DEGREE PROJECT

NETCONF Device Simulator

Developing a NETCONF based testing platform

William Ennefors

Computer Game Programming, bachelor's level
2018

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

NETCONF Device Simulator

William Ennefors[*]

June 21, 2018

gscept

Lulea University of Technology

Bachelors thesis for Computer Engineering in Game Development

[*]Ennefors@gmail.com

Abstract

When developing network configuration/orchestration applications, it is often convenient to run automated as well as manual tests towards simulated rather than actual hardware devices. Though there exists software of this type, they rarely contain features that can be convenient for testing purposes. The simulation needs to be lightweight, be able to generate special events, and use the NETCONF RPC protocol. This thesis will explain the decisions that were made while developing a simulation software of this type, it will also explain the underlying functionalities and terms associated with the simulation.

Sammanfattning

När man utvecklar applikationer som är till för att bygga och orkestrera nätverkskonfigurationer så är det behändigt att köra automatiserade och manuella tester mot en simulerad enhet mot en riktig enhet. Även fast det exsisterar ett flertal simulatiorer av denna typ så innehåller dem sällan interaktiva funktionaliteter som kan vara behändigt för när tester utförs. Simulatorn ska vara baserad på NETCONF och behöver också vara lättviktig nog att kunna köras på en enkel enhet och snabb att använda. Denna rapport kommer förklara besluten som gjorts under utvecklingen av en simulator av denna typ och kommer även förklara de underliggande systemen och termerna bakom simulatorn.

Contents

1 Introduction

Developing applications that configure networks may require extensive testing [8], a versatile and easy way to do this is to test it against a simulator. These simulators work by having a ”fake” configuration data store on which various NETCONF; the Network Configuration protocol [1], operations are performed. The currently available NETCONF-Device simulators are often very static and slow, they are structured with a focus on the database part. They can also be difficult to modify and add functionalities to. This interactivity and the abillity to modify it could provide scenarios which could test workflows and new solutions for network maintenance.

An example scenario could be the simulator sending a notification to the subscribed clients about a closed port, and the user needs to rethink their current configuration or open it again with an appropriate RPC; Remote Procedure Call. The benefits of doing this are being able to represent a more realistic simulation by providing more scenarios for configurations and RPC testing than just a NETCONF message failing or being an invalid message.

To allow the simulators interactive capabilities to be as flexible as possible the simulator will allow users to develop their functional plugins to the simulator to allow for any scenario. The API; Application Programming Interface, for these user-created plugins needs to be simple and straightforward yet powerful enough to allow the users create any scenario that they need.

This device simulator also needs to be lightweight enough to be able to run on a commodity PC and have a simplicity making it easy to use. To keep the simulator as simple as possible it will not be using the YANG data modeling language and it will only use a few functionalities from the YANG 1.1 RFC; Remote Function Call [4]. By not using YANG the simulator foregoes the complications of its RFC and allows quick and easy testing.

This thesis will explain the choices and decisions made in developing a simulator of this type. It will also explain the basics of NETCONF and its intricacies and the workings around it like SSH and YANG in an understandable way.

2 Previous Work

This work uses the i2cats NETCONF4j server and client code as a base, the reasons for this is that it allows for the base NETCONF functionalities and is easily expandable and modifiable [7]. Another benefit with this server is that it already contains the functionality of saving a configuration datastore; a datastore is an XML document that describes the current configuration of the device, and the ability to get its data. Since all of the server backend and NETCONF message parsing has been implemented and the handling of the <get-config>, <hello> and <close-session> messages is present, this lets the focus be on adding the interactive elements and adding the few remaining commands.

The aditions that are made to this base are:

- Giving the server the ability to keep its datastore in memory instead of saving it to a file. This helps when a user needs to run several simulations on the same computer and not having unnecesary files beeing created.
- Adding support for plugins and customiseable remote procedure calls or RPCs
- Adding few base functionalities for Edit-Config and lock/unlock
- Adding support for notifications and the ability to subscribe to the server
- Adding YANG actions and giving the user the ability to add their own supported actions through the plugin system

3 NETCONF

3.1 About NETCONF

During the early years of the internet, the IETF[1] developed the Simple Network Managment Protocol or SNMP[2] with the intention to make managing and configuring network devices straightforward. During the early 21sth century it became clear that people did not use SNMP in this initially intended way, it was instead mainly being used for network monitoring instead [12]. This lead to the development of NETCONF.

