SOFTWARE DEFINED SURVIVABLE OPTICAL INTERCONNECT FOR DATA CENTERS

By

SONALI CHANDNA

Thesis submitted in partial fulfillment of the requirements for the Master of Applied Science in Electrical and Computer Engineering degree

School of Electrical Engineering and Computer Science

Faculty of Engineering

University of Ottawa

Abstract

For service providers, extending the Software Defined Network (SDN) concept from packet switching in Layers 2 and 3 to circuit switching in transport layers is a promising approach to meet high burstiness and high bandwidth requirements. A multi-layer controller that can provide automated controller-based restoration and protection, even for unprotected links in a multi-administrative domain, would be a significant improvement. It would allow service providers to ensure provision of guaranteed Service Level Agreement (SLA) maintenance, with optimal bandwidth usage, high availability and reduced errors.

In this thesis, we propose a Software Defined Survivable Optical Interconnect (SDSOI) architecture for Data Centers (DC). This unique architecture will address service providers' challenges related to bandwidth management, and optimize the time required while interconnecting numerous DCs to meet the high SLA demands. The architecture is built according to the overlay SDN concept, and categorizes the application layers into online, offline and third party applications. The offline application performs the routine DC tasks, while the online application manages various dynamic DC demands. An SDSOI driven Data Center Optical Interconnect (DCOI) can handle the extensive, high quality, on-demand access to the contents. The feasibility of SDSOI is verified and demonstrated using Open Network Operating System (ONOS) as the SDN controller, Mininet as the network emulator and Optical Transport Network (OTN) as the optical framework.

Our work primarily focusses on the creation of business applications in the SDSOI model, and the northbound protocols or interfaces used by the applications to interact with the controller. To verify the effectiveness of the proposed SDSOI architecture and its business applications, we simulated 'Day Night Scheduling', an application that combines characteristics of offline and online applications. Its primary function is to package and schedule varying DC bandwidths to service providers and optimize bandwidth usage at different times of day, along with the survivability of the interconnects.

Dedication

This Thesis is completely dedicated to my **parents**, who have imbibed since my childhood nothing is undoable in this world. For their endless faith, prayers, love, trust, support and dedication in raising and supporting me. My parents support was the greatest source of inspiration, dedication and motivation to accomplish this work. They have always been my true guiding light throughout.

Acknowledgement

First, I would like to express my gratitude to my supervisor, **Prof. Hussein T. Mouftah**, for potentially providing me with his outstanding guidance, support and motivation throughout my studies to complete this thesis. His valuable technical help and encouragement was a great source of inspiration in advancing this research area into the Next Generation Networks. Along with his continuous motivation, his teachings in the course "Survivable Optical Network" helped a lot in building the in-depth understanding of optical networks and interconnects and survivability in them.

I am also very grateful to my co-supervisor, **Dr. Nabil Naas**, who has always helped with his valuable suggestions and feedbacks to strengthen this research topic. The implementation of this proposed model was more challenging with his continuous motivation to simulate something new for industrial requirements prospective. He was a constant source of help and encouragement with in this work.

I am also grateful to the entire team of CENGN for giving me the opportunity as an Intern to conduct the PoC of Fujitsu Optical SDN Controllers and for training me with fundamentals of Optical Transport Network.

I would also like to thank Puneet for being a continuous motivation in my Master's program and always pushing me to learn more and more concepts as well as encouraging me each day to do the things, which I was reluctant in learning.

Last but not the least; I want to thanks my family for supporting me in my decision to pursue my masters overseas. Without your support and love, living alone for these two years would have been challenge. You all were the source of continuous courage with words "Don't give up until succeeded". I am also very appreciative and indebted to my brother, CA Sahil Chandna for his immense support during my master's program journey. He is the one who made me dream of my career and supported me in achieving heights always in my life. He is the true ideal of my life on whose footsteps I would always like to follow my life. You three are the real god in my life who have always struggled to give me everything I desired and dreamed.

Table of Content

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface.
APIC	Application Centric Infrastructure Controller.
ARP	Address Resolution Protocol.
BGP	Border Gateway Protocol.
CLI	Command Line Interface.
CSO	Cross Stratum Optimization.
CSS	Cascading Style Sheets.
DC	Data Center.
DCI	Data Center Interconnects.
DCOI	Data Center Optical Interconnects.
DNS	Day Night Scheduling.
DOP	Dynamic orchestration plane.
DWDM	Dense Wavelength Division Multiplexing.
EVPN	Ethernet Virtual Private Network.
GMETAD	Ganglia Meta Daemon.
GMOND	Ganglia Monitoring Daemon.
GMPLS	Generalized Multiprotocol Label Switching.
GRIR	Global Resources Integrated Resilience.
GUI	Graphical User Interface.
HTML	Hyper Text Markup Language.
ILP	Integer Linear Programming.
IoT	Internet of things.
ISP	Internet Service Provider.
JAR	Java Archive.
LAN	Local Area Network.
LaSS	Latency aware service scheme.
LSP	Labelled Switch Path.
MAN	Metro Area Network.
MPLS	Multiprotocol Label Switching.
MSRIR	Multi-Stratum Resources Integrated Resilience.

