Tampere University

Zahra Golmohammadi

CENTRALIZED MODEL DRIVEN TRACE ROUTE MECHANISM FOR TCP/IP ROUTERS

Remote traceroute invocation using NETCONF API and YANG data model

Communication Systems and Networking
Master level
October 28

ABSTRACT

Zahra Golmohammadi: Centralized model driven trace route mechanism for TCP/IP routers
Master Level
Tampere University
Degree Programme
October 2019

During the recent years, utilizing programmable APIs and YANG data model for service configuration and monitoring of TCP/IP open network devices from a centralized network management system as an alternative to SNMP based network management solutions has gained popularity among service providers and network engineers.

However, both SNMP and YANG lacks any data model for tracing the routes between different routers inside and outside the network that has not addressed. Having a centralized traceroute tool provides a central troubleshooting point in the network. And rather than having to individually connect to each router terminal, *traceroute* can be invoked remotely on different routers. And the responses can be collected on the network management system.

The aim of this thesis is to develop a centralized traceroute tool called *Trace* that invokes traceroute CLI tool with a unique syntax from a centralized network management system on a TCP/IP router, traces the hops and BGP AS and measures RTT between a router and specific destination and returns the response back to the network management system. And evaluates the possibility of utilizing this traceroute tool along with YANG based network management solutions.

This implementation has shown that YANG based data models enables a unique syntax on the network management system for invoking traceroute command on different TCP/IP devices. This unique syntax can be used to invoke the traceroute CLI command on the routers with the different operating systems. And the evaluation has shown that using NETCONF as an API between the network management system and the network devices, enables the *Trace* to be utilized in YANG and NETCONF based network management solutions.

Keywords: network management, NETCONF, YANG, RESTCONF, CLI, RPC

PREFACE

Writing and implementing the Master thesis was not an easy work but having an opportunity to work on my favourite topic made it more pleasant.

I am thankful for all the advice and support on each step of my thesis provided by Prof. Alexander Pyattaev. I also would like to thank my manager Ms. Marja Seppä at BC Platforms Oy for all the flexibility and support that she provided during writing my Master thesis.

Many thanks to Prof. Dmitry Moltchanov and Prof. Sergey Andreev for all their support.

At the end, I would like to thank my father whose full trust in me, encouraged me to continue my studying in master's degree.

Tampere, 28th October 2019

Zahra Golmohammadi

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

ACL	Access Control List
AID	Area ID
API	Application Programming Interface
ARP	Address Resolution Protocol
BEEP	Blocks Extensible Exchange Protocol
BGP	Border Gateway Protocol
CE	Customer Edge
CLI	Command Line Interface
CRUD	Create, Read, Update, and Delete
DES	Data Encryption Standard
DNS	Domain Name Service
FIB	Forwarding Information Base
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
OC	Open Config
OID	Object Identifier
OSPF	Open Shortest Path First
IP	Internet Protocol
MAC	Media Access Control
UDP	User Datagram Protocol
URL	Uniform Resource Locator
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
MD5	Message-Digest Algorithm
MIB	Management Information Base for SNMP OIDs
MPLS	Multiprotocol Label Switching
NETCONF	Network Configuration Protocol
OID	Object Identifier for SNMP managed objects
OS	Operating System
PID	Process Identifier
REST	Representational State Transfer
RESTCONF	Representational State Transfer Configuration Protocol
RFC	Request for Comments
RID	Router Identifier
RPC	Remote Procedure Call
RTT	Round Trip Time
SMI	Structure of Management Information, is a language for creating MIBs
SNMP	Simple Network Management Protocol
SHA	Secure Hash Algorithm
SSH	Secure Shell Transport Layer Protocol
SSL	Secure Sockets Layer
TCL	Tool Command Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
PDU	Protocol Data Unit
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
XML	eXtensible Markup Language

1. INTRODUCTION

For decades, SNMP protocol and MIB data models have been used for the network management purpose. The SNMP based network management systems, provides configuration and monitoring capabilities for TCP/IP network infrastructure.

However, SNMP read-only MIBs makes configuration of network devices by SNMP almost impossible. From monitoring perspective, SNMP polling mechanism regularly produces unwanted UDP broadcast management traffic in the network, and this traffic increases the latency and load on the network devices and network management system. And from troubleshooting perspective, only limited vendors have developed troubleshooting tools with SNMP MIB data models for their products.

During the recent years, the notion of open network has changed the network management solutions. The open network and programmable APIs helps to provide a central access to the device operating system and developing services based on service needs rather than relying on the out of the box protocols.

The aim of this thesis was to develop a centralized traceroute tool called *Trace* that invokes the *traceroute* CLI tool with a unique syntax from a centralized network management system on a TCP/IP router, trace the hops, BGP AS, and measures RTT between a router and specific destination and returns the response back to the network management system. And evaluates the possibility of utilizing this traceroute tool along with YANG based network management solutions.

For this aim, chapter 2 starts with explaining the network device logical structure and existing solutions for configuring, monitoring and troubleshooting network devices. And later explores the concepts of new generation of protocols and solutions for configuring, monitoring and troubleshooting TCP/IP network devices such as NETCONF API, RESTCONF API and YANG.

Chapter 3 evaluates the performance of RESTCONF and NETCONF APIs for making configuration changes on a router's running config bases on YANG data models on the router with considering the number of TCP sessions, the number of transactions and the number of operations as metric.

Chapter 4 develops a *Trace* YANG module for finding the hops and the BGP AS path and measuring RTT between a router and a specific destination. And invokes the *Trace* remotely from network management system over NETCONF API on an open hardware router and collects the

response on the network management system. Also repeats the invocation by calling the *Trace* locally on the device CLI.

Chapter 5 provides a summary of RESTCONF API and NETCONF API performance. Compares the results of invoking the *Trace* remotely and locally, and analysis the possibility of utilizing the *Trace* tool along with YANG based network management solutions.

And finally, Appendix A describes the complete source code of the *Trace* YANG module in YANG language.

2. THEORETICAL BACKGROUND

Network management solutions are a way to provide monitoring, configuration and troubleshooting of network devices. Every TCP/IP network device based on the component's functionality can be categorize into management plane, control plane and data plane logical structures (Schudel, 2008). And the management of each device tightly depends on the device's management plane technologies and protocols. In this chapter the device logical structure, network traffic types, existing network management solutions and the concepts of YANG, NETCONF API and RESTCONF API are discussed.

2.1 Device logical structure

The management plane is an administrative interface for controlling the network device functionality. The CLI is a popular management plane technology that provides access to the device operating system for controlling the device functionalities. There are some protocols that can provides remote access to the device CLI such as Telnet and SSH (Schudel, 2008).

The management plane traffic is mainly generated for managing the device functionality. This traffic can be generated inside the device or come from a remote network management system (Schudel, 2008).

The control plane is responsible for performing the operational functionalities such as learning routes towards different destinations. The route processors inside network devices are a building block of the control plane (Schudel, 2008).

The control plane traffic is an operational traffic and is usually generated inside the network devices by means of control plane protocols like OSPF and ARP (Schudel, 2008).

The data plane is responsible for transmitting the end to end user traffic. Each network device line card is part of the data plane, it mainly receives the users IP packets from an ingress interfaces and forwards the packets to a right egress interface based on the FIB table (Schudel, 2008).

The figure below, shows the logical structure of a router:

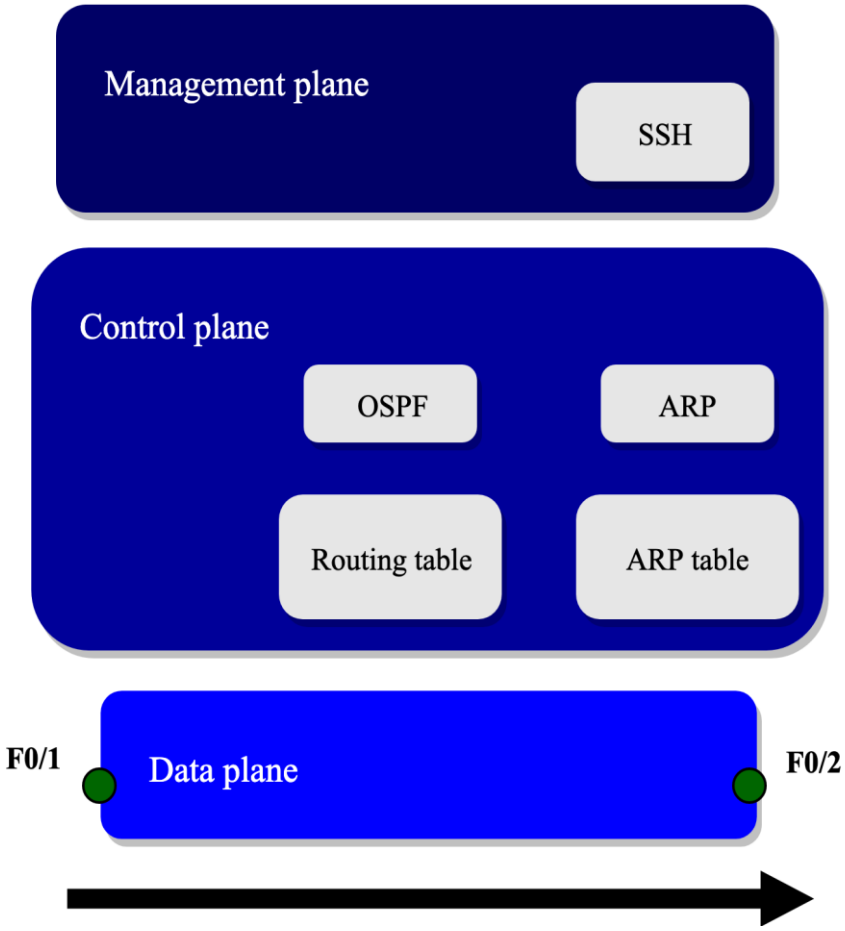

***Figure 1**. Device logical structure*

In this example, SSH as a management plane protocol provides remote access to the device CLI.

ARP as a control plane protocol creates the ARP table with binding the IP addresses and the MAC addresses. And OSPF as a control plane protocol is responsible for finding the shortest paths towards all known IP destinations and creating route entries for the routing table.

And finally, the data plane receives the IP packets from the ingress port F0/1, decapsulates the packet up to IP layer of TCP/IP protocol stack and based on the destination IP address, looks up for the best route match in the routing table and forwards the packet to the F0/2 egress port towards the next hop device.

