Tampere University

ANIL BOLAT

# CLOUD-NATIVE REALIZATION OF NETWORK CONFIGURATION PROTOCOL

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Kari Systä
Tapio Elomaa
March 2020

# ABSTRACT

Anil Bolat: Cloud-Native Realization of Network Configuration Protocol
Master of Science Thesis, 36 pages
Tampere University
Master's Degree Programme in Information Technology
March 2020

---

Many of the telecommunication companies aim to support Network Configuration Protocol (NETCONF) to manage their large network in cloud-native environment. The NETCONF protocol provides automation and security using permanent SSH and TLS connections as well as cloud-native brings scalability advantages. However, supporting the NETCONF protocol in cloud-native environment represents challenges since the NETCONF protocol is not stateless.

The thesis implements a proof of concept for cloud-native Network Configuration Protocol and investigates issues of such an implementation. The approach in this thesis is to have two implementations of standard Network Configuration Protocol and Network Configuration Protocol Call Home in cloud-native environment. A solution is applied together with these implementations by terminating the permanent established sessions in the end of messaging. The evaluations are made by analysing changing number of connections and events per connection in the both implementations.

Based on the evaluation of the proof of concept, the results indicate that terminating the established NETCONF sessions in the end of messaging is an operable solution. However, it is also observed that throughput and CPU could be limitations for such an implementation in cloud-native environment. In addition, it must be considered that authentication time is affected based on chosen security provider.

Keywords: Cloud-native, NETCONF, scalability, network management, stateless

# PREFACE

I have shared good memories and learned a lot during my time at Tampere University.

I would like to thank Nokia, and especially Miia Forssell for her continued support and our regular discussion meetings during the writing process of the thesis. I would also like to thank my thesis supervisor Kari Systä for his continued feedbacks and guidance.

I would like to thank my family for their great support during my studies and the thesis. I feel grateful for them.

Tampere, 1 March 2020

Anil Bolat

# CONTENTS

# LIST OF FIGURES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| LTE | Long-Term Evolution |
| NETCONF | Network Configuration Protocol |
| IETF | Application Programming Interface |
| API | Internet Engineering Task Force |
| RPC | Remote Procedure Call |
| XML | Extensible Markup Language |
| SSH | Secure Shell |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| IaaS | Infrastructure as a Service |
| IETF | Internet Engineering Task Force |
| IaaS | Infrastructure as a Service |
| PaaS | Platform as a Service |
| SaaS | Software as a Service |
| NMS | Network Management System |
| NE | Network Element |
| JNC | Java NETCONF Client |
| CLI | Command-Line Interface |

# 1. INTRODUCTION

Along with the advancements of 5G in telecommunication industry, 5G network will provide serious speed and latency improvements over existing LTE technologies. Additionally, the changings will affect network size, endpoint devices, and network traffic. It is predicted that network will be dramatically expanded and grown. The significant growth in network means that the number of endpoint devices will be increased extremely, and larger networks will surround people. Hence, the large networks need to be supported by the communication companies to serve their customers well.

The business needs of telecommunication industry go towards managing these large networks. Network companies and service providers focus on reducing cost, time and human interaction as far as possible in network element configuration. Service oriented approach is considered to have in network element management to increase effectiveness and automation. Besides that, vendor independence and security are the other essentials in the business needs of the industry. Majority of customers in the industry want to securely manage the configurations of their network devices. Therefore, the vendors ask the Network Configuration Protocol (NETCONF) to be used since it supports encrypted connections using Transport Layer Security (TLS) and Secure Shell (SSH) and is reliable and automated. On the other hand, the telecommunication companies plan supporting management of the large network via cloud-native environment due to scalability advantages.

Supporting the NETCONF protocol in cloud-native environment becomes problematic since, first, the protocol is not stateless. Permanent established connections are needed for each network element in the NETCONF protocol system. So, the NETCONF interface is based on a dedicated secure session and this is rather incompatible with cloud principles. There are serious concerns about scalability of such a solution from network management system point of view.

In this work, the aim is to analyze the behavior of NETCONF from network management system perspective in cloud-native environment with large networks. In this thesis, finding the answers to the following research questions are studied:

- What are the actual problems and challenges of NETCONF in cloud-native environment?
- How should NETCONF Event Notifications in cloud-native environment be supported from network management system perspective in case of large networks?
- What are needs and limitations in the environment on scalability?

Two proof of concept implementations are developed to investigate the research questions. The first implementation is for standard NETCONF Event Notifications and the another one is with NETCONF Call Home featured of NETCONF Event Notifications. By comparing these implementations and their test results, limitations and needs are studied and stated.

The thesis is divided into 5 sections. Section 1 is introduction where the research question and structure of the thesis are provided. Section 2 presents the NETCONF protocol and event notifications, cloud-native and scalability topics, and problems between the NETCONF protocol and cloud-native. Section 3 explains details about the implementations. In section 4 evaluations and findings of the thesis are presented. Conclusion is given in section 5.

# 2. THEORETICAL BACKGROUND

This chapter discusses fundamentals of the NETCONF protocol, NETCONF Event Notifications, cloud-native, scalability and the problems which cloud-native brings to NETCONF. The study in this thesis is specifically related to the NETCONF in cloud-native environment. Therefore, this chapter focuses on topics which are important in order to understand the concepts of the NETCONF protocol with Event Notifications on cloud environment.

In the first section 2.1, basics of NETCONF is introduced. NETCONF Event Notifications are presented as well as NETCONF Call Home are discussed. The second section 2.2 gives a brief overview of cloud-native topic. The third section 2.3 goes through scalability. The fourth section 2.4 presents the problems and challenges of working with cloud-native and NETCONF.

## 2.1 Network Configuration Protocol (NETCONF)

The NETCONF protocol or The Network Configuration Protocol is a standard interface for installing, deleting and modifying the configuration data on network devices. It is a protocol defined by Internet Engineering Task Force (IETF) in RFC-6241. The NETCONF protocol is developed to standardize remote configuration of network devices via an application programming interface (API). The protocol implements the API to be exposed by the network devices. The NETCONF API is used to manage configuration data of the network devices by network manager applications [1]. In the terminology of NETCONF, the client indicates a NETCONF manager and the server is a NETCONF agent which is a network element as well.

The NETCONF protocol is defined with four layers as shown in Figure 1 [1].

