



UNIVERSIDAD
DE CANTABRIA



INTERNSHIP REPORT

Nabil ECH-CHOKHMANY

Implementing Network Topologies and Creating Software for Remote Device Operation Control via NETCONF

Ensil-Ensci supervisors: **Marie-Sandrine Denis**

Tutor: **Luis Francisco Diez**



Contents

1	Abstract	5
2	Introduction	6
3	Background	7
4	Overview of Netconf	9
4.1	Network Configuration Protocol - NETCONF	9
4.2	Yet Another Next Generation - YANG	11
4.3	Extensible Markup Language - XML	12
4.4	Remote Procedure Call - RPC	12
5	Implementation	14
5.1	GNS3	14
5.1.1	Appliances	14
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 NETCONF-YANG	18
5.4	SSH connection establishment	19
5.4.1	Secure SHell (SSH)	19
5.4.2	Establishing an SSH connection between the router an host	19
5.4.3	Result	20
6	Configuring Networks with NETCONF	21
6.1	NETCONF base operations	21
6.2	Python code structure	22
6.2.1	Libraries	22
6.2.2	Session establishment	22
6.2.3	Filter	23
6.2.4	Configuration modification/retrieval and commit	23
6.3	Python automation	25
6.3.1	Scenario	25
6.3.2	Solving the Subnet Allocation Challenge	25
6.3.3	Example: Router IP Address Requirement Assessment and Subnet Allocation	26
7	Annex	30
7.1	Annex A: Project Code	30
7.2	Annex B: Router configuration	32

List of Figures

4.1	Diagram illustrating how NETCONF works.	10
4.2	Example of a YANG data model structure.	11
5.1	Cisco CSR 1000v.	15
5.2	Switch	15
5.3	Virtual PC	16
5.4	Cloud	16
5.5	Network topology	17
5.6	IP assignment.	18
6.1	Visualizing the IP Pool as a Circular Space.	26

Listings

4.1	<rpc> Model.	13
4.2	<rpc> <get> Model.	13
4.3	<rpc-reply> <get> Model.	13
5.1	Excerpt from "hello" message.	20
6.1	Session establishment.	22
6.2	Filter for interfaces.	23
6.3	Changing the interface.	23
6.4	Router CSR1000v capabilities.	24
6.5	Routers informations input.	28
6.6	Output.	28
7.1	Automated Subnet Allocation.	30
7.2	Get-config.	31
7.3	Get-capabilities.	32
7.4	Configure DHCP.	32
7.5	Configure NAT.	32
7.6	Configure NETCONF.	33
7.7	Configure SSH.	33

Acronyms

CCNA	Cisco Certified Network Associate.	14
CCNP	Cisco Certified Network Professional.	14
CMIP	Common Management Information Protocol.	7
DHCP	Dynamic Host Configuration Protocol.	5
GNS3	Graphical Network Simulator 3.	5, 6, 8
IETF	Internet Engineering Task Force.	6, 7
IP	Internet Protocol.	5, 16
LAN	Local Area Network.	15
MAC	Media Access Control.	15
NAT	Network Address Translation.	5
NETCONF	Network Configuration.	5–9
netmod	NETCONF Data Modeling Language Working Group.	7
OSI-SM	OSI System Management.	7
RPC	Remote Procedure Call.	9
SDN	Software Defined Networking.	6
SNMP	Simple Network Management Protocol.	7
SSH	Secure Shell.	5
VMs	Virtual Machines.	14
VPC	Virtual Port Channel.	15
XML	eXtensible Markup Language.	9
YANG	Yet Another Next Generation.	5, 7, 9, 11

Chapter 1

Abstract

This internship report explores the realm of network automation and configuration management using Network Configuration (NETCONF) and Yet Another Next Generation (YANG) in a virtual network environment with Graphical Network Simulator 3 (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 Dynamic Host Configuration Protocol (DHCP), Network Address Translation (NAT), and Secure Shell (SSH), enabling seamless communication and secure access to routers. By leveraging the *ncclient* Python 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 sub-network configuration, a critical aspect in managing a large institution's network. By assessing the Internet Protocol (IP) address requirements of each sub-network, appropriate addresses ranges 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 sub-network allocation code.

