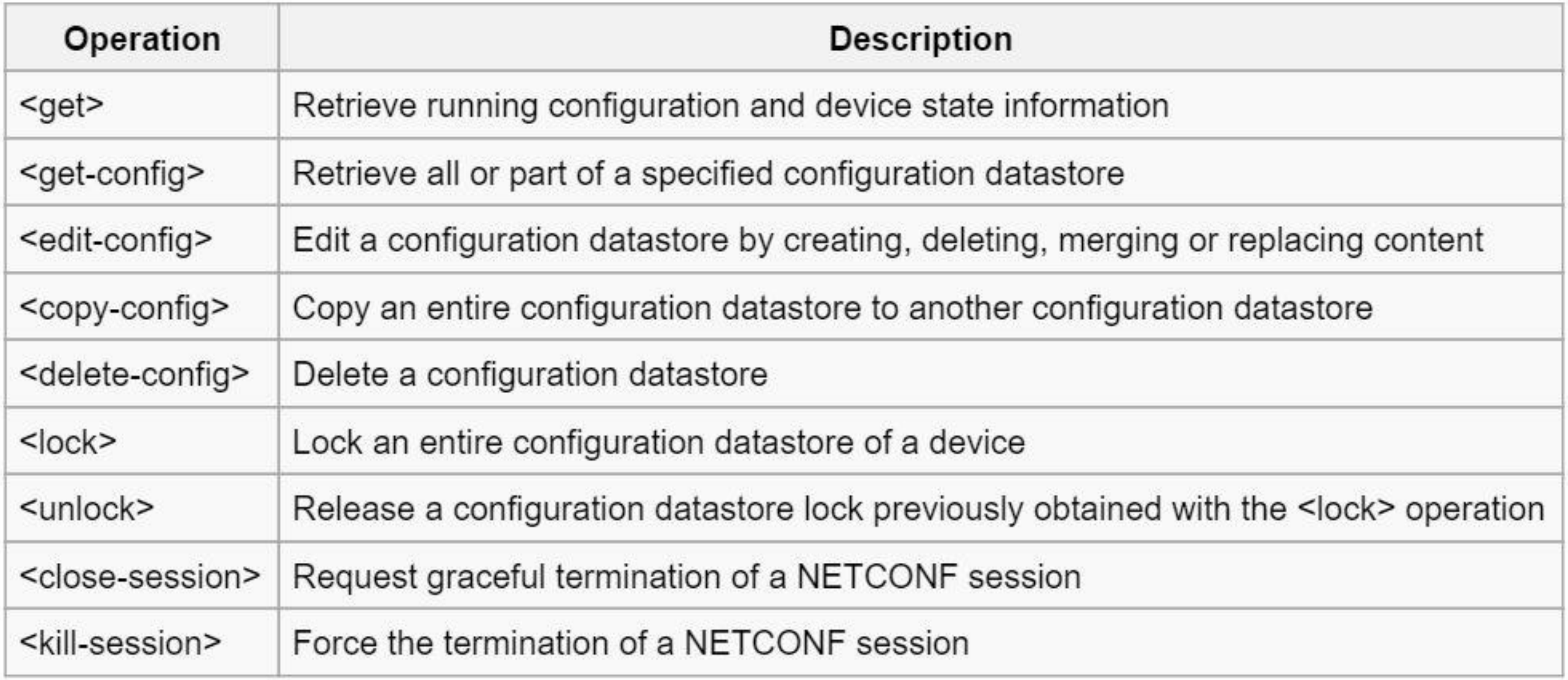
- **Human-friendly modeling language for semantics of:**
  - – Operational data
  - – Configuration data
  - – Notifications
  - – Operations

- **YANG**



```
# pyang -f tree ietf-ip.yang

module: ietf-ip
  augment /if:interfaces/if:interface:
    +--rw ipv4!
    |  +--rw enabled?      boolean
    |  +--rw forwarding?   boolean
    |  +--rw mtu?          uint16
    |  +--rw address* [ip]
    |  |  +--rw ip              inet:ipv4-address-no-zone
    |  |  +--rw (subnet)
    |  |     +--:(prefix-length)
    |  |     |  +--rw prefix-length?   uint8
    |  |     +--:(netmask)
    |  |        +--rw netmask?      yang:dotted-quad {ipv4-non-contiguous-netmasks}?
    |  +--rw neighbor* [ip]
    |     +--rw ip                    inet:ipv4-address-no-zone
    |     +--rw link-layer-address    yang:phys-address
    +--rw ipv6!
       +--rw enabled?                 boolean
       +--rw forwarding?              boolean
       +--rw mtu?                     uint32
       +--rw address* [ip]
       |  +--rw ip               inet:ipv6-address-no-zone
       |  +--rw prefix-length    uint8
       +--rw neighbor* [ip]
       |  +--rw ip                    inet:ipv6-address-no-zone
       |  +--rw link-layer-address    yang:phys-address
       +--rw dup-addr-detect-transmits?   uint32
       +--rw autoconf
          +--rw create-global-addresses?       boolean
          +--rw create-temporary-addresses?    boolean {ipv6-privacy-autoconf}?
          +--rw temporary-valid-lifetime?      uint32 {ipv6-privacy-autoconf}?
          +--rw temporary-preferred-lifetime?  uint32 {ipv6-privacy-autoconf}?
  augment /if:interfaces-state/if:interface:
    +--ro ipv4!
    |  +--ro forwarding?   boolean
    |  +--ro mtu?          uint16
    |  +--ro address* [ip]
    |  |  +--ro ip              inet:ipv4-address-no-zone
    |  |  +--ro (subnet)?
    |  |  |  +--:(prefix-length)
    |  |  |  |  +--ro prefix-length?   uint8
    |  |  |  +--:(netmask)
    |  |  |     +--ro netmask?      yang:dotted-quad {ipv4-non-contiguous-netmasks}?
    |  |  +--ro origin?         ip-address-origin
    |  +--ro neighbor* [ip]
    |     +--ro ip                     inet:ipv4-address-no-zone
    |     +--ro link-layer-address?    yang:phys-address
    |     +--ro origin?                neighbor-origin
    +--ro ipv6!
       +--ro forwarding?   boolean
       +--ro mtu?          uint32
       +--ro address* [ip]
       |  +--ro ip               inet:ipv6-address-no-zone
       |  +--ro prefix-length    uint8
       |  +--ro origin?          ip-address-origin
       |  +--ro status?          enumeration
       +--ro neighbor* [ip]
          +--ro ip                     inet:ipv6-address-no-zone
          +--ro link-layer-address?    yang:phys-address
          +--ro origin?                neighbor-origin
          +--ro is-router?             empty
          +--ro state?                 enumeration
```