NETCONF or the Network Configuration Protocol is a network management protocol. What this means is that NETCONF can perform several tasks like manipulate, delete or install configurations on network devices, NETCONF does these operations with a easy to understand Remote Procedure Call base layer. The essential base operations of NETCONF are few and easy to use, and they will always be in the protocol, look in Table 1 for a table containing these base capabilities.

[1] Internet Engineering Taskforce
[2] Simple Network Management Protocol

Table 1: Table describing the basic NETCONF functions [1]

Operation	Brief Description
<get>	Retrives the currently running configuration and device information
<get-config>	Retrives part of or all of the configuration datastore
<edit-config>	Edit a configuration datastore by creating a candidate to merge or replace the current with
<copy-config>	Copy the configuration of another configuration datastore
<delete-config>	Delete a configuration datastore
<lock>	Lock a configuration datastore in a device
<unlock>	Unlock a previously locked configuration datastore in a device
<close-session>	Request a termination of a NETCONF session
<kill-session>	Force a termination of a NETCONF session

It is easy to extend and modify the functionality of NETCONF and it is easily readable by humans. Because of this NETCONF is becoming the standardised way of configuring servers remotely [13].

NETCONF is also XML based, when an exchange between client and server occurs the client encodes an RPC in XML format containing the desired information and sends it over a secure connection-oriented session. The NETCONF server, when it has received the new information, responds with a reply also encoded in XML. The benefits to this are that it is more or less readable by a human and that both parties, client, and server, recognize what the exchange entailed.

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

Figure 1: NETCONF get-config RPC

3.2 NETCONF basics

NETCONF has a standard for constructing the RPCs which must be followed [1]. Figure 1 shows how a <get-config> RPC can look. At the top is the root container. This root container tells the NETCONF server that this is an RPC with a specific message ID, this ID is used to make sure that the user and client can identify which message got which reply. There is also an attribute called xmlns which is the namespace of the RPC; this tells the NETCONF server which capability this message lies within. The container below is the NETCONF operation <get-config> this container tells the NETCONF server that the message wants to get an existing configuration from the NETCONF server. Since there can be more than one configuration on a NETCONF server, there needs to be a specified source target. In this case, this message requests the configuration of the current running datastore. When the message has been parsed and handled the NETCONF server will react appropriately, the NETCONF server constructs a reply which contains the datastore and sends it back as an <rpc-reply>.

There are more features to <get-config>, it can also take a filter as a parameter. This filtering method is called subtree filtering, it allows an application, in this case, a NETCONF server, to target and select a specific part or several parts of an XML document, for a NETCONF server the document is the datastore. With the help of this technique the user can filter out information that is not immediately interesting to the user, this can be a tremendous help when working with more massive datastores with vast amounts of data. By having the ability to get specific parts the user can more easily identify what parts that might need editing in the configuration, for one it shortens the time spent on searching for errors, and it helps pinpoint suspected parts of the configuration to change.

