NTNU

Norwegian University of Science and Technology

Software Defined Networking for FUSIION (Integrated Hybrid Optical) Networks)

Weldmicheal Berhanu Hailu

Master of Telematics - Communication Networks and Networked Services

Submission date:	June 2016
Supervisor:	Steinar Bjørnstad, ITEM
Co-supervisor:	Raimena Veisllari, Transpacket

Norwegian University of Science and Technology
Department of Telematics

Title:	Software Defined Networking for FUSION (Integrated Hybrid Optical) Networks
Student:	Weldmicheal Berhanu Hailu

Problem description:

Software Defined Networking (SDN) is a new networking paradigm that separates the network control plane from the packet forwarding plane. A logically centralized controller that has a global network view is responsible for all the control decisions and it communicates with the network-wide distributed forwarding elements via standardized interfaces. In this context, Netconf Configuration Protocol (NETCONF) has been proposed as a southbound protocol that supports programmability of network functions and protocols in an SDN network. An SDN controller is a server that has a global view of the network and runs control applications.

Integrated hybrid optical networks (IHON) are networks that integrate both packet and circuit switching in the same network nodes on the same links. Fusion H1 node is an Ethernet switch from Transpacket Company which is built based on NETCONF protocol implementing the IHON principles as its core functionality. The communication between the controller and the nodes is carried through the NETCONF protocol. While SDN for packet-switching is being widely researched and deployed (e.g. Google, Huawei, Cisco etc.), circuit-switching in the optical domain is still under review and extension. The goal of the project is to enable an SDN controlled integrated hybrid optical network. The work includes choosing compatible SDN controller and establishing connection with the H1 nodes via NETCONF protocol. As a result, perform a set of NETCONF operations on the nodes from the controller.

Responsible professor:	Steinar Bjørnstad, ITEM, NTNU
Supervisor:	Raimena Veisllari, Transpacket

Abstract

Nowadays large transmission capacity is offered through deployment of optical fiber. At the same time, new design approaches have emerged, like the Integrated Hybrid Optical Network (IHON) architecture which combines the advantages of both circuit and packet switching while diminishing their disadvantages. IHON offers both guaranteed Quality of Service (QoS) and efficient utilization of the transmission capacity. Transpacket is a startup company that has developed Fusion H1, a prototype node based on IHON principles which uses Ethernet for data plane while employing Yet Another Next Generation (YANG) data modeling language and Network Configuration Protocol (NETCONF) protocol for control plane. In traditional networks, control and data planes are coupled within the network hardware which leads to expensive network management and administration. In order to lower costs and increase the network flexibility in provisioning new services and adapting to the emerging range of applications, an Software Defined Network (SDN) is believed to be the promising solution. SDN dissociates the control from the underlying network devices, and centralizes the intelligence and state of the network at the controller to allow network automation.

In this thesis work is demonstrated for the first time an SDN controlled integrated hybrid optical network through emulation and also experiment with Fusion H1 nodes in the Uninett lab. Furthermore, this report covers brief background to IHON in general and Fusion H1 nodes in particular. Additionally, an SDN platform, and evaluation of NETCONF and OpenFlow (OF) southbound protocols is included. We also discuss both the emulation testbed and lab experiment.

OpenDaylight (ODL) platform is chosen as the testbed to emulate NETCONF-based SDN framework for IHON. We described the core procedures to simulate the IHON nodes, and establish successful communication with the chosen SDN testbed. First, we developed YANG model for the configuration of IHON nodes. As a result, configuration, management and monitoring of the network is achieved using the controller. Moreover, the same ODL testbed which is installed in a stationary server at the Telematics department is used to experiment with the H1 network at Uninett labs. As a result, we succeeded to connect each node with the testbed platform, and retrieved their configuration. Lastly, possible future work is presented as a continuation of this thesis.

Acknowledgement

My thesis has been challenging, but I have enjoyed every moment of my journey: It is the most fruitful period of my academic work. As SDN controlled IHON is a newly emerging network paradigm, I have been energetic and stimulated in my work. I forward my special thanks to my professor, Steinar Bjørnstad, for his support throughout the thesis period which boosted my determination. Moreover, my supervisor, Raimena Veisllari, has played unceasing role in achieving my goals. Her guidance is priceless, and her encouragement in times of difficulties helped me to march forward. By combining my work with their efforts together with the excellent facilities of NTNU specifically the Telematics department and Uninett, I have accomplished my thesis objective. Lastly, I would like to thank for all Telematics community, my friends, and my family for their daily cooperation and encouragements.

Contents

List of Figures **ix**

List of Tables **xi**

List of Algorithms **xiii**

1 Introduction **1**

1.1 Motivation 1

1.2 Objective and Methodology 3

1.3 Project Structure 4

2 Integrated Hybrid Optical Network (IHON) **5**

2.1 Classes of Hybrid networks 5

2.2 Ethernet Streaming in Fusion H1 nodes 6

3 Software Defined Networking (SDN) **9**

3.1 SDN Architecture and Working Principles 9

3.2 Network Configuration Protocol (NETCONF) 10

3.2.1 NETCONF Layering Model 10

3.2.2 Running NETCONF over SSH Session 12

3.2.3 NETCONF Datastores 14

3.2.4 Capability Exchange 15

3.2.5 Remote Procedure Call (RPC) Model 18

3.2.6 NETCONF Protocol Operations 21

3.3 YANG Modeling Language 22

4 OpenDaylight Platform **27**

4.1 Introduction to ODL 27

4.2 Why ODL 28

4.3 ODL Architecture 29

4.3.1 ODL Controller 30

4.3.2 MD-SAL Basics 32

4.4 RESTCONF 33

4.4.1 Supported Operations 33
4.4.2 RESTCONF Tools 35

5 SDN framework for IHON 37
5.1 Evaluating NETCONF- and OF-based SDN for IHON nodes 37
5.2 SDN/NETCONF setup for IHON 38
5.2.1 IHON nodes that Support NETCONF monitoring 39
5.2.2 IHON nodes that do not support NETCONF monitoring, but list YANG models 41
5.2.3 IHON nodes that neither support NETCONF monitoring nor list YANG models 41

6 SDN and Emulated IHON Testbed Results 43
6.1 SDN framework for IHON 43
6.1.1 Step1: Simulating IHON nodes with Netconf Testtool 45
6.1.2 Step2: Developing YANG Schema for IHON Nodes 47
6.1.3 Step3: Starting the IHON Nodes 50
6.1.4 Step4: Connecting the IHON Nodes with ODL controller . . 53
6.2 Management of IHON Nodes with the ODL Controller 55

7 SDN and Fusion Experiment Results 61
7.1 H1 Network Diagram Setup 61
7.2 Deploying ODL platform for the H1 network 61
7.3 Management of H1 Nodes 64

8 Conclusion and Future Work 67
8.1 Conclusion 67
8.2 Future Work 68

References 69

Appendices

A OpenDayLight Installation and Running 73

B IHON node YANG Schema and Configuration 75
B.1 IHON YANG Schema 75
B.2 NodeA Configuration 77
B.3 NodeB Configuration 79
B.4 NodeC Configuration 82
B.5 Retrieved Configuration of NodeA 84
B.6 Retrieved Configuration of NodeB 86
B.7 Retrieved Configuration of NodeC 89

C Creating Netconf-connector **93**
C.1 Creating New Netconf-connector 93
C.2 Updating netconf-connector 95
C.3 Capability of Netconf Testtool 96

D Capabilities of Emulated IHON Nodes **99**
D.1 NodeA Capabilities 99
D.2 NodeB Capabilities 100
D.3 NodeC Capabilities 101
D.4 Operational data 102

E Configuration of H1 nodes at Uninett **105**
E.1 N1 Capability 105
E.2 N1 Configuration 111
E.3 N2 Configuration 116
E.4 N3 Configuration 121

List of Figures

2.1 Fusion H1 node is composed of ten 1GE client interfaces and two 10 GE line interfaces. The transmission illustrated in this figure is only unidirectional (left-to-right direction). Moreover, the 10GE interfaces transports either single Guarented Service Transport (GST) stream or carries five 1GE GST streams. On the other hand, the 1GEs are used for local add/drop. GST gap detector measures the exact free time gap which allows the Statistical Multiplexing (SM) packet scheduler to select and insert a suitable SM packet from the queues. Figure taken from [VBB13]. 7

3.1 Conceptual NETCONF logical layers, and mapping of a sample NETCONF operation to each layer, adopted from [EBS11]. 11

3.2 NETCONF session, showing client and server transaction model. After establishing connection, both peers exchange their capabilities using hello messages. After exchanging their capabilities, the client sends requests to the server as a Remote Procedure Call (RPC) models. The server, then, responses for each request. Lastly, the server initiates session closing, and the server sends confirmation and terminates the session. 13

3.3 Datastores of NETCONF protocol: a) *<running>*: Edits are saved in Non-Volatile (NV)-storage automatically, b) *<startup>*: Edits are saved manually, and c) *<candidate>*: Edits are saved automatically, adopted from [Cen]. 15

4.1 Architecture of Beryllium ODL platform, taken from [ODL16]. 29

4.2 Simplified diagram of ODL platform with two sample applications, adopted from [ODL16]. 30

5.1 SDN framework based on NETCONF protocol. The control plane consists of an SDN controller with built-in NETCONF client, and manges the IHON nodes in the data plane via NETCONF southbound plug-in. . . . 39

6.1 SDN setup based on NETCONF protocol with three IHON nodes in the data plane. Only unidirectional (left-to-right) Ethernet streams are considered. Red and blue correspond to SM and GST streams, respectively. IHON nodes are first connected to the ODL controller (indicated as flow 1). Flow 2 indicates management of the nodes from the controller. . . . 44
6.2 Establishing NETCONF session between client and server. In this case, the client is the ODL controller while the server is the IHON node. After establishing connection, both peers exchange their capabilities using hello messages. Afterwards, NETCONF operations are executed as RPC models, and the sessions is closed from the controller. 46
6.3 Network connection of the IHON nodes in the data plane. Each node transports GST and SM streams from different connections which are depicted as blue and red, respectively. Moreover, only left-to-right flow is considered. 48
6.4 Log output for the simulated IHON nodes from the Beryllium karaf distribution which shows successful starting of the nodes at specific ports. 53
6.5 Log output from the Beryllium which indicates successful initialization of netconf-connector for NodeA. 55
6.6 Log output from the Beryllium which indicates successful initialization of netconf-connector for NodeB. 56
6.7 Log output from the Beryllium which indicates successful initialization of netconf-connector for NodeC. 56

7.1 Network setup diagram at Uninett with three H1 nodes and Spirent SPT-2000 packet generator. GST traffic are depicted in green while SM in red, and their corresponding paths are depicted at the right bottom side. . . 62
7.2 Log output from the Beryllium controller which shows successful starting of netconf-connector for N1, N2, and N3. 65
7.3 Captured traffic with Wireshark between the controller and N1 node which shows the connection establishment, and data delivering with the PSH and ACK flags. The IP address of the N1 node is changed for security issues. 65
7.4 Captured traffic between the controller and N1 node; it specifically shows the closing of the ongoing session using FIN, and ACK flags. 65
7.5 Flow graph between ODL controller and N1 node 66

List of Tables

3.1 NETCONF operations along with their descriptions, adopted from [EBS11]. . . . 24

4.1 ODL releases [ODLe]: Hydrogen, Helium, Lithium, Beryllium, and Boron along with their features. . . . 28
4.2 Basic RESTCONF operations together with their formats and purposes, adopted from [ODLb]. . . . 34

6.1 Connections in the network given by Virtual Local Area Network (VLAN) source and destination addresses. . . . 49
6.2 Configuration plan for IHON nodes in the SDN setup. Each node's interface transports either SM or GST Ethernet stream. Moreover, each interface transports either single or multiple VLANs. "-" indicates that the interface is free. . . . 50
6.3 Internet Protocol (IP) address and port number of the IHON nodes in the SDN setup . . . 54

7.1 Connection set up of N1, N2, and N3; illustrates the stream types and VLANs in each node. . . . 63
7.2 IP address and port number for the H1 nodes in the network setup. The Ip address, username, and password of the H1 nodes are changed for security reasons. . . . 63

List of Algorithms

3.1 Capability format, adopted from [Cen] 15
3.2 An example of server capabilities, adopted from [EBS11]. 18
3.3 An example of client capabilities, adopted from [EBS11]. 18
3.4 Sample method invoking in <rpc>, adopted from [EBS11]. 19
3.5 Sample method invoking in <rpc> and <rpc-reply> format, adopted from [EBS11]. 20
3.6 Sample <rpc-error> format, adopted from [EBS11]. 20
3.7 Sample <ok> element in <rpc-reply> message, adopted from [EBS11]. 20
3.8 Sample <get-config> operation, adopted from [EBS11]. 22
3.9 Sample <edit-config> operation, adopted from [EBS11]. 23
3.10 YANG schema, e.g, interface schema of a router 25
3.11 NETCONF Extensible Markup Language (XML) representation for the YANG given in Source code 3.10. 26
5.1 Creating Netconf-connector for node 40
5.2 Modification of netconf-connector configuration for a node that does not list its YANG models nor supports netconf monitoring [Com16c]. 42
6.1 YANG schema of IHON node interface 51
6.2 YANG schema for listing the VLANs within IHON node 52
6.3 Default configuration data of IHON nodes (an empty data container) 57

Chapter 1 Introduction

1.1 Motivation

Nowadays, fiber deployment in networks is dynamically growing which implies that the transmission capacity is increasing to a level beyond the electronic switch processing capacity [GKB+06]. To overcome the switching bottlenecks and slow processing capacity of electronics, much research have been carried out to introduce fast and efficient switching solutions with fiber switching technologies [BNO+05], such as wavelength, and all-optical switching. Wavelength switching has slow adaptation for bursty traffic. As a result, it is less applicable since the bulk capacity of the fiber is underutilized with this switching technique. Even though all-optical switching is not practically deployed yet, it is believed to optimize the utilization while maintaining the advantages of switching in the optic domain and avoiding electronics [GKB+06]. So, researchers come up with a new adaptive solution by combining the advantages of both optical circuit and packet switching while minimizing their disadvantages. This solution is called hybrid optical architecture [GKB+06], and offers efficient performance and reduces the overall cost [GKB+06]. An Integrated Hybrid Optical Network (IHON) is a hybrid type of architecture which completely integrates both packet and circuit switched traffic, and transmits them simultaneously in a single wavelength. The circuit switched traffic (known as Guaranteed Service Transport (GST) stream) is assigned an exclusive lightpath, thus has an absolute priority. On the other hand, the packet switched traffic (known as Statistically Multiplexed (SM) stream) has best effort priority. Even though the lightpath is exclusively assigned for the GST traffic, the transmission capacity is not always fully utilized. By inserting SM traffic in the left over gaps between GST traffic with effective management, the IHON enhances the utilization of transmission capacity while offering guaranteed Quality of Service (QoS) for the absolute priority traffic.

Moreover, the evolution of networking technology is slow compared with other technologies of communication systems, e.g. application layer systems [JMD14]. Multiple set of vendors manufacture different set of routers and switches in accordance with their own designs to operate in a closed and proprietary manner. As a result, networking innovation was hindered, and did not evolve as expected. Hence, integration of new hardware technologies and services with the existing networks increases both Operational Expense (OPEX) and Capital Expense (CAPEX). Since control and data planes are tightly coupled in the architecture of the networks hardware, management and administration is very expensive in traditional networks. Hence, such network scenario is known as "inside the box" paradigm. To overcome these limitation, various industrial and network research communities cooperated to modify the traditional network and bring solutions.

Among others, Software Defined Networking (SDN) is the most popular scenario aimed to reshape the programmable network proposals and control-data plane separations projects [FRZ14]. It focuses on bringing the network "out of the box" paradigm. SDN objective is to separate the control and data plane thereby centralizing the intelligence of the network and its state at the controller. As a result, the network elements in the underlying infrastructure are abstracted from the applications [ONF13b]. Moreover, SDN employs standardized Application Programming Interfaces (APIs) (for example, OF [ONF13a] and NETCONF [EBS11]) as northbound and southbound interfaces. While the northbound interface is used to connect applications with the controller, the southbound interface is used to connect network devices in the data plane with the controller. So, network devices in the data plane perform forwarding based on the rules from the controller. Additional advantage of SDN is that it supervises network resources and transmission of data across heterogeneous domains. Hence, SDN solves the interoperability problems of network devices from different vendors [JMD14].

Network devices in the underlying network infrastructure communicate with the SDN controller via southbound interfaces to allow fast and intelligent network performance. Fusion H1 prototype nodes are Ethernet switches from Transpacket company designed based on the IHON principles to optimize the utilization of transmission capacity while offering absolute QoS [Fus13]. They are built based on NETCONF protocol and YANG data modeling language [VSBR14]. Management and supervision of traffic flows in the Fusion networks is, so far, NETCONF-based Network Management System (NMS) and Simple Network Management Protocol (SNMP) [VBB15]. As a result, we are motivated to experiment SDN controlled IHON focusing on Fusion nodes to replace the existing NMS scenario and to leverage all the benefits of an SDN controlled network and the IHON itself.

SDN has gained special attention by many industrial and academic institutions, thus multiple set of commercial and open SDN platforms [JMD14] have been proposed. OpenDayLight (ODL) platform is the testbed that we have chosen for our project; it is an open source SDN project aimed to realize centralized, programmatic control and monitoring of network devices [ODLe]. No work has been done to experiment ODL platform for IHON. Thus, we are inspired to evaluate NETCONF-based SDN for IHON in general and Fusion H1 nodes in specific with this testbed. Achieving SDN controlled IHON offers mainly two advantages. First, employing IHON in the data plane enables differentiated QoS for packet networks and absolute QoS for the circuit networks while optimizing the utilization of the transmission capacity. Second, implementing SDN allows flexibility and network programmability, and interoperability with other network domains and devices.

1.2 Objective and Methodology

SDN implementation in today's network is gaining acceptance as it simplifies the networking management and reduce associated OPEX and CAPEX costs. Even though SDN is highly researched for packet switching technology, it is still undergoing extension for circuit switching in the optical domain. For example, [DPM+10], [CNRF+13], and [DPM09], are SDN framework proposals for hybrid optical networks using OF as a southbound protocol. On the other hand, Fusion nodes are a special type of hybrid that uses the same physical wavelength for both circuit switched traffic and packet switched traffic in an interleaved manner. Additionally, the available IHON prototype node, Fusion H1 is designed based on on NETCONF which is the main southbound protocols for communicating with the SDN controller. This facilitates the possibility of an SDN framework based on NETCONF for IHON in general and Fusion H1 in specific.

This thesis work is a continuation of this author's specialization project [Ber15], and the objective is to implement SDN framework for integrated hybrid optical network focusing on Fusion networks. In the specialization project, it was evaluated that NETCONF protocol is better than OpenFlow (OF) with respect to complexity and cost to realize SDN for IHON networks because of the flexibility of the YANG schema to model all features without needing to extend the protocol itself, as would be the case with OF. Since multiple set of SDN platforms are widely available, our primary goal is to select compatible SDN controller for building our testbed, and enable programmability of the network functions and protocols with the selected testbed.

We have preferred ODL platform as the working testbed to implement SDN for IHON network. The reasons behind this choice are discussed in detail in Chapter 4. Since H1 nodes are NETCONF based devices, NETCONF testtool which is an open source NETCONF/YANG simulator is used to simulate the IHON nodes. As a result, we proposed SDN network setup with the simulation tool in the data plane, and ODL platform in the control plane. In order to allow management and monitoring of the node configurations with ODL, postman plug-in is used in the application layer. Postman allows us to send Hyper Text Transfer Protocol (HTTP) requests to the controller using RESTCONF northbound plug-in [Bro15]. RESTCONF is like a REST protocol that runs over HTTP used to access YANG based data by applying NETCONF datastores [Wat15]. The controller in return translates the requests and forwards to the nodes in the data plane using Netconf southbound protocol. Moreover, a network of three H1 nodes in the Uninett lab is used for experimenting SDN deployment with ODL. The objective is to successfully connect and configure the network through the SDN controller which is setup remotely in the offices at the ITEM department, NTNU.

1.3 Project Structure

The document is organized as follows: Chapter 2 describes the key characteristics and functionality of what IHON is, emphasizing on scheduling techniques and packet forwarding schemes in H1 prototype nodes. The SDN architecture is next described in Chapter 3; it includes a brief background for NETCONF protocol, NETCONF client-server communication procedures, and associated NETCONF operations. The chosen SDN controller platform for this project, the ODL architecture, is further described in Chapter 4. In addition, it includes the description of the other testbed components: RESTCONF protocol and associated operations together with testing tools. The results of the theoretical studies of this project are given in Chapter 5 which provides the SDN framework for controlling IHON. The framework illustrates an SDN setup for IHON in general with the necessary steps to establish NETCONF based communication with the controller. Chapter 6 provides the results for the emulation of NETCONF-based SDN with ODL platform for IHON. Furthermore, Chapter 7 discusses the experimental results of ODL implementation for H1 network at Uninett labs. Finally, Chapter 8 gives the conclusion for our thesis work and possible future work.

Chapter 2

Integrated Hybrid Optical Network (IHON)

In recent years, the deployment of fiber technology offers bulk transmission capacity, but the utilization is inefficient with the existing switching technologies. As a result, new proposals such as the hybrid architectures, specifically the IHON has emerged which combines the advantages of both circuit and packet switching techniques while diminishing their disadvantages. IHON offers guaranteed QoS and optimizes the utilization of the transmission capacity [GKB$^+$06]. In Section 2.1, we will present the different categories of hybrid architectures focusing on the Fusion H1 node. Moreover, Section 2.2 describes the forwarding schemes of Guaranteed Service Transport (GST) and Statistically Multiplexed (SM) Ethernet streams in H1 network.

2.1 Classes of Hybrid networks

Hybrid networks combine both packet and circuit switching technologies. Accordingly, there are three categories of hybrid networks based on the integration and interaction depth of the switching technologies: parallel, client-server, and integrated, i.e. IHON [GKB$^+$06]. While parallel and client-server architectures perform switching in distinct optical network resources, e.g. wavelengths, the IHON completely integrates both circuit and packets switching and transmits them in a shared wavelength simultaneously in an interleaved manner without affecting the QoS of the circuit switching class. These categories of hybrid networks are briefly discussed in Section 2.2 of [Ber15].

Researchers have proposed different solutions based on the IHON principles; Optical packet-switched migration-capable networks with service Guarantees (OpMiGua) [BNH03] and Fusion solutions [Fus12] are the two common types of IHON. As described in Section 2.3 of [Ber15], the core objective of these scenarios is to maximize the utilization of the transmission capacity by inserting packet switched traffic in the time gaps between circuit switched traffic. While the circuit traffic (known as GST traffic) has absolute priority, the packet traffic (known as SM traffic) has best effort (BE) priority. A GST gap detector is employed in IHON to precisely detect the time gaps between GST streams. SM packet scheduler, then, inserts suitable SM packets that fit with the gaps thereby avoiding preemption of SM flows. As a result, the utilization of the transmission capacity is optimized while offering guaranteed QoS for GST flows [LTG90].

The main distinctions between OpMiGua and Fusion solution are basically the mechanism used to distinguish between GST and SM flows, and the forwarding schemes within the network. OpMiGua splits the incoming flow in to GST and SM flows using either frequency shift keying (FSK) or fast optical switching techniques at the input node, and forwards them to the circuit and packet switch, respectively. By delaying the GST flows using Fiber Delay Line (FDL), suitable SM packets are inserted in the gaps. Both flows are, then, combined at the output to be forwarded to the next node [BNH03]. Section 2.3.1 of [Ber15] provides a brief description for OpMiGua.

On the other hand, Fusion solution, specifically the H1 node [Fus13], allows Ethernet based transport system over optical medium for both GST and SM flows which assigns distinct Virtual Local Area Network label (VLAN-ID) tags at the input edge node to distinguish between the two flows. Then, the intermediate nodes forward them depending only the added tags. At the destination, the edge node strips off the added tag and delivers the original stream to the corresponding receiver. H1 consists of two trunk interfaces each 10GE that used for transport of GST transparently via the Ethernet lines, and ten access interfaces each of 1GE for aggregating either SM or GST sub-wavelengths to improve the utilization of the channel.

2.2 Ethernet Streaming in Fusion H1 nodes

Although the GST traffic occupies an end-to-end lightpath, there is still unused capacity which lowers the utilization. So, H1 node provides a solution to detect the free time gaps, and insert there suitable packets to optimize the utilization without touching the dedicated services. Figure 2.1 illustrates the gap filling mechanism in H1 nodes. By correctly tagging the incoming flows at the input node, GST flows arriving at the input interface are forwarded to the output trunk-interface with absolute priority. On the other hand, SM flows are first processed, buffered, and transmitted only if free time gaps are detected, otherwise wait in the queue (buffer) for latter forwarding until an available fitting time-gap.

Unlike OpMiGua which employs FDL to detect the gaps between GST flows, Fusion H1 node applies an electronic delay, a fixed time window, to precisely measure the gap. Then, a packet scheduler scans the queues for a suitable SM packet that fits with the detected gap and inserts it without affecting the GST flow. As a result,

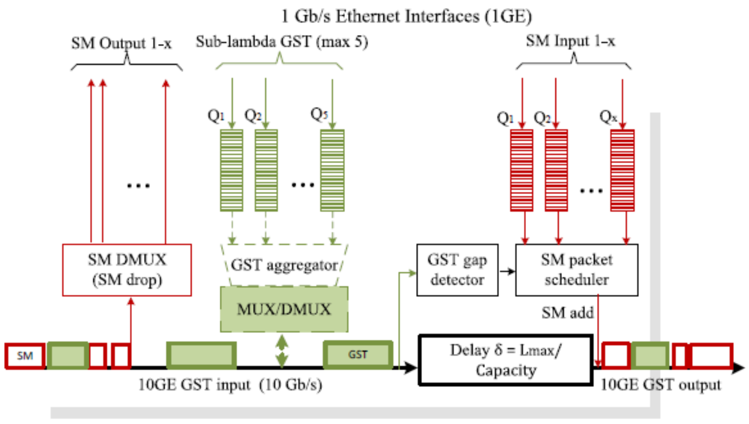


Figure 2.1: Fusion H1 node is composed of ten 1GE client interfaces and two 10 GE line interfaces. The transmission illustrated in this figure is only unidirectional (left-to-right direction). Moreover, the 10GE interfaces transports either single GST stream or carries five 1GE GST streams. On the other hand, the 1GEs are used for local add/drop. GST gap detector measures the exact free time gap which allows the SM packet scheduler to select and insert a suitable SM packet from the queues. Figure taken from [VBB13].

packet delay variation of GST packet is avoided in Fusion H1 nodes unlike the electronic scheduling algorithm, while not pre-empting SM packets either [VBB13]. A very brief discussion about the forwarding scheme and scheduling techniques of the H1 nodes is given in Section 2.4 of [Ber15].