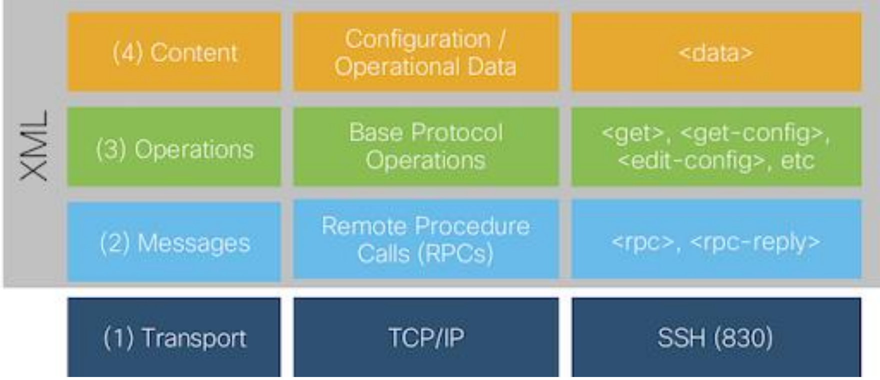
Operations Layer



Operation	Description
<get>	Retrieve running configuration and device state information
<get-config>	Retrieve all or part of a specified configuration datastore
<edit-config>	Edit a configuration datastore by creating, deleting, merging or replacing content
<copy-config>	Copy an entire configuration datastore to another configuration datastore
<delete-config>	Delete a configuration datastore
<lock>	Lock an entire configuration datastore of a device
<unlock>	Release a configuration datastore lock previously obtained with the <lock> operation
<close-session>	Request graceful termination of a NETCONF session
<kill-session>	Force the termination of a NETCONF session

- These are similar to SNMP but focus more on configuration not monitoring

Messages Layer



XML	(4) Content	Configuration / Operational Data	<data>
	(3) Operations	Base Protocol Operations	<get>, <get-config>, <edit-config>, etc
	(2) Messages	Remote Procedure Calls (RPCs)	<rpc>, <rpc-reply>
	(1) Transport	TCP/IP	SSH (830)

- **RPC invocations**
- **RPC results**
- **Event notifications**
- **Messages are XML documents**

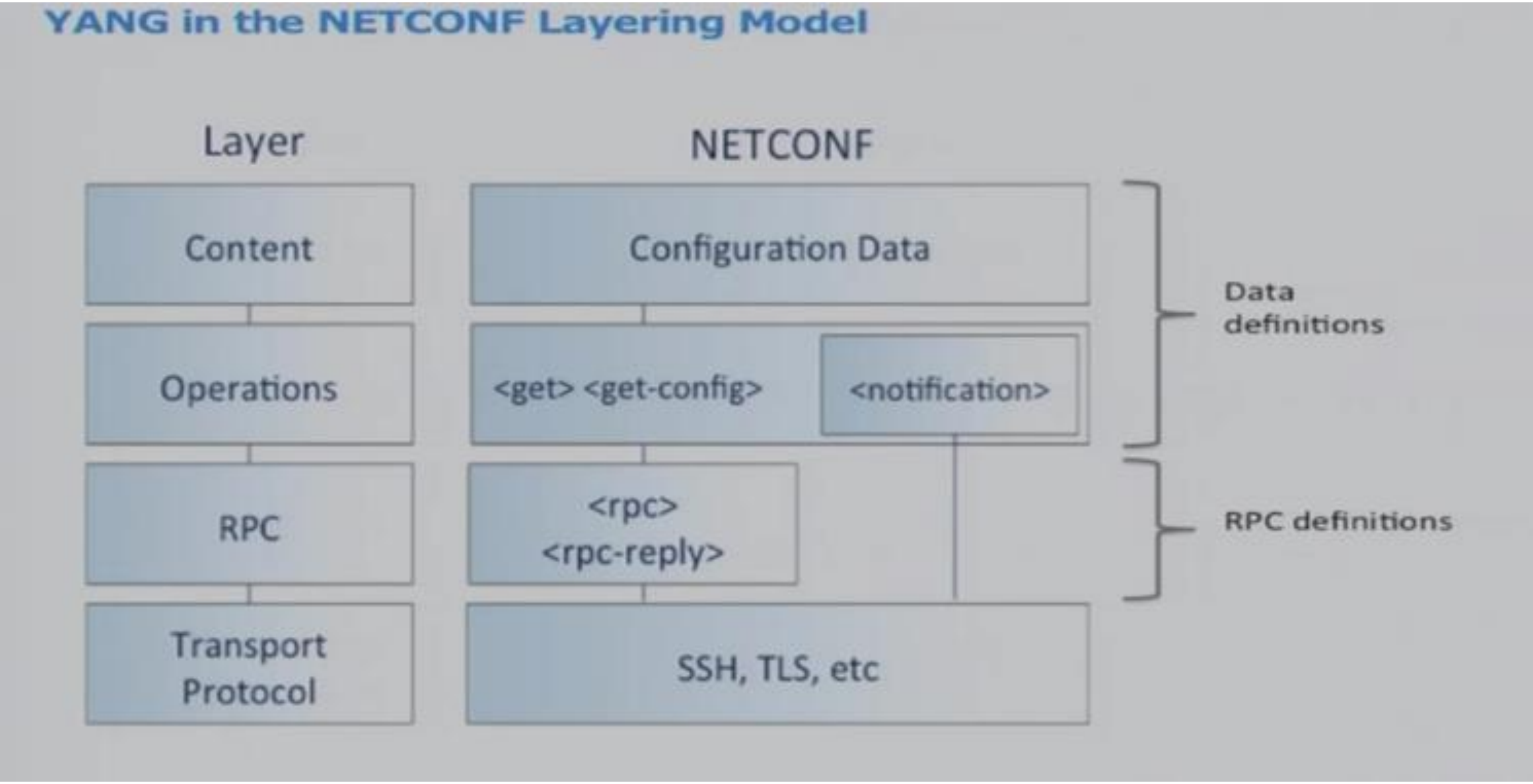
Secure Transports Layer

- **“Secure” must have:**
  - Authentication
  - Data integrity
  - Confidentiality
  - Replay protection

