Report

Nabil ECH-CHOKHMANY

July 19, 2023

Contents

1 Abstract **4**

2 Introduction **5**

3 Background **6**

4 Overview of Netconf **8**
4.1 Network Configuration Protocol (NETCONF) 8
4.2 YANG 10
4.3 XML 11
4.4 RPC 11

5 Implementation **13**
5.1 GNS-3 13
5.1.1 Appliances 13
5.2 Configure DHCP 16
5.2.1 Dynamic Host Configuration Protocol (DHCP) 16
5.2.2 DHCP configuration on Cisco router: 17
5.3 Enabling NETCONG-YANG 18
5.4 SSH connection establishment 18
5.4.1 Secure SHell (SSH) 18
5.4.2 Establishing an SSH connection between the router an host 18
5.4.3 Result 19

6 Configuring Networks with NETCONF **20**
6.1 NETCONF base operations 20
6.2 Python code structure 21
6.2.1 Libraries 21
6.2.2 Session establishment 21
6.2.3 Filter 22
6.2.4 Configuration modification/retrieval and commit 22
6.3 Python automation 24
6.3.1 Scenario 24
6.3.2 Solving the Subnet Allocation Challenge 24

7 Conclusion **28**

List of Figures

4.1 Diagram illustrating how Network Configuration Protocol (NETCONF) works. 9
4.2 YANG Structure. 10

5.1 Cisco CSR 1000v. 14
5.2 Switch 14
5.3 Virtual PC 15
5.4 Cloud 15
5.5 Network topology 16
5.6 IP assignment. 17

6.1 Visualizing the IP Pool as a Circular Space. 25

Listings

4.1 <rpc> Model. 11
4.2 <rpc> <get> Model. 12
4.3 <rpc-reply> <get> Model. 12
5.1 Excerpt from "hello" message. 19
6.1 Session establishment. 21
6.2 Filter for interfaces. 22
6.3 Changing the interface. 22
6.4 Router's capabilities. 23
6.5 Routers informations input. 26
6.6 Output. 26

Chapter 1

Abstract

This internship report explores the realm of network automation and configuration management using NETCONF and YANG in a virtual network environment with GNS3. The primary focus of the internship was to work with Cisco CSR1000v routers, enabling NETCONF and YANG support, and establishing secure connections for efficient configuration management.

The report details the process of configuring essential network services such as DHCP, NAT, and SSH, enabling seamless communication and secure access to the routers. By leveraging Python's ncclient library, the report showcases how to interact programmatically with the routers, effectively retrieving and modifying configurations using NETCONF.

A significant aspect of the report centers around subnet allocation, a critical component in managing a large institution's network. By assessing the IP address requirements of each router, appropriate subnets are allocated to ensure resource optimization and minimal IP address wastage.

Furthermore, the report provides a comprehensive demonstration of modifying and retrieving configurations with a focus on maintaining correctness and accuracy. Input validation mechanisms are discussed to handle unexpected inputs gracefully, enhancing the overall reliability of the subnet allocation code.

The internship experience has equipped the author with valuable skills and knowledge in network automation technologies, Python programming, and GNS3 simulations. The report concludes with insights into the application of these acquired skills in real-world network engineering scenarios.

Overall, this internship report serves as a valuable resource for understanding NETCONF and YANG-based network management and offers practical insights into subnet allocation techniques, enabling efficient and automated network configuration.

Chapter 2

Introduction

The heterogeneity of communication networks, along with the ever-changing requirements of services, presents a significant challenge in developing effective techniques for network management. In response to this challenge, software-defined networking (SDN) initiatives have emerged with the aim of providing common and vendor-agnostic control planes for network devices and traffic management. This internship project focuses on understanding the operation of NETCONF, one of the main protocols for device management, and leveraging the GNS-3 network simulation tool to deploy simple network topologies. Additionally, a software tool will be developed, preferably in Python, to remotely modify the operation of devices using NETCONF.

The management of modern communication networks is essential to ensure optimal performance, scalability, and flexibility. However, the diverse range of network technologies and equipment from different vendors complicates the management process. Traditional network management approaches often rely on proprietary interfaces and protocols, leading to fragmented control planes and limited interoperability.

SDN initiatives address these challenges by decoupling the control plane from the underlying network infrastructure. By providing a centralized and programmable control plane, SDN allows administrators to define network behavior and policies through software-based controllers. This approach promotes flexibility, agility, and scalability in managing heterogeneous network environments.

One of the key protocols used in SDN is NETCONF (Network Configuration Protocol). NETCONF, defined by the IETF (Internet Engineering Task Force), is a standardized protocol that enables secure and efficient configuration and management of network devices. It provides a programmatic interface for accessing and modifying device configurations, monitoring device state, and retrieving operational data.

During this internship at the University of Cantabria under the supervision of Luis, the objective was to gain hands-on experience in network management using SDN principles and NETCONF protocol. The internship involved leveraging the GNS-3 network simulation tool to deploy simple network topologies, simulating real-world network environments. Additionally, a software tool was developed using Python to remotely modify the operation of devices through NETCONF.

In the subsequent sections of this report, we will delve into the background of software-defined networking, discuss the objectives and scope of the project, outline the methodology employed, present the results and findings, and conclude with reflections and recommendations. Throughout the report, we will explore the practical implementation of NETCONF and its effectiveness in managing network devices in a simulated environment.

Chapter 3

Background

Since network management became an essence for computer networks, the development of related technology has always been coupled with protocol standardizations, which are mainly OSI-based CMIP and TCP/IP-based SNMP. On one hand, OSI-SM has been the most powerful technology but is complicated and expensive and relies on OSI protocols that have gone out of fashion. On the other hand, SNMP is the very solution that has been used by most of the industry, but fell victim to its own simplicity: its data modeling capabilities are rudimentary and it does not support configuration management well due to its lack of transaction capabilities.

With the development of computer networks in multiple dimensions (number of devices, time scale for configuration, etc), configuring large networks becomes an increasingly difficult work. A set of configuration management requirements for IP-based networks are then identified, focusing on network–wide configurations, which provide a level of abstraction above device-local configurations. In this case, the function of configuration data translator must be seriously considered. Other requirements consist of distinguishing between configuration state and operational state, providing primitives to support concurrency in transaction-oriented way, the persistence of configuration changes, security considerations, and so on.

XML-based configuration management is now under hot research, especially using NETCONF. Main characteristics of the NETCONF protocol are briefly introduced as follows, with a detailed specification in Reference.

NETCONF defines a simple mechanism, through which a network device can be managed, configuration data information can be retrieved, and new configuration data can be uploaded and manipulated. The paradigm that it uses is named Remote Procedure Call (RPC). A key aspect of NETCONF is that, it allows the functionality of the management protocol to closely mirror the native functionality of the device. And applications can access both the syntactic content and the semantic content of the device's native user interface. In addition, NETCONF allows a client to discover the set of protocol extensions supported by a server. These so-called "capabilities" permit the client to adjust its behavior to take advantage of the features exposed by the device.

