Analysis of OpenFlow and NETCONF as SBIs in Managing the Optical Link Interconnecting Data Centers in an SDN Environment

by

KARPAKAMURTHY MUTHUKUMAR

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science
in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

Abstract

SDN technology has primarily been applied to all types and size of network ranging from Ethernet services to large cloud environment. More recently, interest has turned towards extending programmability of the OTN. In the SDN architecture, SBIs are used to communicate between the SDN controller and the switches or routers in the network. In this thesis we deploy two major protocols as SBIs in managing BoD across the interconnected data centers over the OTN. The OpenFlow and NETCONF are the two SBIs deployed between the OpenDayLight controller and the BTI7800 optical edge transponders. We present the OpenVSwitch architecture modified for referencing the optical ports of the BTI7800 devices. Experimental demonstration and performance of both BTI YANG-based NETCONF and OpenFlow protocols are presented. Our experiments show that NETCONF is faster and efficiently handles the control message; whereas in a fully loaded system, OpenFlow offers better bandwidth utilization.

Acknowledgements

This thesis would have been impossible without the support and the encouragement I received from several people.

First and foremost, I am deeply grateful to my supervisor, Professor Thomas Kunz for guiding me throughout my master's program and supporting me in every step of the thesis work.

I am grateful to Peter Landon (Juniper, Canada) for his support and guidance during my thesis work.

I am thankful to my colleagues and friends for sharing their experiences and thoughts on various aspects of the research.

Finally, I want to thank my parents and my sisters who have always encouraged me.

Table of Contents

List of Tables

List of Illustrations

List of Acronyms

AAA	Authentication, Authorization, and Accounting
AC	Application Controller
API	Application Programming Interface
BICs	BTI Interface Cards
BoD	Bandwidth on Demand
CLI	Command Line Interface
CMM	Chassis Management Module
DCON	Data Center Optical Networks
DWDM	Dense Wavelength Division Multiplexing
DynPaC	Dynamic Path Computation
eSDN	enhanced Software Defined Networking
FTP	File Transfer Protocol
GMPLS	Generalized Multiprotocol Label Switching
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IRS	Internet Routing System
MD-SAL	Model Driven – Service Abstraction Layer
NBIs	NorthBound Interfaces
NETCONF	Network Configuration Protocol
OAM	operations, administration, and maintenance

ODL	OpenDayLight
OF	OpenFlow
OIF	Optical Internetworking Forum
O-NEs	Optical Network Elements
ONF	Open Network Foundation
OpenSig	Open Signaling
O-SDNC	Optical SDN Controller
OTN	Optical Transport Network
OTWG	Optical Transport Working Group
PCE	Path Computation Element
PCEP	Path Computation Element Protocol
QoS	Quality of Service
QoT	Quality of Transmission
REST	REpresentational State Transfer
ROADMs	Reconfigurable Optical Add Drop Multiplexers
RPC	Remote Procedure Call
SAL	Service Abstraction Layer
SBI	SouthBound Interface
SDN	Software Defined Networking
SD-OTN	Software Defined Optical Transport Network
SFTP	SSH File Transfer Protocol
SLA	Service Level Agreements
SNMP	Simple Network Management Protocol

SWAN-C&M	SDN Wide Area Network Control and Management
TC	Transport Controller
UFMs	Universal Forwarding Modules
XML	eXtensible Markup Language
YANG	Yet Another Next Generation

List of Appendices

Chapter 1: Introduction

This chapter introduces the objectives of the thesis. It explains the motivation and lists the contributions of the thesis research. The chapter wraps up by explaining the organization of this thesis document.

1.1 Motivation

Recent years have witnessed an unprecedented growth in the number of data centers being built by large cloud service providers. To provide flexible and reliable services at the global scale, cloud service providers have deployed multiple data centers in different geographical areas, often spanning continents that are interconnected via private high-speed backbone networks, offering hundreds of Gbps or tens of Tbps bandwidth [21].

The inefficiency of current inter data center backbone networks stems from three aspects [21]. First, the lack of effective control techniques cannot make efficient use of the network resources. With no coordination, each application or service now can send however much traffic whenever it wants, in oblivion to the current network load. As a result, the bandwidth needs to be over-provisioned in the network to be able to handle the superimposed peak demand. In reality, with a little coordination, the demand for bandwidth could be reduced by postponing the delivery of delay-tolerant services to off-peak periods. Second, the widely varying performance requirements of applications are usually ignored. For example, interactive applications (e.g. web search) are delay-sensitive, while background applications (e.g. data synchronization between data centers) are throughput-sensitive. Third, it is known that traffic engineering with traditional distributed routing protocols (e.g. link state) is suboptimal in most cases. Distributed

approaches are often inflexible and hardly lend themselves to the deployment of sophisticated resource sharing principles such as fair bandwidth sharing between services with priorities or multi-path forwarding to balance application traffic in response to link failures and demand changes.

The emerging Software Defined Networking (SDN) technique has recently been used for inter data center traffic management to address the above-mentioned inefficiencies of traditional approaches. Transport networks are evolving to be more and more automated and driven by software to minimize the operational costs and to provide new services and applications in a quicker and more efficient way [25]. Several transport technologies such as Dense Wavelength Division Multiplexing (DWDM) and Reconfigurable Optical Add Drop Multiplexer (ROADM) etc. are available for interconnecting the data centers, each of which provides various transport optical features [19]. Moreover, management, configuration or debugging procedures remain a daunting task in data centers mainly because of multiple vendor-specific proprietary assets such as switches/routers requiring their own proprietary procedures rather than a simple and unified process. Similarly, traffic management and policy enforcement can become very important and critical issues, as data centers are expected to continuously achieve high levels of performance [18]. But if one wants to extend the SDN approach to the physical photonic layer, then the SDN controller must take the nature of the optical transmission into account.

1.2 Objective

The thesis starts with understanding how SDN plays an important role in shaping an emerging network architecture that deals with efficient bandwidth utilization depending

on the nature of today's applications. We review the current state-of-art in managing Bandwidth on Demand (BoD) across interconnected data centers in an SDN environment and the advancement of SDN in the field of optical networking. In the architecture of a Software Defined Network, the SouthBound Interfaces (SBIs) are used to communicate between the controller and the SDN network elements such as switches and routers. The main step towards the thesis is to effectively manage BoD across the data centers that are interconnected by an Optical Transport Network (OTN). The SDN controller, with the help of the SBIs is capable of controlling the data plane devices located at the data centers that are interconnected. In our thesis, we have deployed two SBIs. The first one is the OpenFlow protocol, developed by the Open Networking Foundation (ONF). It is an industry standard that defines how to interact with the SDN forwarding plane to make adjustments to the network, so it can better adapt to the changing business needs. Network Configuration Protocol (NETCONF) is an existing network management protocol that is alternatively supported as a SBIs in the SDN architecture. The main objective of the thesis is to evaluate the performance of both protocols as SBIs in an interconnected data center over an optical backbone.

1.3 Contribution

The thesis compares two protocols that serve as SBIs in handling BoD requirements across the data centers that are interconnected with the help of an optical backbone and managed using SDN. As discussed in Section 1.2, OpenFlow and NETCONF are two protocols that are implemented. The contributions of this thesis are:

- The OpenFlow specification does not support any optical extensions for optical devices to reference the information regarding optical ports and interfaces. The thesis focuses on controlling the transport optical network resources formed by interconnecting the data centers. In order to reference the optical ports and connections of BTI7800 using OpenFlow protocol, the optical extensions were developed within the OpenVSwitch code and ported onto the BTI7800's kernel environment.
- We evaluate the capabilities of both protocols as SBIs in managing BoD requirement between two data centers. Depending on the availability of resources and nature of traffic demands, the SDN controller dynamically determines the flow rules and communicates those rules to the network elements using the implemented SBIs. The use cases discussed in this thesis help us to quantitatively and qualitatively compare both protocols implemented as SBIs.
- NETCONF as SBI in managing BoD across the BTI7800 devices interconnecting data centers was showcased as a demo at the "SDN & OpenFlow World Congress 2015, Dusseldorf, Germany". Refer to Appendix E for the information related to the demo and the use case.

1.4 Outline of the Thesis

The chapters of the thesis are organized in the following manner. Chapter 2 provides background information about the nature and challenges faced by the traditional interconnected data centers. It introduces the SDN architecture and discusses the benefits of SDN in network management. Then, the traditional network management protocol

NETCONF and the ONF recommended OpenFlow protocol as SBIs in an SDN environment are presented. Finally, it reviews the state-of-art literature related to BoD management across data centers in an SDN environment and literature related to the SDN advancement in the field of optical networks. Chapter 3 introduces the BTI7800 optical network element and its high-level architecture. It presents the architecture of the NETCONF-based SBI implementation on the BTI7800. It further presents the developed optical extension to the OpenFlow protocol and its implementation on the BTI7800. The chapter presents the working of both implemented SBIs in managing the BTI7800 optical data plane devices interconnecting data centers. Chapter 4 discusses the details of the test environment. Chapter 4 and Chapter 5 discusses the use cases derived from the ONF SDN use case document [7] and it presents the results collected when exercising those use cases for both protocols as SBIs. Chapter 6 concludes the thesis and provides the directions for possible future work.

Chapter 2: Background Work and Literature Review

To better understand the scope of research, this chapter talks about the issues faced by traditional networking. It introduces the concept of SDN and briefs about the characteristics and the principles behind this new technology. This chapter further presents the applicability of SDN in selective fields of networking based on the research focus. Section 2.3 introduces the OpenDayLight (ODL) controller and its high level architecture, followed by a brief introduction of NETCONF and OpenFlow SBIs. Finally, the chapter concludes with a review of related research work.

2.1 Issues Faced with Traditional Data Center and Network Management

Nowadays, the Internet develops at a rate that outpaces all existing data networks in both speed and scale. As the variety of real-time services such as video, audio, cloud data center, and mobile services continue to develop and as a result, they form a bottleneck for bandwidth requirement. The existing traditional network management cannot deliver the expected service quality, satisfying all bandwidth requests [18]. The traditional network faces these problems:

- **Inefficient Service Deployment**: In the traditional environments, services and networks are deployed separately. Most networks are configured using commands or through network management systems, and are managed statically or through network automation, which limits the control over the network and their entities. These networks are essentially static and inefficient at deploying dynamic services that require timely adjustments. In extreme cases, these networks may even fail to support such services.

- **Slow Adaptation to New Services**: It takes a long time in a traditional network environment to upgrade features, adjust architectures, or to introduce new device(s) to meet the new service requirements. For example: in a traditional optical data centric network, if one of the applications is requesting additional bandwidth, the network cannot quickly adopt to the requirement as it requires manual provisioning. The network manager has to provision each and every device separately through its respective Command Line Interface (CLI) in order to accommodate the requirement and it turns out to be time consuming and inefficient.
- **Lack of User Experience**: Most of the network applications are connectionless, providing only minimum bandwidth service quality. The lack of quality in the service delivered, results in situations that thwart user expectations for a quality experience.

Over the years, many solutions have been proposed to solve the problems faced by the traditional networks. These problems cannot be solved without focusing on the fundamental design of the network infrastructure, which requires decoupling between the network and the service that are transparent to each other. The two most significant early efforts, which proposed ways of separating the control software from the underlying hardware and provide an open interface for management and control, are the Open Signaling (OpenSig) working group and the Active Networking initiative. Although these early approaches envisioned innovative open networking programmability, they could not achieve widespread success because of these shortcomings [16]:

a) The approaches were promoting data instead of control plane programmability.

b) They focused on proposing innovative programming architecture models and platforms, paying little or no attention to practical issues like the performance and the security they offered.

c) The approaches advocated, as one of its advantages, the flexibility it would give to end users to program the network. Even though, in reality, the scenario of end user programmability was really rare.

The traditional networking devices were designed to support specific protocols essential for the operation of the network and these network devices were less dynamic and were not really flexible enough to adapt to major changes. Due to the vast expansion of the Internet of Things, the network grew bigger in size and it became tedious to manage each and every device individually. At the same time, modifying the control logic of such devices was not a good option, causing hindrance to the network evolution. To remedy this situation, various efforts were taken focused on finding novel solution(s) that eventually lead to a new era of more open, extensible and programmable networks [18].

2.2 Software Defined Networking

SDN, is arguably one of the most significant paradigm shifts in the networking industry in recent years. SDN is a control framework that supports programmability of network functions and protocols, by decoupling the data plane and the control plane, which are currently integrated vertically in most network equipment. The decoupling of planes allows the underlying infrastructure to be abstracted and used by the application(s) and the network service(s) as a virtual entity.

Designing and managing the computer networks can become a very daunting task due to the high level of complexity involved. The tight bonding between a network control plane (where the decisions of handling traffic are made) and data plane (where the actual forwarding of traffic takes place) gives rise to various challenges related to its management and evolution. Introducing or upgrading new functionality like intrusion detection systems and load balancers would impact the network infrastructure and possess a direct impact on its logic. The concept of programmable networks has been proposed as a means to remedy this situation, by promoting innovation in network management and deployment of network services using some sort of an open network Application Programming Interface (API). Therefore, it produces flexible networks that are able to operate according to the user's need, are reprogrammable and perform numerous tasks without the need for continuous modification of the underlying hardware platform. SDN is seen as one way to solve some problems of the Internet, including security, managing complexity, multi-casting, load balancing, and energy efficiency. The split architecture of SDN makes it feasible to control, monitor, and manage the network(s) from a centralized node also termed as SDN controller [1][2][15].

Figure 2.1 depicts the SDN layered architecture. The bottom most layer is the data plane, and it is composed of physical hardware and virtual network entities (routers, switches etc.) that are interconnected. Data plane devices act as per the controller's instructions, and are mainly responsible for forwarding packets onto their destination. The middle layer is the control plane and it may be a single or series of controller(s) that are logically centralized. According to SDN, the software control program is referred to as the

controller, and it is responsible for managing and storing information about the underlying data plane devices, the network and its topology.

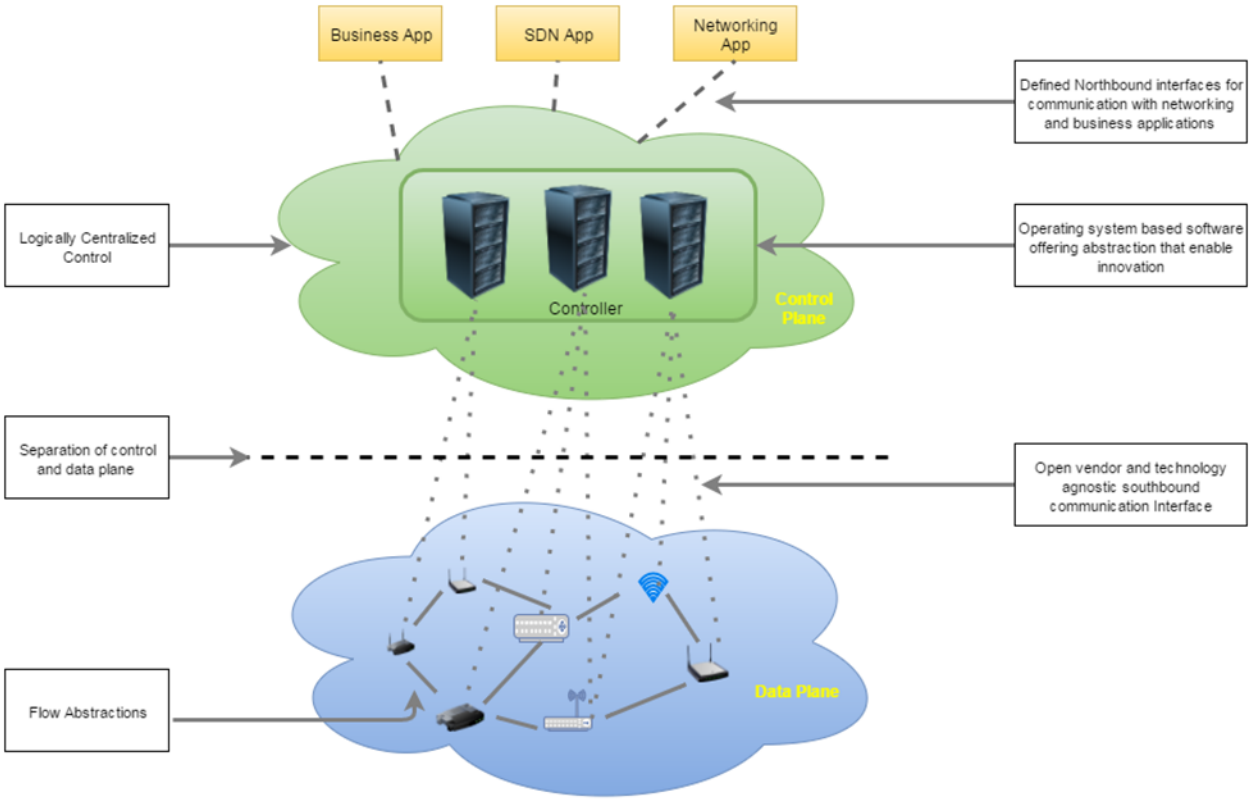


Figure 2.1 SDN Overview

The controller translates network rules and regulations in term of flows that help in packet handling decisions and routing. The control plane communicates with the data plane devices through the SBIs. OpenFlow is the first standard communication interface defined between the control and forwarding layers of the SDN architecture. OpenFlow allows direct access for manipulating the forwarding plane of the network devices such as switches and routers, both physical and virtual (hypervisor based). The top most layer is the application layer that transforms the user requirements to the underlying controller and vice versa. The application layer communicates with the controller elements using the NorthBound Interfaces (NBIs) and vice versa.

2.2.1 Principle of Software Defined Networking

In SDN, the controller functions more like a device driver software that initiates and configures the devices to some specific operational settings by reading input from the applications and writing into the device's memory. To the underlying network, the controller is a piece of software that manages and shares all the resources of the underlying network infrastructure and amongst the applications.

The concept of SDN is to place a network operating system between the network infrastructure and the application layer. It is the responsibility of the network operating system to coordinate and manage the resources of the complete network and to reveal an abstract unified view of all components of the network to the applications executed on top of it. The concept is analogous to the principle of a typical computer system, where the operating system lies between the hardware and the user space and is responsible for managing the hardware resources and providing common services for user programs. Similarly, SDN offers a logically centralized environment where network administrators and developers can typically program, configure and control the network.

2.2.2 Characteristics of SDN

SDN is proposed as a solution for the problems faced with the traditional network architecture and to support future network growth. The characteristics of SDN technologies are summarized as follows [2]:

- **Decoupling of Control and Data Plane**: This principle calls for separable controller and data planes. However, it is understood that control must necessarily be exercised within data plane systems. The SBI between the SDN controller and

the network element is defined in such a way that the SDN controller can delegate significant functionality to the network element, while remaining aware of the network element states.

- **Centralized Control Architecture**: The SDN architecture removes the control functions from the network devices and moves them into a logically separate server, called the controller. In this architecture, the network devices forward traffic based on the control data delivered by the controller. The controller requires no direct knowledge of the network architecture. The Open Network Foundation (ONF) currently sponsors this architecture and SDN-related protocols like OpenFlow. However, the IETF and some equipment vendors believe that traditional device control protocols, such as NETCONF, Simple Network Management Protocol (SNMP), Path Computation Element Protocol (PCEP), and the latest, Internet Routing System (IRS), are sufficient to meet the requirements for a centralized controller.
- **Open Network Capabilities**: This SDN characteristic provides additional benefits. The core concept involves packaging network capabilities into an operating-system-like controller. In the SDN architecture, the upper layer applications and services are used to obtain the network capabilities through APIs from the controller, and the lower layer interfacing between the controller and network elements are both defined by the open network capability standards. The SDN operation and specifications are majorly designed and regulated by ONF (Open Networking Forum). The open APIs and advanced orchestration capabilities create a flexible and modular services platform.

2.2.3 SDN Application Domain

This research work is focused on transport optical networks that interconnect data centers. Henceforth this section briefly presents two application areas in which SDN could prove to be beneficial: data centers and optical networks. SDN is also applied in many other networks such as enterprise networks, WLANs and heterogeneous networks, cellular networks and the Internet of Things.

2.2.3.1 Data Center Networks

In the traditional data center network, the network management is typically met through the careful design and configuration of the underlying network. This operation in most cases is performed manually by defining the preferred routes for traffic and by placing middle boxes at strategic choke points on the physical network. Obviously, this approach contradicts the requirement for scalability, since manual configuration can become a very challenging and error prone task, especially as the size of the network grows. Additionally, it becomes increasingly difficult to make the data center operate at its full capacity, as it cannot dynamically adapt to the application requirements [16].

The advantages that SDN offers in network management is filling these gaps. By decoupling the control from the data plane, the forwarding devices become much simpler and therefore cheaper. At the same time all control logic is delegated to one logically centralized entity. SDN opens the opportunity for the network operators for implementing policies so as to dynamically manage data center interconnections and assign bitrate for flows based on the user [14][27]. It also allows for the dynamic management of flows,

the load balancing of traffic and the allocation of resources in a manner that best suits the operation of the data center based on the needs of running applications, which in turn lead to increased performance [15][16][17].

2.2.3.2 Optical Network

Optical networks are under pressure, the demand for bandwidth continues to grow rapidly with no apparent end in sight. Traffic patterns are non-uniform and drastically shifting. For some years people have discussed the impact of the dramatic increase in video usage, the newer trends of adopting cloud services and the emergence of mega-sized data centers should also be considered as a part of the traffic to be managed. At the same time, device mobility and the Internet of Things altogether has changed where and how bandwidth is being consumed.

High peak-to-average and/or transient bandwidth demands between certain locations requires transport service that can be turned up, modified, and torn down in near real time. The current transport network cannot effectively address these pressure points, as they are generally static and operated separately from the client layers or by the applications they serve. To support such a highly dynamic environment, connectivity services must be turned up in minutes or seconds and be modifiable by the client users or the software applications without operator intervention. An orchestration is required to manage connectivity services over the network covering potentially multiple network domains, through multiple layers of networking technology, and across multiple vendors' equipment [4][16].

In order for new services to be provisioned or changed in an optical network via software without pre-knowledge of the service type, the physical ports that are attached to the network must be software defined both in protocol and speed definition as well as wavelength definition. While SDN started to visualize the physical layer as merely a supporting layer of the logical network, with optical networking there is no such separation. Therefore, it is the physical layer of optical networks that must be software defined in Optical SDN. User definable port speeds, protocols, and wavelengths have been available in optical network hardware in recent years, with the level of flexibility varying from simple LAN/WAN Ethernet selection to fully flexible transponders covering a wide range of rates and protocols [5][6].

The Optical Transport Working Group (OTWG) defined under the umbrella of ONF's, provides the architecture and mechanisms to address the trends and challenges associated with providing flexible and dynamic optical switching. Until recently, the OpenFlow standard focused on the packet-oriented Layers 2 and 3. Transport SDN extends OpenFlow to support Layer 0 (photonic) and Layer 1 (SONET/SDH, OTN) network, allowing the same support for logically centralized control and independent software and hardware development.

Fundamentally, extensions are added to OpenFlow to program switch ports and fabrics that operate on fibers, wavelengths, and time slots as well as the packet headers. In our research, to address the specific requirements for the transport network, OpenFlow is being extended to support protection, performance monitoring, and other critical Operations, Administration, and Maintenance (OAM) capabilities [5][6][22].

2.3 OpenDayLight Controller

The SDN controller acts as the strategic control point in an SDN network, it relays information to the switches/routers “below” via the SBI and to the application and business logic “above” via the NBI.

The SDN controller platform is typically comprised of a collection of pluggable modules that are capable of performing different network tasks. Some of the basic tasks include inventorying of devices in the network, identifying the capabilities of each and gathering their network statistics. Extensions can be added to enhance the functionality and support more advanced features, such as orchestrating new rules throughout the network, virtualization of network functions and performing analytics by running algorithms on the network traffic [8]. There are multiple choices of open source and commercial SDN controllers, and the controller for our work is selected based on our requirements as stated below:

1) Selecting an open source SDN controller.
2) The controller should support NETCONF and OpenFlow as part of the supported SBIs.
3) The controller should provide well-developed NBIs that support in building the business logic that manages the connection between interconnected data centers.
4) A Graphical User Interface (GUI) based interface that makes user interaction with the controller simpler.

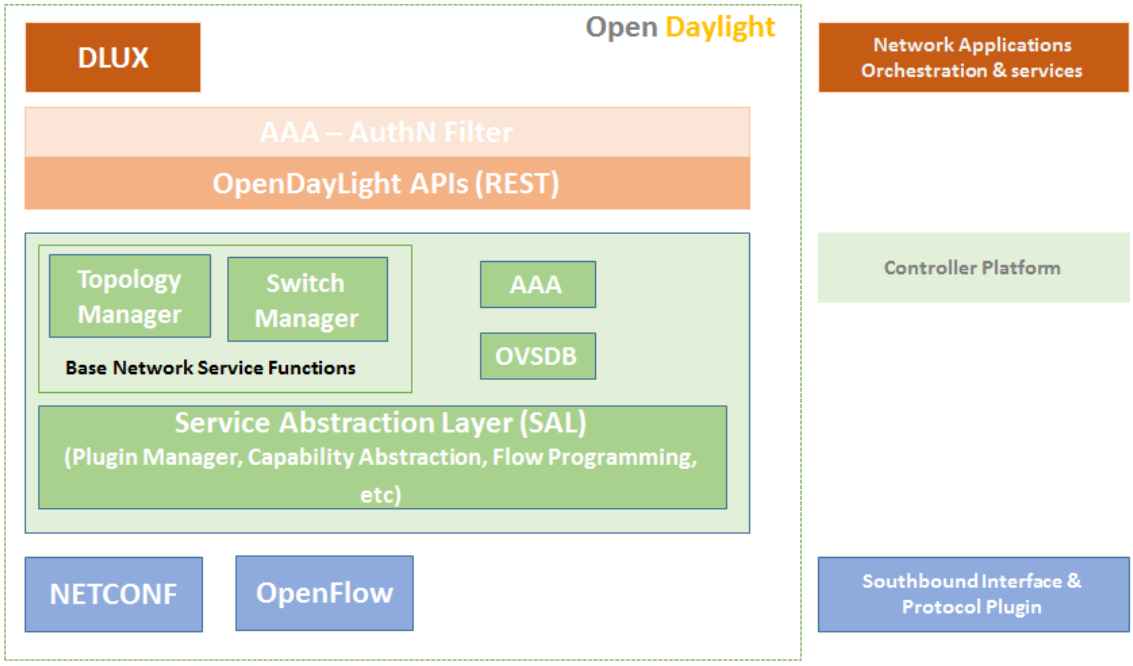


Figure 2.2 OpenDayLight Architecture

The ODL project is an open source platform for SDN. It uses open protocols to provide centralized, programmatic control and network device monitoring. Like many other SDN controllers, ODL supports OpenFlow, as well as offering ready to install network solutions as part of its platform. The ODL architecture is represented in Figure 2.2 and key factors that are related to the research work are discussed below [8]:

- It supports a microservices architecture, in which a "microservice" is a particular protocol or service that a user wants to enable within their installation of the ODL controller. For example: the plugin provides connectivity between the control plane and the data plane devices via protocols implemented as SBIs or offer services such as Authentication, Authorization, and Accounting (AAA) or support switch functionalities such as L2-Switching.

- Various network services and orchestration are supported at the application layer. DLUX is the ODL web interface that provides a modern GUI to interact with the controller for simplified setup and administration.
- REST (REpresentational State Transfer) is the software architectural style of the World Wide Web, which communicates over Hypertext Transfer Protocol (HTTP) with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages and to send data to remote servers. The ODL controller supports REST as its NBI. REST uses Remote Procedure Calls (RPCs) and message notifications to communicate with the control plane of the controller.
- The controller platform and its services support various use cases; multiple plugins are supported as protocols for SBIs. ODL supports the Model Driven – Service Abstraction Layer (MD-SAL) architecture, additional services and plugins could be added or extended accordingly.
- The core Service Abstraction Layer (SAL) is composed of data stores, MD-SAL data stores are further divided into config and operational. The config data store contains the configuration information that allows users to change configurations, for example through a REST API. The second data store contains the operational information that comes from the system and normally offers users to view and understand if their configurations have been successfully pushed into the devices.

2.4 SouthBound Interfaces

The main objective of the SBI is to provide communication and management interface between the network's SDN controller(s) and data plane (nodes, physical/virtual switches

and routers). The SBI facilitates efficient control over the network, and enables the SDN controller to dynamically make changes according to real time demands and needs. The controller breaks down the whole network connections into smaller technical details known as flows, that are specifically geared toward the lower layer component(s) using SBI. Multiple traditional and SDN-specific protocols are supported as SBIs between the decoupled control and data plane.

OpenFlow is a SBI designed to meet the requirements of SDN. In addition, most of the controllers support some of the existing configuration protocol as SBIs that help in establishing the connection between a SDN controller(s) and the network element(s). A few of those protocols are listed below:

- NETCONF/YANG (Yet Another Next Generation)
- SNMP
- File Transfer Protocol (FTP) or SSH File Transfer Protocol (SFTP).

2.4.1 NETCONF and YANG

The NETCONF is an IETF network management protocol. It was developed by the NETCONF working group and published in December 2006 as RFC 4741 and later revised in June 2011 and published as RFC 6241 [11].

NETCONF provides mechanisms to install, manipulate, and delete the configuration of the network devices. Its operations are realized on top of a simple RPC layer. The NETCONF protocol uses an eXtensible Markup Language (XML) based data encoding for the configuring data as well as the protocol messages. The protocol messages are

exchanged on top of a secure transport protocol. NETCONF can be conceptually partitioned into four layers as shown in Figure 2.3 [11].

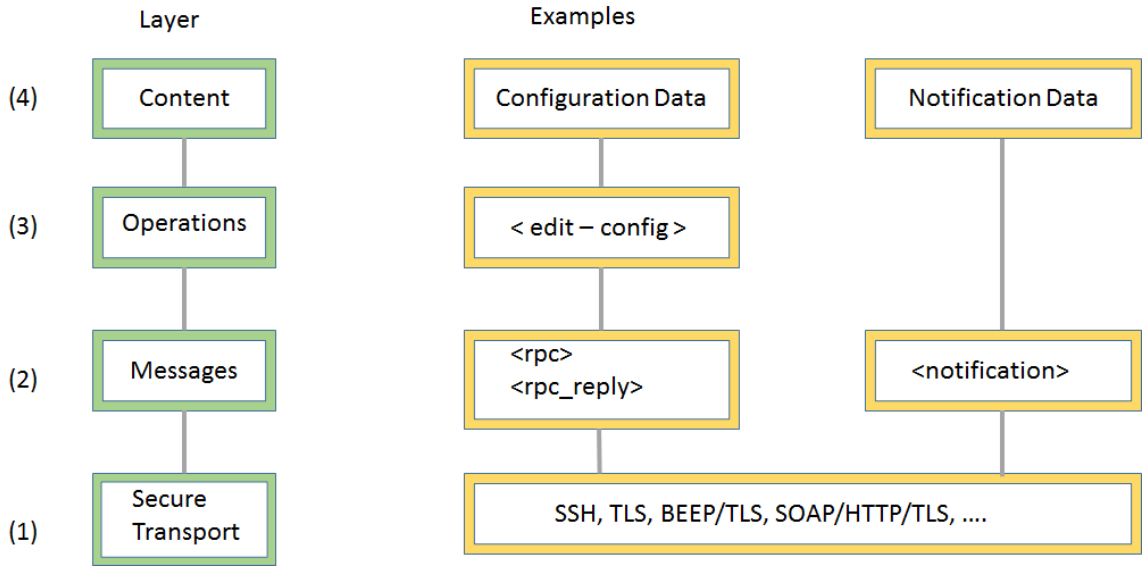


Figure 2.3 NETCONF Layers [11]

1) The **Secure Transport layer** provides a communication path between the client and server. It uses an RPC-based communication paradigm over the SSH transport protocol mapping. NETCONF can be layered over any secured connection-oriented transport protocol that provides security, integrity, and reliable sequenced data delivery. NETCONF supports a long-lived persistent connection between protocol operations among the peers [11].
2) The **Messages layer** provides a simple, transport-independent framing mechanism for encoding RPCs and notifications. A NETCONF peer uses RPC-based <rpc> and <rpc-reply> element tags to provide transport-protocol-independent framing of NETCONF requests and responses [11].

3) The **Operations layer** defines a set of base protocol operations invoked as RPC methods with XML encoded parameters. The NETCONF protocol supports a small set of low level operations to manage device configurations and retrieve device state information. The base protocol provides operations to retrieve, configure, copy, and delete configuration data stores. Additional operations are provided, based on the capabilities advertised by the device [11].
4) The **Content layer** is used to provide information in a "human readable" format. It uses a data modeling language that helps in representing the model configuration and state data manipulated by the NETCONF, NETCONF RPCs, and NETCONF notifications [11].

YANG is the data modeling language for the NETCONF. The YANG data modeling language was developed by the NETMOD working group of the Internet Engineering Task Force (IETF) and was published as RFC 6020 in October 2010. The data modeling language can be used to model the configuration as well as state information of the network element(s). Furthermore, YANG can be used to define the format of the event notifications emitted by the network element and it allows the data model to define the signature of the RPCs that can be invoked on the network element via the NETCONF protocol [12].

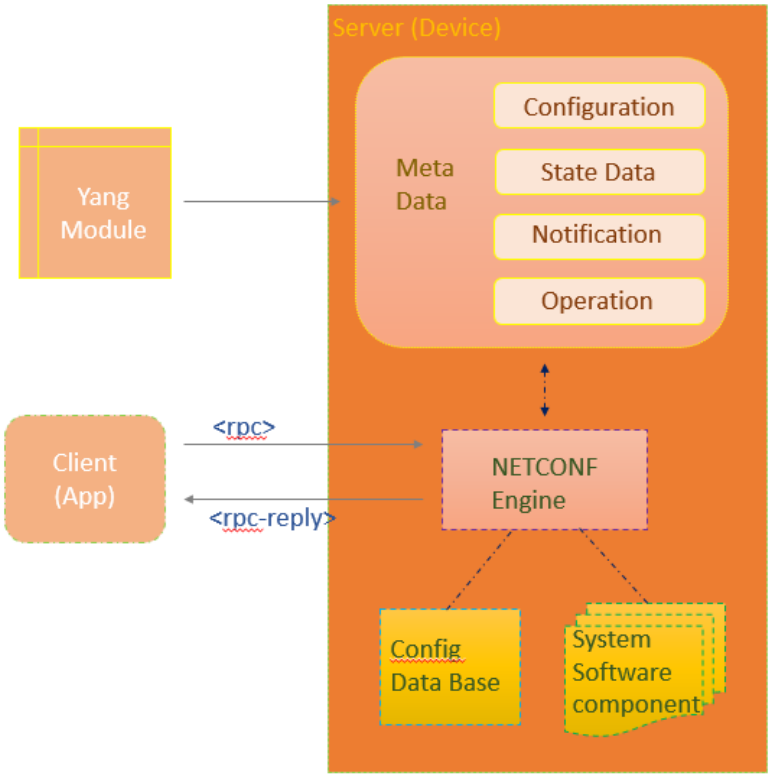


Figure 2.4 NETCONF and YANG Operation

A YANG module defines a hierarchy of data that can be used for NETCONF-based operations, including configuration, state data, RPCs, and notifications. This allows a complete description of all data sent between a NETCONF client and server.