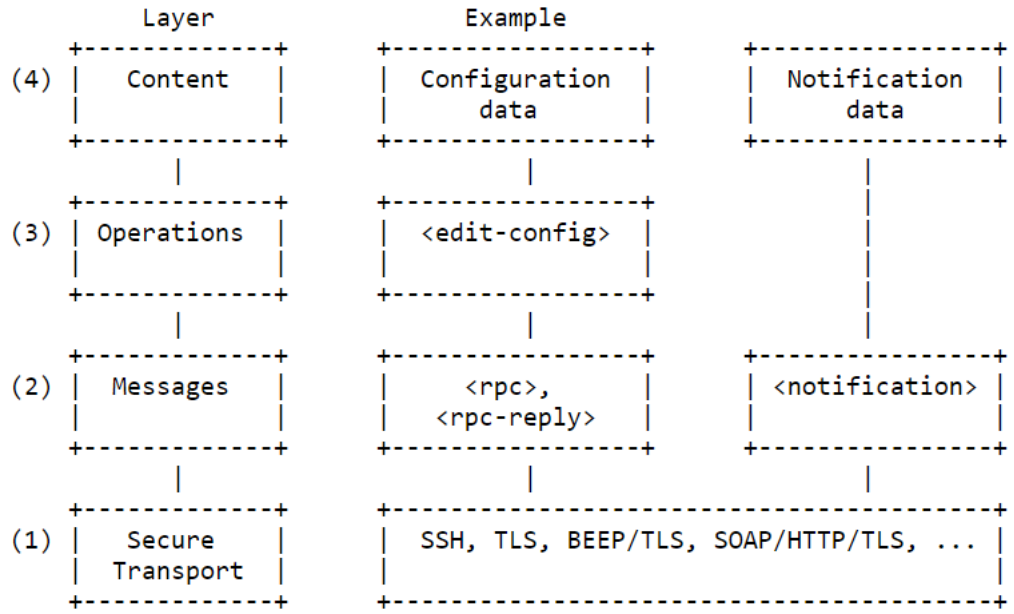


***Figure 1.*** *The NETCONF protocol layers*

Network management system, which is client, and network element, which is server, are the actors of the NETCONF protocol. First, the network management system initiates the communication towards the network element. The communication starts with a request for a secure connection between these two parties. According to the protocol, a secure transport layer needs to be established to provide a persistent connection, authentication and encryption. Then a Remote Procedure Call (RPC) based communication or a notification is applied between a client and a server. An Extensible Markup Language (XML) encoding is used for both the configuration data and the protocol messages in the NETCONF protocol. During the communication, a NETCONF client encodes the data in XML format and sends it over a secure connection-oriented session. The other party, which is a NETCONF server, receives the data and replies the request with a response encoded in XML. Since both the request and the response are fully described in XML format, the message contents are recognizable. An RPC message contents below are both examples of request and response of the NETCONF <get> method [1].

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

***Figure 2.*** *The NETCONF <get> request message [1]*

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

***Figure 3.*** *The NETCONF <get> response message [1]*

The NETCONF protocol provides a set of base operations by default with regards to configuration data manipulation and session termination. The base operations of the NETCONF protocol are shown in Table 1 [1]. The base operations could be extended by additional NETCONF capabilities. Event notification, partial locking configurations and monitoring are some of the additional capabilities supported by the NETCONF protocol for further operations. The additional capabilities are also defined in their own RFCs.

***Table 1.*** *The NETCONF protocol base operations*

| **Base Operation** | **Description** |
|---|---|
| get | Retrieve running configuration and device state information. |
| get-config | Retrieve all or part of a specified configuration datastore. |
| edit-config | Edit a configuration datastore by creating, deleting, merging or replacing content. |
| copy-config | Create or replace an entire configuration datastore with the contents of another complete configuration datastore. |
| delete-config | Delete a configuration datastore. |
| lock | Lock the entire configuration datastore system of a device. |
| unlock | Release a configuration lock, previously obtained with the <lock> operation. |
| close-session | Request graceful termination of a NETCONF session. |
| kill-session | Force the termination of a NETCONF session. |

The following procedures take place for a secure NETCONF session from session establishment to closing session.

1. An SSH session is set up between manager and agent, and the manager initiates NETCONF as an SSH subsystem called "netconf".
2. The supported capabilities are exchanged by sending hello packets between manager and agent.

3. For configuration management purposes, a request is sent by manager to agent.
4. As a response of the request, a reply is sent by agent to manager.
5. Once the required operations are done, a request is sent to end the NETCONF session.
6. A response is sent by agent and the NETCONF session is ended.

### 2.1.1 **NETCONF Event Notifications**

The event notification capability in the NETCONF protocol aim to provide an asynchronous message notification service. This capability was added on top of the base NETCONF definition and defined by Internet Engineering Task Force (IETF) in RFC-5277 [3]. An event notification indicates that some configuration changes exist in the NETCONF agent where the changes might be new configurations, deleted configurations, or modified configurations.

A sequence diagram of NETCONF Event Notifications is given in Figure 4 [3].

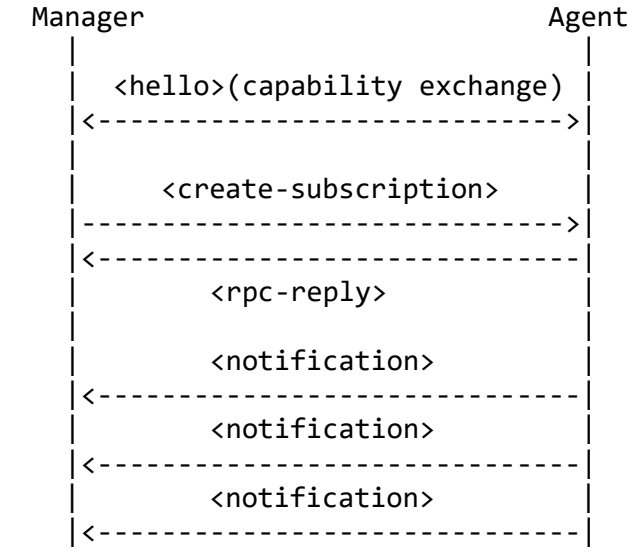


***Figure 4.*** *NETCONF Event Notifications sequence diagram*

As soon as SSH and NETCONF sessions are established, hello messages must be sent for a capability exchange. The NETCONF Event Notifications capability must be added in the content of the hello messages. During the supported capability exchange between the NETCONF agent and manager, the both sides indicate that they support the event notification capability with the adding.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:xml:ns:netconf:base:1.0
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:notification:1.0
    </capability>
  </capabilities>
  <session-id>1</session-id>
</hello>
```

***Figure 5.*** *Hello message example with the event notification capability*

The NETCONF manager needs to create a subscription request and send it to the NETCONF agent to indicate which events the manager wants to receive. A parameter called ‘stream’ in the subscription request shows the event of interest and its value must be ‘NETCONF’ to receive the event notifications.