As network management system mostly interact with the device management plane, recognizing the network traffic types based on the device logical structure can help to optimize the network management traffic by utilizing the most performant management plane protocols.

2.2 Existing network management solutions

2.2.1 CLI

The CLI is one of the most common technologies to access the management plane. The CLI provides a set of tools for configuration and troubleshooting of network devices. These tools are premade scripts on the device operating system. For this purpose, CLI receives commands as an input from the administrator and executes specific functions on the device operating system.

Each device CLI is accessible locally via console interface and remotely through TCP/IP Application layer protocols such as Telnet and SSH.

For decades, CLI has been used for configuring and troubleshooting network devices. Every network service requires a set of configuration changes on different network devices. As the CLI command of each device is specific to the vendor operating system, service deployment usually involves with a huge amount of manual device configuration.

CLI based scripts are a way to provide automation of device level and service level configuration commands, but with CLI scripts there is a possibility that each upgrade of the device operating system breaks the scripts.

Every device operating system has a wide range of the CLI tools that can used to facilitate the troubleshooting of the network infrastructure. For example, *traceroute* is one of the CLI troubleshooting tools that traces the hops and measures the RTT between a router and a specific destination. Almost all routers operating system has this tool built-in installed, but the *traceroute* syntax can vary in different device operating systems.

The CLI commands of network devices are usually specific to each vendor product. For example, Juniper JUNOS has a FreeBSD based operating system which runs on the Juniper router, switch and firewall products (Juniper, 2019). NX-OS is based on the Wind River Linux operating system and runs on the Cisco Nexus Ethernet and Fiber Channel switch products (Cisco, 2019). IOS XE is a Linux based operating system which runs on a specific Cisco router and switch products (Molenaar, 2016). The following table describes a short summary of the JUNOS, IOS XR, IOS XE and NX-OS network device operating systems:

Table 1. *Summary of the vendor device operating system*

Vendor Device	Vendor OS	Base OS
Juniper router, switch, firewall	JUNOS	FreeBSD
Cisco carrier-grade routers	IOS XR	Linux
Cisco ISR routers, Catalyst switches	IOS XE	Linux
Cisco Nexus switches	NX-OS	Wind River Linux

From configuration and troubleshooting perspective, the main limitation of the CLI is vendor dependency of commands syntax that makes automation of the CLI scripts fragile. But CLI tools are still a powerful way to configure and troubleshoot each network device individually.

.

2.2.2 **SNMP**

In the early days of the internet establishment, SNMP was developed and standardized by the IETF as a solution for configuring and monitoring TCP/IP network devices and traffic flow.

SNMP has a client/server architecture for network infrastructure management. Each component of a network device is an object and has an identifier. SNMP manager is a software component on the network management system for collecting information from network objects. And SNMP agent is a software component on each network device for collecting local data for the SNMP manager. In order to collect information about OIDs, SNMP manager over regular time intervals creates a GET UDP packet about specified items inside an OID with destination port 161 and sends a broadcast message in the network, all SNMP agents listening on port 161 UDP receive the packet and create a unicast response message about requested items and send it back to the SNMP manager, SNMP manager receives and processes the messages. And then for collecting information about the next OID SNMP manager repeat the process by sending a GETNEXT request to the SNMP agents (Qian and Lu, 2010).

In the security point of view producing UDP broadcast traffic makes SNMP vulnerable. Thus, IETF has added some security enhancement in SNMP version 3 including SHA/MD5 and DES

encryption to secure the authentication and message transmission between the SNMP agent and the SNMP manager (Davis, 2004).

In addition to the pulling mechanism, SNMP has a mechanism for pushing the object state changes to the SNMP manager by means of creating and sending a Trap message to the UDP port 162. However, Trap messages usually do not contain enough information about the problem and manual investigation is needed to clarify the problem (Claise, Clarke and Lindblad, 2019).

SNMP protocol provides network configuration by means of SNMP SET messages on writable MIBs. For configuring a network device writable MIB, the SNMP manager creates a unicast SET message containing values about specific items, sends the message to UDP port 161 of the SNMP agents, SNMP agent receives the message, writes the configuration changes on the MIBs, creates and sends a response message back to the SNMP manager (Qian and Lu, 2010).

Although the SNMP protocol has gained popularity amongst many network vendors and has been supported on a wide range of network devices. But network operators usually leverage the SNMP to collect data from TCP/IP devices on their network monitoring system. Despite the SNMP configuration capability, SNMP has never been widely utilized for network device configuration purpose due to some limitations. SNMP does not have any discovery method for the supported MIBs on the device, most of the SNMP MIBs are read-only and SNMP does not support configuration rollback (Wallin and Wikström, 2011). In addition, the SNMP protocol does not have any mechanism to invoke remote commands on the device CLI.

Based on the discussion in this section, there is a need to have an alternative network management solution that provides a vendor independent service configuration and troubleshooting mechanism and reduces the management plane broadcast traffic inside the network infrastructure.

2.3 **YANG**

In 2010, IETF released YANG in RFC 6020 (Bjorklund, 2010) as a common language between the network management system and network devices.

2.3.1 **YANG objectives**

YANG is a new approach for managing TCP/IP network devices that provides a common language between the network management system and network devices in order to have a vendor independent syntax for making configuration changes on the network devices, collect the state data from network devices and invoke remote CLI commands on the device CLI.

2.3.2 **YANG language**

YANG is a user-friendly language for describing a device functionality and creating a data model based on the device functionalities in a hierarchical structure.

YANG provides reusability of these data models which means a YANG data model can be used in other YANG data models.

Based on (Bjorklund, 2010), some well-known syntax keywords of the YANG language for creating a data model are including module, imported, containers, lists, leaf, and type:

- **Module**: Module describes the name of each YANG data model.
- **Imported**: Imported keyword defines enables reusing of a YANG module inside another YANG data model.
- **Include**: Include is used for reusing the YANG submodules inside a YANG data model.
- **Containers**: YANG containers are a way to categorize device element into different groups.
- **Lists**: YANG lists, defines the elements in each container
- **Leaf**: Each individual element within a network device can be defined as a leaf inside a YANG data model.
- **Leaf-list**: Leaf-list is a Leaf with multiple instances.
- **Type**: Each leaf needs to have a specific data type that can be defined with the type keyword.

2.3.3 **YANG data model**

The YANG data models consist of a set of easy to read syntax and notations defining the desired configuration and service state in a hierarchical tree structure. These data models can be used by network devices as a model-based structure for storing the configuration and state data.

The YANG augmentation and deviation feature makes the creation of new YANG data models easier (Bjorklund, 2010). A set of YANG data models can be combined and be used as a YANG module.

YANG data models vary depending on the device roles and services in the network infrastructure. For example, interface, VLAN and ACL data models are considered as device level data model but VRF and MPLS VPN are considered as service level data model.

For developing and sharing the YANG data models and YANG modules in a community, network vendors, IETF and OC have established three groups. The OC is a community that many vendors such as Google, Facebook, and Verizon collaborating to develop and share YANG data models. Vendor data models are used for each specific network device but IETF and OC define vendor neutral YANG data models. During the past years, these organizations have been created plenty of device level and service level YANG data models. And all YANG modules supported on the network devices can be retrieved or discovered from the device by means of NETCONF and RESTCONF API protocols.

IETF and vendor proprietary YANG modules source code is available on Yang Github repository (GitHub, 2019) and OC YANG modules source code is available on OpenConfig Github repository (OpenConfig, 2019).

2.3.4 **YANG data**

YANG data is a data for configuring and monitoring network devices based on YANG modules structure within the network devices. The YANG data can be transmitted between the network management system and network device in XML and JSON data encoding formats.

JSON and XML are two text-based data encoding format that is used by network devices and the network management system as a mutual data exchange format. JSON and XML can be converted to YANG data and vice versa.

For service configuration from a network management system, the desired state of each device can be defined in JSON or XML format and transmitted to the device. Within the device, these

data formats will be converted to YANG data models. The followings describe a YANG data model with one container describing the OSPF elements:

```
container ospf {
 leaf area {
 type int;
   mandatory true;
 }
}
```

The corresponding YANG data for this data model can be encoded in XML format as following:

```
<ospf>
 <area>0</area>
</ospf>
```

YANG data models provide a way to have vendor independent data models on network management system for configuring, monitoring and troubleshooting network devices. But as each vendor devices can have its own functionality and set of protocols, community effort for sharing YANG data models based on open hardware devices is needed to make YANG data models independent of vendor devices.
Although YANG provides the data models on network devices but in order to transmit the YANG data from network management system to each network device and vice versa there is a need to API protocols.

2.4 **NETCONF**

To address the need of transmitting the YANG configuration data from a network management system to network devices, and YANG state data from the network devices to the network management system IETF has released NETCONF and RESTCONF APIs. This section will discuss about NETCONF API concepts in detail.

2.4.1 **NETCONF history**

Despite the SNMP capability for network configuration, SNMP has never gained popularity among network operators as a configuration mechanism as SNMP protocol has limitations in device MIB discovery and most of the available MIBs are read-only.

For monitoring network devices, SNMP has a polling mechanism in which the SNMP manager sends a periodic broadcast requests to the network devices and receives replies about the latest changes in the network devices. And this whole procedure produces lots of unwanted traffic and increases the load not only on the network management system but also on the infrastructure devices.

In May 2003, IETF established a NETCONF team for developing a network configuration protocol as a solution to overcome the limitations induced by traditional network configuration technologies. And later in December 2006, NETCONF protocol standardized by IETF as a standard protocol for network management, and network vendors started to support NETCONF protocol on their products (Radford et al., n.d.).

2.4.2 **NETCONF objectives**

NETCONF is designed to overcome SNMP limitations in configuration and monitoring of network devices. NETCONF API capability exchange method enables finding information about the supported YANG modules within each network device (Radford et al., n.d.). And, NETCONF API and YANG monitoring capabilities, eliminates the UDP broadcast traffic in the network, by generating and pushing notifications about state data changes to the network management system over TCP unicast session (Claise, Clarke and Lindblad, 2019).

NETCONF provides configuration changes on network devices in a transactional approach. Which means, configuration changes remain consistent amongst network devices, and in the case of failure during configuration changes, NETCONF reverts the previous device configuration back (Wallin and Wikstrom, 2011).

2.4.3 **NETCONF architecture**

NETCONF protocol has a client/server architecture for managing the network devices and network services. The main components of NETCONF are including NETCONF manager and NETCONF agent. NETCONF manager is a central component for managing multiple network devices and NETCONF agent is a software component on each network device.