NETCONF Data Modeling Language (netmod) Working Group (WG) proposed by the IETF aims in supporting the ongoing development of IETF and vendor-defined data models for NETCONF, since NETCONF needs a standard content layer and its specifications do not include a modeling language or accompanying rules that can be used to model the management information to be configured using NETCONF. The main purpose of the netmod WG is to provide a unified data modeling language to standardize the NETCONF content, by defining a "human-friendly" language and emphasizing readability and ease of use. The very language seems to be able to serve as the normative description of NETCONF data models. Thus from this point of view, this WG plans to use YANG as its starting point for this language. In the context of this internship project, the GNS3 network simulation tool is utilized to deploy simple network topologies. GNS3 allows for the creation of network environments

that closely resemble real-world networks. By utilizing GNS3, interns can gain practical experience in network deployment and configuration, providing a suitable environment for testing and evaluating the operation of NETCONF and its interaction with network devices.

By combining the practical use of GNS3 and the implementation of NETCONF, this internship project aims to explore the deployment and management of network devices using software-defined networking principles.

Chapter 4

Overview of Netconf

4.1 Network Configuration Protocol (NETCONF)

The Network Configuration Protocol (NETCONF) is an Internet Engineering Task Force (IETF) network management protocol that provides a secure mechanism for installing, manipulating and deleting the configuration data on a network device, such as a firewall, router or switch. NETCONF was developed by the NETCONF working group and published in December 2006 as RFC 4741. The protocol was then revised in June 2011 and published as RFC 6241. This is the most current version. The IETF also published several other RFCs related to NETCONF. For example, RFC 5277 defines a mechanism for supporting an asynchronous message notification service for NETCONF. The NETCONF protocol was designed to make up for the shortcomings of the Simple Network Management Protocol and the command-line interface scripting used to configure network devices.

NETCONF uses the Remote Procedure Call (RPC) protocol to carry out communications between clients and servers. RPC is a client/server protocol that lets a program request a service from another program without understanding the details of the underlying network. RPC messages are encoded in Extensible Markup Language (XML) and transmitted via secure connection-oriented sessions. A NETCONF client, which is often part of a network manager, can be a script or application. A server is usually a network device. RFC 6241 uses the terms client and application interchangeably and the terms server and device interchangeably. The client sends RFC messages that invoke operations on the server. The client can also subscribe to receive notifications from the server. The server executes the operations invoked by the client, and it can send notifications to the client. A NETCONF server contains one or more configuration datastores. A configuration datastore is a datastore that holds all the configuration data needed to take a device from its default state to a configured operational state. A NETCONF datastore is simply a place to store and access configuration information. For example, the datastore might be a database, a set of files, a location in flash memory or any combination of these.

The NETCONF protocol facilitates secure RPC communications between the client and server, providing a standards-based approach to network device management. The protocol can be conceptualized as having four layers:

- Secure Transport Layer. The first layer provides the core communication path between the client and server. NETCONF is not bound to any transport protocol, but it can be layered over any transport protocol, including Transport Layer Security and Secure Shell. However, the protocol must provide the necessary functionality. The transport layer makes it possible for the client and server to communicate through a series of RPC messages.

- Messages Layer. The second layer provides a transport-independent framing mechanism for

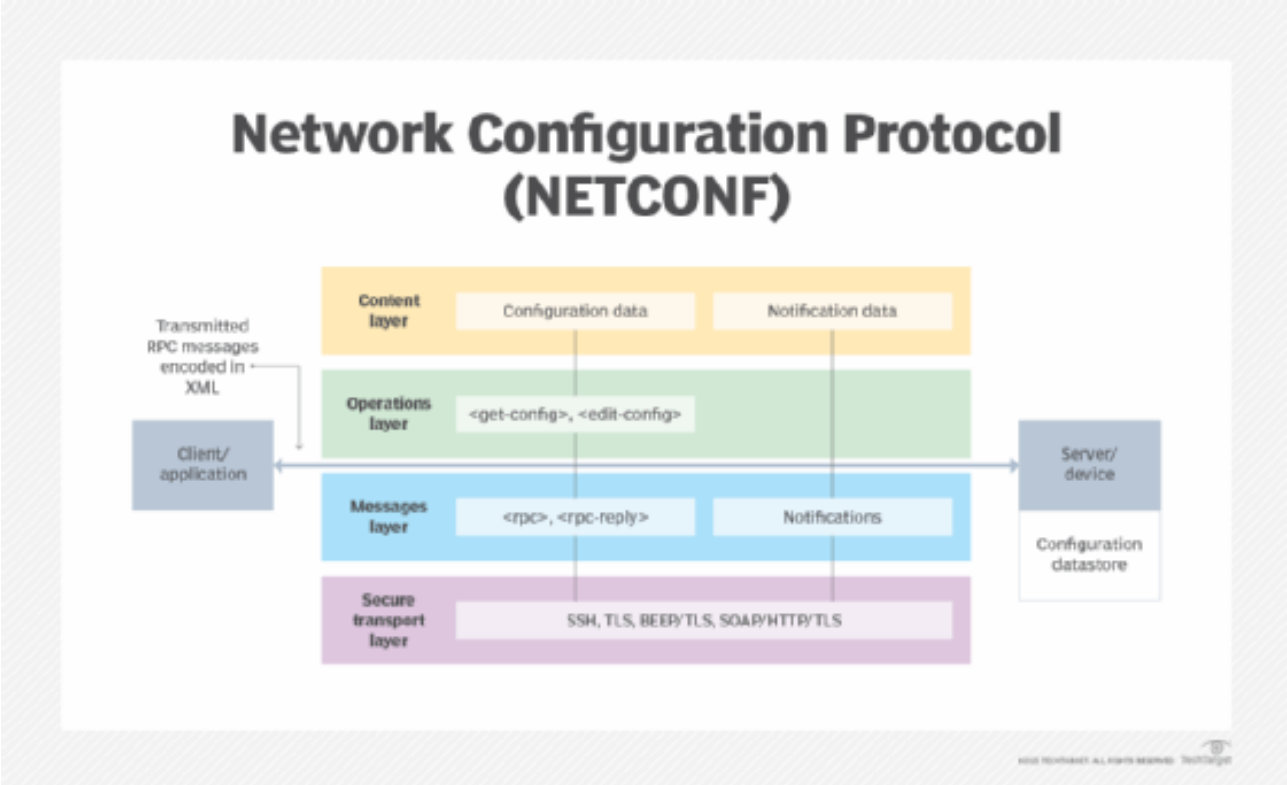

Figure 4.1: Diagram illustrating how Network Configuration Protocol (NETCONF) works.

encoding RPCs and notifications. NETCONF uses an RPC-based communication model to provide the framing necessary to support requests and responses between the client and server. In documenting the Messages Layer, RFC 6241 focuses primarily on RPC communications, rather than notifications, which are instead documented in RFC 5277.