- **NETCONF over SSH**

- **TLS**

- **SSL**



YANG in the NETCONF Layering Model

Layer

NETCONF

Content

Operations

RPC

Transport Protocol

Configuration Data

<get> <get-config>

<notification>

<rpc>
<rpc-reply>

SSH, TLS, etc

Data definitions

RPC definitions

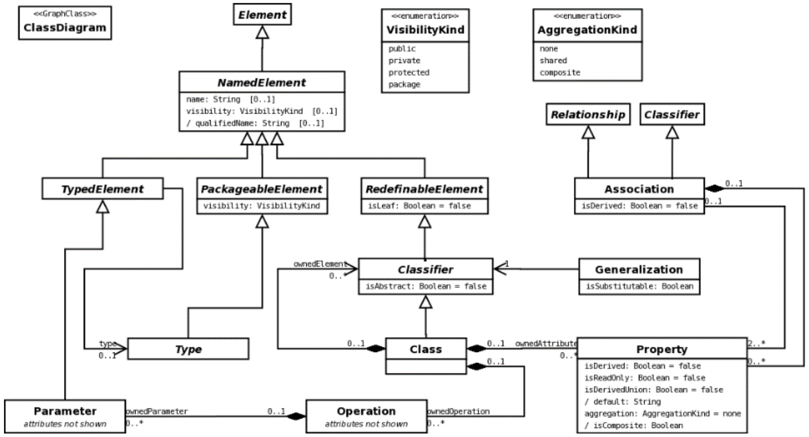
INFORMATION MODELING

- Information Modeling identifies the information and data which flows through the defined Business Process Flows.

- A concise Information Model is used to model information within components as well as interfaces between components.

- Common Information Models used across the end-to-end product and service lifecycle provide a cohesive, non-duplicative view of the data within and across systems.

- Information Models may define both the static/structural and dynamic/behavioral views of a solution.

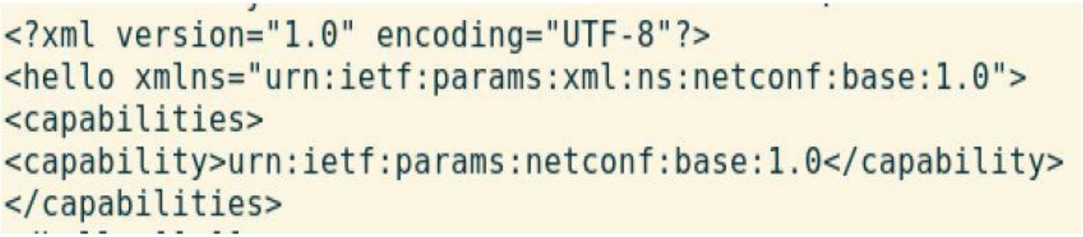


YANG

- **Data modeling language designed to write data models for NETCONF (RFC 6020)**
  - Data-model – explicitly and precisely determines the structure, syntax, and semantics of the data
  - Content layer (top) of NETCONF

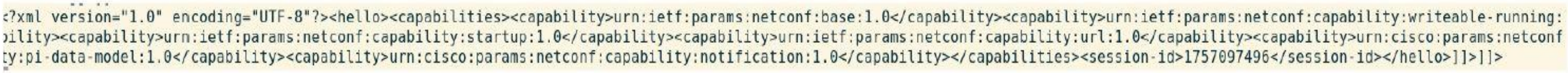
- **“What can be read/write on the device”**
  - i.e. SNMP MIBs
  - Instead of getting MIBs from vendor, server replies with capabilities, and you load that YANG module into the NMS
    - ***How is this better than SNMP?***

HELLO SENT TO CHECK CAPABILITIES OF ROUTER:

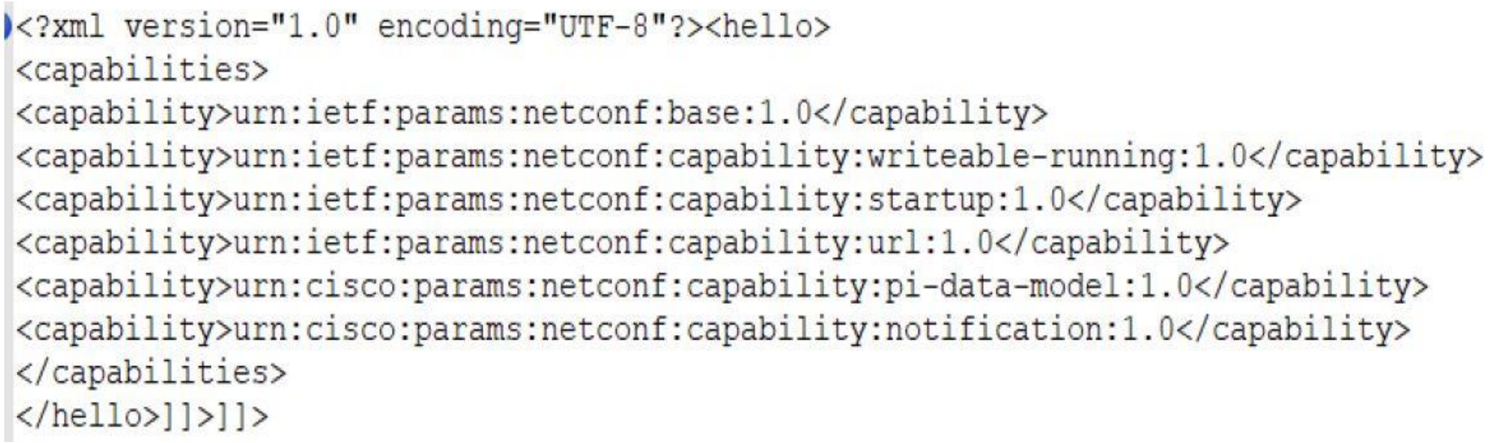


```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
</capabilities>
```

RECEIVED MESSAGE:



```
<?xml version="1.0" encoding="UTF-8"?><hello><capabilities><capability>urn:ietf:params:netconf:base:1.0</capability><capability>urn:ietf:params:netconf:capability:writeable-running:
bility><capability>urn:ietf:params:netconf:capability:startup:1.0</capability><capability>urn:ietf:params:netconf:capability:url:1.0</capability><capability>urn:cisco:params:netconf
ty:pi-data-model:1.0</capability><capability>urn:cisco:params:netconf:capability:notification:1.0</capability></capabilities><session-id>1757097496</session-id></hello>]]>]]>
```



```
<?xml version="1.0" encoding="UTF-8"?><hello>
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:capability:writeable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:startup:1.0</capability>
<capability>urn:ietf:params:netconf:capability:url:1.0</capability>
<capability>urn:cisco:params:netconf:capability:pi-data-model:1.0</capability>
<capability>urn:cisco:params:netconf:capability:notification:1.0</capability>
</capabilities>
</hello>]]>]]>
```

YANG

- **Human-readable**
- **“Easy to learn” representation**
- **Hierarchical configuration data models**
- **Extensibility through augmentation mechanisms**
- **Supports definitions of operations (RPCs)**
  - gRPC
    - ***gNMI***
      - gRPC Network Management Interface
- **Definitions directly map to NETCONF XML content**