In conclusion, IHON is a promising hybrid architecture which combines the advantages of both circuit and packet switched traffic while diminishing their disadvantages thereby optimizing the capacity utilization and achieving guaranteed QoS for the absolute priority traffic. In the next chapter, we will discuss SDN platform, NETCONF southbound protocol and associated operations.

Chapter 3

Software Defined Networking (SDN)

This chapter presents SDN and the southbound protocol to communicate with the network nodes (data plane), the NETCONF protocol. Section 3.1 describes about SDN roots and its architecture while Section 3.2 gives a brief background to NETCONF protocol. In Section 3.2.1, we provide the layering architecture of NETCONF. After that, we present NETCONF-based client-server communication using Secure Shell (SSH) session. We also include an overview of NETCONF datastores, capability exchanges, and set of NETCONF operations. Finally, we discuss YANG data modeling language which is needed to model the nodes (abstract them) and present the information to the controller to be able to operate/control the data plane in Section 3.3.

3.1 SDN Architecture and Working Principles

SDN is a recently evolving networking technology which decouples the control and data plane which are integrated in a vertical architecture in traditional network equipment. As a result, it enables programmability of both network functions and protocols using a centralized controller which hosts network operating system [CNRF$^+$13]. The purpose of the logical controller is to abstract the switching technology and transport system in the underlying infrastructure, and build and present a logical map of the complete network to applications and services that run on top it. Network operators are, therefore, able to manipulate logical map of the network and create virtual networks. Since the control and data planes are separated, the logically centralized control plane supports supervision of multiple and heterogeneous transport technologies from various domains.

Furthermore, SDN has three main layers: application, control, and infrastructure layers [ONF13b]. Application layer is a collection of end-user applications that run on top of the controller for specific function, for example, provisioning end-to-end connectivity/services, traffic engineering, load balancing, etc. Control plane, on the other hand, consists of SDN controllers which manage the underlying network devices in the data plane through standardized southbound APIs. It also provides, northbound interfaces to communicate with the applications. Lastly, Infrastructure layer is the data plane which includes Network Elements (NEs) that perform forwarding and switching of packets based on the instructions from the controller.

Multiple network vendors have focused their research on the southbound interface, for example, [JMD14] and [CNRF$^+$13] to allow SDN realization with a centralized controller over heterogeneous network devices, thus avoid the interoperability problems. To mention a few, OF and NETCONF protocols, are examples of the southbound interfaces. OF protocol [ONF13a], is the most researched protocol for packet switched networks [GDS$^+$10]. On the other hand, NETCONF is mostly applicable for NMS [SWJR$^+$94] [HGRG06].

In this project, the objective is to apply NETCONF protocol as a southbound interface in realization of SDN for Fusion nodes. Hence, we will discuss NETCONF protocol and associated operations in the section below.

3.2 Network Configuration Protocol (NETCONF)

The Network Configuration Protocol (NETCONF) is a protocol used for installing, manipulating, and deleting of network devices configuration [EBS11]. Both configuration data and protocol messages are based on XML encoding data. In NETCONF, the functionality of the management protocol and native functionality of the device are mirrors of each other. As a result, implementation costs are greatly reduced.

3.2.1 NETCONF Layering Model

NETCONF is logically divided in to four layers [EBS11], secure transport, messages, operations, and content layers as illustrated in Figure 3.1.

1. Secure Transport layer: is a medium used client-server communication, and can be any transport protocol that satisfies a set of requirements. Below are listed the requirements that the underling transport protocol should satisfy in order to fulfill the NETCONF functionality.
   - Connected-Oriented Operation: The connection in NETCONF is maintained throughout the communications of the peers, i.e. NETCONF is a connection-oriented protocol. Thus, the connection is always reliable, long-lived, and delivers ordered data. Moreover, when connection terminates, the resources being requested from server must be automatically released which allows easier failure recovery and robust system.
   - Confidentiality, Integrity, and Authentication: Confidentiality, Integrity, and Authentication (CIA), are the main security features that NETCONF

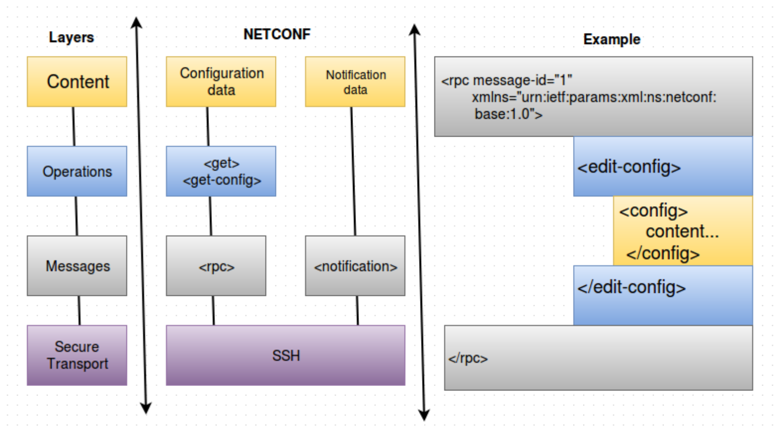


Figure 3.1: Conceptual NETCONF logical layers, and mapping of a sample NETCONF operation to each layer, adopted from [EBS11].

expects from the transport protocol. In addition, NETCONF must allow the reply to be protected. Based on the underlying protocol, connection between peers can be encrypted either using SSH [YL06d] or Transport Layer Security (TLS) [DR08].

– Mandatory Transport Protocol: Since NETCONF is transport independent protocol, mapping of various transport protocols is possible. Similarly, multiple set of session layers can be mapped to the session-layer of NETCONF protocol. Hence, implementation of NETCONF protocol must support mapping of SSH transport protocol which allows a user or an application to execute NETCONF from a secure shell session.

2. Messages Layer: RPCs and notifications are encoded using easy and transport-independent framing techniques in this layer.
3. Operation Layer: Base protocol operations supported by NETCONF are defined in operation layer, and invoked in the form of RPC methods including XML-encoded parameters, e.g. edit-config, get, get-config, etc.
4. Content Layer: This layer contains both notification and configuration data. Moreover, content layer is well formed in XML.

The YANG data modeling language is used to define protocol operations and NETCONF data models which mainly includes the content and operation layers of the NETCONF layers [MB10]. As a result, the hierarchical organization of NETCONF operations, configuration and state data, notifications, and RPCs are modeled using YANG as a tree. Each node in the tree has a name, and either a value or child nodes. A detailed discussion of YANG is given in Section 3.3.

3.2.2 Running NETCONF over SSH Session

SSH protocol is a mechanism to provide secure network services mainly secure remote login over an insecure network. SSH protocol has three main components [YL06d].

1. Transport Layer Protocol [SSH-TRANS]: This component is responsible for providing confidentiality, Integrity, and authentication of a server over Transmission Control Protocol (TCP)/IP connection with perfect forward secrecy [YL06c].
2. User Authentication Protocol [SSH-USERAUTH]: Provides authentication of the user in the client-side to the server [YL06a].
3. Connection Protocol [SSH-CONNECT]: Multiple set of logical channels are multiplexed in to an encrypted tunnel using the connection protocol [YL06b].

NETCONF is implemented in SSH session using the SSH connection protocol over SSH transport protocol [WG06]. As a result, an application or a user can execute NETCONF operations from a secure shell session. Applying the existing SSH protocol allows to reduce the key management operational costs. Furthermore, a single transport layer carries more than one logical channel. The two ends of the SSH transport connection are called client and server; the server is a network device, e.g. the H1 node, whereas the client can be either a script or an application running as a network manager, e.g. the SDN controller. NETCONF uses RPC-based scenario to enable client and server communications. The client-server communication is depicted in Figure 3.2.

The connection is first established from the client with SSH transport protocol [YL06a]. Afterwards, both peers exchange an integrity and encryption keys to authenticate each other. In order to authenticate the user, the client invokes "SSH-USERAUTH" service. Following successful authentication of the user, the client invokes "SSH-CONNECT" to establish SSH session. Using the established SSH session, the client invokes "netconf" to run NETCONF as an SSH subsystem. By doing this, the script will ignore shell prompts and extraneous information, like system messages at shell start-up. NETCONF servers are set to access the "netconf" SSH subsystem after the SSH session is established. As a result, network devices and firewalls easily identify and filter NETCONF traffic. In order to invoke the NETCONF as an SSH subsystem over a specific port, the user (or application) uses the command [YL06a]: *[user@client]$ ssh -s server -p <port-number> netconf*, where -s option enables the "netconf" command to be invoked as SSH subsystem.

After the establishment of NETCONF session, the server forwards a <hello> message in an XML format that contains its capabilities. Referring to the received capability, the user parses it to determine the server's NETCONF capabilities. Similarly, the client sends its capabilities to the server within a <hello> message. In reality, however, both peers send the <hello> messages immediately after the NETCONF subsystem is initiated, may be simultaneously [YL06a].

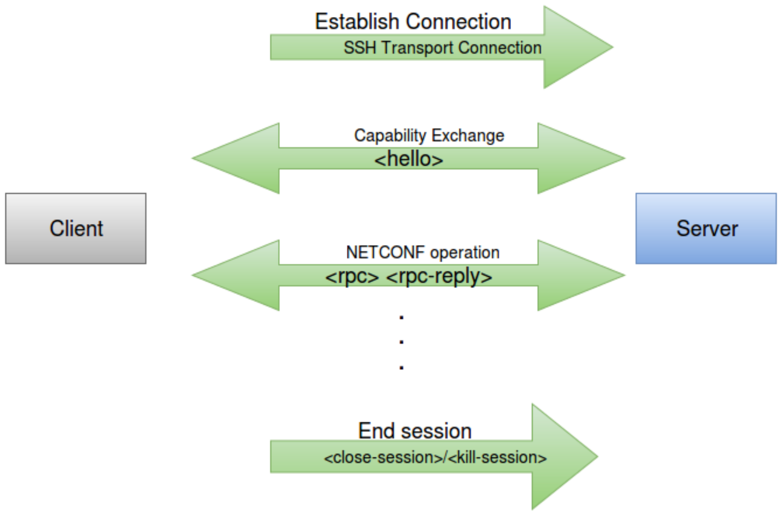


Figure 3.2: NETCONF session, showing client and server transaction model. After establishing connection, both peers exchange their capabilities using hello messages. After exchanging their capabilities, the client sends requests to the server as a RPC models. The server, then, responses for each request. Lastly, the server initiates session closing, and the server sends confirmation and terminates the session.

By this time, NETCONF protocol session is established, thus the client side acts as a manager while the server side becomes an agent. Hence, the manager executes NETCONF operation by sending XML documents which contain <rpc> request to the server. The agent in turn responses back for each <rpc> request with full XML documents that are contained in <rpc-reply> messages.

To end the NETCONF session, the client executes either <close-session> or <kill-session> operation. Since the agent processes RPC messages in a sequential order, all RPC messages received after <close-session> are ignored. Having received this message, the agent responds with <ok> message, and closes the SSH session channel immediately [YL06a].

A network system contains multiple network devices connected to a single client. Since NETCONF is able to establish multiple sessions [EBS11], parallel NETCONF sessions are created between a single client and multiple devices. As a result, multiple remote devices are easily managed through a single client. In the following sections we will briefly discuss the basic concepts regarding NETCONF datastores, capability exchanges, and RPC which help to accomplish client-server transaction.

3.2.3 NETCONF Datastores

A running system is composed of state and configuration data [EBS11]. Configuration data is a well defined writable data which essentially brings the system to its current state from its initial default state. On the other hand, data other than the configuration data which exists in the system is called state data. For example, statistics and read-only data are state data. As a result, the NETCONF protocol not only differentiates configuration data from state data but it also provides distinct operations to access each data separately. Moreover, at least one configuration datastore is available that handles configuration operations. A configuration datastore is a full set of configuration data that transforms a device to the desired operational level from initial default level. Neither state data nor executive commands can not exist in the configuration datastore. Overall, there are three types of standard configuration datastores [EBS11].

1. Running configuration datastore: It contains the whole set of configuration which is currently running on the device. There is only one running configuration datastore in a device, and it must be always active which is illustrated in Figure 3.3 part a. NETCONF protocol operations use the <running> element to access the running configuration datastore. If a device supports additional configuration datastores, it advertises in its capabilities to the client. Both <candidate> and <startup> are the two types of configuration data stores that a device can advertise.

2. Candidate configuration datastore: A device which supports a candidate datastore expresses its capability as *:candidate* during capability exchange. Candidate is a storage with complete configuration data set where we can manipulate and create configuration data. It is possible to modify, add, delete the data so as to build the required configuration data. As a result, running configuration of the device is set to the current value of the candidate configuration via <commit> operation as depicted in part c of Figure 3.3.

3. Startup configuration datastore: As its name indicates, startup configuration is implemented at the next reboot, and is advertised as *:startup* capability. Since operations in the running configuration are not immediately copied to the startup configuration, an explicit copy operation, i.e. <copy-config>, is executed to update the contents. This is illustrated in part b of Figure 3.3.

Therefore, a device can support all the three datastores. However, the running datastore may or may not be directly writable (this capability is expressed as *:writable-running* which is described in the following section, Section 3.2.4). If device does not advertise this capability, editing and copying of configuration data are executed in the

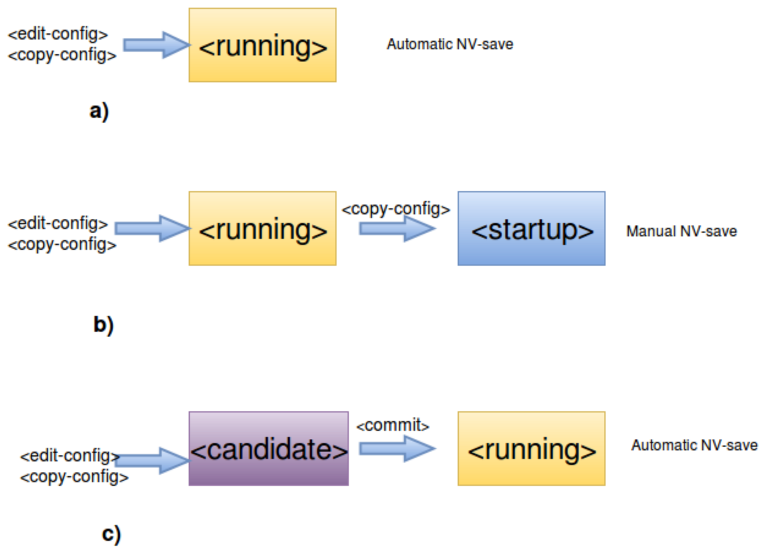


Figure 3.3: Datastores of NETCONF protocol: a) *<running>*: Edits are saved in NV-storage automatically, b) *<startup>*: Edits are saved manually, and c) *<candidate>*: Edits are saved automatically, adopted from [Cen].

candidate datastore. Both the IHON nodes used in our emulation and the H1 nodes from Uninett lab, support *:candidate* capability as illustrated in Appendix C.3 and Appendix E.1, respectively. Hence, we execute NETCONF operations, e.g. editing and retrieving their configurations using the candidate datastore which is briefly discussed in Chapter 6 and Chapter 7.

3.2.4 Capability Exchange

Capability defines the functionality supported above the base NETCONF specifications, and is specified using Uniform Resource Identifier (URI) [Cen]. During the initial capability exchange, both server and client advertise their capabilities. As a result, each peer is considers the capabilities that it supports otherwise ignore if received from unknown source. Source code 3.1 shows the capability format [EBS11]. In this case, name is the capability name. Whenever we want to reference capabilities in emails or discussions, we use shorthand :{name}, or we concatenate the version as :{name}:{version} when the capability is found in multiple versions.

Source code 3.1 Capability format, adopted from [Cen]

```
urn:ietf:params:netconf:capability:{name}:1.x
```

Both client and server send a <hello> message that contains their capabilities when the NETCONF session is opened. It is required that each peer must forward minimally the base NETCONF capability, *"urn:ietf:params:netconf:base:1.1"* [EBS11]. Moreover, both peers must have common protocol version, and should compare the URIs of the protocol version capability. When the URI contains any encoded parameters at its end section, the base part is only used. If there is no common protocol version, the peers must terminate the session. If multiple version URIs occur, the peers must use the most recent (i.e. the highest numbered) protocol version. [EBS11]

When a server sends the <hello> message, it must add a <session-id> element which identifies the NETCONF session. On the contrary, the client does not add a <session-id> element when it sends <hello> element. Moreover, a server must end the NETCONF session if it receives <hello> message along with <session-id> element. The client, however, terminates the session if it does not found a <sesson-id> element within the <hello> message of the server. Below are listed the NETCONF base capabilities [EBS11].

1. Writable-Running Capability: The *:writable-running* indicates the <running> configuration datastore can be written directly. If the target datastore is <running> configuration, both <edit-config> and <copy-config> operations are supported in the device. This specific capability has the following format:

   urn:ietf:params:netconf:capability:writable-running:1:0

2. Candidate Configuration capability: The *:candidate* implies that the candidate configuration data exists in the device, and it is supported. The *:candidate* capability is expressed and identified as follows:

   urn:ietf:params:netconf:capability:candidate:1.0

3. Confirmed commit Capability: The *:confirmed-commit:1.1* capability is used to indicate that <cancel-commit> operation is supported in the server. Additionally, the parameters in <commit> operation, such as <confirmed> and <confirm-timeout> are also supported. The *:confirmed-commit:1.1* capability is available as long as the *:candidate* capability is supported, and is expressed as follows:

   urn:ietf:params:netconf:capabiilty:confirmed-commit:1.1

4. Rollback-on-Error Capability: The *:rollback-on-error* capability implies that whenever an error is encountered, the server will rollback to the previous state. In other words, when performing <edit-config> operation, the <error-option> parameter has "rollback-on-error" value which has the following format:

   urn:ietf:params:netconf:capability:rollback-on-errror:1.0

5. Validate Capability: Validation is checking the syntactic and semantic errors of a complete configuration before implementing it to the server. The *:validate* capability indicates that the <validate> protocol operation is supported, and is used to check for syntax errors. The following string identifies the *:validate:1.1* capability:

   urn:ietf:params:netconf:capability:validate:1.1

6. Distinct Startup Capability: In this scenario the startup and running configuration datastores are separated in the server. As soon as the device boots, it is loaded with the startup configuration. However, the startup configuration will not get automatic copy of the running configuration operations. Thus, it is explicitly copied using the <copy-config> operation. the *:startup* capability has the following format:

   urn:ietf:params:netconf:capability:startup:1.0

7. Uniform Resource Locator (URL) Capability: When the *:url* capability is advertised, the <target> and <source> parameters of the NETCONF peer will accept <url> element. URL arguments which indicate all the supported URL schemes, are used to identify *:url* capability which is identified as follows:

   urn:ietf:params:netconf:capability:url:1.0?scheme={name,...}

   The "scheme" argument is mandatory for the *:url* capability, which contains the list of schemes supported by the NETCONF peer separated by a comma as shown in the example below:

   urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file

8. XML Path (XPath) Capability: The *:xpath* capability implies that the <filter> element uses XPath expressions. The root node of the XPath expression in the data model contain children probably with any number of element nodes. The following string identifies the *:xpath* capability:

   urn:ietf:params:netconf:capability:xpath:1.0

Source code 3.2, and 3.3, are sample NETCONF capabilities of server and client, respectively. As can be seen, both peers advertise the base NETCONF capability.

Furthermore, in our emulation, the simulation tool supports *:candidate* and *:base* capabilities as shown in Appendix C.3, and the supported NETCONF operations are discussed in Section 6.1.1. On the other hand, for the lab experiment, we use Fusion H1 nodes from Uninett lab. The H1 nodes support extra capabilities in addition to those supported by the simulation tool; for example, *:confirmed-commit*, *:validate*, and others as illustrated in Appendix E.1. Consequently, the H1 nodes support more NETCONF operations as compared with the simulation tool.

Source code 3.2 An example of server capabilities, adopted from [EBS11].

```
<?xml version="1.0" encoding="UTF-8"?>
 <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
      <capability>
        urn:ietf:params:netconf:base:1.1
      </capability>
      <capability>
        urn:ietf:params:ns:netconf:capability:startup:1.0
      </capability>
    </capabilities>
    <session-id>4</session-id>
 </hello>
]]>]]>
```

Source code 3.3 An example of client capabilities, adopted from [EBS11].

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

3.2.5 Remote Procedure Call (RPC) Model

NETCONF protocol applies RPC paradigm in client-server communication [EBS11]. Before the client sends an RPC to a server via a secure and connection-oriented session, it first encodes the RPC using XML. The response from the server is also encoded in XML. In order for both client and server to understand the syntax constraints during the transaction, all the contents of request and reply are described in XML schema. In RPC, the NETCONF requests and responses are framed using transport-protocol independent mechanism; the client uses <rpc> to frame the requests while the server uses <rpc-reply> elements to frame the responses. The basic RPC elements are <rpc>, <rpc-reply>, <rpc-error>, and <ok> [EBS11], and we will briefly discuss each element below.

<rpc> Element

When a client sends NETCONF request to the server, the request is enclosed using <rpc> element which is identified by "message-id" attribute [EBS11]. This attribute is a mandatory field, and encodes an increasing integer in a monotonic way. Moreover, the RPC contains a name and parameters which are all encoded as values of the <rpc> element. Source code 3.4 illustrates, an interface configuration invoked two parameters, <port> and <ip-address>.

Source code 3.4 Sample method invoking in <rpc>, adopted from [EBS11].

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <interface-method
       xmlns="http://example.net/interfaces/interface/1.0">
        <port>80</port>
        <ip-address>127.17.0.2</ip-address>
    </interface-method>
</rpc>
```

<rpc-reply> Element

When <rpc> message request is sent, the response is <rpc-reply>, and has mandatory and equal "message-id" attribute as the request [EBS11]. If the <rpc> element contains other additional attributes, the server must return them unmodified within the <rpc-reply> element. Moreover, the <rpc-reply> element contains child elements that represent the response data. For example, Source code 3.5 illustrates a <get> method invoked in the <rpc> element which includes "interface-type" attribute. Hence, we can see that the "interface-type" attribute and the requested content are returned in the <rpc-reply> element.

<rpc-error> Element

If an error is encountered while processing the <rpc> request, the <rpc-reply> includes an <rpc-error> element to indicate the failure [EBS11]. Even though more than one errors could occur in the <rpc> request processing, the server only detects or reports a single <rpc-error> element. Moreover, the server does not follow specific procedure for checking error conditions that occur during processing, rather it must response <rpc-error> for any occurrence of error conditions. Source code 3.6 shows the reply from a server with <rpc-error> element.

Source code 3.5 Sample method invoking in <rpc> and <rpc-reply> format, adopted from [EBS11].

```
<rpc message-id="101"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
         xmlns:ex="http://sample.net/interfaces/1.0"
         ex:interface-type="eth">
      <get/>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:ex="http://sample.net/interfaces/1.0"
     ex:interface-type="eth">
  <data>
    <!-- all contents... -->
  </data>
</rpc-reply>
```

Source code 3.6 Sample <rpc-error> format, adopted from [EBS11].

```
    <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <rpc-error>
         <!-- error contents... -->
      </rpc-error>
    </rpc-reply>
```

<ok> Element

If the <rpc> request is successful (i.e. without warnings or errors), and the operation responds no data, then <ok> element is replied in the <rpc-reply> messages [EBS11]. Source code 3.7 shows the <rpc-reply> with <ok> element.

Source code 3.7 Sample <ok> element in <rpc-reply> message, adopted from [EBS11].

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

3.2.6 NETCONF Protocol Operations

Using NETCONF operations, the client can manage the network device configurations and its information state. Although the device can advertise multiple set of operations during capability exchange, the base protocol operations are mainly configuring, retrieving, copying, and deleting of the configuration datastores [EBS11].

<get-config>

<get-config> is an important NETCONF operation used both in our emulation and experiment. Since configuration and state data are separated in NETCONF protocol, they are accessed using base operations either separately or together. The <get-config> is used to access configuration datastore, and retrieves all or part of it. When the request is successful, the sever sends <data> element containing the results for the query within the <rpc-reply> message. Source code 3.8 illustrates <get-config> operation to retrieve the all interfaces. The <get-config> has the following parameters [EBS11]:

– source: Indicates the configuration datastore to be queried, such as <running/>.
– filter: If only specific section is needed to be retrieved, filter element is used. Otherwise, the response contains all the configuration data. The syntax filter within the <filter> element is represented using the "type" attribute, though it is optional.

<edit-config>

The target configuration datastore can be loaded with a specific configuration using the <edit-config> operation. The target configuration datastore is created unless it previously existed. Moreover, the <config> parameter can be replaced by <url> element if *:url* capability is supported by a NETCONF peer [EBS11].

Like the <get-config> operation, the <edit-config> is an important operation that we implemented in our testbed to edit the configurations of simulated IHON nodes and H1 nodes at Uninett lab. As oppose to <copy-config> which replaces the target configuration, the <edit-config> operation simply modifies the target configuration depending on the requested operations and the data from the source. As a result, the target configuration datastore is edited with the new config elements. Source code 3.9 shows adding a new interface to the node.

In addition to the above operations, NETCONF has other operations which are summarized in Table 3.1.

Source code 3.8 Sample <get-config> operation, adopted from [EBS11].

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <top xmlns="http://sample.com/schema/1.2/config">
        <interfaces/>
      </top>
    </filter>
  </get-config>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <top xmlns="http://sample.com/schema/1.2/config">
      <interfaces>
        <interface>
          <name>Ethernet0/1</name>
          <ip-address>127.12.13.1</ip-address>
          <prefix>24</prefix>
        </interface>
        <!-- additional <interface> elements appear here... -->
      </interfaces>
    </top>
  </data>
</rpc-reply>
```

3.3 YANG Modeling Language

The NETCONF protocol (mainly configuration and state data, RPC, notifications) is modeled using YANG data modeling language. Moreover, All the data within the client-server transaction is defined using YANG [MB10]. Data models are structured into modules and submodules, where a module can import data nodes from other modules. As a result, YANG allows to augment the module hierarchy. A data is modeled using four types of nodes: container, list, leaf-list, and leaf nodes. These nodes have their own YANG syntax and can be converted in to equivalent NETCONF XML format. Like other programming languages, YANG also has built-in types. Furthermore, YANG allows users to define their own types derived from the built-in

Source code 3.9 Sample <edit-config> operation, adopted from [EBS11].