The internship experience has equipped the author with valuable skills and knowledge in network automation technologies, Python programming, and the GNS3 emulation framework. 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 sub-netting 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 the NETCONF [4], one of the main protocols for device management, and leveraging the GNS3 network simulation tool to deploy simple but realistic 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 hinders 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. NETCONF, defined by the Internet Engineering Task Force (IETF), 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 the NETCONF protocol. The internship involved leveraging the GNS3 emulation 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 standardization efforts, which are mainly driven by the OSI-based Common Management Information Protocol (CMIP) and TCP/IP-based Simple Network Management Protocol (SNMP). On one hand, OSI System Management (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 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 task. 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 configuration. In this case, the function of configuration data translator must be seriously considered. Other requirements consist of distinguishing between configuration and operational state, providing primitives to support concurrency in a 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 [4].

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. Besides, 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 (NETCONF Data Modeling Language Working Group (netmod)) Working Group (WG) proposed by the IETF aims at 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 defined language is 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 [6].

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

Along this chapter we will describe the operation of NETCONF, indicating the role played by the solutions mentioned above: YANG, eXtensible Markup Language (XML), and Remote Procedure Call (RPC).

4.1 Network Configuration Protocol - NETCONF

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 [3]. The protocol was then revised in June 2011 and published as RFC 6241 [4], which is the most current version. The IETF also published several other RFCs related to NETCONF. For example, RFC 5277 [5] 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 one 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, whose management IP address is known by the network manager. RFC 6241 uses the terms client and application, and the terms server and device interchangeably.

The client (application) sends RFC messages that invoke operations on the server (device). 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 back to the client. A NETCONF server contains one or more configuration datastores for each device. A configuration datastore 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 [1]. Figure 5.5 provides an overall picture of the NETCONF operation, which can be conceptualized as having four layers as follows:

- 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 executed over any

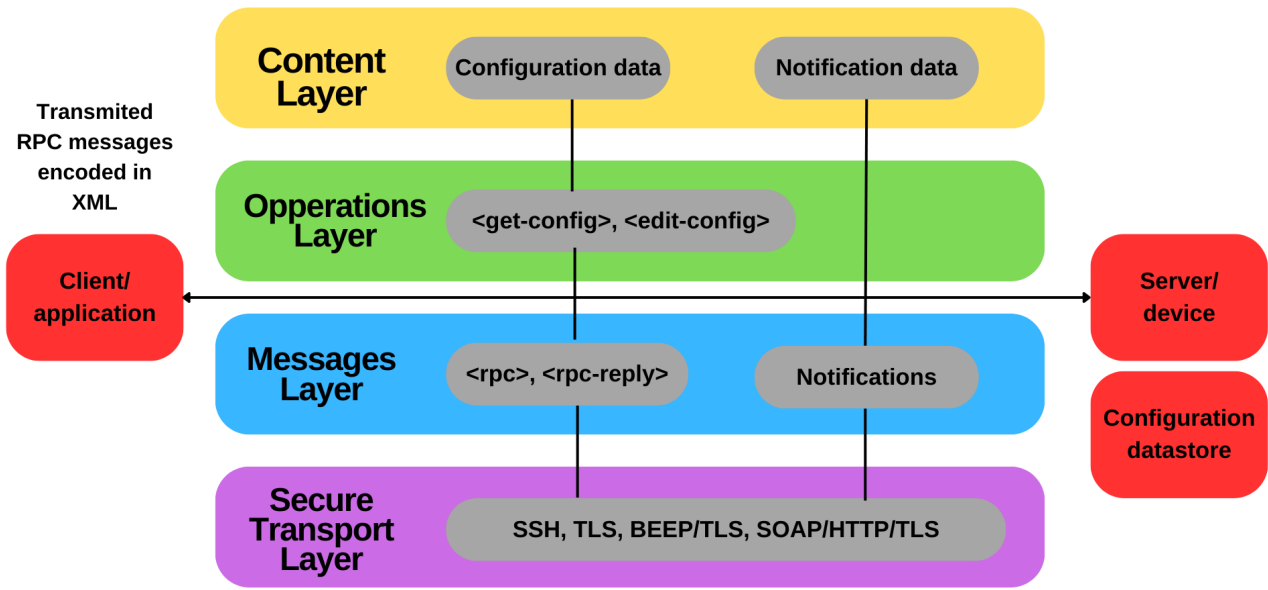


Figure 4.1: Diagram illustrating how NETCONF works.