NETCONF protocol make use of a combination of multiple protocols which can be defined as a protocol stack. The NETCONF manager performs RPC operations in XML format based on the device YANG data model over TCP protocol on network device datastores. The following figure shows NETCONF protocol stack layers:

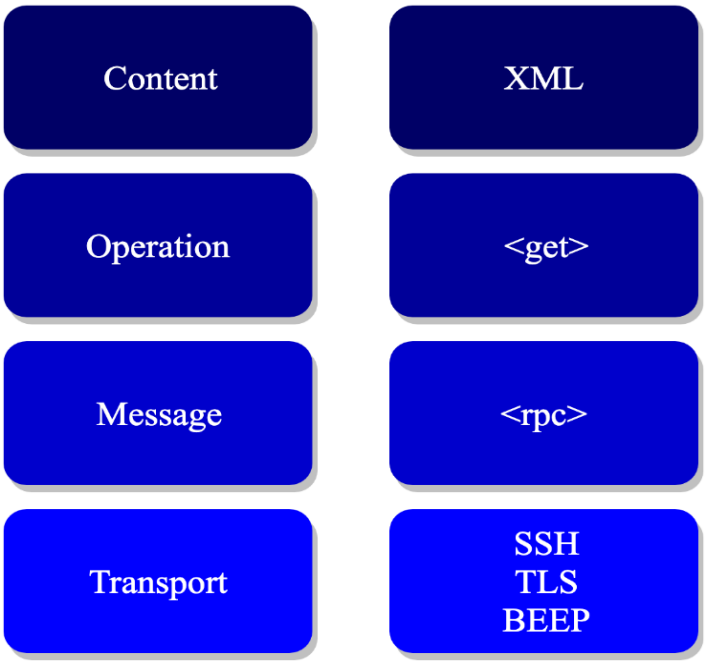

***Figure 2**. NETCONF protocol stack*

The content layer of this protocol stack defines the encoding format of the configuration and state data that NETCONF manager and NETCONF agent agreed on to transmit. Which means all NETCONF configuration and state data can be transferred in XML format, based on device YANG data modules (Wallin and Wikstrom, 2011).

The operation layer defines the RPC operations that a NETCONF manager can perform on the NETCONF agent to configure the network devices (Wallin and Wikstrom, 2011).

The NETCONF operations are carried out within special XML elements and each element can have different attributes. Three main elements of XML for carrying messages are including <rpc>, <rpc-reply>, <rpc-error>. The <rpc> is used for carrying messages, <rpc-reply/> is used for carrying response messages and <rpc-error/> is used for carrying failure messages (Bjorklund, 2010).

NETCONF is a connection-oriented protocol and supports three transport protocols SSH, BEEP and SSL. BEEP provides a bidirectional session between the NETCONF manager and the NETCONF agent (Lear and Crozier, 2006). In TLS connectivity, the NETCONF manager sets up a TLS connection with the NETCONF agent TCP port 6513 (Badra, 2009). However, SSH protocol supports multiple TCP channels over one SSH session, which enables NETCONF API to establish one SSH session with multiple channels to TCP port 830 for configuration and monitoring purposes (Claise, Clarke and Lindblad, 2019).

The NETCONF manager always initiates the SSH session to the NETCONF agent TCP port 830. In this process, NETCONF manager and NETCONF agent, first exchange their supported capabilities inside a <hello> elements. Upon the session establishment, the NETCONF manager possess the supported capabilities on the NETCONF agent, and can send requests with the <rpc> element to the NETCONF agent, the request first goes to the NETCONF agent queue and after processing the request, NETCONF agent sends the replies with <rpc-reply> back to the NETCONF manager (Bjorklund, 2010).

The figure below shows the NETCONF messages between the NETCONF manager and the NETCONF agent for performing an operation:

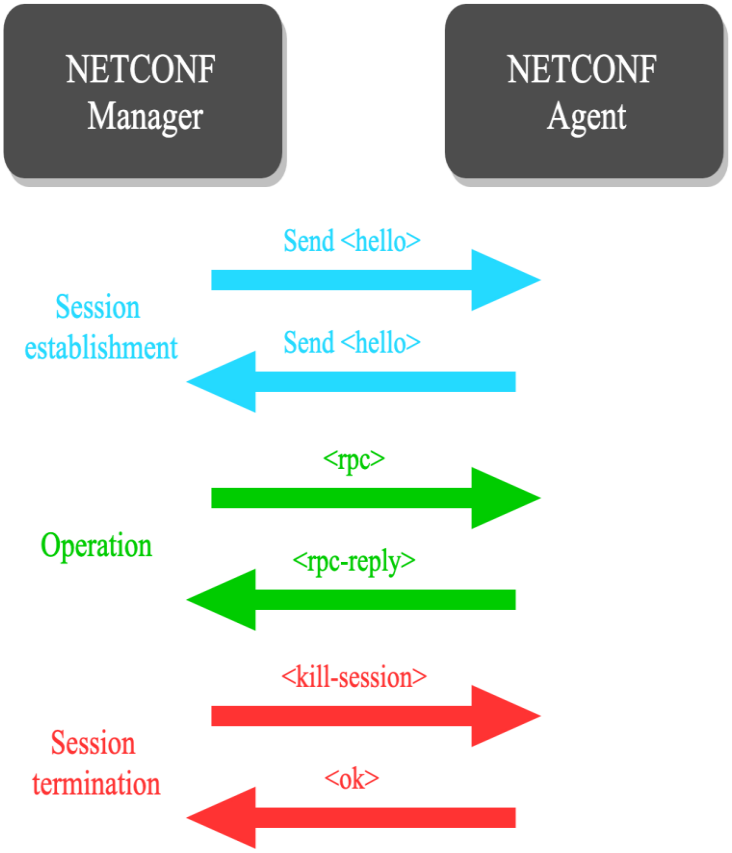

Figure 3. *NETCONF operation*

The following table based on Claise (Claise, Clarke and Lindblad, 2019) describes a list of NETCONF RPC operations:

Table 2. *NETCONF operation*

Operation	Description
<close-session>	Terminates a NETCONF session
<commit>	Commits the configuration changes in candidate datastore to the running datastore
<copy-config>	Removes the configuration from the candidate datastore and add the new configuration
<create-subscription>	Subscription to the YANG modules
<delete-config>	Deletes the configuration from the candidate datastore
<discard-changes>	Clears the configuration changes from the candidate datastore
<edit-config>	Edits a datastore configuration
<filter>	Filters the required data
<get>	Retrieves the state and configuration data
<get-config>	Retrieves the configuration data
<kill-session>	closes a NETCONF session
<lock>	Locks a datastore during configuration changes
<unlock>	Unlock a datastore configuration for all the session writes
<validate>	Validates a datastore configuration
<target>	Applies the configurations on a specific datastore

2.4.4 **NETCONF for configuration**

The NETCONF API provides access to the device datastores in order to change the device initial configuration to a desired state. Datastores inside the network devices contain YANG data. Based on Wallin and Wikstrom (Wallin and Wikstrom, 2011), NETCONF operations are done on three conceptual datastores:

- **Running datastore:** Running datastore is a mandatory datastore on each NETCONF agent which represent the current active configuration within each device.
- **Candidate datastore:** Candidate datastore is an optional data store which holds a set of configuration changes on NETCONF agent. And commit operation synchronize the running datastore with the candidate datastore.
- **Startup datastore:** Startup datastore is an optional data store which holds the device initial configuration that is used upon the device boot up.

The below figure depicts the NETCONF direct model of configuration. In this model, the NETCONF operations are applied directly to the running datastore.

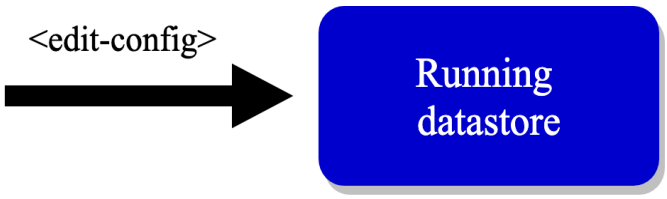

***Figure 4**. NETCONF direct model operation*

The Figure 5 depicts the NETCONF candidate model operation. In this model NETCONF edit-config operation makes configuration changes on the candidate datastore and then NETCONF commit operation, applies the configuration changes to the running datastore.

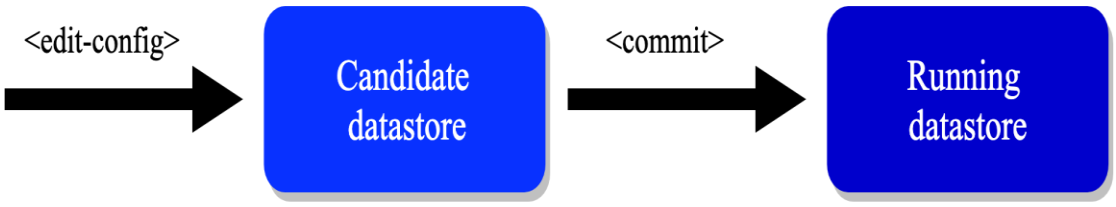

***Figure 5.** NETCONF candidate model operation*

And in startup model operation, running datastore configuration is copied with <copy-config> operation into the startup datastore as shown in the figure 6.

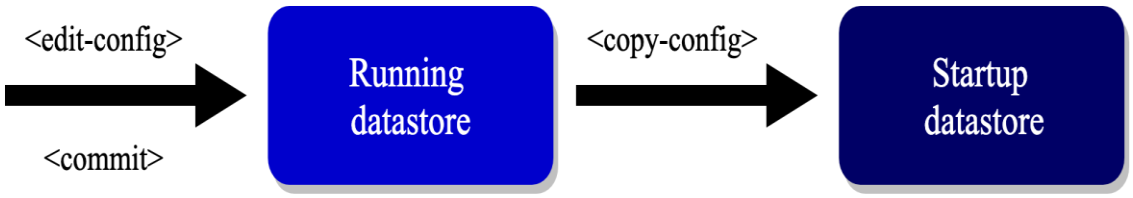

***Figure 6**. NETCONF startup model operation*

The figure below depicts a full chain of the NETCONF operations for configuring the network devices in candidate model based on Moberg (Moberg, 2015). In the first step, for mutual exclusion, NETCONF locks the datastore with <lock> operation.