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
  <target>
    <running/>
  </target>
  <default-operation>merge</default-operation>
  <config>
    <interfaces
    xmlns="urn:ietf:params:xml:ns:yang:ietf-interface">
     <interface>
       <name>if2 </name>
       <mtu>1600</mtu>
     </interface>
   </interfaces>
  </config>
 </edit-config>
</rpc>
]]>]]>
```

Figure 2: edit-config RPC [1]

3.3 edit-config

Like <get>, <edit-config> targets a specific part inside the datastore. Looking at Figure 2 which shows an example of the <edit-config> operation. After the standard <rpc> container and the <edit-config> container comes the target datastore, in this case, the running datastore is the target. After that comes the <default-operation> this is the edit operation the user wants to perform on the datastore. There are three options, delete, merge and replace. This example will perform the merge operation.

The <config> container tells the NETCONF server that everything inside the container is the new configuration to be merged. What happens next is not entirely clear and needs a bit more explaining. The datastore of the NETCONF server can be massive and have several containers with the same name. In this example, <edit-config> wants to merge the configuration, meaning that it will go step by step to see if each container tag exist. It will first check if a container named <interfaces> exist inside the datastore but because there might be several containers named <interfaces> the XML namespace also needs to be checked. For example, there might exist another <interfaces> container under a different namespace, these two <interfaces> containers are entirely separate. Because of this, it is crucial for the user to know what they are doing or problems like editing or removing the wrong target might occur. If the <interfaces> under the specified namespace can not found then the content in the <config> container will be added into the datastore. If the <interfaces> container with the specified namespace exists the operation will continue to the next step. Now, this is where the simulator will diverge from regular YANG based simulators. In this case, the name acts as a key to identify different interfaces inside the <interfaces> container, and the mtu is the value that will change to the RPC value. If the NETCONF server can depend on YANG, the server will look if there exists an interface module with the name if2, if it exists the mtu will be either added or changed to 1600. The current version of the simulator will not do this, and this is a flaw because of the problems that arise when deciding not to use YANG. The simulator will instead add the everything inside the RPCs <interfaces> container into the datastores

<interfaces> container.

If the edit is a delete operation, it will remove everything inside <interfaces>, and if the edit is a replace operation, then everything in the datastore below the container is replaced with the config in the message. Like previously stated, the namespace also matters.

3.4 The rest

The <get> RPC does the same as <get-config> only that it always targets the running configuration and device state information. <copy-config> copies the target datastore into another datastore <delete-config> deletes a target datastore. The <lock> and <unlock> locks and unlocks datastores, this means that if a client session locks a datastore, other connected sessions cannot perform NETCONF operation on it until the session locking the datastore unlocks it via <unlock> or if the session is closed or killed. <close-session> is the formal way of closing an ongoing session and this is almost always the option used, <kill-session> is the informal way and it will force close the ongoing session.

This section has been a brief explanation of what the basics of NETCONF entail, since NETCONF is such a broad subject one could not go through it all in detail in one report.

YANG was touched on lightly previously, but for the most part, YANG and NETCONF go hand in hand, and it deserves some explanation.

```
module example-sports {

  namespace "http://example.com/example-sports";
  prefix sports;

  import ietf-yang-types { prefix yang; }

  typedef season {
    type string;
    description
      "The name of a sports season, including the type and the year, e.g,
       'Champions League 2014/2015'.";
  }

  container sports {
    config true;

    list person {
      key name;
      leaf name { type string; }
      leaf birthday { type yang:date-and-time; mandatory true; }
    }

    list team {
      key name;
      leaf name { type string; }
      list player {
        key "name season";
        unique number;
        leaf name { type leafref { path "/sports/person/name"; } }
        leaf season { type season; }
        leaf number { type uint16; mandatory true; }
        leaf scores { type uint16; default 0; }
      }
    }
  }
}
```

Figure 3: Typical YANG module [2]

4 YANG

Yang is a data modeling language that is used to model data for the NETCONF Protocol. A typical Yang module; shown in Figure 3, contains a name, description of the module and one or several leaves. These leaves have names and specify different states, information storages, etcetera. They can be keys used to identify data and to verify if the sent configuration is valid against the current datastore. As one reads this one might think that YANG looks like a tree data structure and that is correct [2]. YANG organizes data in a hierarchical tree structure, and this makes the modules easy to understand.

In the example given in Figure 3 one can see that there is one module called example-sports, It contains a namespace, which helps in identifying this module, and different data types. YANG has these types predefined [2] for developers writing their own modules to use but users can also define their own types, in the example, there is a type definition of "season" which is a string and

has a short description of the type. Then in the example-sports module, there is a container named sports, the ”config true;” statement tells us that this data is editable with <edit-config> and can be retrieved with the <get> and <get-config> operations. Inside the sports module, there is a list called ”person”. In the example, it might not be entirely clear on how the list works. The list can, in this case, contain several unique persons. The list stores persons with a key called ”name” which is a leaf with the type string. The key works like a key from a hashmap or dictionary and is used to identify the unique persons in the list. ”Person” also contains another leaf which is called birthday which is a date and time, and this leaf is also mandatory for a ”person” item.

There is also another list in the container sports. This list is called team. Each team in this list is identified with a key that is called name. Each team also contains a list of players. These players are a bit more interesting, and it works like the other two lists but with a constraint and some twists. The constraint is the ”unique number” item, the number is a mandatory uint16, and the interesting thing is its unique statement. The unique statement is an additional way of identifying valid entries into the list, just like in a real sports team the list does not allow players with the same number, each player requires a unique number or else it will not do [2]. The twist is how each player is keyed. This time the key is a leaf reference to a person inside the person list instead of just a name. The last value is a leaf called scores which is just an uint16 with a default value of 0.

The module seems clean and simple but can understandably grow complicated with leaf references etcetera. These complications are one of the reasons this simulator will not use YANG.

These YANG modules also have extension features like notifications or actions; Example of a module with a notification in Figure 4. These extensions, when added to the module, allows a NETCONF server to use these to extend its functionalities. For example, our simulator that allows for both notifications and actions can send notifications to subscribed clients for different events like if a port closes. The actions are also easy to understand since one can see them as functions from regular programming. An action allows a remote client to restart the device the NETCONF server is on or close ports etcetera.