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <top xmlns="http://sample.com/schema/1.2/config">
        <interface>
          <name>Ethernet0/2</name>
          <ip-address>127.12.13.2</ip-address>
          <prefix>24</prefix>
        </interface>
      </top>
    </config>
  </edit-config>
</rpc>

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

types. A brief description of the YANG data modeling language is briefly discussed in Section 3.3.2 of [Ber15].

A sample YANG module is given in source code 3.10 which represents an interface configuration of a router. Since the router can have more than one interface, we use the list node in order to list all the interfaces of the router. Moreover, the specification of each interface is defined using leaf node which includes its name, type, and address. The equivalent NETCONF XML representation for this particular YANG schema with sample interface configurations is depicted in source code 3.11.

In this chapter we briefly discussed SDN, NETCONF protocol, and YANG. We have seen that, implementing SDN in today's network allows programmable and cost effective management of heterogeneous networks. Furthermore, we presented the NETCONF datastores and its basic operations. Dividing NETCONF in to logical layers allows easier client-server interaction. Moreover, we have seen that client and server establish secure connection, and exchange their capabilities before executing NETCONF operations. As YANG is used to model NETCONF configuration of IHON, we have also covered the basic concepts. In the following chapter, we will

Table 3.1: NETCONF operations along with their descriptions, adopted from [EBS11].

NETCONF Operation	Description
<copy-config>	Configuration is copied from source to target configuration datastore.
<delete-config>	Delete a configuration datastore identified by the target parameter with the exception of <running> datastore.
<lock>	Used to lock the target configuration datastore.
<unlock>	The configuration datastore is unlocked.
<get>	Uses filter option in order to select a specific running configuration and/or state information of the device.
<close-session>	Closes ongoing session in a graceful manner.
<kill-session>	Forceful kiling of the session session.
<commit>	When device advertises candidate capability, candidate configuration datastore is committed to running configuration.
<discard-changes>	Candidate configuration is restored back to running configuration datastore.
<validate>	Used to validate a source configuration datastore contents.
<create-subscription>	A notification stream is subscribed by specifying the start and stop times, and a filter

discuss the background to ODL testbed, its implementation and associated tools used to execute operations.

Source code 3.10 YANG schema, e.g, interface schema of a router

```
module router {
    yang-version 1;
    namespace "urn:ntnu:router";
    prefix router;

    description "router interface configuration";

    revision "2016-06-06" {
        description "Initial version.";
    }
    container router {
        list interfaces{//contains a definition for a port
            key "given-name";/*the port name is set as
                              a unique identifier of the port*/
            leaf given-name{
                type string;
            }
            leaf-list type{
                type string;
            }
            list address { //defines the ip address for the interface
                key "ip-address"; //set ip address as a key
                leaf ip-address {
                    type yang:ip-address;
                }
                leaf net-mask{//set the subnet mask
                    type int32;
                }
            }
        }
    }
}
```

Source code 3.11 NETCONF XML representation for the YANG given in Source code 3.10.

```
<router xmlns="urn:ntnu:router">
    <router>
        <interfaces>
            <given-name>FastEthernet0/0</given-name>
            <type>atm</type>
            <address>
                <ip-address>192.168.208.1</ip-address>
                <net-mask>24</net-mask>
            </address>
        </interfaces>

        <!--other interface configurations-->

    </router>
</router>
```

Chapter 4

OpenDaylight Platform

This chapter presents the testbed that we have built and implemented (extended) SDN for the IHON. In Section 4.1, we provide a brief background to the chosen ODL platform focusing on its role to achieve the SDN goals. Section 4.2 gives the core reasons for choosing ODL platform over other SDN platforms while Section 4.3 covers the layering architecture of the ODL platform including the controller and Model Driven (MD)-Service Abstraction Layer (SAL) data storage. Moreover, Section 4.4 describes RESTCONF protocol and basic operations along with the testing tools.

4.1 Introduction to ODL

SDN separates the control and data planes, thus all network functions and protocols are programmable [JMD14]. A central controller abstracts the data plane, and network devices in the data plane perform forwarding based on the instructions from the controller. Multiple network vendors are providing SDN controllers in order to accomplish the SDN objectives [ODLe]. Even though some controllers like NOX[1], are commercial versions, many open source controllers are readily available and still going developments (for example, POX[2], Beacon[3], and ODL).

The ODL project which is an open-source facilitates the SDN adoption and Network Functions Virtualization (NFV) [ODLe]. Irrespective of the differences between vendor environments, ODL offers network services across heterogeneous network devices. In order to provide such services, ODL employs microservices architecture which enables users to successfully control protocols, applications, and plug-ins, and create connections between providers and customers. Large community is involved in the ODL development, and new versions are released every six months as summarized in Table 4.1.

[1]http://www.noxrepo.org/nox/about-nox/
[2]http://www.noxrepo.org/pox/about-pox/
[3]https://openflow.stanford.edu/display/Beacon/Home

Table 4.1: ODL releases [ODLe]: Hydrogen, Helium, Lithium, Beryllium, and Boron along with their features.

Version	Release year	Features
Hydrogen	Feb 2014	Contains open code controller, protocol plug-ins and virtualizations capability
Helium	Nov 2014	Applies karaf and model-driven network management for more appropriate network environment
Lithium	Jun 2015	Clustering is introduced, and new network protocols are supported via additional plug-ins
Beryllium	Feb 2016	Provides clustering and high availability, data handling is improved, transport messaging, large scale of network management, network models are highly abstracted, and new Graphical User Interface (GUI).
Boron	Not released yet	under review

The architecture of most networks is designed to support the workloads and requirements of the existing demand. With the implementation of SDN, however, the existing network can be optimized to fit today's needs, as well as adapt to changing requirements. Since SDN can be implemented in different ways, ODL provides a common platform which is possible to be configured in variety of ways so as to address a range of network challenges. By integrating open source, open API, and open standards, ODL delivers an SDN scenario which allows programmable, intelligence, and adaptable network [ODLd]. The steps for installing and running ODL platform is given in Appendix A.

4.2 Why ODL

Compared to traditional SDN platforms, ODL based SDN has the following basic features [ODLa].

- ODL project is an opensource where huge developing community is participating and support its implementations, such as Cisco, Ericsson, and at&t[4].
- ODL has microservices architecture. During installation of the ODL controller, a user activates a particular service or protocol depending on the application requirements; this is referred to as a micro-service. For example, NETCONF feature is installed to allow Netconf-based network devices, like IHON to connect with the ODL through NETCONF southbound plug-in.

[4]https://www.opendaylight.org/membership

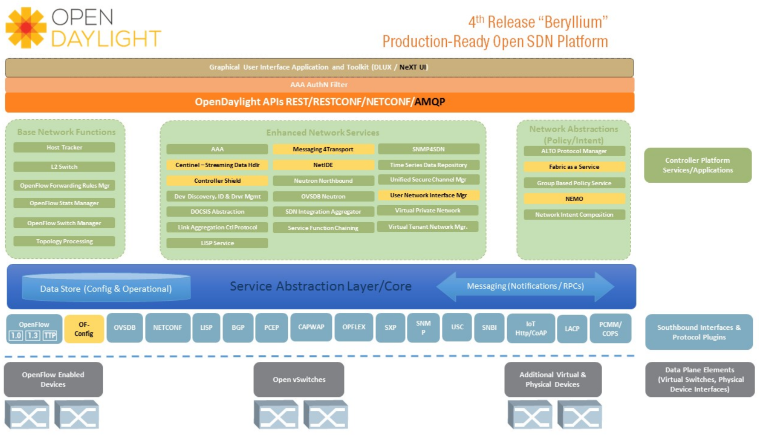

OPEN DAYLIGHT
4th Release "Beryllium"
Production-Ready Open SDN Platform
Graphical User Interface Application and Toolkit (DLUX / NeXT UI)
AAA AuthN Filter
OpenDaylight APIs REST/RESTCONF/NETCONF/AMQP
Base Network Functions
Host Tracker
L2 Switch
OpenFlow Forwarding Rules Mgr
OpenFlow Stats Manager
OpenFlow Switch Manager
Topology Processing
Enhanced Network Services
AAA
Messaging 4Transport
SNMP4SDN
Centinel – Streaming Data Hdlr
NetIDE
Time Series Data Repository
Controller Shield
Neutron Northbound
Unified Secure Channel Mgr
Dev Discovery, ID & Drvr Mgmt
OVSDB Neutron
User Network Interface Mgr
DOCSIS Abstraction
SDN Integration Aggregator
Virtual Private Network
Link Aggregation Ctl Protocol
Service Function Chaining
Virtual Tenant Network Mgr.
LISP Service
Network Abstractions (Policy/Intent)
ALTO Protocol Manager
Fabric as a Service
Group Based Policy Service
NEMO
Network Intent Composition
Controller Platform Services/Applications
Data Store (Config & Operational)
Service Abstraction Layer/Core
Messaging (Notifications / RPCs)
OpenFlow 1.0 1.3 TTP
OF-Config
OVSDB
NETCONF
LISP
BGP
PCEP
CAPWAP
OPFLEX
SXP
SNMP
USC
SNBI
IoT Http/CoAP
LACP
PCMM/COPS
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)

Figure 4.1: Architecture of Beryllium ODL platform, taken from [ODL16].

– In addition to NETCONF, ODL supports other network protocols such as OF, Border Gateway Protocol (BGP), SNMP, and more.
– Allows for development of new functionality including network services and protocols.

4.3 ODL Architecture

ODL is a modular platform where most modules reuse common interfaces and services [ODLd]. By using the model-driven service abstraction of the ODL, users are able to develop applications which run over multivendor hardware and southbound protocols. As a result, functionality of the platform bundles is leveraged, and each bundle uses java interfaces to export relevant services. Additional functions and services are added via internal plug-ins. For example, the topology and statistics of the network are gathered using dynamic plug-ins.

The architecture of the ODL platform for the most current release, i.e. the Beryllium version, is depicted in Figure 4.1; it shows a very detailed internal architecture of the platform. A simplified view is illustrated in Figure 4.2. Overall, ODL is three layered architecture [ODL16]:

1. Network Applications and Orchestrations: This layer is composed of various applications which are responsible for computing network traffic engineering, mainly controlling and monitoring of the controller. Furthermore, it contains solutions which use virtualization services.
2. Controller: The abstraction of SDN is implemented in the central layer, i.e. the controller layer. Additionally, network applications use various modules located in the controller layer to retrieve information regarding the network status. The

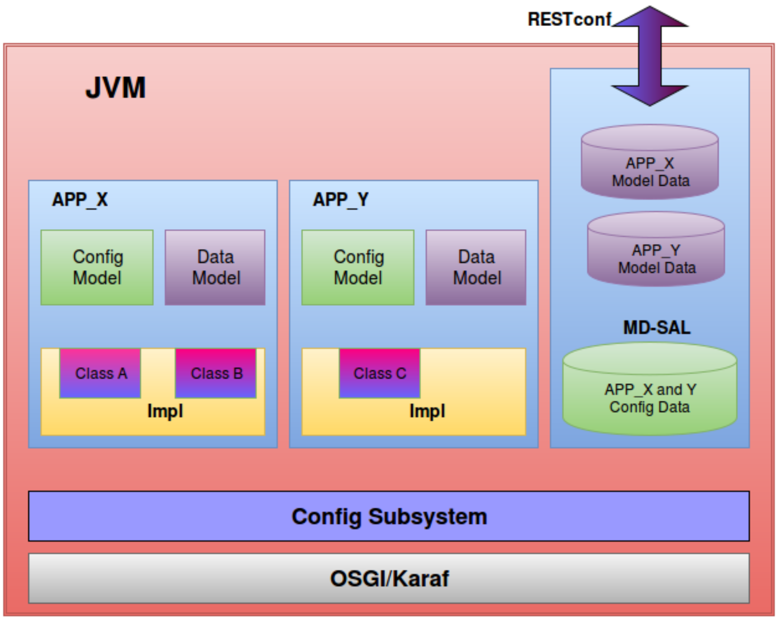


Figure 4.2: Simplified diagram of ODL platform with two sample applications, adopted from [ODL16].

controller further contains plug-ins which allow communications with various network protocols like, NETCONF, BGP, and OF. Different applications can be developed using the APIs from the controller layer.

3. Data plane: Either virtual or physical interfaces constitutes to the data plane layer. Using the implemented protocols, the network elements can be programmed through the controller. Since ODL implements a service abstraction layer, heterogeneous network elements are supported with the ODL platform.

4.3.1 ODL Controller

ODL controller by itself is a Java Virtual Machine (JVM) software, thus any hardware or operating system that supports Java can be used to run the controller [Com16c]. More specifically, ODL is java-based, and model-driven controller. It uses YANG for modeling applications and different aspects of the system. Since SDN scenario is implemented in the controller, the following listed tools are necessary for the controller [Com16c]:

– Maven: Maven is a tool for project management, thus ODL make use of Maven to allow build automation. By using of this tool, developers are able to control the necessary plug-ins and the application dependencies.

- Open Service Gateway Interface (OSGI): JAR packages and bundles are loaded dynamically using the OSGI tool which is mainly the back-end of the ODL controller. To exchange information, OSGI lets bundles bind together.
- JAVA interfaces: Specific bundles require event listeners, formal patterns, and specifications in order to apply call-back functions for events. All these tasks are realized using Java interfaces.
- REST APIs: Are APIs that used as northbound interface to allow communication between the controller and applications such as static routing, topology manager, and so on.
- Karaf: Karaf runs on top of OSGI which simplifies package operations and application installations.
- YANG. All applications, notifications, and remote procedure calls manipulate configuration and state data which are modeled using YANG data modeling language.

Applications contain algorithms and business logic. The applications that run over the controller use northbound APIs to communicate with the controller. Among others, OSGI and representational state transfer (REST) (web-based) [FT02], are the two APIs supported in the northbound. If the controller and the applications have the same address, OSGI framework is used as a northbound API. On the other hand, when the controller run either in another system than used by application or has different address space, then REST API is used; this is a bidirectional API.

On the other hand, the controller is responsible for running the algorithms, collecting network intelligence, and finally forwarding the rules over the whole network through southbound protocols (for example, OF1.0/1.3, BGP, NETCONF) [Com16c]. All the southbound plug-ins are dynamically linked to the SAL. As a result, the services corresponding to the modules are exposed through the SAL. The SAL to handles an incoming service regardless of the southbound protocol. Hence, investment cost is reduced even if new protocols evolve through time [Com16c].

Furthermore, ODL enables data and applications to be accessed externally via the following model-driven protocols [Com16a]:

- NETCONF: NETCONF allows clients to invoke RPCs that are modeled based on YANG, retrieve notifications, and manipulate data which is modeled using YANG.
- RESTCONF: Is a protocol based on HTTP that enables manipulating of data which is modeled using YANG through REST-like APIs. It uses either XML or JavaScript Object Notation (JSON)[5] to retrieve RPCs.

[5]http://json.org/

4.3.2 MD-SAL Basics

When NETCONF is used as a southbound plug-in to connect with remote NETCONF based devices (in our case, IHON nodes in general and H1 nodes in specific), the datastores, RPCs, and notifications of these devices are exposed as MD-SAL mount points. As a result, application and remote users can successfully interact with such kind of devices through RESTCONF protocol. The MD-SAL is an extensible message-bus which stores data and perform messaging depending on the interface models and data declared by application developers. The tasks of MD-SAL are summarized as follows [Com16a]:

- Contains definitions for data model building blocks, concepts, messaging patterns, and common-layer. Additionally, the communication framework between applications and inter-applications is supplied by MD-SAL.
- Provides common framework such as XML, and JSON for different payload formats and user-defined transport, and payload adaptation and serialization.

Depending on the YANG models supplied from a developer, the MD-SAL applies Basic concepts to provide definitions for message patterns, services and behavior. Basic concepts are basically building blocks for which applications use. Below are listed the common Basic concepts [Com16a]:

- Data Tree: Used to represent and model all state-related data, and any element or subtree is accessible. There are two classes of data Trees:
  1. Configuration Data Tree: Is the network or system intended state which is advertised by consumers.
  2. Operational Data Tree: Unlike the configuration data tree, operational data tree is the system's reported state and is populated by providers through MD-SAL. This is used by applications as a feedback loop to check the system/network status.
- Instance Identifier: Within a data tree the nodes and subtrees are uniquely identified using an instance identifier. It also gives exact information for distinguishing and accessing node/subtree from existing data trees.
- Notification: A transient event which is an asynchronous type of message. Moreover, subscribers can consume notification and manipulate it.
- RPC: A request-reply message sequence which is also asynchronous type; consumer initiates a request, forward it to the provider, and responds with reply message. Moreover, RPC is used to define the input and output of MD-SAL.

MD-SAL transactions are provided by MD-SAL data broker in order to access conceptual data trees which represent both operational and configuration state. The state of modeled data is given as a data tree which represents the state of the applications, controller, and network/system [Com16a].

4.4 RESTCONF

Using standard techniques (for example, RESTCONF), web applications can access the operational and configuration data, notifications, and data-model operations of a device in the network [ODLb]. RESTCONF is a northbound protocol which allows communication between applications (or remote users) and SDN controller (in our case, ODL) as depicted in Figure 4.2. Moreover, RESTCONF is REST-like protocol that runs over HTTP to access YANG based data by applying NETCONF datastores [Wat15].

As discussed in 3.2.3, configuration datastores are defined using NETCONF protocol. In addition, NETCONF provides basic operations (i.e. retrieve, create, delete, and update operations) so as to access the datastores. On the other hand, YANG language provides the semantics and syntax definitions for operational data, datastore content, event notifications, and finally protocol operations. As a result, RESTCONF implements HTTP to connect with the controller, and the controller translates the requests to manage the NETCONF datastores. In order to accomplish this, the "Content-Type" field in the HTTP header is used to set the request (input) media type whereas the "Accept" field is used to set the response (output) media type [Wat15].

In ODL, the *sal-rest-connector* feature contains an implementation for RESTCONF, and is installed in the karaf distribution [ODLb]. *karaf debug* can be used to start RESTCONF in debug mode. By doing so, remote debugging can be performed by connecting to port 5005. Depending on the ODL controller version, RESTCONF listen on different ports. In Beryllium version, RESTCONF listens HTTP requests on port 8181.

4.4.1 Supported Operations

RESTCONF supports GET, PUT, OPTIONS, DELETE, and POST operations [ODLb] which are described in Table 4.2. Every request URI for the restconf calls has to start with prefix /restconf.

The RESTCONF URI (endpoint paths) implements instance identifier which is represented as *<identifier>*. The *<identifier>* should fulfill the following requirements [ODLb]:

– It must begin with *<moduleName>:<nodeName>*; in this case *<moduleName>* and *<nodeName>* are YANG module and top level node names, respectively.
– *<nodeName>* indicates a data node. The data node can be either container or list yang type. When the data node is a list, the key names of the list must be inserted after the node name, for example, *<nodeName>/<Key1Value>/<Key2Value>*.

– <moduleName>:<nodeName> is also used to solve for ambiguity.

It is possible that a node can reside after a mount point, and the URI format is *<identifier>/yang-ext:mount/<identifier>* [ODLb]. While the path to a mount point is given in the first *<identifier>*, the last *<identifier>* indicates the path of the node located behind the mount point. Moreover, we use *<identifier>/yang-ext:mount* when the URI ends in the mount point.

Table 4.2: Basic RESTCONF operations together with their formats and purposes, adopted from [ODLb].

HTTP operation	URI link	Description
OPTIONS	/restconf	– gives the XML definition of resources along with the necessary media types of both request and response in Web Application Description Language (WADL).
GET	/restconf/config /<identifier>	– Retrieves a data node value within the Config datastore. – Data node is identified using the <identifier>.
GET	/restconf/operational /<identifier>	– Accesses the data node value from the Operational datastore. – <identifier> indicates the data node to be retrieved.
PUT	/restconf/config /<identifier>	– Data in the config datastore is either created or updated. – Success status is then returned.
POST	/restconf/config	– Unless the data is available, it will be created.

POST	/restconf/config /<identifier>	– If config datastore does not contain the data, it will be created and success status is returned. – Data root element has either namespace (XML type of data) or module name (JSON type of data).
POST	/restconf/operations /<moduleName>: <rpcName>	– Used to invoke RPC. – <moduleName> and <rpcName> represent the module and RPC names in the module, respectively. – The name "input" must be assigned to the data root element forwarded to RPC. – Either status code or data with "output" root element is returned as a response.
DELETE	/restconf/config /<identifier>	– Data node which is available in the Config datastore is removed, and status of success will be returned – <identifier> distinguishes the data not to be removed.
GET	/restconf/streams	– Provides the list of existing event notification streams.
GET	/restconf/streams /stream /<stream_name>	– Receives notifications by subscribing to the stream_name.

4.4.2 RESTCONF Tools

RESTCONF requests are sent from the applications running on top of the controller using RESTCONF tools such as cURL utility [Ste] and Chrome Postman plug-in [Bro15]: in our case, we will use Postman plug-in.

Chrome Postman plug-in

Postman is an HTTP client used for testing web services. Moreover, Postman allows to test, document, and develop APIs which enables users to easily and quickly put simple to complex HTTP requests. By installing the postman plug-in, the request is sent to the URL: *http://<controller-ip>:8181/URI.* Then, we execute the RESTCONF request based on the following steps [Bro15]:

1. By locating the **authorization** tab, we select the **Basic Auth** type, and put admin as username and password.
2. Selecting the required headers: Since the input and output for restconf requests are both XML type, we select the following type of headers:
   – Accept: application/xml value.
   – Content-type: application/xml.
3. Choosing the Rest method; GET, POST, PUT, or DELETE operation.
4. Finally, we put the configuration file in an XML format, and send it to the specified URL.

Postman plug-in is more user interactive, and it allows to save restconf requests for latter use, and it is easier to modify requests.

To sum up, in this chapter we discussed the ODL platform testbed, its architecture, and basic features. Additionally, we presented RESTCONF protocol and supported operations that allow an application to manage the configuration and operational data of remote netconf devices. To accomplish this, we have presented RESTCONF tools focusing on the postman plug-in to execute restconf operations. In the next chapter, we will apply the concepts that we developed so far, and propose an SDN framework for IHON.

Chapter 5

SDN framework for IHON

This chapter presents SDN framework for IHON in general and Fusion nodes in particular. In Section 5.1, we evaluate NETCONF and OF protocol as a southbound protocol to realize SDN for IHON nodes. Following the evaluation, we discuss the SDN framework for IHON nodes with the ODL platform in Section 5.2. After that, the classifications of IHON nodes depending on their support for NETCONF monitoring, and realizing SDN for all types of nodes are discussed in Section 5.2.1, Section 5.2.2, and Section 5.2.3.

5.1 Evaluating NETCONF- and OF-based SDN for IHON nodes

The objective of the project is to enable SDN for IHON focusing on Fusion H1 nodes. Either OF or NETCONF protocol can be implemented as a southbound interface to realize SDN for IHON nodes.

When OF is used as a southbound interface to realize SDN for IHON nodes, the nodes should allow OF based communication with the controller. First, an OF agent which detects OF protocol should be added in every node as OF protocol operates with OF capable switches [GDS+10]. Thus, with the help of the agent, the IHON nodes can understand OF rules, and perform forwarding based on the rules from the controller.

Second, IHON nodes completely integrate both GST and SM flows. The OF specification [ONF13a] which defines rules only for packet flows should be extended to support the circuit flows (i.e. the GST flows) as well [Das10]. As a result, OF protocol requires further extension to support complete realization of SDN for IHON nodes [CNRF+13]. In OF-based SDN framework, flowtables are first created in each node, then an incoming flow is matched against the flowtables. After matching, the switch takes action on the flow based on the rules from the controller. Since IHON nodes integrate both GST and SM flows, the number of flowtables in a node grow radically as have been discussed in Section 4.2 of [Ber15]. Hence, matching an arriving flow against a large number of flowtable increases cost and complexity.

On the other hand, NETCONF is more easier to implement SDN for IHON nodes. Fusion H1 nodes, the IHON prototypes from Transpacket are already available with NETCONF, so we have chosen NETCONF as a southbound plug-in. Therefore, NETCONF-based SDN for IHON is better with respect to cost and complexity as compared with OF-based SDN framework. The evaluation of OF- and NETCONF-based SDN for IHON nodes is briefly presented in Section 4.3 of [Ber15].

5.2 SDN/NETCONF setup for IHON

Based on the evaluation, we employ NETCONF as a southbound interface in designing SDN framework for IHON nodes. Figure 5.1 illustrates an SDN layout consisting of the three main SDN layers: application, control, and infrastructure layers. Application layer is a collection of end-user applications while the control plane consists of SDN controller. On the other hand, the infrastructure layer or data plane is composed of the underlying network devices, in this case, the IHON nodes. Since NETCONF protocol is used as a southbound plug-in to connect the controller with the nodes, the controller should have NETCONF capability; this is indicated as NETCONF client within the controller.

In our case, the chosen testbed, i.e. the ODL controller supports multiple plug-ins both as a southbound and northbound protocols. Among others, NETCONF protocol is one of the common protocols that the ODL uses as a southbound plug-in to connect with remote netconf devices. Hence, ODL supports NETCONF client capability. Moreover, NETCONF enables the controller to discover the node datastores (i.e. configuration, operational, or both), notifications, and RPCs as MD-SAL mount points. Consequently, Applications and remote users interact with ODL controller via RESTCONF protocol, and access the remote nodes as MD-SAL mount points [Com16c] [Bur15].