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 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 [4] focuses primarily on RPC communications, rather than notifications, which are instead documented in RFC 5277 [5].
- **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 request messages to that server, which responds with its own reply messages. The two most common XML elements used for RPC 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. Then, the `<rpc-reply>` element is used to respond to the previous messages, and it is sent by the server. 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>` element, along with another one (`<capabilities>`) announcing what operations (capabilities) it supports.
- Operation Request: The client then sends its request (operation) to the server via an `<rpc>` message. The response is then sent back to the client within an `<rpc-reply>`.
- Session Close: The session is then closed by the client via `<close-session>`.

4.2 Yet Another Next Generation - YANG

YANG is a data modeling language, providing a standardized way to model the operational and configuration data of a network device. YANG, being a language, is protocol independent, and then can be embedded into any encoding format, e.g. XML or JSON [2].

The data models defined in YANG to be used by NETCONF are classified as either Open or Native, with different groups working across each one:

- Open Models: These models are designed to be independent of the underlying platform, and so they are vendor agnostic. In turn, different vendors implement the normalization tasks within their devices to process open data models. Open YANG Models are developed by Vendors and Standardization bodies, such as IETF, ITU, OpenConfig etc.
- Native Models: Native Models are vendor specific. They relate and are designed to integrate to features or configuration only relevant to specific vendors or devices.

A YANG model is made up from various components, as depicted in Figure 4.2:

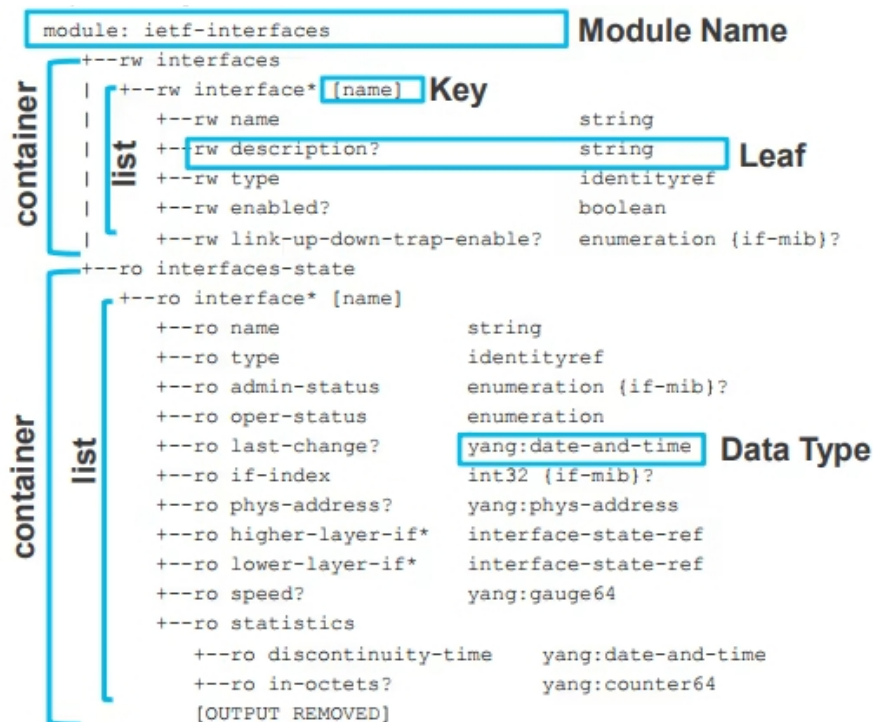


Figure 4.2: Example of a YANG data model structure.