```
module example-event {
       yang-version 1.1;
       namespace "urn:example:event";
       prefix "ev";

       notification event {
         leaf event-class {
           type string;
         }
         leaf reporting-entity {
           type instance-identifier;
         }
         leaf severity {
           type string;
         }
       }
     }
```

Figure 4: YANG module with a notification

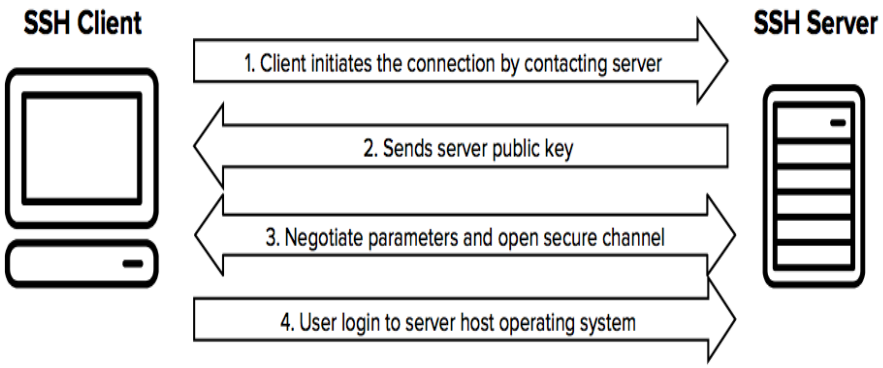

Figure 5: Image of how SSH works [6]

Yang is not a solution to every problem when it comes to modeling data; it is designed to be an expression of NECONF data models not for general XML documents or generall data models [2].

Even though the simulator does not utilize YANG, the importance of knowing what it is and what it can allow is essential. NETCONF and YANG are interconnected, and some useful operations, like actions and notifications require the use of YANG, so replacements and different solutions were made to replace YANG for these functionalities.

5 SSH

For two devices to remotely communicate with each other safely over an unsecured network one can use SSH or secure shell. SSH is a cryptographic network protocol meaning that it allows a server and client stay secure by utilizing a public-key based cryptography to authenticate users and connections. Figure 5 shows an example of this exchange.

These keys are generated in two parts, a public and a private. The host will keep the private key and will not hand it out while the public is given out to trusted sources to authenticate them [6]. This encryption generally makes the connections extraordinarily safe, and it is also rather straightforward to use.

6 Setup of the Simulator and Simulation

This simulator will be built upon the free NETCONF4j Server using JAVA [7]. The NETCONF4j server provides a simple, modifiable framework for a NETCONF server.

A benefit of this is that NETCONF4j Server, as a base, allows for easy saving of a ”fake” configuration of a server-device, meaning that this configuration is not what the actual server runs on. This allows the client to configure and edit the fake config without ruining anything running or getting unwanted errors.

In order to install the simulator all the user needs to do is to build the jar in a desired folder then run it with the terminal or windows commandline. The simulator requires one parameter but can take a second too, the first parameter is the folder which the user wants to use as their plugin folder, the second parameter is a path to an exsisting datastore letting the simulator use that as its initial configuration instead of being empty.

When the simulator starts, it initializes an SSH server, loads all the plugins inside the plugin folder, and if given, loads a datastore from a file. Connecting clients connect to this server, and then the simulator attaches a NETCONF subsystem to the connection. This subsystem parses and picks apart the NETCONF messages received from the connected client. Once this has been set up the simulator will send a hello message to the connected client; this message contains the session ID and the capabilities of the NETCONF server simulation. The hello message works as a handshake of sorts. Firstly it lets the client know that the connection was successful; secondly, it tells the client what it can do and what the connection ID is. This information is useful to know for the connecting user, telling them what they can and can’t do with the simulator.

After this handshake, the connected client can be used to send various NETCONF operations. There is another feature that makes this simulator different from others, and it is that it will have inter-activity and ability for users to tailor it to suit their demands using an easy to use plug-in API.

7 Plugin-api and functionalities

7.1 Introduction

This simulator will allow the users to create custom plugins for their simulator using a simple API. The API allows the user to tailor their simulators functionality to their needs with their plugins, this adds to what the simulation can achieve and makes the base simulator even more lightweight.

There will be three types of plugins the users will be able to develop, Action, Notification and RPC plugins. Each one of these plugin types covers parts of the Yang RFC.

7.2 The types of plugins

Figure 6 shows an example of an action called reset. Action plugins allow the simulator to react to an RPC containing an action. An action is like a normal function in programming, it takes parameters and does something with these parameters. The example plugin that is provided for actions is able to receive two different parameters, <reset-at> and <reset-options>. The <reset-at> parameter is a time on which you want the simulator to reset, and the <reset-options> is the second parameter that, at the moment does not do anything but change the reply message that the simulator sends when accepting the action, it exists to give the user a clue on how to implement multiple parameters to their actions.

Next is the notification plugin, this plugin allows the user to design a notification that will be sent on specified events. This notification contains the time of the event and a notification message. The simulator comes with an example plugin for this called Temperature. The Temperature plugin starts a thread that gradually raises a value representing the temperature of a theoretical server-device, once a threshold has been reached the plugin constructs a notification, warning subscribed clients that the temperature has become too high.

While the examples are simple, these plugins will allow the user to create any scenario or effect they want without the use of YANG, which simplifies the usage of the simulator and lets the user focus on their NETCONF operations.

7.3 Creating a plugin

In order to write a plugin the user needs to first determine what type of plugin they want to create. They will then create a new java class that implements the appropriate interface, NotificationPluginInterface for notifications, ActionPluginInterface for actions and RPCPluginInterface for RPCs. Then the user needs to override the interface functions and then implement their desired functionality into the plugin. After the plugin has been compiled into a java class the user can create or use an existing plugin folder and put their class into that folder. In order to load their plugin the user simply needs to input the path to the plugin folder as an argument when launching the simulator, the

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <server xmlns="urn:example:server">
        <name>apache-1</name>
      <reset>
        <reset-at>2018-05-09T13:19:16</reset-at>
        <reset-option>Server Reset</reset-option>
      </reset>
    </server>
  </action>
</rpc>
]]>]]>
```