***Figure 7**. NETCONF operation chain*

Afterward, with <delete-config> the NETCONF manager clears the candidate datastore and then with <edit-config> applies the new configuration. And after configuration validation, NETCONF manager commits the changes to the running datastore. After committing, the NETCONF manager waits for NETCONF agent confirmation, if NETCONF manager does not receive the confirmation message in the defined time period, it rollbacks the configuration to the devices previous state.

All configuration changes with the NETCONF on YANG data models are based on transactional approach. Based on Claise and Clarke (Claise, Clarke and Lindblad, 2019), each NETCONF transaction has four properties:

- **Atomicity:** A set of configuration changes must be applied at the same time on the NETCONF agents.
- **Consistency:** The order of sending configuration changes to a NETCONF agent does not make any difference, NETCONF agent is responsible for executing configuration changes in the right order.
- **Independence:** Each NETCONF transaction is independent of the other transactions.
- **Durability**: Configuration changes in network devices are persistent.

2.4.5 **NETCONF for monitoring**

Based on RFC 7923 (Voit, Clemm and Gonzalez Prieto, 2016), IETF has defined two approaches for network device monitoring with NETCONF, periodic notification, and on-change notification. Both approaches leverage push mechanism for notifying the NETCONF manager.

For receiving periodic notifications, NETCONF manager subscribes to the YANG data models on the network device. Based on the subscription policies, over regular time intervals, NETCONF agent sends notifications about the specified variables in XML format to the NETCONF manager (Claise, Clarke and Lindblad, 2019).

On the other hand, for receiving on-change notifications, NETCONF manager subscribes to a specific YANG data model on the device and upon a change on the YANG data model, NETCONF agent sends a notification about specified variables to the NETCONF manager in XML format (Claise, Clarke and Lindblad, 2019).

This monitoring approach requires persistent SSH connections between the NETCONF manager and the NETCONF agents. But all notifications are unicast messages destinated to the NETCONF manager TCP port 186 and, using SSH protocol can be make notifications more secure (Claise, Clarke and Lindblad, 2019). However, in congestion condition TCP congestion detection mechanism can make accessing to the real-time state data challenging.

2.5 RESTCONF

RESTCONF is a programmable API for accessing YANG modules on the network devices by sending configuration YANG data and retrieving state YANG data in XML/JSON format over HTTP/1.1 via REST methods. This section will discuss about RESTCONF API concepts in more detail.

2.5.1 REST API

API is a method of communication between two entity of code. REST is a stateless API that works on top of the HTTP protocol and leverage HTTP methods for accessing the resources (Gazarov, 2016).

Based on Gazarov, all HTTP requests has some specific properties; every HTTP request contains all necessary information, the server does not need to store the state information about previous requests and a failure of one request does not influence other requests (Gazarov, 2016).

The figure below shows the REST client and HTTP server interaction with REST methods:

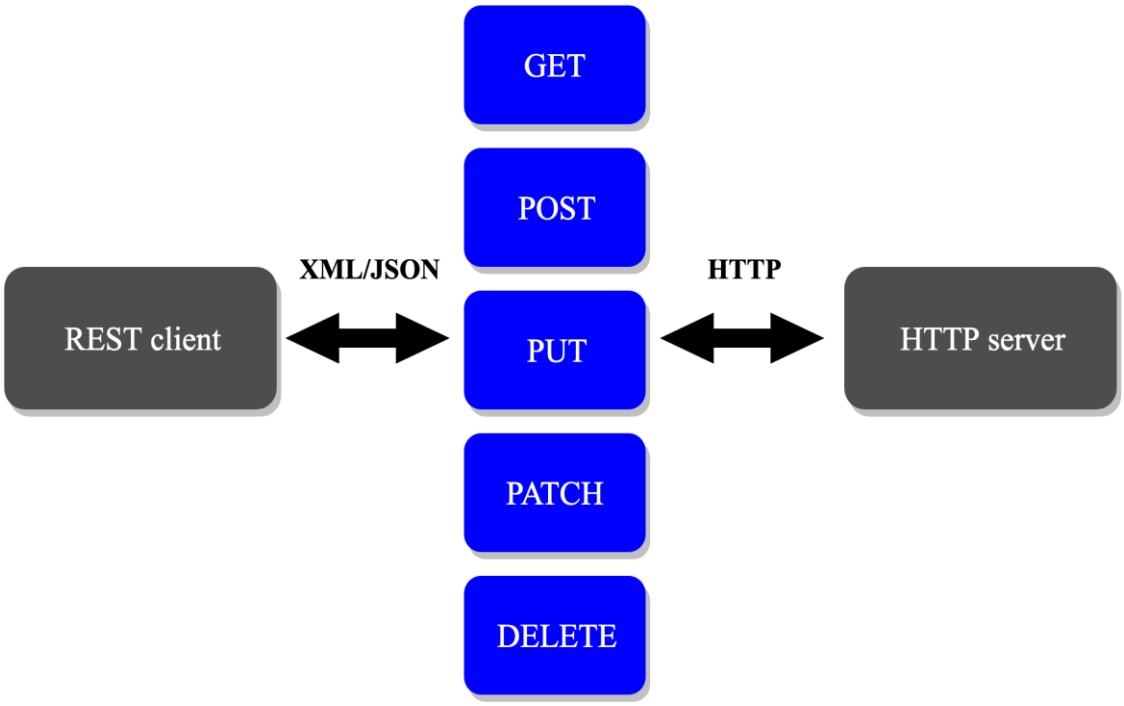

***Figure 8**. HTTP methods*

In this architecture, HTTP client sends an HTTP GET request for retrieving the data from the HTTP server URL resources. HTTP protocol supports both XML and JSON data encoding formats, and HTTP response data encoding format can be defined in the HTTP request header.

For sending data to an HTTP server, an HTTP client makes use of the POST method. And the sending data can be encoded in XML or JSON structure.

The HTTP PUT method provides a way to update the data resource in the HTTP server by replacing the data with new data. Also, HTTP PATCH methods are used for updating the data resource in the HTTP server by modifying the data.

The HTTP DELETE method is used by the HTTP client to delete the data resources on the HTTP server.

2.5.2 **RESTCONF objectives**

RESTCONF (Bierman, Bjorklund and Watsen, 2017) is an IETF standardized HTTP-based protocol that provides access to YANG data models on network devices. This protocol leverages HTTP REST API methods and XML/JSON encoding formats to read state data from the device and write configuration on the device datastore.

The RESTCONF make use of set of protocols that can be defined as a protocol stack. The following figure shows the RESTCONF protocol stack:

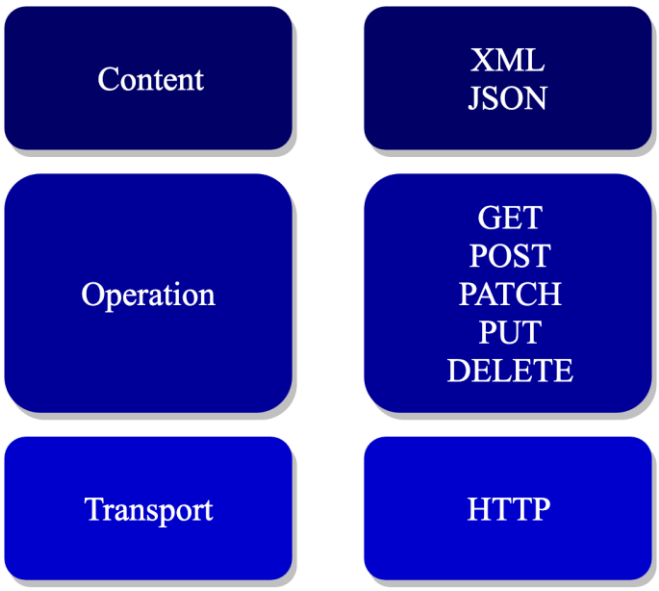

Figure 9. *RESTCONF protocol stack*

This protocol stack is made of three layers. The content layer defines the encoding format for transmitting the data between the RESTCONF manager and the RESTCONF agent.

Operation layer defines the HTTP REST API methods that are used by RESTCONF API to make configuration changes on the network devices (Bierman, Bjorklund and Watsen, 2017).

For the transport layer, RESTCONF manager makes use of the stateless HTTP protocol to connect to the RESTCONF agent TCP port 8080 (Bierman, Bjorklund and Watsen, 2017).

2.5.3 **RESTCONF architecture**

RESTCONF has a client/server approach for managing network devices. The main components of RESTCONF are including RESTCONF manager and RESTCONF agent. The RESTCONF manager is a web application on the network management system that is responsible for handling the URLs and sending the request to the network devices. And RESTCONF agent is a piece of software on the network devices that receives the requests from the NTCONF manager. The following figure shows the RESTCONF architecture for managing network devices:

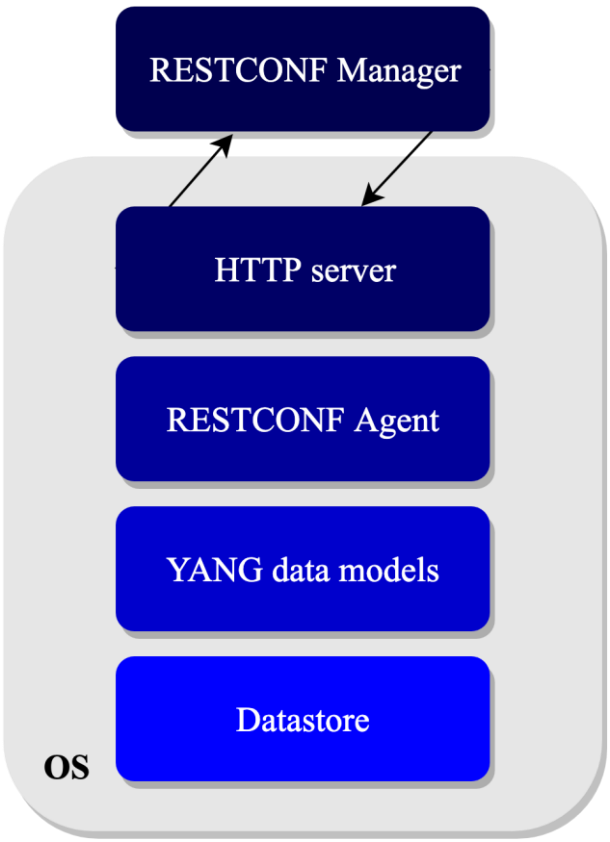

Figure 10. *RESTCONF architecture*

In this picture, the RESTCONF manager sends HTTP requests to the HTTP server on device operating system. The RESTCONF agent receives the request, applies the configuration changes on the YANG data models and all changes are applied to directly to the device running configuration.