- Operations Layer. The third layer defines a small set of low-level base operations for retrieving information and managing configurations. The set includes operations such as ¡get-config¿ or ¡edit-config¿. The operations are invoked as RPC methods with XML-encoded parameters, passed in as child elements of the RPC elements.

- Content Layer. The top layer is concerned with configuration and notification data; however, this layer lies outside the scope of RFC 6241. Instead it relies on the device's own data model. NETCONF carries the model's configuration information within the ¡config¿ element but treats it as opaque data. The YANG data modeling language (RFC 6020) was developed for specifying NETCONF data models and protocol operations.

When a client communicates with a server, it sends one or more RFC request messages to that server, which responds with its own RFC reply messages. The two most common XML elements used for RFC communications are ¡rpc¿ and ¡rpc-reply¿. The ¡RPC¿ element encloses a request sent from the client to the server. The request information within the element includes the RPC's name and its parameters. The ¡rpc-reply¿ element is used to respond to ¡rpc¿ messages. All response data is encoded within the ¡rpc-reply¿ element.

Within the communication flow of a NETCONF session there are 3 main parts. These are:

- Session Establishment – Each side sends a ¡hello¿, along with its ¡capabilities¿. Announcing what operations (capabilities) it supports.

- Operation Request – The client then sends its request (operation) to the server via the¡rpc¿ message. The response is then sent back to the client within ¡rpc-reply¿.
- Session Close – The session is then closed by the client via ¡close-session¿.

4.2 YANG

YANG (Yet Another Next Generation) is a data modelling language, providing a standardized way to model the operational and configuration data of a network device. YANG, being a language is being protocol independent, can then be converted into any encoding format, e.g. XML or JSON Open/Native Models You may be asking who creates these models? The models are classified as either Open or Native based, with different groups working across each one.

- Open Models – Designed to be independent of the underlying platform and normalize the per-vendor configuration of network devices. Open YANG Models are developed by Vendors and Standards bodies, such as IETF, ITU, OpenConfig etc.
- Native Models – Native Models are developed by the vendors. They relate and are designed to integrate to features or configuration only relevant to that platform.

Components

A YANG model is made up from various components. Let's look at these components, in relation to our example (seen within Figure 3.2).

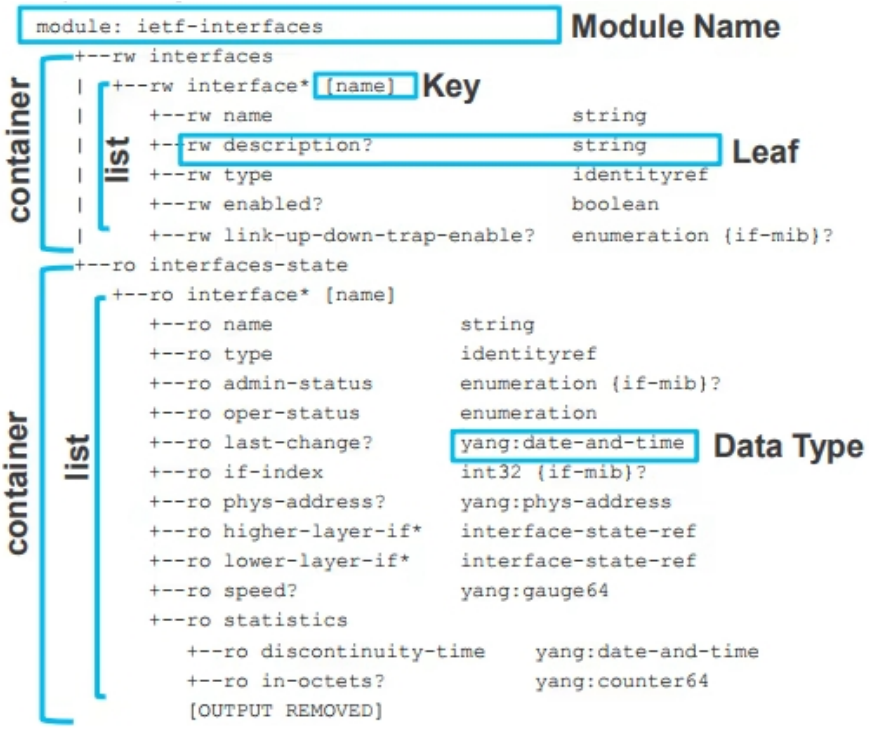

Figure 4.2: YANG Structure.

- Container – A collection of information logically grouped. Such a container for configuration, and one for state.

- List – Within a container you can have a list or even multiple lists. Such as a list of interfaces.
- Key – Each item within the list is references via a key.
- Leaf – Inside our list we have leaf's. Containing our information.
- Data Type – Each leaf is associated against a data type.

4.3 XML

Extensible Markup Language (XML) lets you define and store data in a shareable manner. XML supports information exchange between computer systems such as websites, databases, and third-party applications. Predefined rules make it easy to transmit data as XML files over any network because the recipient can use those rules to read the data accurately and efficiently.

In the context of NETCONF, XML is used as the format for exchanging configuration information between the client and the network device. NETCONF defines a set of standardized XML-based messages that are used to manage the configuration and operational state of network devices.

When a client sends a request to a network device using NETCONF, it typically includes an XML document that specifies the desired configuration changes or queries. The network device processes the XML request, performs the necessary operations, and sends a response back to the client in XML format.

The use of XML in NETCONF provides several benefits. First, XML is a widely adopted standard for data representation, making it compatible with a variety of systems and tools. Second, XML allows for structured and hierarchical representation of complex network configurations. Finally, XML's human-readable nature makes it easier for administrators and developers to understand and work with the configuration data.

4.4 RPC

Remote Procedure Call is a software communication protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details. RPC is used to call other processes on the remote systems like a local system. A procedure call is also sometimes known as a function call or a subroutine call.

RPC Model

The NETCONF protocol uses an RPC-based communication model. NETCONF peers use <rpc> and <rpc-reply> elements to provide transport-protocol-independent framing of NETCONF requests and responses.

<rpc> Element

The <rpc> element is used to enclose a NETCONF request sent from theclient to the server.

The <rpc> element has a mandatory attribute "message-id", which is a string chosen by the sender of the RPC that will commonly encode a monotonically increasing integer. The receiver of the RPC does not decode or interpret this string but simply saves it to be used as a "message-id" attribute in any resulting <rpc-reply> message.

Listing 4.1: <rpc> Model.