The NETCONF southbound plug-in in ODL is commonly known as netconf-connector. It is based on YANG modeling language, and is fully model driven plug-in. Moreover, any YANG modeled type of data, notifications, and RPCs that are implemented in the nodes are allowed as well [Com16c]. In order to enable the NETCONF southbound in the ODL, we install the netconf-connector feature with the following command [Bur15]:

```
opendaylight-user@root>feature:install odl-netconf-connector-all
```

The default ODL configuration includes a controller-config which is a single instance of the netconf-connector. Using RESTCONF request, new netconf-connector

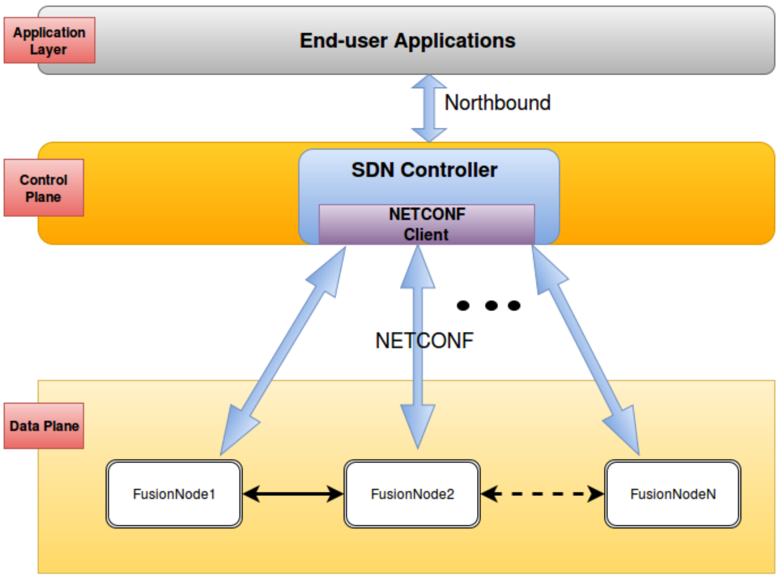


Figure 5.1: SDN framework based on NETCONF protocol. The control plane consists of an SDN controller with built-in NETCONF client, and manges the IHON nodes in the data plane via NETCONF southbound plug-in.

is created for every additional node to allow connection with the ODL controller. If the nodes support netconf monitoring, thus the netconf-connector can download and list the YANG schema used by the nodes [Bur15]. In order to allow communication, the netconf connector should know all or part of the schema of the IHON nodes. Even though the nodes use YANG schema internally, they may or may not support netconf monitoring. By sideloading the YANG schema of the IHON nodes into the YANG model cache of the ODL, the netconf-connector can communicate with the nodes which do not even support netconf monitoring. Overall we have three situations:

5.2.1 IHON nodes that Support NETCONF monitoring

While the ODL controller is running, a new netconf-connector should be created to connect each node with the controller. In our case, we have N nodes in the data plane as depicted in Figure 5.1, so we create N number of netconf-connectors corresponding to each node. First, we should install netconf-connector feature in the ODL karaf distribution which enables automatic configuration and activation of the loopback NETCONF mountpoint.

Following this, the netconf-connector for each node is configured by sending POST request via RESTCONF with the following parameters [Com16c]:

- Operation: POST,
- URL:*http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules*,
- Headers:
  - Accept: application/XML,
  - Content-Type: application/XML,
- Body: Given in Appendix C.1; part of the code which contains the node parameters is shown in source code 5.1 for explanation purpose. As we can see from the code, a new netconf-connector is created by specifying the node name, IP address, port number, username, and password of the node. The device-name is assigned by the user, and serves as unique name identifier for the node. Moreover, device_port is the port number on which the node is to be connected with the controller. We also need to replace the device_IP_address, username, and password with the actual values of the node.

Source code 5.1 Creating Netconf-connector for node

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:
    controller:config">
  <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:
      controller:md:sal:connector:netconf">prefix:sal-
      netconf-connector</type>
  <name>device-name</name>
  <address xmlns="urn:opendaylight:params:xml:ns:yang:
      controller:md:sal:connector:netconf">
      device_IP_address</address>
  <port xmlns="urn:opendaylight:params:xml:ns:yang:
      controller:md:sal:connector:netconf">
      device_port_number</port>
  <username xmlns="urn:opendaylight:params:xml:ns:yang:
      controller:md:sal:connector:netconf">username</
      username>
  <password xmlns="urn:opendaylight:params:xml:ns:yang:
      controller:md:sal:connector:netconf">password</
      password>

  <!--Remaining part of the code--!>

  </module>
```

Once the netconf-connector is correctly created, it is stored in the configuration datastore of the ODL controller. After the creation of the new netconf-connector, a useful metadata is written into the MD-SAL datastore in a subtree of the network-topology. The metadata contains information such as node capabilities and connection status.

Moreover, the netconf-connector can be reconfigured if any of the node parameters are changed while the controller is running. This is achieved by executing PUT operation with the whole configuration along with the updated parameters. For example, if username or password is changed, we send PUT request with the payload given in Appendix C.2.

5.2.2 IHON nodes that do not support NETCONF monitoring, but list YANG models

IHON nodes can list their YANG models during capability exchange in the HELLO message, but may not support ietf-netconf-monitoring capability. If these type of nodes are possibly using only the 2010-09-24 revision of ietf-inet-types YANG model, the reported HELLO message consists of: *urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-types&revision=2010-09-24* [Com16c]. To enable communication between the ODL controller and such nodes, the YANG schema of the nodes is placed in the cache/schema folder in the ODL karaf distribution, and it must be named as ietf-inet-types@2010-09-24.yang since this is the required cache format name.

5.2.3 IHON nodes that neither support NETCONF monitoring nor list YANG models

Unlike the IHON nodes which list their YANG models, in this case the nodes do not have ietf-inet-types capability in the HELLO message. Since such type of nodes do not advertise their YANG schema, the user is responsible for configuring a netconf-connector for them.

Moreover, the netconf-connector has yang-module-capabilities as an optional configuration attribute. This attribute contains a set of "YANG module based" capabilities [Com16c]. If the HELLO message of the node has "yang-module-based" capabilities, these capabilities can be overridden by setting the configuration attribute. As a result, the netconf-connector configuration given in Appendix C.1 is modified by appending the XML given in source code 5.2 after the username and password configuration elements. Afterwards, the YANG schema of the node is placed in the cache folder of the karaf distribution.

In conclusion, if the IHON nodes are modeled using YANG schema, they can be connected successfully to the SDN controller even if they do not support netconf monitoring by modifying the netconf-connector and storing their schema in the ODL karaf distribution.

Source code 5.2 Modification of netconf-connector configuration for a node that does not list its YANG models nor supports netconf monitoring [Com16c].

```
<yang-module-capabilities xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
  <capability xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-types&amp;revision=2010-09-24
  </capability>
</yang-module-capabilities>
```

To generalize, in this chapter we evaluated that the NETCONF protocol is more suitable than OF protocol to implement SDN for IHON in general and Fusion nodes in specific. As a result, we proposed a NETCONF-based SDN framework, and illustrated how to create a new netconf-connector for every node to allow connection with the ODL controller. After establishing successful communication, the controller manages the nodes via the southbound protocol. In the next chapter, we will discuss the emulation results of SDN controlled IHON nodes with the ODL testbed.

Chapter 6 SDN and Emulated IHON Testbed Results

IHON nodes are built based on NETCONF Protocol and YANG modeling language [VBB15]. Hence, IHON nodes use NETCONF as a southbound protocol to connect with the ODL controller, i.e. the Beryllium version. In Section 6.1 we discuss NETCONF-based SDN testbed for IHON network consisting of three nodes with different set of connections with different priorities. Section 6.1.1 presents about Netconf testtool and simulating the IHON nodes. In Section 6.1.2, we briefly discuss about YANG schema that represents the configuration of typical IHON node. While Section 6.1.3 describes running and starting of the simulated IHON nodes, Section 6.1.4 illustrates how to connect the nodes with the SDN controller by creating netconf-connector to each node. Finally, Section 6.2 describes management of the IHON nodes with the ODL testbed.

6.1 SDN framework for IHON

The framework of SDN for IHON nodes based on the NETCONF protocol, and illustrated in Figure 6.1, consists of three nodes (i.e NodeA, NodeB, and NodeC) in the data plane. The application layer on the other hand, consists a postman plug-in which communicates with the SDN controller via RESTCONF. The control plane is composed of the ODL platform. Moreover, the control plane provides a consolidated control functionality to configure, manage, and monitor the network forwarding behavior through NETCONF protocol. RESTCONF is a northbound protocol used to interface the application, i.e. postman plug-in, with the ODL controller whereas NETCONF is a southbound protocol which allows communication between the nodes in the data plane and the ODL controller in the control plane.

In our case, the control plane is realized with the Beryllium controller to supervise the configuration and forwarding behavior of the three nodes in the data plane. As we have discussed in Section 5.2, the ODL controller supports NETCONF protocol as a southbound plug-in; we named it as "NETCONF client" only for

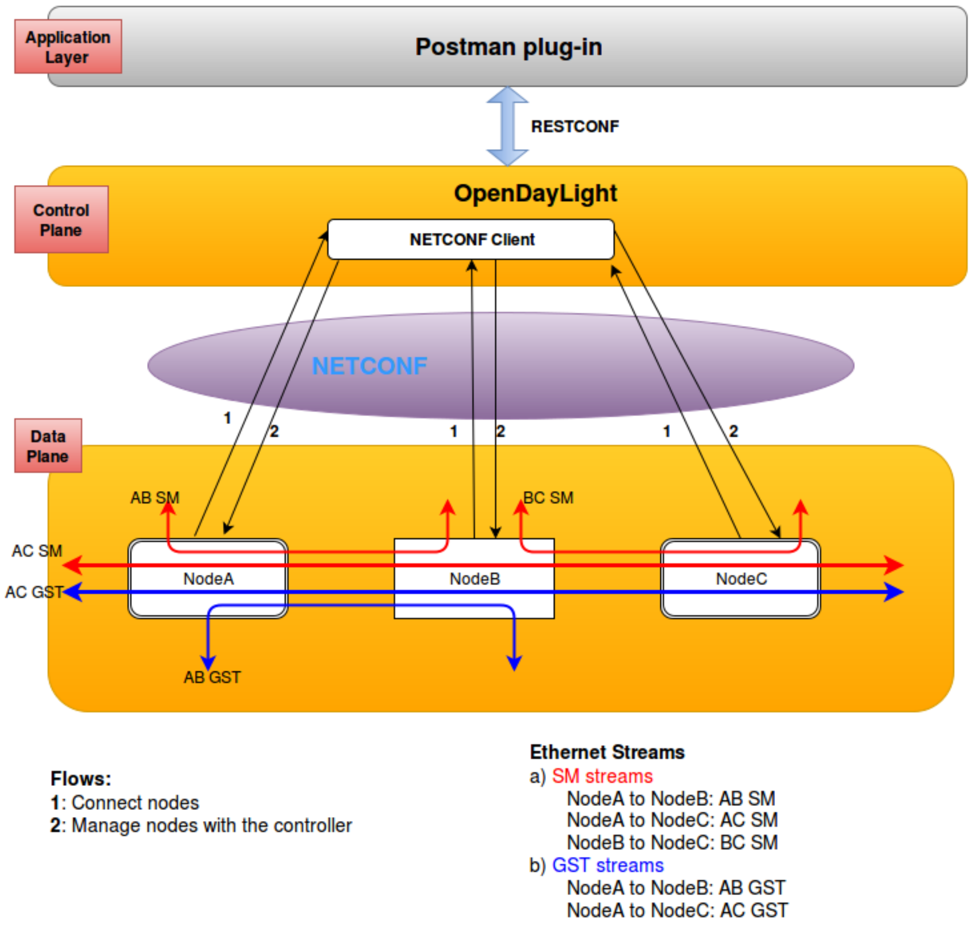


Figure 6.1: SDN setup based on NETCONF protocol with three IHON nodes in the data plane. Only unidirectional (left-to-right) Ethernet streams are considered. Red and blue correspond to SM and GST streams, respectively. IHON nodes are first connected to the ODL controller (indicated as flow 1). Flow 2 indicates management of the nodes from the controller.

readability purpose. Otherwise, it is the netconf-connector feature that should be installed in the Beryllium karaf distribution (using the *"odl-netconf-connector-all"* command) to support NETCONF based communication with the nodes. Moreover, the RESTCONF protocol is activated by installing *"odl-restconf"* feature in the karaf distribution. As a result, end-to-end orchestration and control is, thus, easily accomplished using the Beryllium testbed.

In Section 3.2.2, we have briefly discussed the steps to enable NETCONF over SSH session. In this SDN setup, the NETCONF session between the ODL controller and the three simulated nodes is established in similar fashion. To achieve this, SSH is first activated by installing the "odl-netconf-connector-ssh" feature in the ODL controller karaf distribution. Therefore, the communication between the controller and a single IHON node is depicted in Figure 6.2. While the client in the diagram corresponds to the ODL controller, the server represents an IHON node. Before exchanging their capabilities, both peers perform key exchange, and establish netconf session by invoking the netconf as a subsystem. Having clear awareness of each others capabilities, the application running on top of the controller use restconf protocol to interface with the SDN controller so that the user can send the commands from the controller to the IHON nodes through Netconf. Finally the controller closes the session after finishing execution.

In the the following sections, we discuss the core procedures to realize SDN controlled IHON through emulation. We present management of the IHON through the ODL testbed, i.e. Beryllium controller.

6.1.1 Step1: Simulating IHON nodes with Netconf Testtool

Netconf testtool is a tool that allows to simulate one or more NETCONF devices [ODLc]. Testtool is the most convenient application used to test large set of devices. Moreover, testtool generates configuration files for the ODL controller. Since IHON nodes are built based on NETCONF protocol and YANG modeling language, each node in the SDN setup is simulated with the netconf testtool. As a result, the Beryllium controller is able to connect and manage the configuration of the simulated nodes.

Simulating IHON with the testtool allows us to execute the following set of NETCONF operations remotely from the controller [ODLc]:

– get-schema: Yang schema that are loaded from user specific directory are returned using the get-schema operation.
– edit-config: Allows to edit the configuration of the nodes, and saves the xml derived from input in a local variable which can be then easily accessed by

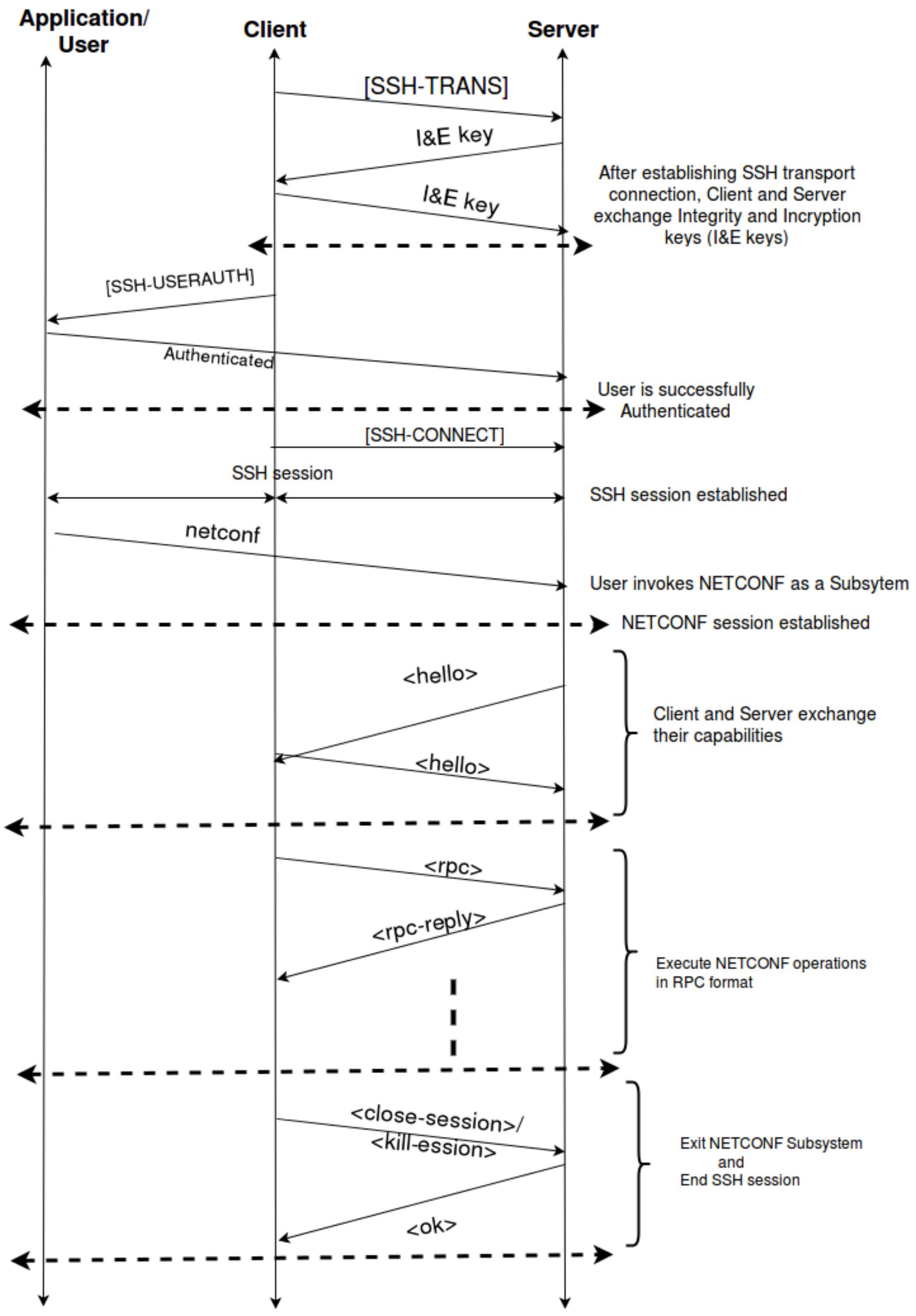

Application/ User
Client
Server
[SSH-TRANS]
I&E key
I&E key
After establishing SSH transport connection, Client and Server exchange Integrity and Incryption keys (I&E keys)
[SSH-USERAUTH]
Authenticated
User is successfully Authenticated
[SSH-CONNECT]
SSH session
SSH session established
netconf
User invokes NETCONF as a Subsytem
NETCONF session established
<hello>
<hello>
Client and Server exchange their capabilities
<rpc>
<rpc-reply>
Execute NETCONF operations in RPC format
<close-session>/ <kill-ession>
<ok>
Exit NETCONF Subsystem and End SSH session

Figure 6.2: Establishing NETCONF session between client and server. In this case, the client is the ODL controller while the server is the IHON node. After establishing connection, both peers exchange their capabilities using hello messages. Afterwards, NETCONF operations are executed as RPC models, and the sessions is closed from the controller.

get-config and get-rpc. Moreover, edit-config operation replaces the already existing configuration.

- commit: Although commit operation does not practically commit the data, it returns OK.
- get-config: Retrieves the local configuration in xml format which is stored by edit-config.
- get: like the get-config operation, get returns xml configuration file stored by edit-config. However, it also includes netconf-state subtree and supports filtering.
- (un)lock: Used to unlock or lock the configuration of the nodes, and returns OK.
- create-subscription: Any existing netconf notifications are provided to the client once this operations is initiated. However, neither stream recognition nor filtering is supported by this operation.

In order to simulate the IHON nodes, we first build the netconf testtool based on the following steps:

1. Netconf repository is pulled using: git clone *https://git.opendaylight.org/gerrit/netconf*,
2. We changed our working directory to netconf/netconf/tools/netconf-testtool/ folder to access the netconf testtool.
3. Finally, we use **mvn clean install** command to build the testtool.

By this time, the testtool is ready, and we have to design, implement and load the YANG schema of the IHON nodes; the YANG schema is described in the following section.

6.1.2 Step2: Developing YANG Schema for IHON Nodes

IHON node has two 10Gb/s (10GE) line interfaces, and ten 1GE client interfaces. The line interfaces are also called trunk-interfaces which implies that they aggregate and transport a group of VLANs. Similarly, the client interfaces are known as access-interfaces since each interface can transport a single VLAN. Furthermore, the access-interfaces transport either SM stream or GST sub-wavelengths to improve the channel utilization. A detailed network connection of the IHON nodes in the data plane for the SDN setup is illustrated in Figure 6.3; it shows the network connections by VLAN in each node with their corresponding interfaces. Moreover, only left-to-right Ethernet streams are assumed in this scenario. Taking a specific VLAN, the one in the left side is considered as a source while the VLAN at the right side with the same name but in different site implies its destination (for example, Vlan1 in siteX is a source whereas Vlan1 in siteY is its destination). Furthermore, all ge(i.e. ge[0-9]) interfaces are access interfaces. Moreover, blue indicates GST while red implies SM Ethernet streams.

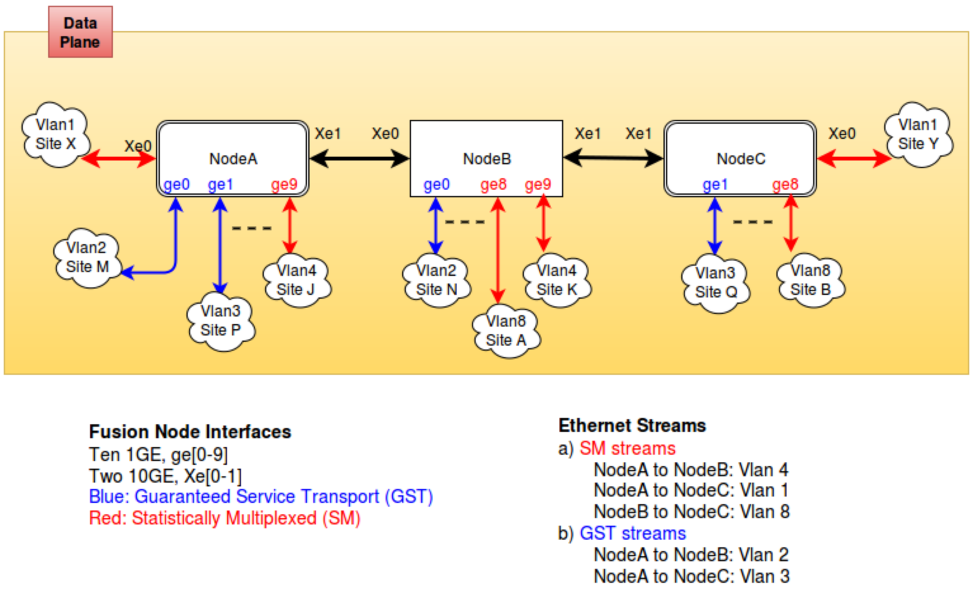


Figure 6.3: Network connection of the IHON nodes in the data plane. Each node transports GST and SM streams from different connections which are depicted as blue and red, respectively. Moreover, only left-to-right flow is considered.

The Ethernet streams from the different VLANs are forwarded out their destination based on VLAN ID. The IHON node appends VLAN ID to an incoming Ethernet stream, i.e. either GST or SM stream. VLAN ID identifies the priority of the stream, port, VLAN, and destination node. Hence, a node which receives the Ethernet stream first checks the corresponding VLAN ID. If the destination lies within the node, the node strips off the VLAN ID and forwards the stream to the corresponding output port. Otherwise, the node passes the Ethernet stream to the next node. The forwarding techniques of Ethernet streams in IHON, is briefly discussed in Section 4.1 of [Ber15].

NodeA is composed of four different VLANs (i.e. Vlan1, Vlan2, Vlan3, and Vlan4) from different sites each connected via four different interfaces. Vlan1 and Vlan4 are SM streams. Vlan1 is connected via Xe0 access interface; its source is SiteX, and is delivered to SiteY through Xe0 of NodeC. Similarly, Vlan4 which originates from SiteJ is connected via ge9, and is delivered to SiteK via ge9 interface of NodeB. On the other hand, Vlan2 and Vlan3 which are connected via ge0, and ge1 of NodeA, respectively, are both GST streams. They are forwarded to siteN via ge0 of NodeB,

Table 6.1: Connections in the network given by VLAN source and destination addresses.

VLAN	Path			
	Source		Destination	
	Node	Port	Node	Port
Vlan1	NodeA	Xe0	NodeC	Xe0
Vlan2	NodeA	ge0	NodeB	ge0
Vlan3	NodeA	ge1	NodeC	ge1
Vlan4	NodeA	ge9	NodeB	ge9
Vlan8	NodeB	ge8	NodeC	ge8

and SiteQ via ge1 of NodeC, respectively. Finally, Vlan8 which is an SM stream from siteA is connected via ge8 of NodeB, and is delivered towards SiteB through ge8 of NodeC. All the VLANs in the SDN setup are summarized in Table 6.1 with their corresponding source and destination addresses.

Furthermore, the Xe0 interface of both NodeA and NodeC transport only a single VLAN (i.e. Vlan1). On the other hand, Xe1 of NodeA transports Vlan1, Vlan2, Vlan3, and Vland4 from SiteX, SiteM, SiteP, and SiteJ, respectively. Similarly, Xe0 of NodeB receives all the VLANs from Xe1 of NodeA. On the other hand, Xe1 of NodeB aggregates Vlan1, Vlan3, and Vlan8 and transports them to NodeC. Xe1 of NodeC receives Vlan1, Vlan3, and Vlan8 from NodeB. Lastly, NodeC dispatches the streams to their respective destinations. All VLANs along with the corresponding interfaces of the IHON nodes are summarized in Table 6.2.

Therefore, we developed a YANG schema which is given in Appendix B.1 to model the typical configuration of IHON nodes. As we can see from the schema, each node is required to have an assigned name. In our case, NodeA, NodeB, and NodeC are the given names based on the SDN setup. Since each node contains ten 1GE, and two 10GE interfaces, we provide a list node to define the interfaces that are connected to the VLANs in each node based on the SDN setup, for example, ge0, ge1, ge9, and xe0 of NodeA. The interface definition of the IHON node from Appendix B.1 is illustrated in Source code 6.1. Each interface in the node is identified by its unique name, so we set the name as a key element in the list. Furthermore, the interface is either trunk-interface or access-interface. Trunk interface transports a group of VLANs while the access interface transports single VLAN which is either GST or SM type of Ethernet stream. Hence, we provide two options to allow selection of the interface type along with the carried VLANs and type of Ethernet stream. This is achieved using the Ethernet-switching choice type definition. Additionally, we provide VLAN container definition to include all the group of VLANs in a node

Table 6.2: Configuration plan for IHON nodes in the SDN setup. Each node's interface transports either SM or GST Ethernet stream. Moreover, each interface transports either single or multiple VLANs. "-" indicates that the interface is free.

Node	Interfaces						
	ge0	ge1	ge8	ge9	Xe0	Xe1	
	GST	GST	SM	SM	SM access-interface except in NodeB which is Trunk	Trunk	Type of Ethernet Stream
NodeA	Vlan2	Vlan3	-	Vlan4	Vlan1	Vlan1 + Vlan2 + Vlan3 + Vlan4	VLAN
NodeB	Vlan2	-	Vlan8	Vlan4	Vlan1 + Vlan2 + Vlan3 + Vlan4	Vlan1 + Vlan3 + Vlan8	
NodeC	-	Vlan3	Vlan8	-	Vlan1	Vlan1 + Vlan3 + Vlan8	

along with their unique identification number; it is shown in Source code 6.2.

In order to sideload the YANG schema of the IHON node to the simulation testtool, the schema is saved as *filename@YYYY-MM-DD.yang* since this is the required naming format. The *YYYY-MM-DD* implies the revision number of the proposed YANG schema. The file is then stored in the working directory so that netconf testtool can access the schema when it runs. In our case, the schema is saved as *ihonnode@2016-04-15.yang* in *ihon-schemas/* directory.

6.1.3 Step3: Starting the IHON Nodes

After building the netconf testtool, and developed the YANG schema of the IHON node, we run and start the testtool. As a result, the testtool can successfully simulate the nodes with the proposed YANG schema.

First, we dive to netconf/netconf/tools/netconf-testtool/target directory. We checked that an executable file for the testtool is created, and it implies that the testtool is successfully built. Afterwards, the three simulated nodes are started from this directory with the following command: *java -Xmx1G -XX:MaxPermSize=256M -jar netconf-testtool-1.1.0-SNAPSHOT-executable.jar –device-count 3 –distribution-folder*

Source code 6.1 YANG schema of IHON node interface