```
  <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
    <create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notifi-
cation:1.0">
      <stream>NETCONF</stream>
    </create-subscription>
  </rpc>
```

***Figure 6.*** *A subscription request example with NETCONF stream*

Thus, the procedure of NETCONF Event Notifications begins with the creation of subscription request by the NETCONF manager and sending to the NETCONF agent to receive event notifications. Then, the NETCONF agent sends a subscription response to the NETCONF manager if the subscription is successful.

Once the subscription has been set up, the NETCONF agent sends the event notifications to the NETCONF manager asynchronously over the connection when events of interest has occurred. The event notifications are sent at the end of a successful configuration operation as one message that shows the set of changes rather than showing individual messages for each line that is changed in the configuration. The event notifications will be sent by the NETCONF agent until the NETCONF session is terminated [3].

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"
  <eventTime>2019-05-22T10:24:48.422488-026:00</eventTime>
  <netconf-config-change xmlns="urn:ietf:params:xml:ns:yang:ietf-netco
nf-notifications">
    <changed-by>
      <username>admin</username>
      <session-id>1</session-id>
      <source-host>127.0.0.1</source-host>
    </changed-by>
    <datastore>running</datastore>
    <edit>
      <target    xmlns:notif="http://tail-f.com/ns/test/notif">/notif:
test</target>
      <operation>replace</operation>
    </edit>
  </netconf-config-change>
</notification>
```

***Figure 7.*** *Notification message example*

### 2.1.2 **NETCONF Call Home**

In the NETCONF protocol, the communication flow starts with the NETCONF manager's secure connection initialization request and continues until the established sessions are closed or killed by the parties. The NETCONF communication steps are listed and explained in section 2.1 in detail. However, in a few certain circumstances, the NETCONF manager may not trigger the communication flow. Possible reasons are listed below [4].

- A firewall might not allow the NETCONF manager to connect the NETCONF agent.
- The NETCONF agent might not have any open ports for the management connections.
- The NETCONF agent may be deployed behind a firewall that implements Network Address Translation for all internal network IP addresses.
- It is reliable, secure and easy to maintain to open one port in the NETCONF manager-side rather than opening one port on each NETCONF agent.

The listed bullet points specify communication problems in the NETCONF protocol. In order to solve these problems, NETCONF Call Home is defined by Internet Engineering Task Force in RFC-8071 [4]. NETCONF Call Home specifies a standard way to initialize a NETCONF session by the network element. NETCONF Call Home brings a solution by reversing the communication initialization. When the agent is deployed behind a fire-

wall that doesn't allow any management access to the network, the communication cannot be initialized by the manager. By reversing the direction with NETCONF Call Home, the communication can happen without adjusting the firewall configurations.

NETCONF Call Home provides a security enhancement on top of standard NETCONF protocol. In NETCONF Call Home protocol, the manager has one open port to make the NETCONF communication available and the agents send the initial requests to this port. Hence, the NETCONF agents do not need to open any ports and listen to them for making NETCONF communications. In addition to that, the manager is also able to manage the multiple agents using this one open port with the NETCONF protocol. So, NETCONF Call Home helps for centralized management system for the operators.

A sequence diagram of NETCONF Call Home is given in Figure 8 [4].

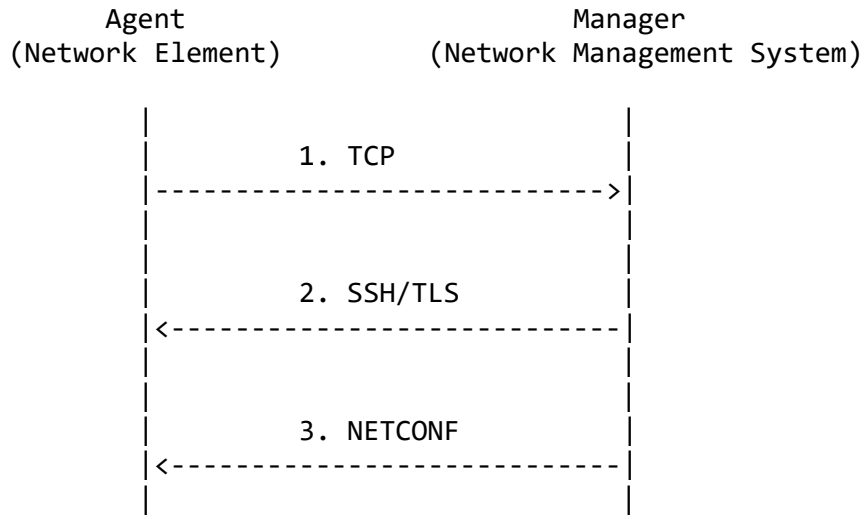


***Figure 8.*** *NETCONF Call Home sequence diagram*

The following steps are taken place in NETCONF Call Home.

1. The manager listens for TCP connection requests at 4334, which is the recommended port for NETCONF Call Home protocol.
2. A TCP connection request is sent by the agent towards the manager.
3. The manager accepts the TCP connection request and sends a TCP response to the agent.
4. The manager initializes the SSH connection towards the agent.
5. The SSH session is established once cipher negotiation and key exchange are completed between the parties.
6. The manager starts NETCONF session towards the agent.
7. The supported capabilities are exchanged by sending hello packets between the manager and the agent.

Since the communication protocol in NETCONF Call Home starts with NETCONF agent's TCP connection request, the NETCONF manager must listen incoming TCP connections in the first place. The manager must support listening TCP connections on the IANA-assigned ports [4]. In that case, this port number must be 4334 and 4335 for communicating NETCONF Call Home over SSH and TLS, respectively. However, this port configuration can be changed to accept incoming TCP connections. After the TCP request is accepted by the NETCONF manager, using the already established TCP connection, the NETCONF manager starts a secure connection. Once the secure connection is established, the NETCONF manager initiates NETCONF connection.

It should be noticed that the client and server initial roles are different in NETCONF Call Home from standard NETCONF protocol. In standard NETCONF protocol, a network element is the server. However, in NETCONF Call Home, the network element is the TCP client since it initializes the TCP connection in the first place. After the initial role, the next roles of the network element remain the same. The roles are the SSH/TLS server for the secure transport-layer and the NETCONF server for the application-layer. On the other hand, the NETCONF manager becomes the TCP server at first. In the sequel, its roles are the SSH/TLS client and the NETCONF client.

## 2.2 Cloud-Native

Cloud computing is the on-demand delivery of IT resources such as compute power, storage, network, database, software applications, and other IT resources via the internet. Cloud computing providers offer three main services which are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). These cloud computing services are provided with pay-as-you-go pricing model so that customers pay only for the resources which they consumed and used. With usage-based billing option, in the first place, cloud computing reduces the business area's cost. Additionally, the cloud computing services are simply available and accessible over the internet. By taking advantages of virtualization and accessibility in cloud computing, the cloud computing services are expandable and reducible with adding or releasing extra infrastructural resources in a self-service mode quickly.

Along with the developments and popularity of cloud-computing, cloud-native term has started to be used more often in the software engineering field. Cloud-native computing has become a modern way of the software development in recent years. On top of that, a foundation named the Cloud Native Computing Foundation was launched by the Linux Foundation in 2015. The foundation aims building ecosystems and communities for high-quality cloud-native projects.