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
        <some-method>
            <!-- method parameters here... -->
        </some-method>
    </rpc>
```

The name and parameters of an RPC are encoded as the contents of the <rpc> element. The name of the RPC is an element directly inside the <rpc> element, and any parameters are encoded inside this element.

The following example invokes the NETCONF <get> method with no parameters:

Listing 4.2: <rpc> <get> Model.

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get/>
</rpc>
```

<rpc-reply> Element

The <rpc-reply> message is sent in response to an <rpc> message.

The <rpc-reply> element has a mandatory attribute "message-id", which is equal to the "message-id" attribute of the <rpc> for which this is a response.

A NETCONF server MUST also return any additional attributes included in the <rpc> element unmodified in the <rpc-reply> element.

The response data is encoded as one or more child elements to the <rpc-reply> element.

For example:

The following <rpc> element invokes the NETCONF <get> method and includes an additional attribute called "user-id". Note that the "user-id" attribute is not in the NETCONF namespace. The returned <rpc-reply> element returns the "user-id" attribute, as well as the requested content.

Listing 4.3: <rpc-reply> <get> Model.

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
        xmlns:ex="http://example.net/content/1.0"
        ex:user-id="fred">
    <get/>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:ex="http://example.net/content/1.0"
    ex:user-id="fred">
    <data>
        <!-- contents here... -->
    </data>
</rpc-reply>
```

Chapter 5

Implementation

5.1 GNS-3

GNS3 (Graphical Network Simulator-3) software allows us to emulate complex network designs such as Cisco Router/Switch.

It enables to run real Cisco IOS images using Dynamips in the GNS3 infrastructure, and with this software, we can create complex network designs on our computer and study in more detail for exams such as Cisco Certified Network Associate (CCNA) and Cisco Certified Network Professional (CCNP), but in the context of this internship, we will exploit the GNS-3 to deploy simple network topologies and develop a simple software (preferably in Python) to remotely modify the devices operation using NETCONF.

5.1.1 Appliances

In the context of GNS3, appliances refer to pre-configured virtual network devices that can be integrated into GNS3 for network emulation and simulation purposes. These appliances are typically virtual machines (VMs) running specialized network operating systems or software, designed to replicate the behavior and functionalities of real-world networking devices.

Appliances in GNS3 are available in various forms, such as virtual routers, switches, firewalls, load balancers, and other network devices. They allow us to create complex network topologies and simulate different network scenarios without the need for physical hardware. By utilizing these virtual appliances, we can gain practical experience in network design, testing, and troubleshooting. In the case of our implementation, we will need the following appliances:

Routers:

To work with NETCONF and YANG in GNS3, we will need a virtual router that supports these protocols. One highly recommended option is the Cisco CSR 1000v, renowned for its compatibility and seamless integration with NETCONF and YANG.

The Cisco CSR1000v is a virtual router that can be utilized within the GNS3 network simulation environment. It emulates the behavior and features of physical Cisco routers, specifically the Cisco IOS XE operating system. The CSR1000v is widely recognized for its versatility, offering extensive networking capabilities and supporting various protocols (mainly NETCONF-YANG). It enables users to create virtual network topologies and experiment with network configurations, making it a valuable tool for learning, testing, and developing networking solutions in a virtual environment using GNS3.

Figure 5.1: Cisco CSR 1000v.

Switches:

Switches are network devices that operate at the data link layer (Layer 2) of the OSI (Open Systems Interconnection) model. They are responsible for facilitating communication between devices within a local area network (LAN). We will benefit from the key functions and features of switches:

- Forwarding Frames: Switches receive network traffic in the form of frames and examine the destination MAC (Media Access Control) address of each frame.
- Broadcast and Multicast Handling: Switches efficiently handle broadcast and multicast traffic within a LAN
- MAC Address Learning: Switches maintain a MAC address table that maps MAC addresses to specific switch ports. When a frame arrives at a switch, it checks the source MAC address and associates it with the port on which the frame was received.

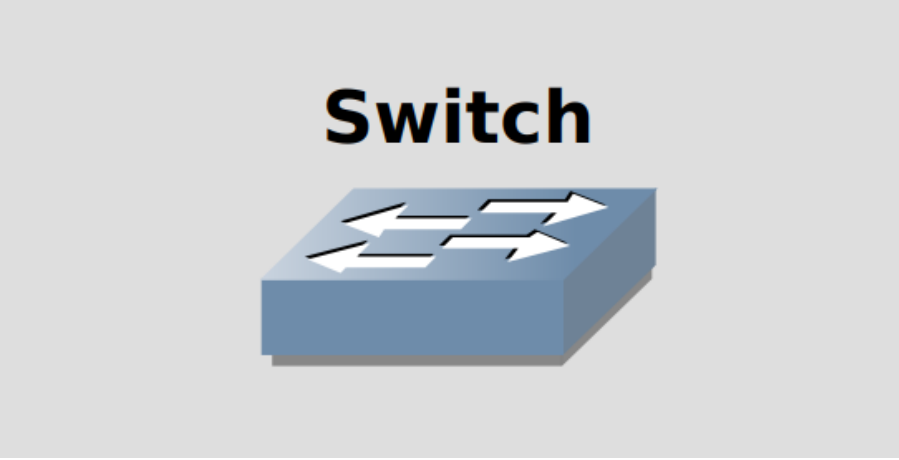

Figure 5.2: Switch

Virtual PC (host):

In GNS3, a virtual PC (VPC) is a simulated computer that represents an endpoint device within a network topology. It is a virtual machine (VM) running a specific operating system that emulates the behavior and functionalities of a physical computer. Virtual PCs are used in GNS3 to simulate the interaction between network devices and end-user devices, allowing us to test and troubleshoot network configurations.

Figure 5.3: Virtual PC

Cloud:

In GNS3, the "Cloud" appliance is a versatile tool that represents external networks or connectivity points in a network topology. It allows users to establish connections between the GNS3 network and real-world networks, such as the internet or local area networks (LANs). The cloud appliance acts as a bridge between the virtual network within GNS3 and the physical network environment.

In our case, we can utilize the cloud appliance in GNS3 to connect the PC to the network. This enables the PC to function as a NETCONF administrator and perform tasks related to network management using the NETCONF protocol. Additionally, the cloud appliance can also be configured to provide Network Address Translation (NAT) functionality, allowing our virtual network in GNS3 to access resources in the external network.

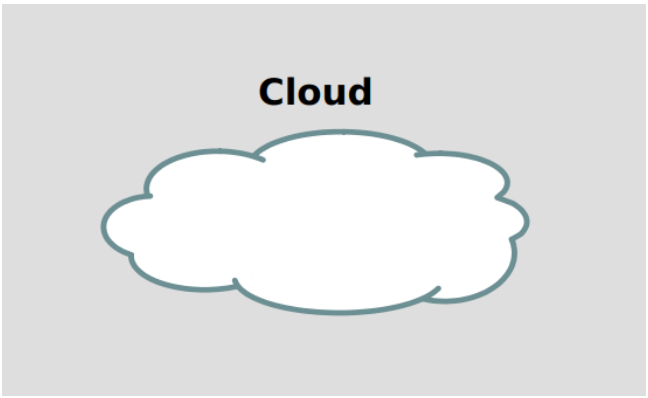

Figure 5.4: Cloud

Network topology:

Therefore, the resulting network topology will be as follows:

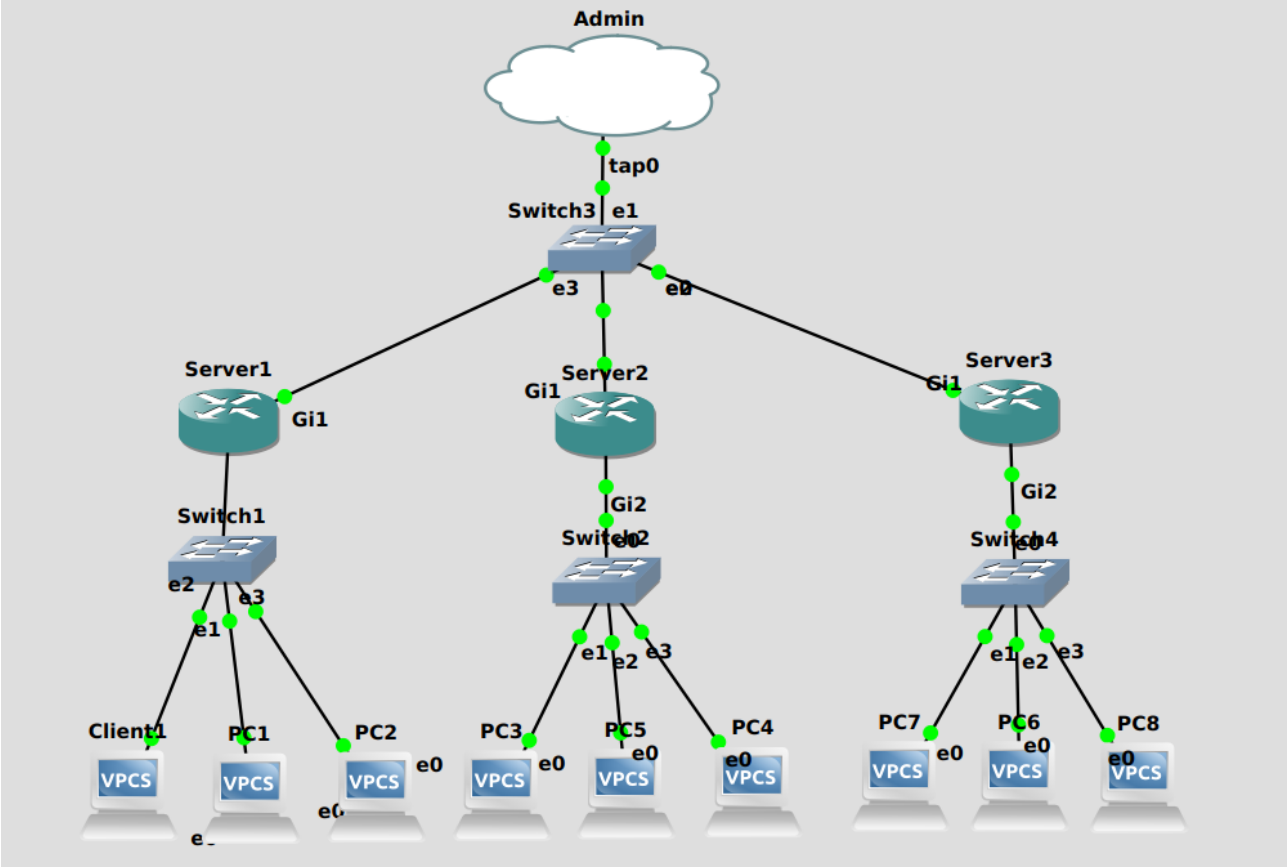

Figure 5.5: Network topology

5.2 Configure DHCP

5.2.1 Dynamic Host Configuration Protocol (DHCP)

Dynamic Host Configuration Protocol (DHCP) is a client/server protocol that automatically provides an Internet Protocol (IP) host with its IP address and other related configuration information such as the subnet mask and default gateway. RFCs 2131 and 2132 define DHCP as an Internet Engineering Task Force (IETF) standard based on Bootstrap Protocol (BOOTP), a protocol with which DHCP shares many implementation details. DHCP allows hosts to obtain required TCP/IP configuration information from a DHCP server (router in our case).

Every device on a TCP/IP-based network must have a unique unicast IP address to access the network and its resources. Without DHCP, IP addresses for new computers or computers that are moved from one subnet to another must be configured manually; IP addresses for computers that are removed from the network must be manually reclaimed.

With DHCP, this entire process is automated and managed centrally. The DHCP server maintains a pool of IP addresses and leases an address to any DHCP-enabled client when it starts up on the network. Because the IP addresses are dynamic (leased) rather than static (permanently assigned), addresses no longer in use are automatically returned to the pool for reallocation.

5.2.2 DHCP configuration on Cisco router:

Step 1:

Defining the pool. The name of the pool we will give is DHCP-POOL for the subnet 10.10.10.0/24