```
list interface {
    key name;
    leaf name{
        type string;
    }
    leaf describe {
        type string;
    }
    leaf type {
        type string;
    }
    leaf mtu {
        type uint32;
    }
    choice ethernet-switching {
        case access-interface {
            container access-interface {
                list vlan {
                    key vlan-name;
                    leaf vlan-name {
                        type string;
                    }
                    leaf priority {
                        type enumeration {
                            enum gst;
                            enum sm;
                        }
                    }
                }
            }
        }
        case trunk-interface {
            container trunk-interface {
                container vlans {
                    list vlan {
                        key vlan-name;
                        leaf vlan-name {
                            type string;
                        }
                    }
                }
            }
        }
    }
}
```

Source code 6.2 YANG schema for listing the VLANs within IHON node

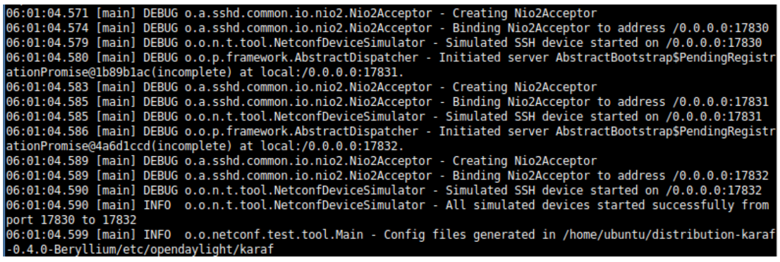
```
container vlans {
    description "Defines a group of VLANs in a Node";
    list vlan {
        key id;
        leaf id {
            type uint32;
        }
        leaf name {
            type string;
        }
    }
}
```

/distribution-karaf-0.4.0-Beryllium/ –debug true –schemas-dir ∼/ihon-schemas/. In this case, the executable jar file is the netconf testtool which actually simulates the IHON nodes. Moreover, *device-count 3* indicates that three nodes depicted in the SDN setup are to be simulated with the netconf testtool. Moreover, the testtool fetches the YANG schema of the IHON nodes from the specified directory(i.e. */ihon-schemas/*). Furthermore, the testtool is directed to the Beryllium distribution; the distribution-folder parameter enables the testtool to modify the controller by adding the netconf-connector configurations. This allows to automatically connect the simulated IHON nodes to the Beryllium controller. As a result, the user does not require to send configuration for each simulated node through restconf every time the nodes start.

Second, by starting the Beryllium distribution, we installed both odl-netconf-connector-ssh and odl-restconf features. The ssh feature enables SSH communication between the emulated nodes and the controller while the restconf feature allows to send restconf requests from applications running on top of the controller. When netconf uses SSH protocol, the netconf testtool requires large resources [ODLc]. As a result, it is necessary to reserve memory for the testtool. This is accomplished by specifying the memory size of the testtool. In our case, the command *-Xmx1G -XX:MaxPermSize=256M* implies 256Mb of memory is reserved for the testtool.

After successful starting of the simulated nodes, the restconf status is checked using the URI: *http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/*. As a result, the restconf is active, and only the loopback connection which is the default controller-config is running. Moreover, the testtool runs with default values for all parameters, and the log output is depicted in Figure 6.4.

As we can see from the log output given in Figure 6.4, the three simulated nodes



```
06:01:04.571 [main] DEBUG o.a.sshd.common.io.nio2.Nio2Acceptor - Creating Nio2Acceptor
06:01:04.574 [main] DEBUG o.a.sshd.common.io.nio2.Nio2Acceptor - Binding Nio2Acceptor to address /0.0.0.0:17830
06:01:04.579 [main] DEBUG o.o.n.t.tool.NetconfDeviceSimulator - Simulated SSH device started on /0.0.0.0:17830
06:01:04.580 [main] DEBUG o.o.p.framework.AbstractDispatcher - Initiated server AbstractBootstrap$PendingRegistr
ationPromise@1b89b1ac(incomplete) at local:/0.0.0.0:17831.
06:01:04.583 [main] DEBUG o.a.sshd.common.io.nio2.Nio2Acceptor - Creating Nio2Acceptor
06:01:04.585 [main] DEBUG o.a.sshd.common.io.nio2.Nio2Acceptor - Binding Nio2Acceptor to address /0.0.0.0:17831
06:01:04.585 [main] DEBUG o.o.n.t.tool.NetconfDeviceSimulator - Simulated SSH device started on /0.0.0.0:17831
06:01:04.586 [main] DEBUG o.o.p.framework.AbstractDispatcher - Initiated server AbstractBootstrap$PendingRegistr
ationPromise@4a6d1ccd(incomplete) at local:/0.0.0.0:17832.
06:01:04.589 [main] DEBUG o.a.sshd.common.io.nio2.Nio2Acceptor - Creating Nio2Acceptor
06:01:04.589 [main] DEBUG o.a.sshd.common.io.nio2.Nio2Acceptor - Binding Nio2Acceptor to address /0.0.0.0:17832
06:01:04.590 [main] DEBUG o.o.n.t.tool.NetconfDeviceSimulator - Simulated SSH device started on /0.0.0.0:17832
06:01:04.590 [main] INFO  o.o.n.t.tool.NetconfDeviceSimulator - All simulated devices started successfully from
port 17830 to 17832
06:01:04.599 [main] INFO  o.o.netconf.test.tool.Main - Config files generated in /home/ubuntu/distribution-karaf
-0.4.0-Beryllium/etc/opendaylight/karaf
```

Figure 6.4: Log output for the simulated IHON nodes from the Beryllium karaf distribution which shows successful starting of the nodes at specific ports.

run in three different ports: the first node at 18730 (this is the default starting port), second node at 17831, and the third node at 18732. Additionally, the testtool uses ietf-netconf-monitoring, ietf-inet-types, and ietf-yang-types YANG models by default which implies that the testtool supports NETCONF monitoring and advertises ietf-inet-types capability during HELLO message exchange. These YANG models are immediately added by the ODL controller to the working directory (i.e. the *ihon-schema* directory). We also verified this by accessing the MD-SAL datastore of the Beryllium controller using the URL: *http://localhost:8181/restconf/operational/network-topology:network-topology/* (for example, the capabilities of simulated NodeA are given in Appendix D.2). As a result, each simulated nodes supports netconf monitoring which indicates that the SDN controller easily manges and monitors the node configurations.

Moreover, we verified that the simulated nodes are up and running with SSH tool. From the command line, we use the following command to connect with NodeA: *ssh admin@localhost -p 17830 -s netconf*. After authenticating the simulated node (in this case, any password is acceptable), the node returns its capabilities in the HELLO message in XML format followed by netconf end of message (]]>]]>) which is shown in Appendix C.3. As we can see from the HELLO message, the simulated node supports netconf monitoring and advertises its ietf-inet-type capability.

By default, the Beryllium karaf distribution is not automatically connected to the simulated nodes. In order to connect the nodes with the controller, we have to create netconf-connector for every node as described in the following section.

6.1.4 Step4: Connecting the IHON Nodes with ODL controller

Once the netconf testtool runs successfully, a netconf connector is created (indicated as flow 1 in the SDN setup depicted in Figure 6.1) corresponding to each node. This allows to connect the simulated nodes with the controller through Netconf protocol. Since we have started three simulated nodes on three different ports, we created three

Table 6.3: IP address and port number of the IHON nodes in the SDN setup

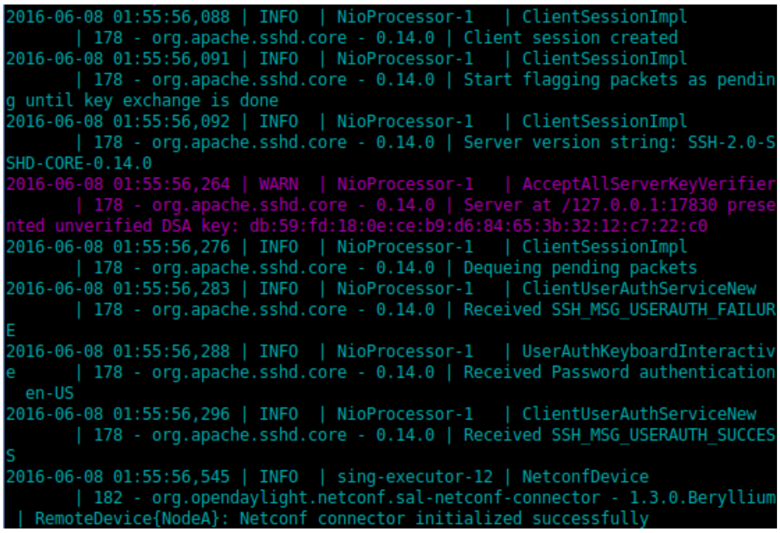
device-name	Port	IP address	Username and Password
NodeA	17830	127.0.0.1	admin
NodeB	17831	127.0.0.2	
NodeC	17832	127.0.0.3	

netconf-connectors corresponding to each node with their respective IP addresses and port numbers. Based on the given SDN network setup, the device names are NodeA, NodeB, and NodeC, and their corresponding IP addresses and port numbers are illustrated in Table 6.3. Moreover, admin is used as both username and password for all nodes. Having defined the parameters for the nodes, three netconf-connectors corresponding to the three IHON nodes are created one after the other by sending POST request through postman plug-in as follows:

- Method: POST,
- URL: *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/controller-config/yang-ext:mount/config:modules*,
- Headers: Accept:application/xml, and Content-Type:application/xml,
- Body: Using source code C.1, we created three different payloads corresponding to NodeA, NodeB, and NodeC. As we have explained in Section 5.2.1, the device-name, device_IP_address, device_port_number, username, and password in the source code are substituted with the corresponding parameters of the nodes as given in Table 6.3.

Hence, we initiated three netconf-connectors for NodeA, NodeB, and NodeC by executing POST request using their respective body. Consequently, issuing *log:display* in the Beryllium karaf distribution returns successful initialization of netconf-connectors for the three nodes, i.e. NodeA, NodeB, and NodeC; The screenshots taken from the log output are depicted in Figure 6.5, Figure 6.6, and Figure 6.7, respectively. We also verified this via restconf with the URL: *http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/*. As a result, NodeA, NodeB, and NodeC are added to the restconf which are illustrated in Appendix D.1, Appendix D.2, and Appendix D.3, respectively. As we can see from the result, each node advertises ietf-netconf-monitoring, ietf-yang-types, ietf-inet-types, and ihonnode (i.e. the developed YANG schema which is explained in Section 6.1.2) as their capabilities. This implies that each simulated node is internally modeled with the ihonnode YANG schema, and supports NETCONF monitoring.

By executing GET request, we also checked the operational data of the simulated nodes. At this level, the operational data for all three nodes is the same because we have not configured any of the nodes yet. For example, the operational data of NodeA is shown in Appendix D.4 after sending GET request to the



```
2016-06-08 01:55:56,088 | INFO  | NioProcessor-1   | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Client session created
2016-06-08 01:55:56,091 | INFO  | NioProcessor-1   | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Start flagging packets as pendin
g until key exchange is done
2016-06-08 01:55:56,092 | INFO  | NioProcessor-1   | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Server version string: SSH-2.0-S
SHD-CORE-0.14.0
2016-06-08 01:55:56,264 | WARN  | NioProcessor-1   | AcceptAllServerKeyVerifier
       | 178 - org.apache.sshd.core - 0.14.0 | Server at /127.0.0.1:17830 prese
nted unverified DSA key: db:59:fd:18:0e:ce:b9:d6:84:65:3b:32:12:c7:22:c0
2016-06-08 01:55:56,276 | INFO  | NioProcessor-1   | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Dequeing pending packets
2016-06-08 01:55:56,283 | INFO  | NioProcessor-1   | ClientUserAuthServiceNew
       | 178 - org.apache.sshd.core - 0.14.0 | Received SSH_MSG_USERAUTH_FAILUR
E
2016-06-08 01:55:56,288 | INFO  | NioProcessor-1   | UserAuthKeyboardInteractiv
e      | 178 - org.apache.sshd.core - 0.14.0 | Received Password authentication
  en-US
2016-06-08 01:55:56,296 | INFO  | NioProcessor-1   | ClientUserAuthServiceNew
       | 178 - org.apache.sshd.core - 0.14.0 | Received SSH_MSG_USERAUTH_SUCCES
S
2016-06-08 01:55:56,545 | INFO  | sing-executor-12 | NetconfDevice
       | 182 - org.opendaylight.netconf.sal-netconf-connector - 1.3.0.Beryllium
 | RemoteDevice{NodeA}: Netconf connector initialized successfully
```

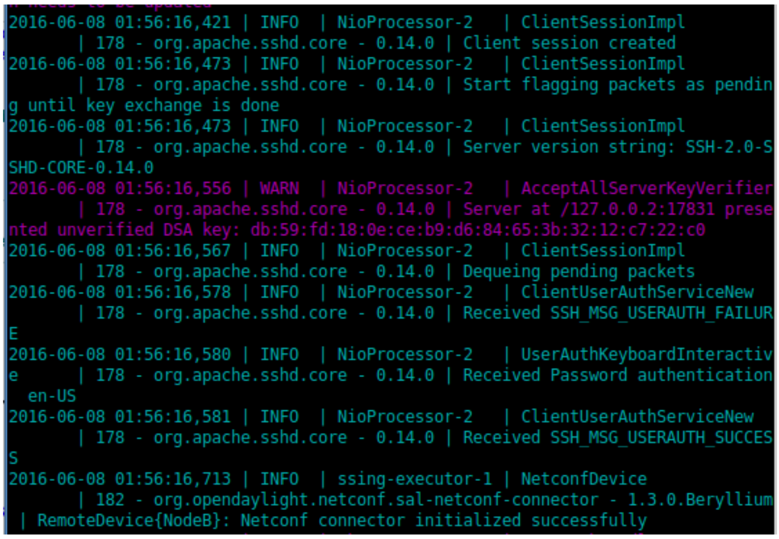
Figure 6.5: Log output from the Beryllium which indicates successful initialization of netconf-connector for NodeA.

following URL: *http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/NodeA/yang-ext:mount/*. From the output, we verified that each node supports netconf monitoring and uses the ihonnode YANG schema.

6.2 Management of IHON Nodes with the ODL Controller

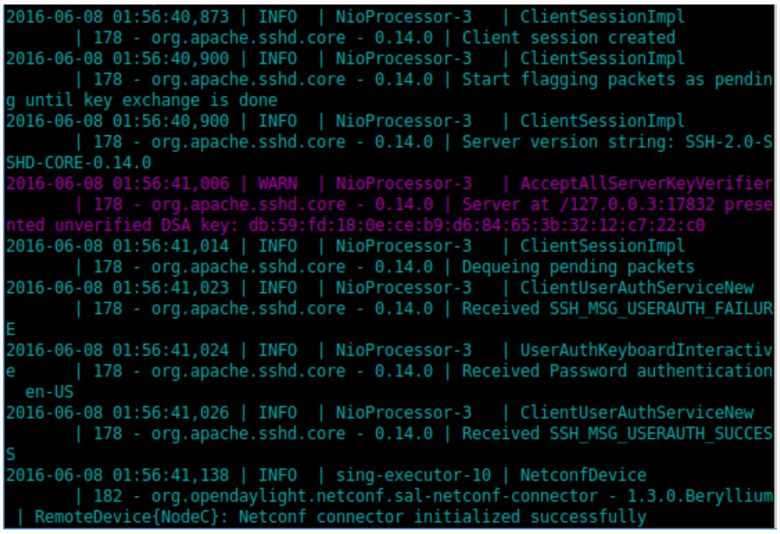
After connecting the simulated IHON nodes, the ODL controller supervises their configuration and forwarding scheme (which is indicated as flow 2 in the SDN setup). By accessing the config datastore of the simulated nodes, we executed operations on the nodes from the controller through Netconf and restconf from the user application (the controller interface to the user). In our case, postman plug-in is employed to send restconf requests. Both controller and restconf assume that YANG schema is used to represent the configuration data of the simulated nodes.

Having started the three simulated nodes, and created corresponding netconf-connectors, we use restconf request to communicate with the ODL controller. Then, the controller translates the commands sent from the restconf to Netconf and communicates with the nodes. In order to realize the SDN setup, the three nodes are configured according to the configuration plans given in Table 6.1 and Table 6.2. So, the configuration of each node includes list of interfaces, group of VLANs, and type of Ethernet streams. All the applicable interfaces in the nodes are listed including their assigned name, type of interface (i.e. access or trunk), and the stream type



```
2016-06-08 01:56:16,421 | INFO  | NioProcessor-2    | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Client session created
2016-06-08 01:56:16,473 | INFO  | NioProcessor-2    | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Start flagging packets as pendin
g until key exchange is done
2016-06-08 01:56:16,473 | INFO  | NioProcessor-2    | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Server version string: SSH-2.0-S
SHD-CORE-0.14.0
2016-06-08 01:56:16,556 | WARN  | NioProcessor-2    | AcceptAllServerKeyVerifier
       | 178 - org.apache.sshd.core - 0.14.0 | Server at /127.0.0.2:17831 prese
nted unverified DSA key: db:59:fd:18:0e:ce:b9:d6:84:65:3b:32:12:c7:22:c0
2016-06-08 01:56:16,567 | INFO  | NioProcessor-2    | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Dequeing pending packets
2016-06-08 01:56:16,578 | INFO  | NioProcessor-2    | ClientUserAuthServiceNew
       | 178 - org.apache.sshd.core - 0.14.0 | Received SSH_MSG_USERAUTH_FAILUR
E
2016-06-08 01:56:16,580 | INFO  | NioProcessor-2    | UserAuthKeyboardInteractiv
e      | 178 - org.apache.sshd.core - 0.14.0 | Received Password authentication
  en-US
2016-06-08 01:56:16,581 | INFO  | NioProcessor-2    | ClientUserAuthServiceNew
       | 178 - org.apache.sshd.core - 0.14.0 | Received SSH_MSG_USERAUTH_SUCCES
S
2016-06-08 01:56:16,713 | INFO  | ssing-executor-1 | NetconfDevice
       | 182 - org.opendaylight.netconf.sal-netconf-connector - 1.3.0.Beryllium
 | RemoteDevice{NodeB}: Netconf connector initialized successfully
```

Figure 6.6: Log output from the Beryllium which indicates successful initialization of netconf-connector for NodeB.



```
2016-06-08 01:56:40,873 | INFO  | NioProcessor-3    | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Client session created
2016-06-08 01:56:40,900 | INFO  | NioProcessor-3    | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Start flagging packets as pendin
g until key exchange is done
2016-06-08 01:56:40,900 | INFO  | NioProcessor-3    | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Server version string: SSH-2.0-S
SHD-CORE-0.14.0
2016-06-08 01:56:41,006 | WARN  | NioProcessor-3    | AcceptAllServerKeyVerifier
       | 178 - org.apache.sshd.core - 0.14.0 | Server at /127.0.0.3:17832 prese
nted unverified DSA key: db:59:fd:18:0e:ce:b9:d6:84:65:3b:32:12:c7:22:c0
2016-06-08 01:56:41,014 | INFO  | NioProcessor-3    | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Dequeing pending packets
2016-06-08 01:56:41,023 | INFO  | NioProcessor-3    | ClientUserAuthServiceNew
       | 178 - org.apache.sshd.core - 0.14.0 | Received SSH_MSG_USERAUTH_FAILUR
E
2016-06-08 01:56:41,024 | INFO  | NioProcessor-3    | UserAuthKeyboardInteractiv
e      | 178 - org.apache.sshd.core - 0.14.0 | Received Password authentication
  en-US
2016-06-08 01:56:41,026 | INFO  | NioProcessor-3    | ClientUserAuthServiceNew
       | 178 - org.apache.sshd.core - 0.14.0 | Received SSH_MSG_USERAUTH_SUCCES
S
2016-06-08 01:56:41,138 | INFO  | sing-executor-10 | NetconfDevice
       | 182 - org.opendaylight.netconf.sal-netconf-connector - 1.3.0.Beryllium
 | RemoteDevice{NodeC}: Netconf connector initialized successfully
```

Figure 6.7: Log output from the Beryllium which indicates successful initialization of netconf-connector for NodeC.

they carry along with the VLAN name.

In the following sections we discuss the configuration of the three nodes according to the developed YANG schema, i.e. the ihonnode YANG. By converting to its equivalent XML format, the configuration of every node is executed using postman by sending restconf requests to the controller. Then, the controller translates the commands and use Netconf southbound to configure the nodes.

Source code 6.3 Default configuration data of IHON nodes (an empty data container)

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
</data>
```

NodeA Configuration

To begin with, we executed GET request to: *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/NodeA/yang-ext:mount/* to view the node configuration. As a result, we checked that the config datastore of NodeA is merely an empty data container because configuration data is not yet sent to the node which is illustrated in Source code 6.3. Configuration of the node is then achieved with edit-config request which is executed by sending the following POST request using postman plug-in:

- URL: *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/NodeA/yang-ext:mount*,
- Headers:
  Accept: application/xml,
  Content-Type: application/xml
- Body: Given in Appendix B.2; it is the xml schema of NodeA configuration based on the configuration plan of the SDN setup.

After executing the POST request, 200 OK messages is returned with empty data which indicates that NodeA is successfully configured. GET request is, then, executed to: *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/NodeA/yang-ext:mount/* to retrieve the configuration data. From the response, we get the configuration of the NodeA as shown in Appendix B.5 which implies that the restconf operation successfully pushed the configuration from the ODL controller to the configuration datastore of the node.

NodeB Configuration

Before sending the configuration, we checked that the config data container for NodeB is empty using GET request to: *http://localhost:8181/restconf/config/*

opendaylight-inventory:nodes/node/NodeB/yang-ext:mount/. Like NodeA, the initial configuration for NodeB is given in Source code 6.3. Like NodeA, we executed POST request to edit the configuration of NodeB as follows:

- URL: *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/NodeB/yang-ext:mount*,
- Headers:
  Accept: application/xml,
  Content-Type: application/xml
- Body: Shown in Appendix B.3 which is the configuration of NodeB.

Like NodeA, the 200 OK is returned once the POST request for NodeB is executed; this implies that the configuration of NodeB is edited successfully. By executing GET request to *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/NodeB/yang-ext:mount/*, we accessed NodeB's configuration data as illustrated in Appendix B.6.

NodeC Configuration

Like both NodeA and NodeB, we follow the same procedure to achieve successful configuration. First we made sure that its config data container is empty as illustrated in Source code 6.3 by executing GET request to: *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/nodeC/yang-ext:mount/*. After that, we execute POST request to send its configuration as follows:

- URL: *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/NodeC/yang-ext:mount*,
- Headers:
  Accept: application/xml,
  Content-Type: application/xml
- Body: Illustrated in Appendix B.4 which is the configuration of NodeC.

After sending the configuration with postman, the 200 OK is returned which indicates that NodeC is configured successfully. Afterwards, we execute GET request to: *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/NodeC/yang-ext:mount/*, and we retrieved its configuration data which is given in Appendix B.7.

To sum up, we have emulated SDN for IHON nodes. ODL platform is a suitable testbed to realize SDN for IHON nodes based on NETCONF southbound protocol. Using the netconf simulation testtool, we proposed SDN framework consisting of three IHON nodes which transport Ethernet streams from different number of nodes, with different connections and different priorities, i.e. both GST and SM connections between each node pair. The netconf testtool supports netconf monitoring, and netconf operations are executed in a secured manner. In order to accomplish this, we installed SSH feature in the Beryllium karaf distribution, then each node is connected to the controller using netconf-connector. After developing YANG schema for IHON node, we loaded it to the simulation nodes. When all nodes are connected successfully, the user (network operator) controls the nodes through the SDN controller. Therefore, the control plane passes the configurations of all nodes in the network remotely which overcomes the manual configuration of each node separately. In the next chapter, we will discuss an SDN experiment conducted at Uninett labs with the prototype Fusion H1 nodes from Transpacket and the ODL controller installed at the stationary server at the department of Telematics.

Chapter 7

SDN and Fusion Experiment Results

This chapter presents the results for SDN experiment conducted with H1 network at Uninett labs. In Section 7.1, we describe the Fusion H1 network setup together with the forwarding of Ethernet streams in the network. Section 7.2 illustrates the implementation of ODL platform for the H1 network including the basic procedures to connect each node with the controller. Lastly, Section 7.3 describes the management of nodes with the ODL controller.