VPN Example – Transactional Config. Changes & Lock / Commit Features

NETCONF Implementations

- **Commercial**
  - MG-Soft
  - Oracle
  - Tail-f
  - WebNMS
  - YumaPro

- **Open Source**
  - Netopeer
  - Netconfx
  - OpenYuma
  - SDN Controllers
    - ***OpenDaylight***
    - ***ONOS***
  - Custom (Python)

NETCONF Hardware

- **“Major” NEMs (such as) on “newer devices” (not backwards compatible):**
  - Juniper
  - Cisco
  - Arista

YANG Implementations

- **Commercial**
  - MG-Soft
  - Tail-f (ConfD)
  - YumaPro

- **Open Source**
  - Pyang
    - ***Converts yang data models to Python hierarchies***
  - Yuma

SDN & NFV

- **NETCONF is a single protocol for managing configuration for BOTH traditional and SDN**
- **Fast, reliable, vendor-neutral solution**
- **Future Proof?**
  - – SDN can use NETCONF (“southbound”)
  - – High scale automation - NFV

OpenConfig

- **Informal industry collaboration of network operators**

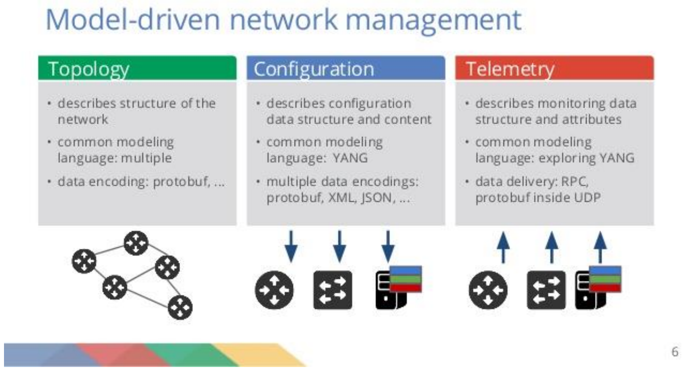
- **Focus: define vendor-neutral configuration and operational state models**
  - Goal of moving networks toward:
    - ***a more dynamic, programmable infrastructure***
    - ***by adopting SDN principles***
  - Adopted YANG data modeling language

- **Participants: Apple, AT&T, BT, Cisco, Comcast, Cox, Meta, Google, Juniper, Level3, Microsoft, Verizon/Yahoo!**

- **Public repo: https://github.com/openconfig**
- **Aims to complement standards efforts, including those in IETF**
  - Promotes the model that uses YANG over the NETCONF protocol
  - Faster than IETF

- **Two major concerns-**
  - Will vendors support OpenConfig?
  - Is OpenConfig flouting the IETF?
- **Killer app – Streaming telemetry**

Telemetry (INT)



Model-driven network management

Topology

- describes structure of the network
- common modeling language: multiple
- data encoding: protobuf, ...

Configuration

- describes configuration data structure and content
- common modeling language: YANG
- multiple data encodings: protobuf, XML, JSON, ...

Telemetry

- describes monitoring data structure and attributes
- common modeling language: exploring YANG
- data delivery: RPC, protobuf inside UDP

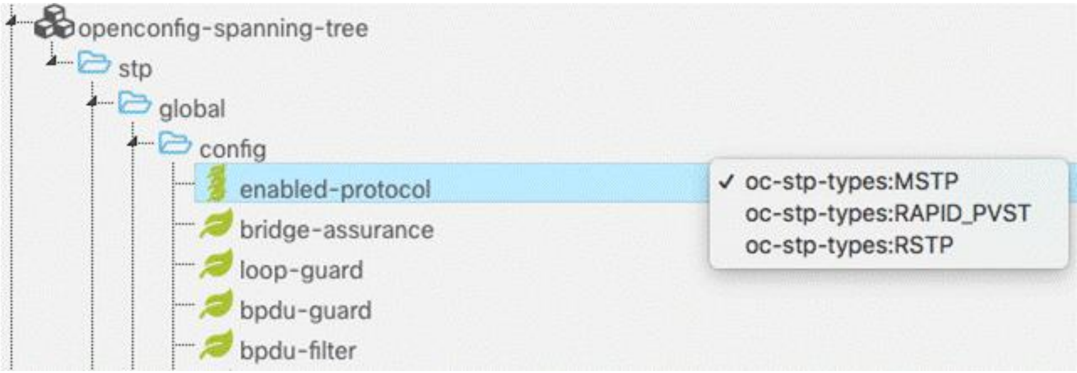
6

- **New paradigm for network monitoring**
  - – Data is streamed from devices continuously with efficient, incremental updates
  - – Operators can subscribe to the specific data items they need, using OpenConfig data models as the common interface

- **What is good and bad about this?**

- Places emphasis on models rather than APIs

- JunOS supports these data models
  - *BGP*
  - *Interfaces*
  - *LACP*
  - *LLDP*
  - *Local routing*
  - *MPLS*
  - *Network instance*
  - *Platform*
  - *Routing policy*
  - *VLAN*

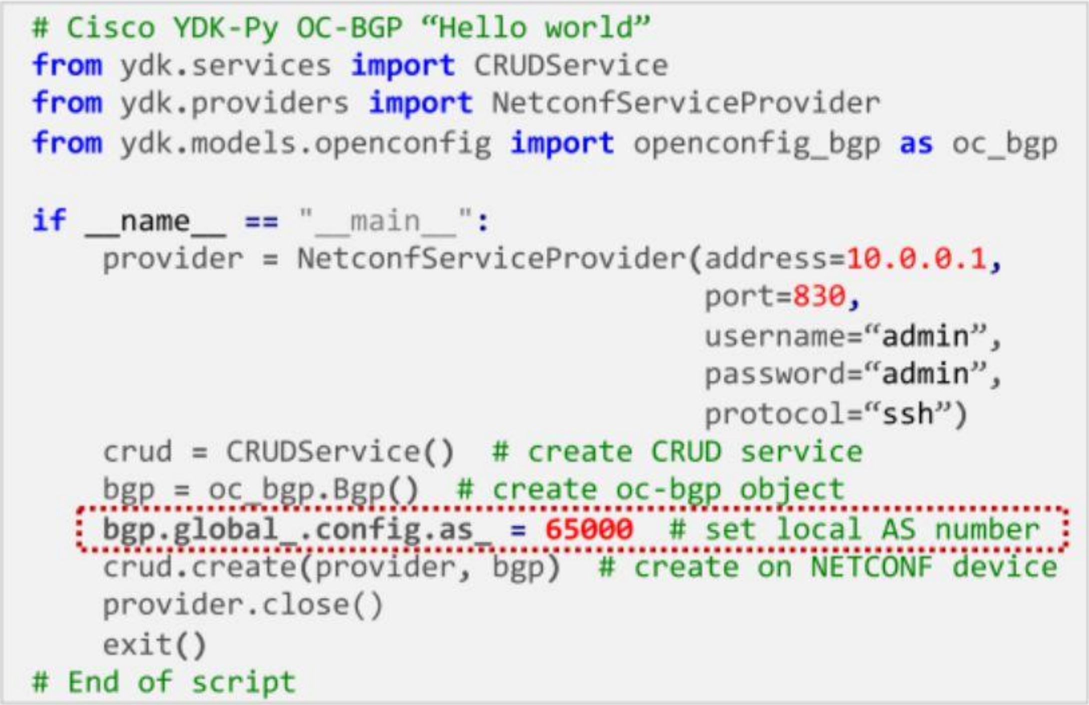


```
openconfig-spanning-tree
  stp
    global
      config
        enabled-protocol          ✓ oc-stp-types:MSTP
                                    oc-stp-types:RAPID_PVST
                                    oc-stp-types:RSTP
        bridge-assurance
        loop-guard
        bpdu-guard
        bpdu-filter
```