Figure 6: Option to invoke a server reset

```
public void INIT()
{
    item containerItem = new pathItem("container", this.name);

    pathItem keyItem = new pathItem("key", this.name);
    keyItem.key = "apache-1";

    pathItem actionItem = new pathItem("action", this.name);
    actionItem.action = "reset";

    pathItem containerItemI = new pathItem("containter", this.name);

    pathItem actionItemI = new pathItem( "action", this.name);
    actionItemI.action = "shutdown-at";

    containerItemI.container = new HashMap<>();
    containerItemI.container.put("shutdown", actionItemI);

    containerItem.container = new HashMap<>();

    containerItem.container.put("name", keyItem);
    containerItem.container.put("reset", actionItem);
    containerItem.container.put("secret", containerItemI);

    this.path = new HashMap<>();

    this.path.put("server", containerItem);
}
```

Figure 7: INIT function of the reset plugin

plugin subsystem then loads all JAVA .class files in the specified plugin folder and stores them in a corresponding array. If the plugin is an action or RPC plugin the subsystem will store their user specified path in a hashmap.
Since notifications are supposed to be sent by a NETCONF server whenever something happens to the NETCONF servers datastore or something more specific to the plugin triggers, the plugin needs to be run as a thread that can constantly check if changes have been made. That is why the notification plugins need the run function which contains the code that will be executed as a thread.