7.1 H1 Network Diagram Setup

As illustrated in Figure 7.1, the network setup at Uninett consists of three H1 nodes (the node names are N1, N2, and N3, and their corresponding IP addresses are shown in Table 7.2), and a Spirent traffic generator/analyser (SPT-2000). For security reasons the IP addresses of the H1 nodes have been replaced with 192.168.209./24. Moreover, GST and SM Ethernet streams are represented with green and red, respectively. N1 transports three GST streams each of 1Gbps, and a single SM stream of size 10Gbps all generated by the traffic generator. On the other hand, N2 drops one of the GST streams it received from N1 through its 1GE interface, but inserts additional two GST and a single SM streams to be forwarded to N3. In N3, all the streams received from N1 and N2 are delivered back to the SPT-2000 through their corresponding interfaces. Moreover, all VLANs in the network are assigned the same names as the interfaces of the nodes to which they are connected; for example, the interface ge0 of N1 corresponds to VLAN ge0 from the traffic generator. In this case, ge5 and ge8 correspond to the same VLAN. All the VLANs and Ethernet streams in the network setup are summarized in Table 7.1.

7.2 Deploying ODL platform for the H1 network

As described in the previous chapters of this thesis, we used the same SDN controller platform testbed, i.e. the ODL controller, specifically the Beryllium version to

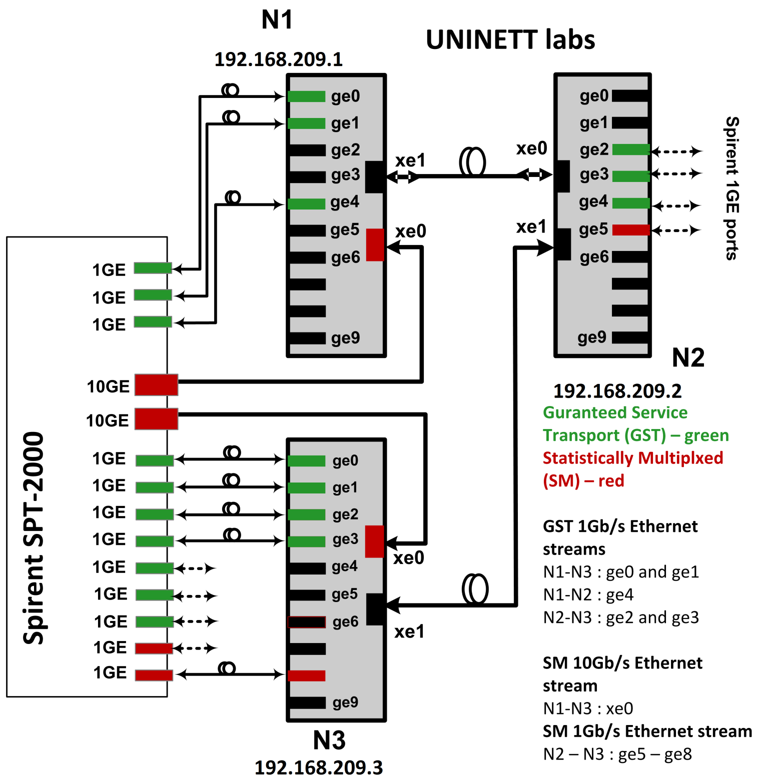


Figure 7.1: Network setup diagram at Uninett with three H1 nodes and Spirent SPT-2000 packet generator. GST traffic are depicted in green while SM in red, and their corresponding paths are depicted at the right bottom side.

manage the configuration of the H1 nodes at Uninett labs. The Beryllium controller is installed at stationary server at the department of Telematics. The H1 nodes are built based on NETCONF protocol and YANG data modeling language. As a result, NETCONF protocol is used as a southbound plug-in to connect the H1 nodes with the ODL testbed. In order to achieve management of the network remotely with the Beryllium platform, we installed the NETCONF feature, i.e. netconf-connector in the Beryllium distribution.

As we have discussed in Section 5.2.1, a new netconf-connector is created for each node to enable communication with the controller. To accomplish this, we installed the netconf-connector and restconf features in the karaf distribution as given in Appendix A. Next, we created three netconf-connectors corresponding to the three H1 nodes at the Uninett labs by executing POST request using postman plug-in with the following parameters:

Table 7.1: Connection set up of N1, N2, and N3; illustrates the stream types and VLANs in each node.

Node	Interfaces									
	ge0	ge1	ge2	ge3	ge4	ge5	ge8	Xe0	Xe1	
	GST	GST	GST	GST	GST	SM	SM	SM access-interface (except in N2; it is Trunk)	Trunk	Type of Ethernet Stream
N1	ge0	ge1	-	-	ge4	-	-	xe0	ge0 + ge1 + ge4 + xe0	VLAN
N2	-	-	ge2	ge3	ge4	ge5	-	ge0 + ge1	ge0 + ge1 + ge2 + ge3 + ge5 + xe0	
N3	ge0	ge1	ge2	ge3	-	-	ge8	xe0	ge0 + ge1 + ge2 + ge3 + ge8 + xe0	

Table 7.2: IP address and port number for the H1 nodes in the network setup. The Ip address, username, and password of the H1 nodes are changed for security reasons.

device-name	IP address	Port	Username and Password
N1	192.168.209.1	830	admin
N2	192.168.209.2		
N3	192.168.209.3		

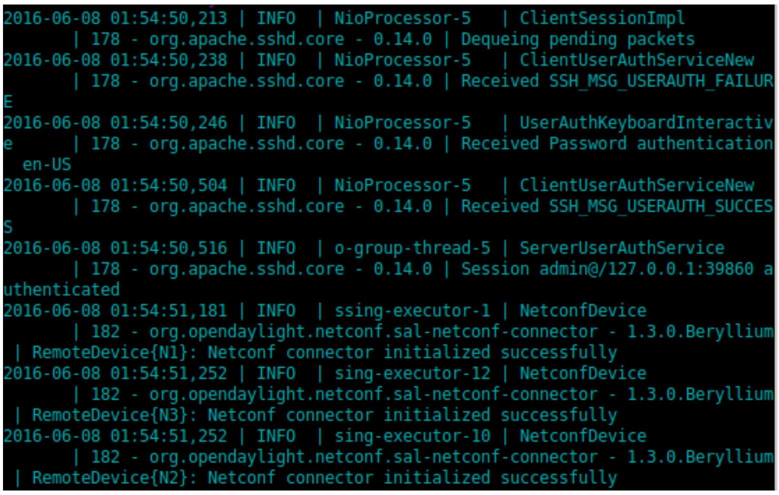
- Operation: POST,
- URL:
  http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules,
- Headers:
  - Accept: application/XML,
  - Content-Type: application/XML,
- Body: Depicted in Appendix C.1. In order to create netconf-connector for N1, N2, and N3, we replaced the device_name, device_IP_address, device_port_number, username, and password in the source code with the corresponding parameters of the nodes as given in Table 7.2.

Hence, the three netconf-connectors are created by executing POST request using postman with their corresponding body. After creating the netconf-connectors, we verified that each node is successfully connected to the ODL controller using *log:display* command in the karaf distribution; the log output is illustrated in Figure 7.2. Moreover, we checked this by executing GET request to: *http://localhost:8181/restconf/operational/network-topology:network-topology/*. For example, Appendix E.1 shows the output for N1. Additionally, N2 and N3 return similar outputs. So, the capabilities of the H1 nodes are stored in the MD-SAL datastore. Moreover, we can see that the node includes ietf-inet-types, and ietf-yang-types capabilities, thus the H1 node supports NETCONF monitoring and advertises ietf-inet-types capability during HELLO message exchange. Besides, the H1 node has additional capabilities that provide full functionality of the IHON principles as discussed in Chapter 2.

7.3 Management of H1 Nodes

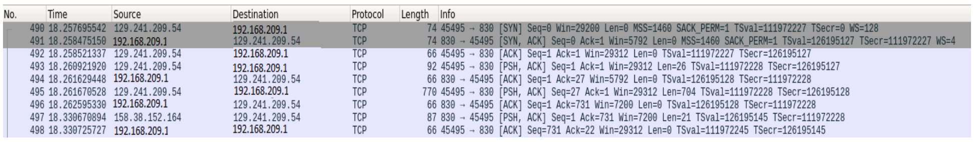
Deploying the ODL controller for the H1 network enables us to manage the configuration and forwarding schemes of the nodes. After starting the netconf-connector for each node, we can manipulate (i.e. read, write, update, and delete) the configuration of the nodes from the controller. The YANG schema of the H1 nodes is proprietary, and we get privilege to access the internal schema of the node in the XML schema representing the configuration. As a result, we can retrieve the schema, modify it to relate the wanted configuration and push it back to the nodes. This way we can fully configure the H1 nodes in the network.

First, we execute GET request to: *http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/Ni/yang-ext:mount/* where Ni indicates the node name, in this case, N1, N2, or N3. So, replacing Ni by the either of the nodes returns the complete configuration of the corresponding node. The retrieved XML configurations of N1, N2, and N3 are illustrated in Appendix E.2, Appendix E.3,



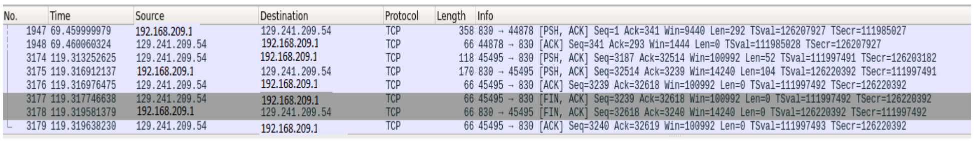
```
2016-06-08 01:54:50,213 | INFO  | NioProcessor-5   | ClientSessionImpl
       | 178 - org.apache.sshd.core - 0.14.0 | Dequeing pending packets
2016-06-08 01:54:50,238 | INFO  | NioProcessor-5   | ClientUserAuthServiceNew
       | 178 - org.apache.sshd.core - 0.14.0 | Received SSH_MSG_USERAUTH_FAILUR
E
2016-06-08 01:54:50,246 | INFO  | NioProcessor-5   | UserAuthKeyboardInteractiv
e      | 178 - org.apache.sshd.core - 0.14.0 | Received Password authentication
  en-US
2016-06-08 01:54:50,504 | INFO  | NioProcessor-5   | ClientUserAuthServiceNew
       | 178 - org.apache.sshd.core - 0.14.0 | Received SSH_MSG_USERAUTH_SUCCES
S
2016-06-08 01:54:50,516 | INFO  | o-group-thread-5 | ServerUserAuthService
       | 178 - org.apache.sshd.core - 0.14.0 | Session admin@/127.0.0.1:39860 a
uthenticated
2016-06-08 01:54:51,181 | INFO  | sing-executor-1 | NetconfDevice
       | 182 - org.opendaylight.netconf.sal-netconf-connector - 1.3.0.Beryllium
 | RemoteDevice{N1}: Netconf connector initialized successfully
2016-06-08 01:54:51,252 | INFO  | sing-executor-12 | NetconfDevice
       | 182 - org.opendaylight.netconf.sal-netconf-connector - 1.3.0.Beryllium
 | RemoteDevice{N3}: Netconf connector initialized successfully
2016-06-08 01:54:51,252 | INFO  | sing-executor-10 | NetconfDevice
       | 182 - org.opendaylight.netconf.sal-netconf-connector - 1.3.0.Beryllium
 | RemoteDevice{N2}: Netconf connector initialized successfully
```

Figure 7.2: Log output from the Beryllium controller which shows successful starting of netconf-connector for N1, N2, and N3.



No.	Time	Source	Destination	Protocol	Length	Info
490	18.257695542	129.241.209.54	192.168.209.1	TCP	74	45495 → 830 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=111972227 TSecr=0 WS=128
491	18.258475150	192.168.209.1	129.241.209.54	TCP	74	830 → 45495 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=126195127 TSecr=111972227 WS=4
492	18.258521337	129.241.209.54	192.168.209.1	TCP	66	45495 → 830 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=111972227 TSecr=126195127
493	18.260921920	129.241.209.54	192.168.209.1	TCP	92	45495 → 830 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=26 TSval=111972228 TSecr=126195127
494	18.261629448	192.168.209.1	129.241.209.54	TCP	66	830 → 45495 [ACK] Seq=1 Ack=27 Win=5792 Len=0 TSval=126195128 TSecr=111972228
495	18.261670528	129.241.209.54	192.168.209.1	TCP	770	45495 → 830 [PSH, ACK] Seq=27 Ack=1 Win=29312 Len=704 TSval=111972228 TSecr=126195128
496	18.262595330	192.168.209.1	129.241.209.54	TCP	66	830 → 45495 [ACK] Seq=1 Ack=731 Win=7200 Len=0 TSval=126195128 TSecr=111972228
497	18.330670894	158.38.152.164	129.241.209.54	TCP	87	830 → 45495 [PSH, ACK] Seq=1 Ack=731 Win=7200 Len=21 TSval=126195145 TSecr=111972228
498	18.330725727	192.168.209.1	192.168.209.1	TCP	66	45495 → 830 [ACK] Seq=731 Ack=22 Win=29312 Len=0 TSval=111972245 TSecr=126195145

Figure 7.3: Captured traffic with Wireshark between the controller and N1 node which shows the connection establishment, and data delivering with the PSH and ACK flags. The IP address of the N1 node is changed for security issues.



No.	Time	Source	Destination	Protocol	Length	Info
1947	69.459999979	192.168.209.1	129.241.209.54	TCP	358	830 → 44878 [PSH, ACK] Seq=1 Ack=341 Win=9440 Len=292 TSval=126207927 TSecr=111985027
1948	69.460060324	129.241.209.54	192.168.209.1	TCP	66	44878 → 830 [ACK] Seq=341 Ack=293 Win=1444 Len=0 TSval=111985028 TSecr=126207927
3174	119.313252625	129.241.209.54	192.168.209.1	TCP	118	45495 → 830 [PSH, ACK] Seq=3187 Ack=32514 Win=100992 Len=52 TSval=111997491 TSecr=126203182
3175	119.316912137	192.168.209.1	129.241.209.54	TCP	170	830 → 45495 [PSH, ACK] Seq=32514 Ack=3239 Win=14240 Len=104 TSval=126220392 TSecr=111997491
3176	119.316976475	129.241.209.54	192.168.209.1	TCP	66	45495 → 830 [ACK] Seq=3239 Ack=32618 Win=100992 Len=0 TSval=111997492 TSecr=126220392
3177	119.317746638	129.241.209.54	192.168.209.1	TCP	66	45495 → 830 [FIN, ACK] Seq=3239 Ack=32618 Win=100992 Len=0 TSval=111997492 TSecr=126220392
3178	119.319581379	192.168.209.1	129.241.209.54	TCP	66	830 → 45495 [FIN, ACK] Seq=32618 Ack=3240 Win=14240 Len=0 TSval=126220392 TSecr=111997492
3179	119.319638230	129.241.209.54	192.168.209.1	TCP	66	45495 → 830 [ACK] Seq=3240 Ack=32619 Win=100992 Len=0 TSval=111997493 TSecr=126220392

Figure 7.4: Captured traffic between the controller and N1 node; it specifically shows the closing of the ongoing session using FIN, and ACK flags.

and Appendix E.4, respectively. Hence, we confirmed that the configuration of the nodes which is obtained by executing the GET request exactly matches with the configuration plan illustrated in Table 7.1.

Moreover, we captured the traffic flow between the controller and the nodes using wireshark. For example, the screenshots depicted in Figure 7.3 and Figure 7.4

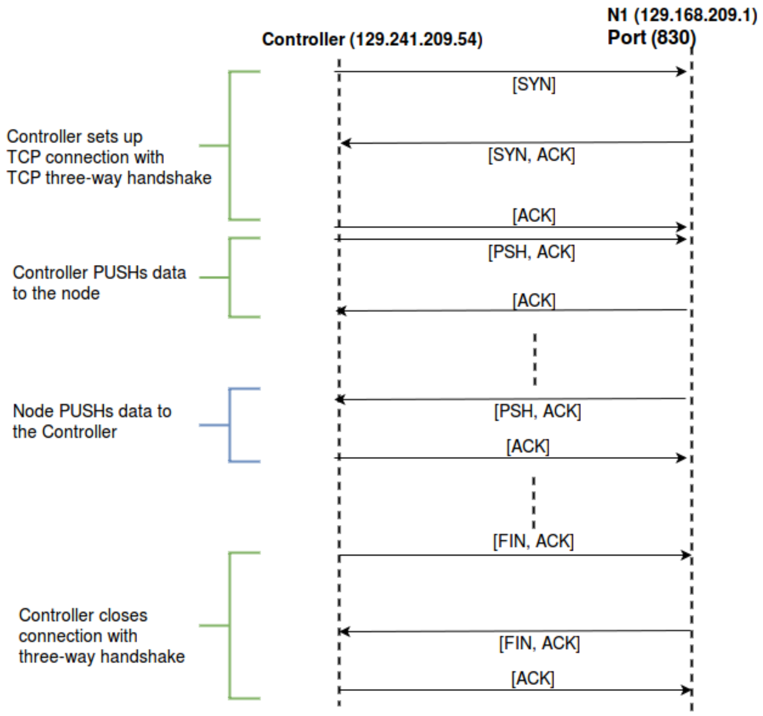

N1 (129.168.209.1)
Port (830)
Controller (129.241.209.54)
[SYN]
Controller sets up TCP connection with TCP three-way handshake
[SYN, ACK]
[ACK]
[PSH, ACK]
Controller PUSHs data to the node
[ACK]
Node PUSHs data to the Controller
[PSH, ACK]
[ACK]
[FIN, ACK]
Controller closes connection with three-way handshake
[FIN, ACK]
[ACK]

Figure 7.5: Flow graph between ODL controller and N1 node

illustrate the message flows between the controller and N1. The actual IP address of N1 is replaced manually by private IP address for security reasons. The corresponding flow graph is illustrated in Figure 7.5. We can see that the controller first initiates a TCP connection over an Internet Protocol based network using three-way handshake (SYN-SYN-ACK). As a result, a TCP session is established between the controller and the H1 node. When either of the entities needs to send data to the other peer, it applies the PUSH (PSH) [Dou06] flag, thus the TCP stack at the receiver side pushes the received data without buffering it directly to the receiving application. Finally, the controller closes the connection by sending a FIN (final) flag to the node. The node receives the termination request, and it immediately closes the session.

In conclusion, implementing SDN with ODL platform for the Fusion H1 network setup at Uninett accomplishes the decoupling of the control and data plane scenario. Moreover, network functions, configurations, and protocols are made programmable with the controller. Applications that run over the controller get abstracted map of the underlying network topology, and manipulate the logical map. In the next chapter, we will present the conclusion and possible future work to our thesis.

Chapter 8 Conclusion and Future Work

8.1 Conclusion

In this thesis work for the first time was proposed and demonstrated through emulation and experiments an SDN controlled Integrated Hybrid Optical Network. The emulation is carried out to achieve NETCONF-based SDN realization for IHON network with ODL platform testbed. We proposed an SDN setup consisting of three IHON nodes each carrying different types of streams with different QoS from various connections. Based on the working principles of the IHON nodes, we developed a YANG schema that represents the configuration of a typical IHON node. Each node is simulated with a netconf testtool, and the YANG schema is loaded to the simulation tool. Upon installation of the required features in the ODL distribution, we successfully established NETCONF based communication between the simulated nodes and the controller. As a result, the separation of control and data plane is achieved with the ODL testbed using the NETCONF southbound protocol. From application running on the controller, we accessed the nodes and manipulated the configuration, thus the logical connectivity, of the network with multiple services.

Furthermore, we conducted SDN experiment for H1 network at Uninett labs. The same ODL testbed which is installed in a server in the Telematics department is used to implement SDN/NETCONF for the H1 network. With the ODL controller, we have established successful communication with each node and retrieved their configuration. Having the right privilege, the network connection can be expanded, and complete network management and node configuration can be accomplished from the controller. Hence, the GST and SM flows from different VLANs can be measured. As a result, we can verify efficient utilization of the transmission capacity and guaranteed QoS in SDN controlled H1 network.

We have explained the theoretical background to IHON nodes in general and H1 prototype nodes in particular. IHON optimizes capacity utilization and offers guaranteed QoS by combining the advantages of both packet and circuit switched traffic while diminishing their disadvantages. Moreover, we have seen that SDN allows programmability of network functions and protocols with a logically centralized controller. As a result, SDN reduces costs and improves network flexibility in provisioning new services and avoids interoperability problems between heterogeneous domains. Besides, we have explained NETCONF as a southbound plug-in to realize SDN for IHON in general and H1 node in specific. We have discussed NETCONF client-server communication over SSH session and executing operation remotely. Finally, we evaluated that ODL platform is a suitable testbed to accomplish SDN for IHON, specifically for Fusion H1 network.

8.2 Future Work

This section discusses some additional points to consider in future works as a continuation of this work.

For the emulation, we apply netconf testtool to simulate the IHON node configurations. The YANG schema that we developed represents basic configuration of H1 nodes; it requires further extension to define the complete functionality of the Fusion nodes according to the IHON principles. For example, the definitions for GST and SM streams can be described more in detail, and the forwarding scheme in the IHON nodes which is based on Q-in-Q IEEE 802.1ad [Fus13] can be modeled using YANG schema. Moreover, the network can be expanded by adding Ethernet streams, i.e. GST and/or SM from different connections. The above listed extensions and other additional definitions are, then, modeled using YANG schema. As a result, configuration, management, and monitoring of the network is achieved using the ODL testbed.

Moreover, OpenDaylight User Interface (DLUX) [Com16b] [Com16c] can be used to view the network topology of both emulation and lab experiment, and test application development. DLUX and associated features are installed using the command: *feature:install odl-dlux-core-all odl-l2switch-switch-ui* in the Beryllium karaf distribution. The *odl-dlux-core-all* installs the DLUX, and the topology application is activated immediately. On the other hand, *odl-l2switch-switch-ui* allows to view the topology details. Other applications like node, yang User interface (yang UI) can also be enabled to view the list of nodes, and yang development, respectively [Com16b]. The DLUX is then accessed using the login URL: *http://<karaf-ip>:8181/index.html*, in our case *localhost* is used as *karaf-ip* . As a result, the statistics for GST and SM streams between the nodes can be recorded, and measurement on the performance of the SDN controlled framework for the emulated IHON can be achieved.

References

[Ber15] Weldmicheal Berhanu. Software defined networking for fusion (integrated hybrid optical) networks. Specialization project report (TTM4501), NTNU, 2015.

[BNH03] S Bjornstad, M Nord, and DR Hjelme. Qos differentiation and header/payload separation in optical packet switching using polarisation multiplexing. In *Proceedings of ECOC 2003*, pages 28–29, 2003.

[BNO$^+$05] Steinar Bjornstad, Martin Nord, Torodd Olsen, D Hjelme, and NORVALD Stol. Burst, packet and hybrid switching in the optical core network. *TELEKTRONIKK*, 101(2):148, 2005.

[Bro15] Brocade sdn controller user guide. http://www.brocade.com/content/dam/common/documents/content-types/user-guide/SDN-Controller-2.1.0-User-Guide.pdf, Nov 2015.

[Bur15] Alagalah (Keith Burns). Southbound (netconf-connector). https://github.com/opendaylight/docs/blob/master/manuals/user-guide/src/main/asciidoc/controller/netconf/odl-netconf-southbound-user.adoc, July 2015.

[Cen] Netconf Central. Network configuration protocol. http://www.netconfcentral.org/netconf_docs.

[CNRF$^+$13] M Channegowda, R Nejabati, M Rashidi Fard, S Peng, N Amaya, G Zervas, D Simeonidou, R Vilalta, R Casellas, R Martínez, et al. Experimental demonstration of an openflow based software-defined optical network employing packet, fixed and flexible dwdm grid technologies on an international multi-domain testbed. *Optics express*, 21(5):5487–5498, 2013.

[Com16a] OpenDaylight Community. Opendaylight developer guide. https://drive.google.com/file/d/0B_rLr6so6DZ8cXRJczlJSVBzOE0/view, Feb 2016.

[Com16b] OpenDaylight Community. Opendaylight installation guide. https://drive.google.com/file/d/0B_rLr6so6DZ8S0ZRaDlIQ091V0k/view, Feb 2016.

[Com16c] OpenDaylight Community. Opendaylight user guide. https://drive.google.com/file/d/0B_rLr6so6DZ8RVJyWXpVcEdhdVE/view, Feb 2016.

[Das10] Saurav Das. Extensions to the openflow protocol in support of circuit switching. *Addendum to OpenFlow protocol specification (v1. 0)—Circuit Switch Addendum v0*, 3, 2010.

[Dou06] Comer Douglas. Internetworking with tcp/ip: Principles, protocols, and architecture. *fourth edtion*, 1, 2006.

[DPM09] S. Das, G. Parulkar, and N. McKeown. Unifying packet and circuit switched networks. In *GLOBECOM Workshops, 2009 IEEE*, pages 1–6, Nov 2009.

[DPM$^+$10] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong. Packet and circuit network convergence with openflow. In *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*, pages 1–3, March 2010.

[DR08] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2, August 2008.

[EBS11] Rob Enns, Martin Bjorklund, and Juergen Schoenwaelder. Netconf configuration protocol. *Network*, 2011.

[FRZ14] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[FT02] Roy T Fielding and Richard N Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[Fus12] Fusion networking explained: Bringing true circuit and packet properties to the packet network. White paper, July 2012.

[Fus13] Transpacket h1; a fusion networking add-drop muxponder. White paper, Apr 2013.

[GDS$^+$10] Vinesh R Gudla, Saurav Das, Anujit Shastri, Guru Parulkar, Nick McKeown, Leonid Kazovsky, and Shinji Yamashita. Experimental demonstration of openflow control of packet and circuit switches. In *Optical Fiber Communication Conference*, page OTuG2. Optical Society of America, 2010.

[GKB$^+$06] Christoph M Gauger, Paul J Kuhn, Erik Van Breusegem, Mario Pickavet, and Piet Demeester. Hybrid optical network architectures: bringing packets and circuits together. *Communications Magazine, IEEE*, 44(8):36–42, 2006.

[HGRG06] Taqi Hasan, Elango Gannesan, Allen B Rochkind, and Sagar Golla. Network management system, July 25 2006. US Patent 7,082,464.