The RESTCONF URLs are a way to access YANG data models on the network devices. Based on Bierman (Bierman, Bjorklund and Watsen, 2017) the RESTCONF URL schema consists of address, root, data, operations, module, container, leaf and options as following:

- **Address:** IP address of the RESTCONF Agent.
- **Root:** Root is the RESTCONF request starting point.
- **Data:** Data specifies the resource types; data, operations. and yang-library-version. Data is used for accessing state and configuration data, and operations are used for performing RPC operation in the data resource.
- **Module:** Module defines the base YANG module name.
- **Container:** Container specifies the YANG module name.
- **Leaf:** Leaf specifies the YANG module name.
- **Options:** Optional field for filtering data resource.

RESTCONF API leverages REST methods to access all URL resources on the network devices. In order to find the supported HTTP REST method on the YANG data models, an HTTP request with OPTIONS method can be sent to the device URL resource.

2.5.4 **RESTCONF and NETCONF comparison**

The RESTCONF and NETCONF protocols have a similar approach for making configuration changes on the network devices. The RESTCONF API leverages HTTP REST methods and the NETCONF API leverages RPC operation. The following table based on RFC 8040 (Bierman, Bjorklund and Watsen, 2017) describes the RESTCONF and NETCONF operations:

Table 3. *RESTCONF and NETCONF operations*

RESTCONF Operations	NETCONF Operations
GET	get, get-config
POST	edit-config create operation
PUT	edit-config replace operation
PATCH	edit-config merge operation
DELETE	edit-config delete operation

The RESTCONF supports both XML and JSON data encoding formats while the NETCONF only supports XML format. However, both JSON and XML are text-based data encoding formats that cannot be parsed on the ASICs and requires device CPU involvement (Pepelnjak, 2019)

Both RESTCONF and NETCONF operations for making configuration changes are done based on the transactional approach which enables rollback to the device previous state in case of failure (Radford et al., 2018).

Unlike NETCONF, RESTCONF does not have multiple datastores. But RESTCONF supports the datastores configured by NETCONF protocol. Which means If the device is configured with the NETCONF running datastore, the RESTCONF can commit the changes directly to the device running datastore. And if the device is configured with the NETCONF running and candidate data stores, the RESTCONF first commits the changes to candidate datastore and then commit them to the running datastore (Radford et al., 2018).

NETCONF supports locking and unlocking datastore to keep datastores consistent during the transaction but RESTCONF does not support either of those (Radford et al., 2018).

3. NETCONF AND RESTCONF API EMULATION

The aim of this chapter is to evaluate the performance of the NETCONF and RESTCONF APIs for configuring YANG data models on a router running config by considering the number of transactions, the number of operations and TCP connections as metric.

3.1 Emulation environment

The Figure 11 and Figure 12 respectively depicts the NETCONF and RESTCONF API emulation environment:

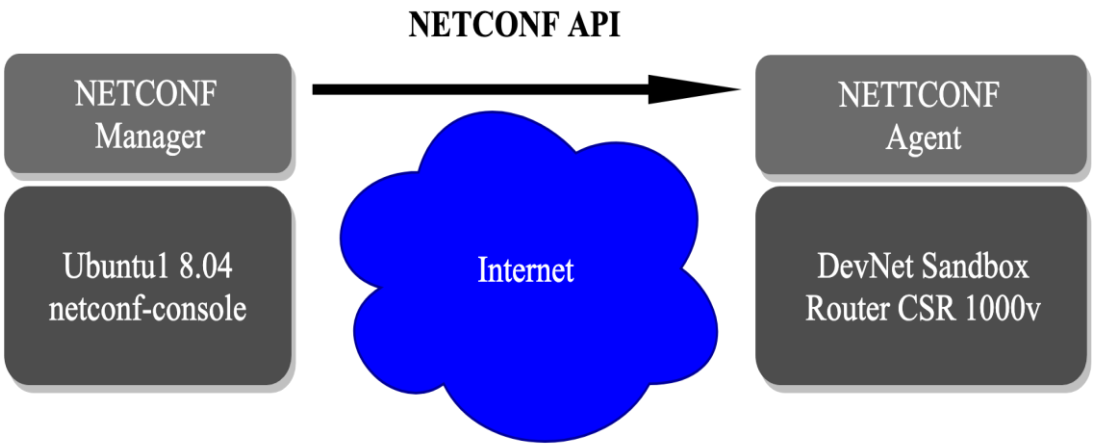

Figure 11. *NETCONF emulation environment*

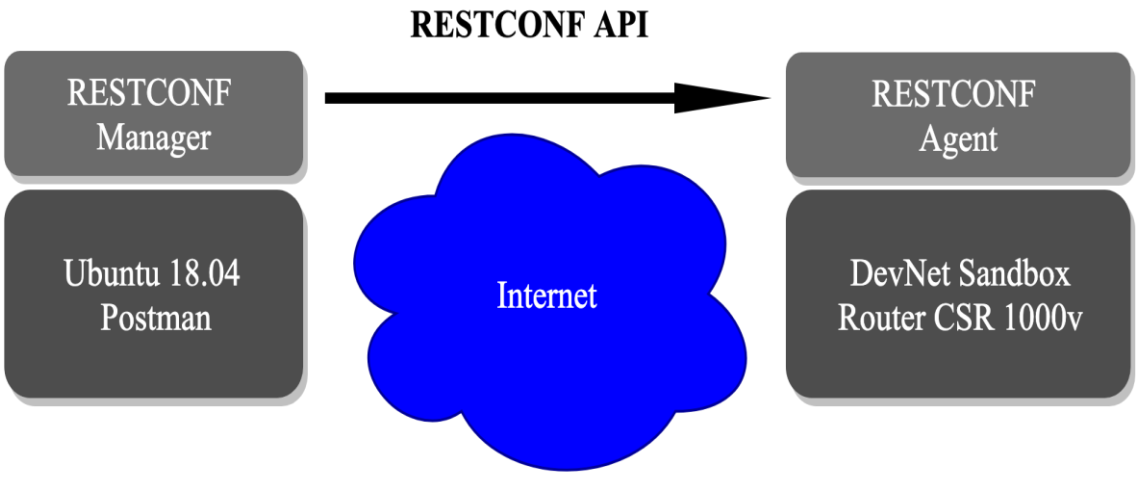

Figure 12. *RESTCONF emulation environment*

In this emulation environment the NETCONF manager and RESTCONF manager are on an Ubuntu 18.04 (Ubuntu, 2019) virtual machine and the NETCONF agent and RESTCONF agent is on a router CSR 1000v located on the internet. The following section describes the tools has been used to prepare the emulation environment.

Python (Python.org, 2019) is a dependency for the NETCONF manager implementation and has been configured on Ubuntu 18.04.

Pyang (Bjorklund, n.d.) is a YANG tool written in Python language that in this emulation used for checking out the YANG modules in tree structure.

Netconf-console is a command line tool on the network manager that makes API calls to the NETCONF agent on the network devices (Volf, n.d.).

Postman (Postman, 2019) is a REST client which is used in this emulation as a RESTCONF manager in order to make REST API calls to the RESTCONF agent.

JSONLint is a JSON format validator (Jsonlint, 2019) which has used to validate the structure of JSON YANG data before sending the YANG data to the RESTCONF agent.

Cisco router CSR1000v supports both NETCONF and RESTCONF protocols and in this thesis a free version of this device which is available on Cisco DevNet Sandbox (Devnetsandbox. 2019) has been utilised as NETCONF and RESTCONF agent.

Wireshark (Wireshark.org, n.d.) is a network analyser and in this thesis has used to capture the packets from the NETCONF manger and RESTCONF manager to the NETCONF agent and RESTCONF agent.

Connectivity to the NETCONF agent has been verified by sending a NETCONF hello message in XML format to the NETCONF agent via netconf-console and receiving a list of YANG modules metadata from the router CSR 1000v.

```
netconf-console --host=restconf.com –hello
```

Connectivity to the RESTCONF agent verifies by retrieving the /restconf entry point from the CSR 1000v via Postman. Based on Cisco (Cisco, 2018), the RESTCONF entry point metadata information can be retrieved from the router by sending an HTTP request to the router's IP address or DNS address:

```
GET /.well-known/host-meta HTTP/1.1
    Host: restconf.com:9443
    Accept: application/xrd+xml
```

A separate request based on the Cisco (Cisco, 2018), has been used to retrieve the list of YANG modules metadata from the router:

```
GET /restconf/data/ietf-yang-library:modules-state HTTP/1.1
     Host: restconf.com:9443
     Accept: application/yang-data+json
```

3.2 **Emulation parameters**

This section evaluates the performance of NETCONF and RESTCONF APIs for writing a set of configuration changes on the routers running config.

A set of configuration data for creating a loopback interface and OSPF routing instance has prepared in XML and JSON format respectively for the NETCONF and RESTCONF APIs, based on the YANG modules metadata retrieved from the router. The table 4 describes the interface configuration parameters:

Table 4. *Interface configuration parameters*

Name	Status	IP address	Subnet mask	API
Loopback6	Up	192.168.1.6	255.255.255.240	NETCONF
Loopback7	Up	192.168.1.7	255.255.255.240	RESTCONF

And the table 5 describes the OSPF routing configuration parameters:

Table 5. *OSPF configuration parameters*

PID	RID	IP prefix	Subnet mask	AID	API
6	192.168.1.6	192.168.1.6	255.255.255.240	600	NETCONF
7	192.168.1.7	192.168.1.7	255.255.255.240	700	RESTCONF

The table below describes a summary of the main YANG modules that has used in this emulation to make configuration changes on the router.

Table 6. *Summary of YANG modules*

YANG module	Description
ietf-interfaces	IETF module for managing the device interfaces
Cisco-IOS-XE-ospf	Vendor module for managing the OSPF routing domain

Configuration has been done in three steps by using the configuration parameters described in table 4 and table 5 as a reference.