Actions are more direct since they are triggered by a specific RPC rather than a status change in the NETCONF servers or a change in its datastore. This means that the action plugins just have an execute function which will perform the execution of the action.

The custom RPCs covers all the other functionalities that does not fall under notification or action, this can be, for example, a <get-temperature> RPC which returns the current temperature of a NETCONF server-devices CPU.

The action and RPC plugins are a bit more complicated than the notification plugin. That is because of the INIT function. In this function the user will have to create and specify something that could be called a simplified YANG module. They do this with the help of the pathItem class. This class can be 3 different types, a key, an action or a container. The key is an identifier used to identify a path, this could be a name, an ID etcetera. The action item is a string used to identify the actual name of the action/actions that the plugin is able to execute. And finally a container, this

container can contain actions, other containers or more keys, with these 3 the user can design the path on which the simulator uses to identify and execute the correct actions. See Figure 7 for an example of this function.

7.4 Plugin system

This section will explain what the simulator does when a user wants to invoke an action with the help of the example action plugin.

Firstly the user needs to construct a NETCONF RPC containing an action, see Figure 6 for an example on how the message for this specific plugin looks.

When the simulator receives the example message, it will start searching for the plugin in the plugin subsystem. The example RPC has a container called <server> under the XML namespace ”urn:example:server” and the simulator searches in the combined path for the term until it finds it. Then the simulator, if the term has been found, will continue going deeper into the combined path until it finds the corresponding action.

The action RPC message must only contain one action [4], in the example that action is <reset>. The action is identified via the stored action plugins. When the user creates their plugins, they construct a path on which the message is compared against to determine if the action exists. In this case the simulator first identifies that the RPC is an action, it will then move to the plugin subsystem. The simulator will now search for the next term ”server” in a combined hashmap containing a merged path of every loaded action plugin. When and if the simulator found the ”server” term it will see that it is a container and go into it, it will now search inside the server container for the key name and the actual action <reset>. When and if the action <reset> has been found, a separate java DOM will be created.

This DOM only contains everything inside the <reset> tag and will be passed as a parameter for the plugins execute function. The reason for having a combined hashmap with these containers is because the user might want to create several plugins with the same path but different execute functions and actions. For example, another action could exist inside the <server> tag beside <reset> called <close-port> which takes a port number instead of time. Thanks to this, plugins are clean and straightforward but the simulator does support a single super-plugin that can do every action, but it will get messy having such a large java class do so much.

Allowing users to create useful plugins with minimum java programming knowledge allows them to create hypothetical YANG modules without the contrivances of the YANG language, it lets the user to quickly get going with the testing of their NETCONF operations.

8 Social, Ethical and Environmental Considerations

Since this simulation software is working locally on a users computer it will not have an impact on the user security nor will it impact the environment around them, and even if it were to run on an open port allowing for remote connections it would not be much of a problem because of the

SSH server. What it will provide is a quick and easy NETCONF simulator which might reduce headaches when testing their configurations, it will also make sure that the NETCONF RPCs have been properly tested and that they work as intended, this can save companies lots of money and time since they do not have to send configurations that might risk crashing or carry some fatal error.

9 Results

The simulator, as of May 18 contains all the original features that were requested. It is lightweight both processor and memory wise, it allows all the base functionalities of NETCONF, it allows for user created and customised actions, notifications and RPCs via a plugin system and it does not require the use of YANG. This means that the simulator is completely customisable and simple to understand, the only requirements of using the simulator is to have an understanding of NETCONF and basic JAVA.

The simulator could be expanded on to allow the use of filters, subtree filters, and the multitude of YANG statements from the YANG RPC [2] and the YANG 1.1 RPC [4].

While developing the simulator, it was decided that implementing as many features as possible would be preferable to error handling. Therefore, the most significant flaw of the simulator is its poor error handling. The user is expected to know what they are doing and does not handle some of the different errors that may occur while doing this. There are some features that could not be implemented in its entirety due to time constraints. These include subtree-filtering and some optional, but useful, <get-config> functionalities. Something else to add would be a fix to the <edit-config> problem mentioned in section 3.3.