[JMD14] Y. Jarraya, T. Madi, and M. Debbabi. A survey and a layered taxonomy of software-defined networking. *Communications Surveys Tutorials, IEEE*, 16(4):1955–1980, Fourthquarter 2014.

[LTG90] Aurel A Lazar, Adam Temple, and Rafael Gidron. An architecture for integrated networks that guarantees quality of service. *International Journal of Digital & Analog Communication Systems*, 3(2):229–238, 1990.

[MB10] Ed. M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol(NETCONF). RFC 6020, RFC Editor, October 2010.

[ODLa] Getting started with opendaylight. https://docs.google.com/document/d/12O86Nc4IbHvBC_ZGE_tHl0IWYd75Lk2HzfcV4YxhWTg/pub.

[ODLb] Opendaylight controller:md-sal:restconf. https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Restconf.

[ODLc] Opendaylight controller:netconf:testtool. https://wiki.opendaylight.org/view/OpenDaylight_Controller:Netconf:Testtool.

[ODLd] Opendaylight platform. https://www.opendaylight.org/.

[ODLe] What are sdn controllers (or sdn controllers platforms). https://www.sdxcentral.com/resources/sdn/sdn-controllers/.

[ODL16] Odl beryllium (be) - the fourth release of opendaylight. https://www.opendaylight.org/odlbe, Feb 2016.

[ONF13a] Openflow switch specification, version 1.4.0 (wire protocol 0x05). Open Networking Foundation, Oct 2013.

[ONF13b] Software-defined networking: The new norm for networks. White paper, Apr 2013.

[Ste] Daniel Stenberg. Manual – curl usage explained. https://curl.haxx.se/docs/manual.html.

[SWJR$^+$94] Jerry W Sprecher, Donald J Winters Jr, Amirali S Rajwany, Michael W Dodson, Gene R Penning, Darryl F Harrington, and Simon Chou. Network management system, February 8 1994. US Patent 5,285,494.

[VBB13] R. Veisllari, S. Bjornstad, and K. Bozorgebrahimi. Integrated packet/circuit hybrid network field trial with production traffic [invited]. *Optical Communications and Networking, IEEE/OSA Journal of*, 5(10):A257–A266, Oct 2013.

[VBB15] Raimena Veisllari, Steinar Bjornstad, and Kurosh Bozorgebrahimi. Load balancing in sdn enabled integrated packet/circuit networks, first experimental demonstrations. 2015.

[VSBR14] Raimena Veisllari, Norvald Stol, Steinar Bjornstad, and Carla Raffaelli. Scalability analysis of sdn-controlled optical ring man with hybrid traffic. pages 3283–3288, 2014.

[Wat15] K Watsen. Network working group a. bierman internet-draft yumaworks intended status: Standards track m. bjorklund expires: December 6, 2015 tail-f systems. June 2015.

[WG06] M. Wasserman and T. Goddard. Using the NETCONF Configuration Protocol over Secure SHell (SSH). RFC 4742 (Proposed Standard), December 2006.

[YL06a] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252 (Proposed Standard), January 2006.

[YL06b] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol. RFC 4254 (Proposed Standard), January 2006.

[YL06c] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006.

[YL06d] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Protocol Architecture. Technical Report 4251, RFC Editor, Fremont, CA, USA, January 2006.

Appendix A
OpenDayLight Installation and Running

For running an ODL controller, a JVM is required, and any operating system which supports Java can be used to run the controller. To successfully run ODL, the following systems are needed [Com16b]:

- Multi-core processor with at least 8 GB of RAM,
- Java 7 JDK, and
- Linux operating system is more recommended.

Depending on the operating system, the Java JDK can be installed using different commands. In Ubuntu, for example, we use the following command [Com16b]:

```
> sudo apt-get install openjdk-7-jdk
```

In order to emulate SDN for IHON and carry out the experiment at Uninett labs, we imported pre-built all-in-one VM used for SDN development which is a 64-bit ubuntu 14.04 image in to Oracle VM VirtualBox. Our working machine is Linux distribution, and the VirtualBox version is VBoxGuestAdditions-4.2.18 (this is checked using the command: *ls /opt/*). Hence, we downloaded the 64-bit OVA of the SDN VM (SDN_tutorial_VM_64bit.ova[1]) file and imported to the VirtualBox. By allocating the recommended memory (8GB), and setting the network adapter in NAT mode, we boot the VM: the username and password for this VM are both "ubuntu".

The current ODL version, i.e. Beryllium (distribution-karaf-0.4.0-Beryllium/) is downloaded from OpenDaylight software download page[2]. Since karaf distribution has no any installed features by default, additional features are added depending on the implementation requirements. Furthermore, it is not recommended to enable all features at the same time for reasons of compatibility [Com16b].

Once the Beryllium distribution is downloaded, we follow the below listed proce-

[1]http://sdnhub.org/tutorials/sdn-tutorial-vm/
[2]http://www.opendaylight.org/software/downloads

dures so as to run the karaf distribution [Com16b]. The Karaf distribution simplifies package operations and application installations, for example, user can install features, display log outputs, etc.

1. Unzip the distribution zip file.
   ```
   $ ls distribution-karaf-0.4.0-Beryllium.zip
   $ unzip distribution-karaf-0.4.0-Beryllium.zip
   ```
2. Change working directory to the distribution directory.
   ```
   $ cd distribution-karaf-0.4.0-Beryllium
   ```
3. Run the karaf
   ```
   $ ./bin/karaf.
   ```

An ODL user can install features whenever required using the following command [Com16b]:

```
> feature:install <feature-name>
```

Moreover, Multiple features can be installed with the command:

```
> feature:install <feature1-name> <feature2-name> ... <featureN-name>
```

A complete list of the karaf feature, are found using the following command:

```
> feature:list
```

Moreover, the installed features are listed using the following command:

```
> feature:list -i
```

In our case, we have installed the following features in the karaf distribution:

1. *odl-netconf-connector-all*: Activates the NETCONF southbound plug-in to allow remote connection with NETCONF based devices, i.e. the emulated IHON nodes and Fusion H1 nodes at Uninett labs.
2. *odl-restconf–all*: Enables RESTCONF features which allows to create, update, delete, retrieve the configuration of the remote nodes as HTTP requests.
3. *odl-netconf-connector-ssh*: Allows secured communication between the nodes (i.e., both emulated IHON and H1 nodes) and the ODL controller using the SSH protocol.

Appendix B

IHON node YANG Schema and Configuration

B.1 IHON YANG Schema

```
module ihonnode {
    yang-version 1;
    namespace "urn:opendaylight:ihonnode";
    prefix "ihonnode";
    organization "NTNU-Telematics";
    contact "michock.mit@gmail.com";


    description "Node configuration";

    revision "2016-04-15" {
        description "Initial version.";
    }
    container ihonnode {
        leaf node-name {
            description "node name";
            type string;
        }
        container interfaces {
            description "Contains all interfaces of H1
                nodes two Xe and ten ge interfaces";
            list interface {
                key name;
                leaf name{
                    type string;
                }
                leaf describe {
                    type string;
```

```
}
leaf type {
    type string;
}
leaf mtu {
    type uint32;
}
choice ethernet-switching {
    description "defines access or trunk type of interface";
    case access-interface {
        container access-interface {
            description "Identifies the vlan connected to the interface";
            list vlan {
                key vlan-name;
                leaf vlan-name {
                    type string;
                }
                leaf priority {
                    type enumeration {
                        enum gst;
                        enum sm;
                    }
                }
            }
        }
    }
    case trunk-interface {
        container trunk-interface {
            container vlans {
                list vlan {
                    key vlan-name;
                    leaf vlan-name {
                        type string;
                    }
                }
            }
        }
    }
```

```
                }
            }
        }
        container vlans {
            description "Defines a group of VLANs in a Node";
            list vlan {
                key id;
                leaf id {
                    type uint32;
                }
                leaf name {
                    type string;
                }
            }
        }
    }

}
```

B.2 NodeA Configuration

```
<ihonnode xmlns="urn:opendaylight:ihonnode">
        <node-name>NodeA</node-name>
        <interfaces>
                <interface>
                        <name>ge0</name>
                        <describe>connection to Vlan2</describe>
                        <type>ianaift:ethernetCsmacd</type>
                        <access-interface>
                                <vlan-name>Vlan2</vlan-name>
                                <priority>gst</priority>
                        </access-interface>
                </interface>
                <interface>
                        <name>ge1</name>
                        <describe>connection to Vlan3</describe>
```

```
                    <type>ianaift:ethernetCsmacd</type>
                    <access-interface>
                            <vlan-name>Vlan3</vlan-name>
                            <priority>gst</priority>
                    </access-interface>
            </interface>
            <interface>
                    <name>ge9</name>
                    <describe>connection to Vlan4</describe>
                    <type>ianaift:ethernetCsmacd</type>
                    <access-interface>
                            <vlan-name>Vlan4</vlan-name>
                            <priority>sm</priority>
                    </access-interface>
            </interface>
            <interface>
                    <name>xe0</name>
                    <describe>connection to Vlan1</describe>
                    <type>ianaift:ethernetCsmacd</type>
                    <access-interface>
                            <vlan-name>Vlan1</vlan-name>
                            <priority>sm</priority>
                    </access-interface>
            </interface>
            <interface>
                    <name>xe1</name>
                    <describe>connection to xe0 of NodeB</describe>
                    <type>ianaift:ethernetCsmacd</type>
                    <trunk-interface>
                            <vlans>
```

```
                <vlan-name>Vlan1</vlan-name>
                <vlan-name>Vlan2</vlan-name>
                <vlan-name>Vlan3</vlan-name>
                <vlan-name>Vlan4</vlan-name>
                </vlans>
            </trunk-interface>
        <mtu>9700</mtu>
        </interface>
    </interfaces>
    <vlans>
        <vlan>
            <id>1</id>
            <name>Vlan1</name>
        </vlan>
        <vlan>
            <id>2</id>
            <name>Vlan2</name>
        </vlan>
        <vlan>
            <id>3</id>
            <name>Vlan3</name>
        </vlan>
        <vlan>
            <id>4</id>
            <name>Vlan4</name>
        </vlan>
    </vlans>
</ihonnode>
```

B.3 NodeB Configuration

```
<ihonnode xmlns="urn:opendaylight:ihonnode">
    <node-name>NodeB</node-name>
    <interfaces>
        <interface>
            <name>ge0</name>
            <describe>connection to Vlan2</describe>
            <type>ianaift:ethernetCsmacd</
                type>
            <access-interface>
                <vlan-name>Vlan2</vlan-
                    name>
                <priority>gst</priority>
            </access-interface>
        </interface>
        <interface>
            <name>ge8</name>
            <describe>connection to Vlan8</
                describe>
            <type>ianaift:ethernetCsmacd</
                type>
            <access-interface>
                <vlan-name>Vlan8</vlan-
                    name>
                <priority>sm</priority>
            </access-interface>
        </interface>
        <interface>
            <name>ge9</name>
            <describe>connection to Vlan4</
                describe>
            <type>ianaift:ethernetCsmacd</
                type>
            <access-interface>
                <vlan-name>Vlan4</vlan-
                    name>
                <priority>sm</priority>
            </access-interface>
        </interface>
        <interface>
            <name>xe0</name>
            <describe>connection to xe1 of
                NodeA</describe>
            <type>ianaift:ethernetCsmacd</
                type>
            <trunk-interface>
                <vlans>
```

```
                    <vlan-name>Vlan1</vlan-
                       name>
                    <vlan-name>Vlan2</vlan-
                       name>
                    <vlan-name>Vlan3</vlan-
                       name>
                    <vlan-name>Vlan4</vlan-
                       name>
                    </vlans>
                </trunk-interface>
                <mtu>9700</mtu>
            </interface>
            <interface>
                <name>xe1</name>
                <describe>connection to xe1 of
                   NodeC</describe>
                <type>ianaift:ethernetCsmacd</
                   type>
                <trunk-interface>
                    <vlans>
                    <vlan-name>Vlan1</vlan-
                       name>
                    <vlan-name>Vlan3</vlan-
                       name>
                    <vlan-name>Vlan8</vlan-
                       name>
                    </vlans>
                </trunk-interface>
                <mtu>9700</mtu>
            </interface>
        </interfaces>
        <vlans>
            <vlan>
                <id>1</id>
                <name>Vlan1</name>
            </vlan>
            <vlan>
                <id>2</id>
                <name>Vlan2</name>
            </vlan>
            <vlan>
            <id>3</id>
            <name>Vlan3</name>
        </vlan>
        <vlan>
            <id>4</id>
            <name>Vlan4</name>
        </vlan>
        <vlan>
            <id>8</id>
            <name>Vlan8</name>
        </vlan>
    </vlans>
</ihonnode>
```

B.4 NodeC Configuration

```
<ihonnode xmlns="urn:opendaylight:ihonnode">
    <node-name>NodeC</node-name>
    <interfaces>
        <interface>
            <name>ge1</name>
            <describe>connection to Vlan3</describe>
            <type>ianaift:ethernetCsmacd</type>
            <access-interface>
                <vlan-name>Vlan3</vlan-name>
                <priority>gst</priority>
            </access-interface>
        </interface>
        <interface>
            <name>ge8</name>
            <describe>connection to Vlan8</describe>
            <type>ianaift:ethernetCsmacd</type>
            <access-interface>
                <vlan-name>Vlan8</vlan-name>
                <priority>sm</priority>
            </access-interface>
        </interface>
        <interface>
            <name>xe0</name>
            <describe>connection to Vlan1</describe>
            <type>ianaift:ethernetCsmacd</type>
            <access-interface>
                <vlan-name>Vlan1</vlan-name>
                <priority>sm</priority>
            </access-interface>
        </interface>
        <interface>
            <name>xe1</name>
            <describe>connection to xe1 of NodeB</describe>
            <type>ianaift:ethernetCsmacd</type>
            <trunk-interface>
                <vlans>
                <vlan-name>Vlan1</vlan-name>
                <vlan-name>Vlan3</vlan-name>
                <vlan-name>Vlan8</vlan-name>
                </vlans>
            </trunk-interface>
            <mtu>9700</mtu>
        </interface>
    </interfaces>
    <vlans>
        <vlan>
            <id>1</id>
            <name>Vlan1</name>
        </vlan>
        <vlan>
            <id>3</id>
            <name>Vlan3</name>
        </vlan>
```

```
                <vlan>
                        <id>8</id>
                        <name>Vlan8</name>
                </vlan>
        </vlans>
</ihonnode>
```

B.5 Retrieved Configuration of NodeA

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ihonnode xmlns="urn:opendaylight:ihonnode" xmlns:a="urn:ietf:params:xml:ns:netconf:base:1.0" a:operation="replace">
        <node-name>nodeA</node-name>
        <interfaces>
                <interface>
                        <name>ge0</name>
                        <describe>connection to Vlan2</describe>
                        <type>ianaift:ethernetCsmacd</type>
                        <access-interface>
                                <vlan-name>Vlan2</vlan-name>
                                <priority>gst</priority>
                        </access-interface>
                </interface>
                <interface>
                        <name>ge1</name>
                        <describe>connection to Vlan3</describe>
                        <type>ianaift:ethernetCsmacd</type>
                        <access-interface>
                                <vlan-name>Vlan3</vlan-name>
                                <priority>gst</priority>
                        </access-interface>
                </interface>
                <interface>
                        <name>ge9</name>
```

```
            <describe>connection to Vlan4</describe>
            <type>ianaift:ethernetCsmacd</type>
            <access-interface>
                <vlan-name>Vlan4</vlan-name>
                <priority>sm</priority>
            </access-interface>
        </interface>
        <interface>
            <name>xe0</name>
            <describe>connection to Vlan1</describe>
            <type>ianaift:ethernetCsmacd</type>
            <access-interface>
                <vlan-name>Vlan1</vlan-name>
                <priority>sm</priority>
            </access-interface>
        </interface>
        <interface>
            <name>xe1</name>
            <describe>connection to xe0 of NodeB</describe>
            <type>ianaift:ethernetCsmacd</type>
            <trunk-interface>
                <vlans>
                    <vlan-name>Vlan1</vlan-name>
                    <vlan-name>Vlan2</vlan-name>
                    <vlan-name>Vlan3</vlan-name>
                    <vlan-name>Vlan4</vlan-name>
                </vlans>
            </trunk-interface>
        <mtu>9700</mtu>
```

```
                    </interface>
            </interfaces>
            <vlans>
                    <vlan>
                            <id>1</id>
                            <name>Vlan1</name>
                    </vlan>
                    <vlan>
                            <id>2</id>
                            <name>Vlan2</name>
                    </vlan>
                    <vlan>
                            <id>3</id>
                            <name>Vlan3</name>
                    </vlan>
                    <vlan>
                            <id>4</id>
                            <name>Vlan4</name>
                    </vlan>
            </vlans>
    </ihonnode>
</data>
```

B.6 Retrieved Configuration of NodeB

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ihonnode xmlns="urn:opendaylight:ihonnode" xmlns:a="
        urn:ietf:params:xml:ns:netconf:base:1.0" a:
        operation="replace">
            <node-name>nodeB</node-name>
            <interfaces>
                    <interface>
                            <name>ge0</name>
                            <describe>connection to Vlan2</
                                describe>
                            <type>ianaift:ethernetCsmacd</
                                type>
                            <access-interface>
                                    <vlan-name>Vlan2</vlan-
                                        name>
                                    <priority>gst</priority>
                            </access-interface>
        </interface>
        <interface>
                <name>ge8</name>
                <describe>connection to Vlan8</
                    describe>
                <type>ianaift:ethernetCsmacd</
                    type>
                <access-interface>
                        <vlan-name>Vlan8</vlan-
                            name>
                        <priority>sm</priority>
                </access-interface>
        </interface>
        <interface>
                <name>ge9</name>
                <describe>connection to Vlan4</
                    describe>
                <type>ianaift:ethernetCsmacd</
                    type>
                <access-interface>
                        <vlan-name>Vlan4</vlan-
                            name>
                        <priority>sm</priority>
                </access-interface>
        </interface>
        <interface>
                <name>xe0</name>
                <describe>connection to xe1 of
                    NodeA</describe>
                <type>ianaift:ethernetCsmacd</
                    type>
                <trunk-interface>
                        <vlans>
                                <vlan-name>Vlan1
                                    </vlan-name>
                                <vlan-name>Vlan2
                                    </vlan-name>
                                <vlan-name>Vlan3
                                    </vlan-name>
                                <vlan-name>Vlan4
                                    </vlan-name>
                </trunk-interface>
                <mtu>9700</mtu>
            </interface>
            <interface>
                <name>xe1</name>
                <describe>connection to xe1 of
                    Nodec</describe>
                <type>ianaift:ethernetCsmacd</
                    type>
                <trunk-interface>
                    <vlans>
                        <vlan-name>Vlan1
                            </vlan-name>
                        <vlan-name>Vlan3
                            </vlan-name>
                        <vlan-name>Vlan8
                            </vlan-name>
                    </vlans>
                </trunk-interface>
                <mtu>9700</mtu>
            </interface>
        </interfaces>
        <vlans>
            <vlan>
                <id>1</id>
                <name>Vlan1</name>
            </vlan>
            <vlan>
                <id>2</id>
                <name>Vlan2</name>
            </vlan>
            <vlan>
                <id>3</id>
                <name>Vlan3</name>
            </vlan>
            <vlan>
                <id>4</id>
                <name>Vlan4</name>
            </vlan>
            <vlan>
                <id>8</id>
```

```
                    <name>Vlan8</name>
                </vlan>
            </vlans>
        </ihonnode>
</data>
```

B.7 Retrieved Configuration of NodeC

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ihonnode xmlns="urn:opendaylight:ihonnode" xmlns:a="
        urn:ietf:params:xml:ns:netconf:base:1.0" a:
        operation="replace">
        <node-name>nodeC</node-name>
        <interfaces>
                <interface>
                        <name>ge1</name>
                        <describe>connection to Vlan3</
                            describe>
                        <type>ianaift:ethernetCsmacd</
                            type>
                        <access-interface>
                                <vlan-name>Vlan3</vlan-
                                    name>
                                <priority>gst</priority>
                        </access-interface>
                </interface>
                <interface>
                        <name>ge8</name>
                        <describe>connection to Vlan8</
                            describe>
                        <type>ianaift:ethernetCsmacd</
                            type>
                        <access-interface>
                                <vlan-name>Vlan8</vlan-
                                    name>
                                <priority>sm</priority>
                        </access-interface>
                </interface>
                <interface>
                        <name>xe0</name>
                        <describe>connection to Vlan1</
                            describe>
                <type>ianaift:ethernetCsmacd</type>
                <access-interface>
                        <vlan-name>Vlan1</vlan-name>
                        <priority>sm</priority>
                </access-interface>
        </interface>
        <interface>
                <name>xe1</name>
                <describe>connection to xe1 of NodeB</describe>
                <type>ianaift:ethernetCsmacd</type>
                <trunk-interface>
                        <vlans>
                        <vlan-name>Vlan1</vlan-name>
                        <vlan-name>Vlan3</vlan-name>
                        <vlan-name>Vlan8</vlan-name>
                        </vlans>
                </trunk-interface>
                <mtu>9700</mtu>
        </interface>
</interfaces>
<vlans>
        <vlan>
                <id>1</id>
                <name>Vlan1</name>
        </vlan>
        <vlan>
                <id>3</id>
                <name>Vlan3</name>
        </vlan>
        <vlan>
                <id>8</id>
                <name>Vlan8</name>
        </vlan>
</vlans>
```

```
        </ihonnode>
</data>
```

Appendix C
Creating Netconf-connector

C.1 Creating New Netconf-connector

In order to connect a netconf device with ODL controller, a new netconf-connector is spawned with the following source code [ODLc].

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">prefix:sal-netconf-connector</type>
  <name>device-name</name>
  <address xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">device_IP_address</address>
  <port xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">device_port_number</port>
  <username xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">username</username>
  <password xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">password</password>
  <tcp-only xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">false</tcp-only>
  <event-executor xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">prefix:netty-event-executor
```

```
      </type>
    <name>global-event-executor</name>
  </event-executor>
  <binding-registry xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">prefix:binding-broker-osgi-registry</type>
    <name>binding-osgi-broker</name>
  </binding-registry>
  <dom-registry xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">prefix:dom-broker-osgi-registry</type>
    <name>dom-broker</name>
  </dom-registry>
  <client-dispatcher xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:config:netconf">prefix:netconf-client-dispatcher</type>
    <name>global-netconf-dispatcher</name>
  </client-dispatcher>
  <processing-executor xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">prefix:threadpool</type>
    <name>global-netconf-processing-executor</name>
  </processing-executor>
  <keepalive-executor xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">prefix:scheduled-threadpool</type>
    <name>global-netconf-ssh-scheduled-executor</name>
  </keepalive-executor>
</module>
```

C.2 Updating netconf-connector

While the controller is running, the netconf-connector can be reconfigured with new parameters, for example, username, and password [ODLc].

