Report

Nabil ECH-CHOKHMANY

July 13, 2023

Contents

Chapter 1

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 2

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 3

Overview of Netconf

3.1 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

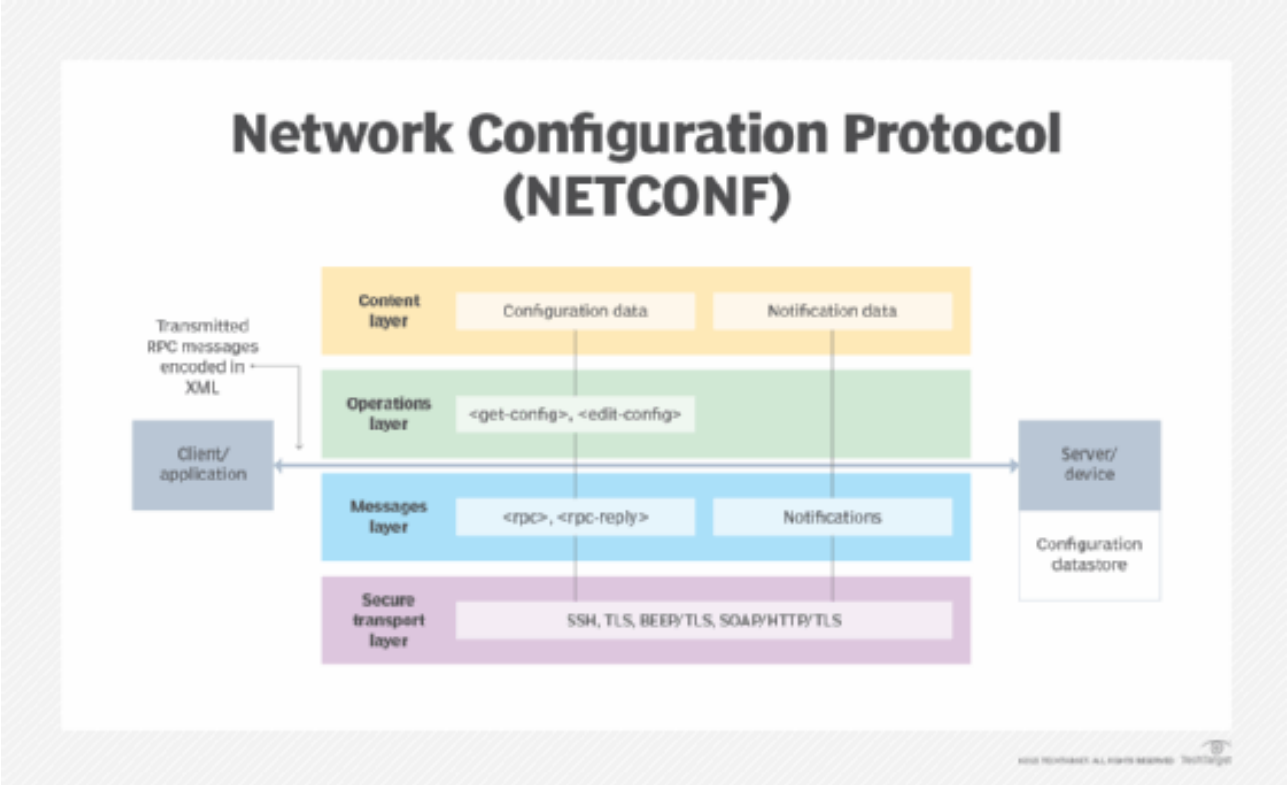

Figure 3.1: Diagram illustrating how Network Configuration Protocol (NETCONF) works.

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¿.

3.2 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).

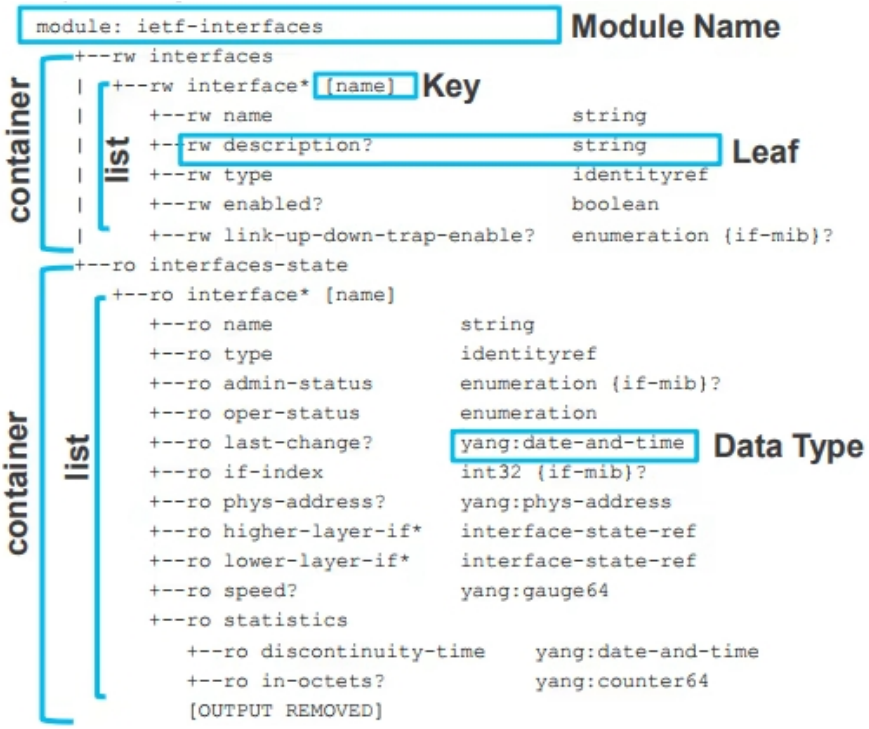

Figure 3.2: YANG Structure.

- 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.

3.3 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.

3.4 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.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <some-method>
    <!-- method parameters here... -->
  </some-method>
</rpc>
```

Figure 3.3: <rpc> Model.

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:

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

Figure 3.4: <rpc> <get> Model.

<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.

```
<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>
```

Figure 3.5: <rpc-reply> <get> Model.