The Cloud Native Computing Foundation defines, in short, cloud-native as building and running scalable applications in modern and dynamic environments. More precisely, cloud-native is a particular approach for designing, building and running applications based on infrastructure as a service with modern tools such as continuous integration, container engines and orchestrators. Cloud-native applications are running in a container, building microservice architecture and using continuous integration and continuous delivery.

Architectures of cloud-native utilize the on-demand delivery and provide significant improvements in software engineering, scalability, high availability, and cost savings. Cloud-native approach gives advantages releasing software projects earlier from business ideas to the market. This speed change provides incremental improvements and efficiency to the business area.

One of the key attributes of cloud-native applications is that they are packaged as lightweight containers. This aspect makes the applications independent service and scalable. Hence, the applications can scale-out and scale-in rapidly. Another aspect is loosely coupled design. The applications have loosely coupled services which discover each other when the applications are up and running. The services exist independent of the others [5].

The services use API calls via representational state transfer (REST) protocol. REST APIs solve the problem about communications between different services. The APIs offer an interface that can be called by way of a standardized protocol. Having a standard way of communication help for interacting between services whether from another service in the same application or one located across the internet [6].

Stateless is one of the key aspects of cloud-native and the thesis focus more on that aspect. Stateless cloud-native applications exist independent of stateful applications. When applications are stateless, it means that the applications do not have to remember any data. Because, requests have all data which is needed to be responded.

Another important thing is that the cloud-native applications are deployed on elastic, cloud infrastructure. With this type of an infrastructure, the applications can adjust their resources dynamically in case the load is increased or decreased [5].

## 2.3 Scalability

Scalability refers to the capability of a system where its infrastructure can be expanded to handle increased load without losing performance. In other words, it is a characteristic

of a system that presents the capacity of the infrastructure to meet a growing demand for use.

Scalability is one of the most valuable feature of cloud computing. One core benefit of scalability in the cloud is that it facilitates performance. A scalable system has the ability to handle the heavy workloads and network traffic.

For the companies, scalability makes cost-efficient system available for their business. Because, the system can scale up or down to meet the demands of growing business. It allows the systems to grow without making any expensive changes in the environment. This reduces the cost of resources and makes scalability in the cloud very cost effective. Therefore, while having applications working in the cloud, scalability becomes a necessity to manage increased demands [7].

## 2.4 Problems Cloud-Native Brings to NETCONF

In the NETCONF protocol, an SSH session is set up between manager and agent, and then the communication flows on the established SSH connection. Therefore, permanent connections are needed before any data can be transferred. The connections should be established from manager and kept open to receive the notifications from Agent. In case of large network, thousands or millions SSH sessions must be kept permanently open during NETCONF communication between managers and agents. For this reason, there is a conflict between the NETCONF protocol based on SSH session and cloud-native essentials.

Stateless is one of the key features of cloud-native applications. It means that when a request is sent, it has all inputs that are needed, and response will be returned based on the inputs provided. The application does not need to store either remember any data. The application could be deployed in a cloud environment with multiple instances, and the network traffic is spread towards the instances. If any of the instances are down, the other instances continue to handle the requests.

However, with the NETCONF protocol, there must be permanent sessions to make the communication successful. This way of working brings problems to the stateless manner. Additionally, scalability becomes an issue since each agent must have a permanent session to send a request. All these problems mean that more resources are needed on network management side; not being exactly cloud-native and scalable.

# 3. IMPLEMENTATION

This chapter goes through the implementation of the proof of concept application and its architecture. The chapter is structured as follows: section 3.1 provides software overview of the implementations. Section 3.2 describes the first implementation, which is the application of NETCONF Event Notifications, and section 3.3 also explains the second implementation, which is the application of NETCONF Event Notifications using Call Home feature.

Two different implementations were developed during the study of the thesis. In the first phase of the implementation, the idea was to create an application to see the behavior of working with NETCONF Event Notifications from network management perspective on a cloud-native environment. For that purpose, a simulator of network element and a network management system were implemented. A software application named ConfD, which is provided by Tail-f [8], was selected to simulate network elements. Because, the application supports the NETCONF protocol and it is important to have a simulator which must generate realistic behavior on TCP/IP and SSH layers. Because, the focus is to have many unique connections open and the impact of this on the manager-side. Two bash scripts were written to work with the application to simulate network element. The network management system was written in Java to communicate with the network element simulator. The implementation details are explained in section 3.2.

The aim of the second part of the implementation is to create an application to work with NETCONF Event Notifications within a NETCONF Call Home implementation on a cloud-native environment. The network element simulator and the network management system were written in Java in this implementation phase. The main difference here is that the network element starts the initiation of the NETCONF connection. The implementation details are explained in section 3.3.

## 3.1 Software Overview

The aim of the thesis is to investigate how to design and support the NETCONF from network management system perspective in cloud-native environment. Two proof of concept applications were implemented to analyze the study. Both implementations were designed with client-server architecture to accomplish the thesis.