The network devices have a system software component that is responsible for performing network operations. The software component operates based on the information that are stored in a config database. The device supporting NETCONF is composed of a NETCONF engine and a protocol stack that enables a NETCONF client to interact with the device and to operate on their data set, as shown in Figure 2.4. Any NETCONF client can initiate an RPC-based reliable socket communication to interface with the NETCONF engine running on a device. This reliable connection and the NETCONF protocol layer enable the client to operate on the data set. The information is broadly classified into metadata based on the category and purpose. The classified

metadata information is modeled and stored in the form of a YANG data set. The system software is responsible for updating the changes to the config database for enabling the services.

On the one hand, each network device and service offers a wide variety of configurable parameters, and each configurable parameter contains a set of values to be chosen. The selection of proper values for these parameters depends not only on the network device itself, but also on the overall consistency among the network devices residing on the same network. The NETCONF protocol defines operations for managing network devices where configuration data can be retrieved, uploaded, manipulated, and deleted. The standard also defines the API as well as the connectivity requirements for NETCONF. It is not a new technology, as work started on this approximately 10 years ago, but what it gives us is an extensible and robust mechanism for managing network devices. Many commercial and open source controller support NETCONF as one SBI to interact and control the devices. The controller acts as a NETCONF agent (NETCONF client) that establishes an RPC communication with the underlying NETCONF device.

2.4.2 OpenFlow

OpenFlow is considered one of the first SDN standards maintained by the Open Networking Forum (ONF) [3]. An OpenFlow Switch, as shown in Figure 2.5, acts as an abstract packet processing machine and is composed of one or more flow tables and a group table, and the OpenFlow switch performs packet lookups and forwarding based on this table information. The switch processes the packets using a combination of the packet contents and switch configuration state. Through the OpenFlow protocol, it is

capable to manipulate the switch's configuration state as well as receiving certain switch events. The SDN controller is an element that speaks the OpenFlow messages to manage the configuration state of the switches, responds to events and updates the flow table.

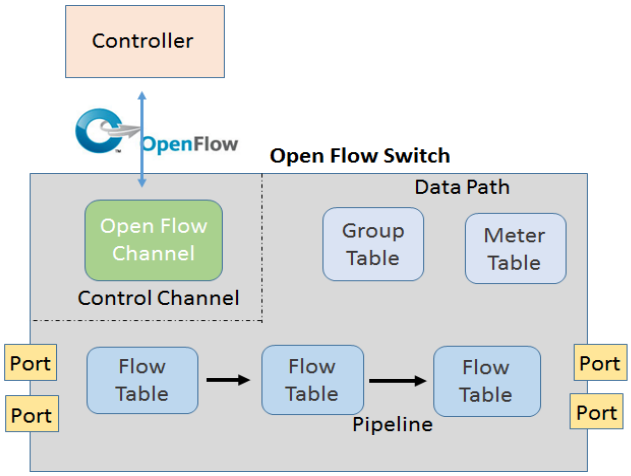


Figure 2.5 OpenFlow Switch [3]

OpenFlow is a new method for controlling flows in the network. Networking has always focused on managing frames and packets with routing protocols, but applications do not use single packets to deliver services.

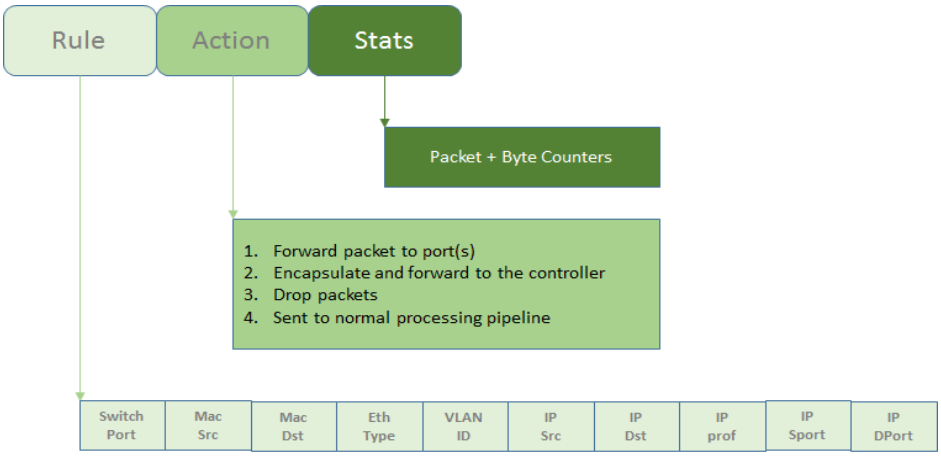


Figure 2.6 Flow Rules [3]

Rather, they exchange data between a server and a client, by creating a stream of packets from a source to a destination that is commonly known as a flow. OpenFlow defines a

standard for sending flow rules to network devices so that the control plane can add them to the flow table of the data plane entities.

The flow rules, as shown in Figure 2.6, contain fields for elements such as source and destination MAC address, source and destination IP address, source and destination port number, VLAN tag, QoS (Quality of Service) and MPLS tags and more. The flow table is what all routers and switches use to dispatch frames or packets to their egress ports. Each flow has a priority in matching precedence and timeouts or idle time based on which the flow is expired by the switch. Counters are maintained for each flow entry that help in obtaining packet statistics [3].

2.4.2.1 OpenFlow Flow Handling and Flow Matching

Using the OpenFlow protocol, the controller(s) can add, update, and delete flow entries in flow tables, both reactively and proactively. The action is carried out on all packets received at the switch, based on the information stored in the flow tables. On receipt of a packet, the OpenFlow switch starts to perform a table lookup in the first flow table and, based on pipeline processing, may perform table lookups in other flow tables. A device performs the OpenFlow functions as shown in Figure 2.7, and discussed in detail below [3].

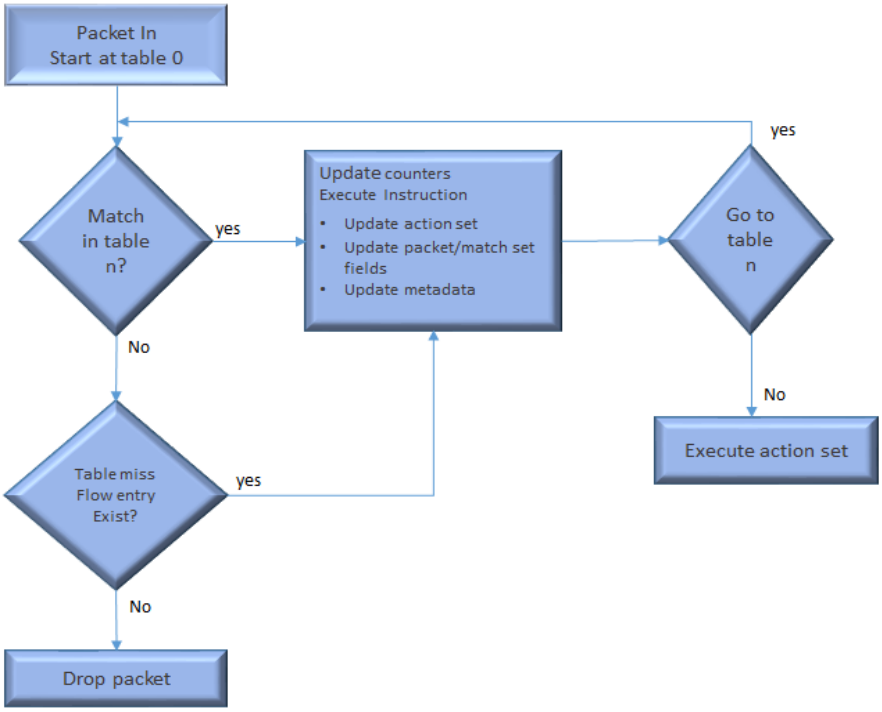


Figure 2.7 Flow Matching [3]

The process involved in matching the received packets and the flow tables are described below.

- The matching starts at the first flow table and may continue to additional flow tables as represented in Figure 2.8. The flow tables of an OpenFlow switch are sequentially numbered, starting from 0 to n. Pipeline processing always starts at the first flow table: the packet is first matched against flow entries of flow table 0. Other flow tables may be used depending on the outcome of the match in the first table.
- When a packet is matched with a flow entry, the instruction set included in that flow entry is executed. Those instructions can direct the packet to another flow table with the help of a goto statement and the same process of flow search can be repeated again. The flow entry can direct a packet to a flow table number greater

than its own flow table, in other words the pipeline processing can only go forward and not backwards. If the matching flow entry does not direct packets to another flow table, pipeline processing stops at this table and when pipeline processing stops, the packet is processed with its associated action executed. If no match is found in a flow table, the outcome depends on the configuration of the table miss flow entry: for example, the packet may be forwarded to the controller over the OpenFlow channel or dropped.

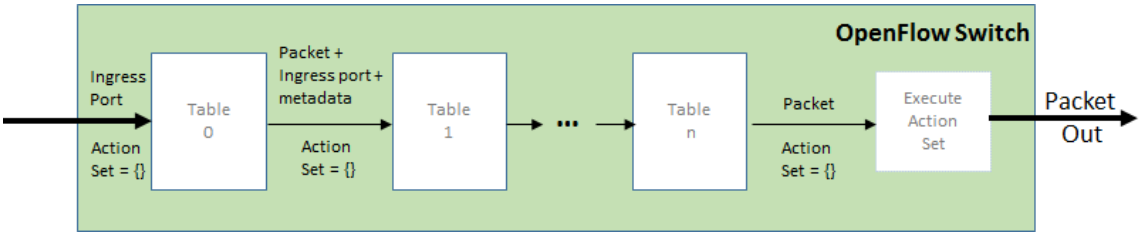


Figure 2.8 Packet Matching Across Multiple Flow Tables [3]

2.4.2.2 OpenFlow Messages

The OpenFlow protocol creates a channel that connects each OpenFlow switch with a controller. This channel helps the controller to configure and manage the switch, to receive events from the switch, and to send packets out of the switch [3]. The OpenFlow protocol supports three types of messages.

- **Controller-to-Switch Messages** are initiated by the controller and used to directly manage or inspect the state of the switch, they may or may not require a response from the switch [3].

- **Asynchronous Messages** are sent without a controller soliciting them from a switch. Switches send asynchronous messages to controllers to denote a packet arrival, switch state change, or error [3].
- **Symmetric Messages** are sent without solicitation, in either direction, such as hello and echo message [3].

2.5 Related work

This section provides a survey of literature works related to "Managing BoD Across Interconnected Data Center Using SDN" and "SDN in the Field of Optical Network". These related works collective helped us to design an SDN optical network and further manage BoD request across interconnected optical data centers using SDN protocols.

2.5.1 Managing Bandwidth on Demand Across Interconnected Data Center using SDN

BoD has been studied for many years by both commercial carriers and members of the global research and education community [14]. Since its appearance, SDN has been considered as one of the most promising enablers for traffic engineering. The controller can be made aware of the entire network state and is also capable of making powerful advanced traffic engineering strategies. Many Internet service providers and network operators have started to deploy solutions based on SDN technologies, aiming to improve their network utilization or the QoS provided to their users [20].

a) Towards a Carrier SDN: An Example for Elastic Inter Data Center Connectivity

In [27] the authors proposed a network driven transfer mode for cloud operations in a carrier SDN environment. The carrier SDN was deployed between the interconnected data center middleware, managed by the ABNO (Application Based Network Operations) controller and an SDN controller. The SDN controller is in charge of managing inter data center connectivity. The ABNO is responsible for PCE (Path Computation Element) and policy enforcement. For a traffic operation, the cloud middleware requests for data transfers using its native semantic: amount of data to be transferred, data center destination and completion time. Notifications (similar to interruptions in computers) are sent from the ABNO to the SDN controller, if fewer than the required resources are available during the time of request. The SDN controller releases specific resources each time. Upon receiving a notification, the SDN controller takes a decision whether to increase the bitrate associated with a transfer. The source cloud manager sends a transfer request to the SDN controller in the specified native semantic format. Upon its reception, the SDN controller requests the ABNO controller to find the greatest spectrum width available, taking into account local policies and current Service Level Agreements (SLA) and sends the response back to the cloud manager with the best completion time. The controller interactively communicates with the cloud middleware and allocates bandwidth based on the resource availability and the bandwidth required to complete the services. The proposed network-driven model opens up the opportunity for network operators to implement policies to dynamically manage connections, the bitrate of a set of customers and to fulfill simultaneously their SLAs.

- The authors proposed an elastic inter data center connectivity for managing carrier SDN at layer 2 and layer 3 with the help of the label switched path.
- In [27] the authors have used a native semantic for communication between cloud manager and SDN. Use of standard protocols supports interoperability and ensures common open network capabilities, as they are the key aspect of SDN.

b) Using SDN Technology to Enable Cost Effective Bandwidth on Demand for Cloud Services

In [20] the authors described the BoD in an evolved multilayer, SDN-based cloud services model for the WAN. The data centers comprise of switches that seamlessly route traffic through the multilayer network architecture. The authors deployed an SDN Wide Area Network Control and Management (SWAN-C&M) orchestrator. They also created a common API to manage the data center through SWAN-C&M, which helps in creating a dynamic cloud service environment. The SWAN-C&M software uses a modular architecture to make routing decisions, and it accesses the database containing the network topology. To configure the IP and subwavelength transport networks, the SWAN-C&M controller has modules that translate configuration into instructions for the appropriate interface in the equipment. The IP layer devices are being controlled using OpenFlow and a proprietary interface (JunOS), while the optical transport layer devices are controlled using CLI. The user interacts with SWAM-C&M using a REST web interface for the management of BoD connections. With the help of SWAN-C&M, the authors demonstrate load balancing and explicit routing and rerouting use cases that showcases the impact of SDN over the multilayer architecture.

- The authors classified the flows based on the rates requested and load-balances the traffic by allocating higher traffic rates at the optical layer and lower rates at the IP layer.
- The ability to manage the connections across multiple network layers with the help of common SDN-based APIs provides inter data center services at lower cost and good performance [20].

c) Multi Domain Bandwidth on Demand Service Provisioning using SDN

The proposed solution in [14] relies on a framework called DynPaC (Dynamic Path Computation), which can provide resilient L2 services taking into account the bandwidth and VLAN utilization constraints. The DynPaC framework has been implemented as an application running on the ONOS controller, which provides a basic set of services and features, such as device, link or host discovery. Once it obtains the topology information of the domain, DynPaC computes the best possible intra domain path taking into account the requested bandwidth, the VLAN availability and the already reserved services. In this regard, DynPaC differentiates Gold and Regular services, depending on whether they have a backup path guaranteed or not. Finally, in order to demonstrate the resilience capabilities of the DynPaC framework, they tear down one of the links used by both services. The framework forces the installation of the backup path for the Gold service. As a result, traffic exchanged using the Gold service continues without any disruption, whereas the traffic exchanged using the Regular service will be stopped.

- The authors demonstrate BoD service provisioning in layer 2 devices using OpenFlow as communication protocol to manage the layer 2 devices.

- Both data centers are connected using the optical network interface at layer 0 and layer 1, managed using a proprietary control interface. The optical layer device does not exhibit dynamicity.

d) SDN Based Multi-Class QoS-Guaranteed Inter Data Center Traffic Management

In [21] the authors presented a utility-optimization-based joint bandwidth allocation for IP-based inter data center communication with multiple traffic classes that handles priorities between traffic classes and explicit consideration of the delay requirement that meets their end to end communication. In the test environment all the data centers are equipped with OpenFlow enabled switches that report the network events and traffic statistics to the central traffic management server. The bandwidth broker estimates the bandwidth demands of the application and reports the information to the traffic management server, which further allocates the bandwidth based on the classes and availability. This centralized approach provides flexibility to enable various traffic engineering goals. The authors state that the proposed algorithm advocates joint bandwidth allocation of multi class traffic, leading to a higher network bandwidth utilization.

- The authors of [21] demonstrate that handling traffic based on multiple classes and priorities leads to higher bandwidth utilization.
- The authors demonstrate BoD service provisioning and management of layer 2 devices using OpenFlow as communication protocol. The same can be implemented on the optical backbone interconnecting data centers.

e) Bursting Data between Data Centers: Case for Transport SDN

Data center WAN interconnects today are pre-allocated, static optical trunks of high capacity. These optical pipes carry aggregated packet traffic originating from within the data centers while routing decisions are made by devices at the data center edges. The authors of [25] propose an SDN-enabled optical transport architecture that meshes seamlessly with the deployment of SDN within the data centers. The proposed programmable architecture abstracts a core transport node into a programmable virtual switch that leverages the OpenFlow protocol for control. This not only allows multilayer, multidomain, multivendor orchestration, but also allows cleaner, programmable network abstractions. The network can be viewed as a pool of intelligent bandwidth, with resource reservation and path selection done end-to-end in a technology-agnostic fashion. The authors proposed two modes of operation:

a) Implicit Mode: In this mode of operation, the SDN controller has a view of only the edge nodes in every transport domain.
b) Explicit Mode: Here, the complete topology of every network element across domains is exposed to the controller.

- The experiments demonstrate the use case of an OpenFlow-enabled optical virtual switch managing a small OTN for a big data application.
- With appropriate extensions to OpenFlow, they discuss how SDN brings the programmability and flexibility to packet optical data center interconnects which can be substantial in solving some of the complex multivendor, multilayer, multidomain issues that hybrid clouds raise.

The review of related works with respect to BoD in an SDN environment focuses mainly on introducing flexibility and manageability at higher layers in the network. When we closely look at the proposed solutions, the optical network infrastructures are either treated as fixed physical links [14][27] or have very minimal instruction for adding a flow [20] by using a non-SDN protocol. The authors in [14][20][21][27] talk about how SDN helped them to manage data center interconnections at the IP layer and their benefits. The optical network entities (layer 0 and layer 1) can also be managed by SDN [25]. In all the literatures reviewed above the bandwidth requests are handled based on any one of the classification factors such as priorities, class of services or rates.

2.5.2 SDN in the Field of Optical Network

SDN as a newer technology has progressed its advancement in optical networks and still faces some trends and challenges in defining its potential in the optical networks. In this section we review some of the contributions related to SDN optical networks that show us the potential, and help us to design our system.

a) DWDM Optical Extension to the Transport SDN Controller and Towards Widespread SDN Adoption: Need for Synergy between Photonics and SDN Within the Data Center

Transport networks are evolving to be more and more automated and driven by software to minimize the operational costs and to provide new services and applications in a quicker and more efficient way. In [19] the author claims that, if one wants to extend the

SDN approach to the physical photonic layer, then the SDN controller must take the analogue nature of the optical transmission into account. In the transport networks the physical impairments of light paths in optical networks can limit the connections that are feasible using SDN. The problem in evaluating the feasibility of an optical connection becomes particularly difficult in the case of mixed channel types and rates and where the optical links and photonics with different properties have to be considered.

In [19][22] the authors propose a procedure to suggest the protocol extensions that can capture, process and set power levels in the optical networks. This procedure mainly appropriates in dynamic scenarios, in which the opportunity to adjust the channel power levels on the fly could allow for the introduction of new optical traffic which would not otherwise be feasible, all the while ensuring that the existing connections remains unaffected.

The authors in [19][22] look at the use of photonics in future data center networking and stresses on the need for SDN Controller and its applicability. Introducing programmability, virtualization, end to end optimization, able to control, manage and automate these elements offers an elastic and agile data center.

- The authors in [19][22] talk about the need and advantages of SDN in the transport network, and also state the challenges associated with an implementation of the SDN transport network considering the nature of the optical backbone.
- The authors also propose solutions that can be implemented to address the future of the transport optical network along the lines of the SDN technology.

b) NETCONF as Proactive OAM Protocol

Francesco Paolucci et al [23] talk about building a software defined elastic optical network that offer a high degree of flexibility in enabling dynamic configurable light path provisioning and re-optimization. In order to guarantee Quality of Transmission (QoT), novel Operation Administration and Maintenance (OAM) solutions are necessary with respect to existing standard management protocols. The authors proposed the NETCONF protocol, typically used for SDN-based node configuration purposes. NETCONF serves as an OAM protocol, in order to achieve a high degree of convergence and limit the number of utilized protocols.

- The authors have implemented NETCONF as an OAM protocol to deal with optical power level, alarms and statistics.
- The literature does not provide any experiments to demonstrate NETCONF as SBI protocol for managing optical network. The authors' demonstration exhibits the potential of NETCONF to act as an SBI to manage the optical network.

c) An Optical SDN Controller for Transport Network Virtualization and Autonomic Operation

Marcos Siqueira et al [26] proposed a Software Defined Optical Transport Network (SD-OTN) architecture. The system architecture is comprised of Optical Network Elements (O-NEs) and an Optical SDN Controller (O-SDNC). The O-SDNC includes a network abstraction layer allowing the implementation of cognitive controls and policies for autonomous operation, based on the global network view. Additionally, the controller

implements a virtualized Generalized Multiprotocol Label Switching (GMPLS) control plane, offloading and simplifying the network elements, while unlocking the implementation of new services such as optical VPNs, optical network slicing, and keeping the standard Optical Internetworking Forum (OIF) interfaces. The concepts have been implemented and validated in a real testbed network formed by five DWDM nodes equipped with flexi grid wavelength selective switching ROADMs (Reconfigurable Optical Add Drop Multiplexers).

The SDN controller provides a NETCONF/REST interface for plugging in control applications to perform specific tasks by taking advantage of the network abstraction. By modeling these network entities using the NETCONF modeling language YANG, the authors state that they gain the possibility to export or transform the data needed for their configuration, as well as to model their interconnections and restrictions.

- The proposed optical SDN architecture in [26] is managed using NETCONF as an interfacing protocol to manage the ROADM functionality of the O-NEs. The same approach can be used to manage the optical ports and multiplexing.
- The experiments showcase the optical network being managed using NETCONF as a controlling interface, the same could be used to manage the bandwidth and connections across the data centers and manage each data center separately.

d) Data Center Optical Networks (DCON) with OpenFlow based Software Defined Networking

The authors of [28] propose the flexi grid optical network as the solution for managing inter data center networks. The environment is composed of an enhanced Software

Defined Networking (eSDN) control architecture designed for the application scenario. The eSDN architecture over flexi grid optical networks is composed of the distributed data center networks that are interconnected with dynamic, tunable and efficient spectral optical resources, which are deployed with the application (e.g., CPU and memory) and network stratum resources respectively. Each stratum resource is software defined with OpenFlow and controlled by the Application Controller (AC) and the Transport Controller (TC) respectively in a unified manner. To control the heterogeneous networks for data center service migration with an extended OpenFlow Protocol, OpenFlow enabled flexi grid optical device nodes with OpenFlow Protocol agent software are implemented.

- The proposed solution has an AC that manages the business application logic and the TC to manage the optical connection between the interconnected optical nodes.
- The authors have implemented an extended OpenFlow Protocol to support optical switching of spectrum randomly from 50 GHz to 400 GHz based on the traffic request.

To summarize the preceding reviews, due to the demand, the networks have to scale out with high capacities, ensure end-to-end guarantees and an ability to be agile and elastic. The authors in [22] claim that the future data center networks will be built to be more dynamic in nature. As a part of this evolution the transport networks are evolving to be more and more automated and driven by software to minimize the operational costs and to provide new services and applications in a quicker and more efficient way [19]. The authors in [23][26] propose NETCONF as an interface for managing the optical network.

On the other hand, ONF is working on OpenFlow extensions for optical networks. The authors in [28] have demonstrated optical spectrum switching using a proprietary extended OpenFlow interface. Few references use traditional NETCONF protocol for managing the optical network and a few other references use an extended OpenFlow environment to manage the optical network in an SDN environment.

2.6 Motivation

To conclude from our review of background work, SDN plays a significant role in shaping today's network infrastructure. The key attributes for migrating towards SDN are programmability, openness, heterogeneity and maintainability. Moreover, SDN also facilitates the re-architecture required to address the increasing demand on the network due to the dynamic connectivity [28]. From the background study, we do observe that traditional optical networks are usually more static in nature and each device is typically managed individually. At the same time, it becomes unfeasible to manage large networks manually as this results in an error-prone process. The SD-OTN is being in the spotlight in recent times, as the early start of SDN mainly focused on IP-based networks [14] [21]. As more network operators adopt open SDN, there are multiple contributions and proposals that facilitate the migration of existing networks and services to SDN. The SBIs play a vital role in managing the data plane devices in an SDN environment. The ONF suggests OpenFlow as SBI to manage the network devices in an SDN environment. From the literature review, we observed that the OpenFlow protocol can be implemented over an optical network by defining optical extensions [28] for supporting the optical data plane devices. Our review of related works also showcases the potential of NETCONF as

an active protocol for managing optical data plane devices [24][27]. We do realize that, as a result of technological advancement in the area of cloud computing, big data, virtualization, etc., the data centers grew bigger in size and numbers. SDN has the potential to address the problem of BoD requirements across data centers as it offers centralized network control architecture and unified protocols for managing the network entities. The review of related works showcases that optical data centers interconnection are treated as static pipes and on the other hand, the IP layer devices demonstrate the potential of SDN in managing the BoD. There exist also efforts to introducing SDN into the optical network(s) and to manage them. In our research works we demonstrate the possibility to manage BoD between the data centers (i.e. optical inter data center communications) using an open source SDN controller with two different protocol implementations supported as SBI. They are: the ONF recommended OpenFlow and the other is the traditional NETCONF protocol supported as SBIs in the SDN environment. The literature on SDN optical network predominately talking about any one of the both protocols as SBI based on their implementation. There is no literature that compares both these protocols. The motivation of the thesis is to compare and provide a quantitative and qualitative analysis of both these protocols operating in an optical SDN environment managing BoD across interconnected data centers.

Chapter 3: Managing the Interconnected Optical Data Centers Using Software Defined Networking

This chapter begins by stating the requirements and introduces the BTI7800 optical NE. It presents the implementation of NETCONF and OpenFlow as SBIs to manage the BTI7800 network element using the ODL controller. This chapter describes the working of both SBIs.

3.1 Requirement Analysis

First of all, it is important to understand the requirements of the research work. Chapter 2 concludes with motivation towards SDN in managing optical data center interconnection. The BTI7800 is a layer zero and layer one optical device that offers data center interconnection as a service. The BTI7800 is a traditional network device that is managed using traditional management protocols.

In an SDN environment, the BTI7800 is treated as a data plane device. The major tasks involved are listed below:

1) The BTI7800 supports XML-based NETCONF as one of the network management protocols in the traditional environment. We need to support NETCONF as a SBI to manage the BTI7800 using the ODL controller. In order for the NETCONF connector within the ODL controller to communicate with the BTI7800 device, the ODL NETCONF connector is provided with the details about the BTI7800 device that needs to be managed.
2) The central component of SDN is the protocol being developed by the Open Networking Forum (ONF), called OpenFlow, which communicates flow

information from a centralized controller to the abstracted data plane. One of the tasks is to develop an optical extension to the OpenFlow specification that allows control of BTI7800 using an OpenFlow controller.

3) To develop an application layer that communicates the BoD requirements to the control plane. Using the implemented SBIs (NETCONF or OpenFlow), the control plane manages the BTI7800 data plane devices that act as an edge node for each data center that are interconnected. The application layer helps the control plane to make decisions about managing the bandwidth allocations based on the nature of traffic across different users located at both data centers.

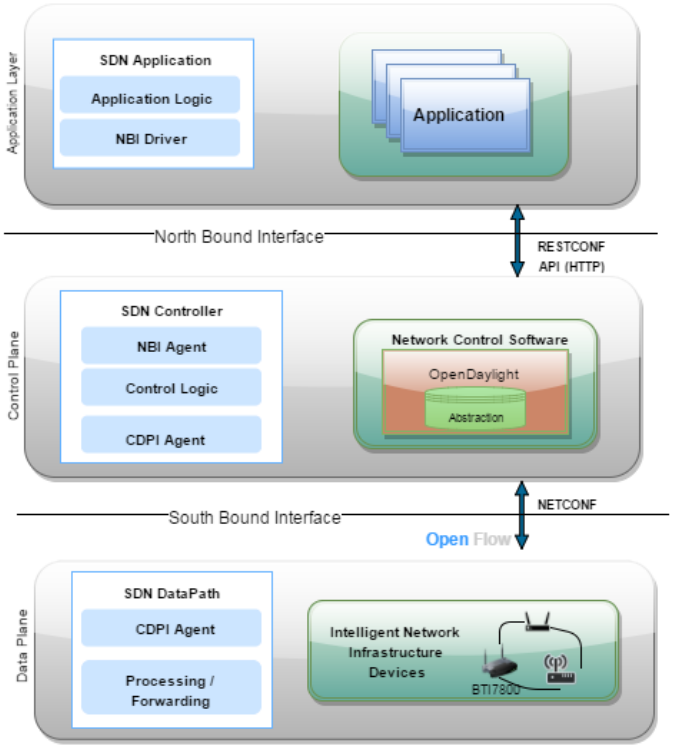


Figure 3.1 Architecture Overview

To achieve the main goal of the thesis is to evaluate both protocols implemented as SBI, it is essential to support SDN capability in the BTI7800 device by supporting NETCONF

and OpenFlow as SBIs as shown in Figure 3.1. Therefore, the optical layer interconnecting data centers can be managed effectively using SDN controller.

3.2 BTI7800 Network Element

Our research work is mainly focused on the BTI7800 network elements that interconnects data centers. Therefore, we present the high level architecture of the equipment. The BTI7800 architecture is majorly composed of three components as shown in Figure 3.2. The Chassis Management Module (CMM) is a software control module, UFM is the host of optics and optical ports. HA is a fabricated silicon hardware component.

- CMM – It is a Linux-based CMM of the device. It is composed of the database, management services and other required services. The cdb is a configuration database that stores information about the device. The confd acts as a management module. A tailored hardware-specific Linux is supported.
- UFM - At the heart of the BTI7800 are the Universal Forwarding Modules (UFMs) hosting optics, configurable for muxponder and transponder applications. The BICs is known as BTI Interface Cards that holds the SFP + and CFP. The module supports
    - 12 ports of 10 GE for intra data center connection (SFP +).
    - 1 port 100 GE for inter data center connection (CFP).
- HA – The hardware module comprises of three components as shown in Figure 3.2. The framer performs synchronization, frame overhead processing, interleaving, parity bit monitoring, etc. The packet forward engine is responsible

for routing the packets received based on the provisioned cross-connects. The clos fabric is a silicon chip that manages the hardware interfaces and the tunnels.

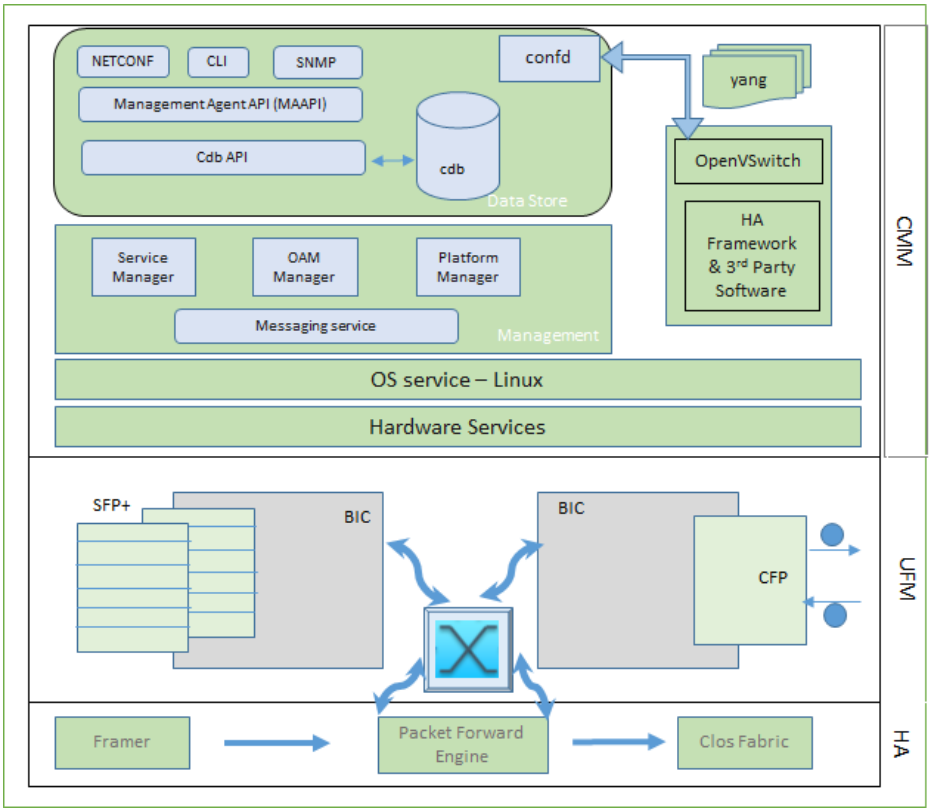


Figure 3.2 BTI7800 Architecture

3.3 Implementation of NETCONF as SouthBound Interface for Controlling BTI7800

We discussed the NETCONF protocol earlier in Chapter 2. An ODL SDN Controller operates both as NETCONF server and as NETCONF client. As a server, the controller manages general network communication and processes RPCs. As a client, the controller connects to the NETCONF-enabled device(s) and manages them through the NETCONF connector(s). We need to connect the NETCONF agents that are residing within the ODL controller and BTI7800 device.

3.3.1 Connecting BTI7800 Device with ODL Using NETCONF Connector

There are multiple ways in which the ODL controller and data plane devices can be connected. It depends on the behavior of the data plane device: whether and how it advertises its YANG capabilities as expected by the ODL controller. The BTI7800 device falls under the category of “NETCONF device does not support IETF-NETCONF-monitoring and it does not list its YANG models as capabilities in hello message “[8]. The BTI7800 device has a vast set of YANG modules ranging from 70 to 100 files based on the software load and features supported by the equipment. Each file is a YANG dataset and it either represents a module or sub-module YANG description of the device and is termed as the YANG capability.