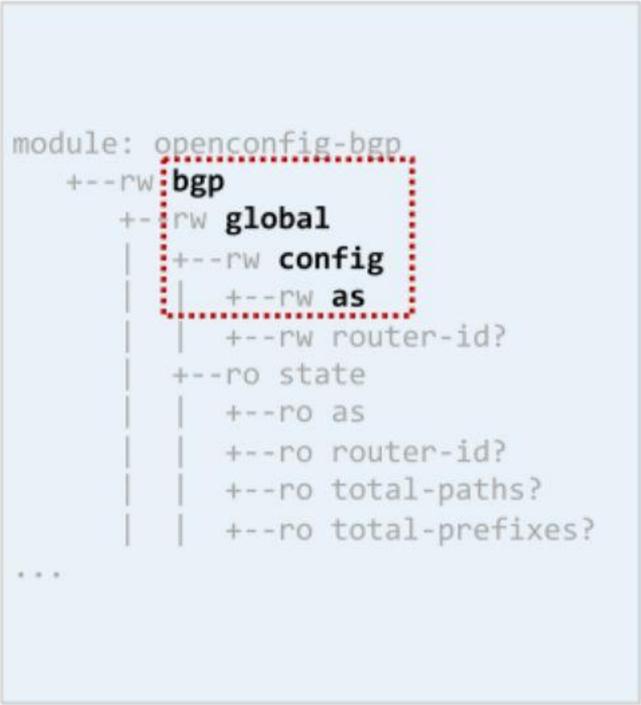
Cisco Devnet – YDK-Py

- **YANG Development Kit (YDK-PY)**
  - Provides API's that are modeled in YANG
    - ***Network programmability using data models***
    - ***API***
    - ***Python***
    - ***Netconf***
  - Reduces the learning curve of YANG
    - ***Abstracts protocol/encoding details***

A YDK-Py "Hello World" Using OpenConfig BGP



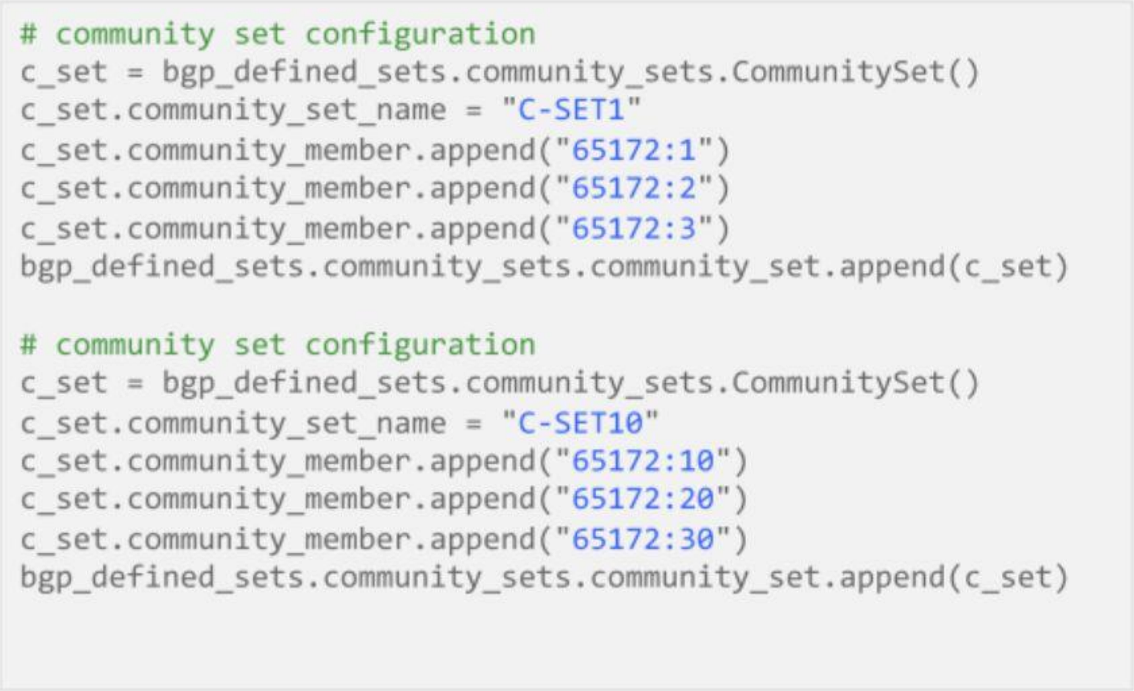
```
# Cisco YDK-Py OC-BGP “Hello world”
from ydk.services import CRUDService
from ydk.providers import NetconfServiceProvider
from ydk.models.openconfig import openconfig_bgp as oc_bgp

if __name__ == "__main__":
    provider = NetconfServiceProvider(address=10.0.0.1,
                                      port=830,
                                      username=“admin”,
                                      password=“admin”,
                                      protocol=“ssh”)
    crud = CRUDService()  # create CRUD service
    bgp = oc_bgp.Bgp()  # create oc-bgp object
    bgp.global_.config.as_ = 65000  # set local AS number
    crud.create(provider, bgp)  # create on NETCONF device
    provider.close()
    exit()
# End of script
```



```
module: openconfig-bgp
   +--rw bgp
      +--rw global
      |  +--rw config
      |  |  +--rw as
      |  |  +--rw router-id?
      |  +--ro state
      |  |  +--ro as
      |  |  +--ro router-id?
      |  |  +--ro total-paths?
      |  |  +--ro total-prefixes?
...
```