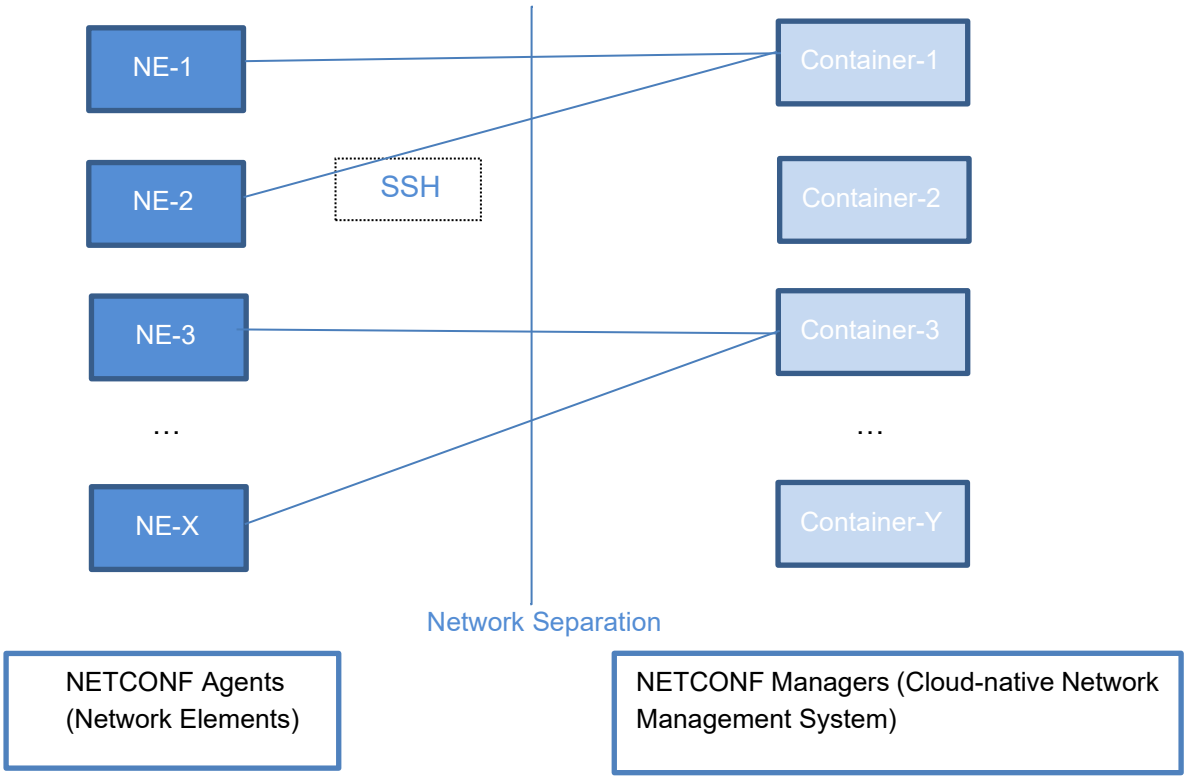


***Figure 9.*** *High-level view of the proof of concept architecture*

Figure 9 shows a high-level view of the overall architecture for the proof of concept. In the architectural design, the client and server-side roles are shared between network element and network management system. Since the aim of the thesis is mainly to investigate the behavior of network management system with NETCONF, only network management system is planned to deploy and run in cloud-native environment.

Java NETCONF Client (JNC) and Apache MINA SSHD Java libraries were used to develop the implementations on network management system side. JNC is integrated in the project to make NETCONF connections. On the other hand, Apache MINA SSHD makes asynchronous connections available. ConfD is benefited for the simulation of the network elements.

## 3.2 NETCONF Event Notification Application

This chapter describes the first implementation of proof of concept, which works with standard NETCONF Event Notifications. As mentioned in section 3.1, the proof of concept implementation was designed with client-server architecture. In the implementation

of NETCONF Event Notifications application, the server-side is network element as well as the client-side is network management system. In other words, in NETCONF terminology, network element is NETCONF agent and network management system is NETCONF manager.

### 3.2.1 **Agent Application**

The NETCONF agent application was decided to be implemented with the available software applications since the focus of the thesis is to study network management system behavior. After researching multiple options, ConfD [8] was selected since it supports NETCONF Event Notifications.

ConfD works with the NETCONF and RESTCONF protocols for providing unified, programmable management capabilities to network equipment providers and software vendors. ConfD promises reducing the time and resources needed to implement northbound network management applications and provide programmability to the network market.

Tail-f presents two versions of ConfD to be used by network equipment providers and software vendors: ConfD Premium and ConfD Basic. ConfD Premium is the paid version of the management agent software framework as well as ConfD Basic is the free version. They support NETCONF, customer support, CLI and some other features [8]. ConfD Basic was chosen to be used due to its NETCONF capability.

Basically, ConfD Basic is delivered as a tar file with documentation, and a few demo applications regarding the protocols which are supported. These applications give initial ideas regarding how to work with ConfD and guide the users to achieve implementation of software applications. Among the example applications, the NETCONF Event Notifications example application was selected to be used in the thesis.

The application acts as a network element which is NETCONF agent and can establish the NETCONF connections and sends the event notifications to a network management system. It emulates the NETCONF agent and listens the defined port to make NETCONF connections with the network management system.

The NETCONF Event Notifications example application is written in C. The application has a Makefile that supports clean, all, start, cli and the NETCONF operation commands. It has also a configuration file named confd.conf to adjust the values regarding the NETCONF protocol such as enabler and tracing settings, SSH IP and port to be listened by the application.

The application was modified to be used in the project. The idea is here to make configurable the payload size and number of event notifications per connection in the application. Hence, the application of NETCONF Event Notifications could be taken advantage by providing different values of these parameters during the test phase. These necessary changes needed to be added into ‘notifier_builtin_replay_store.c’, which has the operational functions and the main function of the application.

The file was customized so that the application could take the payload size and number of event notifications per connection as inputs. The payload size options per connection was defined as follows: 1 KB, 2 KB, 3 KB and 4 KB. 1 KB was also determined as default value in case of providing erroneous value for the payload size.

After the modifications have been done, two bash scripts were implemented to use the application of NETCONF Event Notifications in this phase. The first script named ‘run_simulator.sh’ has the following tasks:

1. Initialize environment setup of ConfD Basic.
2. Wait for the payload size and number of events per connection as inputs.
3. After getting the inputs, start the application of NETCONF Event Notifications.

Another script named ‘send_notification.sh’ has the following tasks:

1. Get the payload size and number of events per connection as well as repeat times as arguments.
2. Provide the payload size and number of events per connection to run_simulator.sh script.
3. Repeat triggering sending event notifications as many as repeat times argument in second.

First of all, the first script is run to initialize the agent, so that the network management system application can send request and reach the simulator. Once connection establishments have been done, the second script is run immediately afterwards. The event notifications are sent by initialization of the second script towards to the network management system.

### 3.2.2 **Manager Application**

A NETCONF manager application was implemented using Java programming language. The application reads environment variables to be able to have information of execution type and destination address information. In total, 7 parameters are provided in environment variables as follows: IP and port of destination, username and password to be used

to login as well as total number of sessions to be created and the name of event notification stream to subscribe.

Java NETCONF Client (JNC) is a Java library provided by Tail-f to implement NETCONF manager applications for making NETCONF connections with NETCONF agent. JNC library provides software capability to create network management systems for any NETCONF devices. It is widely used to manipulate configuration data as a means of NETCONF Java components. The NETCONF Java components are benefited via using Ganymed SSH-2 to communicate with the NETCONF agent.

Since it is stable and useful, JNC software library was selected to communicate with the NETCONF Event Notifications application which is the NETCONF agent application explained in section 3.2.1. Java NETCONF Client is open source and can be found from [9].

A class was created regarding to handling the NETCONF communication and event notifications via Java NETCONF Client library in this part of the thesis. The first step in here is to initialize a fixed size thread pool executor for a better performance and resource utilization by limiting the maximum number of threads. There are also classes for keeping ongoing NETCONF sessions and measurements of number of event notifications, payload sizes and sessions during testing phases.

The environment variables are given to provide the required destination information of NETCONF agent. Threads were executed as many total sessions as in order to send a connection request and event notification subscription request for NETCONF agent. The prior initialized threads were used from the thread pool to produce the NETCONF requests towards the agent. In this way, the NETCONF manager application triggers the connection and subscription requests to the NETCONF agent.

The JNC java class has the following tasks:

1. Create threads from a fixed size thread pool.
2. Trigger executions of the threads to make NETCONF connection and subscription of NETCONF Event Notifications stream.
3. Wait until all NETCONF sessions and subscriptions are established successfully.
4. Read NETCONF Event Notifications coming from NETCONF agent.

An SSH connection is established between the agent and the manager. Every session is identified by a unique session ID. Once the SSH connection has been established, a request is sent to open a NETCONF connection over the established SSH connection. A NETCONF session is created and each session corresponds to one SSH channel towards the agent. The created every session is stored here to be modified afterwards.