NETCONF as a standard helps to configure the optical ports, interfaces, cross-connects etc. But the YANG data model helps in storing information regarding the BTI7800 optical ports, interfaces, cross-connects and other equipment details. Using the NETCONF protocol it is able to modify those configurations. The existing BTI7800 YANG data model along with the NETCONF as a configuration protocol supports the required optical extension and capabilities. For the NETCONF agent within the ODL to successfully connect with the data plane device, the BTI7800 YANG files have to be placed within the ODL as mentioned in the category “NETCONF device does not support IETF-NETCONF-monitoring and it does not list its YANG models as capabilities in the hello message “[8]. All the YANG files supported by the BTI7800 have to be imported within the ODL controller without any import error. The NETCONF connector uses those YANG files for identifying and interacting with the devices.

3.3.2 Establishing NETCONF Connection between NETCONF Connector and BTI7800

The ODL NETCONF connector establishes a NETCONF connection over SSH for exchanging complete capabilities. The first step is to establish a secure connection and then to exchange capabilities. For the NETCONF connector residing within the ODL to establish a connection with the data plane device, it is essential to educate the connector with the required information about the device. By means of the REST API the NETCONF connector is provided with the necessary information about the device such as name, IP address, port number, username and password, etc. These information helps the ODL NETCONF connector to identify the device in the network and further proceed with establishing the secure connection as described below and shown in Figure 3.3.

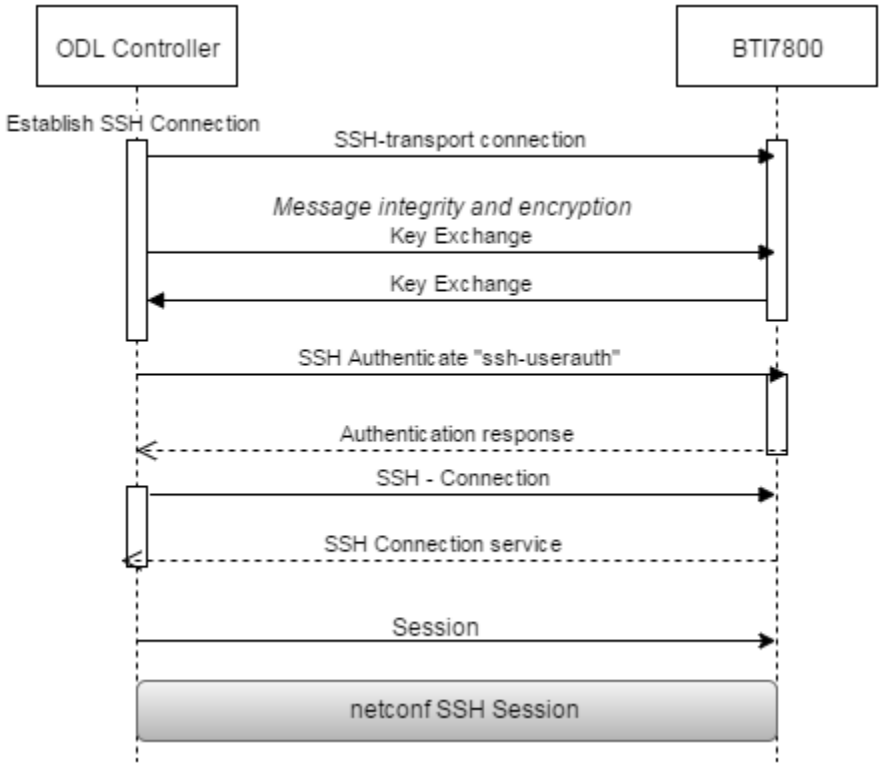


Figure 3.3 NETCONF Session Establishment

1) The ODL NETCONF connector establishes an SSH transport connection using the SSH transport protocol to the BTI7800. Both entities will exchange keys for message integrity and encryption.
2) The NETCONF connector will then invoke the ssh-userauth service to authenticate the connection. Once the connection has been successfully authenticated, the NETCONF connector will invoke the ssh-connection service, also known as the SSH connection protocol.
3) The username provided in the SSH implementation will be used by the NETCONF connector to identify the devices. If the username is not representable in XML, the SSH session will be dropped.
4) After the ssh-connection service is established by the BTI7800, the NETCONF connector will open a channel of type session, which will result in an SSH session. Once the SSH session has been established, the NETCONF connector will invoke BTI7800 as an SSH subsystem called "netconf".

3.3.3 Capabilities Exchange

Once the secure connection is established, the next step is to exchange the capabilities as shown in Figure 3.4. The NETCONF connector will send an XML request containing <rpc> elements to the BTI7800, and the device will respond with the XML response containing <rpc-reply> elements [11][12]:

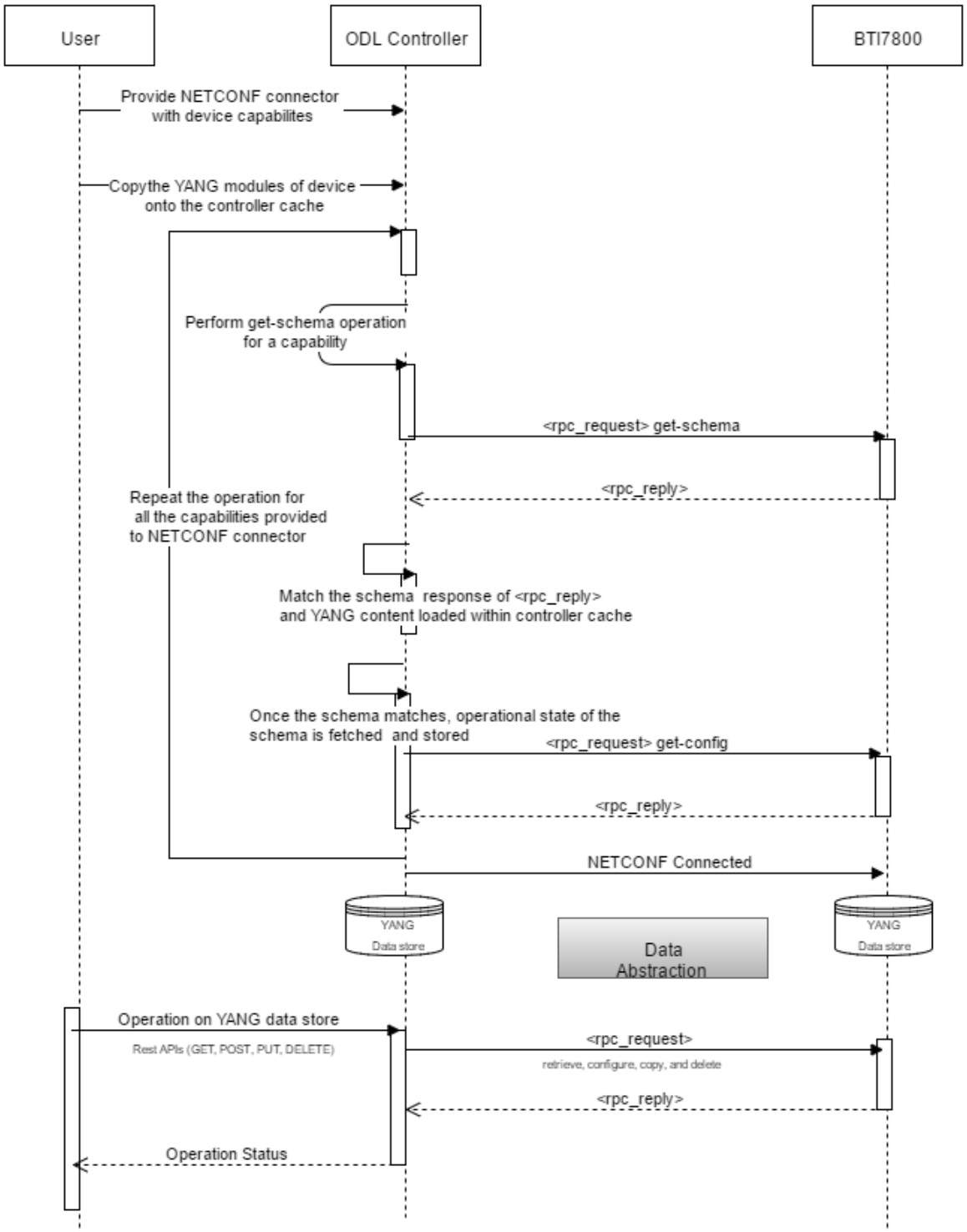


Figure 3.4 NETCONF Device Capability Exchange

1) It is essential for the ODL NETCONF connector to be aware of the capabilities supported by the NETCONF device. The ODL NETCONF connector is not aware of the list of YANG capabilities supported by the BTI7800 device, as they do not advertise their capabilities. The user needs to provide the NETCONF connector with the list of supported YANG capabilities using the REST API.
2) Now the connector is aware of the capabilities supported by the NETCONF device. It is also essential for the NETCONF connector to be aware of the YANG dataset or in other words the content of each capability supported. The user needs to store the contents of all the YANG files in the ODL cache, as the information is not advertised by the device.
3) For each YANG capability listed in the payload, the NETCONF connector performs the <rpc> get-schema operation on the BTI7800 device to obtain the YANG schema content.
4) Further, it validates the content of each YANG file stored in the ODL cache to match the content of <rpc_reply> from the device.
5) If the schema matches, the controller fetches the current operational state of the device corresponding to that schema. The controller retrieves and store the operational state of the information using NETCONF <rpc> get-config element.
6) When all the capabilities and their corresponding YANG contents are matched, the device will be in connected state with the ODL NETCONF connector.
7) The process described above helps in validating the YANG capabilities and its contents provided by the user (the list of capabilities and their corresponding YANG content). This process also helps the controller in abstracting the

information about the device (device data model and operational state) within the controller.

8) The device cannot successfully connect to the NETCONF connector if any of the following issues are faced:
    a. Any of the listed YANG capability provided as input to the NETCONF connector is not supported by the device.
    b. A capability is supported, but the contents of the YANG file within the ODL cache mismatches from the fetched information from the device using <rpc> get-schema.
9) It is possible to view the YANG data set and manage the connected device by performing modifications on the operational state of the device by issuing the REST API request to the controller.
10) The flows are termed as cross-connects in the YANG dataset. The controller has verified all YANG models and has abstracted the operational state of the device. The operational state of the controller stores the provisioned cross-connects within the device. The current state of the cross-connect can be modified by using the REST API calls to access the operational state of the device, and post modifications to the operational state. The BTI7800 device receives the modification in terms of NETCONF RPC <edit-config> using the NETCONF communication subsystem established in Section 3.3.2. If the device can successfully commit the received flow modification to the cdb, it returns success, else it returns failure to the controller.

3.3.4 Managing the BTI7800 using NETCONF

Once the SDN controller is aware about the data model and the operational state of a device in the network, the next step is to allow the controller to manage that device. The ODL controller stores the operational state information about the device in two different states, operational and config.

- Operational -> State in which the underlying devices are operating.
- Config -> Allows user to modify and work on the abstracted information. Once the changes are successfully committed to the respective device, the operational state is automatically updated.

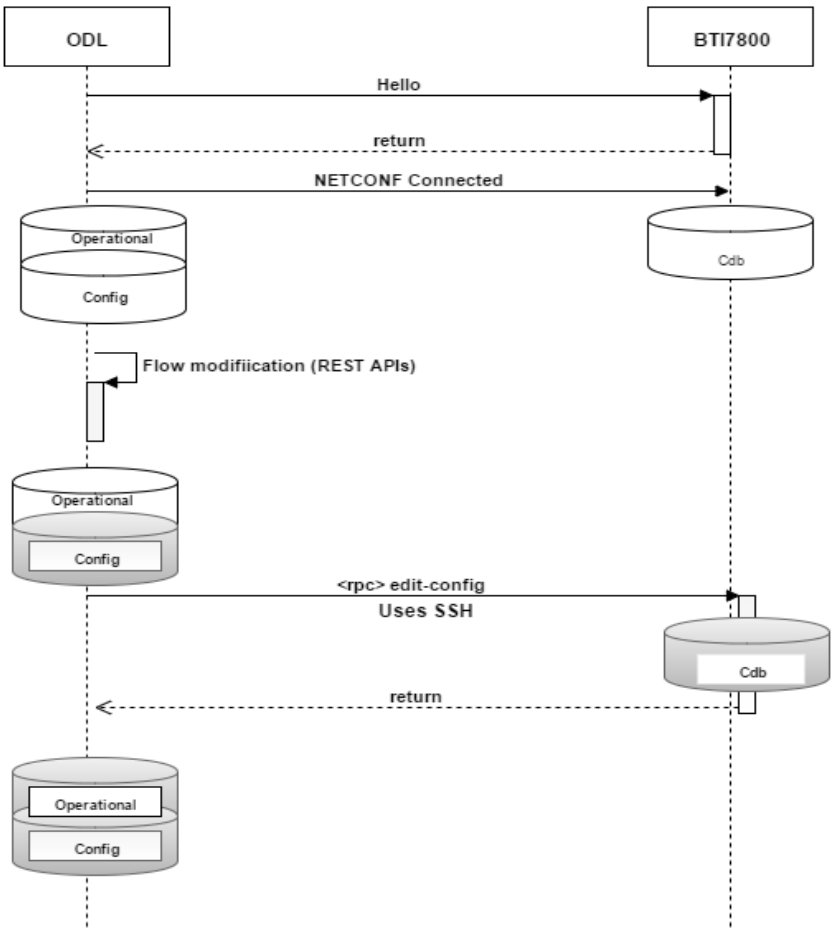


Figure 3.5 Managing the Device using NETCONF

The detailed description about how the NETCONF protocol as a SBI can be used to manage the device is shown in Figure 3.5 and described below.

1) We have discussed how the BTI7800 device can be connected with the ODL NETCONF connector in Section 3.3.2. The cdb is a database that stores the operational state or configuration of the BTI7800 device as discussed in Section 3.3.3.
2) The information stored in operational and config state within the controller and cdb within the device is the same.
3) When a flow modification is performed with the help of the REST API, the modification is stored in the config state of the controller.

4) Immediately the NETCONF connector within the controller issues a NETCONF <rpc> edit-config to the BTI7800 device. If the modification were accommodated successfully the device replies success, else it replies failure.
5) If the response from the device is success, the operational state is modified. If the response is a failure, the config state will roll back the modification.

3.4 Implementation of OpenFlow as SouthBound Interface for Controlling BTI7800

In order to implement the OpenFlow standard within BTI7800 routers, the OpenVSwitch open source software [9] is considered. The OpenVSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension based on the OpenFlow standard.

3.4.1 BTI7800 OpenVSwitch SDN Architecture

The OpenFlow specification standard Version 1.3 does not accommodate the required optical extensions to represent the optical layer data plane devices [6]. The optical extension is developed by modifying the open source OpenVswitch code to support the BTI7800 devices. As shown in Figure 3.6, the optical extension supported by OpenVSwitch was ported on the CMM of the BTI7800. The modified BTI OpenVSwitch plugin helps in creating ports and flow connections that are referenced using OpenFlow protocol and managed by an OpenFlow controller such as ODL [3]. In the subsequent section, we present the OpenFlow optical extension developed to support BTI7800

devices. The OpenFlow protocol was not implemented from the scratch. The open source OpenVSwitch code was modified to support the BTI7800 optical extension.

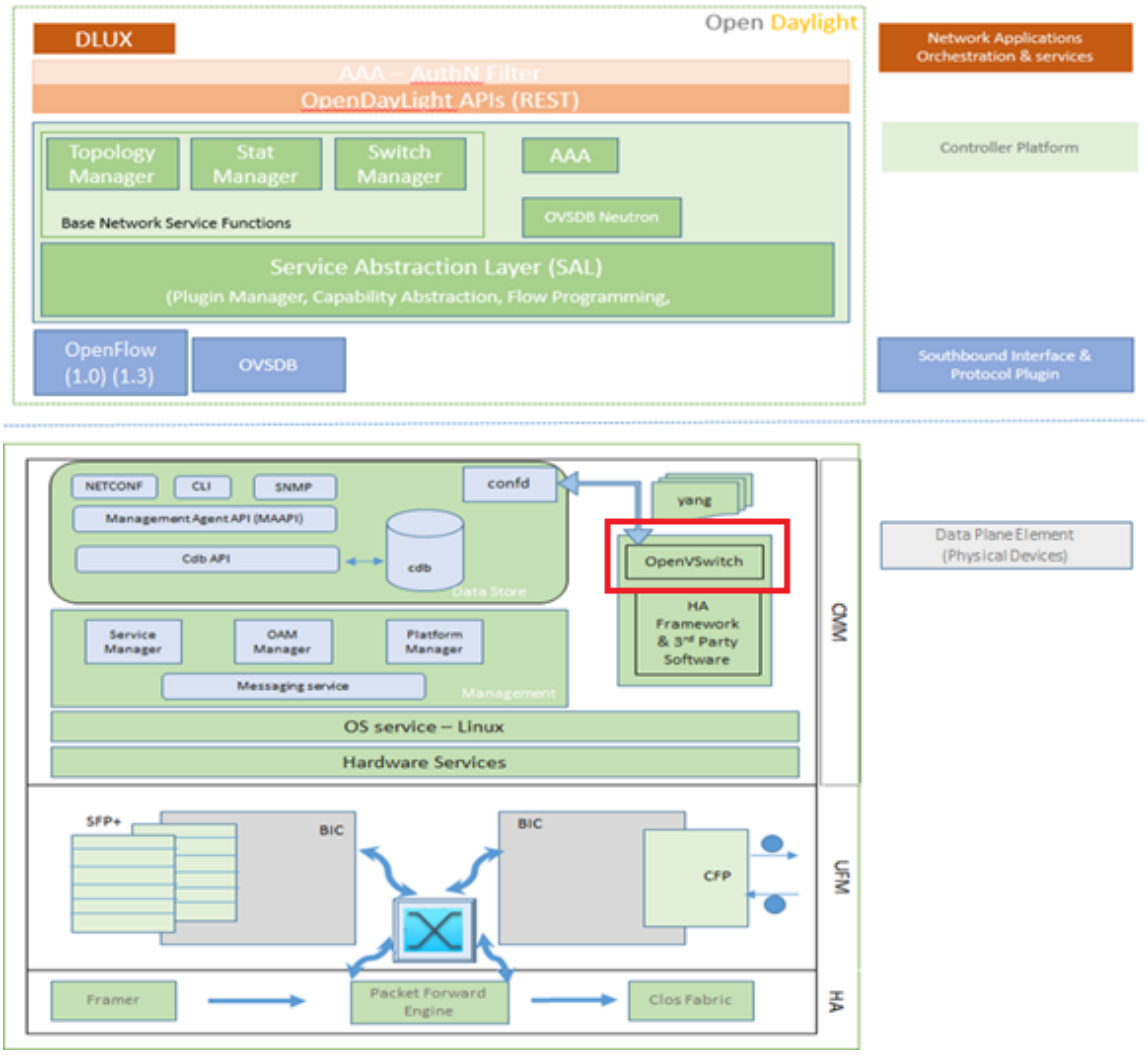


Figure 3.6 BTI7800 OpenVSwitch Architecture Overview

3.4.2 OpenFlow Optical Extension

The OpenFlow specification V1.3 released by the ONF is designed to support IP layer devices [3]. It does not accommodate any optical layer information to support the BTI7800 optical network elements. The transport extensions to OpenFlow is published

by the ONF OTWG under TS-022 [6]. The OpenVSwich opensource library does not support transport extensions specified by the OTWG. The transport extensions required to support BTI7800 network elements are built within OpenVSwitch.

The OpenFlow messages are exchanged between the OpenFlow supported devices and the controller. The OpenFlow message structures or the format of the message(s) exchanged between both entities has to strictly adhere to the OpenFlow standard. So it is not possible to modify the existing structure and by doing so the controller will not be able to decode the information properly, as the message formats at both ends will not be decoded in similar fashion. We will discuss how we utilized the existing OpenFlow message(s) to accommodate the optical extensions required for the BTI7800 without changing the message formats.

The connection establishment procedure involves some version and capability negotiation, which has to be done before any other messages can be exchanged. Therefore, we capture the connection establishment procedure in the state machine as shown in Figure 3.7.

- The hello message does not require any changes for BTI7800, the hello message specified in the OpenFlow specification [3] was used.
- The second step after the hello message is the connection establishment. As a part of the connection establishment process, the information about the optical ports supported by the BTI7800 device have to be communicated to the controller. The "portstatus" are asynchronous events sent from the switch to the controller indicating the status of the ports.

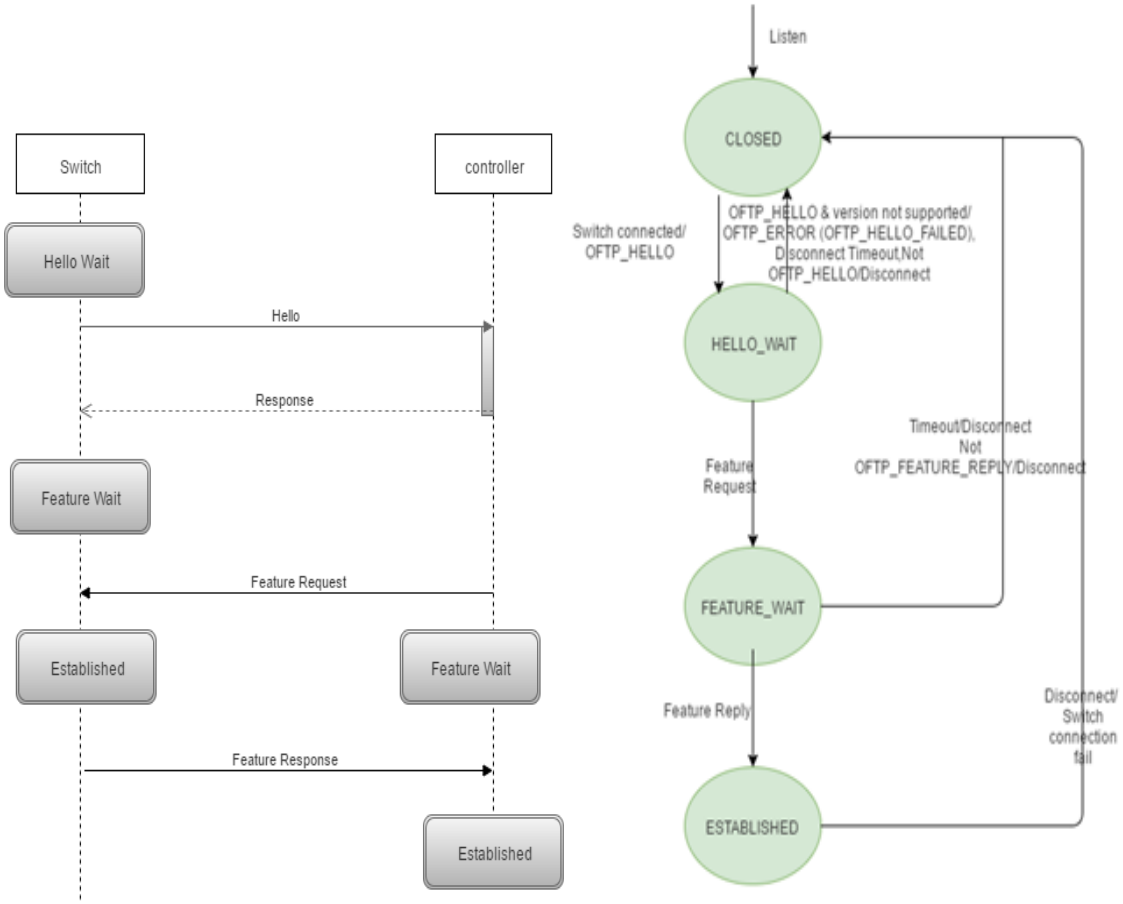


Figure 3.7 Connection Establishment Procedure

3.4.2.1 Port Status

struct ofp_port_status	
uint32_t port_no	The port_no field uniquely identifies a port within a switch. OpenVSwitch assigns integer from 1. The existing logic was reused.
uint8_t hw_addr[OFP_ETH_ALE] OFP_ETH_ALEN = 16	A unique MAC address assigned by the BTI7800 Linux kernel
char name [OFP_MAX_PORT_NAME_LE]	/* Specifies the BTI port */
uint32_t config	/* Current state of the physical port. These are not configurable from the controller. */
The curr, advertised, supported, and peer fields indicate link modes (speed and duplexity), link type (copper/fiber) and link features [3].	
uint32_t state	OFPPF_FIBER = 1 << 12, /* Fiber medium. */
All bits zeroed if unsupported or unavailable and the bitrate is specified in kbps [3].	
uint32_t curr	0
uint32_t advertised	0
uint32_t supported	0
uint32_t peer	0
uint32_t curr_speed	10GE = 10000000 100G = 100000000 /* Only these two speeds are supported by the device */
uint32_t max_speed	10GE = 10000000 100G = 100000000 /* Only these two speeds are supported by the device */

Table 3.1 OpenFlow Port Status Structure

The Port status message structure is represented in Figure 3.8(a). The header field is not modified and is implemented according to the OpenFlow standard [3]. The reason field specifies if it is an add, delete or modify operation. When the device establishes the connection with the controller, the information is communicated to the controller by specifying the reason field as add. The modify/delete operation on the ports are supported

to communicate if any changes are made while being connected with the controller [3]. The structure used to represent the status of the port is represented in Figure 3.8(b), and discussed in Table 3.1.

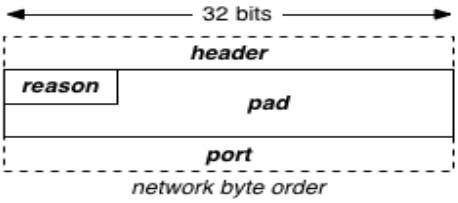


Figure 3.8(a) OpenFlow Port Status Message

Figure 3.8(b) OpenFlow Port Status Message

We used the name field to identify the BTI7800 ports and any other names not on the list will not be accepted. The state field specifies it is a fiber optics [3], the optional field was used to specify the capacity of the link.

3.4.2.2 Flow Modification

struct ofp_flow_mod	
struct ofp_header header	OpenFlow Standard
uint64_t cookie	OpenFlow Standard
uint64_t cookie_mask	OpenFlow Standard
uint8_t table_id	0x00 to 0xFF (Any table can be used)
uint8_t command	/* OpenFlow structure used enum ofp_flow_mod_command */ Applicable: OFPFC_ADD = 0, /* New flow. */ OFPFC_MODIFY_STRICT = 2, /* Modify entry strictly matching wildcards and priority. */ OFPFC_DELETE_STRICT = 4, /* Delete entry strictly matching wildcards and priority. */
uint16_t idle_timeout	NA
uint16_t hard_timeout	NA
uint16_t priority	NA
uint32_t buffer_id	NA
uint32_t out_port	**Specify the uint32_t port_no (refer to Table 3.1) information about one end port**
uint32_t out_group	NA
uint16_t flags	NA
uint8_t pad[2]	
struct ofp_match match	Refer the Standard [3] (No Change, default values according to the specification used)
struct ofp_instruction instructions[0]	Refer to the Figure 3.10 and Table 3.3

Table 3.2 OpenFlow Flow Modification Structure

The controller is aware of the ports available within the device. Now the flow rules for rerouting the packets within the device have to be communicated. The OpenFlow flow modification message [3] implemented to support the BTI7800 device is discussed below. In the BTI7800, the nature of the flow is to link two ports, known as cross-

connect. The information about the port from Table 3.1, uint32_t port_no uniquely identifies each port and is used to specify the port that needs to be cross-connected.

The flow modification information has to contain two end port information (uint32_t port_no) referenced in struct ofp_port_status (refer to Table 3.1) and their operation (add, modify, delete). The flow modification message is represented in Figure 3.9 and discussed in Table 3.2.

Figure 3.9 OpenFlow Flow Modification Message

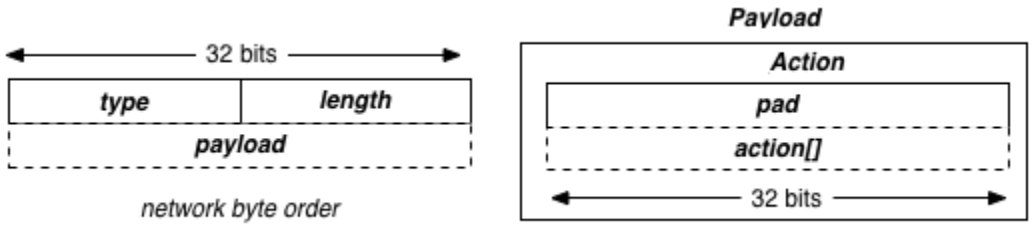


Figure 3.10 Flow Instruction Message

The instruction field specifies what action has to be taken at the port specified in uint32_t out_port (refer to Table3.2). The flow instruction structure is represented in Figure 3.10

and discussed in Table 3.3, and it is a part of the OpenFlow Flow Modification Structure represented in Table 3.2.

<table>
<tr><td colspan="2">struct ofp_instruction</td></tr>
<tr><td>uint16_t type</td><td>There are different flow instruction types supported, refer to ofp_instruction_type [3].
Only the following type belonging to ofp_instruction_type is applicable.
OFPIT_APPLY_ACTIONS = 4, /* Applies the action(s) immediately */</td></tr>
<tr><td>uint16_t len</td><td>/* Length of this struct in bytes. */</td></tr>
<tr><td>Payload</td><td>struct ofp_action_output
<table><tr><td>uint16_t type</td><td>/* OFPAT_OUTPUT. */ [3]</td></tr><tr><td>uint16_t len</td><td>Length of the message</td></tr><tr><td>uint32_t port</td><td>Specify the uint32_t port_no (refer to Table 3.1) information about other end port</td></tr><tr><td>uint16_t max_len</td><td>Always set to zero [3]</td></tr><tr><td>uint8_t pad[6]</td><td></td></tr></table></td></tr>
</table>

Table 3.3 Flow Instruction Structure

3.4.2.3 Launching OpenFlow Plugin

As specified earlier, the BTI7800 device supports a customized Linux environment with very few support packages. The required dependencies [9] for compiling the modified OpenVSwitch code are installed and the binaries created are linked to the existing BTI7800 binaries.

3.4.3 BTI7800 OpenVSwitch Architecture

The implemented OpenVSwitch architecture is depicted in Figure 3.11. The OpenVSwitch modules are broadly classified into management, user space and kernel space module.

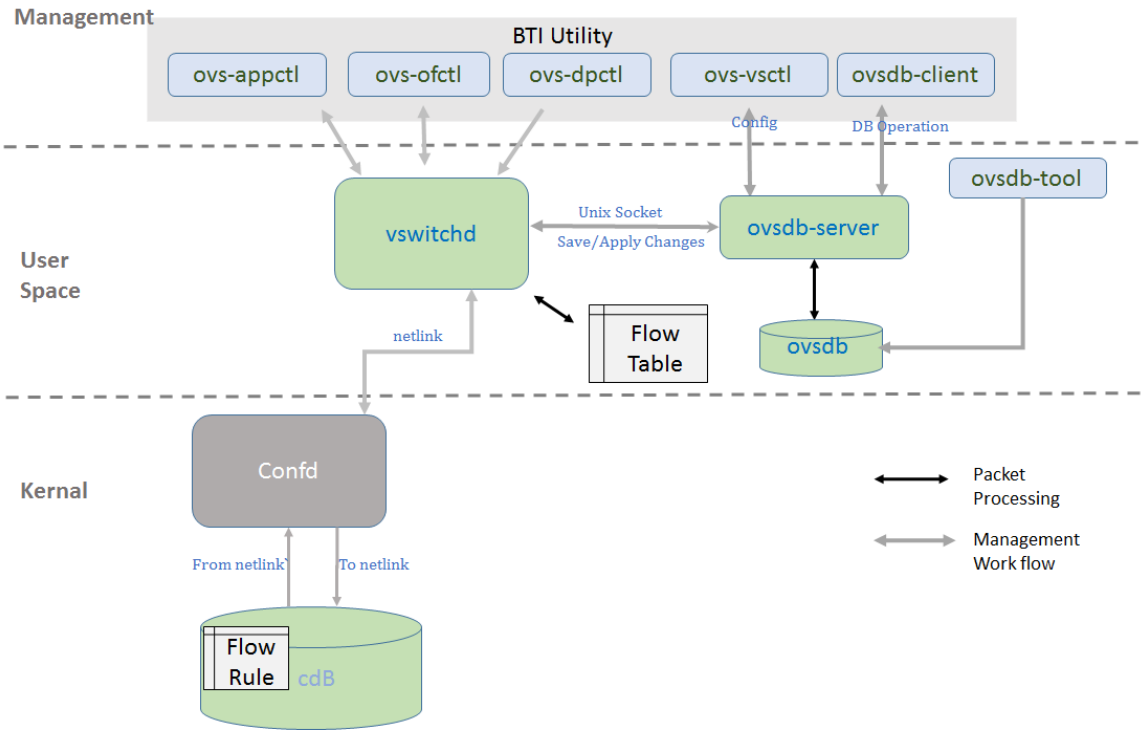


Figure 3.11 BTI7800 OpenVSwitch OpenFlow Architecture

3.4.3.1 Management

The Management module of the OpenVSwitch is composed of utilities that help to configure the OpenFlow supported BTI7800 devices and they offer a high level interface for users to provide inputs. They also provide an interface to interact with the ovsdb - server module.

- **ovs-appctl** - The utility for configuring and running the OpenVSwitch process. The OpenVSwitch runs as a separate Linux process and it also manages the OpenVSwitch process as part of the BTI7800 switch daemon. The BTI7800 switch daemon is the background parent process that controls the device and the host of all the sub processes required to support the various switch functionality. It is essential to host the OpenVSwitch process as a sub process, to ensure effective memory allocation and inter process communication.

- **ovs-ofctl** - The utility for administering the OpenFlow supported BTI7800 switches and to manage the connections with controllers.
- **ovs-dpctl** - The utility for administering BTI7800 datapaths.
- **ovs-vsctl** - The utility for querying and configuring the ovs-vswitchd to support BTI7800 ports and interfaces.
- **ovsdb-client** - The CLI to ovsdb-server running in user space.

The above listed utilities are supported by the OpenVSwitch, the support was added by modifying the OpenVSwitch code to function in the BTI7800 environment.