- Container: A collection of information logically grouped. For instance there can be a containers defined for configuration, and for state.
- List: Within a container you can have a list or even multiple lists. An example would be a list of interfaces.
- Key: Each item within the list is referenced via a key.
- Leaf: Each element of a list has leafs, which contain the actual information.
- Data Type: For each leaf, the data type of the information is indicated.

4.3 Extensible Markup Language - XML

Extensible Markup Language (XML) lets us 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 server). 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, the human-readable nature of XML makes it easier for administrators and developers to understand and work with the configuration data.

4.4 Remote Procedure Call - RPC

Remote Procedure Call is a 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 underlying characteristics. 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>` XML 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 the client 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 GNS3

GNS3 (Graphical Network Simulator-3) software allows us to emulate complex network designs comprising real devices such as Cisco Router/Switch. In this sense, the emulation framework provides virtualization solutions over which actual images of the operating system of such devices can run.

This way, it enables to run real Cisco IOS images using Dynamips in the GNS3 infrastructure, and with this software, we can create realistic network designs on our computer and study in more detail for certifications such as Cisco Certified Network Associate (CCNA) and Cisco Certified Network Professional (CCNP). In the context of this internship, we will exploit the GNS3 to deploy simple network topologies and develop a simple software, that implements a NETCONF client, to remotely modify the devices operation using NETCONF.

5.1.1 Appliances

In the context of GNS3, appliances refer to pre-configured virtual network elements 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 following sections we describe the appliances used in this work.

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, such as 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 destinationMedia Access Control (MAC) (Media Access Control) address of each frame.
- **Broadcast and Multicast Handling:** Switches efficiently handle broadcast and multicast traffic within a Local Area Network (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.

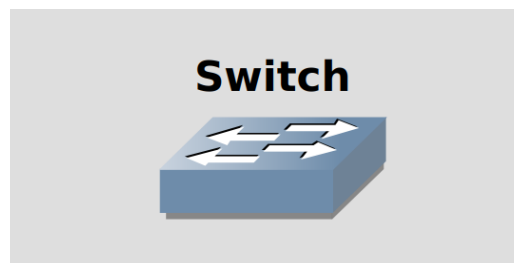


Figure 5.2: Switch

Virtual PC (host)

In GNS3, a Virtual Port Channel (VPC) is a simulated computer that represents an endpoint device within a network topology, or a user. 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.

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

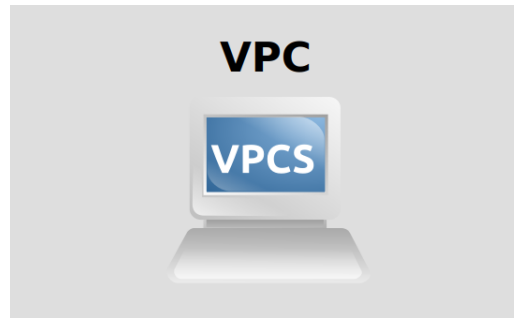


Figure 5.3: Virtual PC

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. 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.

In our case, we utilize the Cloud appliance to communicate the NETCONF client running on the host with the devices within GNS3. This enables the PC to function as a NETCONF administrator and perform tasks related to network management using the NETCONF protocol.

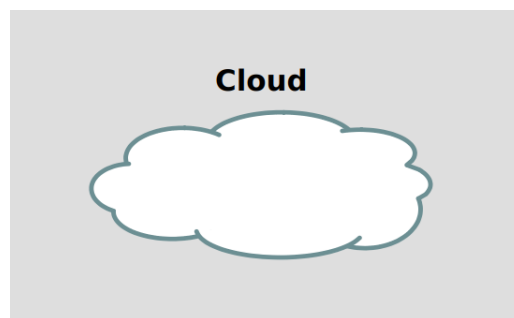


Figure 5.4: Cloud

Network topology:

Therefore, the resulting network topology will be as follows:

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 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