A synchronized list Java component is used to store SSH sessions since it is thread-safe solution.

Once the NETCONF session has been established successfully, an event notification subscription request is sent to the agent. The agent and the manager must support the event notification capability since this capability makes it possible to receive and send event notifications specified in a subscription. By sending a subscription request, the manager implies that it wants to receive event notifications specified by name in the subscription request.

Each established SSH session is continuously checked to read incoming notifications. When an event notification arrives, the event notification counter is increased, and its payload size is added in sum of payload counter. Java synchronized block is benefited to work with measurement logic since it is a thread-safe way.

## 3.3 NETCONF Event Notification Using Call Home Feature

This implementation of proof of concept, which supports NETCONF Event Notifications using Call Home feature is described in this chapter. NETCONF agent and manager applications are presented in section 3.3.1 and section 3.3.2, respectively.

NETCONF Event Notification using Call Home feature was used in the second implementation. The main business need is that the initial request needs to be sent by NETCONF agent. The operators ask to have agents initiate management connections, believing it is easier to secure one open port in the data center than to have an open port on each agent in the network [4]. More details were presented in section 2.1.2.

The NETCONF protocol communicates over a secure connection, which must be permanently established between the sides. So, constant connections need to be opened to get messages. However, this behavior is not suitable for cloud-native applications. Cloud-native applications must be stateless and scalable in the cloud environment. Being stateless makes applications to be scalable without a problem since sessions do not need to be continued. Therefore, constant established connections bring problems to cloud-native applications. The problems were explained in section 2.4.

For solving the problem, NETCONF Call Home is used to initialize the NETCONF communication by the agent. Once the agent sends the events to the manager, the agent finishes the session and closes the connection. In the second implementation, the closing the session after sending the event by the agent is the most important part of the communication flow. In case the agent needs to send events to the manager again, the agent triggers the communication flow and eventually ends the connection. By having

such communication, the connections are not kept open permanently in cloud environment. The design is more convenient for a cloud-native application.

The applications were planned and implemented base on the flow below.

Flow of communication,

1. Start *Manager*, listens for TCP connection at 4334 recommended port
2. Start *Agent*, sends connection request to *Manager*
3. *Manager* accepts the TCP connection and initiates SSH connection to *Agent*
4. SSH connection established after cipher negotiation and key exchange completed
5. *Manager* starts NETCONF session and sends hello capabilities and notification subscription request to *Agent*
6. *Agent* sends hello capabilities after the NETCONF session established
7. *Agent* sends a notification to *Manager*
8. *Agent closes the connection*

Apache MINA SSHD library was used in the applications since it provides asynchronous communication between the sides. It is a Java library provided by The Apache Software Foundation. It supports the SSH protocols on both the client and server side with a scalable and high performance asynchronous IO library. Apache MINA SSHD is open source and can be found from [10].

### 3.3.1 **Agent Application**

The manager application, which is mentioned in section 3.2.2, was used to start the implementation of the agent application. For this reason, in the second implementation, the NETCONF agent application was developed using Java programming language. The inputs for the application are hostname, port of the manager, username and password for the SSH connection. These are read from environment variables of the application. Providing execution type to environment variables as ‘callhome_server’ starts the application.

First, an SSH server is needed to be created for receiving the initial SSH connections to the agent from the manager. SSHServer class in the Apache library was used to initialize SSH server with default settings. Password and key were provided to the server for the authentication. Additionally, the agent application must communicate with the NETCONF protocol. Thereupon, a subsystem was set to the SSH server. The subsystem class implements SubsystemFactory class from the Apache library and has two methods which provide xml file of the ‘hello message’, and subsystem name, which is ‘netconf’. The SSH server was started after the initialization was done.

By using the connection information of the SSH server, a TCP connection was initialized and started towards the manager with the hostname and port provided in environment variables. After the TCP connection was started, TCP client awaits a response from the manager. Once the TCP connection was established, the SSH server is ready to serve the SSH connection and the NETCONF protocol.

NetconfSubSystem class, which is set to the SSH server, provides the ability to send hello message and event notification. After sending the both messages to the manager application, the agent application closes the all connections and then the application.

### 3.3.2 **Manager Application**

As used in the agent application, the manager application was implemented with the same way using Java programming language. This time, ‘callhome_client’ needs to be given into execution type of environment variables to start the manager application. The inputs for the application are port, username, password. These are read from environment variables of the application.

An SSH client was initialized using Apache library and started with default settings of the library. A session listener was created and set to the SSH client to be invoked once the a new SSH session was just created. Username and password were provided into the session for the authentication in the listener.

Once the SSH session has been created, a new listener was set for the asynchronous authentication process. SSHFutureListener class was benefited from Apache library to listen SSH operations. Once the authentication of the SSH session has been completed, a subsystem channel was created named ‘netconf’. The subsystem is needed to establish the NETCONF connection. Once the session has been opened, a subscription request is read from the xml file and sent to the agent.

By using the SSH client session information, a TCP server was prepared to receive TCP connections from the agent application.

# 4. EVALUATION AND RESULTS

This chapter focuses on the evaluation of the applications described in the previous section 3. It gathers findings of using the NETCONF protocol in cloud-native environment. The evaluation has been done based on the research questions of the thesis:

- What are the actual problems and challenges of NETCONF in cloud-native environment?
- How should NETCONF Event Notifications in cloud-native environment be supported from network management system perspective in case of large networks?
- What are needs and limitations in the environment on scalability?

To answer to the research questions, the applications were deployed on cloud environment and their performances were measured. The measurements include event notification counts and payload size during the evaluation.

The first section 4.1 introduces the test environment and test cases. In sections 4.2 and 4.3 explain the evaluations as well as the results and findings are presented.

## 4.1 Test Environment and Test Cases

Since the main idea is to see the behavior from network management system perspective during the evaluations, the manager applications were deployed in a cloud environment which is a Platform as a Service (PaaS). It provides easiness to the developers for focusing only deploying their applications and managing them. The PaaS cloud environment gets images of the developers' applications and their configuration data, containerizes the applications and runs them into a container orchestrator. The platform provides a user interface for organizing applications, resources and configurations. The cloud environment has Kubernetes as container-orchestration tool for automating application deployment, scaling, and management and Docker for containers. The environment resources were 32 GB RAM, 20 CPU and 2GB storage.

There are two implementations in the thesis as described in section 3. The first one is the application of standard NETCONF Event Notifications. The second one is the application of NETCONF Event Notifications using Call Home feature. The test cases are to run the both applications individually and get the measurements of the durations of communication flows. The main difference in these two cases is initialization of the communication flow. In the first one, which is with the application of standard NETCONF Event

Notifications, the manager initializes the session. On the other hand, the agent initializes the session in the second one, which is with the application of NETCONF Event Notifications using Call Home feature.