3.4.3.2 User Space

The user space consists of the vswitchd module, that functions as an OpenVSwitch process. It is responsible for managing and controlling the OpenVSwitch process in the device. It connects with the ovsdb-server. The vswitchd is divided into two sub modules as shown in Figure 3.12 and described below.

ovs-vswitchd - It is the OpenVSwitch user space program, which creates the vswitchd process. The CMM module of BTI7800 is composed of multiple different processes as a part of the BTI7800 switch daemons. The OpenVSwitch process is handled to be a part of BTI7800 switch.

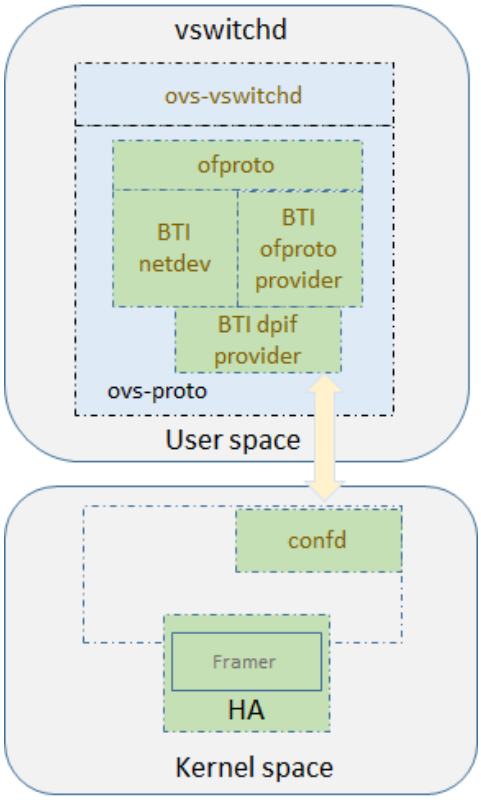


Figure 3.12 BTI OpenVSwitch User Space Architecture

ovs-vswitchd - It is the OpenVSwitch user space program, which creates the vswitchd process. The CMM module of BTI7800 is composed of multiple different processes as a part of the BTI7800 switch daemons. The OpenVSwitch process is handled to be a part of BTI7800 switch.

ovs-proto – It is the OpenVSwitch library that implements an OpenFlow specification. It is responsible for establishing the communication with the SDN controller. It receives the configuration and modification from the utilities discussed in Section 3.4.3.1 or as OpenFlow message discussed in Section 3.4.2. The ovs-proto translates the received information as input(s) to the ovsdb and to the kernel space and handles them accordingly.

- **ofproto** - It is responsible for establishing communication with the SDN controller. It handles the OpenFlow messages and communicates with the controller.
- **BTI netdev** – It is the thin library layer that links the actual ports and interfaces on the BTI7800 and their corresponding OpenFlow references. With the help of BTI netdev it is possible to open BTI7800 device ports.
- **BTI ofproto provider** – It supports flow table entries. It receives and validates the flow information before passing it on to lower layers. It also writes the validated flow information to ovsdb.
- **BTI dpif provider** – BTI dpif provider is a software message queue implementation that interfaces with the confd module of the existing BTI7800 architecture. Any information from the upper layer (the BTI ofproto provider or the BTI netdev) is coined as message and posted to confd.

3.4.3.3 Kernel Space

The confd is a management module present in the BTI7800, it supports the API for making changes onto the device hardware. The confd is the module of BTI7800 that receives the messages from the BTI ofproto provider. The confd will process the information and as a result the flows or port will be modified accordingly. The confd configures the cdb based on the changes triggered by the BTI OpenVSwitch plugin. As discussed earlier, cdb is the configuration database of the BTI7800. The flow rules are placed in the cdb as shown in Figure 3.11 for two reasons:

1) Fast processing of the packet, as it minimizes the effort of multiple context switching and the effort involved in flow matching across the tables.
2) To ensure reuse of the code and device architecture. When the packets are received on the interface the routing of the packets will be done by the interface.

3.4.4 BTI7800 OpenVSwitch Operation

Figure 3.13 shows the various operations involved in the OpenFlow supported BTI7800, and are described in detailed below.

OVS Process creation

1) The OpenVSwitch (vswitchd) process starts automatically when the device boots up.
2) The vswitchd process automatically creates the data structure representing the optical ports supported by the BTI7800 and references them using OpenFlow.

BTI7800 port creation

3) The information about the ports are received at the ovs-ofproto. The BTI netdev sub module within the ovs-ofproto validates and processes the information and also updates the ovsdb.
4) The BTI dpif provider within the ovs-proto will post the messages to confd (ovs-kernel) and will provision the hardware accordingly.
5) It is also possible for the users to view and modify the ports manually using BTI ovs-vsctl management utilities through the CLI.

Controller Information

6) The user provides the controller information (IP address and port number) using the management utility function ovs-ofctl discussed in Section 3.4.3.1.

7) The ovs-ofproto uses that information to connect to the controller and establishes a secure protocol communication for exchanging OpenFlow messages.

8) If the socket communication is established successfully the controller is capable of managing the device using OpenFlow.

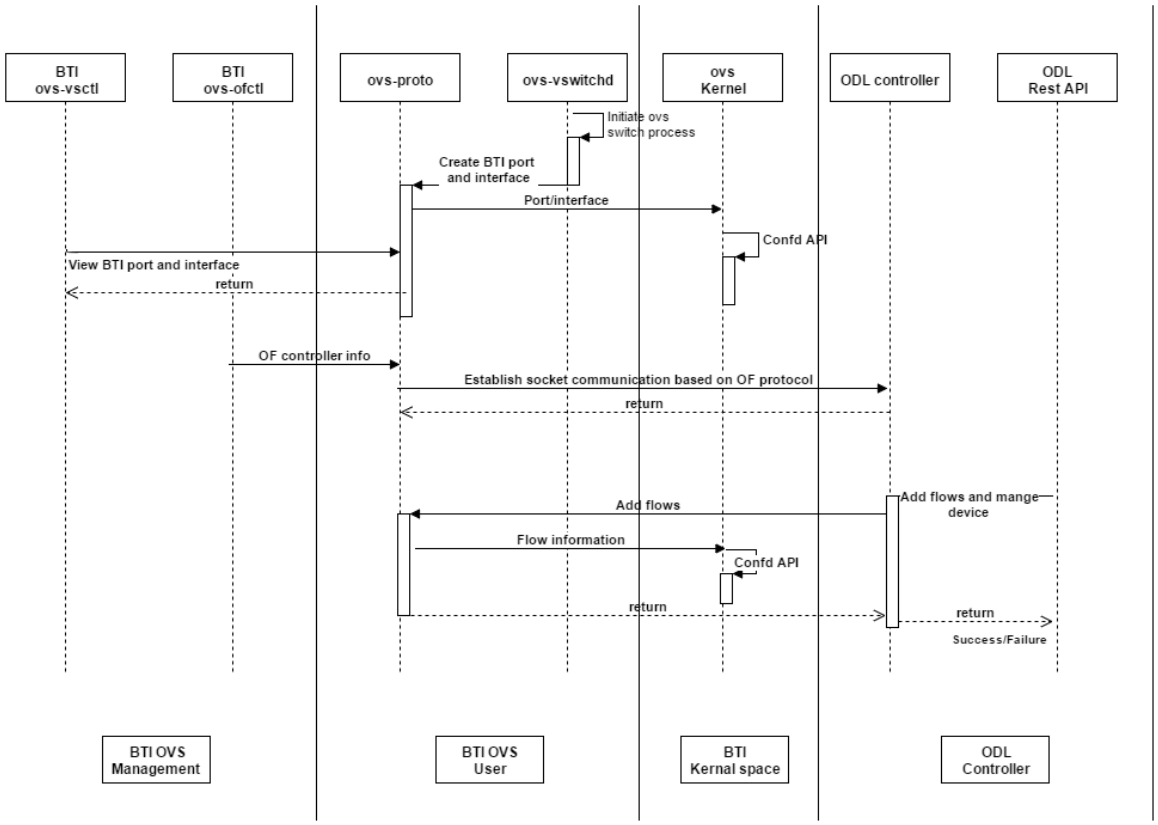


Figure 3.13 BTI7800 OpenVSwitch Process

Add Flows

To provide a detailed understanding of flow modifications, the sequence is also explained with a state diagram

9) Through the ODL REST API it is possible to provide input to the controller for flow modification. The controller issues a controller-to-switch message indicating there is modify-state information to the connected BTI7800 device as shown in Figure 3.14.

10) The message is received at vswitchd, the ofproto within vswitchd handles the messages. The received information is validated by the BTI ofproto provider. If successful, the flow is added into the ovsdb and flow table, else it returns failure.

11) The BTI dpif provider receives the flow information and a message is posted to the message queue and it returns success.

12) The message is received in the kernel space; the message handler provides the message content to confd. The confd updates the cdb and other existing BTI7800 device modules respectively.

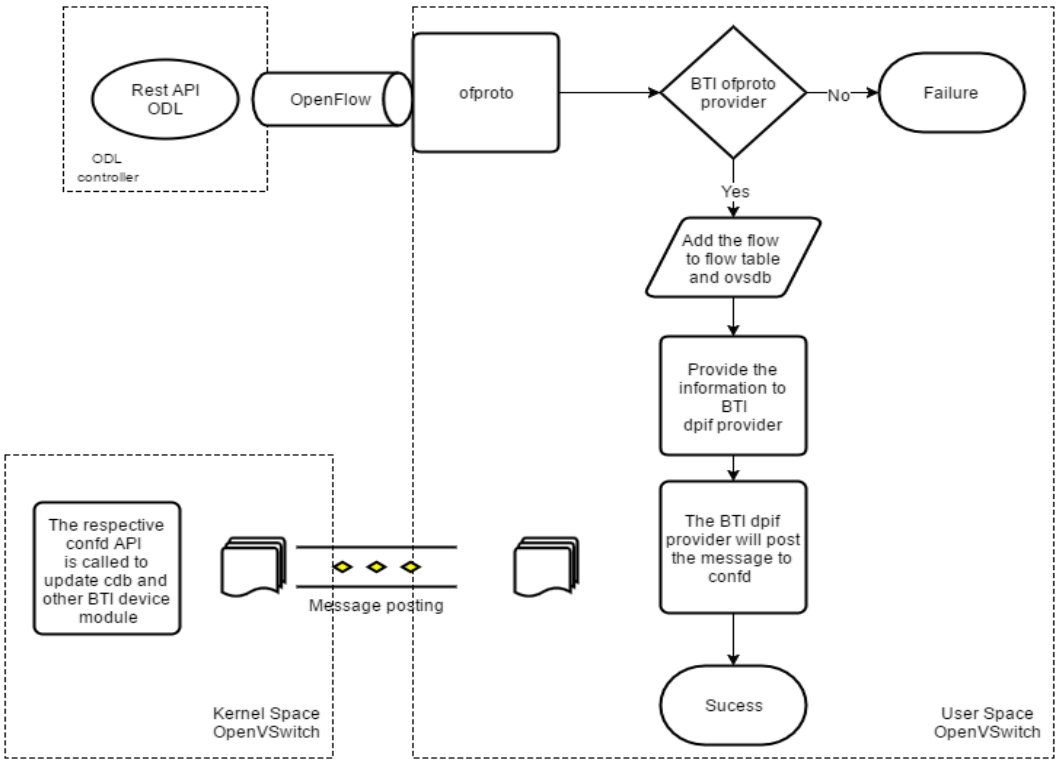


Figure 3.14 OpenFlow Flow Modification State Diagram

3.4.5 Managing the BTI7800 using OpenFlow

We discussed the BTI OpenFlow plugin architecture and its functionality. Figure 3.15 represents the OpenFlow messages between the ODL controller and OpenFlow supported BTI7800 devices.

1) Once the BTI7800 switch is aware of the controller IP address, it initiates a TCP connection on port 6633 to connect to the controller.
2) Once the TCP communication is established, both the controller and the BTI7800 exchange OPFT_HELLO [3] messages with supporting OpenFlow versions. The hello message and corresponding echo hello message are symmetric messages in OpenFlow.
3) The flow modification (add, update, delete) operation can be performed using the REST API supported by the ODL controller.
4) The ODL informs the connected OpenFlow BTI7800 device about the modification through OpenFlow messages.
5) The information is updated in the ovsdb and in the cdb as discussed in Section 3.4.4.
6) If the response is success the operational state of the device is updated, else the modification stored in the config state is rolled back.

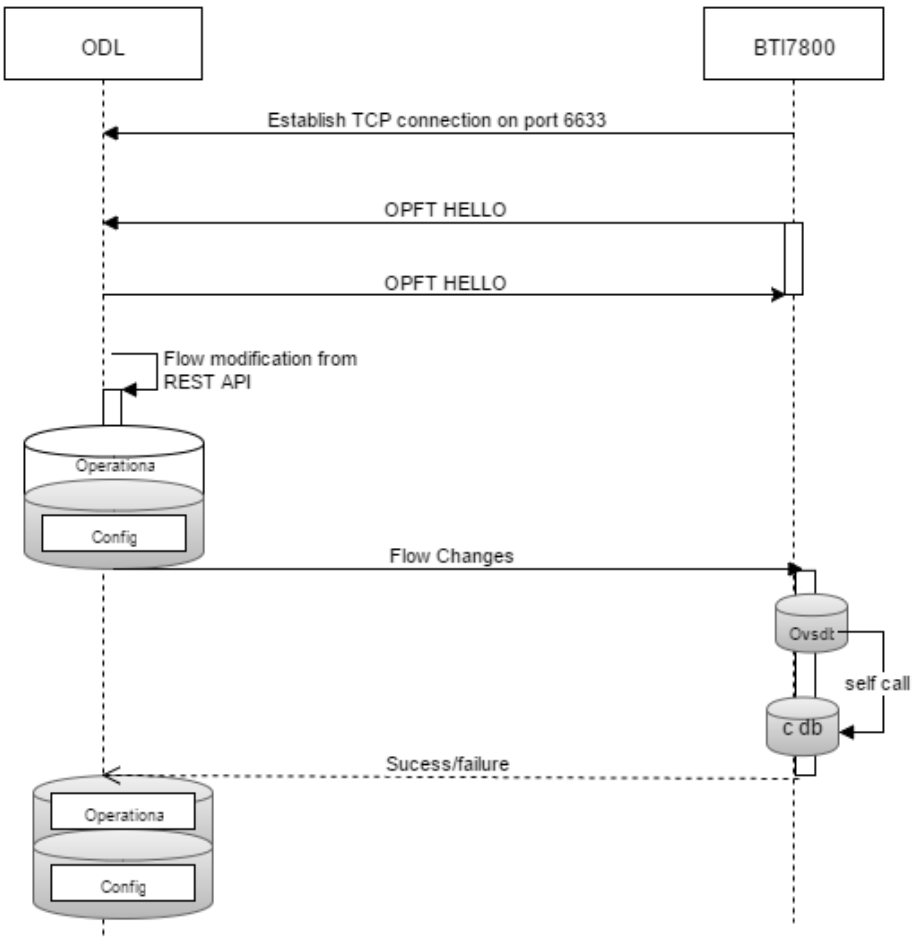


Figure 3.15 OpenFlow Interaction between the Controller and BTI7800

3.5 Managing Data Center Interconnection

Until now we have discussed managing a single device using NETCONF and OpenFlow as the SBI. In our research, we deal with optical data center interconnection typically as shown in Figure 3.16. The BTI7800 acts as an edge node in the data centers, we need to manage two nodes using an ODL controller. In case of NETCONF, the ODL controller creates an individual NETCONF connector for each device attempting to connect. Similarly, for devices using OpenFlow as SBI, a new instance of the OpenFlow connector is assigned to every connected device. All attached devices have an operational

and config state maintained individually. So it is also possible to manage multiple devices with a controller and each node is identified separately. When the two edge BTI7800 nodes are connected to the controller using their respective supported SBI, it is possible to manage and control both the devices. Therefore, it is also possible to manage the link interconnecting those devices.

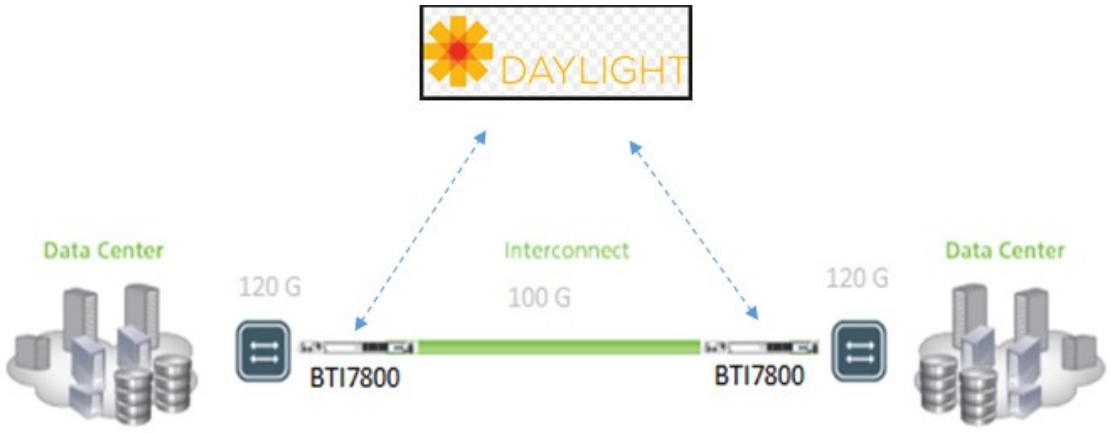


Figure 3.16 Data Center Interconnection

In order to demonstrate the different scenarios of BoD, a python-based application layer is written. The python script translates the user requirement into a REST API request for the controller. The user requirements are to manage the flows between the interconnected data center. The controller processes the REST API request and as a result the flow(s) between data centers can be managed. The detailed description of the test environment and use cases are provided in the next chapter.

Chapter 4: Test Environment and Experiments

In this chapter, the proposed SBIs, NETCONF and OpenFlow implemented on the BTI7800 network elements, are evaluated. We have discussed about the technology, the history, the evolution and the implementation of the interconnected SDN optical data centers, in our previous chapters. In this chapter we present the test environment and the use cases that help us in evaluating both protocols that act as a SBI. The use cases in this chapter are designed based on the ONF SDN optical transport use cases document [7].

The demonstrated use cases address the data center interconnection, serving traffic flows between two separate data centers connected over an OTN. One of the main characteristics of this use case is that the provider network is shared with other client traffic. Another characteristic of this use case is the control separation of the data center and the provider network. The use cases are specifically designed to handle BoD for users across data centers with the help of SDN at the optical layer and switching [7]. In this chapter, we explain the test environment, use cases, performance and evaluation of both protocols in each scenario. The use cases are broadly classified as detailed below:

- High Priority Flow (20 G) Request for User B
- High Priority Flow (20 G) Request for User A
    - When no low priority flow exists
    - When the lower priority flow exists
- Two Users' Request for High Priority Flow, Mutual Sharing

4.1 Test Environment and Evaluation Metrics

We describe the test environment and the metrics used to evaluate both protocols implemented as SBIs in an SDN environment.

4.1.1 Environment Description

The test environment, as shown in Figure 4.1, is comprised of the following equipment:

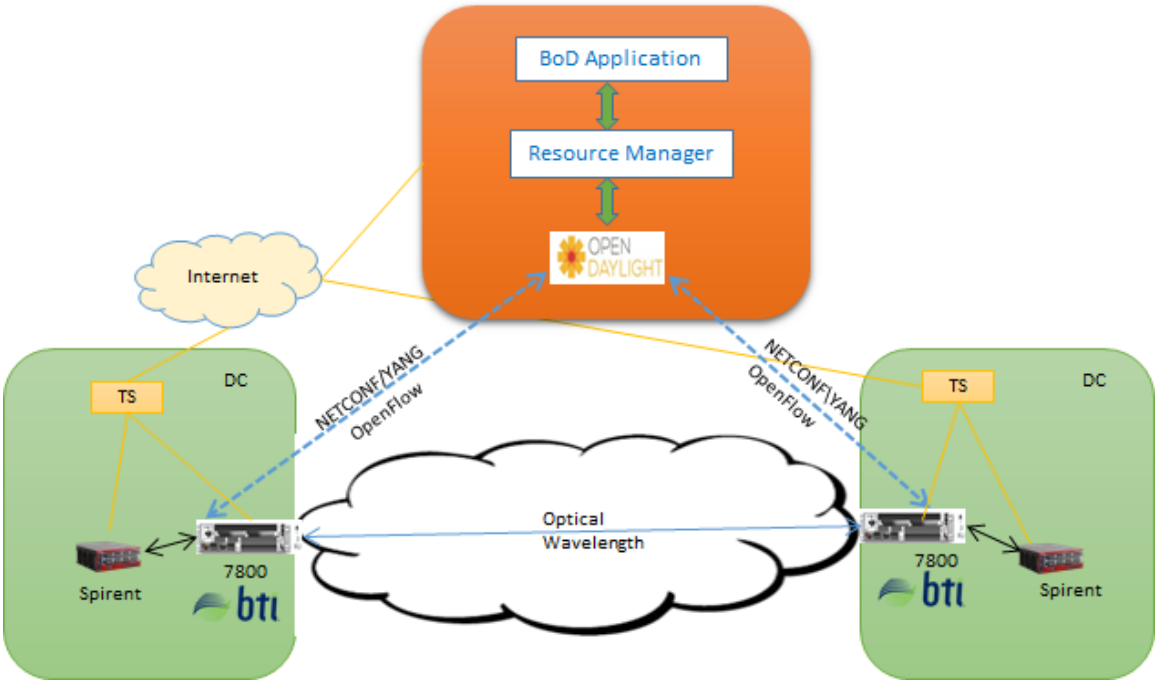


Figure 4.1 Test Environment Setup

A. **Two BTI7800**: The BTI7800 optical devices form the data center interconnection with the help of 12*10 G link capacity and a 100 G OTU4 link. The 12*10 G links support 12 users of each 10 G bandwidth for user traffic within a data center and the 100 G OTU4 link connects across the data centers. The Spirent traffic generator acts as user traffic source. There is a possibility of 120 G traffic input from users within the data center. The OTU4 link connecting between data centers supports a maximum link bandwidth of 100 G. The optical router receives the input traffic of 120 G but can

only multiplex and forward traffic of 100 G on the OTU4 link interconnecting data centers. Therefore, the bandwidth requirement pertaining to 10*10 G links can be satisfied, with potentially dropping the traffic from the remaining 2*10 G links. The SDN controller has to meet the requirements of BoD by granting bandwidth to more essential user traffic and dropping other users whose requirement cannot be satisfied.

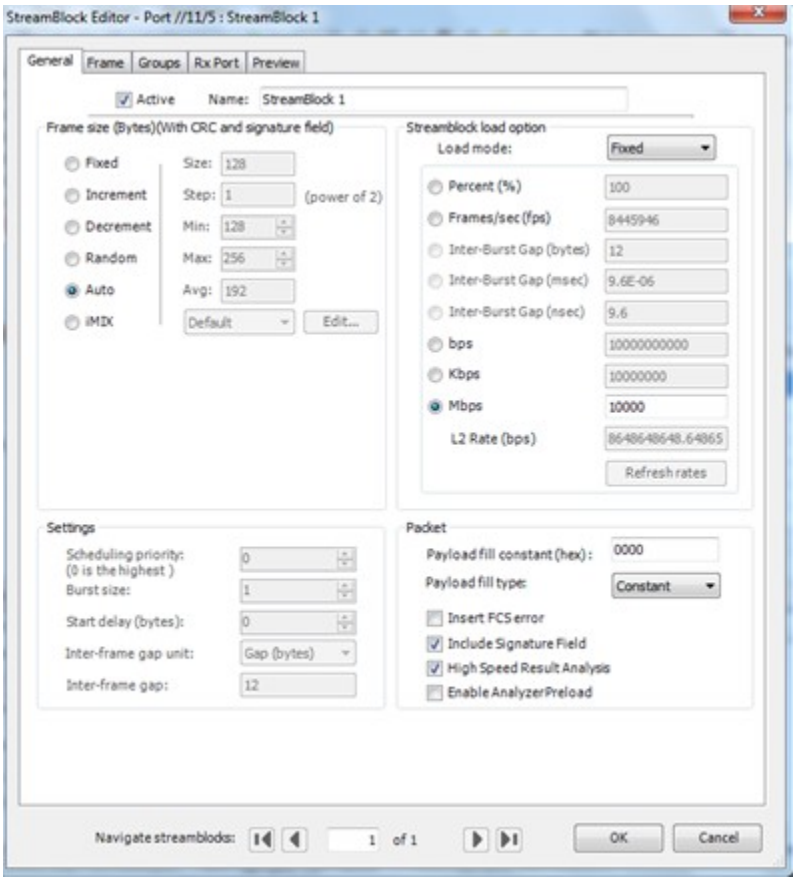


Figure 4.2 Spirent Traffic Generator Configuration

B. **Spirent**: Spirent is a traffic generator; it helps in creating user traffic within the data center. It is connected with the 12 ports on the BTI7800 with the help of 10 G optical fibers. Each link supports auto generated traffic source as shown in Figure 4.2. The Spirent at one end of the data center acts as the sender and the Spirent at the other data center acts as the receiver. The Spirent traffic generator reports the bits

transferred and received at each link every second. This information is reported in the form of graphs and used to evaluate both protocols for all use cases.

C. **ODL Controller**: The ODL SDN controller is running on a centos server. The SDN controller interfaces with network devices (BTI7800) using the SBI, either OpenFlow or NETCONF. Each connected BTI7800 device is identified separately by the controller. The SDN BoD application (python script) is running on the same server. The resource manager collects information about the devices and its topology using the REST NBIs supported by the ODL controller. The BoD application receives a bandwidth request from the user and the information is passed on to the resource manager. The resource manager, based on the stored information and depending on the nature of the traffic, handles the BoD scenarios by translating user requirements as ODL REST API information to the SDN control plane. The REST API contains information that uniquely identifies the connected device(s) and their connection changes [8].

   - Operating system – centos OS (Linux pronx10 2.6.32-504.el6.x86_64)
   - RAM – 16 GB
   - ODL version - ODL Lithium Karaf

The Spirent traffic generator cannot be controlled using the SDN controller. Therefore, they always generate traffic based on the configurations discussed above. The application layer of the ODL controller offers an interactive mode to select a use case and it also signals the control plane about the type of traffic and the flow modification decision(s).

4.1.2 Evaluation Metrics

In all the use cases the behavior of both protocols as SBIs are observed. In order to discuss the observations, it is essential to have concrete metrics to evaluate both protocols. The following metrics are considered.

- Time: We are comparing two protocols that act as SBIs i.e., that communicate between the SDN control plane and the data plane devices. An efficient protocol should exhibit a fast and reliable response. The time taken to complete the required set of configurations by both SBIs are compared. The time period for each use case differs depending on when the system attains the steady state. The use case reaches the steady state when all the configurations are committed, the traffic flows end-to-end in all the links that are active and the system remains in a stable state. In the steady state all the links are provisioned end-to-end and the system is in stable state.
- Bandwidth Utilization: The research is focused on handling BoD requests across the interconnected data centers managed using the SDN SBIs. One of the key aspects for adopting SDN in the current network management is because it provides a fast and efficient way to manage the networks. Therefore, resources can be optimally used. The bandwidth utilization is calculated based on this formula.

$$Bandwidth\ Utilization\ (\%) = \frac{Total\ No.of\ bits\ successfully\ transferred}{Total\ No.of\ bits\ inputted} * 100$$

For a use case the bandwidth utilization is calculated over a period of time from an event a use case is triggered by the user and until system attains steady state. For each use case the period of time differs based on when the system completes the

requirement(s) and attains the steady state. All the experiments start with the system operating in the initial state. New user traffic demands additional bandwidth and the SDN controller decides to service the bandwidth requests based on the nature of the traffic and availability of the resources. The SDN control plane communicates the changes through the implemented SBIs and waits until the system completes the requirement(s). Once the requirement(s) are completed, the system returns to its initial state. The time period for both SBIs in each use case is equal: if one of the implemented SBIs attains the steady state faster than the other, the environment is maintained in the steady state until the environment implementing the other SBI also attains the steady state. In steady state, if all links are active, the bandwidth utilization is always 100 %.

- Control Messages: The ODL SDN controller manages the data plane devices via the implemented SBIs. The SBIs communicate the flow modification information as control messages. Each SBIs handles the messages differently based on the nature of the protocol which leads to difference in message size and number of messages. An efficient protocol should less control signaling and network overhead.

Having discussed three parameters, time and bandwidth are equally important to achieve QoS among the supported users and the different kind of traffic. Control messages are not very crucial, but network overhead caused by control signals needs to be accounted for designing a system.

4.2 Initial Environment

The initial environment shown in Figure 4.3 is used for all the use cases, unless a different initial environment is presented in the beginning of a use case. Both data centers are identical in network topology and are interconnected with the help of the OTU4 optical link. In both data centers the BTI7800 device acts as an edge node. Each data center has two users that are connected with the BTI7800. The nature of traffic and the amount of bandwidth occupied by each user is discussed below.

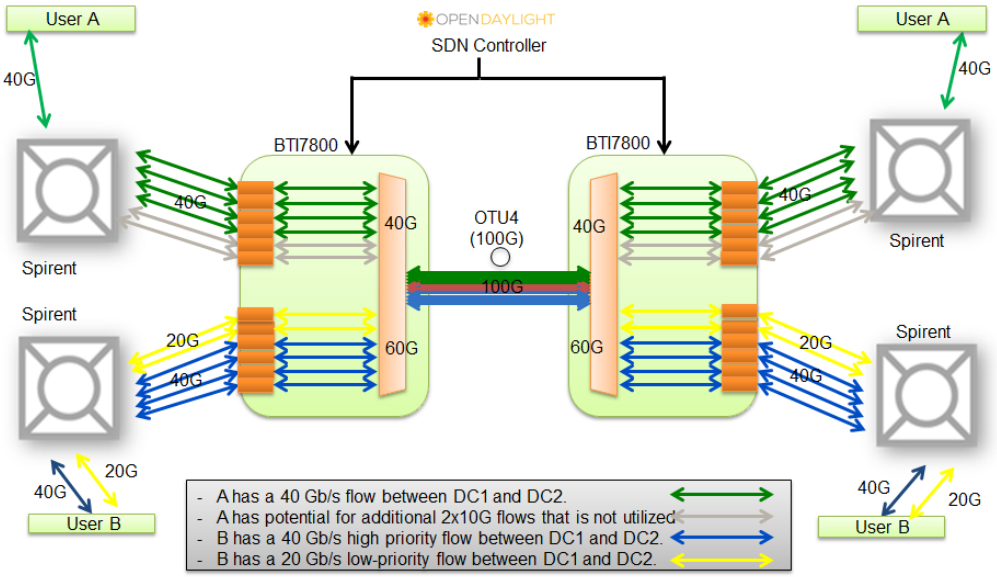


Figure 4.3 Initial Environment

User A	User B	Type of Flow
40 G (Green)	40 G (Blue)	Base Flow
	20 G (yellow)	Low priority

Table 4.1 Initial Environment Allocation

- Both data centers have two users, user A and user B respectively. Each user owns a potential bandwidth of 6*10 G connections patched with the BTI7800. The OTU4 100 G link connects both optical routers across the data center as shown in Figure 4.3,

and it is capable of multiplexing user traffic on the optical link connecting data centers.

- Each user has a dedicated bandwidth of 40 G each, constituting 80 G out of 100 G bandwidth (blue and green links). The sub flows or cross-connects 1 to 4 are termed as the base flow of user A and represented with green links. The sub flows or cross-connects 5 to 8 are termed as the base flow of user B and represented with blue links.
- One of the users, "user B", is assigned the remaining 20 G for low priority traffic (yellow color links), as represented in Table 4.1. The sub flows or cross-connects 9 and 10 are termed as the low priority flow of user B and represented with yellow links.
- In all scenarios discussed below, when a high priority traffic (red color links) of 20 G is requested, the low priority traffic (yellow color links) is preempted. Once the high priority traffic is serviced, the system will return back to the initial state.
- The sub flows or cross-connects 11 and 12 are termed as the high priority flow of user A and represented with red color links. The sub flows or cross-connects 9 and 10 are termed as the high priority flow of user B and represented with red color links.
- The user B can either have lower priority flow (sub flows 9 and 10 – yellow color links) or higher priority flow (sub flows 9 and 10 – Red color links) depending on the use case.

The links (arrows) represented within the BTI7800 are known as cross-connects or sub flows. They help in multiplexing/bridging any 10 of 12 (*10 G) ports on to the OTU4 link. Based on the user traffic, the SDN controller signals which 10 ports need to be cross-connected to the OTU4 link.

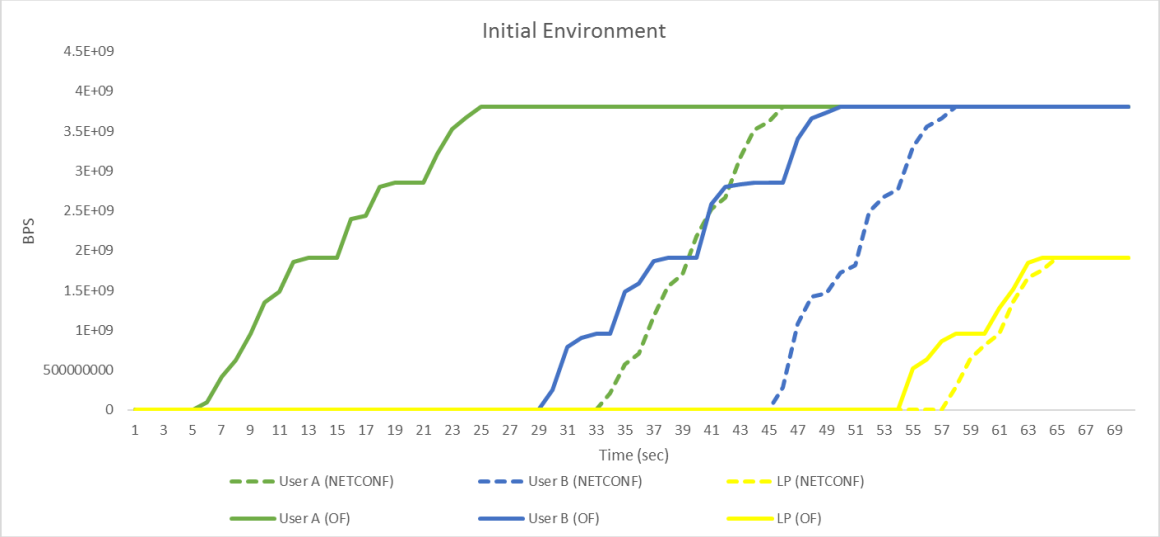


Figure 4.4 Initial Environment Provisioning Using NETCONF and OpenFlow

4.2.1 Flow Management Using NETCONF Interface

The process involved in the flow management with the help of the controller using NETCONF as an SBI are discussed in Section 3.3.4. The NETCONF and YANG-based protocol represents the information as a collection of dataset(s) as described in Section 2.4.1. The list of cross-connections is seen as a single dataset and the ODL controller cannot access each cross-connect individually. Figure 4.4 shows the time taken to set up the initial environment shown in Figure 4.3 implementing NETCONF and OpenFlow as a SBIs. In the interconnected data centers, for the traffic to flow from one end to another, it is essential to provision the end-to-end tunnel. The application layer first provisions the data center at one end, but on the other end there is no connection provisioned. Therefore, the traffic cannot reach the other end. Next, the application layer begins to provision the data center at the other end. As the result the end-to-end tunnel is created for each sub flow and eventually traffic begins to flow end-to-end.