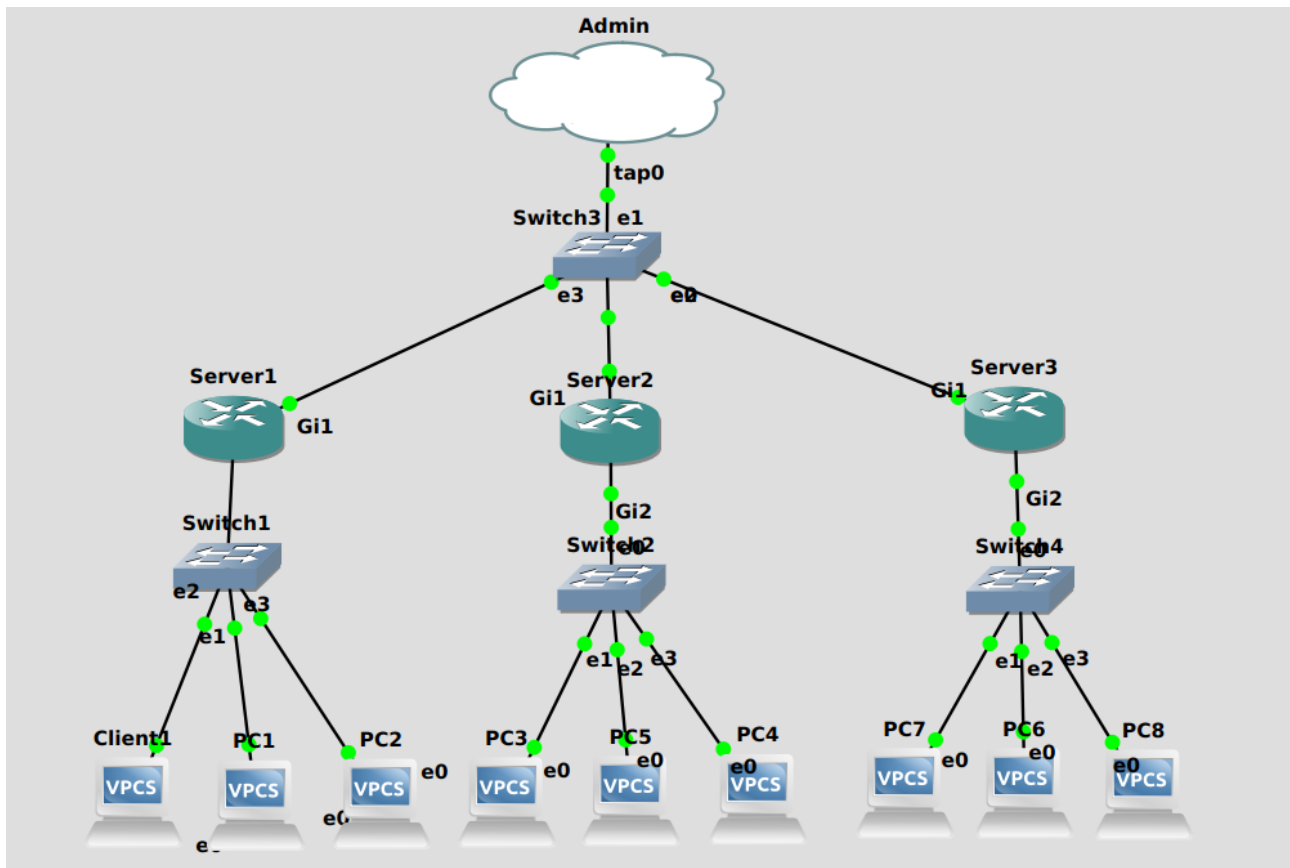


Figure 5.5: Network topology

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:

In the following we describe the DHCP configuration in an interface of the Cisco router used in this work.