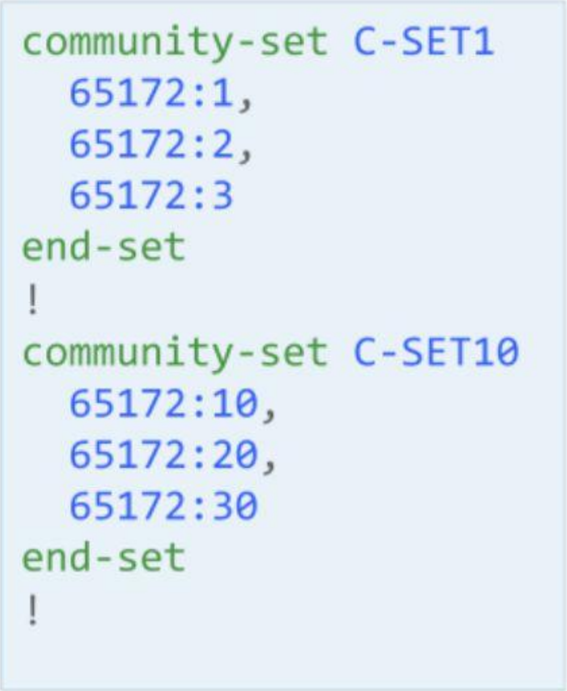
A YDK-Py Routing Policy Example

Python

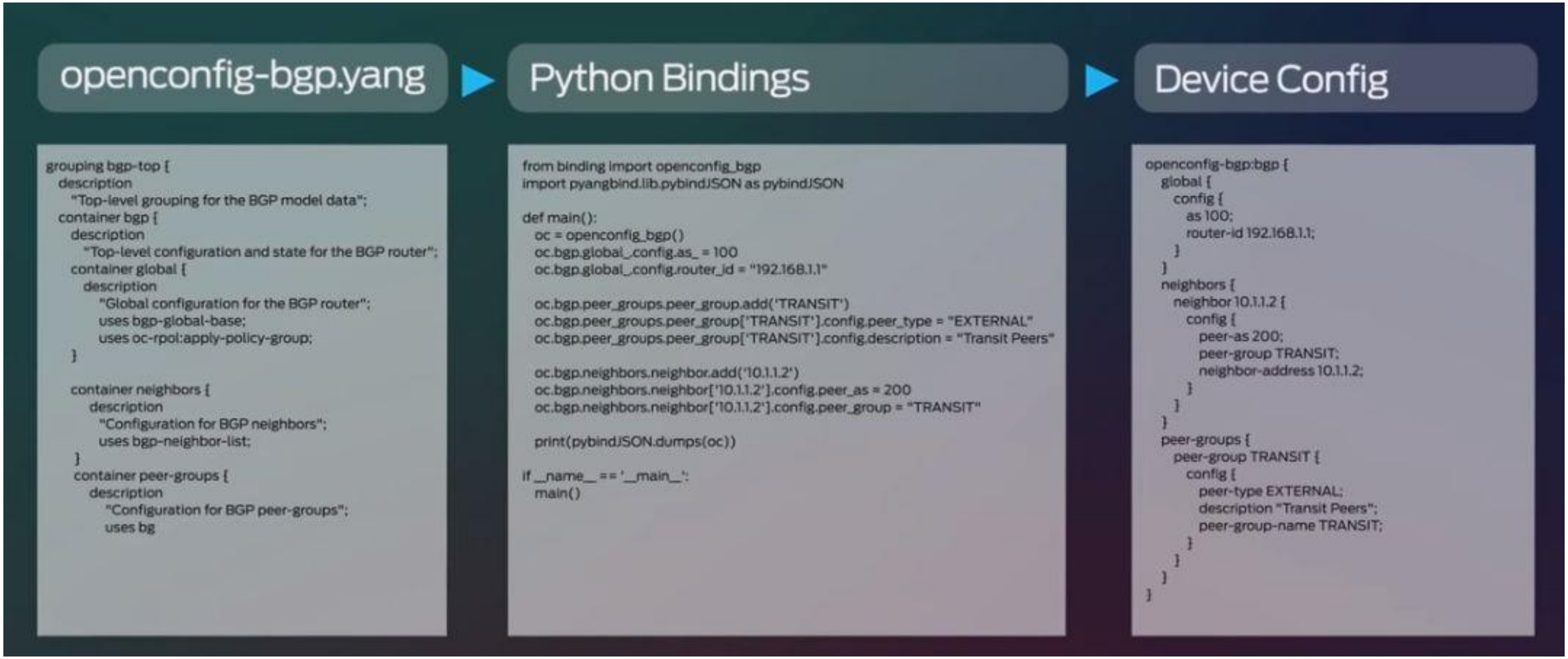


```
# community set configuration
c_set = bgp_defined_sets.community_sets.CommunitySet()
c_set.community_set_name = "C-SET1"
c_set.community_member.append("65172:1")
c_set.community_member.append("65172:2")
c_set.community_member.append("65172:3")
bgp_defined_sets.community_sets.community_set.append(c_set)

# community set configuration
c_set = bgp_defined_sets.community_sets.CommunitySet()
c_set.community_set_name = "C-SET10"
c_set.community_member.append("65172:10")
c_set.community_member.append("65172:20")
c_set.community_member.append("65172:30")
bgp_defined_sets.community_sets.community_set.append(c_set)
```

CLI



```
community-set C-SET1
  65172:1,
  65172:2,
  65172:3
end-set
!
community-set C-SET10
  65172:10,
  65172:20,
  65172:30
end-set
!
```