The sequence of steps involved in provisioning the initial configuration using NETCONF as SBI is shown in Figure 4.5 and are discussed below.

1. The user interacts with the BoD application requesting the SDN controller to provision the initial environment.
2. The resource manager receives the user's request from the BoD application and it determines the list of sub flows or cross-connects required to be provisioned on both BTI7800 devices.
3. The resource manager provides the control plane with the sender BTI7800 device information as identified by the NETCONF connector and the list of all 10 cross-connects needed to be provisioned.
4. The control plane translates the received information as NETCONF <edit-config> message, and the message is directed toward the sender BTI7800 NETCONF agent.
5. The NETCONF agent residing on the BTI7800 device will process the requested cross-connects. On configuring the requested cross-connects it returns success back to the control plane and the response is further delivered to the resource manager.
6. The resource manager provides the control plane with the receiver BTI7800 device information as identified by the NETCONF connector and the list of all 10 cross-connects that needs to be provisioned.
7. Step 4 and step 5 are repeated again for the receiver BTI7800 device.

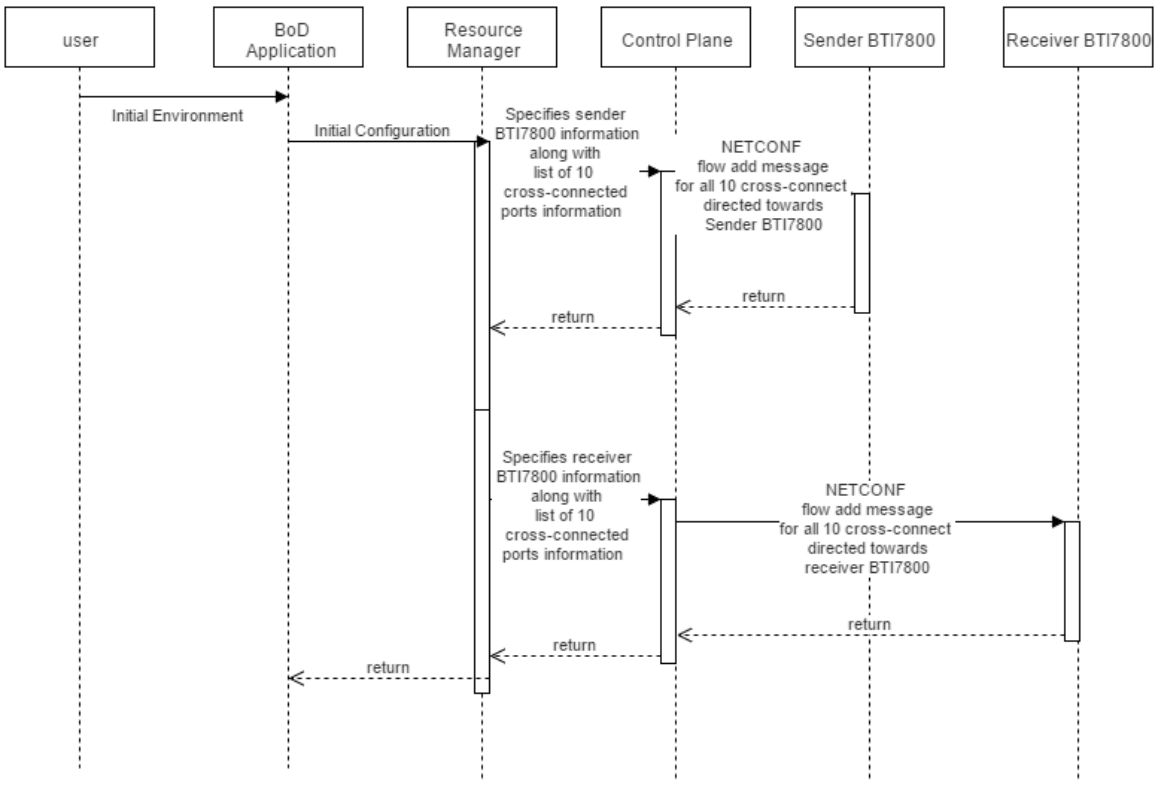


Figure 4.5 Sequence for Initial Configuration Using NETCONF

Time: At time t = 1 sec in the graph represented in Figure 4.4, the SDN controller communicates the set of cross-connects to be provisioned on the sender BTI7800 device. In the beginning there is no cross-connects and the packets are dropped within the sender BTI7800 device. Once the sender BTI7800 device is provisioned, on the receiver data center there is no cross-connects provisioned, and any packets received are dropped within the BTI7800 device located at the receiver. At the end of t = 31 sec, cross-connects on the sender BTI7800 device are provisioned. The controller, using the implemented NETCONF SBI, communicates the cross-connect information that needs to be provisioned to the BTI7800 device on the receiving end. We observe the traffic being successfully received at the user for each link eventually over the time t = 32 sec to t = 70 sec. At t = 34 sec the first sub flow end-to-end communication is established. At t = 44

sec the base flow of user A is established end-to-end. At t = 63 sec all the links are provisioned end-to-end. From t = 63 sec until t = 70 sec the system is in steady state with all the 10 links active and fully utilized.

The BTI7800 YANG model handles cross-connects as a single dataset. The controller does not allow to access them as individual cross-connect and it is forced to access them as a set of cross-connects. Using the implemented NETCONF SBI, the list of cross-connects that needs to be provisioned are specified, and the BTI7800 device receives the request and further process each cross-connect information specified in the list. At the end of the request it returns the status back to the controller.

The BTI7800 devices do not support buffering of data, therefore packets cannot be stored. The reason for the long delay is because the BTI7800 have to create an optical tunnel between the two end ports that are cross-connected. The BTI7800 device takes approximately 3 seconds to process each cross-connect. When we reverse the order of the BTI7800 devices provisioned, starting with the BTI7800 device at the receiver end first, followed by the BTI7800 at the sender side, we observe the same overall behavior.

Bandwidth Utilization: The bandwidth utilization is calculated for a time period of t = 70 sec. The bandwidth utilization is calculated from the beginning of the experiment at t = 1 sec to t = 70 sec, the end of the experiment until the system attains a steady state and remains in steady state for a few seconds. The bandwidth utilization is 32.01% to attain the initial state as described in Section 4.2.

4.2.2 Flow Management Using OpenFlow Interface

OpenFlow, unlike NETCONF, operates on each sub flow individually. As discussed in Section 4.2.1 the application layer provisions both data centers for each flow separately. Eventually we see traffic flowing across data centers as represented in Figure 4.4. As each flow is provisioned separately, the amount of time the link is not utilized is less compared to NETCONF. The sequence of steps involved in provisioning the initial configuration using OpenFlow as SBI is shown in Figure 4.6 and are discussed below.

1. The user interacts with the BoD Application to request the SDN controller to provision the initial environment.
2. The resource manager receives the user's request from the BoD Application and it determines the list of sub flows or cross-connects required to be provisioned for the both BTI7800 devices.
3. The resource manager provides the control plane with the information about the sender BTI7800 device and first sub flow or cross-connects needed to be provisioned.
4. The control plane translates the received information as OpenFlow add message, and the message is directed toward the sender BTI7800 device.
5. The sender BTI7800 device, on receiving the OpenFlow add message, provisions the new cross-connect and returns success.
6. The resource manager provides the control plane with the information about the receiver BTI7800 device and the first sub flow or cross-connect needed to be provisioned.
7. The control plane translates the received information as OpenFlow add message, and the message is directed toward the receiver BTI7800 device.

8. The receiver BTI7800 device, on receiving the OpenFlow add message, provisions the new cross-connect and returns success.
9. Step 3 through step 8 are repeated for all remaining 9 sub flows or cross-connects.

Time: At time t = 1 sec in the graph represented in Figure 4.4, the SDN controller communicates the first sub flow information to the sender data center, and following the same information is communicated to the other data center. At t = 6 sec, the first sub flow is setup end-to-end and we see traffic sent from one end of the data center and successfully received at the other end of the other data center. At t = 61 sec all ten sub flows are provisioned at both data centers and traffic in all links are active. From t = 61 sec until t = 70 sec the system is in steady state with all the 10 links active and fully utilized. The device takes 3 seconds to provision a cross-connect in an OpenFlow environment.

The time taken to create or tear down the cross-connect(s) presented in this thesis are specific to the BTI7800 devices. The authors of [13] present the hardware time of 7 seconds to setup a cross-connect in an extended optical OpenFlow environment, the hardware setup time varies for different devices based on their performance and functionality.

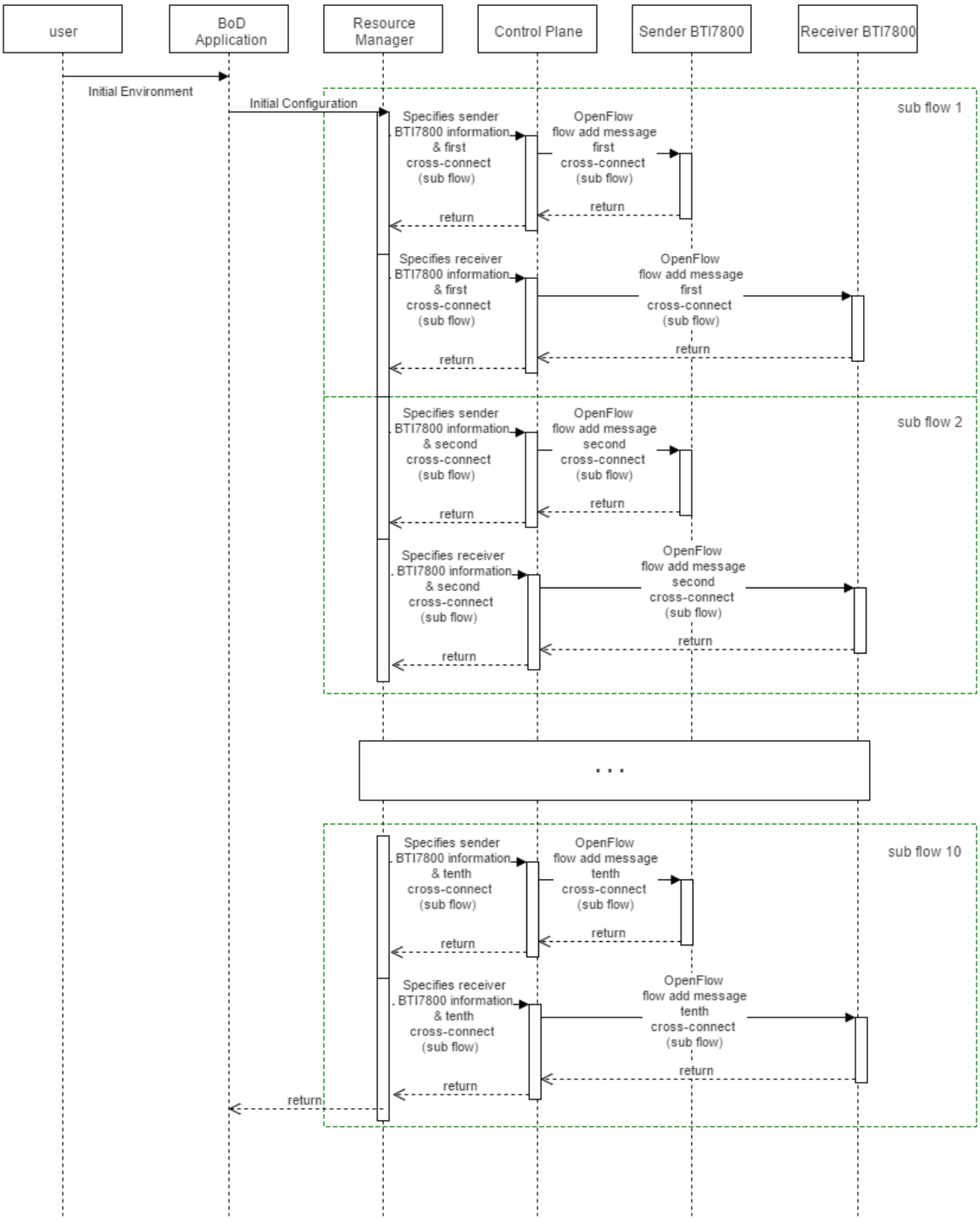


Figure 4.6 Sequence for Initial Configuration Using OpenFlow

Bandwidth Utilization: OpenFlow demonstrate a bandwidth utilization of 53.89%, which is more than the NETCONF results. The bandwidth utilization is calculated from the initial start time t = 1 sec until t= 70 sec.

4.3 High Priority Request (20G)

We considered a set of use cases that demonstrate scenarios comprising of two users at each data center that are managed using a single SDN controller. These use cases are focused on handling, high priority bandwidth request across the users. In all use cases discussed below the user will request for 20G of bandwidth to transmit the high priority traffic. Depending on the use case it can vary if only one of the users or both users are requesting bandwidth for the high priority flows.

4.3.1 High Priority Flow (20 G) Request for User B

Step 1: The environment is brought to the initial state as described in Section 4.2.

Step 2: User B needs 20G of bandwidth for the high priority traffic and user B does not have any low priority traffic. The 20 G of resources assigned to the low priority traffic of user B is utilized by the high priority traffic as shown in Figure 4.7.

Step 3: On completion of high priority traffic demands, the environment returns back to the initial state.

The resources for the low priority traffic is assigned to user B and the same user needs bandwidth for the high priority traffic. Therefore, the high priority traffic will utilize the bandwidth that was assigned to the low priority traffic. There is no change in the cross-connection between 10 G ports and OTU4 link. The user interacts with the BoD application and the bandwidth request for user B's high priority flow is forwarded to the resource manager, as shown in Figure 4.8. The resource manager does not initiate any signaling effort. In this scenario as an aggregate of all flows, there is a bandwidth

requirement of 100 G and all the bandwidth requirements are satisfied, the allocation details are represented in Table 4.2.

User A	User B	Type of flow	Action
40 G (Green)	40 G (Blue)	Base Flow	Serviced
	20 G (Red)	High priority	Serviced

Table 4.2 High Priority Request from User B

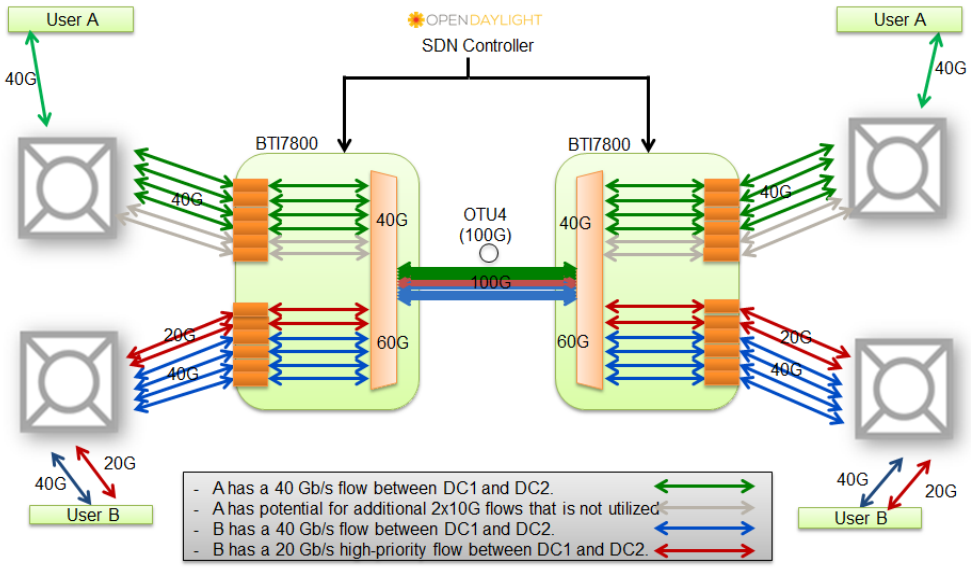


Figure 4.7 High Priority Request from User B

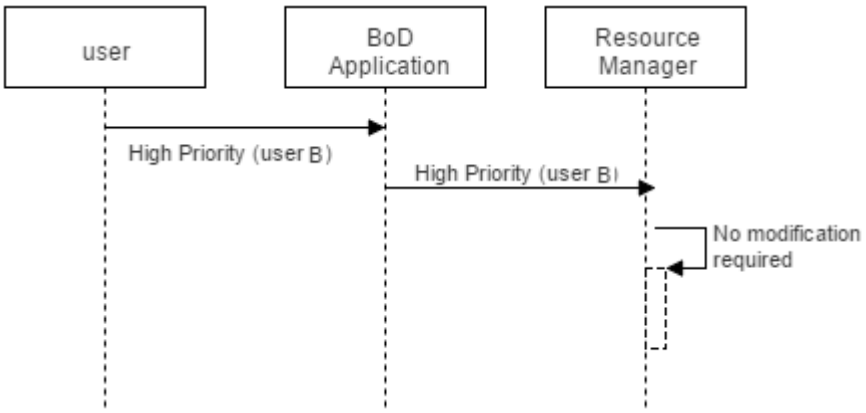


Figure 4.8 Sequence for High Priority Request from User B

4.3.2 High Priority Flow (20 G) Request for User A

4.3.2.1 Case 1: When No Low Priority Flow Exists

User A	User B	Type of flow	Action
40 G	40 G	Base Flow	Serviced

Table 4.3 Bandwidth Allocation for High Priority Request from User A - Case 1

Step 1: Both users (user A and user B) have 40G of bandwidth allocated for the base flows as shown in Table 4.3. There is 20G of unutilized bandwidth available on the link interconnecting the data centers.

Step 2: User A has an additional high priority traffic demand and it needs 20G bandwidth. The controller allocates the 20G of available bandwidth to user A as shown in Figure 4.9.

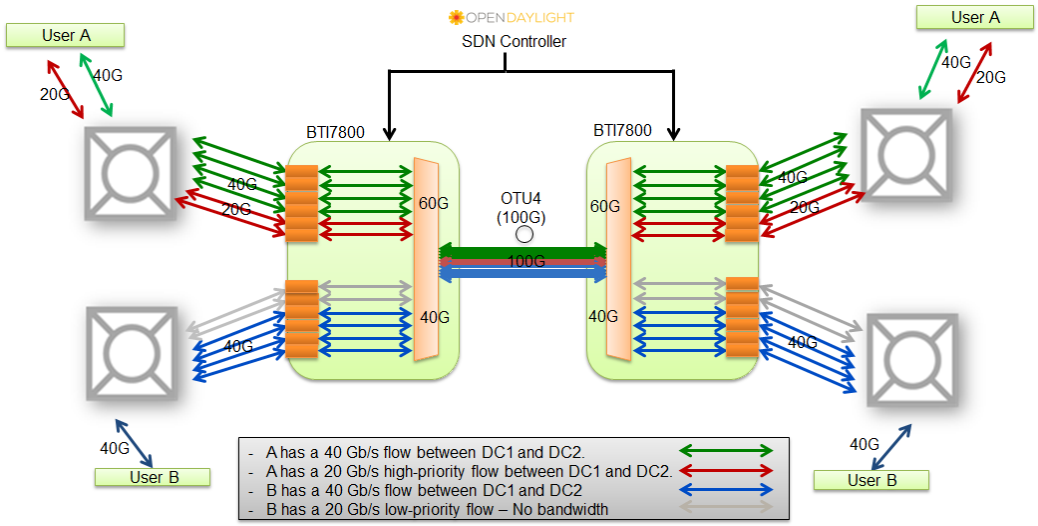


Figure 4.9 High Priority Request from User A - Case 1

Step 3: On completion, user A releases the bandwidth and the system returns to step 1.

For the use case represented in Figure 4.9, user A needs bandwidth for the high priority flow and the controller allocates the available unutilized bandwidth to service user A's requirement. After 8 seconds the application layer signals the controller about the completion of the high priority flow and modifies the connection accordingly, so that the environment returns back to the initial state. The fixed time is considered because it helps us in comparing the performance and the behavior of both protocols, as the environment and the timing remains the same. The detailed working of both protocols as SBIs are discussed below:

Figure 4.10 represents the sequence of steps involved in configuring the high priority request using NETCONF protocol. The user interacts with the BoD application and the resource manager receives the request. The system exists in an initial state with 8 sub flows or cross-connects provisioned. The user has requested for 2 additional sub flows for handling the high priority traffic of user A. The resource manager communicates a bunch of 10 cross-connects information (the existing 8 cross-connects belonging to the initial

environment and the additional two cross-connects) to the sender BTI7800 device with the help of the control plane. On completion, the sender BTI7800 device replies the status. The resource manager, through the control plane, communicates the same information to the receiver BTI7800. The BoD application is aware that the cross-connects are provisioned to handle the high priority requirement. At the end of 8 seconds the BoD application signals the resource manager to return back to the initial configuration. The resource manager, through the control plane, signals a bunch of 8 cross-connects or sub flows belonging to the initial configuration to both BTI7800 devices. As a result, the cross-connects created for handling the high priority traffic of user A is torn down.

Figure 4.11 represents the sequence involved in configuring the high priority request using OpenFlow. The user interacts with the BoD application and the resource manager receives the request. The system exists in an initial state with 8 sub flows or cross-connects provisioned. The user has requested 2 additional sub flows for servicing the high priority traffic of user A. The resource manager communicates the 9th cross-connect or sub flow information to the sender and the receiver BTI7800 devices through the control plane. Once the 9th sub flow is created successfully at both the BTI7800 devices, the 10th sub flow information is communicated. The BoD is aware of the sub flows being established for satisfying the high priority bandwidth. At the end of 8 seconds the BoD application signals the resource manager to return back to initial configuration. The resource manager through the control plane signals the 9th cross-connect or sub flow to be deleted at both BTI7800 devices. The same is continued for deleting sub flow 10.

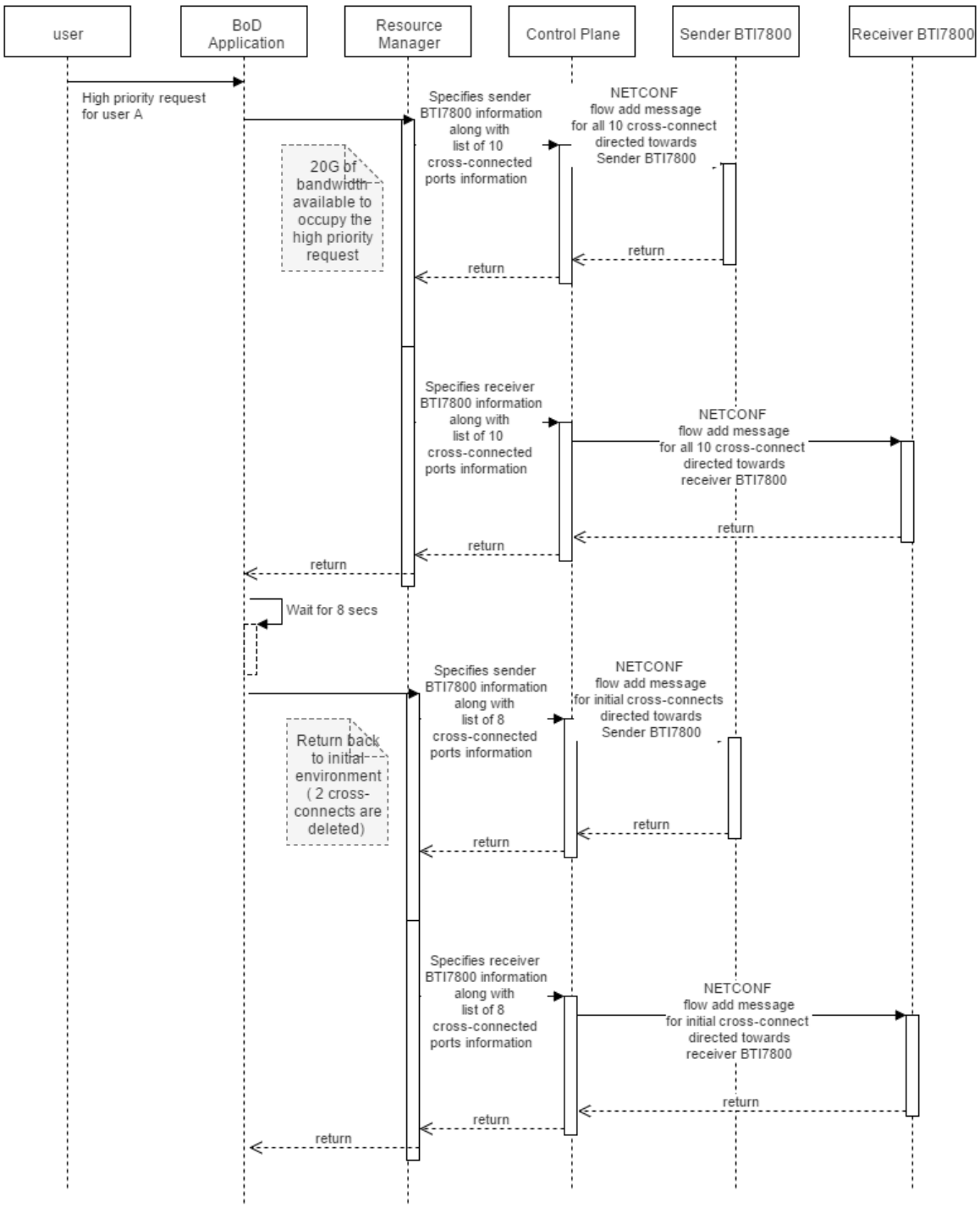


Figure 4.10 Sequence for High Priority Request from User A Using NETCONF – Case 1

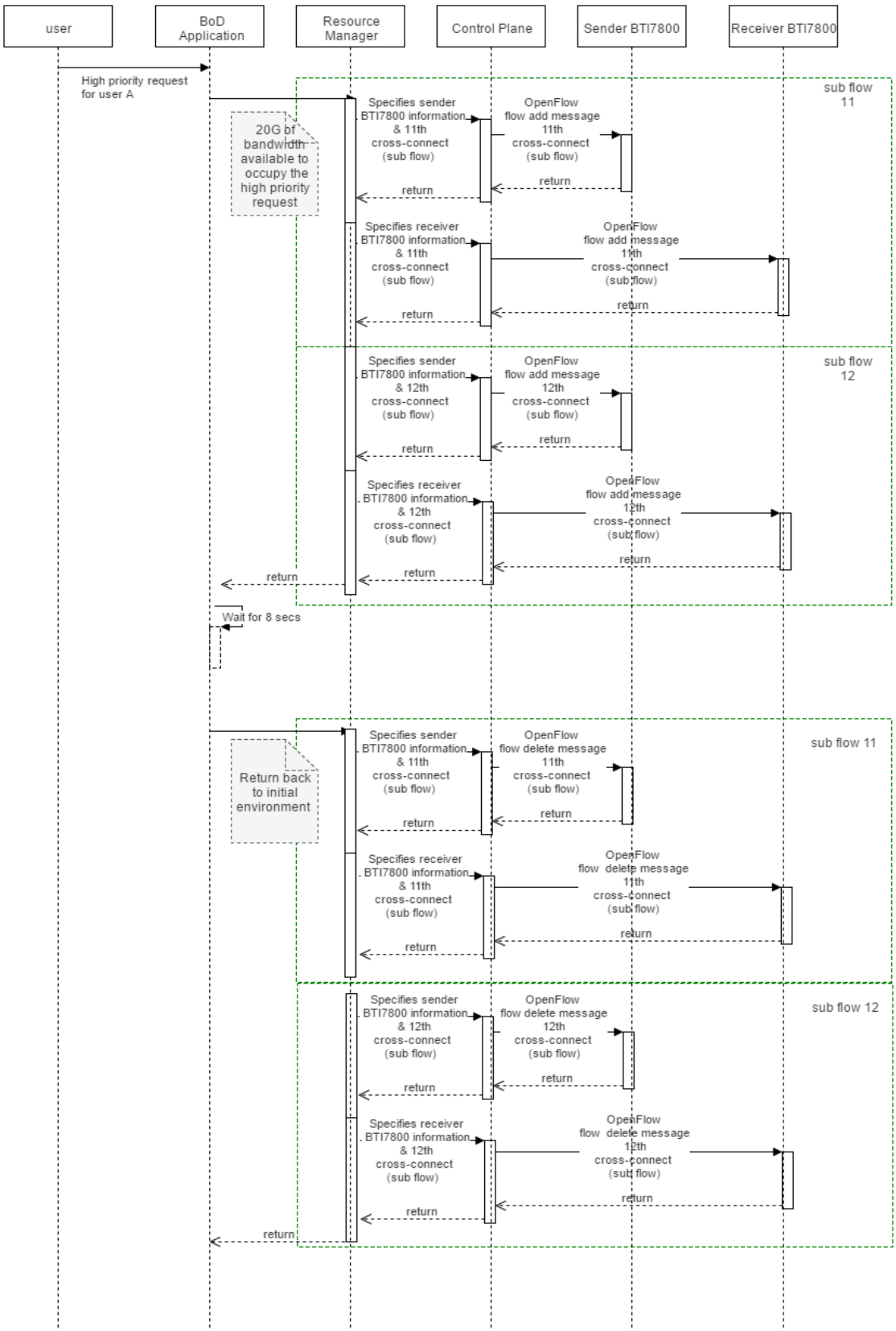


Figure 4.11 Sequence for High Priority Request from User A Using OpenFlow – Case 1

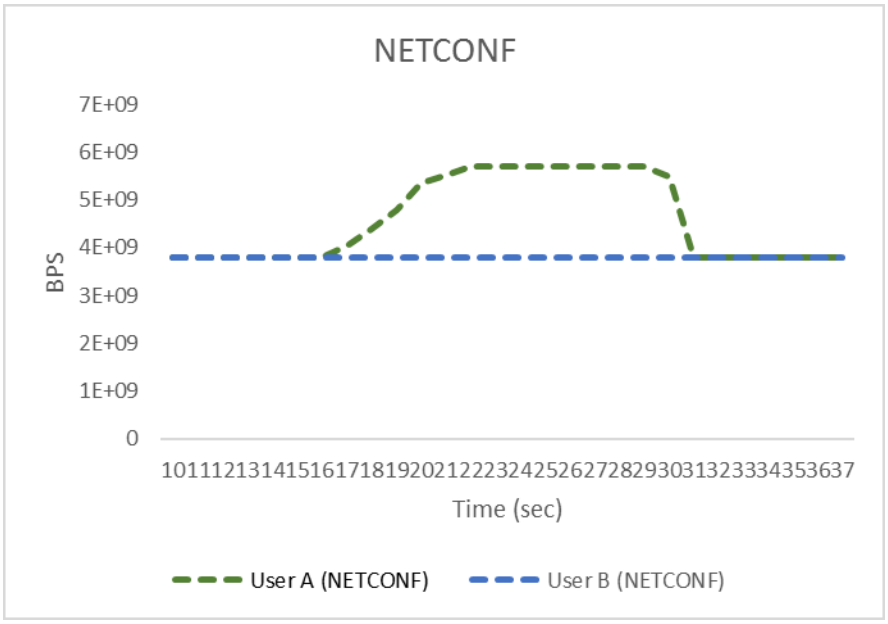


Figure 4.12(a) Provisioning for High Priority Flow Request for User A Using NETCONF –Case 1

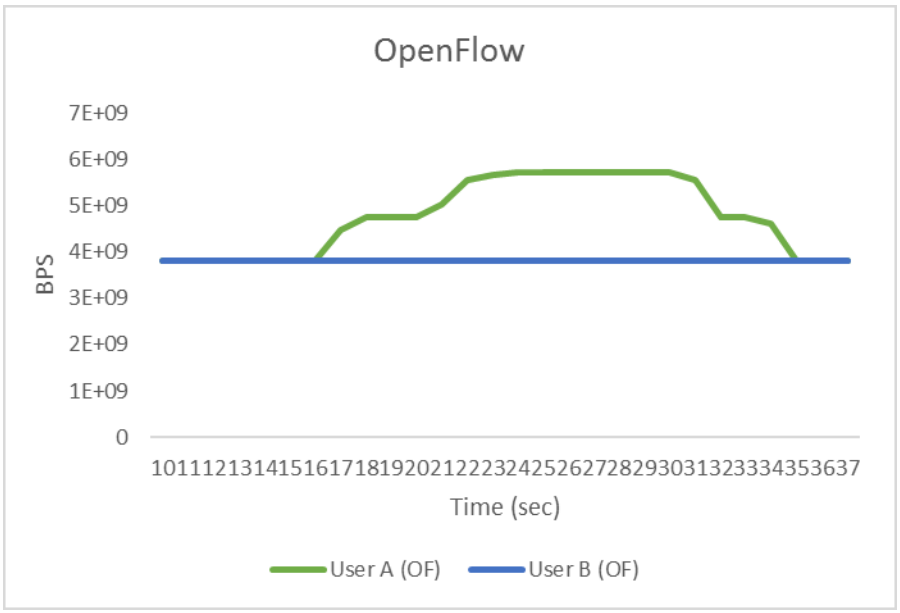


Figure 4.12(b) Provisioning for High Priority Flow Request for User A Using OpenFlow – Case 1

Time: The NETCONF and OpenFlow handling is represented in Figure 4.12 (a) and (b) respectively. The experiment begins at time t = 10 sec and the system exists in the initial state. At t = 10 sec the application layer signals the resource manager about the high priority request. Using the NETCONF protocol, the 9th sub flow is configured at t = 18

sec and the 10th flow is configured at t = 22 sec. From t = 22 sec until t = 30 sec for 8 seconds the high priority traffic requirement is satisfied. Using the OpenFlow protocol, the 9th flow is configured at t = 17 sec and the 10th flow is configured at t = 22 sec. From t = 22 sec until t = 30 sec for 8 seconds the high priority traffic requirement is satisfied. For NETCONF and OpenFlow it takes 12 seconds to process two cross-connects at each end. NETCONF and OpenFlow takes 3 seconds as hardware setup time for each cross-connect. At t = 31 sec the environment implementing the NETCONF protocol communicates the modification for the initial environment and it takes 0.50 second for each end to delete both cross-connects. The OpenFlow communicates the modification for the initial environment at t = 31 sec and at t = 34 sec the environment reaches the initial state and it takes 0.75 second to delete each cross-connect. In the case of OpenFlow each cross-connect to be deleted is a separate message and the device handles each message received and provision the hardware accordingly.