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:

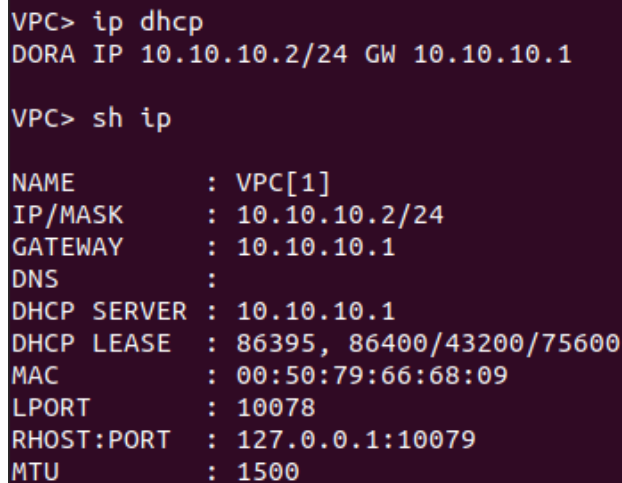
Configuring the default gateway for this subnet.

```
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, and 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 NETCONF-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 ??.

5.4.2 Establishing an SSH connection between the router and host

In this section we describe the process to enable SSH communication between an application running on the host and the Cisco router within GNS3.

On Cisco router

Step 1:

First, we will need to configure the hostname (e.g. Server1) and domain name

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

Step 2:

Generate rsa modulus and enter modulus length (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 the user and password for SSH client to access, and set privilege level 15 to the username 'admin'

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

Finally, we do not forget to save the change by copying the running configuration to the startup configuration, so that it will be persistent when restarting the router.

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.

As commented before, the “hello” message serves as a handshake to establish the capabilities and supported features between the client and server. This 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. As detailed before, the operations are specified through XML elements. 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 is 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 implementation 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 is 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 CSR1000v 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 5 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 sub-networks within this shared IP pool, a systematic approach is required. By analyzing the specific requirements of each sub-network, such as the number of IP addresses needed for its operations, it becomes possible to devise an allocation strategy that fulfills the requirements. 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 allocation process can be streamlined. We will create a script that enables network administrators to input the desired number of IP addresses for each sub-network, and based on this information, it calculates the appropriate range of addresses and assigns them accordingly. The generated sub-networks can be further configured with relevant network parameters such as 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 IP addresses based on individual sub-network 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 sub-networks. 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 sub-network 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 sub-networks 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 sub-network 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 sub-networks based on the proportions determined by each router's IP requirement. Starting from the router with the largest IP requirement, we assign a sub-network 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 sub-networks.

d. Ensuring Efficient Resource Utilization

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

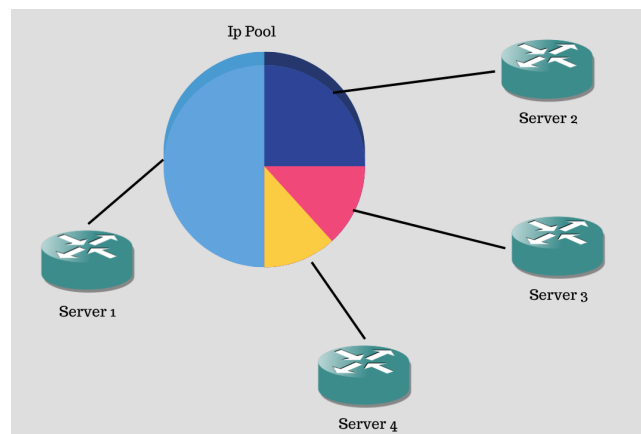


Figure 6.1: Visualizing the IP Pool as a Circular Space.

3. **IP Pool Management:** It is essential to maintain an accurate inventory of the IP pool and track the allocation of sub-networks 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 sub-network.
5. **Configuration and Documentation:** After the sub-network allocation process, configuring each router with the assigned sub-network, subnet masks, default routers, and other relevant network parameters is essential. Additionally, maintaining proper documentation, including address allocation records and network diagrams, aids in network troubleshooting and future expansion.

6.3.3 Example: Router IP Address Requirement Assessment and Subnet Allocation

To illustrate the process of router IP address requirement assessment and address 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 sub-network slice that fulfills its need for 150 IP addresses. Router A receives a subnet mask /24, which provides up to 256 addresses. It is worth noting that smaller sub-networks are not possible, since mask /25 only provides 128 addresses, so that the requirement of 150 addresses is not satisfied. Then a DHCP Server is configured correspondingly: 192.168.0.0 (192.168.0.0 - 192.168.0.255). At this point there is another /24 sub-network available (or 2 /25, or 4 /26, etc.) to share among the rest of routers. Next, we move on to Router A, allocating a mask /25, and a DHCP Server Network: 192.168.0.0 (192.168.0.0 - 192.168.0.127). Again, we need to use the mask /25 to satisfy the requirement of addresses, since the sub-networks with mask /26 only provide 64 addresses. For Router B, a sub-networks with a subnet mask /26 is granted, satisfying its need for 50 IP addresses. Finally, Router C is given another sub-network with mask of /26 to accommodate its requirement of 40 IP addresses. Eventually, the whole pool of IP addresses is used.

Testing Subnet Allocation Code

The following listings showcase the input and outputs of the implemented management solution when used for the example described above.

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
```