## 4.2 Measurements of The First Test Case

The first test case is to run the application of standard NETCONF Event Notifications, which is described in section 3.2. Since the aim of the thesis is to see the behavior of the manager application during the communication flow, the measurements are done on the manager side.

In the first implementation, as presented in section 3.2.1, the agent application is ConfD, which is a software agent and helps to manage network elements. VirtualBox [11] is selected to run the agent application when getting measurements of the application of standard NETCONF Event Notifications. Because, VirtualBox is a powerful virtualization product and helps users to load multiple guest operation systems under a single host operation system. Each guest operation can be started, paused and stopped independently within its own virtual machine. The host and guest operation systems can be configured and so that they can communicate with each other. VirtualBox is free and open source developed by Oracle Corporation.

A virtual machine was created using VirtualBox software. Ubuntu 64x [12] was installed into the virtual machine as the guest operation system. The resources were set to 8MB RAM and 1 CPU. This virtual machine was used for both development and testing environment for the agent implementation. The manager application was also deployed into the cloud environment to be run in the test scenario.

As mentioned in section 3.2.1, two bash scripts were implemented to run the ConfD simulator and send notifications, respectively. The test scenario starts by running the simulator to initialize the agent. Then the manager application is started, and it triggers the connection establishment. As explained in section 2.1.1, first, SSH session and then NETCONF session is established according to NETCONF specifications, which is described in RFC-6241 [1]. After the establishment of the connections, the events are generated in the agent side. Receiving the events on the manager side is observed.

The agent application gets ‘number of connections’, ‘payload size’, and ‘number of events per second’ as inputs, so that they are configurable during the test scenarios. The evaluation is performed with changing the parameters and observing the results.

Table 2 below presents the results of the evaluation with changing number of connections while keeping the same payload and number of events. The measurements are

done by observing the notification events count and payload size within 30 seconds period of time on the manager side.

***Table 2.*** *Results based on changing connections count with 30 seconds test duration*

| Connections Count | Payload size (KB) | Events Count per Second | RESULTS (each row in 30 sec) | |
|---|---|---|---|---|
| | | | Events Count | Payload Size (MB) |
| 32 | 1 | 1 | **960** | **~1** |
| 64 | 1 | 1 | **1920** | **~2** |
| 128 | 1 | 1 | **3840** | **~4** |
| 256 | 1 | 1 | **7680** | **~8** |
| 512 | 1 | 1 | **15360** | **~16** |
| 1024 | 1 | 1 | **30720** | **~32** |

The analysis of the results in Table 2 says that when the number of connections is increased, the events reach the manager side within 30 seconds of time period in the test environment for each round. The event counts indirectly went up in the evaluation, however the measurements on the manager side were stable and regular.

On the other hand, another test round was done by keeping the connections count and payload size constant while increasing the events count. The measurements of the notification events count and payload size on the manager side are noted within 30 seconds of time period. The results are shown in Table 3.

***Table 3.*** *Results with changing events count per second within 30 seconds test duration*

| Connections Count | Payload size (KB) | Events Count per Second | RESULTS (each row in 30 sec) | |
|---|---|---|---|---|
| | | | Events Count | Payload Size (MB) |
| 4 | 1 | 100 | **12000** | **~12.5** |
| 4 | 1 | 1000 | **77178** | **~80.5** |
| | | | **42822** | **~44** |
| 4 | 1 | 2000 | **78387** | **~81.7** |
| | | | **77928** | **~81.2** |
| | | | **73984** | **~77.1** |
| | | | **9701** | **~9.2** |
| 8 | 1 | 100 | **24000** | **~25** |
| 8 | 1 | 1000 | **61853** | **~64.5** |
| | | | **100418** | **~104.7** |

| | | | | |
|---|---|---|---|---|
| | | | **77729** | **~81.2** |
| 8 | 1 | 2000 | **130901** | **~136.5** |
| | | | **132708** | **~138.4** |
| | | | **139633** | **~145.6** |
| | | | **76758** | **~80** |
| 16 | 1 | 100 | **48000** | **~50** |
| 16 | 1 | 1000 | **162558** | **~169.5** |
| | | | **172052** | **~179.4** |
| | | | **145390** | **~152** |
| 16 | 1 | 2000 | **167241** | **~174.4** |
| | | | **164325** | **~171.3** |
| | | | **162679** | **~169.6** |
| | | | **164145** | **~171.2** |
| | | | **165309** | **~172.1** |
| | | | **136301** | **~143.1** |

In Table 3, on the manager side, each result was measured in 30 seconds period of time and noted. According to the results, the maximum throughput is up to 5 MB payload/second and up to 5000 events/second. Additionally, CPU level was very high during the evaluation. As a consequence, data transfer limited by these conditions in given test environment.

## 4.3 Measurements of The Second Test Case

The second test case was done by running the application of NETCONF Event Notifications using Call Home feature, which is described in section 3.3. For the test, the manager application was deployed into the cloud environment to be observed the measurements during the evaluation. The agent application of NETCONF Event Notifications using Call Home feature was also deployed into the same virtual machine which was described in the previous section and used in the first test case.

The communication flow of the test case is given below:

- Manager is initialized and ready to receive request from Agent.
- Agent is initialized and ready to send request from Manager.
- Agent “calls home” (single connection) to Manager.
- Manager performs all mandatory handshakes and verifications according to the Call Home specification.
- Manager subscribes for events.

- Agent sends event notification.
- Agent closes the connection.

During the implementation of the application, the flow was designed in the given order and the connection was closed at the end of the flow. Because, NETCONF Call Home feature is considered to be used to deal with permanent connections in the cloud environment. In NETCONF Call Home flow, the agent starts the communication flow contrary to the flow of standard NETCONF protocol and sends its event notification. However, the NETCONF protocol needs permanent SSH connection. So, the session is closed by the agent itself after sending the notification event. By closing the connection, the design is considered to be more suitable for a stateless application in cloud environment.

Table 4 and Table 5 show the results of the evaluation with changing number of connections while keeping the same payload and number of events. The measurements are noted within 30 seconds period of time.