```
DHCP1(dhcp-config)#enable
DHCP1(dhcp-config)#config terminal
DHCP1(dhcp-config)#ip dhcp pool DHCP-POOL
```

Step 2:

Adding the subnet that we are going to use.

```
DHCP1(dhcp-config)#network 10.10.10.0/24
```

Step 3:

Configuring the default gateway for this subnet.

```
DHCP1(dhcp-config)#default-router 10.10.10.1
```

We can also exclude the first 10 IP for manual reservations from each subnet if we are planning to add any printers or servers in the location we could use the static IP configuration from those 10 IPs later, also set a DNS server and domain, but in our case we wont need these functionalities.

Result :

Consequently, when the end device (VPC) requests an IP address from the DHCP server (router), the client is assigned the initial IP address 10.10.10.2 :

```
VPC> ip dhcp
DORA IP 10.10.10.2/24 GW 10.10.10.1

VPC> sh ip

NAME        : VPC[1]
IP/MASK     : 10.10.10.2/24
GATEWAY     : 10.10.10.1
DNS         :
DHCP SERVER : 10.10.10.1
DHCP LEASE  : 86395, 86400/43200/75600
MAC         : 00:50:79:66:68:09
LPORT       : 10078
RHOST:PORT  : 127.0.0.1:10079
MTU         : 1500
```

Figure 5.6: IP assignment.

5.3 Enabling NETCONG-YANG

Enabling NETCONF-YANG functionality in a Cisco router is a simple process accomplished through a series of command lines entered directly into the router's console.