openconfig-bgp.yang ▶ Python Bindings ▶ Device Config

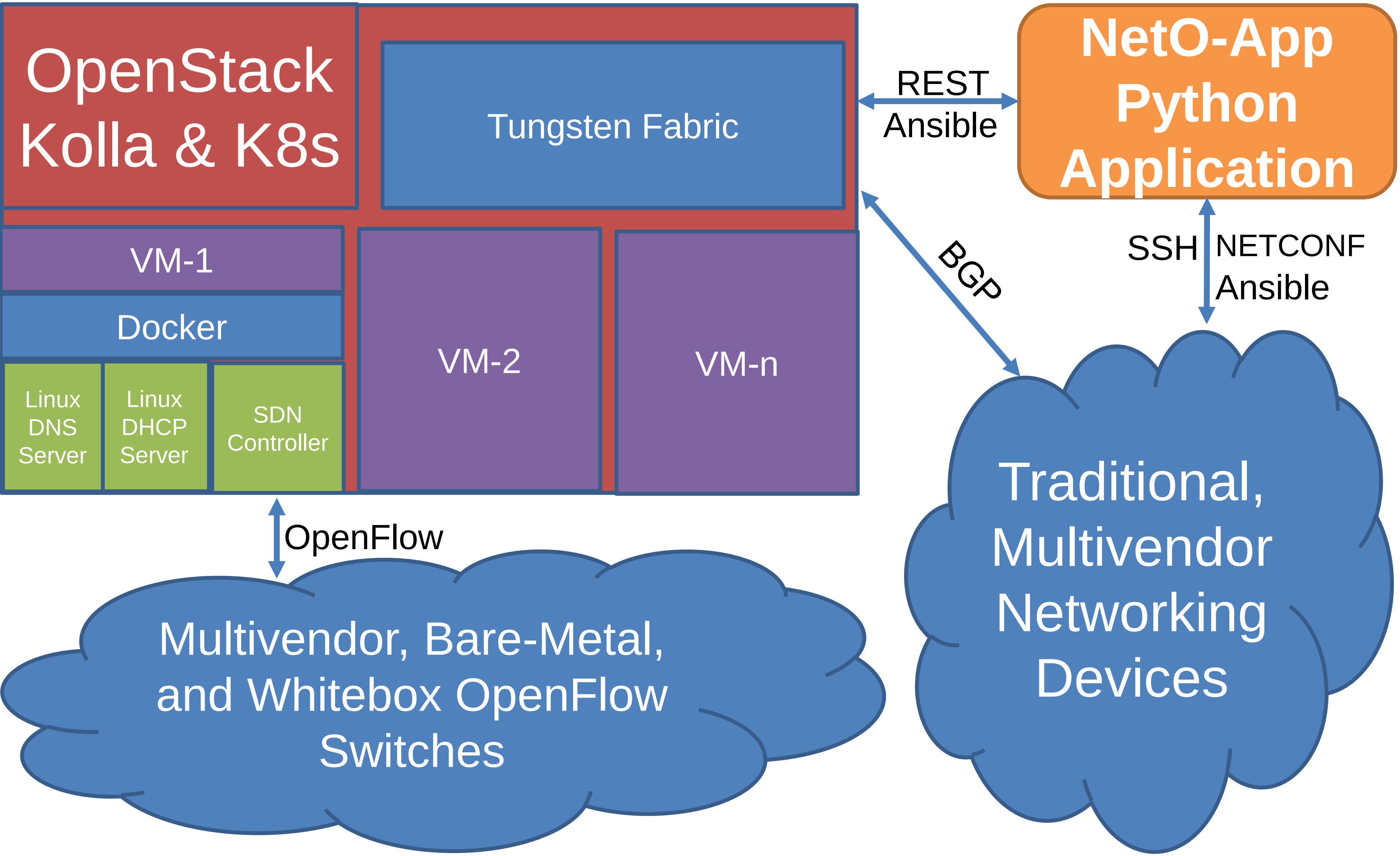
```
grouping bgp-top {
  description
    "Top-level grouping for the BGP model data";
  container bgp {
    description
      "Top-level configuration and state for the BGP router";
    container global {
      description
        "Global configuration for the BGP router";
        uses bgp-global-base;
        uses oc-rpol:apply-policy-group;
    }

    container neighbors {
       description
        "Configuration for BGP neighbors";
        uses bgp-neighbor-list;
    }
    container peer-groups {
       description
         "Configuration for BGP peer-groups";
         uses bg
```

```
from binding import openconfig_bgp
import pyangbind.lib.pybindJSON as pybindJSON

def main():
  oc = openconfig_bgp()
  oc.bgp.global_.config.as_ = 100
  oc.bgp.global_.config.router_id = "192.168.1.1"

  oc.bgp.peer_groups.peer_group.add('TRANSIT')
  oc.bgp.peer_groups.peer_group['TRANSIT'].config.peer_type = "EXTERNAL"
  oc.bgp.peer_groups.peer_group['TRANSIT'].config.description = "Transit Peers"

  oc.bgp.neighbors.neighbor.add('10.1.1.2')
  oc.bgp.neighbors.neighbor['10.1.1.2'].config.peer_as = 200
  oc.bgp.neighbors.neighbor['10.1.1.2'].config.peer_group = "TRANSIT"

  print(pybindJSON.dumps(oc))

if _name_ == '_main_':
  main()
```

```
openconfig-bgp:bgp {
  global {
    config {
      as 100;
      router-id 192.168.1.1;
    }
  }
  neighbors {
    neighbor 10.1.1.2 {
      config {
        peer-as 200;
        peer-group TRANSIT;
        neighbor-address 10.1.1.2;
      }
    }
  }
  peer-groups {
    peer-group TRANSIT {
      config {
        peer-type EXTERNAL;
        description "Transit Peers";
        peer-group-name TRANSIT;
      }
    }
  }
}
```

Questions?