Bibliography

- [1] What is Network Configuration Protocol (NETCONF)? — Definition from TechTarget — techtarget.com. <https://www.techtarget.com/searchnetworking/definition/NETCONF>. [Accessed 22-08-2023].
- [2] R. Donato. An Introduction to NETCONF/YANG - Fir3net — fir3net.com. <https://www.fir3net.com/Networking/Protocols/an-introduction-to-netconf-yang.html#Introduction>. [Accessed 22-08-2023].
- [3] R. Enns. NETCONF Configuration Protocol. RFC 4741, Dec. 2006.
- [4] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder. Network Configuration Protocol (NETCONF). RFC 6241, June 2011.
- [5] H. Trevino and S. Chisholm. NETCONF Event Notifications. RFC 5277, July 2008.
- [6] H. Xu and D. Xiao. Data modeling for netconf-based network management: Xml schema or yang. In *2008 11th IEEE International Conference on Communication Technology*, pages 561–564, 2008.

Chapter 7

Annex

7.1 Annex A: Project Code

Listing 7.1: Automated Subnet Allocation.

```
import ipaddress

def generate_subnets(network, num_routers):
    num_ips_list=[]
    devices = {}
    for i in range(1, num_routers + 1):
        key = int(input(f"Enter the num_ips_router for Device{i}: "))
        num_ips_list.append(key)
        value = f"Device{i}"
        devices[key] = value
    sorted_ips_list = sorted(num_ips_list, reverse=True)
    subnets = []
    mask = []
    mask_dec = []
    power_list=[]
    default_router = []
    power = 0
    index = -1
    left_add=pow(2, 32-ipaddress.IPv4Network("192.168.0.0/" + str(network.netmask)).prefixlen)
    for num_ips in sorted_ips_list:
        for x in range(32-ipaddress.IPv4Network("192.168.0.0/" + str(network.netmask)).prefixlen):
            if (pow(2, x)<= num_ips and num_ips < pow(2, x+1)):
                power = x+1
        power_list.append(power)
    for i in range (num_routers):
        if left_add >= pow(2, power_list[i]):
            if i > 0 and power_list[i]==power_list[i-1] :
                subnet_prefix = network.prefixlen + (32-ipaddress.IPv4Network("192.168.0.0/" + str(network.netmask)).prefixlen)
                subnet = ipaddress.IPv4Network((network.network_address , subnet_prefix))
                new_subnet = subnets[-1]+pow(2, power_list[i])
                subnets.append(new_subnet)
                mask.append(subnet.netmask)
                mask_dec.append(subnet_prefix)
                default_router.append(new_subnet + 1)
                left_add = left_add - pow(2, power_list[i])
            print(devices[sorted_ips_list[i]])
            print("Interface IP:", default_router[index])
```

```

        print("Subnet Mask:", mask[index], "(" ,mask_dec[index],")")
        print("DHCP Server Network:", subnets[index])
        print("DHCP Server Default Router:", default_router[index])
        print()

    else:
        subnet_prefix = network.prefixlen + (32-ipaddress.IPv4Network("192.168
        subnet = ipaddress.IPv4Network((network.network_address , subnet_prefi
        new_subnet = subnet.network_address
        subnets.append(new_subnet)
        mask.append(subnet.netmask)
        mask_dec.append(subnet_prefix)
        default_router.append(new_subnet + 1)
        left_add = left_add - pow(2, power_list[i])
        print(devices[sorted_ips_list[i]])
        print("Interface IP:", default_router[index])
        print("Subnet Mask:", mask[index], "(" ,mask_dec[index],")")
        print("DHCP Server Network:", subnets[index])
        print("DHCP Server Default Router:", default_router[index])
        print()

    else:
        print("No ip left for", devices[sorted_ips_list[i]])

network_input = input("Enter the network and subnet in CIDR notation (e.g., 10.10.0.0/
num_routers = int(input("Enter the number of routers: "))
network = ipaddress.IPv4Network(network_input)
generate_subnets(network, num_routers)