```
enable
configure terminal
netconf-yang
```

In the case of Cisco CSR 1000v, we will need to enable candidate-datastore feature which will be discussed later:

```
netconf-yang feature candidate-datastore
```

5.4 SSH connection establishment

5.4.1 Secure SHell (SSH)

SSH, also known as Secure Shell or Secure Socket Shell, is a network protocol that gives users, particularly system administrators, a secure way to access a computer over an unsecured network.

SSH also refers to the suite of utilities that implement the SSH protocol. Secure Shell provides strong password authentication and public key authentication, as well as encrypted data communications between two computers connecting over an open network, such as the internet.

In addition to providing strong encryption, SSH is widely used by network administrators to manage systems and applications remotely, enabling them to log in to another computer over a network, execute commands and move files from one computer to another.

In our scenario, we will establish an SSH connection between the local PC, acting as an administrator, and the three routers depicted in Figure 4.5.

5.4.2 Establishing an SSH connection between the router an host

On Cisco router

Step 1:

First, we will need to configure the hostname (Ex: Server1) and domain name

```
hostname Server1
ip domain-name Server1-domain
```

Step 2:

Generate rsa modulus and enter modulus lenght (2048 recommended), also configure the device to run SSHv2

```
crypto key generate rsa modulus 2048
ip ssh version
```

Step 3:

Configure the virtual terminal line settings

```
line vty 0 4
login local
transport input all
```

Step 4:

Create user and password for SSH client to access, and set privilege privilege level 15 to the username 'admin'

```
username admin password nabil
username admin privilege 15
```

Finaly, we don't forget to save the change by copying the ruuning configuration to the startup configuration

On the local machine

First, we need to create a new virtual interface (eth1) with an automatically assigned IP address. In our case, we are using an Ubuntu machine:

```
sudo ip link add name eth1 type dummy
sudo ip link set eth1 up
sudo dhclient eth1
```

Following the setup, we can proceed to establish an SSH connection from the local machine, acting as the management server, to the router. The router's IP address is "192.168.1.1", and the default port for NETCONF is "830".

```
ssh admin@192.168.1.1 -p 830 -s netconf
password: nabil
```

5.4.3 Result

As a result of establishing the SSH connection and initiating the NETCONF session, the first message exchanged between the NETCONF client (host) and the NETCONF server (router) is the "hello" message.

The "hello" message serves as a handshake to establish the capabilities and supported features between the client and server.

The "hello" message includes information about the NETCONF version supported by both the client and server, the list of supported capabilities (e.g., supported message types, datastores, and supported YANG models), and other essential details required to set up the NETCONF session.

Once the "hello" message exchange is successful, the NETCONF session is established, and the client can proceed to send specific NETCONF requests (e.g., get, edit-config, validate, commit, etc.) to interact with the device's configuration and operational data. Likewise, the server responds to these requests with corresponding messages, allowing for a standardized and programmable method of managing network devices.

Listing 5.1: Excerpt from "hello" message.

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
    ...........
    <capability>
        urn:ietf:params:netconf:capability:notification:1.1
    </capability>
</capabilities>
<session-id>24</session-id></hello>]]>]]>
```

Chapter 6

Configuring Networks with NETCONF

In this chapter, we explore the practical implementation of NETCONF (Network Configuration Protocol) for configuring networks in a Cisco router. Building upon the foundational understanding of DHCP, NAT, SSH, and the fundamentals of NETCONF established earlier, we delve into the specifics of leveraging NETCONF to streamline network configuration processes.

Before delving further, it is crucial to first explore the fundamental base operations of NETCONF. Understanding these operations provides a solid foundation for effectively leveraging NETCONF in network configuration management.

6.1 NETCONF base operations

The NETCONF protocol supports a set of low-level operations for retrieving and managing device configuration information. The operations are specified through XML elements, which are described in the following table. NETCONF also supports additional operations based on each device's capabilities:

- <get> : Retrieves all or part of the information about the running configuration and device state.
- <get-config>: Retrieves all or part of the configuration information available from a specified configuration datastore.
- <edit-config>: Submits all or part of a configuration to a target configuration datastore.
- <copy-config>: Creates or replaces a target configuration datastore with the information from another configuration datastore.
- <delete-config>: Deletes a target configuration datastore, but only if it's not running.
- <lock>: Locks a target configuration datastore, unless a lock already exists on any part of that datastore.
- <unlock>: Releases a lock on a configuration datastore that was previously locked through a <lock>operation.
- <close-session>: Requests the NETCONF server to gracefully terminate an open session.
- <kill-session>: Forces a session's termination, causing current operations to be aborted.

6.2 Python code structure

There are multiple ways to configure networks using NETCONF. Some common approaches include using command-line interfaces (CLI) on network devices, network management tools with built-in NETCONF support, or developing custom automation scripts. Python is a preferred choice for configuring networks with NETCONF due to its ease of use, rich ecosystem, comprehensive documentation, and most importantly, its integration capabilities.

6.2.1 Libraries

The ncclient library is a popular Python library used for interacting with NETCONF-enabled devices. It provides a high-level API (Application Programming Interface) that simplifies the process of establishing NETCONF sessions, retrieving device information, and configuring network devices. Its key benefits include:

1. Simplified NETCONF Communication:
   - Abstracts low-level details of the NETCONF protocol, providing an intuitive interface for developers.
   - Allows developers to focus on automation logic without the need to delve into protocol intricacies.
2. Easy Connection Establishment:
   - Handles authentication mechanisms (e.g., SSH) for establishing secure NETCONF sessions.
   - Provides a seamless channel for communication with NETCONF-enabled devices.
3. Configuration Management:
   - Facilitates retrieval and manipulation of device configuration using NETCONF.
   - Simplifies the retrieval of configuration data, application of configuration changes, and validation of resulting configurations.
4. Error Handling and Validation:
   - Includes built-in mechanisms for error handling and structured error responses.
   - Enables developers to validate configuration changes and effectively handle exceptions.

6.2.2 Session establishment

In this section, we discuss the process of establishing a NETCONF session using the manager.connect() method as an example. This method is part of the ncclient library and allows us to establish a secure connection with the router.

Listing 6.1: Session establishment.

```
with manager.connect(
      host="192.168.1.1",
      port=830,
      username="admin",
      password="nabil",
      hostkey_verify=False
      ) as m:
```

1. `manager.connect()`: The `manager.connect()` method initiates the process of establishing a NETCONF session.
2. `host`: This parameter specifies the IP address of the router (in this case, "192.168.1.1").
3. `port`: The `port` parameter defines the port number used for the NETCONF communication, which is typically set to 830.
4. `username` and `password`: These parameters provide the necessary authentication credentials to access the router.
5. `hostkey_verify`: By setting this parameter to `False`, host key verification for the SSH connection is disabled. This option can be useful during testing or when connecting to devices with self-signed or invalid SSH certificates. However, exercise caution and ensure the security implications are understood before using this option.
6. `with` statement: The `with` statement ensures that the NETCONF session is properly established and automatically closed when the block of code inside the `with` statement finishes execution.