Interface configuration

For creating a loopback interface with NETCONF API, the edit-config operation with the YANG data in XML format described in program 1 has sent directly to the device running config via netconf-console:

```
  <?xml version="1.0" encoding="UTF-8"?>
  <rpc    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"    message-
id="1">
   <edit-config>
    <target>
     <running/>
    </target>
   <config>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
           <interface>
               <name>Loopback6</name>
               <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-
if-type">
                 ianaift:softwareLoopback
               </type>
               <enabled>true</enabled>
               <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                   <address>
                       <ip>192.168.1.6</ip>
                       <netmask>255.255.255.240</netmask>
                   </address>
               </ipv4>
           </interface>
       </interfaces>
   </config>
  </edit-config>
  </rpc>
```

Program 1. *Interface configuration YANG data over NETCONF API*

The Loopback6 interface configuration has verified over NETCONF API call by sending the following request to the router:

```
netconf-console --host=restconf.com --get-config
-x interfaces/interface[name='Loopback6']
```

And for the same configuration changes, the RESTCONF POST operation with the YANG data in JSON format described in the program 2 has sent to the device running config via Postman:

```
POST /restconf/data/ietf-interfaces:interfaces HTTP/1.1
     Host: restconf.com:9443
     Accept: application/yang-data+json
{
    "ietf-interfaces:interface": {
          "name": "Loopback7",
          "type": "iana-if-type:softwareLoopback",
          "enabled": true,
          "ietf-ip:ipv4": {
                "address": [{
                      "ip": "192.168.1.7",
                      "netmask": "255.255.255.240"
                }]
          }
    }
```

Program 2. *Interface configuration YANG data over RESTCONF API*

The interface Loopback7 configuration has verified via sending a RESTCONF API call to the router:

```
GET /restconf/data/Cisco-IOS-XE-native:native/interface HTTP/1.1
      Host: restconf.com:9443
      Accept: application/yang-data+json
```

Based on Wireshark packet captures, both NETCONF and RESTCONF APIs established only one TCP connection to the device to create the interface Loopback6 and Loopback7.

Interface and OSPF routing configuration

In this step before making any configuration changes, the previous configuration changes have removed from the device.

For configuring the Loopback6 interface and the OSPF routing instance with NETCONF API, the edit-config operation with the XML YANG data described in the program 3 has sent to the device running config via netconf-console:

```
  <?xml version="1.0" encoding="UTF-8"?>
  <rpc    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"    message-
id="1">
    <edit-config>
  <target>
   <running/>
   </target>
   <config>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
           <interface>
               <name>Loopback6</name>
               <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-
if-type">
                 ianaift:softwareLoopback
               </type>
               <enabled>true</enabled>
               <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                   <address>
                       <ip>192.168.1.6</ip>
                       <netmask>255.255.255.240</netmask>
                   </address>
               </ipv4>
           </interface>
       </interfaces>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <router>
  <ospf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ospf">
  <id>6</id>
  <router-id>192.168.1.6</router-id>
  <network>
  <ip>192.168.1.7</ip>
  <mask>255.255.255.240</mask>
  <area>600</area>
  </network>
  </ospf>
  </router>
  </native>
         </config>
    </edit-config>
  </rpc>
```

Program 3. *Interface and OSPF routing configuration YANG data over NETCONF API*

The Loopback6 interface configuration has verified over NETCONF API call by sending the following request to the router:

```
netconf-console --host=restconf.com --get-config
-x interfaces/interface[name='Loopback6']
```

And the OSPF with process ID 6 configuration has verified by sending a NETCONF API call to the router:

netconf-console --host=restconf.com **--get-config -x "native/router/ospf"**

But for the same configuration changes, RESTCONF agent URL described below did not accept the configuration data within one POST operation.

```
POST /restconf/data/ HTTP/1.1
     Host: restconf:9443
     Accept: application/yang-data+json
```

So, the evaluation continued with creating the interface Loopback7 and OSPF instance over separate POST operations described in program 4 and program 5:

```
POST /restconf/data/ietf-interfaces:interfaces HTTP/1.1
     Host: restconf:9443
     Accept: application/yang-data+json
{
    "ietf-interfaces:interface": {
          "name": "Loopback7",
          "type": "iana-if-type:softwareLoopback",
          "enabled": true,
          "ietf-ip:ipv4": {
                "address": [{
                      "ip": "192.168.1.7",
                      "netmask": "255.255.255.240"
                }]
          }
    }
```

Program 4. *Interface configuration YANG data over RESTCONF API*

```
POST /restconf/data/Cisco-IOS-XE-native:native/router HTTP/1.1
     Host: restconf.com:9443
     Accept: application/yang-data+json
{
        "Cisco-IOS-XE-ospf:ospf": [
            {
                "id": 7,
                "router-id": "192.168.1.7",
                "network": [
                    {
                        "ip": "192.168.1.7",
                        "mask": "255.255.255.240",
                        "area": 700
                    }
                ]
            }
        ]
}
```

Program 5. *Interface configuration YANG data over RESTCONF API*

The interface Loopback7 configuration has verified via sending a RESTCONF API call to the router:

```
GET /restconf/data/Cisco-IOS-XE-native:native/interface HTTP/1.1
      Host: restconf.com:9443
      Accept: application/yang-data+json
```

The OSPF configuration changes has been verified by sending a GET request to the RESTCONF agent on the router:

```
GET /restconf/data/native/router/ospf HTTP/1.1
      Host: restconf.com:9443
      Accept: application/yang-data+json
```

Based on Wireshark packet capture, the NETCONF API only established one TCP session to make the whole configuration changes, but the RESTCONF operations needed two separate POST operations over two TCP sessions to create the interface and OSPF instance on the router's running config.

Interface configuration and OSPF routing modification

In this step the previous interface configuration has been removed from the device but the OSPF instance configuration has preserved.

For configuring the Loopback7 interface and modifying the OSPF routing instance with NETCONF API, the edit-config operation with the XML YANG data described in the program 6 has been sent to the device via netconf-console:

```
  <?xml version="1.0" encoding="UTF-8"?>
  <rpc    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"    message-
id="1">
    <edit-config>
  <target>
   <running/>
   </target>
   <config>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
           <interface>
               <name>Loopback6</name>
               <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-
if-type">
                 ianaift:softwareLoopback
               </type>
               <enabled>true</enabled>
               <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                   <address>
                       <ip>192.168.1.6</ip>
                       <netmask>255.255.255.240</netmask>
                   </address>
               </ipv4>
           </interface>
       </interfaces>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <router>
  <ospf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ospf">
  <id>6</id>
  <router-id>192.168.1.6</router-id>
  <network>
  <ip>192.168.1.6</ip>
  <mask>255.255.255.240</mask>
  <area>0</area>
  </network>
  </ospf>
  </router>
  </native>
         </config>
    </edit-config>
  </rpc>
```

Program 6. *Interface configuration and OSPF routing modification YANG data over NETCONF API*

For the same configuration changes, the following RESTCONF agent URL did not accept the configuration data in over one PATCH operation:

```
PATCH /restconf/data/ HTTP/1.1
     Host: restconf.com:9443
Accept: application/yang-data+json
```

So, the evaluation continued with creating the interface Loopback7 over one POST operation described in program 7 and modifying the OSPF routing instance over a separate PATCH operation described in program 8.

```
POST /restconf/data/ietf-interfaces:interfaces HTTP/1.1
     Host: restconf.com:9443
     Accept: application/yang-data+json
{
    "ietf-interfaces:interface": {
          "name": "Loopback7",
          "type": "iana-if-type:softwareLoopback",
          "enabled": true,
          "ietf-ip:ipv4": {
                "address": [{
                      "ip": "192.168.1.7",
                      "netmask": "255.255.255.240"
                }]
          }
    }
```

Program 7. *Interface configuration YANG data over RESTCONF API*

```
  PATCH      /restconf/data/Cisco-IOS-XE-native:native/router/ospf=20
HTTP/1.1
       Host: restconf.com:9443
       Accept: application/yang-data+json
  {
         "Cisco-IOS-XE-ospf:ospf": [
             {
                 "id": 7,
                 "router-id": "192.168.1.7",
                 "network": [
                     {
                         "ip": "192.168.1.6",
                         "mask": "255.255.255.240",
                         "area": 0
                     }
                 ]
             }
         ]
  }
```

Program 8. *OSPF routing modification YANG data over RESTCONF API*

Based on Wireshark packet capture, the NETCONF API only established one TCP session to make the whole configuration changes, but the RESTCONF API sent the POST and PATCH operations over two separate TCP sessions for interface configuration and OSPF routing modification.

4. TRACE YANG MODULE DEVELOPMENT

This section describes the development of *Trace* YANG module; a centralized traceroute tool that invokes traceroute CLI tool with a unique syntax from a centralized network management system on a TCP/IP router, traces the hops and BGP AS and measures the RTT between a router and specific destination and returns the response back to the network management system.

4.1 Development and emulation environment

The figure below depicts the development and emulation environment. In this environment, the NETCONF manager and NETCONF agent are on an Ubuntu 18.04 virtual machine:

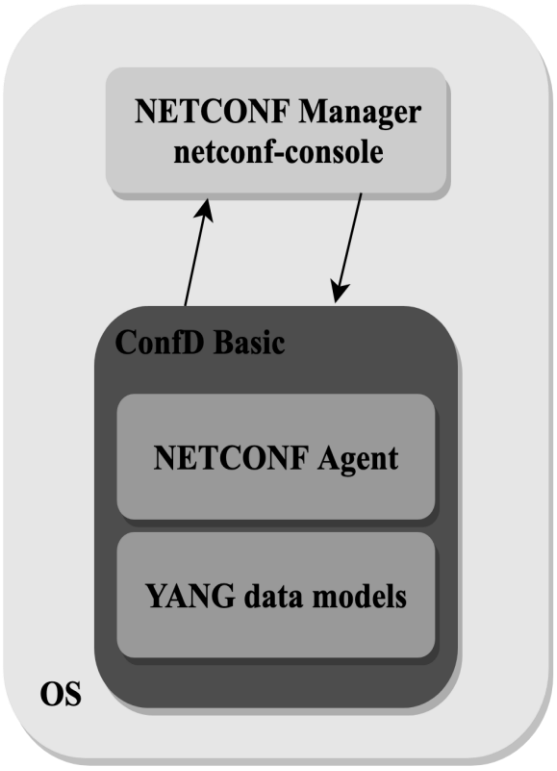

Figure 13. *YANG development and testing environment*

Python is a dependency for the NETCONF manager implementation and has configured on the Ubuntu 18.04.

Pyang mostly used for showing the YANG modules in tree structure and YANG module syntax validation.

Paramiko (Paramiko, n.d.) package is SSH2 protocol written in Python and has been used in this section to provide SSH client on the NETCONF manager

Netconf-console is a command line tool on the network manager that makes API calls to the NETCONF agent on the network devices.