Bandwidth Utilization: The bandwidth utilization for this use case is calculated for the window when high priority flow begins to service until the high priority flow ends. Both protocols have to serve the high priority request for 8 seconds, but the high priority window includes sub flow setup and tear down delays. Both protocols the high priority flows begins at t = 16 sec. NETCONF consumes a window of 16 seconds and the bandwidth utilization is 94.4 %. OpenFlow consumes a window of 20 seconds and the bandwidth utilization is 93.37 %. While the difference is very small, NETCONF results are better compared to OpenFlow.

4.3.2.2 Case 2: When The Lower Priority Flow Exists

Step 1: The environment exists in the initial state as discussed in Section 4.2.

Step 2: User A has high priority traffic and requires additional bandwidth. The low priority traffic of user B owning the resources loses its bandwidth and user A is assigned with the bandwidth to satisfy the high priority traffic demands as shown in Figure 4.13.

Step 3: On completion of the high priority traffic demands, the environment returns back to the initial state. The controller will release the bandwidth occupied by the high priority traffic by user A and assign the flows back to the low priority traffic of user B.

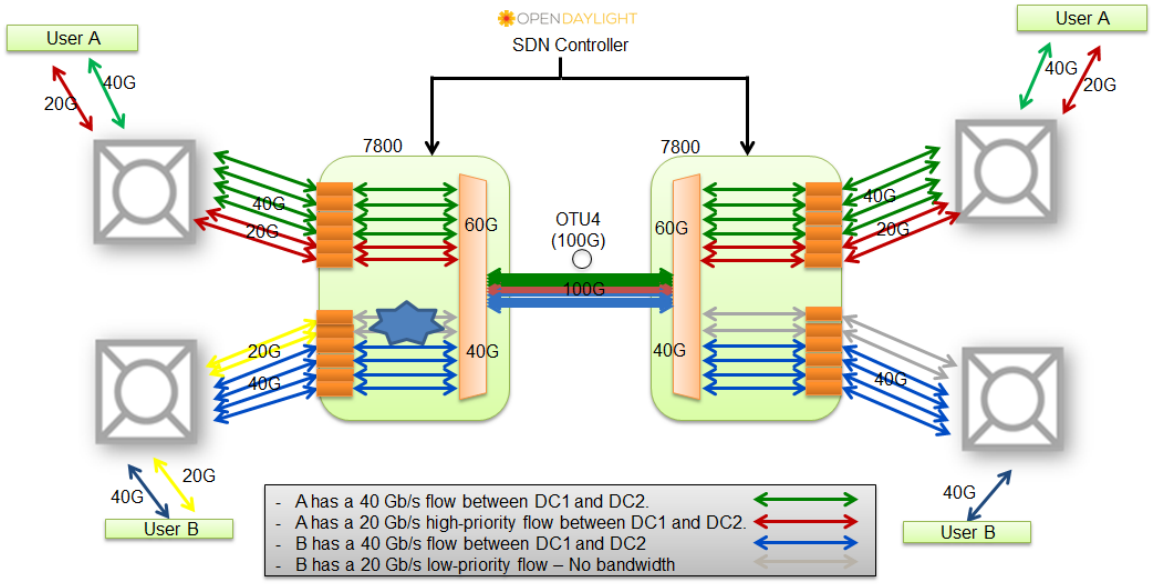


Figure 4.13 High Priority Request from User A - Case 2

User A	User B	Type of flow	Action
40 G	40 G	Base Flow	Serviced
	20 G (preempted)	Low priority	No resources
20 G		High Priority	Serviced

Table 4.4 Bandwidth Allocation for High Priority Request from User A - Case 2

Both users A and B have dedicated bandwidth of 40 G each, and the 20 G of lower priority traffic is assigned to user B. User A has high priority traffic and it demands for

the bandwidth to transmit. The SDN controller, using the implemented SBI will signal the BTI7800 network element to drop the bandwidth pertaining to the low priority traffic and will establish the bandwidth for the high priority traffic belonging to user A for a time period of 8 seconds. The information regarding the flow modification is communicated by the application layer to the control plane. In this scenario, there is a bandwidth request for 120 G and having a limitation of 100 G link between data centers, thus controller decides to allocate bandwidth for the flows based on the nature of the traffic as represented in Table 4.4. The sequence explaining the detailed working of both protocols as SBIs are discussed in Appendix A.

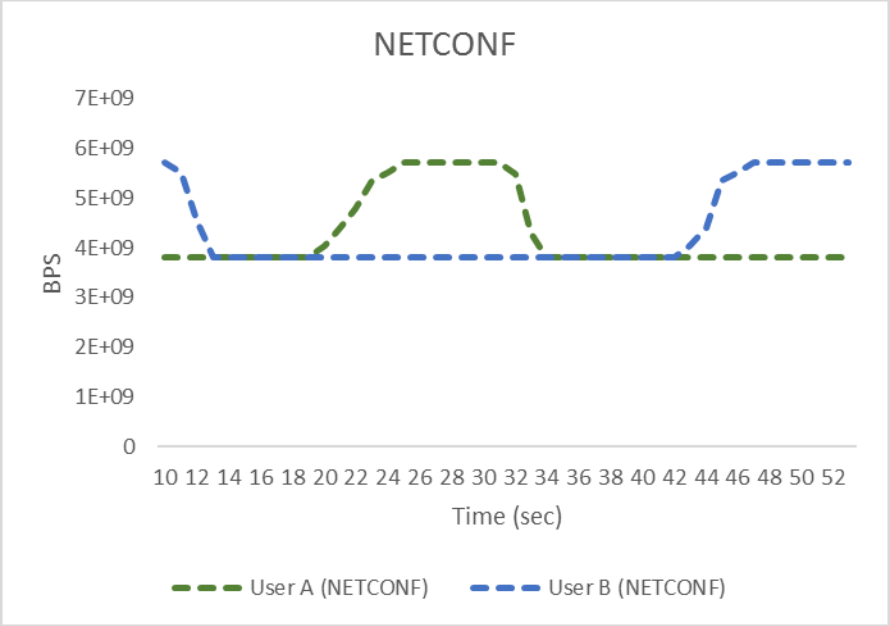


Figure 4.14(a) Provisioning for High Priority Flow Request from User A Using NETCONF – Case 2

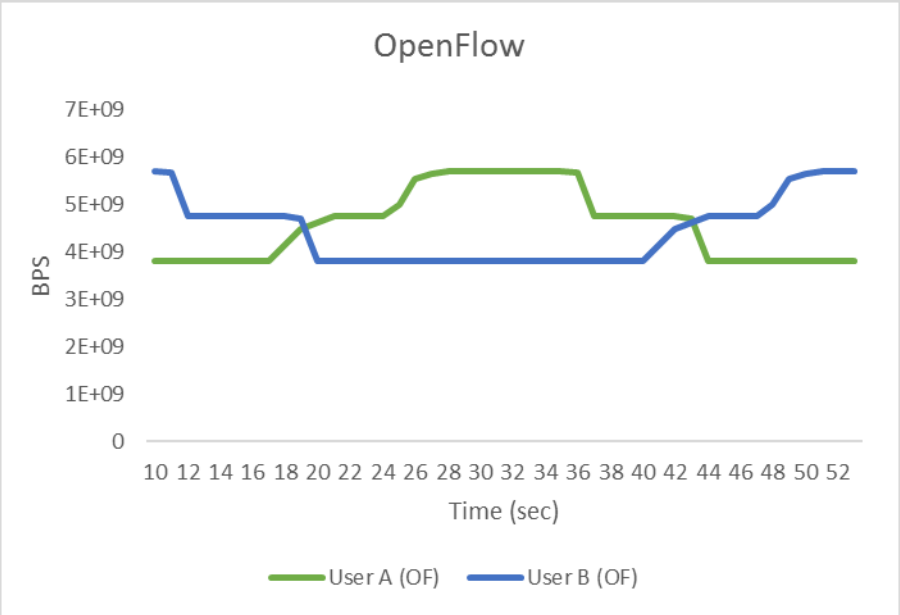


Figure 4.14(b) Provisioning for High Priority Flow Request from User A Using OpenFlow – Case 2

Figure 4.14 (a) and (b) represents the flow handling of both protocols, NETCONF and OpenFlow respectively in this environment.

Time: The experiments begins at time t = 10 sec and the environment exists in the initial state. At time t = 10 sec the application layer signals the resource manager regarding the bandwidth request for the high priority flows from user A. As discussed before, two cross-connects need to be torn down and two new cross-connects are added. In NETCONF and OpenFlow, at t = 23 sec and t = 25 sec, all two sub flows are provisioned. It takes 13 seconds and 15 seconds for NETCONF and OpenFlow respectively to provision the high priority request. NETCONF takes 6.5 seconds to provision one data center, it takes 0.5 second to delete both the cross-connects and 3 seconds to add a cross-connect. In OpenFlow it takes 3 seconds to add a cross-connect, and 0.75 second to delete a single cross-connect. Addition of cross-connects takes longer hardware setup time because an optical tunnel between the two interconnecting ports is created. During the deletion of cross-connect the entry is removed from the cdb flow table and then ports are

shutdown, therefore deletion of cross-connect(s) consume less hardware setup time. The high priority traffic is serviced for 8 seconds. Similarly, NETCONF and OpenFlow environment return back to initial state by dropping the two high priority sub flows and adding the two low priority sub flows.

The OpenFlow protocol takes longer time compared to the NETCONF is because each sub flow modification is processed separately. The resource manager communicates which particular sub flow needs to be modified one after another. Unlike the NETCONF protocol, the OpenFlow protocol is not aware of the cross-connect modification by the protocol itself, the resource manager needs to communicate each flow modification separately. But in NETCONF the resource manager communicates the set of cross-connects needs to be provisioned and the NETCONF agent with the BTI7800 identifies the cross-connects that are modified by comparing with the existing cross-connects provisioned.

In Section 3.4.4 we disused that the implemented OpenVSwitch algorithm posts the cross-connects modification as request message within the queue and the confd kernel module process the request one after another. The cross-connects that needs to be deleted and added are posted as both ends one after another. There is a possibility that the cross-connect modification request is posted on the queue before the confd completes processing of the predecessor modification request.

Bandwidth Utilization: The NETCONF scenario has a bandwidth utilization of 92.78 % and the OpenFlow scenario bandwidth utilization of 95.85%. The OpenFlow interface deletes a sub flow and adds a sub flow and it repeats the procedure for the number of sub flows to be modified, that is why we see a ladder in Figure 4.9 (b). The bandwidth

utilization for both protocols are calculated from the start of experiment t = 10 and until t = 53 sec when both scenarios remain in the steady state for a few seconds.

4.3.3 High Priority Flow (20 G) Request from Both Users A and B (Mutual Sharing)

Step 1: The environment exists in the initial state as discussed in Section 4.2.

Step 2: Both user A and user B have high priority traffic, and they request for the bandwidth. The controller preempts the bandwidth assigned to the low priority traffic and it assigns the resources shared among both high priority requests as shown in Figure 4.15.

User A	User B	Type of flow	Action
40 G	40 G	Base Flow	Serviced
	20 G	High priority	50% of requirement serviced (10 G)
20 G		High Priority	50% of requirement serviced (10 G)

Table 4.5 Mutual Sharing Allocation

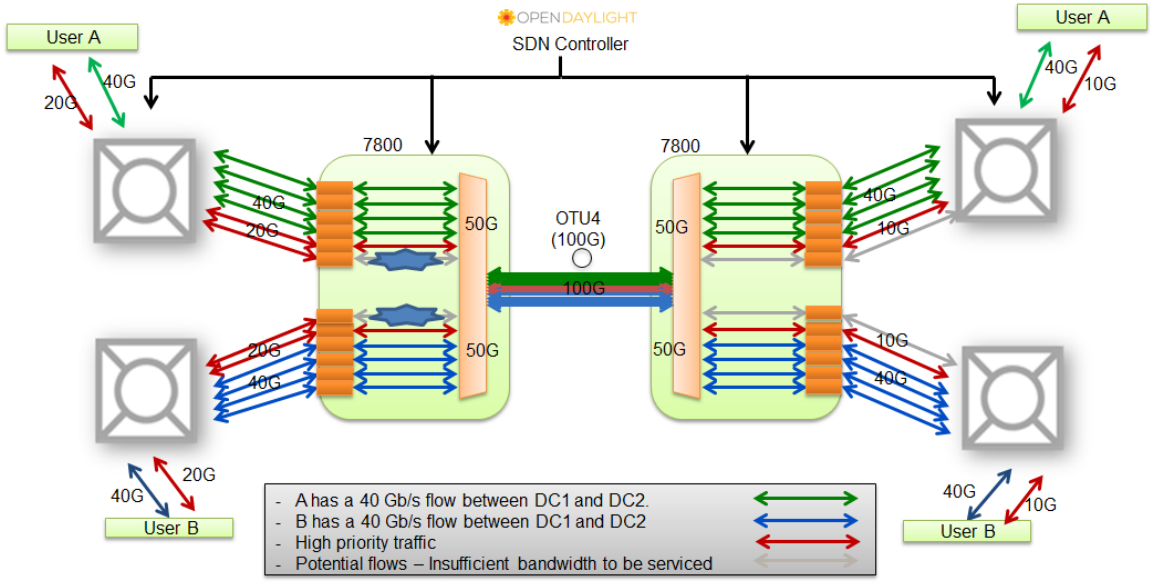


Figure 4.15 Mutual Sharing

Step 3: When both users complete their high priority bandwidth requirement, the controller performs the required signaling to modify the flows so that user B low priority traffic can utilize the sparse bandwidth.

In this scenario, both users request for 20 G bandwidth to satisfy the high priority traffic demands, but there exists only 20 G of bandwidth that can be preempted from lower priority traffic. The controller decides to share the available bandwidth by assigning each user with 10 G of bandwidth allocated to the high priority traffic as represented in Table 4.5, where 50 percent bandwidth requirement of both users will be satisfied. There is no change in the flow that is assigned to user B's high priority traffic, as the link assigned to the low priority traffic of user B will be utilized by high priority flow. User A's high priority traffic is allocated bandwidth by preempting a single low priority flow from user B. If we closely look there is only one flow modification as shown in Figure 4.15. The sequence explaining the detailed working of both protocols as SBIs are discussed in Appendix B.

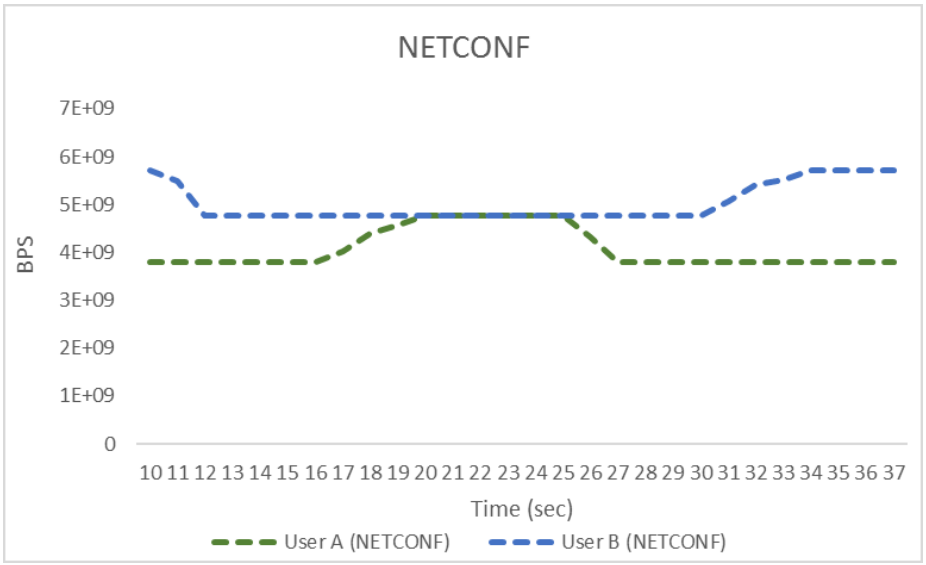


Figure 4.16(a) Provisioning for High Priority Flow Request from User A and B Using NETCONF – Mutual Sharing

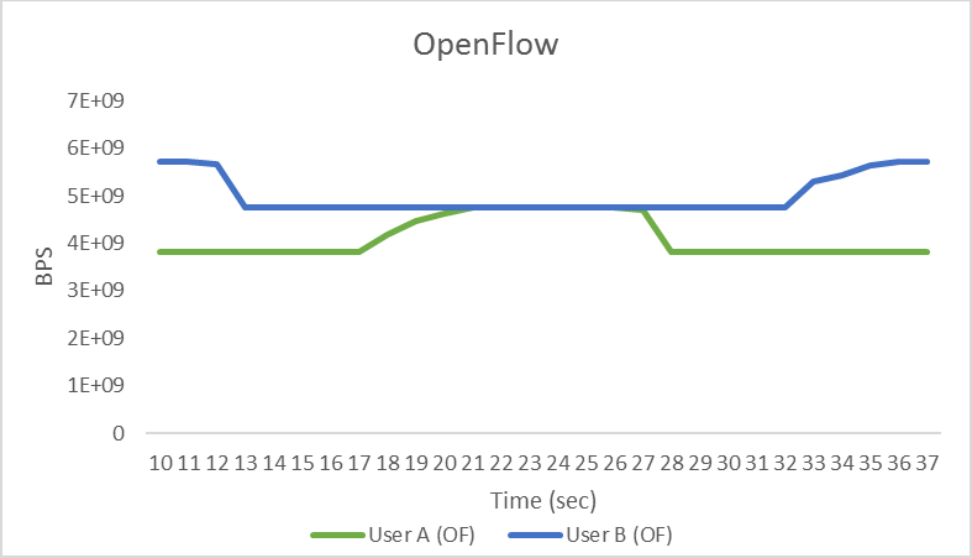


Figure 4.16(b) Provisioning for High Priority Flow Request from User A and B Using OpenFlow – Mutual Sharing

Time: Figure 4.16 (a) and (b) shows the bandwidth utilization of NETCONF and OpenFlow. Both the environment implementing NETCONF and OpenFlow as the SBIs, the system exists in initial state at t = 10 sec. At t = 10 sec the application layer informs the control plane about the high priority flow request. As discussed, one sub flow needs to be deleted and one new cross-connect needs to added. NETCONF takes 7 seconds and OpenFlow takes 8 seconds to provision the high priority flow. In the case of NETCONF it takes 0.5 second to delete a cross-connect and 3 second to add a cross-connect. In the case of OpenFlow it takes 0.75 second to delete a cross-connect and 3 second to add a cross-connect. After high priority request is served for 8 seconds they return back to initial state. Similarly, NETCONF and OpenFlow environment return back to initial state by dropping the one high priority sub flows and adding the one low priority sub flows. The NETCONF and OpenFlow takes 7 seconds and 8 seconds respectively to attain the initial state.

Bandwidth Utilization: NETCONF exhibit 95.62 % utilization and OpenFlow exhibit 95.7 % of utilization. We notice that, as the number of flows to be modified decreases both the protocols exhibit high bandwidth utilization. The bandwidth utilization is calculated from t = 10 sec the initial start of the experiment until t = 37 sec where the system attains a steady state in the initial configurations.

Chapter 5: Stress and Load Testing

In the previous chapter we have discussed about the test environment, evaluation metrics and a few high priority use cases. In this chapter we present a few bandwidth request use cases, where the bandwidth request made from the user(s) is large depending on the scenario. The use cases help us in comparing both protocols when a large number of sub-flows or cross-connects are modified. In this chapter we also present the performance of both protocols in the stress test scenario, when multiple applications try to modify the same data plane device though the control plane simultaneously at a same time. The use cases discussed in this chapter are classified as listed below.

- Bandwidth Request
  - Elephant Request (40 G)
  - Emergency Request (100 G)
- Stress Testing

5.1 Elephant Request (40G)

The elephant flow use case is mainly focused on demonstrating and evaluating the implemented protocols as SBIs on the BTI7800 devices and their responsiveness when a large number of sub flows are modified.

Step 1: The environment exists in the initial state as discussed in Section 4.2.

Step 2: User A requests bandwidth for the elephant flow. The SDN controller drops the low priority flow and a few base sub flows of user B to accommodate the elephant flow request of user A as shown in Figure 5.1.

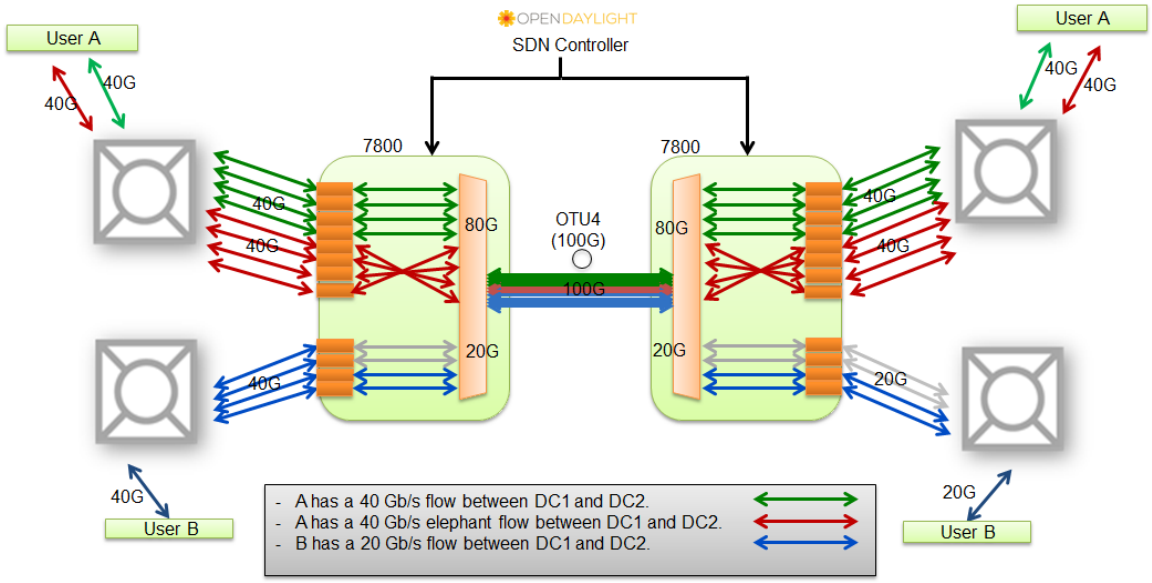


Figure 5.1 Elephant Flow Request by User A

Step 3: Once the requirement is satisfied, the system returns back to the initial configuration. If another user (user B) requests bandwidth for the elephant flows while the elephant flow request of user A is being serviced, user B needs to wait until user A completes its bandwidth requirement.

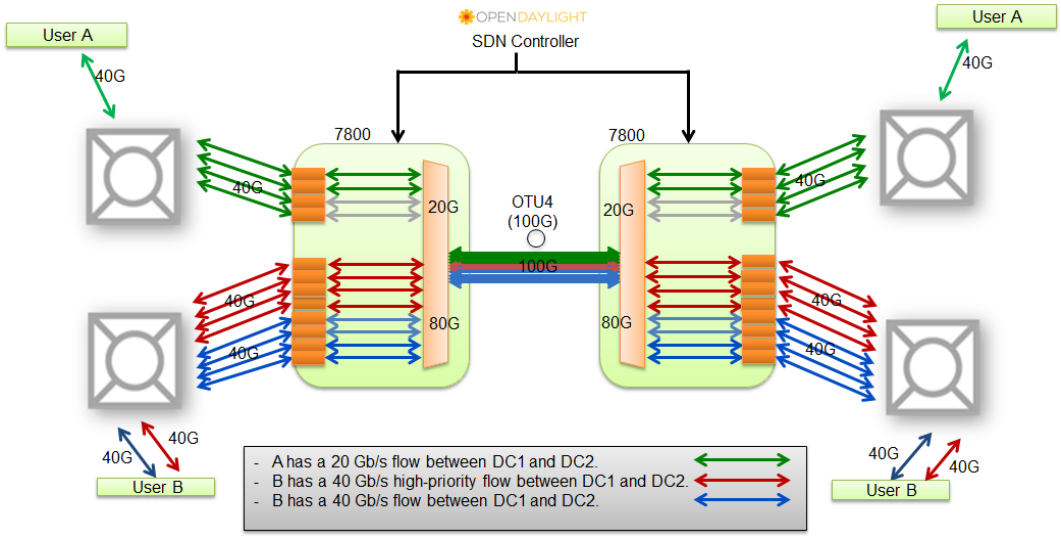


Figure 5.2 Elephant Flow Request by User B

Step 4: Figure 5.2 represents the sub flows provisioned, if user B requests bandwidth for the elephant flow.

Step 5: If there is no further elephant flow request, the system returns back to the initial state.

Figure 5.3 and Figure 5.4 represent the steps involved in configuring the elephant request using NETCONF and OpenFlow respectively. The user interacts with the BoD application and the resource manager receives the request. The system exists in an initial state with 10 sub flows or cross-connects provisioned. As per the use case discussed above, user A requests bandwidth for the elephant flow, and as a result four cross-connects are torn down and added. The elephant flow requirement of user A is served for 8 seconds. At the end of 8 seconds, the resource manager receives a signal to provision the environment to serve the elephant flow for user B. The existing six cross-connects are torn down and six new cross-connects are added to serve the elephant flow request. After 8 seconds, when the system returns to the initial environment, two cross-connects have to be torn down and two new cross-connects have to be added.

In NETCONF the set of cross-connects needed to be provisioned are communicated as a single request each time. The NETCONF agent on the BTI7800 device figures out the modification and will tear down and add cross-connects accordingly. In OpenFlow the resource manager needs to identify the set of cross-connects needed to be changed and rolls out the change for each cross-connect in a sequential order. OpenVSwitch, as it receives a sub flow modification request from the controller, validates the received information and posts the cross-connect modification into the message queue. Confd handles each modification one after another as described in Section 3.4.4.

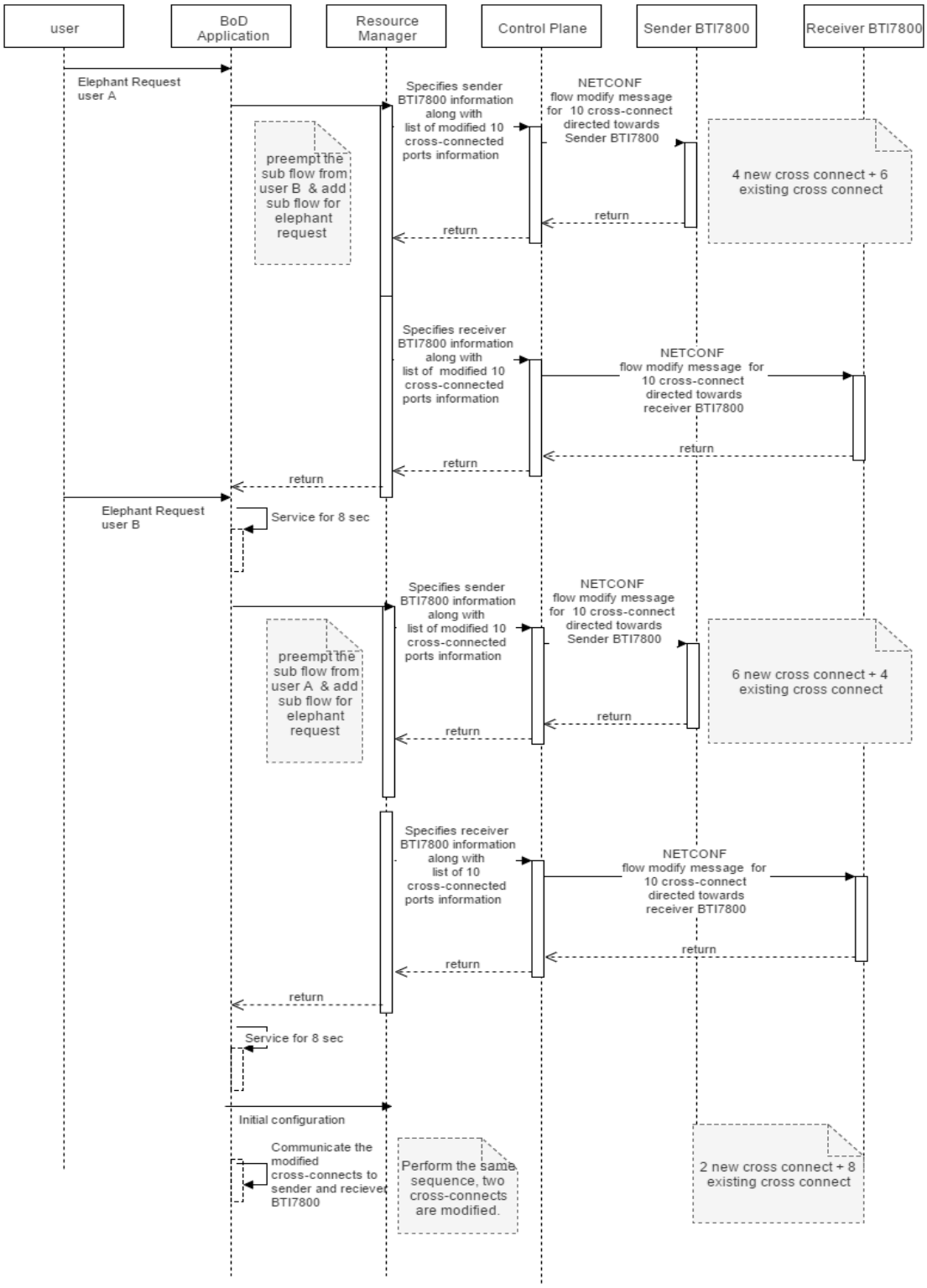


Figure 5.3 Sequence for Elephant Flow Request Using NETCONF

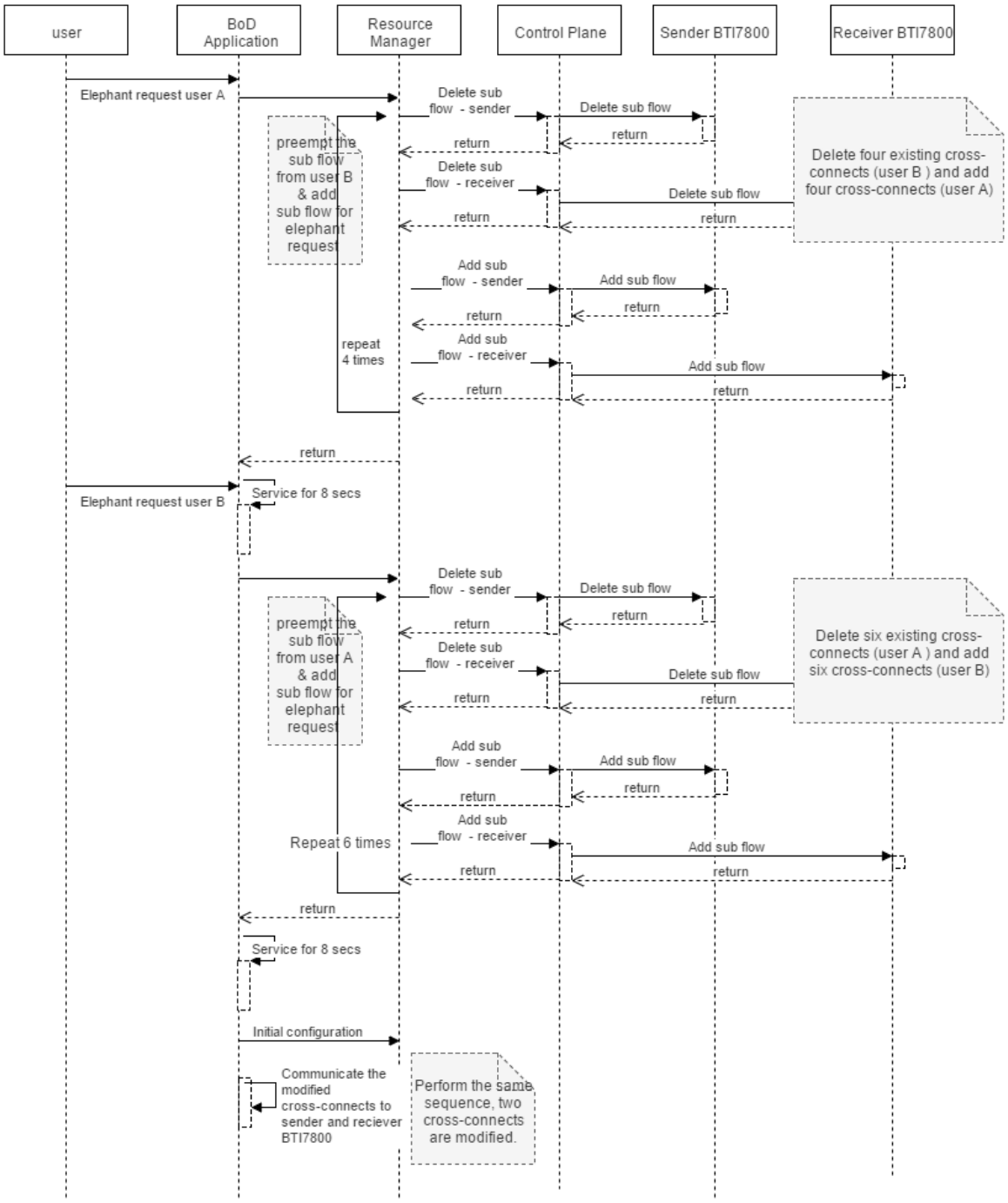


Figure 5.4 Sequence for Elephant Flow Request Using OpenFlow

Time: The NETCONF and OpenFlow handling is represented in Figure 5.5 (a) and (b) respectively. The experiment begins at time t = 10 sec and the system exists in the initial state. At t = 10 sec the application layer signals the resource manager about the elephant request. As discussed before, four cross-connects need to be torn down and four new cross-connects need to be added in both data centers. In NETCONF and OpenFlow, at t = 36 sec and t = 40 sec, all four sub flows are provisioned. It takes 26 seconds and 30 seconds for NETCONF and OpenFlow respectively to provision the elephant request of user A. NETCONF takes 13 seconds to provision one data center and it takes 3 seconds to add a cross-connect and one second for deleting all cross-connects per data center. As number of cross-connects to be deleted in NETCONF increases the time taken to delete all the cross-connect also increases. OpenFlow takes 3 seconds to add a cross-connect and 0.75 second to delete a cross-connect. After 8 seconds the elephant flow requirement of user B is communicated, NETCONF and OpenFlow communicate the information at t = 45 sec and t = 49. The total number of sub flows modified are six, which includes the deletion of six cross-connects and the addition of six cross-connects. NETCONF and OpenFlow take 38 seconds and 45 seconds respectively, at t = 83 sec and t = 94 sec the environment is provisioned for servicing the elephant request of user B. As discussed earlier, NETCONF takes 3 seconds for addition of a cross-connect and deletion of all cross-connects takes 1 sec, which is 18 seconds for addition of 6 cross-connects and 1 second for deletion of all cross-connects per data center. OpenFlow takes 3 seconds to add a cross-connect and 0.75 second to delete a cross-connect, which is 18 seconds for addition of 6 cross-connects and 4.5 seconds for deletion of 6 cross-connects per data center. The scenario waits for 8 seconds before returning to the initial state. NETCONF

attains the initial state at t = 104 sec and it takes 13 seconds. OpenFlow attains the initial state at t = 117 and it takes 15 seconds. The total number of sub flows that need to be modified are the addition and deletion of two cross-connects.