6.2.3 Filter

The filter is a mechanism used to retrieve specific subsets of configuration or operational data from a network device. It allows us to define criteria for selecting the desired data elements using XPath expressions. By utilizing the filter parameter in NETCONF operations, such as ¡get-config¿ or ¡get¿, we can retrieve only the relevant information needed for our application or task. This targeted data retrieval capability enhances network management and automation by minimizing unnecessary data transfer, optimizing response times, and reducing network bandwidth usage.

For example, for the `<get-config>` operation in NETCONF with Python, we can use the *filter* parameter to retrieve only the interface configurations from a network device. By specifying a filter criterion using XPath expressions that target the interface elements, such as `'/interfaces/interface'`, we can narrow down the retrieved data to only the relevant interface information.

Listing 6.2: Filter for interfaces.

```
netconf_filter = """
    <filter>
      <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface></interface>
      </interfaces>
    </filter>"""
```

6.2.4 Configuration modification/retrieval and commit

This section showcases the process of connecting to the router, making modifications to the configuration, retrieving the configuration, and committing the changes.

The following example focuses on modifying a router's configuration, specifically changing its interface configuration, and then committing the changes:

Listing 6.3: Changing the interface.

```
            with manager.connect(**device, hostkey_verify=False) as m:
                  # Lock the candidate datastore
                  m.lock('candidate')

                  # Edit the configuration in the candidate datastore
```

```
                m.edit_config(target='candidate', config=filter)

                # Validate the candidate configuration
                m.validate()

                # Commit the candidate configuration to apply the changes
                m.commit()

                # Unlock the candidate datastore
                m.unlock('candidate')
            print("Interface changed successfully.")
```

1. Establishing a connection using the `manager.connect()` method.
2. Locking the candidate configuration datastore to prevent concurrent modifications.
3. Editing the configuration in the candidate datastore using the `edit_config()` method.
4. Validating the candidate configuration using the `validate()` method to ensure correctness.
5. Committing the modified candidate configuration using the `commit()` method to apply the changes.
6. Unlocking the candidate configuration datastore.

Limitation of Using Running Configuration Directly in Cisco CSR1000v

When working with network devices, it is common to have the ability to directly modify the running configuration. However, it's important to note that not all devices support this feature. One such example is the Cisco CSR1000v router.

The Cisco CSR1000v router, does not allow direct modifications to the running configuration. This limitation is revealed by examining the capabilities of the router through the use of NETCONF. The capabilities provided by the router specify the supported operations and features. In the case of the Cisco CSR1000v, the capabilities indicate that editing the running configuration directly is not available.

Listing 6.4: Router's capabilities.

```
            urn:ietf:params:netconf:base:1.0
            urn:ietf:params:netconf:base:1.1
            urn:ietf:params:netconf:capability:candidate:1.0
            urn:ietf:params:netconf:capability:xpath:1.0
            urn:ietf:params:netconf:capability:validate:1.0
```

This limitation in modifying the running configuration directly with the Cisco CSR1000v router reinforces the importance of using a candidate configuration when making changes (Chapter 4 provides the process of enabling the candidate datastore). By employing a candidate configuration, network administrators can safely apply and validate changes before committing them to the running configuration. This approach ensures greater control and reduces the risk of disrupting the network due to erroneous or incomplete configurations.

Understanding the limitations of the Cisco CSR1000v router in supporting direct modifications to the running configuration enables network administrators to adopt appropriate configuration management practices. By utilizing candidate configurations and committing changes only after thorough testing and validation, network stability and reliability can be maintained effectively.

6.3 Python automation

6.3.1 Scenario

In the context of a large institution such as a university, the establishment and management of a network infrastructure are critical for seamless communication and efficient resource allocation. One common scenario is the implementation of a centralized IP address pool shared by multiple routers within the network. This approach allows for optimal utilization of available IP addresses while accommodating the diverse needs of various departments, faculties, or network segments.

To ensure effective allocation of subnets within this shared IP pool, a systematic approach is required. By analyzing the specific requirements of each router, such as the number of IP addresses needed for its operations, it becomes possible to devise a subnet allocation strategy that best caters to the network's demands. This approach allows for efficient utilization of IP resources, prevents IP address conflicts, and simplifies network management.

In this section, we will focus on the automation and leveraging tools like the Python script provided in this report, the subnet allocation process can be streamlined. We will creat a script that enables network administrators to input the desired number of IP addresses for each router, and based on this information, it calculates the appropriate subnets and assigns them accordingly. The generated subnets can be further configured with relevant network parameters such as subnet masks, DHCP server networks, and default routers.

The proposed solution offers a practical and efficient method for managing a large-scale network infrastructure within an institution. It provides the flexibility to allocate subnets based on individual router requirements, ensuring optimized utilization of IP addresses while maintaining a coherent and manageable network architecture. This approach promotes scalability, adaptability, and ease of network administration, making it well-suited for complex environments like universities and other large institutions.

6.3.2 Solving the Subnet Allocation Challenge

1. Router IP Address Requirement Assessment: The first step is to assess the IP address requirements for each router. By understanding the number of IP addresses needed for their operations, we can allocate appropriate subnets. This information can be gathered through consultation with network administrators, department representatives, or through network traffic analysis.

2. Developing an Automated Subnet Allocation Script:

a. Reordering the List of IP Requirements

To streamline the subnet allocation process, it is beneficial to reorder the list of IP requirements in descending order. This reordering ensures that routers with larger IP address needs are given priority during the allocation process. By prioritizing routers with greater IP requirements, we can allocate larger subnets to accommodate their needs while maximizing the utilization of available IP addresses.

b. Visualizing the IP Pool as a Circular Space

To conceptualize the allocation process, we can envision the IP pool as a circular space. Imagining a circle representing the entire pool of available IP addresses. Each router's IP requirement corresponds to a slice or segment within this circular space. The size of each slice is directly proportional to the number of IP addresses required by the router. By visualizing the IP pool in this manner, we can better understand the allocation process and optimize the subnet assignments.

c. Allocating Subnets based on Slice Proportions

With the IP requirements reordered and the IP pool visualized as a circular space, the next step is to allocate subnets based on the proportions determined by each router's IP requirement. Starting from the router with the highest IP requirement, we assign a subnet slice to that router that corresponds to its proportional share of the circular space. We then move on to the router with the second-highest

IP requirement and allocate the next slice accordingly. This process continues until all routers have been assigned subnets.

d. Ensuring Efficient Resource Utilization

By following this systematic approach, we ensure that routers with larger IP requirements are allocated larger subnets, while routers with smaller IP requirements receive proportionally smaller subnets. This allocation strategy maximizes the utilization of available IP addresses, reduces IP address wastage, and avoids unnecessary subnet fragmentation.