Appendix



OpenStack Kolla & K8s

Tungsten Fabric

VM-1

Docker

Linux DNS Server

Linux DHCP Server

SDN Controller

VM-2

VM-n

REST

Ansible

NetO-App Python Application

BGP

SSH

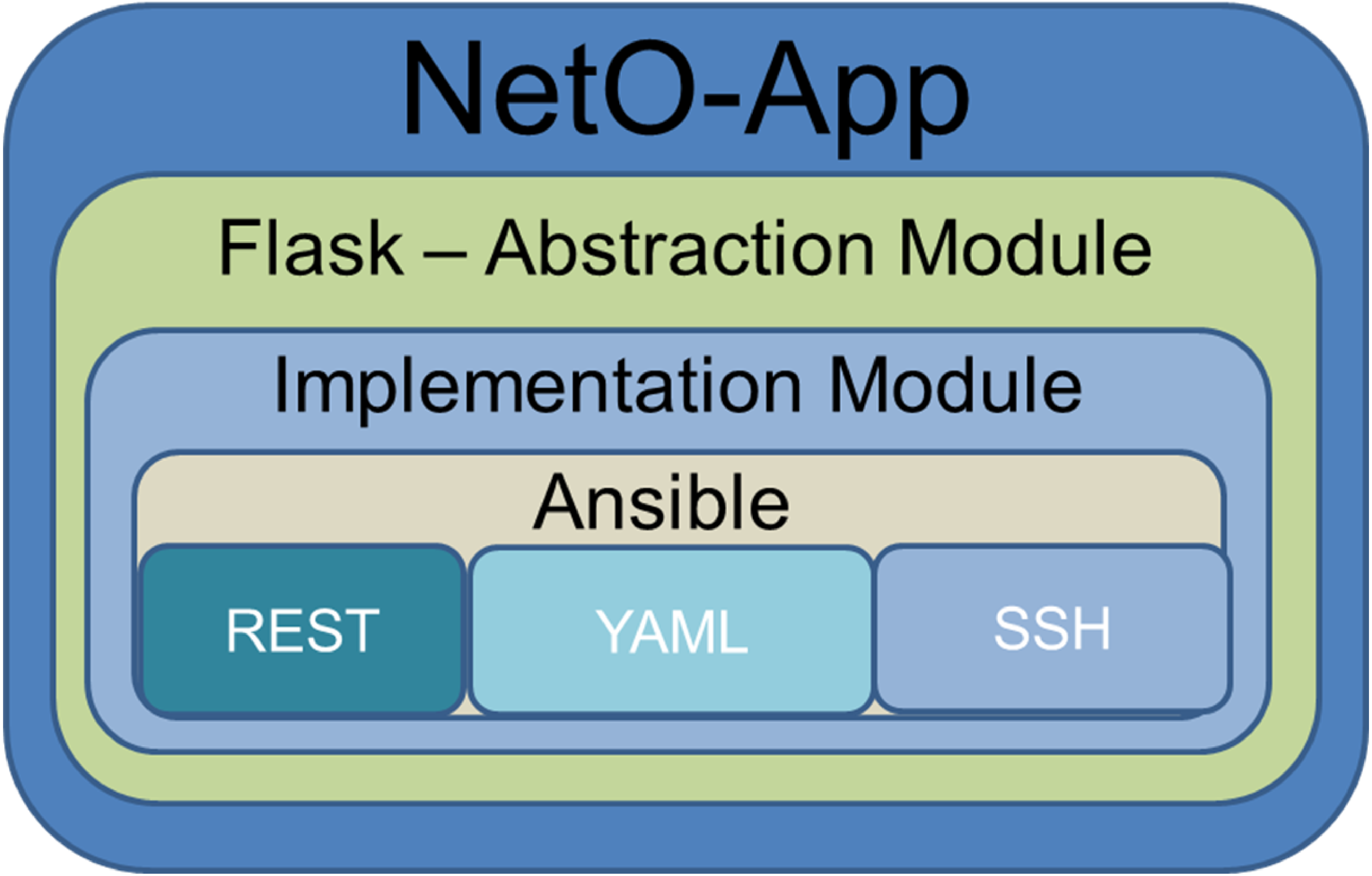
NETCONF

Ansible

OpenFlow

Multivendor, Bare-Metal, and Whitebox OpenFlow Switches

Traditional, Multivendor Networking Devices



NetO-App

Flask – Abstraction Module

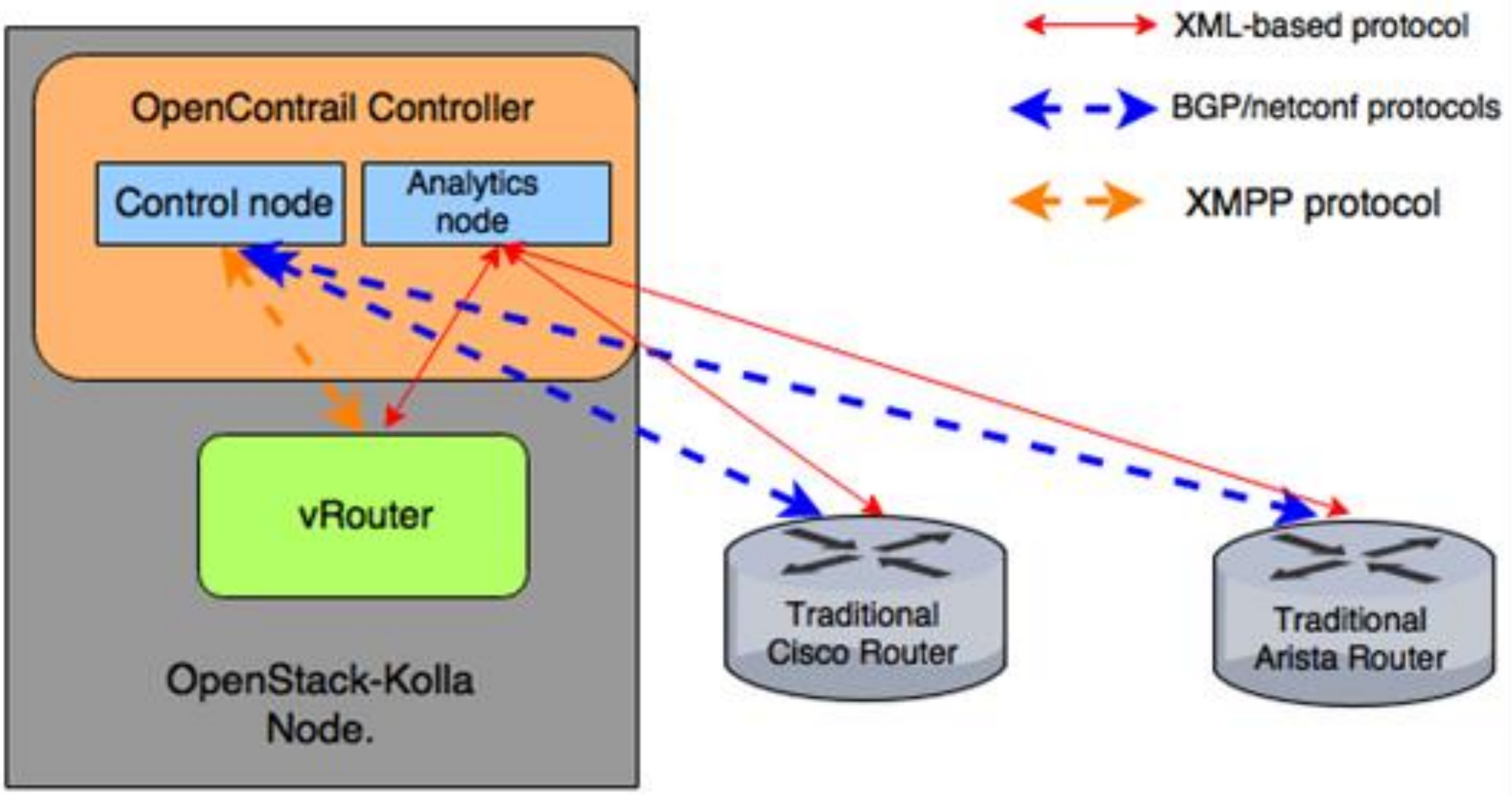
Implementation Module

Ansible

REST

YAML

SSH


XML-based protocol
BGP/netconf protocols
XMPP protocol
OpenContrail Controller
Control node
Analytics node
vRouter
OpenStack-Kolla Node.
Traditional Cisco Router
Traditional Arista Router