Wireshark network analyser in this section has used to capture the packets from the NETCONF manager to the NETCONF agent.

ConfD Basic (CONFD USER GUIDE, 2018) is a free software that enables YANG modules compilation, creating database schemas based on YANG modules, populating database schemas with YANG data, and supports NETCONF agent protocol.

tailf-common YANG module (Tailf GitHub, 2019) on the ConfD Basic has imported to the *Trace* YANG module in order to invoke the traceroute shell on the device CLI.

Connectivity between the NETCONF manger and the NETCONF agent verified by sending a hello message via netconf-console. As both NETCONF manager and NETCONF agent are on the same operating system, NETCONF hello message sent locally to the NETCONF agent on Confd Basic as following:

```
netconf-console –hello
```

4.2 Trace module development

This section describes the procedure for developing, compiling and testing YANG *Trace* module. The *Trace* YANG module intended to invoke the traceroute CLI command with a set of options. For this reason, six YANG containers have been defined. Destination container defines the destination IP address or domain name, mtu container defines the maximum packet size for each traceroute packet, ttlMax container defines the maximum hopes that the traceroute packet can be transmitted, bgpAsPath container enables finding BGP AS numbers on the path to the destination and sourceAddress enables sending traceroute packets from a specific IP address on the router.

The source code of the *Trace* YANG module has given in Appendix A. The program has been compiled using ConfD Basic:

```
confdc -c -o trace.yang --fail-on-warnings
```

And status of the *Trace* module including metadata data information verified by using the command:

```
confdc --get-info trace.fxs
```

The program below describes the compiled YANG module in tree structure:

```
module: trace

  rpcs:
    +---x traceRoute
       +---w input
       |  +---w destination      string
       |  +---w mtu?             ttlRange
       |  +---w ttlMax?          mtuRange
       |  +---w bgpAsPath?       boolean
       |  +---w sourceAddress?   string
       +--ro output
          +--ro traceResult?   String
```

Program 9. *Trace YANG module in tree structure*

For executing the traceroute CLI command based on the YANG modules container on the router, one shell script has been written in program 10.

```
  #!/bin/sh

  context=$2
  destination=$4
  mtu=$6
  ttl=$8
  bgpAsPath=$10
  sourceAddress=$12
  hostname > hostID

  if  [ "$bgpAsPath" =  "false" ] && [ -z $sourceAddress ]; then
  traceroute $destination --mtu $mtu -t $ttl  > traceResult
  elif  [ $bgpAsPath =  false ]; then
      traceroute $destination --mtu $mtu -t $ttl -s $sourceAddress >
traceResult
  elif  [ $bgpAsPath =  true ] && [ -z $sourceAddress ]; then
      traceroute $destination --mtu $mtu -t $ttl  -A  > traceResult
  elif  [ $bgpAsPath =  true ] && [ -n $sourceAddress ]; then
      traceroute $destination --mtu $mtu -t $ttl -s $sourceAddress -A
> traceResult
  fi

  echo "traceResult '\n RPC invoked by $2 \n Agent ID: $(cat hostID),
Destination: $destination \n $(cat traceResult)'"
```

Program 10. *TraceRoute shell script*

This program can receive all the *Trace* YANG data in XML format as input from the NETCONF agent and execute the traceroute command with specified options and returns the traceroute result inside traceResult container back to the network management system.

4.3 **Trace module invocation**

For invoking the *Trace* YANG module and corresponding shell script on the router, NETCONF API has been utilized. The below table describes a set of parameters to invoke the *Trace* module on the Ubuntu server:

Table 7. *Trace YANG modules parameters*

Destination IP	MTU	TTL	BGP AS probe	Source IP	API
8.8.8.8	80	90	Yes	*192.168.100.13*	NETCONF

Sending a hello message to the NETCONF manager, the *Trace* YANG module metadata retrieved from the NETCONF agent:

```
http://github.com/zgm/traceroute?module=trace&amp;revision=2019-07-
05
```

Base on the retrieved metadata and the parameter described in table 7, the YANG data has been prepared in XML format in the below program:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <traceRoute xmlns="http://github.com/zgm/traceroute">
          <destination>8.8.8.8</destination>
          <mtu>80</mtu>
          <ttlMax>90</ttlMax>
          <bgpAsPath>false</bgpAsPath>
          <sourceAddress>192.168.100.13</sourceAddress>
  </traceRoute>
</rpc>
```

Program 11. *YANG data for Trace module*

The prepared *Trace* YANG data including NETCONF hello and close message has sent from the NETCONF manager to the NETCONF agent via netconf-console as following:

```
netconf-console ~/trace.xml
```

During the observation on Wireshark, the NETCONF manager started a TCP session to the NETCONF agent and kept the session open until it received the response in traceResult container via <rpc-reply> from the NETCONF agent.

In addition to invoke the *Trace* remotely, *Trace* has been invoked locally on the device CLI by using the parameters described in table 7 as following:

```
confd_cli
traceRoute destination 8.8.8.8 mtu 80 ttlMax 80 bgpAsPath true sourceAddress
192.168.100.13
```

The observation shown the same result with different packet RTTs.

5. RESULT AND ANALYSIS

The evaluation results in chapter 3 has been summarized into three tables. Table 8 compares the NETCONF and RESTCONF APIs for creating a loopback interface.

For creating a loopback interface, and considering the number of operations, the number of transactions and the number of TCP connections both NETCONF and RESTCONF APIs has shown the same performance.

Table 8. *NETCONF and RESTCONF APIs comparison for Interface configuration*

	NETCONF	**RESTCONF**
OPERATION	edit-config	POST
NO. OPERATIONS	1	1
NO. TRANSACTIONS	1	1
NO. TCP CONNECTIONS	1	1

Table 9 compares the NETCONF and RESTCONF APIs for creating a loopback interface and an OSPF routing instance. And table 9 compares the NETCONF and RESTCONF APIs for creating a loopback interface and modifying the OSPF routing instance.

Table 9. *NETCONF and RESTCONF APIs comparison for interface and OSPF routing configuration*

	NETCONF	**RESTCONF**
OPERATION	edit-config	POST
NO. OPERATIONS	1	2
NO. TRANSACTIONS	1	2
NO. TCP CONNECTIONS	1	2

Based these observed results, both RESTCONF and NETCONF APIs utilize TCP as transport protocol. With RESTCONF APIs, one TCP session carries only one operation to perform the configuration changes in transactional approach on the device running config. However, with

NETCONF, one TCP session can carry multiple operations to perform a set of configuration changes in transactional approach directly on the device running config.

Table 10. *NETCONF and RESTCONF APIs comparison for interface configuration and OSPF routing modification*

	NETCONF	**RESTCONF**
OPERATION	edit-config	POST, PATCH
NO. OPERATIONS	1	2
NO. TRANSACTIONS	1	2
NO. TCP CONNECTIONS	1	2

For a set of configuration changes, NETCONF API has shown better performance than RESTCONF API. As NETCONF API were able to make the bulk configuration changes over one TCP session but RESTCONF API needed two TCP sessions to perform the same configuration changes. Using one transaction by NETCONF API makes NETCONF more suitable for service configuration, however using multiple transactions by RESTCONF makes this protocol suitable for device level configuration.

The downside of the NETCONF and RESTCONF API were using text-based XML and JSON data encoding formats which can make parsing process on the network devices slow.

On the other hand, chapter 4 results for developing and invoking the *Trace* tool has gathered in table 11, 12 and 13. Based on the observation results in table 11, NETCONF API for invoking the *Trace* tool, sent the traceroute operation inside one TCP session to the router.

Table 11 . *NETCONF APIs results for Trace YANG module*

	NETCOF
OPERATION	traceRoute
NO. OPERATIONS	1
NO. TCP CONNECTIONS	1

Table 12 describes the remote invocation of the *Trace* result including the hopes IP address, the BGP AS path, average RTT of the packets and the number of hops that packets traversed from the *192.168.100.13* to get to the 8.8.8.8

Table 12. *Remote Trace module invocation response on the NETCONF manager*

Hop IP address	BGP AS	Avg. RTT (ms)	No. hops
192.168.100.1	*	1.568	1
***	*	*	2
10.64.192.77	*	16.675	3
213.192.186.78	6667	14.675	4
213.192.186.77	6667	15.766	5
213.192.186.74	6667	21.439	6
213.192.185.93	6667	21.587	7
108.170.254.33	15169	37.485	8
74.125.252.63	15169	31.312	9
8.8.8.8	15169	35.998	10

Table 13 describes the results for the same *Trace* data inputs when the *Trace* invoked locally on the router. Based on this observation, hops IP addresses, BGP AS numbers and the number of hobs remained consistent with the results of remote invocation over NETCONF API and only average RTT of the packets slightly differed.

Table 13. *Local invocation of Trace module response on the NETCONF manager*

Hop IP address	BGP AS	Avg. RTT (ms)	No. hops
192.168.100.1	*	1.499	1
***	*	*	2
10.64.192.77	*	16.771	3
213.192.186.78	6667	14.600	4
213.192.186.77	6667	15.780	5
213.192.186.74	6667	21.450	6
213.192.185.93	6667	21.601	7
108.170.254.33	15169	37.479	8
74.125.252.63	15169	31.300	9
8.8.8.8	15169	35.988	10

Using the YANG language to develop the *Trace* module provided a unique syntax for traceroute CLI tools on the network management system and hided the routers syntax difference from the network management point of view. As *Trace* module invoked the shell script written in program 10 on the device CLI, using the same *Trace* module for invoking remote *traceroute* on different router CLIs only requires code changes on the network device.

To sum up, *Trace* tool can be used in a YANG and NETCONF API based network management solutions along with other capabilities such as device configuration, to provide a centralized operating system independent troubleshooting point in the TCP/IP networks.

Table 14, provides a brief summary of steps of writing and updating this thesis:

Table 14. *Summary of the writing this thesis process*

Objectives	Central opensource traceroute tool development, RETCONF and NETCONF API performance evaluation
Time frame	February – September 2019, working hours > 380 h
results	Central opensource *Trace* YANG module development and invocation using NETCONF
main changes	Clarified objectives and updated the sections to match the TAU Thesis Guide
main problems	Unclear objectives, and lacking result section

6. CONCLUSION

Network management solutions are evolving to provide a comprehensive set of capabilities to ease and optimize the network devices monitoring, configuration and troubleshooting. This thesis tried to compare two network management solutions SNMP and YANG from the configuration and troubleshooting perspective.

And showed that YANG modules unlike SNMP MIBs, can provide a unique syntax for making configuration changes and accessing state data notification from network system management on infrastructure devices. In addition to the configuration capabilities, it showed that YANG modules provide command invocation capability on the device CLI, which enables centralized accessing point to the device CLI from the network management system.

For making configuration changes on the network devices with RESTCONF APIs, one TCP session carried only one operation to perform configuration changes in transactional approach on the device running config. However, with NETCONF API, one TCP session could carry multiple operations to perform a set of configuration changes in transactional approach directly on the device running config. Thus, NETCONF API showed a better performance to be candidate for service level configuration.

The *Trace* YANG module invokes the traceroute CLI tool from a centralized network management system on TCP/IP routers, traces the hops and BGP AS, and measures RTT between a router and specific destination and returns the response back to the network management system. The implementation has showed that YANG based data models enables a unique syntax on the network management system for invoking traceroute command on different devices operating system. In addition, *Trace* YANG module provides a central point in the network to run and collect tracing route data from different routers to different destinations. Utilizing NETCONF API showed that the *Trace* only establishes one short time TCP session towards the router and upon receiving the response back from the router, it closes the TCP session.

Although NETCONF is an IETF standard protocol but only a few network vendors started supporting NETCONF on their products. And more community effort for sharing open network YANG modules is needed to make YANG modules independent of devices operating system.

The Trace YANG modules with NETCONF API is a promising centralized troubleshooting tool that can be used along with YANG and NETCONF API based network management solutions for TCP/IP open networks.

REFERENCES

Claise, B., Clarke, J. and Lindblad, J. (2019). *Network Programmability with YANG. pp. 32:79:162:189:241:246*.

Schudel, G. (2008). *Control Plane Policing Implementation Best Practices.* [online] Cisco. Available at: https://www.cisco.com/c/en/us/about/security-center/copp-best-practices.html [Accessed 12 Mar. 2019].

Juniper. (2019). *Junos OS Overview - TechLibrary - Juniper Networks.* [online] Available at: https://www.juniper.net/documentation/en_US/junos/topics/concept/junos-one-software-overview.html [Accessed 13 Mar. 2019].

Cisco. (2019). *Cisco Nexus 9000 Series NX-OS Programmability Guide, Release 7.x - Nexus Application Development - ISO [Cisco Nexus 9000 Series Switches].* [online] Available at: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/7-x/programmability/guide/b_Cisco_Nexus_9000_Series_NX-OS_Programmability_Guide_7x/b_Cisco_Nexus_9000_Series_NX-OS_Programmability_Guide_7x_chapter_010000.html [Accessed 13 Mar. 2019].

Molenaar, R. (2016). *Introduction to Cisco IOS XE | NetworkLessons.com.* [online] NetworkLessons.com. Available at: https://networklessons.com/cisco/ccie-routing-switching-written/introduction-cisco-ios-xe [Accessed 13 Mar. 2019].

Lu, Y. and Qian, S. (2010). *Research on the theory of SNMP and the technology of SNMP programming.* [online] IEEE, pp.23:24. Available at: https://ieeexplore-ieee-org.libproxy.tuni.fi/stamp/stamp.jsp?tp=&arnumber=5564113&tag=1 [Accessed 1 Mar. 2019].

Simple Network Management Protocol. (n.d.). [ebook] *Cisco, p.2.* Available at: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/snmp/configuration/xe-3se/3850/snmp-xe-3se-3850-book/nm-snmp-snmpv1.pdf [Accessed 1 Jun. 2019].

Davis, E. (2004). *SNMPv3 -- User Security Model.* [online] drdobbs. Available at: http://www.drdobbs.com/snmpv3-user-security-model/199100972 [Accessed 3 Mar. 2019].

Bjorklund, M. (2010). *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF).* [online] p.11:32:36:37:55:90. Available at: https://tools.ietf.org/html/rfc6020 [Accessed 14 Mar. 2019].

Wallin, S. and Wikström, C. (2011). *Automating Network and Service Configuration Using NETCONF and YANG. In: 25th international conference on Large Installation System Administration.* [online] CA, USA: USENIX Association Berkeley, pp.1:2:3. Available at: https://www.tail-f.com/wordpress/wp-content/uploads/2013/03/Tail-f-NETCONF-YANG-Service-Automation-LISA-Usenix-2011.pdf [Accessed 14 Mar. 2019].

Lear, E. and Crozier, K. (2006). *Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP).* [online] p.2. Available at: https://tools.ietf.org/html/rfc4744 [Accessed 13 Apr. 2019].

Badra, M. (2009). *NETCONF over Transport Layer Security (TLS).* [online] p.13. Available at: https://tools.ietf.org/html/rfc5539 [Accessed 13 Apr. 2019].

Moberg, C. (2015). *NETCONF by example. 1st ed*. [ebook] IETF, pp.19:26. Available at: https://trac.ietf.org/trac/edu/raw-attachment/wiki/IETF94/94-module-3-netconf.pdf [Accessed 12 Apr. 2019].

Gazarov, P. (2016). *What is an API? In English, please..* [online] freeCodeCamp.org. Available at: https://medium.freecodecamp.org/what-is-an-api-in-english-please-b880a3214a82 [Accessed 14 Mar. 2019].

Voit, E., Clemm, A. and Gonzalez Prieto, A. (2016). *Requirements for Subscription to YANG Datastores*. [online] pp.4. Available at: https://tools.ietf.org/html/rfc7923 [Accessed 2 May 2019].

Bierman, A., Bjorklund, M. and Watsen, K. (2017). *RESTCONF Protocol.* [online] IETF, pp.5:15:40:41:42 Available at: http://tools.ietf.org/html/8040 [Accessed 14 Mar. 2019].

Cisco. (2018). *Programmability Configuration Guide, Cisco IOS XE Everest 16.6.x - RESTCONF Protocol [Cisco IOS XE 16].* [online] Available at: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/166/b_166_programmability_cg/b_166_programmability_cg_chapter_01011.html [Accessed 6 Apr. 2019].

Radford, A., Maccioni, F., Zapodeanu, G., Koren, I., McLaughlin, J., Cohoe, J., Yang, J., Kotha, K., Michraf, N. and Grasby, R. (n.d.). *Cisco IOS XE Programmability Automating Device Lifecycle Management.* [ebook] Cisco, pp.51:70:74:76:77. Available at: https://www.cisco.com/c/dam/en/us/products/collateral/enterprise-networks/nb-06-ios-xe-prog-ebook-cte-en.pdf [Accessed 8 Mar. 2019].

Devnetsandbox (2019). *DevNet Sandbox*. [online] Available at: https://devnetsandbox.cisco.com/RM/Diagram/Index/27d9747a-db48-4565-8d44-df318fce37ad?diagramType=Topology [Accessed 14 Mar. 2019].

Bjorklund, M. (n.d.). *mbj4668/pyang*. [online] GitHub. Available at: https://github.com/mbj4668/pyang [Accessed 2 Mar. 2019].

GitHub. (2019). *YangModels/yang*. [online] Available at: https://github.com/YangModels/yang [Accessed 26 Apr. 2019].

OpenConfig (2019). *openconfig/public.* [online] GitHub. Available at: https://github.com/openconfig/public [Accessed 26 Apr. 2019].

Pepelnjak, I. (2019). *Streaming Telemetry Standards: So Many to Choose From.* [online] Blog.ipspace.net. Available at: https://blog.ipspace.net/2018/03/streaming-telemetry-standards-so-many.html?m=1 [Accessed 2 Jun. 2019].

Python.org. (2019). *Welcome to Python.org.* [online] Available at: https://www.python.org/ [Accessed 24 Mar. 2019].

Ubuntu. (2019). *The leading operating system for PCs, IoT devices, servers and the cloud | Ubuntu.* [online] Available at: https://ubuntu.com/ [Accessed 10 Feb. 2019].

Volf, M. (n.d.). *netconf-console.* [online] PyPI. Available at: https://pypi.org/project/netconf-console/ [Accessed 4 Jun. 2019].

Paramiko. (n.d.). *Welcome to Paramiko! — Paramiko documentation.* [online] Available at: http://www.paramiko.org/ [Accessed 4 Jun. 2019].

Postman. (2019). *Postman.* [online] Available at: https://www.getpostman.com/ [Accessed 6 Apr. 2019].

Jsonlint. (2019). *The JSON Validator.* [online] Available at: https://jsonlint.com/ [Accessed 7 Apr. 2019].

Wireshark.org. (n.d.). *Wireshark.* [online] Available at: https://www.wireshark.org/ [Accessed 19 Jun. 2019].

ConfD User Guide. (2018). *Tail-f Systems, pp.12:16.* [Accessed 1 Jun. 2019].

Tailf GitHub. (2019). *YangModels/yang.* [online] Available at: https://github.com/YangModels/yang/blob/master/vendor/cisco/xe/1671/tailf-common.yang [Accessed 1 Jun. 2019].

APPENDIX A: TRACE YANG MODULE SOURCE CODE

```
module trace {

  namespace "http://github.com/zgm/traceroute";
  prefix traceroute;

  import tailf-common {
    prefix tcommon;
  }

  contact
    "E-mail: zahra.golmohammadi@tuni.fi";

  description
    "YANG module for invoking Linux traceroute command";

  revision 2019-07-05 {
    description
      "Initial revision";
    reference "1.0.0";
  }

  typedef ttlRange {
    type uint32 {
      range "1 .. 255";
    }
}
  typedef mtuRange {
    type uint32 {
      range "28 .. 9000";
    }
}
  rpc traceRoute {
    description
          "Invoke the traceRoute.sh shell script";
    tcommon:exec "./traceRoute.sh" {
    tcommon:args "-c $(context)";
    tcommon:wd ".";
    }

  input {
     leaf destination {
       description
         "Specify the destination IP address or domain name";
       type string;
```

```
        mandatory true;
       }

      leaf mtu {
        description
          "Specify the packet size in byte";
type ttlRange;
        default 64;
       }

      leaf ttlMax {
        description
          "Specify the maximum time to live number";
        type mtuRange;
        default 30;
       }

      leaf bgpAsPath {
        description
          "Looks for BGP AS number";
        type boolean;
        default false;
       }

      leaf sourceAddress {
        description
          "Specify the source IP address for sending ICMP traceroute packets";
        type string;
       }
      }
    output {
      leaf traceResult {
      type string;
       }
     }
   }
}
```