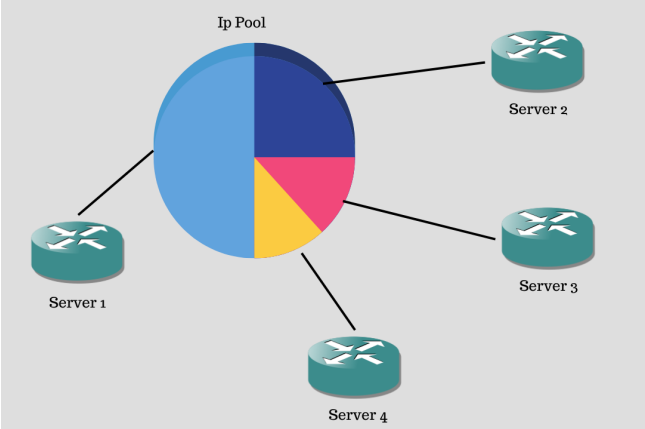

Figure 6.1: Visualizing the IP Pool as a Circular Space.

3. IP Pool Management: It is essential to maintain an accurate inventory of the IP pool and track the allocation of subnets to ensure efficient resource utilization. Network administrators can use IP address management (IPAM) tools or custom scripts to monitor and manage the IP pool effectively.

4. Addressing IP Conflicts: With multiple routers sharing a single IP pool, the risk of IP conflicts increases. Implementing mechanisms like DHCP servers and DHCP reservations can help mitigate IP conflicts and ensure proper IP address assignment within each subnet.

5. Configuration and Documentation: After the subnet allocation process, configuring each router with the assigned subnets, subnet masks, default routers, and other relevant network parameters is essential. Additionally, maintaining proper documentation, including subnet allocation records and network diagrams, aids in network troubleshooting and future expansion.

Example: Router IP Address Requirement Assessment and Subnet Allocation

To illustrate the process of router IP address requirement assessment and subnet allocation, let's consider a scenario with four routers in a large institution network. Each router has a specific number of IP addresses required for its operations, and we have a general subnet available for allocation.

Router Information:

- Router A: Requires 80 IP addresses.
- Router B: Requires 40 IP addresses.
- Router C: Requires 50 IP addresses.
- Router D: Requires 150 IP addresses.

General Subnet:

The available general subnet is "192.168.0.0/23," providing a total of 512 IP addresses.

Router IP Address Requirement Assessment:

To begin, we reorder the list of IP requirements in descending order:

- Router D: 150 IP addresses
- Router A: 80 IP addresses
- Router B: 50 IP addresses
- Router C: 40 IP addresses

Visualizing the IP Pool:

We conceptualize the IP pool as a circular space, representing the available "192.168.0.0/23" subnet. Each router's IP requirement corresponds to a slice or segment within this circular space. The size of each slice is directly proportional to the number of IP addresses required by the router.

Allocating Subnets based on Slice Proportions:

Starting with Router D, which has the highest IP requirement, we allocate a subnet slice proportionate to its need for 150 IP addresses. Router A receives a subnet mask: 24, and a DHCP Server Network: 192.168.0.0 (192.168.0.0 - 192.168.0.255)

Next, we move on to Router A, allocating a Subnet Mask: 25, and a DHCP Server Network: 192.168.0.0 (192.192.0.0 - 192.168.0.127)

For Router B, a subnet with the range (192.168.0.0 - 192.168.0.63) with a subnet mask of 26, satisfying its need for 50 IP addresses.

Finally, Router C receives a subnet with the range (192.168.1.64 - 192.168.1.127) with a subnet mask of 26 to accommodate its requirement of 40 IP addresses.

Testing Subnet Allocation Code

Input:

Listing 6.5: Routers informations input.

```
Enter the network and subnet in CIDR notation
(e.g., 10.10.0.0/23): 192.168.0.0/23
Enter the number of routers: 4
Enter the num_ips_router for Device1: 80
Enter the num_ips_router for Device2: 50
Enter the num_ips_router for Device3: 40
Enter the num_ips_router for Device4: 150
```

Output:

Listing 6.6: Output.

```
Device4
  Interface IP: 192.168.0.1
  Subnet Mask: 255.255.255.0 ( 24 )
  DHCP Server Network: 192.168.0.0
  DHCP Server Default Router: 192.168.0.1
Device1
  Interface IP: 192.168.0.1
```

```
    Subnet Mask: 255.255.255.128 ( 25 )
    DHCP Server Network: 192.168.0.0
    DHCP Server Default Router: 192.168.0.1

  Device2
    Interface IP: 192.168.0.1
    Subnet Mask: 255.255.255.192 ( 26 )
    DHCP Server Network: 192.168.0.0
    DHCP Server Default Router: 192.168.0.1

  Device3
    Interface IP: 192.168.0.65
    Subnet Mask: 255.255.255.192 ( 26 )
    DHCP Server Network: 192.168.0.64
    DHCP Server Default Router: 192.168.0.65
```

Chapter 7

Conclusion

During the course of this internship, I had the valuable opportunity to delve into the world of network automation and configuration management using NETCONF and YANG in a virtual network environment with GNS3. Working with Cisco CSR1000v routers and Python's ncclient library, I explored the possibilities of automating network configurations and harnessing the power of YANG data modeling.

Throughout the internship, I successfully enabled NETCONF and YANG support on the Cisco CSR1000v routers, establishing secure connections to manage configurations. I gained insights into essential network services like DHCP, NAT, and SSH, enabling efficient communication and secure access to the routers.

The experience of configuring networks with NETCONF using Python code was particularly enlightening. Leveraging the ncclient library, I interacted with the routers programmatically, retrieving and modifying configurations with ease. The understanding of base NETCONF operations empowered me to tailor configurations to meet specific requirements effectively.

Furthermore, I tackled the challenge of subnet allocation, a critical aspect in managing a large institution's network. By assessing the IP address requirements of each router and allocating appropriate subnets, I ensured efficient resource utilization and minimized IP address wastage.

In conclusion, this internship has been an enriching journey, providing hands-on exposure to cutting-edge network automation technologies. The knowledge gained in working with NETCONF and YANG, combined with practical skills in Python programming and GNS3 simulations, has equipped me with valuable tools for future endeavors in network engineering and automation.

I am grateful for the guidance and support from my mentors and colleagues, which played a significant role in the successful execution of this internship project. The experience gained here has instilled in me a deeper passion for network automation, and I look forward to applying these skills to contribute to the advancement of network management in real-world scenarios.