To parse and handle the NETCONF messages that are sent from the client, the simulator uses JDOM2 to build virtual XML documents. The benefits of doing this is that JDOM2 can create easily navigatable data from XML files or in the simulators case, strings.

10 Discussion

Even though NETCONF is made to be simple, there are still problems with its readability and simplicity. There can be difficulties especially when the user is new to the concept of NETCONF and Yang.

These are a few questions that generally aren't that well explained. Most of the explanations are understandably brief since this is such a broad subject.

- What is a datastore and how does it work?.
- How does a NETCONF Server work
- Does the NETCONF rfc connect via SSH and how does it utilize it?
- What is the difference between a Yang module and a working NETCONF datastore?

The answer to the first qustion is that a datastore is a place in which a NETCONF server can store and access information. The datastore is for the most part stored inside an XML document but it can be stored inide a database or just inside flash memory locations. The datastore is where all NETCONF operations will take place. Every NETCONF operation is connected to the datastore in some way or another. Even things like actions and notifications exists in the space of the datastore. Therefore a datastore is an essential fundament for NETCONF.

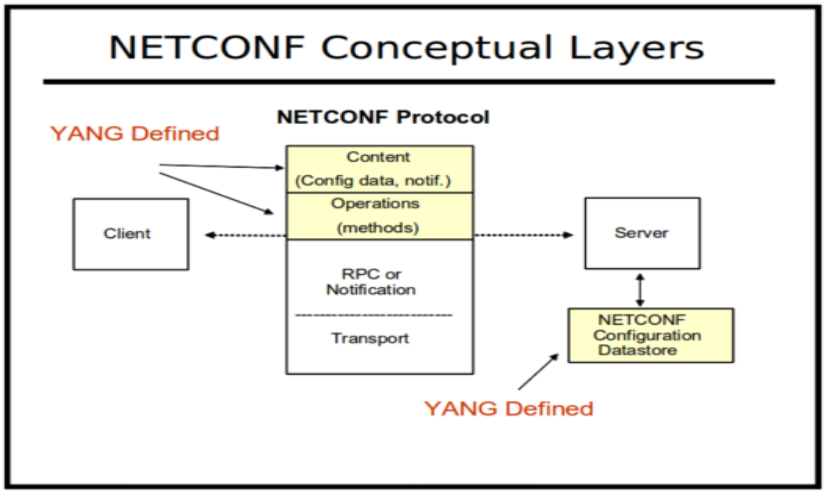

Figure 8: NETCONF Layers [9]

When it comes to the second and third question the answer could be stated as such that there is no ”NETCONF Server” but rather it is an SSH server that runs with NETCONF capabilities running underneath [3], but ”NETCONF server” is the best descriptor for it. A NETCONF message is basically a small XML document and the NETCONF server has the capabilities to parse the message and then react appropriately to that message. Figure 8 shows a conceptual model of the working layers of a NETCONF server.

The answer to the fourth question is that a YANG module is basically a datastore. NETCONF operates against YANG in the same way it operates against a datastore. The differences are that YANG can be more useful to a developer thanks to its structure and that it is its own language instead of just an XML document.

While much might seem unexplained and there are probably remaining questions, both NETCONF and YANG have several IETF RFCs, off-shoots, exceptions and extra features that it could take weeks to explain and understand it all. This is one of the biggest flaws of NETCONF and YANG. This can be daunting to new users and makes older users scratch their head more often than not.

The biggest reason why it was decided to not include YANG to the simulator is because it can be problematic and it does have limitations. Some of these include:

- YANG can’t have true maps
- YANG can’t have keyless lists
- YANG can’t tell from encoded data which ”choice” was taken
- All data in YANG is rooted in a tree which makes it difficult to have free single data

There are more problems that is better explained in Colin Dixon's talk, YANG Modeling: The Good, The Bad, and The Ugly [11]. Thanks to these flaws, when under time constraints or quick and easy testing, skipping YANG entirely speeds up the process of testing NETCONF operations even though skipping it makes the implementation of some functionalities complicated.

11 Future work and Conclusion

This thesis provides an explanation of the choices that was made during the development of this NETCONF Device Simulator. The simulator is lightweight and fast, making testing quick and easy, it is expandable and modifiable via plugins letting the user have total freedom constructing their tests. The goal for this simulator was to provide a more structured base that can be expanded and built upon in order for network developers to have a simple, and YANG less choice when testing NETCONF operations and configurations.