***Table 4.*** *Results with 1 event counts within 30 seconds test duration*

| Connections Count | Payload size of an event (KB) | Events Count per Connection | RESULTS (each row in 30 sec) | |
|---|---|---|---|---|
| | | | Events Count | Payload Size (MB) |
| 1K | ~0.3 | 1 hello request + 1 event | **1734** | **~3.1** |
| | | | **266** | **~0.5** |
| 2K | ~0.3 | 1 hello request + 1 event | **1660** | **~2.9** |
| | | | **2340** | **~4.2** |
| 5K | ~0.3 | 1 hello request + 1 event | **2298** | **~4.1** |
| | | | **2004** | **~3.6** |
| | | | **2314** | **~4.1** |
| | | | **2164** | **~3.8** |
| | | | **1220** | **~2.1** |

***Table 5.*** *Results with 5 event counts within 30 seconds test duration*

| Connections Count | Payload size of an event (KB) | Events Count per Connec-tion | RESULTS (each row in 30 sec) | |
|---|---|---|---|---|
| | | | Events Count | Payload Size (MB) |
| 1K | ~0.3 | 1 hello request + 5 event | **5065** | **~3.1** |
| | | | **935** | **~0.5** |
| 2K | ~0.3 | 1 hello request + 5 event | **5460** | **~4.2** |
| | | | **6318** | **~4.9** |
| | | | **222** | **~0.1** |
| 5K | ~0.3 | 1 hello request + 5 event | **4956** | **~3.8** |
| | | | **8448** | **~6.5** |
| | | | **7632** | **~5.9** |
| | | | **7872** | **~6.1** |
| | | | **1092** | **~0.8** |

Each connection has 1 hello message to initialize the NETCONF connection. Therefore, number of events have an additional event for each connection. Size of the hello request is around 3.3KB while an event size is around 0.3KB.

The measurements show that the event throughput was limited by mostly CPU. More resources would be needed to work with, compared to the application of standard NETCONF Event Notifications.

***Table 6.*** *Results with 1000K events within 30 seconds test duration*

| Connections Count | Payload size of an event (KB) | Events Count per Connec-tion | RESULTS (each row in 30 sec) | |
|---|---|---|---|---|
| | | | Events Count | Payload Size (MB) |
| 1 | ~0.3 | 1 hello request + 1000K event | **115513** | **~31.3** |
| | | | **89208** | **~24.1** |
| | | | **74007** | **~20.1** |
| | | | **94693** | **~25.6** |
| | | | **76833** | **~20.8** |
| | | | **93779** | **~25.4** |
| | | | **90108** | **~24.4** |

| | | | | |
|---|---|---|---|---|
| | | | **108051** | **~29.2** |
| | | | **97097** | **~26.3** |
| | | | **112267** | **~30.4** |
| | | | **48445** | **~13.1** |

Each result was measured in 30 seconds period of time and showed on Table 4, 5 and 6. According to the results, the maximum throughput is up to 3800 events/second and up to 1 MB payload/second.

During the testing of the application of NETCONF Event Notifications using Call Home feature, different security providers were used. Based on the observations, one finding was that authentication was taking only 100 milliseconds, but more time was taken in SSH mac/cipher negotiation and key exchanges based on the security provider.

***Table 7.*** *Time comparison of communication initialization with different security providers*

| **Connections Count** | **TCP Connection (ms)** | **SSH Connection (key exchange)** | | **Password based authentication (ms)** | **Netconf Session (ms)** | **Total time taken to receive notification (ms)** |
|---|---|---|---|---|---|---|
| | | Java Inbuilt Security Provider (ms) | Bouncy-Castle Security Provider (ms) | | | |
| **1** | 30 | 300 | - | 100 | 100 | **700** |
| **1** | 40 | - | 1800 | 100 | 200 | **2300** |

According to the numbers on Table 7, the total duration which was taken from TCP connection till receiving notification is 700ms using Java Inbuilt Security Provider and 2300ms using BouncyCastle Security Provider, which comes default with Apache Mina SSHD library. The time, which is taken to establish SSH connection, drastically vary based on security provider implementation. The measurements show that the communication flow is relatively long. The connection establishment duration might take long depends on the security provider being used.

# 5. CONCLUSIONS

In this thesis a proof of concept for cloud-native Network Configuration Protocol was studied and developed. By way of the proof of concept, the evaluations were made, and the following research questions of the thesis were searched for an answer.

- What are the actual problems and challenges of NETCONF in cloud-native environment?
- How should NETCONF Event Notifications in cloud-native environment be supported from network management system perspective in case of large networks?
- What are needs and limitations in the environment on scalability?

This research aimed to identify the problems and challenges for realization of Network Configuration Protocol in cloud-native environment. Based on the implementations and analysis, which are presented in the thesis, it can be concluded that terminating the established NETCONF sessions after sending messages between network elements and network management system is an operable solution when implementing realization of Network Configuration Protocol in cloud-native environment. The results indicate that using Network Configuration Protocol Call Home for cloud-native environment is more secure and manageable solution to large networks security and maintenance point of view.

Both implementations of standard Network Configuration Protocol and Network Configuration Protocol Call Home were covered in cloud-native environment. By analyzing changing number of connections and events per connection in the both implementations, this thesis shows how throughput and CPU can limit the communication flow when working with NETCONF in cloud-native environment.

Moreover, a topic was discussed about how important security providers selection during an implementation. Based on the analysis of security providers, it can be stated that different type of security providers must be considered since it affects authentication time during secure connection establishment.

As future work, there could be more discussion and analysis about who terminates the session and what the reasons for doing it are. The network management system might close the session as opposed to what was done in this thesis. Furthermore, what if there are still notifications to be sent by the network element when the management system

closes the sessions. This discussion seems important and could be studied to extend the research.

Another plan for later work should be doing a test setup to analyze high availability of the applications in case of being crashed or killed, so that any events are missing. In addition, the thesis could be improved with using TLS for the secure connection instead of SSH.

# 6. REFERENCES


[1] Internet Engineering Task Force (IETF), RFC 6241 - Network Configuration Protocol (NETCONF), IETF, 2011. Available: https://tools.ietf.org/html/rfc6241

[2] Christos Rizos, Why use NETCONF/YANG when you can use SNMP and CLI?, SNMP center, 2016. Available: https://www.snmpcenter.com/why-use-netconf/

[3] Internet Engineering Task Force (IETF), RFC 5277 - NETCONF Event Notifications, IETF, 2008. Available: https://tools.ietf.org/html/rfc5277

[4] Internet Engineering Task Force (IETF), RFC 8071 - NETCONF Call Home and RESTCONF Call Home, IETF, 2017. Available: https://tools.ietf.org/html/rfc8071

[5] Janakiram MSV, 10 Key Attributes of Cloud-Native Applications, The New Stack, 2018. Available: https://thenewstack.io/10-key-attributes-of-cloud-native-applications/

[6] Bernard Golden, 5 Critical Elements to Building Next-Generation Cloud-Native Apps, TechBeacon. Available: https://techbeacon.com/app-dev-testing/5-critical-elements-building-next-generation-cloud-native-apps

[7] What is Scalability in the Cloud, Stonefly, 2018. Available: https://stonefly.com/blog/what-is-scalability-in-the-cloud

[8] Tail-f a Cisco Company, ConfD Basic Tail-f Systems, 2019. Available: https://www.tail-f.com/confd-basic/

[9] Tail-f Systems, JNC, 2019. Available: https://github.com/tail-f-systems/JNC

[10] The Apache Software Foundation, Apache MINA SSHD, 2019. Available: https://github.com/apache/mina-sshd

[11] Oracle VM VirtualBox, 2019. Available: https://www.virtualbox.org/

[12] Ubuntu, 2019, Available: https://ubuntu.com/