```

Listing 7.2: Get-config.

```

from ncclient import manager
import xmltodict
import xml.dom.minidom

# Create an XML filter for targeted NETCONF queries
netconf_filter = """
<filter>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface></interface>
  </interfaces>
</filter>"""

print("Opening NETCONF Connection to {}".format("host: ATN-IPS-R-192.168.100.146"))

# Open a connection to the network device using ncclient
with manager.connect(
    host="192.168.1.1",
    port=830,
    username="admin",
    password="nabil",
    hostkey_verify=False
) as m:

```



```
print("Sending a <get-config> operation to the device.\n")
# Make a NETCONF <get-config> query using the filter
netconf_reply = m.get_config(source = 'running', filter = netconf_filter)
```

Listing 7.3: Get-capabilities.

```
from ncclient import manager
import xmltodict
import xml.dom.minidom

with manager.connect(
    host="192.168.1.10",
    port=830,
    username="admin",
    password="nabil",
    hostkey_verify=False
) as m:

    for c in m.server_capabilities:
        print (c)
```

7.2 Annex B: Router configuration

Listing 7.4: Configure DHCP.

```
configure terminal
ip dhcp pool MyPool // define the pool, the name
network 192.168.1.0 255.255.255.0 // add the subnet 192.168.2.0/24
default-router 192.168.2.1 // configure the default gateway for this subnet
exit
ip dhcp excluded-address 192.168.2.10 192.168.2.20 //to exclud some addresses
from 10 to 20

// you might as well add some a DNS server and the domain
dns-server "address" "address"
domain "your domain"
```

Listing 7.5: Configure NAT.

```
conf t
int FastEthernet 0/0
ip nat inside
exit

int FastEthernet 1/0
ip nat outside
exit

ip nat pool <pool-name = POOL> <starting-IP = 192.168.2.1> <ending-IP = 192.168.2.100>

access-list <acl-number = 1>
permit <source-ip-network = 192.168.1.0> <wildcard-mask = 0.0.0.255>
```

```
//NOTE:The access list configured above matches all hosts from the 192.168.1.0/24 subn

ip nat inside source
list <acl-number = 1> pool <pool-name = POOL>
exit
end

ping 192.168.2.5
show ip nat translations
```

Listing 7.6: Configure NETCONF.

```
enable
configure terminal
netconf-yang
netconf-yang feature candidate-datastore
```

Listing 7.7: Configure SSH.

```
enable
configure terminal

//Setting up the device to run SSH & configure SSH server

hostname Router // configure hostname and domain name
ip domain-name Router-domain
crypto key generate rsa modulus 2048 // enter modulus length of at least 1024, longer
ip ssh version 2 // configure the device to run SSHv2

line vty 0 4 // configure the virtual terminal line settings
login local
transport input all // allow all transport connections, include telnet, ssh
end // return to EXEC mode
show ip ssh // show version and conf info SSH server
show ssh // show the status of SSH server connections
configure terminal
username admin password nabil // create user and password for SSH client to access
username kimdoanh89 privilege 15 // set privilege privilege level 15 to the username "

exit
show running-config // verify the entries
copy running-config startup-config // save the entries in the conf fil

// SSH to the router on the local machine (management server)
ssh admin@192.168.1.1 -p 830 -s netconf
password: nabil
```