The number of sub flows to be deleted does not impact the performance of NETCONF protocol, it always takes less than 2 seconds to delete any number of sub flows. In OpenFlow, each sub flow that needs to be deleted is accompanied by a message which involves processing time and therefore it takes much longer time.

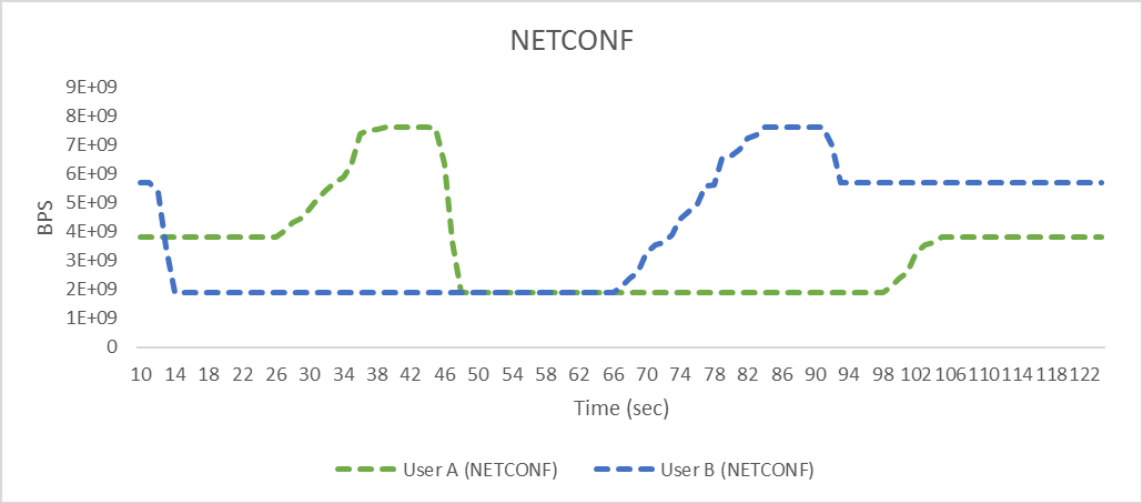


Figure 5.5(a) Elephant Flow Request Using NETCONF

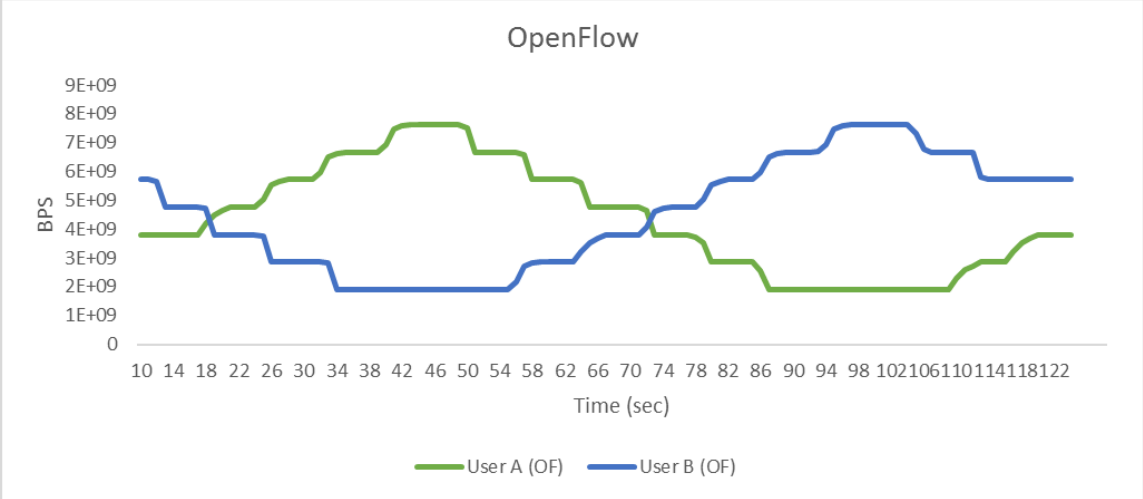


Figure 5.5(b) Elephant Flow Request Using OpenFlow

Bandwidth Utilization: The bandwidth utilization is calculated for the time period from t = 10 sec until t= 124 sec. NETCONF exhibits bandwidth utilization of 77.83 % and OpenFlow represents a bandwidth utilization of 92.92%. The fact that OpenFlow handles sub flow modifications as individual each sub flow the resources remains occupied. On the other hand, the same fact acts as a drawback: as more sub flows needs to be modified, OpenFlow is much slower and more messages are communicated between the controller and the BTI7800 devices. NETCONF is faster in handling the request but the resource remains unutilized for a time period proportional to the number of flows that need to be configured. It is because NETCONF operates on a set of cross-connects.

5.2 Emergency Request (100G)

Similar to the elephant flow request, the emergency request use case introduces a complete change in the flow table based on the user calling the emergency service. This use case helps in evaluating both protocols when there is a complete change in the cross-connects.

Step 1: The environment exists in the initial state as discussed in Section 4.2.

Step 2: If user A calls the emergency service, the SDN controller will de-provision all the sub flows that exist with user B and will provision those to fulfill the emergency request of user A. The base flows of user A will be used to satisfy the emergency requirement as shown in Figure 5.6.

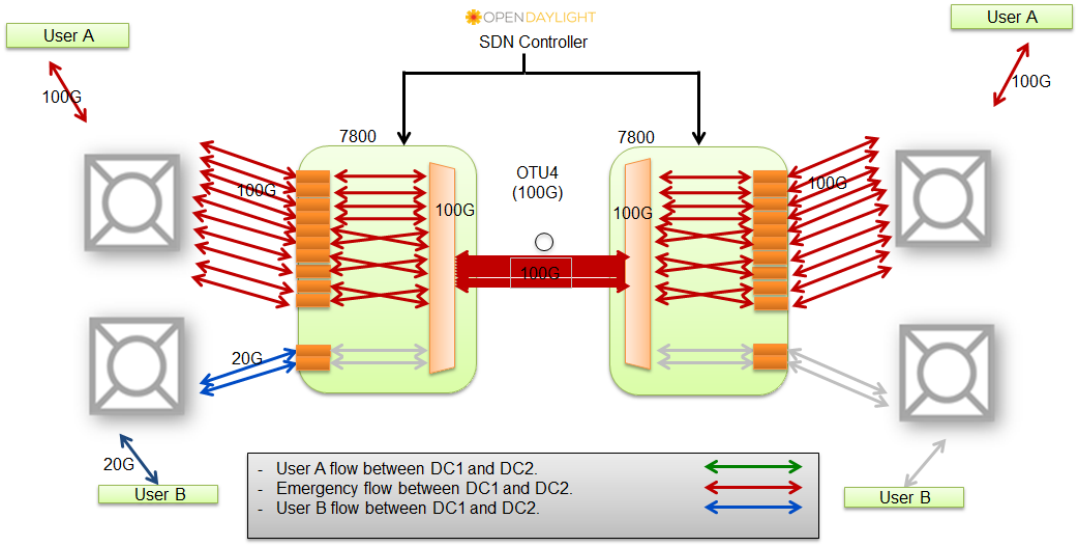


Figure 5.6 Emergency Flow Request by User A

Step 3: Once the emergency requirement is satisfied, the system returns back to the initial state, if there no emergency request from the other user (user B).

Step 4: If user B has an emergency request, the SDN controller provisions the sub flows to satisfy the emergency traffic requirement as shown in Figure 5.7.

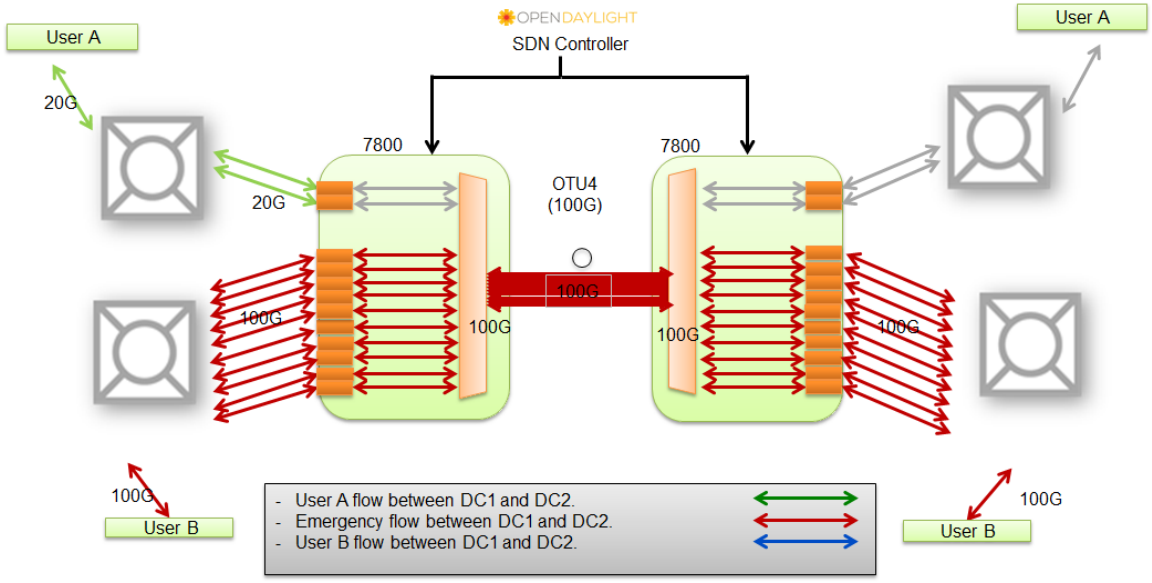


Figure 5.7 Emergency Flow Request by User B

Step 5: The system returns to the initial state if there no any other request. The sequence explaining the detailed working of both protocols as SBIs are discussed in Appendix C.

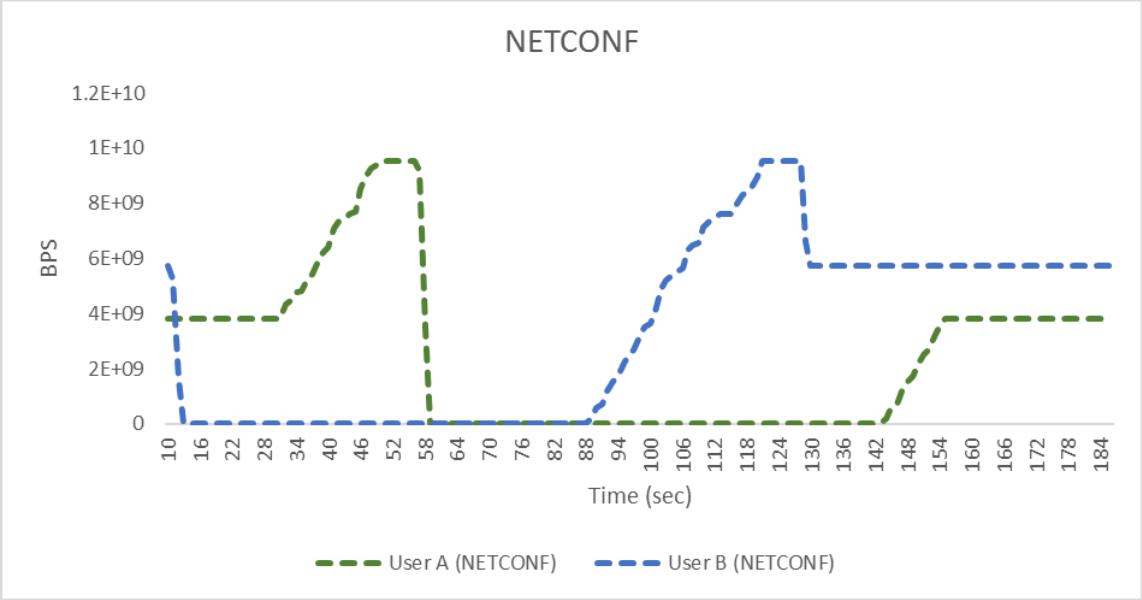


Figure 5.8(a) Emergency Flow Request Using NETCONF

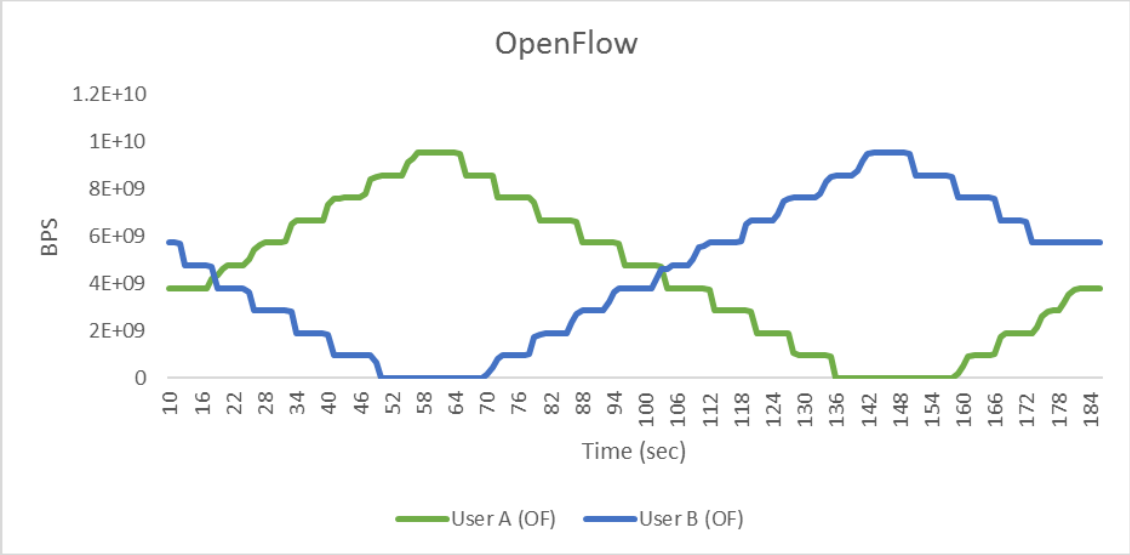


Figure 5.8(b) Emergency Flow Request Using OpenFlow

Time: The NETCONF and OpenFlow handling is represented in Figure 5.8 (a) and (b) respectively. The experiment begins at time t = 10 sec and the system exists in the initial

state. At t = 10 sec the application layer signals the resource manager about the emergency request. As discussed before six cross-connects needs to be torn down and six new cross-connects needs to be added. The environment implementing the NETCONF and OpenFlow, at t = 48 sec and t = 55 sec all the six sub flows are provisioned. It takes 38 seconds and 45 seconds for NETCONF and OpenFlow respectively to provision the emergency request of user A. After 8 seconds the emergency flow requirement of user B is communicated, the NETCONF and OpenFlow communicates the information at t = 55 sec and t = 65. The total number of sub flows modified is ten. The NETCONF and OpenFlow takes 65 seconds and 76 seconds respectively, at t = 120 sec and t = 140 sec the environment is provisioned for the emergency request of user B. The environment waits for 8 seconds before returning to the initial state. The NETCONF attains the initial state at t = 154 sec and it takes 26 seconds. The OpenFlow attains the initial state at t = 179 sec and it takes 30 seconds. The total number of sub flows needs to be modified are the addition and deletion of four cross-connects.

The deletion of sub flows in NETCONF does not cause much overhead, but in OpenFlow it does cause overhead because it is accompanied by individual message for each cross-connect needs to be deleted. The addition of ten flows in satisfying the emergency requirement of user B is similar to the initial environment discussed in Section 4.2. In the initial environment scenario does not involve tearing down of cross-connects. In the initial environment the NETCONF and OpenFlow take 63 seconds and 61 seconds respectively. In the emergency use case the NETCONF and OpenFlow takes 65 seconds and 76 seconds respectively. The difference between both protocols, NETCONF and OpenFlow is 2 seconds and 15 seconds. The large time difference is experienced due to

processing of each sub flow deletion individually. In an emergency situation when ten flows need to be provisioned OpenFlow takes 3 seconds for addition of a cross-connect and .75 second to delete a cross-connect, that involves 20 cross-connects to be added and deleted across both data center. NETCONF takes 31.5 seconds to add 10 cross-connects per data center and 1 second to delete all the existing 10 cross-connects.

Bandwidth Utilization: The bandwidth utilization is calculated for the time period from t = 10 sec until t= 186 sec. NETCONF exhibits bandwidth utilization of 59.55 % and OpenFlow represents a bandwidth utilization of 92.92%. The two things to be noted, the fact that OpenFlow handles the traffic demands by modifying the sub flows one after another, the resources remain occupied. On the other hand, the same acts as a drawback as more number of sub flows needs to be modified, it is much slower and more number of messages are communicated between the controller and the BTI7800 devices. The NETCONF is faster is handling the request but the resource remains unutilized for a time period proportionate to the number of flows needs to be configured. It's because NETCONF operates on a set of cross-connects and does not reference individual cross-connects.

5.3 Simultaneous Multiple Application Access – Stress Testing

In order to analyze the working of both implemented protocols, we considered two different applications that try to configure a particular device using the SDN infrastructure. Both applications," A" and "B", will try to modify a device at the same time. As discussed earlier in Section 2.3, the operational data store saves the operational

state of the device and the config data store temporarily saves the changes requested by the application. The behavior of both protocols are discussed below.

5.3.1 NETCONF

NETCONF operates based on the YANG model and its current operational value. When an application "A" issues a modification request, the config data store within the ODL fetches the operational state along with the requested modifications and saves the information in the config data store (config A). Before the requested modification from application "A" are successfully committed to the device, the application "B" request for modification and the information is stored in a different instance of the config data store (config B) as represented in Figure 5.9. Now there will be one operational data store, and two different instances of the config data store for a device.

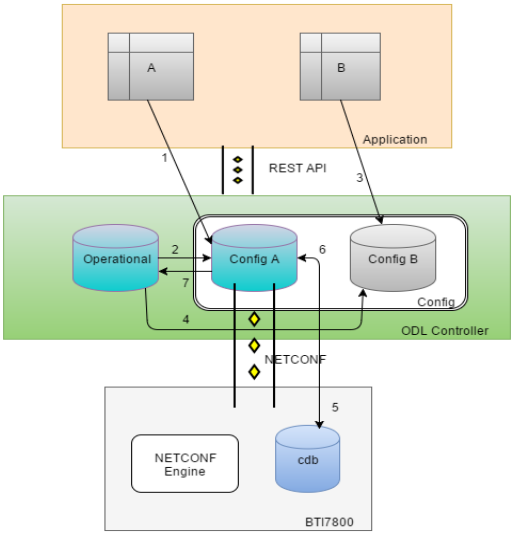


Figure 5.9 Simultaneous Different Application Access - NETCONF

Let us consider that the application "A" modification was received first by the control plane. The controller processes the config A state using NETCONF. If the changes are successfully committed, the device returns NETCONF <edit-config> operation success.

The operational data store is updated and the config A data store is deleted. The application “B” has requested for modification and the information is stored in the config store (config B). Now the ODL NETCONF connector tries to process the config B state, but since the operational state of config B is being modified the NETCONF connector cannot process the request. This causes the ODL controller to throw a NETCONF connector exception error. It is a drawback that the ODL controller cannot handle simultaneous modification of the same device. Moreover, if the NETCONF connector throws exception, it can be solved by restarting the ODL controller.

5.3.2 OpenFlow

The architecture and OpenFlow implementation on BTI7800 is discussed in Section 3.4. We deal with a similar environment using the OpenFlow interface as shown in Figure 5.10, but we observe a different behavior. Unlike NETCONF, as discussed above, OpenFlow does not operate on a data model and it references the information as follows. The SDN controller translates the information as OpenFlow flow modification messages as discussed in Section 3.4.2. When a new flow rule is added or modified, the config data store stores the information. Application A and B requests for the modification at the same time. The modification requests from both applications are received by the controller and stored in the separate config store. Both flow modifications are communicated at the same time to the BTI7800 device. The BTI7800 receives two modification requests at the same time. The implemented OpenVSwitch cannot handle two flow modification messages from the controller at a same time, therefore the BTI7800 enters into a fault mode.

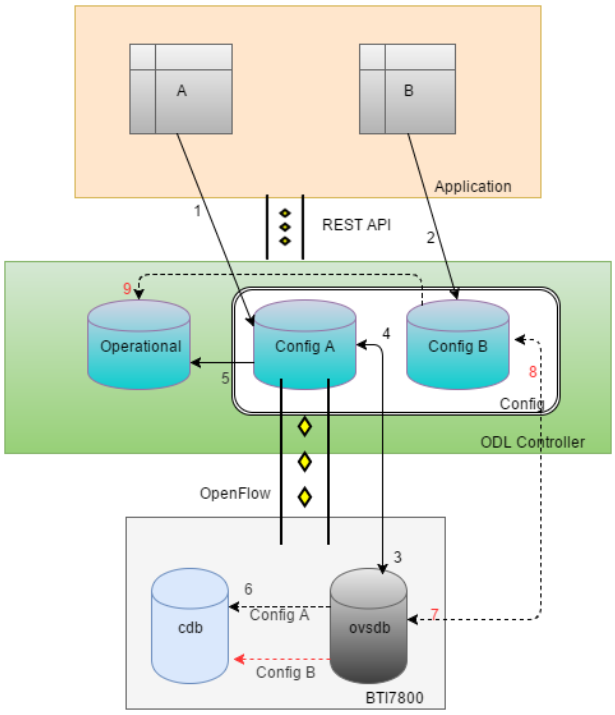


Figure 5.10 Simultaneous Different Application Access – OpenFlow

5.4 Control Messages Between ODL and BTI7800

The ODL controller communicates flow modification control messages to the both BTI7800 devices to manage the BoD across the link interconnecting them. We observed both protocols and from the sequence diagrams of all use cases it is more clear that a number of control message communicated between the ODL controller and BTI7800 devices varies based on the protocol and the number of cross-connects modified.

NETCONF: The cross-connects need to be provisioned are specified as a set of cross-connects in a single NETCONF flow modification message. Immaterial of any number of cross-connects to be modified, the ODL controller communicates one message per data center aggregating the information about all the cross-connects need to provisioned. If 10 cross-connects need to be provisioned the fragmented message of size is 1448 + 1164 bytes are communicated. Our experiment involves two BTI7800 devices, control message

of 2612 bytes is communicated to the both BTI7800 devices. The response from a BTI7800 device is 52 bytes. Depending on the number of cross-connects communicated the control message size varies, and the detailed information is discussed in Appendix D.

OpenFlow: The OpenFlow communicates the flow modification messages depending on number of cross connects need to be modified. Every cross-connect addition and deletion is accompanied by a OpenFlow modification message. The cross-connects need to be modified cannot be aggregated as a single message. The size of the OpenFlow flow modification message from the controller to the BTI7800 device is 206 bytes and the response is 66 bytes. By default, the packet-in OpenFlow modification message is 128 bytes and if match action is configured, the message size is 206 bytes [3]. If 10 new cross-connects need to be added, then 10 OpenFlow modification messages are sent from the controller to the BTI7800 and total message size of 2060 bytes and ten separate responses of 660 bytes are received at the controller. If existing cross-connects are modified it is accompanied by 10 cross-connects to be deleted and 10 cross-connects to be added. In total 20 messages are communicated to a single BTI7800 device, each message of size 206 bytes on total 4120 bytes and response of 720 bytes are received at the controller.

For the use cases discussed in Section 4.3.2.2, includes deletion of two cross-connects and addition of two cross-connect. On total four OpenFlow flow modification of 206 bytes (824 bytes) are communicated to a single BTI7800 device and four response message of each 66 bytes is received. In case of NETCONF one single message of 2612 bytes is communicated to a BTI7800 device and the response of 52 bytes is received. As the experiments involve two BTI7800 devices the same is communicated to the other

BTI7800 device. More the number of sub flows need to be modified, the OpenFlow modification messages communicated on both directions (between the controller and BTI7800 devices) increases and causes network overhead. On the other hand, the size of the NETCONF flow modification message remains stable in both directions.

5.5 Summary

In this chapter and in the previous chapter we have discussed a set of use cases that evaluates both the protocols implemented as SBIs. The Table 5.1 represents the time taken by both SBIs to modify different number of cross-connects.

Number of Cross-Connects Provisioned	**NETCONF (sec)**		**OpenFlow (sec)**	
	Add	Delete	Add	Delete
1	3	0.5	3	0.75
2	6	0.5	6	1.50
4	12	1	12	3
6	18	1	18	4.50
10	31.5	1	30.5	7.50

Table 5.1 Cross-Connects Modification

The Table 5.2 is a summary of observation of all the use cases discussed to evaluate both the protocols implemented as SBIs.

Use case	*NETCONF*			*OpenFlow*		
	Time (sec)	Bandwidth Utilization (%)	No of Control Messages	Time (sec)	Bandwidth Utilization (%)	No of Control Messages
Initial	63	32.01 %	2 Messages	61	53.89 %	10 Messages
High Priority Flow (20 G)						
User B Request	-	100%	-	-	100%	-
	User A Request					
No Low Priority Flow Exists	31	94.4%	4 Messages	34	93.37%	8 Messages
Lower Priority Flow Exists	45	92.78%	4 Messages	49	95.85%	16 Messages
User A and B	31	95.62 %	4 Messages	33	95.7 %	8 Messages
Elephant Request						
Elephant Request	104	77.83 %	6 Messages	117	92.92%	48 Messages
Emergency Request						
Emergency Request	144	59.55%	6 Messages	169	92.42%	80 Messages

Table 5.4 Use Cases Results

Chapter 6: Conclusions and Future Work

6.1 Conclusions

In the era of intense high bandwidth demand growth, and unpredictably shifting traffic patterns, operators need their transport networks to become dynamically programmable in order to offer new services and to satisfy the traffic demand without over-provisioning the network resources. SDN-based transport optical networks allow the resources to be governed by policy management, enabling the network operators to transform their transport networks to function efficiently and also increases operational agility. We evaluate most commonly used SBIs, NETCONF and OpenFlow in managing the BoD across the transport optical interconnect depending on different kinds of traffic flows and demands. The behavior of both implemented SBIs is different based on the nature of the protocol.

The results presented are specific to BTI7800 environment. NETCONF references the BTI YANG model to represent and communicate the information from the SDN controller. BTI YANG-based NETCONF is faster, and more efficient in terms of the number of control messages communicated between the controller and the BTI7800 device, and reduces the complexity of the resource manager. On the other hand, OpenFlow accesses each cross-connect individually; therefore, it efficiently handles the bandwidth by minimizing the sub flow idle time, but the protocol is slower and the number of control messages increases proportionate to the amount of flow rules modified. If it is acceptable for the link to be idle for a few seconds, NETCONF is a

better protocol. OpenFlow is a better protocol if bandwidth being idle is an issue and can compromise with time and the network overhead.

As represented in Table 5.2, an increase in the number of cross-connects that need to be provisioned for each data center will affect the performance of the OpenFlow protocol as the control messages and the time taken to handle each message increases proportionate to number of cross-connects that needs to be modified. In case of NETCONF the control message indicating the modified set of cross-connects will still be a single message, the time taken will be proportionate to the number of flows need to be modified.

6.2 Future Work

Some suggestions are presented here to enhance the thesis work in the future. Having a protocol that can access a set of cross-connects and each cross-connect individually based on the use case is a best approach.

- Aggregation of OpenFlow flow rules is a possible solution to overcome the issues (time and number of messages communicated between the controller and the device) faced by the implemented OpenFlow protocol. The OpenFlow standard and ODL controller do not currently support aggregation of flow rules. On the other hand, flow table size and aggregation of flow rules in the core switches is a relatively new topic of SDN.
- OpenConfig [10] is a new SDN protocol, proposed as one of the standard SBI between the control plane and data plane devices. OpenConfig uses NETCONF and YANG-based structures as the underlying protocol to support OpenConfig messages. OpenConfig as a protocol focuses on the messages and information that

are essential to share to a SDN controller. The OpenConfig working group is in the early stages of designing the protocol messages and defining the YANG model. OpenConfig might help to overcome the issue faced by NETCONF protocol, the ability to access individual cross-connects. The alternative solution is redefining the BTI YANG model such that the set of cross-connects can also be referenced as individual cross-connects. In OpenConfig the YANG models are specified by the working group. On the other hand, use of BTI YANG and NETCONF as an interface provides a flexibility to modify the YANG models allowing variations. Variations to OpenConfig YANG model is not possible.

- Increase in the number of users per data center or addition of different kinds of traffic will result in an increase in the complexity of the resource manager shown in Figure 4.1. The resource manager needs to be extended to support the requirement of increase in the number of users per data center or addition of different kinds of traffic.
- Parallelization of the control messages issued from the SDN ODL controller to the BTI7800 network elements located at the sender and receiver interconnected data centers will speed up the process and removes the delays caused by the sequence of serial control messages issued from the controller. Issuing a cross-connect information to both the BTI7800 network elements at the same time by introducing multithreading in the resource manager will help to reduce the time taken by all the use cases. Advantage of introducing, parallelization of control signals directed towards sender and receiver data centers will help to reduce the

time taken by each use case by 50%. The synchronization between both the threads are important for maintaining a stable system.

Appendices

Appendix A : Sequence for High Priority Request from User A

In this appendix, the sequence explaining the detailed working of both protocols as SBIs for the use cases demonstrated in Section 4.3.2.2 is discussed.

Figure A.1, represents the sequence involved in configuring the high priority request using NETCONF protocol. The user interacts with the BoD application and the resource manager receives the request. The system exists in initial state with 10 sub flows or cross-connects provisioned. The user has requested for 2 additional sub flows for handling the high priority traffic of user A. The resource manager decides to drop the lower priority flow of user B. The resource manager communicates the modified 10 cross-connects information (that includes 8 cross-connects belonging to base flows of user A and B, and 2 cross-connects to the high priority flow of user A) to the sender BTI7800 device with the help of the control plane. On completion of the request, the sender BTI7800 device replies the status. The resource manager, through the control plane, communicates the same information to the receiver BTI7800. At the end of 8 seconds the BoD application signals the resource manager to return back to the initial configuration. The resource manager, through the control plane, signals the set of 10 cross-connects or sub flows that belongs to the initial configuration of both BTI7800 devices. As a result, cross-connects created for handling high priority traffic of user A are torn down and cross-connects to service the low priority flow are configured.