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">prefix:sal-netconf-connector</type>
  <name>device-name</name>
  <username xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">new-username</username>
  <password xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">new-password</password>
  <tcp-only xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">false</tcp-only>
  <event-executor xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">prefix:netty-event-executor</type>
    <name>global-event-executor</name>
  </event-executor>
  <binding-registry xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">prefix:binding-broker-osgi-registry</type>
    <name>binding-osgi-broker</name>
  </binding-registry>
  <dom-registry xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">prefix:dom-broker-osgi-registry</type>
    <name>dom-broker</name>
  </dom-registry>
  <client-dispatcher xmlns="urn:opendaylight:params:xml:
      ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:
        yang:controller:config:netconf">prefix:netconf-
        client-dispatcher</type>
    <name>global-netconf-dispatcher</name>
  </client-dispatcher>
  <processing-executor xmlns="urn:opendaylight:params:xml
      :ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:
        yang:controller:threadpool">prefix:threadpool</
        type>
    <name>global-netconf-processing-executor</name>
  </processing-executor>
  <keepalive-executor xmlns="urn:opendaylight:params:xml:
      ns:yang:controller:md:sal:connector:netconf">
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:
        yang:controller:threadpool">prefix:scheduled-
        threadpool</type>
    <name>global-netconf-ssh-scheduled-executor</name>
  </keepalive-executor>
</module>
```

C.3 Capability of Netconf Testtool

```
    <?xml version="1.0" encoding="UTF-8" standalone="no
        "?>
    <hello xmlns="urn:ietf:params:xml:ns:netconf:base
        :1.0">
    <capabilities>
    <capability>urn:ietf:params:xml:ns:yang:ietf-yang-
        types?module=ietf-yang-types&amp;revision
        =2010-09-24</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-netconf-
        monitoring?module=ietf-netconf-monitoring&amp;
        revision=2010-10-04</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-netconf-
        monitoring-extension?module=ietf-netconf-
        monitoring-extension&amp;revision=2013-12-10</
        capability>
    <capability>urn:ietf:params:netconf:capability:exi
        :1.0</capability>
```

```
<capability>urn:ietf:params:netconf:capability:
    candidate:1.0</capability>
<capability>urn:ietf:params:xml:ns:yang:ietf-inet-
    types?module=ietf-inet-types&amp;revision
    =2010-09-24</capability>
<capability>urn:ietf:params:netconf:base:1.1</
    capability>
<capability>urn:ietf:params:netconf:base:1.0</
    capability>
</capabilities>
<session-id>3</session-id>
</hello>
]]>]]>
```

Appendix D Capabilities of Emulated IHON Nodes

D.1 NodeA Capabilities

```
<node-id>NodeA</node-id>
    <port xmlns="urn:opendaylight:netconf-node-topology
        ">17830</port>
    <connection-status xmlns="urn:opendaylight:netconf-
        node-topology">connected</connection-status>
    <available-capabilities xmlns="urn:opendaylight:
        netconf-node-topology">
        <available-capability>urn:ietf:params:netconf:
            base:1.1</available-capability>
        <available-capability>(urn:ietf:params:xml:ns:
            yang:ietf-netconf-monitoring?revision
            =2010-10-04)ietf-netconf-monitoring</available
            -capability>
        <available-capability>urn:ietf:params:netconf:
            base:1.0</available-capability>
        <available-capability>urn:ietf:params:netconf:
            capability:exi:1.0</available-capability>
        <available-capability>(urn:opendaylight:router?
            revision=2016-04-15)ihonnode</available-
            capability>
        <available-capability>(urn:ietf:params:xml:ns:
            yang:ietf-inet-types?revision=2010-09-24)ietf-
            inet-types</available-capability>
        <available-capability>urn:ietf:params:netconf:
            capability:candidate:1.0</available-capability
            >
        <available-capability>(urn:ietf:params:xml:ns:
            yang:ietf-netconf-monitoring-extension?
```

```
                revision=2013-12-10)ietf-netconf-monitoring-
                extension</available-capability>
            <available-capability>(urn:ietf:params:xml:ns:
                yang:ietf-yang-types?revision=2010-09-24)ietf-
                yang-types</available-capability>
        </available-capabilities>
        <unavailable-capabilities xmlns="urn:opendaylight:
            netconf-node-topology"></unavailable-capabilities>
        <host xmlns="urn:opendaylight:netconf-node-topology
            ">127.0.0.1</host>
    </node>
```

D.2 NodeB Capabilities

```
    <node>
        <node-id>NodeB</node-id>
        <port xmlns="urn:opendaylight:netconf-node-topology
            ">17831</port>
        <connection-status xmlns="urn:opendaylight:netconf-
            node-topology">connected</connection-status>
        <available-capabilities xmlns="urn:opendaylight:
            netconf-node-topology">
            <available-capability>urn:ietf:params:netconf:
                base:1.1</available-capability>
            <available-capability>(urn:ietf:params:xml:ns:
                yang:ietf-netconf-monitoring?revision
                =2010-10-04)ietf-netconf-monitoring</available
                -capability>
            <available-capability>urn:ietf:params:netconf:
                base:1.0</available-capability>
            <available-capability>urn:ietf:params:netconf:
                capability:exi:1.0</available-capability>
            <available-capability>(urn:opendaylight:router?
                revision=2016-04-15)ihonnode</available-
                capability>
            <available-capability>(urn:ietf:params:xml:ns:
                yang:ietf-inet-types?revision=2010-09-24)ietf-
                inet-types</available-capability>
            <available-capability>urn:ietf:params:netconf:
                capability:candidate:1.0</available-capability
                >
        <available-capability>(urn:ietf:params:xml:ns:
            yang:ietf-netconf-monitoring-extension?
            revision=2013-12-10)ietf-netconf-monitoring-
            extension</available-capability>
        <available-capability>(urn:ietf:params:xml:ns:
            yang:ietf-yang-types?revision=2010-09-24)ietf-
            yang-types</available-capability>
    </available-capabilities>
    <unavailable-capabilities xmlns="urn:opendaylight:
        netconf-node-topology"></unavailable-capabilities>
    <host xmlns="urn:opendaylight:netconf-node-topology
        ">127.0.0.2</host>
</node>
```

D.3 NodeC Capabilities

```
<node>
    <node-id>NodeC</node-id>
    <port xmlns="urn:opendaylight:netconf-node-topology
        ">17832</port>
    <connection-status xmlns="urn:opendaylight:netconf-
        node-topology">connected</connection-status>
    <available-capabilities xmlns="urn:opendaylight:
        netconf-node-topology">
        <available-capability>urn:ietf:params:netconf:
            base:1.1</available-capability>
        <available-capability>(urn:ietf:params:xml:ns:
            yang:ietf-netconf-monitoring?revision
            =2010-10-04)ietf-netconf-monitoring</available
            -capability>
        <available-capability>urn:ietf:params:netconf:
            base:1.0</available-capability>
        <available-capability>urn:ietf:params:netconf:
            capability:exi:1.0</available-capability>
        <available-capability>(urn:opendaylight:router?
            revision=2016-04-15)ihonnode</available-
            capability>
        <available-capability>(urn:ietf:params:xml:ns:
            yang:ietf-inet-types?revision=2010-09-24)ietf-
            inet-types</available-capability>
        <available-capability>urn:ietf:params:netconf:
            capability:candidate:1.0</available-capability
```

```
              >
          <available-capability>(urn:ietf:params:xml:ns:
              yang:ietf-netconf-monitoring-extension?
              revision=2013-12-10)ietf-netconf-monitoring-
              extension</available-capability>
          <available-capability>(urn:ietf:params:xml:ns:
              yang:ietf-yang-types?revision=2010-09-24)ietf-
              yang-types</available-capability>
      </available-capabilities>
      <unavailable-capabilities xmlns="urn:opendaylight:
          netconf-node-topology"></unavailable-capabilities>
      <host xmlns="urn:opendaylight:netconf-node-topology
          ">127.0.0.3</host>
  </node>
```

D.4 Operational data

```
  <data>
      <netconf-state xmlns="urn:ietf:params:xml:ns:yang:
          ietf-netconf-monitoring">
          <sessions></sessions>
          <schemas>
              <schema>
                  <identifier>ietf-yang-types</identifier>
                  <version>2010-09-24</version>
                  <format>yang</format>
                  <location>NETCONF</location>
                  <namespace>urn:ietf:params:xml:ns:yang:
                      ietf-yang-types</namespace>
              </schema>
              <schema>
                  <identifier>ietf-netconf-monitoring</
                      identifier>
                  <version>2010-10-04</version>
                  <format>yang</format>
                  <location>NETCONF</location>
                  <namespace>urn:ietf:params:xml:ns:yang:
                      ietf-netconf-monitoring</namespace>
              </schema>
              <schema>
                  <identifier>ietf-netconf-monitoring-
                      extension</identifier>
```

```
                <version>2013−12−10</version>
                <format>yang</format>
                <location>NETCONF</location>
                <namespace>urn:ietf:params:xml:ns:yang:
                    ietf−netconf−monitoring−extension</
                    namespace>
            </schema>
            <schema>
                <identifier>ihonnode</identifier>
                <version>2016−04−15</version>
                <format>yang</format>
                <location>NETCONF</location>
                <namespace>urn:opendaylight:ihonnode</
                    namespace>
            </schema>
            <schema>
                <identifier>ietf−inet−types</identifier>
                <version>2010−09−24</version>
                <format>yang</format>
                <location>NETCONF</location>
                <namespace>urn:ietf:params:xml:ns:yang:
                    ietf−inet−types</namespace>
            </schema>
        </schemas>
    </netconf−state>
</data>
```

Appendix E
Configuration of H1 nodes at Uninett

E.1 N1 Capability

```
<node>
            <node-id>N1</node-id>
            <port xmlns="urn:opendaylight:netconf-node-
                topology">830</port>
            <unavailable-capabilities xmlns="urn:
                opendaylight:netconf-node-topology">
                <unavailable-capability>
                    <capability>(http://netconfcentral.
                        org/ns/yuma-system?revision
                        =2014-11-27)yuma-system</
                        capability>
                    <failure-reason>unable-to-resolve</
                        failure-reason>
                </unavailable-capability>
                <unavailable-capability>
                    <capability>(urn:ietf:params:xml:ns:
                        yang:ietf-netconf-with-defaults?
                        revision=2011-06-01)ietf-netconf-
                        with-defaults</capability>
                    <failure-reason>unable-to-resolve</
                        failure-reason>
                </unavailable-capability>
                <unavailable-capability>
                    <capability>(urn:ietf:params:xml:ns:
                        netmod:notification?revision
                        =2008-07-14)nc-notifications</
                        capability>
                <failure-reason>unable-to-resolve</failure-reason>
            </unavailable-capability>
            <unavailable-capability>
                <capability>(http://netconfcentral.org/ns/yuma-time-filter?revision=2011-08-13)yuma-time-filter</capability>
                <failure-reason>unable-to-resolve</failure-reason>
            </unavailable-capability>
            <unavailable-capability>
                <capability>(urn:ietf:params:xml:ns:netconf:notification:1.0?revision=2008-07-14)notifications</capability>
                <failure-reason>unable-to-resolve</failure-reason>
            </unavailable-capability>
            <unavailable-capability>
                <capability>(urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring?revision=2010-10-04)ietf-netconf-monitoring</capability>
                <failure-reason>unable-to-resolve</failure-reason>
            </unavailable-capability>
        </unavailable-capabilities>
        <connection-status xmlns="urn:opendaylight:netconf-node-topology">connected</connection-status>
        <available-capabilities xmlns="urn:opendaylight:netconf-node-topology">
            <available-capability>(http://transpacket.com/ns/hadm1-interfaces-ethernet-switching?revision=2014-12-16)interfaces-ethernet-switching</available-capability>
            <available-capability>(http://transpacket.com/ns/hadm1-interfaces-inet?revision=2014-07-16)interfaces-inet</available
        -capability>
    <available-capability>(http://transpacket
        .com/ns/interfaces-hadm1?revision
        =2014-07-15)interfaces-hadm1</
        available-capability>
    <available-capability>(urn:ietf:params:
        xml:ns:yang:ietf-system?revision
        =2014-08-06)ietf-system</available-
        capability>
    <available-capability>urn:ietf:params:
        netconf:capability:confirmed-commit
        :1.1</available-capability>
    <available-capability>urn:ietf:params:
        netconf:capability:confirmed-commit
        :1.0</available-capability>
    <available-capability>urn:ietf:params:
        netconf:capability:validate:1.1</
        available-capability>
    <available-capability>urn:ietf:params:
        netconf:capability:validate:1.0</
        available-capability>
    <available-capability>urn:ietf:params:
        netconf:capability:candidate:1.0</
        available-capability>
    <available-capability>(urn:ietf:params:
        xml:ns:yang:iana-crypt-hash?revision
        =2014-04-04)iana-crypt-hash</available
        -capability>
    <available-capability>(http://
        netconfcentral.org/ns/yuma-proc?
        revision=2012-10-10)yuma-proc</
        available-capability>
    <available-capability>(http://transpacket
        .com/ns/rmon?revision=2012-11-01)rmon
        </available-capability>
    <available-capability>(urn:ietf:params:
        xml:ns:yang:ietf-yang-types?revision
        =2013-07-15)ietf-yang-types</available
        -capability>
    <available-capability>(http://transpacket
        .com/ns/protocols?revision=2013-10-14)
```

```
protocols</available-capability>
<available-capability>urn:ietf:params:netconf:base:1.1</available-capability>
<available-capability>urn:ietf:params:netconf:base:1.0</available-capability>
<available-capability>(http://transpacket.com/ns/ntp?revision=2014-02-27)ntp</available-capability>
<available-capability>(urn:ietf:params:xml:ns:yang:smiv2:SNMPv2-TC?revision=1999-04-01)SNMPv2-TC</available-capability>
<available-capability>(http://transpacket.com/ns/routes?revision=2012-11-01)routes</available-capability>
<available-capability>urn:ietf:params:netconf:capability:interleave:1.0</available-capability>
<available-capability>(http://transpacket.com/ns/hwmodel?revision=2012-11-01)hwmodel</available-capability>
<available-capability>(http://transpacket.com/ns/transpacket-system?revision=2014-11-27)transpacket-system</available-capability>
<available-capability>(http://netconfcentral.org/ns/yuma-arp?revision=2012-01-13)yuma-arp</available-capability>
<available-capability>(http://netconfcentral.org/ns/yuma-app-common?revision=2012-08-16)yuma-app-common</available-capability>
<available-capability>urn:ietf:params:netconf:capability:xpath:1.0</available-capability>
<available-capability>(urn:ietf:params:xml:ns:yang:yang-smi?revision=2008-03-20)yang-smi</available-
```

```
        capability>
    <available-capability>urn:ietf:params:
        netconf:capability:rollback-on-error
        :1.0</available-capability>
    <available-capability>(http://
        netconfcentral.org/ns/yuma-mysession?
        revision=2010-05-10)yuma-mysession</
        available-capability>
    <available-capability>(http://transpacket
        .com/ns/syslog?revision=2012-11-01)
        syslog</available-capability>
    <available-capability>(http://transpacket
        .com/ns/hadm1-vlans?revision
        =2012-11-05)vlans</available-
        capability>
    <available-capability>(http://transpacket
        .com/ns/hadm?revision=2013-11-08)hadm
        </available-capability>
    <available-capability>(urn:ietf:params:
        xml:ns:netconf:partial-lock:1.0?
        revision=2009-10-19)ietf-netconf-
        partial-lock</available-capability>
    <available-capability>(http://transpacket
        .com/ns/http?revision=2014-09-09)http
        </available-capability>
    <available-capability>urn:ietf:params:
        netconf:capability:notification:1.0</
        available-capability>
    <available-capability>(http://transpacket
        .com/ns/hadm1-interfaces-mtu?revision
        =2014-07-16)interfaces-mtu</available-
        capability>
    <available-capability>(http://transpacket
        .com/ns/oids?revision=2012-11-01)oids
        </available-capability>
    <available-capability>(http://
        netconfcentral.org/ns/yuma-types?
        revision=2012-06-01)yuma-types</
        available-capability>
    <available-capability>(urn:ietf:params:
        xml:ns:yang:ietf-netconf-acm?revision
```

```
          =2012-02-22)ietf-netconf-acm</
          available-capability>
        <available-capability>urn:ietf:params:
          netconf:capability:partial-lock:1.0</
          available-capability>
        <available-capability>(http://
          netconfcentral.org/ns/yuma-ncx?
          revision=2012-01-13)yuma-ncx</
          available-capability>
        <available-capability>(urn:ietf:params:
          xml:ns:yang:iana-if-type?revision
          =2014-07-03)iana-if-type</available-
          capability>
        <available-capability>(http://transpacket
          .com/ns/uboot?revision=2012-11-03)
          uboot</available-capability>
        <available-capability>(http://transpacket
          .com/ns/snmp?revision=2012-11-01)snmp
          </available-capability>
        <available-capability>(http://transpacket
          .com/ns/hadm1-alarms?revision
          =2014-03-21)alarms</available-
          capability>
        <available-capability>(http://transpacket
          .com/ns/hadm1-interfaces-loopback?
          revision=2014-07-16)interfaces-
          loopback</available-capability>
        <available-capability>(http://transpacket
          .com/ns/hadm1-interfaces-traffic-
          generator?revision=2014-07-16)
          interfaces-traffic-generator</
          available-capability>
        <available-capability>(urn:ietf:params:
          xml:ns:yang:ietf-interfaces?revision
          =2014-05-08)ietf-interfaces</available
          -capability>
        <available-capability>(http://transpacket
          .com/ns/hadm1-interfaces-optics-
          diagnostics?revision=2014-07-10)
          interfaces-optics-diagnostics</
          available-capability>
```

```
                <available-capability>(http://transpacket.com/ns/requests?revision=2012-11-01)requests</available-capability>
                <available-capability>(urn:ietf:params:xml:ns:yang:ietf-inet-types?revision=2013-07-15)ietf-inet-types</available-capability>
                <available-capability>(http://transpacket.com/ns/hadm1-gst-slot-sync?revision=2014-12-12)protocols-gst-slot-sync</available-capability>
                <available-capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&amp;also-supported=trim,report-all,report-all-tagged</available-capability>
                <available-capability>urn:ietf:params:netconf:capability:url:1.0?scheme=file</available-capability>
                <available-capability>(http://transpacket.com/ns/hadm1-interfaces-policer?revision=2014-06-24)interfaces-policer</available-capability>
                <available-capability>(http://transpacket.com/ns/synchronization?revision=2013-11-21)synchronization</available-capability>
            </available-capabilities>
            <host xmlns="urn:opendaylight:netconf-node-topology">192.168.209.1</host>
        </node>
```

E.2 N1 Configuration

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <vlans xmlns="http://transpacket.com/ns/hadm1-vlans">
        <vlan>
            <name>ge3</name>
            <id>13</id>
        </vlan>
        <vlan>
            <name>ge4</name>
            <id>14</id>
        </vlan>
        <vlan>
            <name>ge1</name>
            <gst-slot-sync-index xmlns="http://
                transpacket.com/ns/hadm1-gst-slot-sync
                ">1</gst-slot-sync-index>
            <id>11</id>
        </vlan>
        <vlan>
            <name>ge2</name>
            <id>12</id>
        </vlan>
        <vlan>
            <name>ge0</name>
            <gst-slot-sync-index xmlns="http://
                transpacket.com/ns/hadm1-gst-slot-sync
                ">0</gst-slot-sync-index>
            <id>10</id>
        </vlan>
        <vlan>
            <name>master</name>
            <id>0</id>
        </vlan>
        <vlan>
            <name>xe0</name>
            <id>100</id>
        </vlan>
        <vlan>
            <name>ge5</name>
            <id>15</id>
        </vlan>
    </vlans>
    <snmp xmlns="http://transpacket.com/ns/snmp">
        <trap-groups>
            <trap-group>
                <name>transpacket</name>
            </trap-group>
        </trap-groups>
        <communities>
            <community>
```

```
                <name>transpacket</name>
            </community>
        </communities>
    </snmp>
    <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"></nacm>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
            <name>ge3</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge3</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">gst</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge4</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge4</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">gst</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge5</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge5</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">sm</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>me0</name>
            <inet xmlns="http://transpacket.com/ns/hadm1-interfaces-inet">
                <address>192.168.209.1/24</address>
            </inet>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>xe1</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <trunk-interface>
                    <vlans>
                        <vlan>
                            <vlan-name>ge0</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>xe0</vlan-name>
                        </vlan>
                        <vlan>
```

```
                            <vlan-name>ge2</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge1</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge4</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge3</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge5</vlan-name>
                        </vlan>
                    </vlans>
                </trunk-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>xe0</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <trunk-interface>
                    <vlans>
                        <vlan>
                            <vlan-name>ge0</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge1</vlan-name>
                        </vlan>
                    </vlans>
                </trunk-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge2</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge2</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">gst</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
    </interfaces>
    <routes xmlns="http://transpacket.com/ns/routes">
        <route>
            <destination-prefix>0.0.0.0/0</destination-prefix>
            <next-hop>158.38.152.1</next-hop>
        </route>
    </routes>
    <protocols xmlns="http://transpacket.com/ns/protocols">
        <gst-slot-sync xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">
            <master-sync-vlan>master</master-sync-vlan>
        </gst-slot-sync>
    </protocols>
    <arp xmlns="http://netconfcentral.org/ns/yuma-arp"></arp>
    <hadm xmlns="http://transpacket.com/ns/hadm"></hadm>
</data>
```

E.3 N2 Configuration

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <vlans xmlns="http://transpacket.com/ns/hadm1-vlans">
        <vlan>
            <name>ge3</name>
            <id>13</id>
        </vlan>
        <vlan>
            <name>ge4</name>
            <id>14</id>
        </vlan>
        <vlan>
            <name>ge1</name>
            <gst-slot-sync-index xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">1</gst-slot-sync-index>
            <id>11</id>
        </vlan>
        <vlan>
            <name>ge2</name>
            <id>12</id>
        </vlan>
        <vlan>
            <name>ge0</name>
            <gst-slot-sync-index xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">0</gst-slot-sync-index>
            <id>10</id>
        </vlan>
        <vlan>
            <name>master</name>
            <id>0</id>
        </vlan>
        <vlan>
            <name>xe0</name>
            <id>100</id>
        </vlan>
        <vlan>
            <name>ge5</name>
            <id>15</id>
        </vlan>
    </vlans>
    <snmp xmlns="http://transpacket.com/ns/snmp">
        <trap-groups>
            <trap-group>
                <name>transpacket</name>
            </trap-group>
        </trap-groups>
        <communities>
            <community>
                <name>transpacket</name>
            </community>
        </communities>
    </snmp>
    <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"></nacm>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
            <name>ge3</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge3</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">gst</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge4</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge4</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">gst</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge5</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge5</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">sm</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>me0</name>
            <inet xmlns="http://transpacket.com/ns/hadm1-interfaces-inet">
                <address>192.168.209.2/24</address>
            </inet>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>xe1</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <trunk-interface>
                    <vlans>
                        <vlan>
                            <vlan-name>ge0</vlan-name>
                        </vlan>
                        <vlan>
                        <vlan-name>xe0</vlan-name>
                    </vlan>
                    <vlan>
                        <vlan-name>ge2</vlan-name>
                    </vlan>
                    <vlan>
                        <vlan-name>ge1</vlan-name>
                    </vlan>
                    <vlan>
                        <vlan-name>ge4</vlan-name>
                    </vlan>
                    <vlan>
                        <vlan-name>ge3</vlan-name>
                    </vlan>
                    <vlan>
                        <vlan-name>ge5</vlan-name>
                    </vlan>
                </vlans>
            </trunk-interface>
        </ethernet-switching>
        <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
    </interface>
    <interface>
        <name>xe0</name>
        <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
            <trunk-interface>
                <vlans>
                    <vlan>
                        <vlan-name>ge0</vlan-name>
                    </vlan>
                    <vlan>
                        <vlan-name>ge1</vlan-name>
                    </vlan>
                </vlans>
            </trunk-interface>
        </ethernet-switching>
        <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge2</name>
            <ethernet-switching xmlns="http://transpacket
                .com/ns/hadm1-interfaces-ethernet-
                switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge2</vlan-name>
                        <priority xmlns="http://
                            transpacket.com/ns/hadm1-gst-
                            slot-sync">gst</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:
                iana-if-type">x:ethernetCsmacd</type>
        </interface>
    </interfaces>
    <routes xmlns="http://transpacket.com/ns/routes">
        <route>
            <destination-prefix>0.0.0.0/0</destination-
                prefix>
            <next-hop>158.38.152.1</next-hop>
        </route>
    </routes>
    <protocols xmlns="http://transpacket.com/ns/protocols
        ">
        <gst-slot-sync xmlns="http://transpacket.com/ns/
            hadm1-gst-slot-sync">
            <master-sync-vlan>master</master-sync-vlan>
        </gst-slot-sync>
    </protocols>
    <arp xmlns="http://netconfcentral.org/ns/yuma-arp"></
        arp>
    <hadm xmlns="http://transpacket.com/ns/hadm"></hadm>
</data>
```

E.4 N3 Configuration

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <vlans xmlns="http://transpacket.com/ns/hadm1-vlans">
```

```
        <vlan>
            <name>ge3</name>
            <id>13</id>
        </vlan>
        <vlan>
            <name>ge4</name>
            <id>14</id>
        </vlan>
        <vlan>
            <name>ge1</name>
            <id>11</id>
        </vlan>
        <vlan>
            <name>ge2</name>
            <id>12</id>
        </vlan>
        <vlan>
            <name>ge0</name>
            <id>10</id>
        </vlan>
        <vlan>
            <name>master</name>
            <id>0</id>
        </vlan>
        <vlan>
            <name>xe0</name>
            <id>100</id>
        </vlan>
        <vlan>
            <name>ge8</name>
            <id>15</id>
        </vlan>
    </vlans>
    <snmp xmlns="http://transpacket.com/ns/snmp">
        <trap-groups>
            <trap-group>
                <name>transpacket</name>
            </trap-group>
        </trap-groups>
        <communities>
            <community>
```

```
                <name>transpacket</name>
            </community>
        </communities>
    </snmp>
    <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"></nacm>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
            <name>me0</name>
            <inet xmlns="http://transpacket.com/ns/hadm1-interfaces-inet">
                <address>192.168.209.3/24</address>
            </inet>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge8</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge8</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">sm</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge3</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
```

```
                        <vlan-name>ge3</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">gst</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge4</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge4</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">gst</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge0</name>
            <ethernet-switching xmlns="http://transpacket.com/ns/hadm1-interfaces-ethernet-switching">
                <access-interface>
                    <vlan>
                        <vlan-name>ge0</vlan-name>
                        <priority xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">gst</priority>
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:
                iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>xe1</name>
            <ethernet-switching xmlns="http://transpacket
                .com/ns/hadm1-interfaces-ethernet-
                switching">
                <trunk-interface>
                    <vlans>
                        <vlan>
                            <vlan-name>ge0</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>xe0</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge2</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge1</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge4</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge3</vlan-name>
                        </vlan>
                        <vlan>
                            <vlan-name>ge8</vlan-name>
                        </vlan>
                    </vlans>
                </trunk-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:
                iana-if-type">x:ethernetCsmacd</type>
        </interface>
        <interface>
            <name>ge1</name>
            <ethernet-switching xmlns="http://transpacket
                .com/ns/hadm1-interfaces-ethernet-
                switching">
                 <access-interface>
                     <vlan>
                         <vlan-name>ge1</vlan-name>
                         <priority xmlns="http://
                             transpacket.com/ns/hadm1-gst-
                             slot-sync">gst</priority>
                     </vlan>
                 </access-interface>
             </ethernet-switching>
             <type xmlns:x="urn:ietf:params:xml:ns:yang:
                iana-if-type">x:ethernetCsmacd</type>
         </interface>
         <interface>
             <name>xe0</name>
             <ethernet-switching xmlns="http://transpacket
                .com/ns/hadm1-interfaces-ethernet-
                switching">
                 <access-interface>
                     <vlan>
                         <vlan-name>xe0</vlan-name>
                         <priority xmlns="http://
                             transpacket.com/ns/hadm1-gst-
                             slot-sync">sm</priority>
                     </vlan>
                 </access-interface>
             </ethernet-switching>
             <type xmlns:x="urn:ietf:params:xml:ns:yang:
                iana-if-type">x:ethernetCsmacd</type>
         </interface>
         <interface>
             <name>ge2</name>
             <ethernet-switching xmlns="http://transpacket
                .com/ns/hadm1-interfaces-ethernet-
                switching">
                 <access-interface>
                     <vlan>
                         <vlan-name>ge2</vlan-name>
                         <priority xmlns="http://
                             transpacket.com/ns/hadm1-gst-
                             slot-sync">gst</priority>
```

```
                    </vlan>
                </access-interface>
            </ethernet-switching>
            <type xmlns:x="urn:ietf:params:xml:ns:yang:iana-if-type">x:ethernetCsmacd</type>
        </interface>
    </interfaces>
    <routes xmlns="http://transpacket.com/ns/routes">
        <route>
            <destination-prefix>0.0.0.0/0</destination-prefix>
            <next-hop>158.38.152.1</next-hop>
        </route>
    </routes>
    <protocols xmlns="http://transpacket.com/ns/protocols">
        <gst-slot-sync xmlns="http://transpacket.com/ns/hadm1-gst-slot-sync">
            <master-sync-vlan>master</master-sync-vlan>
        </gst-slot-sync>
    </protocols>
    <arp xmlns="http://netconfcentral.org/ns/yuma-arp"></arp>
    <hadm xmlns="http://transpacket.com/ns/hadm"></hadm>
</data>
```