In the future, like most applications, several improvements could be made. Examples of improvements are a proper, good error handling system instead of a basic error handling system. Adding features like subtree-filtering and adding support for more types of plugins. Creating a better and faster filtering method for parsing and identifying RPCs. Optimizing the code and simplifying it, there might be redundancies that needs to be cleaned up and finishing touches to allow custom RPC plugins. These are a few suggestions on how to continue to improve this simulator.

Implementing a simulator of this type without prior knowledge of NETCONF, YANG or any of these terms have been a valuable challenge requiring extensive research, preparation and testing. But in the end it was completly doable and as previously stated a valuable experience.

References

[1] Internet Engineering Task Force (IETF), R. Enns, Ed., Juniper Networks, M. Bjorklund, Ed., Tail-f Systems, J. Schoenwaelder, Ed., Jacobs University, A. Bierman, Ed., Brocade, [June 2011] *Network Configuration Protocol (NETCONF) RFC 6241* https://tools.ietf.org/html/rfc6241 [Accessed 2018-05-18]

[2] Internet Engineering Task Force (IETF), M. Bjorklund, Ed., Tail-f Systems, [October 2010] *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) RFC 6020* https://tools.ietf.org/html/rfc6020 [Accessed 2018-05-18]

[3] Internet Engineering Task Force (IETF), M. Wasserman, Ed., Painless Security, LLC, [June 2011] *Using the NETCONF Protocol over Secure Shell (SSH) RFC 6242* https://tools.ietf.org/html/rfc6242 [Accessed 2018-05-18]

[4] Internet Engineering Task Force (IETF), M. Bjorklund, Ed., Tail-f Systems, [June 2011] *The YANG 1.1 Data Modeling Language RFC 7950* https://tools.ietf.org/html/rfc7950 [Accessed 2018-05-18]

[5] Network Working Group , S. Chisholm, Nortel, H. Trevino, Cisco *The YANG 1.1 Data Modeling Language RFC 5277* https://tools.ietf.org/html/rfc5277 [Accessed 2018-05-18]

[6] SSH Communications Security, Inc., [March 6, 2018] *SSH (Secure Shell)* https://www.ssh.com/ssh/ [Accessed 2018-05-18]

[7] AIMS '09 Proceedings of the 3rd International Conference on Autonomous Infrastructure, Management and Security: Scalability of Networks and Services Enschede, The Netherlands — June 30 - July 02, 2009, Pages 83 - 94 Ha Manh Tran, Iyad Tumar, and Jurgen Schönwälder, [AIMS 2009] *NETCONF Interoperability Testing*

[8] Internet Architectures and Services (IAS) - i2CAT Foundation, [February 4, 2015] *netconf4j server* https://github.com/dana-i2cat/netconf-server [Accessed 2018-05-18]

[9] Netconf Central, Andy Bierman *Netconf Central, Network Configuartion Protocol* http://www.netconfcentral.org/netconf_docs [Accessed 2018-05-18]

[10] Jason Hunter, Rolf Lear, JDOM Project [February 15, 2015] *JDOM and JDOM 2* http://www.jdom.org [Accessed 2018-05-18]

[11] OpenDaylight Developer Track at the Open Networking Summit 2015 June 15th, 2015, Colin Dixon, YANG Modeling: The Good, The Bad, and The Ugly *YANG Modeling: The Good, The Bad, and The Ugly* http://colindixon.com/wp-content/uploads/2014/05/YANG-good-bad-ugly-ONS-2015.pdf [Accessed 2018-05-18]

[12] Article, James Yu, Carl Moberg, Khalid Elmansor and Imad Ajarmeh, Tail-f Systems, first published [DECEMBER 17, 2009] *NETCONF: A new approach to network management* http://picmg.mil-embedded.com/articles/netconf-new-approach-network-management/ [Accessed 2018-06-01]

[13] Christos Rizos, https://www.snmpcenter.com [Oct 25th, 2016] *Why use NETCONF/YANG when you can use SNMP and CLI?* https://www.snmpcenter.com/why-use-netconf/ [Accessed 2018-06-01]