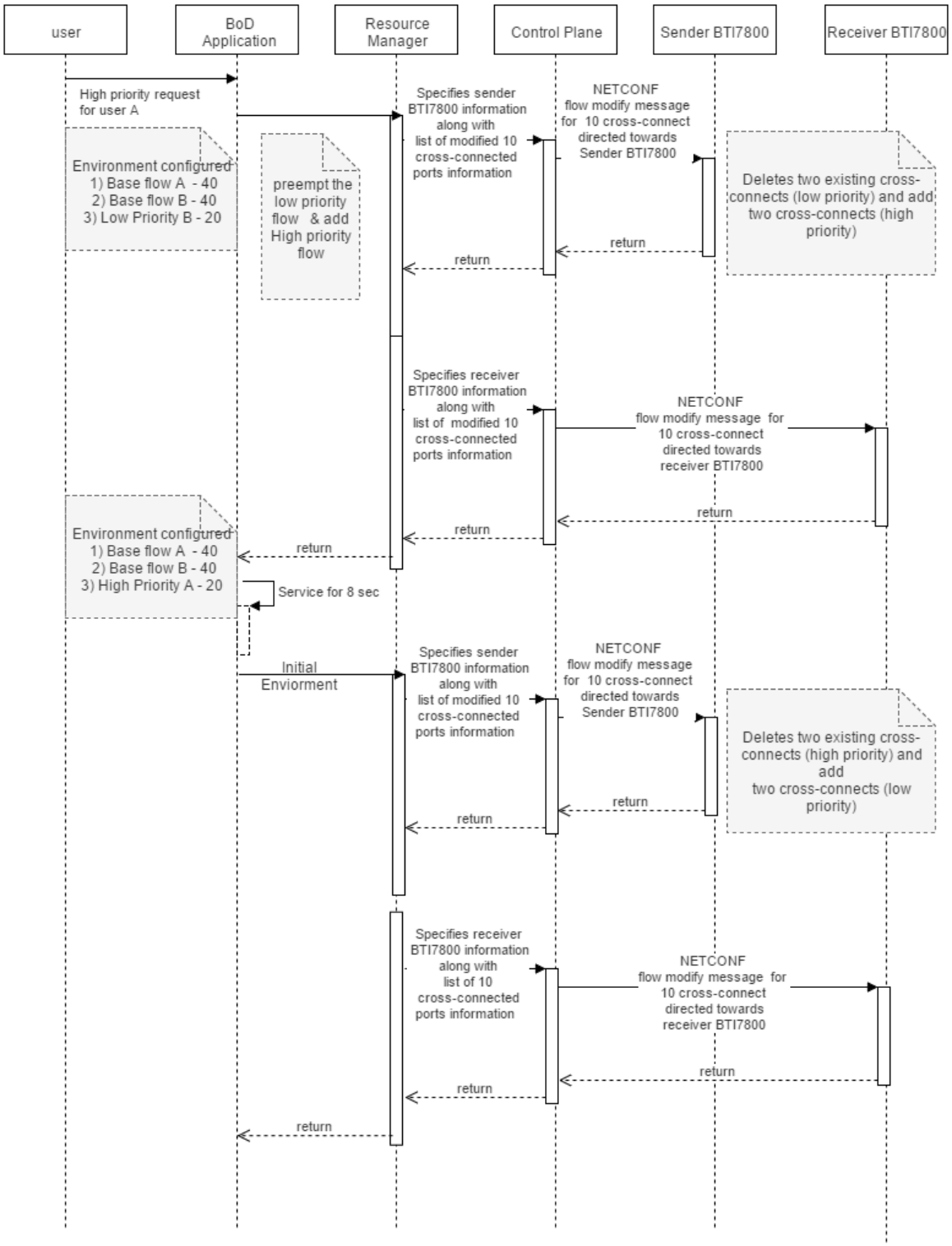


Figure A.1 Sequence for High Priority Request from User A Using NETCONF – Case 2

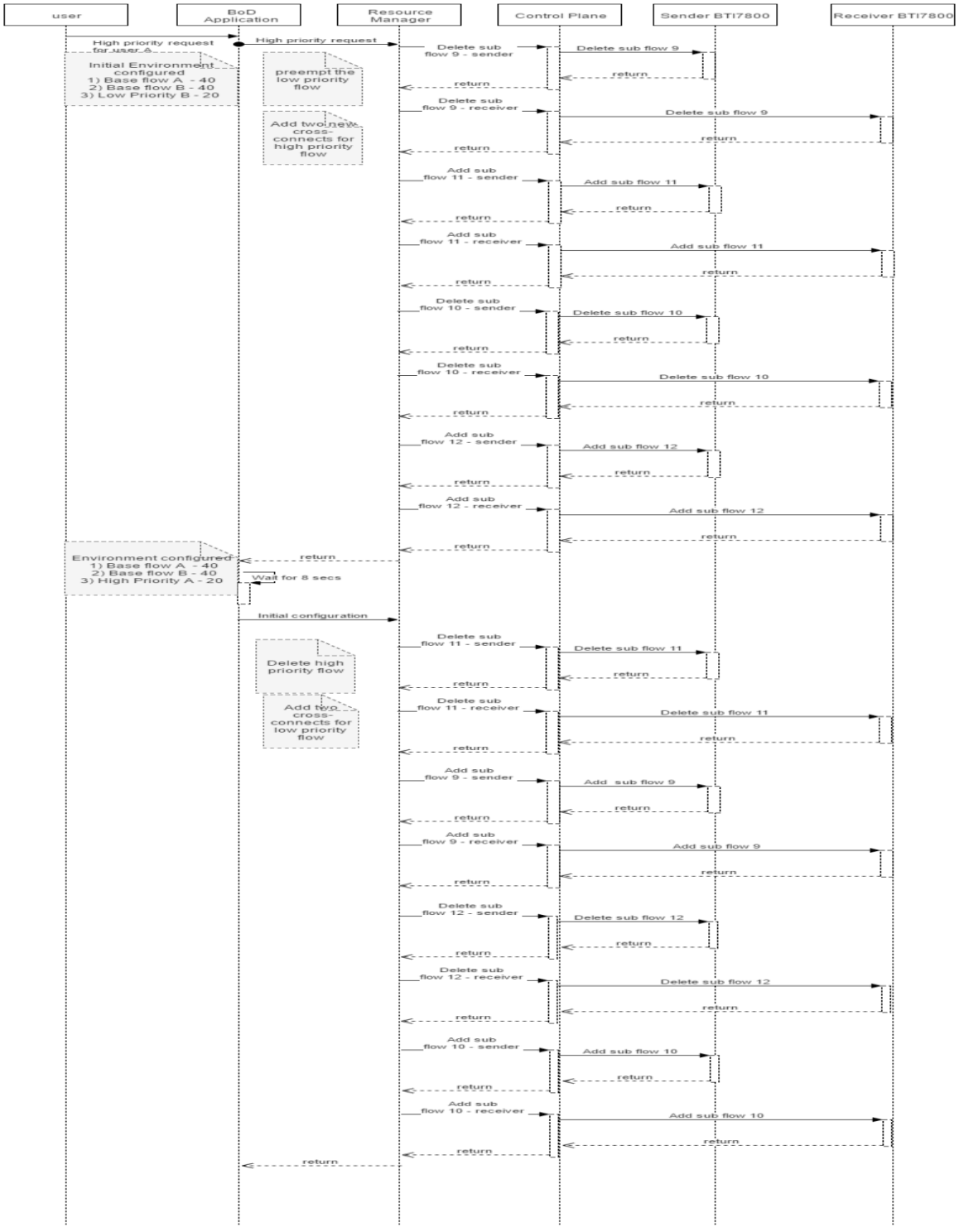


Figure A.2 Sequence for High Priority Request from User A Using OpenFlow – Case 2

Figure A.2, represents the sequence of steps involved in configuring the high priority request using the OpenFlow protocol as SBI. The user interacts with the BoD application and the resource manager receives the request. The system exists in the initial state with 10 sub flows or cross-connects provisioned. The user has requested for 2 additional sub flows for handling the high priority traffic of user A. The resource manager decides to drop the lower priority flow of user B. The resource manager communicates that the 9th cross-connect or sub flow is to be deleted to both BTI7800 devices. Next, the resource manager communicates that the 11th cross-connect or sub flow is to be added in the place of the deleted sub flow to both BTI7800 devices. The same procedure is followed to delete the 10th sub flow and add the 12th sub flow. The BoD application signals the resource manager to set up the initial environment once the system has serviced the high priority bandwidth requirement for 8 seconds. The resource manager follows the same procedure to delete a sub flow and add a sub flow at both ends accordingly.

Appendix B : Sequence for High Priority Request from Both Users A and B (Mutual Sharing)

In this appendix, the sequence explaining the detailed working of both protocols as SBIs for the use cases demonstrated in Section 4.3.3 is discussed.

Figure B.1 represents the sequence involved in handling the high priority requests using the NETCONF protocol as SBI. The user interacts with the BoD application and the resource manager receives the request. The system exists in the initial state with 10 sub flows or cross-connects provisioned. Both users have higher priority request and user B has lower priority flow. The resource manager handles the scenario by mutual sharing of resource among each user. The resource manager communicates the modified 10 cross-connects information (that includes 9 existing cross-connects and 1 modified cross-connect to satisfy the high priority flow of user A) to the sender BTI7800 device with the help of the control plane. On completion of the request, the sender BTI7800 device replies the status. The resource manager, through the control plane, communicates the same information to the receiver BTI7800. At the end of 8 seconds, the BoD application signals the resource manager to return back to the initial configuration. The resource manager, through the control plane, communicates the list of 10 cross-connects or sub flows that belongs to the initial configuration to both BTI7800 devices. As result, the cross-connect created for handling high priority traffic of user A is torn down and a cross-connect for servicing lower priority traffic for User B is configured.

Figure B.2 represents the sequence of steps involved in configuring the high priority request using OpenFlow as SBI. The user interacts with the BoD application and the resource manager receives the request. The system exists in its initial state with 10 sub

flows or cross-connects provisioned. The resource manager handles the high priority request from both users by mutual sharing and there is only one sub flow modification. The resource manager communicates that the 9th cross-connect or sub flow is to be deleted at both BTI7800 devices. Next, the resource manager communicates that the 11th cross-connect or sub flow is to be added in the place of the deleted sub flow to both BTI7800 devices. The BoD application signals the resource manager to set up the initial environment, once the system has serviced the high priority bandwidth requirement for 8 seconds. The resource manager follows the same procedure to delete a sub flow and add a sub flow at both ends accordingly.

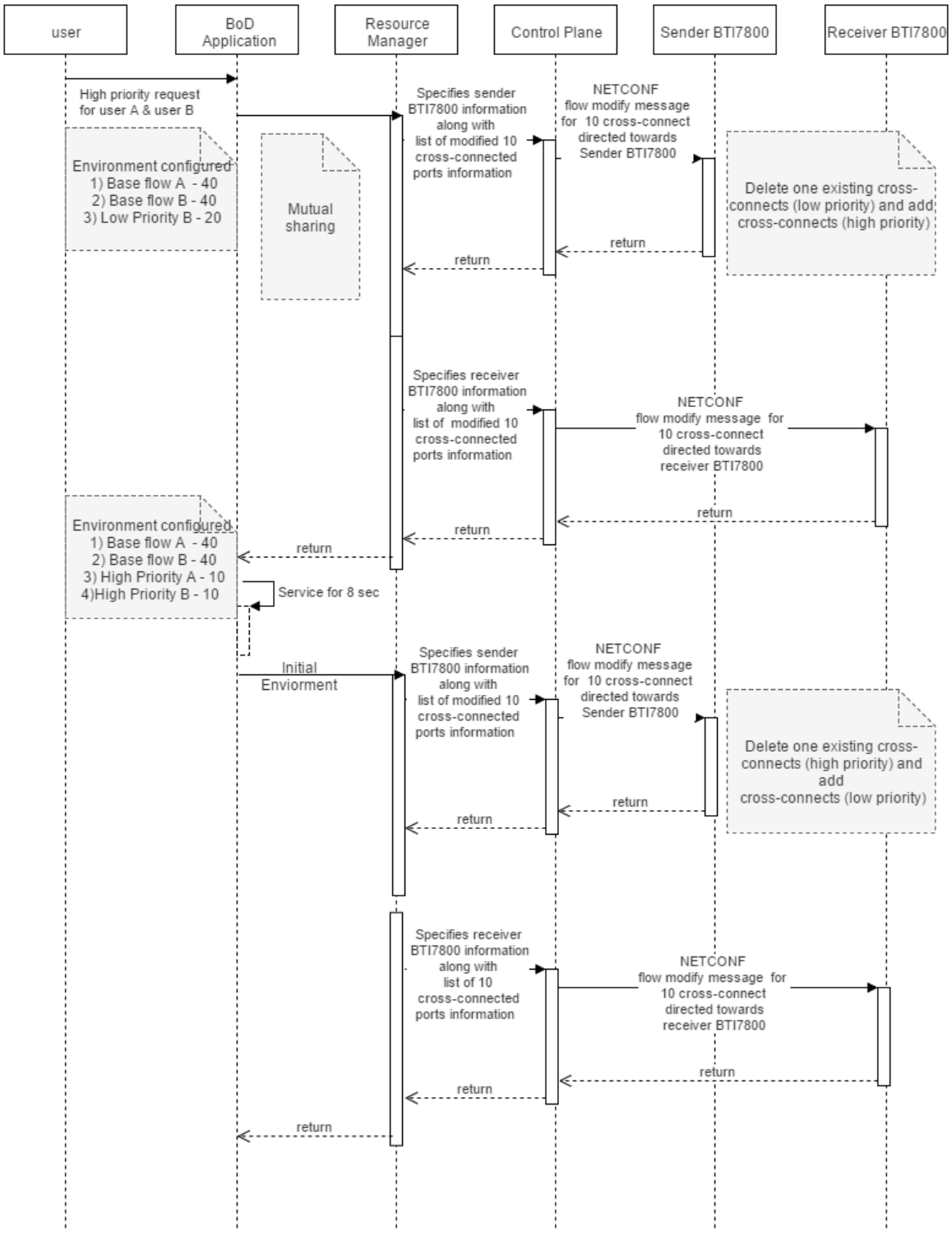


Figure B.1 Sequence for High Priority Request from User A and B Using NETCONF – Mutual Sharing

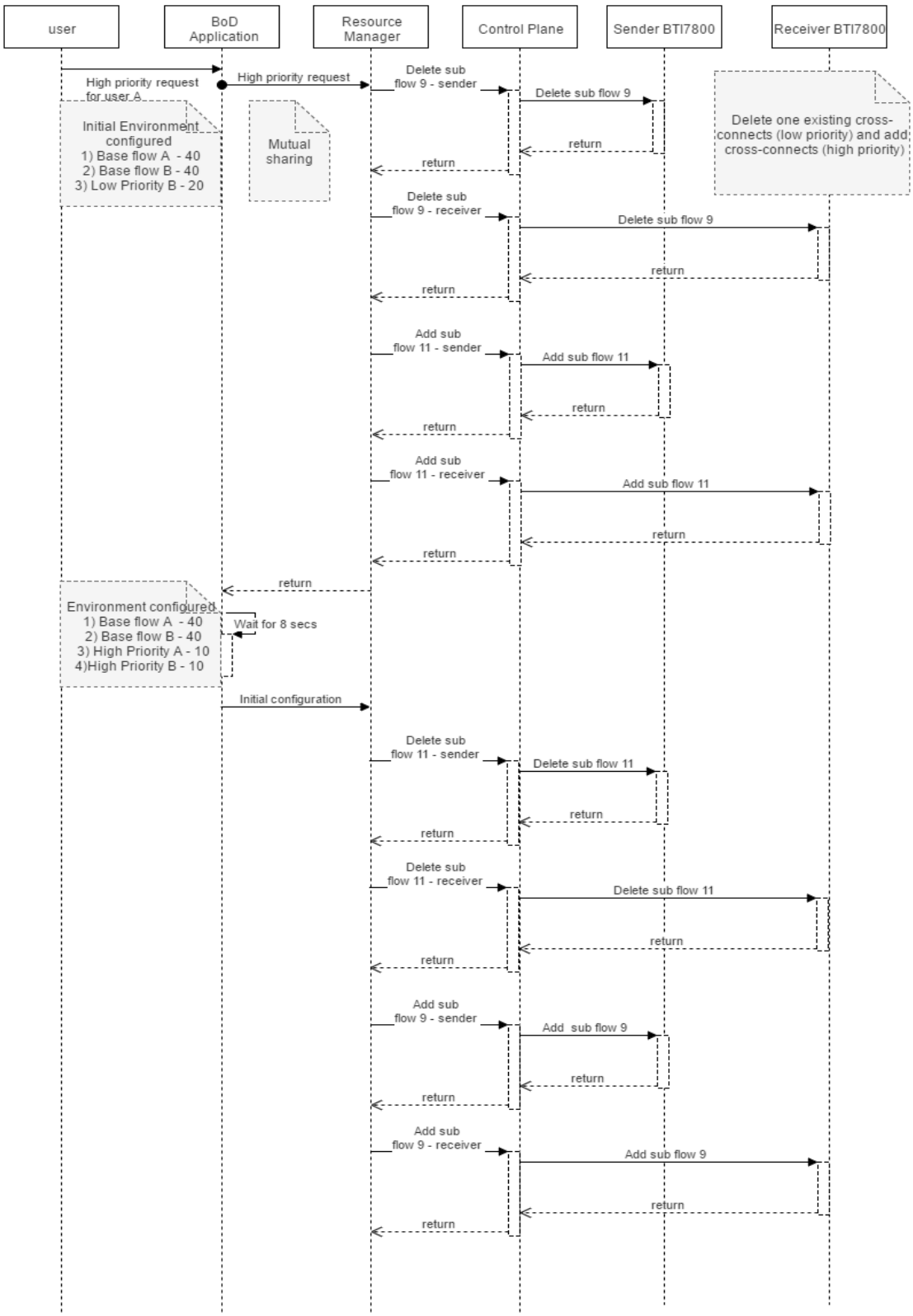


Figure B.2 Sequence for High Priority Request from User A and B Using OpenFlow – Mutual Sharing

Appendix C : Sequence for Emergency Request

In this appendix, the sequence explaining the detailed working of both protocols as SBIs for the use cases demonstrated in Section 5.2 is discussed.

Figure C.1 and Figure C.2 represent the steps involved in configuring the emergency request using NETCONF and OpenFlow. The sequence is similar to the elephant request discussed in Section 5.1. The user interacts with the BoD application and the resource manager handles the request. The major change in the emergency use case is the number of flows modified. The system exists in the initial state, with 10 sub flows or cross-connects provisioned. The user A requests bandwidth for emergency traffic and as a result six existing cross-connects are deleted and six new cross-connects are added. When user B's emergency request is provisioned, all the ten cross-connects are modified. When the system returns back to the initial state, four cross-connects are modified. As discussed earlier, in NETCONF the set of cross-connects that need to be provisioned are communicated as a single request each time. The NETCONF agent on the BTI7800 device figures out the modification and will tear down and add cross-connects accordingly. In OpenFlow the resource manager needs to identify the set of cross-connects that need to be changed and communicates the change for each cross-connect in a sequential order. OpenVSwitch, as it receives a sub flow modification request from the controller, validates the received information and posts the cross-connect modification into the message queue. confd then handles each modification one after another as discussed in Section 3.4.4.

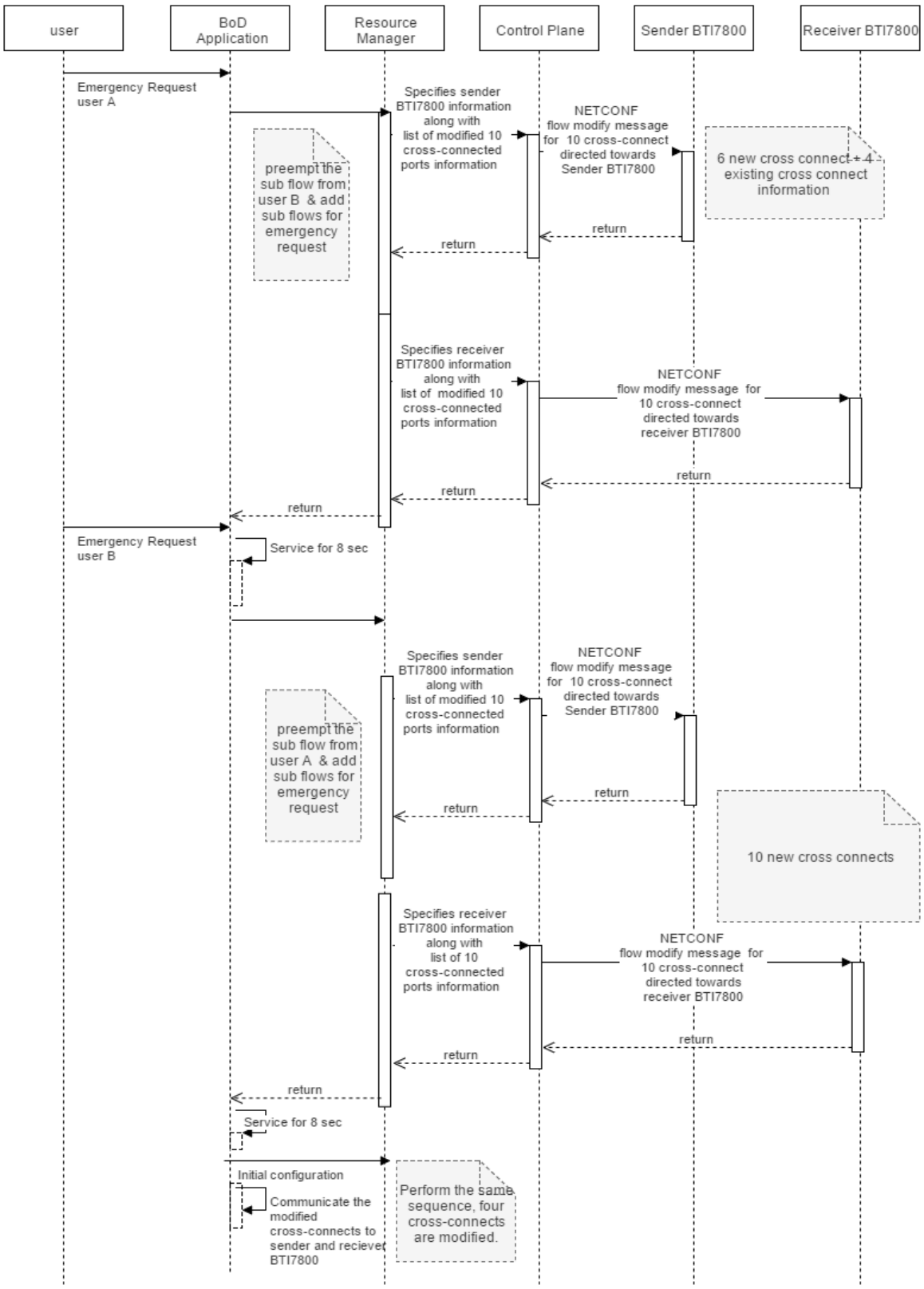


Figure C.1 Sequence for Emergency Flow Request Using NETCONF

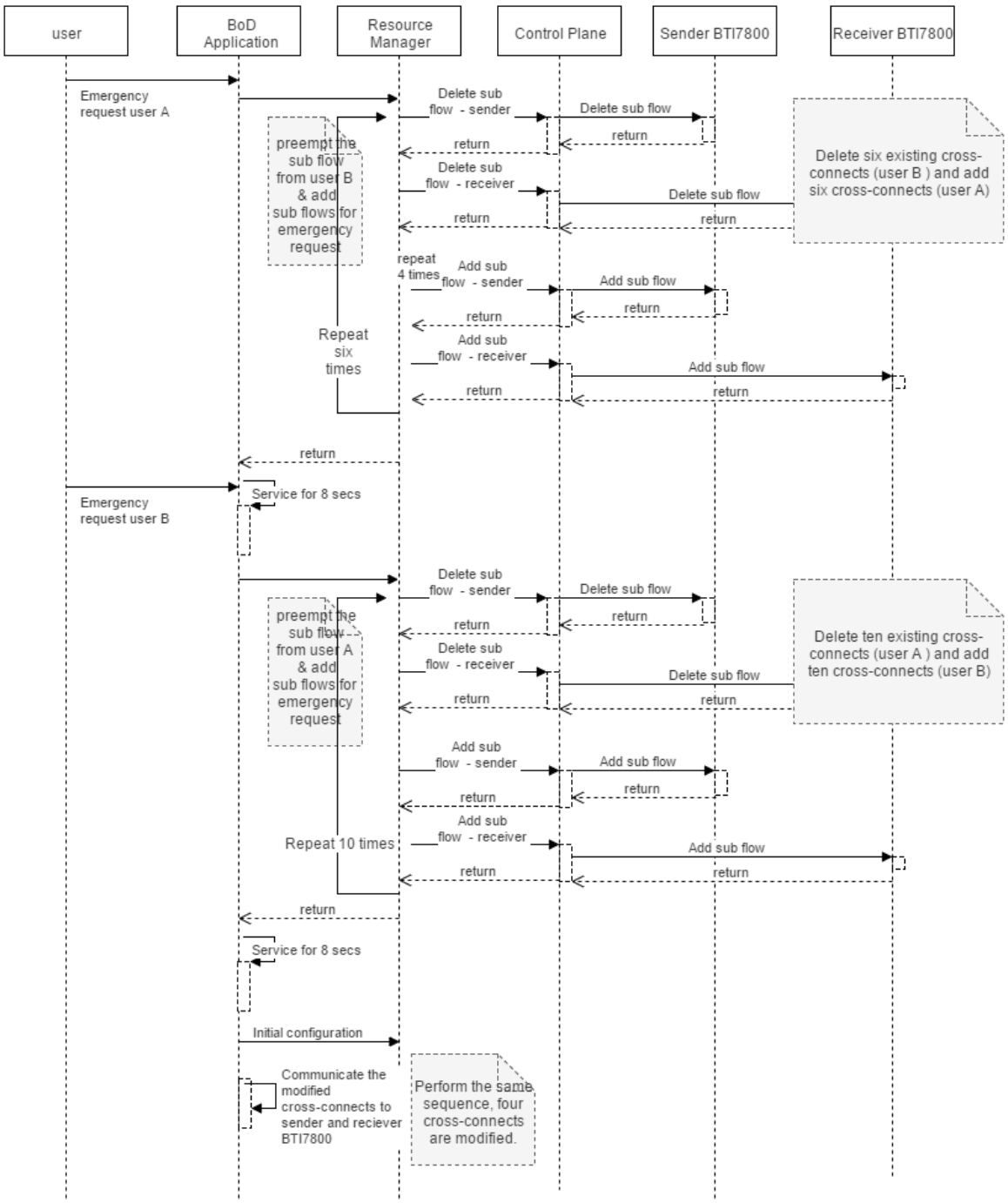


Figure C.2 Sequence for Emergency Flow Request Using OpenFlow

Appendix D : Packet Size of NETCONF Control Message

The ODL controller communicates flow modification control messages to the BTI7800 using NETCONF as a SBI. As we know flow modification in NETCONF is communicated as set of cross-connects. The size of the control messages communicated, varies based on a number of cross-connects specified in the message. The Table D.1 list all possible numbers of cross-connects that can be listed along with the respective control message size and the size of response message. In all the experiments we communicated 10 cross-connects, except Section 4.3.2.1 where 8 cross-connects where communicated. The table presents the control message packet size from controller to one BTI7800 device. In our experiment we have two BTI7800 devices, the same information is communicated to both.

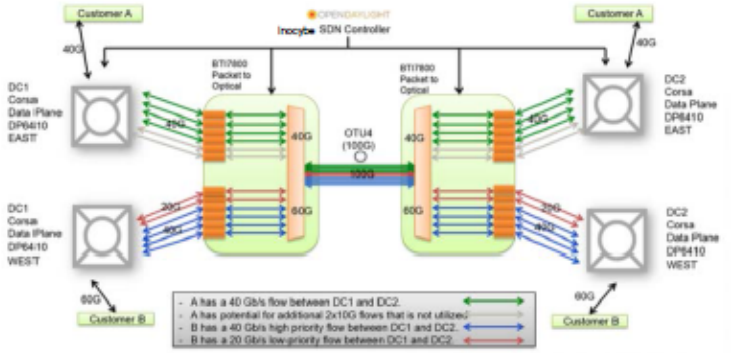
Number of Cross-Connects	Control Message Size	Response Message Size
1 cross-connects	660 bytes	52 bytes
2 cross-connects	900 bytes	52 bytes
3 cross-connects	1124 bytes	52 bytes
4 cross-connects	1332 bytes	52 bytes
5 cross-connects	1448 + 92 bytes	52 bytes
6 cross-connects	1448 + 316 bytes	52 bytes
7 cross-connects	1448 + 524 bytes	52 bytes
8 cross-connects	1448 + 732 bytes	52 bytes
9 cross-connects	1448 + 956 bytes	52 bytes
10 cross-connects	1448 + 1164 bytes	52 bytes

Table D.1 Packet Size of NETCONF Control Message

Appendix E : SDN & OpenFlow World Congress 2015 Demo

SCENARIOS

The following two scenarios utilize a live 100 GB/s CANARIE network WAN link between data centers in Ottawa and Montreal, to show how an elephant flow, during inter-data center transfer, can be addressed automatically with a multi-layer SDN solution – at both the packet and optical layers.



USE CASE 1: HANDLED AT PACKET LAYER USING CORSA QOS

- Customer A has 40G/s flow between DC1 and DC2, and 2x10G/s unused flows
- Customer B has 40G/s flow between DC1 and DC2, and 20G/s low-priority flows
- Customer B exhausts resources during a DC1 workload requiring additional 30G bandwidth for a VM migration to DC2
- Migration needs to be done immediately and is unscheduled
- SDN Controller senses 20G of congestion and signals Corsa device to use low-priority links in conjunction with QoS to provide Elephant Flow minimum bandwidth guarantee (16G)
- In absence of high priority traffic on low-priority links, the low-priority flows consume the entire bandwidth
- Optical device remains unchanged

USE CASE 2: HANDLED AT OPTICAL LAYER VIA BTI OTU4 CROSS CONNECT

- Customer A has 40G/s flow between DC1 and DC2, 2x10G/s unused flows
- Customer B has 20G/s flow between DC1 and DC2, and 20G/s low-priority flows
- Customer A exhausts resources in a DC1 workload requiring additional 20G bandwidth for a VM migration to DC2
- Migration needs to be done immediately and is unscheduled
- SDN Controller issues commands to establish 2 additional 10GE flows across the unused ports, deprovision Customer B's 2x10GE ports, pre-empting the low-priority traffic, and provision the new Customer A 2x10GE ports across the OTU4
- SDN Controller issues SDN commands to re-establish 20Gb/s low-priority bandwidth between DC1 and DC2 for Customer B

CENGN
CENTRE OF EXCELLENCE IN NEXT GENERATION NETWORKS

Michael Weir, VP Technology / Operations
michael.weir@cengn.ca

Peter Landon, Director Product Architecture
plandon@btisystems.com

David Whittaker, Product Management
david.whittaker@corsa.com

Peter Wilenius, VP Business Development
peter.wilenius@canarie.ca

Karim Lacasse, Senior Business Manager
karim@ixiacom.com

Mathieu Lemay, CEO
mlemay@inocybe.com

References

1) SDN The New Norm for Networks, ONF white paper, April 13, 2012. (https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf)
2) SDN Architecture Overview, Version 1.1, November, 2014, ONF TR-504. (https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN-ARCH-Overview-1.1-11112014.02.pdf)
3) OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04), June 25, 2012, ONF TS-006. (https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf)
4) SDN Architecture for Transport Networks, March 15, 2016, ONF TR-522. (https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN_Architecture_for_Transport_Networks_TR522.pdf)
5) OpenFlow enabled Transport SDN, ONF Solution Brief, May 27, 2014. (https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-of-enabled-transport-sdn.pdf)
6) Optical Transport Extensions, Version 1.0, March 15, 2015, ONF TS-022. (https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/Optical_Transport_Protocol_Extensions_V1.0.pdf)
7) Optical Transport Use cases, August, 2014, ONF TR-509. (https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/optical-transport-use-cases.pdf)

8) OpenDayLight Open Source OpenFlow controller wiki page and user manual, November 2016.
(https://wiki.opendaylight.org/view/Main_Page)
9) OpenVSwitch github code and details, Code tag – Master Branch, December 12, 2015.
(https://github.com/homework/openvswitch)
10) OpenConfig Working Group, November 21, 2016
(http://www.openconfig.net/)
11) Network Configuration Protocol (NETCONF), IETF, Request for Comments: 6241, Obsoletes: 4741, Category: Standards, ISSN: 2070-1721.
12) YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF), IETF, Request for Comments: 6020, Category: Standards, ISSN: 2070-1721.
13) Mayur Channegowda; Reza Nejabati; Dimitra Simeonidou, "Software Defined Optical Networks Technology and Infrastructure: Enabling Software Defined Optical Network Operations", IEEE, Pages: A274 - A282, 23 October 2013.
14) Robert Doverspike; George Clapp; Pierre Douyon; Douglas M. Freimuth; Krishna Gullapalli; Bo Han; Jeffrey Hartley; Ajay Mahimkar; Emmanuil Mavrogiorgis; James O'Connor; Jorge Pastor; K. K. Ramakrishnan; Michael E. Rauch; Mark Stadler; Ann Von Lehmen; Brian Wilson; Sheryl L. Woodward, "Using SDN Technology to Enable Cost-Effective Bandwidth on Demand for Cloud Services", IEEE/OSA Journal of Optical Communications and Networking, Pages: A326 - A334, Febuary 2015.

15) Nick Feamster; Jennifer Rexford; Ellen Zegura, “The Road to SDN: An Intellectual History of Programmable Networks”, ACM SIGCOMM Computer Communication Review, vol. 44 no. 2, April 2014.

16) Xenofon Foukas; Mahesh K. Marina; Kimon Kontovasilis,” Software Defined Networking Concepts”, The University of Edinburgh & NCSR Demokritos, Wiley, 2014.

17) D. B. Hoang; M. Pham, “On Software Defined Networking and The Design of SDN Controller”, Network of the Future (NOF), Pages: 1 -3, September 2015.

18) D. Kreutz; F. Ramos; P. Esteves Verissimo; C. Esteve Rothenberg; S. Azodolmolky; S. Uhlig, “Software Defined Networking: A Comprehensive Survey,” Proceedings of the IEEE, Pages: 14 - 76, January 2015.

19) Anna Lidia Soso; Gianmarco Bruno; Jeroen Nijhof , “DWDM Optical Extension to The Transport SDN Controller”, Fotonica AEIT Italian Conference on Photonics Technologies, Pages: 1-4, May 2015.

20) Alaitz Mendiola; Jasone Astorga; Eduardo Jacob; Kostas Stamos; Artur Juszczyk; Krzysztof Dombek; Jovana Vuleta-Radoičić; Jordi Ortiz, “Multi-Domain Bandwidth on Demand Service Provisioning Using SDN”, 2016 IEEE NetSoft Conference and Workshops (NetSoft), Pages: 353 – 354, June 2016.

21) Jason Min Wang; Ying Wang; Xiangming Dai andBrahim Bensaou,” SDN-based Multi-Class QoS-Quaranteed Inter Data Center Traffic Management”, Cloud Networking (CloudNet), IEEE 3rd International Conference, Pages: 401 - 406, October 2014.

22) Vijoy Pandey, “Towards Widespread SDN Adoption: Need for Synergy Between Photonics and SDN Within the Data Center”, IEEE Photonics Society Summer Topical Meeting Series, Pages: 227-228, July 2013.

23) Francesco Paolucci; Andrea Sgambelluri; Nicola Sambo; Filippo Cugini; Piero Castoldi; “Hierarchical OAM Infrastructure for Proactive Control of SDN-Based Elastic Optical Networks”, 2015 IEEE Global Communications Conference (GLOBECOM), Pages: 1 – 6, December 2015.

24) S. A. Shah; J. Faiz; M. Farooq; A. Shafi; S. A. Mehdi, “An Architectural Evaluation of SDN Controllers”, 2013 IEEE International Conference on Communications (ICC), Pages: 3504 – 3508, June 2013.

25) Abhinava Sadasivarao; Sharfuddin Syed; Ping Pan; Chris Liou; Inder Monga; Chin Guok and Andrew Lake, “Bursting Data Between Data Centers: Case for Transport SDN”, IEEE 21st Annual Symposium on High-Performance Interconnects, Pages: 87 - 90, August 2013.

26) M. Siqueira and J. Oliveira, “An Optical SDN Controller for Transport Network Virtualization and Autonomic Operation”, Globecom Workshops (GC Wkshps), IEEE, pages: 1198-1203, December 2013.

27) L. Velasco; A. Asensio; J. L. Berral; A. Castro; V. López, “Towards a Carrier SDN: An Example for Elastic Inter-Datacenter Connectivity”, Optical Communication (ECOC 2013), Pages 1-3, September. 2013.

28) Yongli Zhao; Jie Zhang; Hui Yang and Xiaosong Yu, “Data Center Optical Networks (DCON) with OpenFlow Based Software Defined Networking (SDN)”,

Communications and Networking in China (CHINACOM), Pages: 771 – 775, August 2013.