NCS	Network Control Server.
NFV	Network Function Virtualization.
NOS	Network operating system.
NSP	Network Service Platform.
NVO	Network Virtualization Overlay.
OAM	Operations, Administration and Maintenance.
OAR	ONOS Application Repository.
OCh	Optical Channel hierarchy.
ODL	Open Daylight.
ODU	Optical Data Unit.
OF	Open Flow.
OMS	Optical Multiplex Channel.
ONOS	Open Network Operating System.
OPU	Optical Packet Unit.
OS	Operating System.
OTH	Optical Transport Hierarchy.
OTN	Optical Transport Networks.
OTS	Optical Transmission Channel.
OTU	Optical Transmission Unit.
OVSDB	Open vSwitch Database Management Protocol.
PoC	Proof of Concept.
POM	Project Objected Model.
QoS	Quality of Services.
RARP	Reverse Address Resolution Protocol.
REST	Representational state transfer.
ROADM	Reconfigurable, Add and Drop Multiplexer.
SDDCI	Software Defined Datacenter Interconnects.
SDN	Software Defined Networking.
SDH	Synchronous Digital Hierarchy.
SDON	Software Defined Optical Networks.
SDSOI	Software Defined Survivable Optical Interconnects.

SLA	Service Level Agreement.
SNMP	Simple Network Management Protocol.
SONET	Synchronous Optical Networking.
SUDOI	Software Defined Ubiquitous Data Center Optical Interconnect.
TaSS	Time aware Service Scheme.
ToR	Top of the Rack.
TCM	Tandem Connection Monitoring.
TL1	Transactional Languages 1.
VLAN	Virtual LAN.
VM	Virtual Machine.
VON	Virtual Optical Networks.
VXLAN	Virtual Extensible LAN.
WAN	Wide Area Network.
WDM	Wavelength division multiplexing.

Chapter 1: Introduction

1.1 Introduction

Due to the accelerating pace of technology and increasing traffic demands, next generation technologies such as Cloud, Internet of Things (IoT), OpenStack, Software Defined Networking (SDN) and Network Function Virtualization (NFV) will be the ones with key potential in defining keys to build more agile, automated and orchestrated inter- and intra-datacentre DC networks. As the industries embrace these new technologies for programmable, scalable and automated control, provision of current IT services will change dramatically. With these change inside the DCs, there is a serious need to bring and build intelligent next generation solution for Service Providers (SP's), in the form of transport SDN [1] for interconnecting the data centers. Bringing SDN in service provider layers would ensure high bandwidth Wide Area Network (WAN)/ Metropolitan Area Network (MAN) connectivity among DCs, and accelerate service innovations.

Since transport networks have become large and very complex, an appropriate network model with well-defined, technology agonistic modules and functional entities are required to ensure the provision of optimal, real-time on-demand programmable bandwidth services [1]. To simplify, the Operations, Administration and Maintenance (OAM) service provider controllers must be designed in a multi-domain, multi-vendor and multi-layer architecture, and ensure centralized control of the entire network. With the advances in transport optical layers, particularly Optical Transport Network (OTN) for Layer 1 and DWDM for Layer 0 [2], service provider networks should be designed in such a fashion that most of the attributes are controlled by the software.

Service providers act as a bridge for the communication of data from one location to another. Until now, service providers have difficulty creating fully loaded 100G, 400G and 1TB connections for interconnecting the DCs over transport layers and WANs. Delivering such high bandwidths while maintaining Service Level Agreements (SLA) of latency and bandwidths is a challenge for service providers. To maintain these SLAs by service providers, there is an urgent need for a 'magical black box' at the transport layer. With SDN, it is possible to have such black box that creates the magic and enables flexible optical interconnects for DCs, and ensures optimized bandwidth with time-aware sensitive scheduling of services. By adopting this approach, service providers can offer on-demand connectivity between DCs, and manage survivability of the interconnects.

To implement such an intelligent transport layer architecture, we propose a Software Defined Survivable Optical Interconnect (SDSOI) architecture for DCs, designed on OTN [3]. OTN is recognized as the impending industry-standard protocol to provide an efficient and globally accepted way to multiplex different optical services, such as Ethernet, storage, digital video and SONET/SDH, to an optical path. SDSOI centralized controller has global visibility, and controls both the transport and IP networks, thereby enabling multi-layer provisioning for end-to-end DC communication. Centralized control helps provide operational efficiency by provisioning on-demand and time sensitive scheduled bandwidths and caters to the required transport bandwidth from any DC through a single user interface. It also helps evolving transport providers with high bandwidth, low pay-per-use cost, low latency, low bit-error rate, guaranteed services and SLA.

With this global view, service providers would offer increased network efficiency by increasing the transit traffic through optical gateways by online packaging and scheduling of traffic in the transport layer and computation of optimized paths for traffic to be carried either way. This helps ensure load balancing and online bandwidth management of diverse customer demands, and also acts as the interface for creating the primary network survivability interface.

1.2 Motivation

Service providers play a major role in interconnecting DCs, and computing and supporting different applications among geographically specific DCs. With the growing need for higher bandwidth connections, it is important for service providers to both abstract and virtualize transport network resources. This will enable higher layer transport controllers to bring up, reroute and tear down optical paths, which leads to less traffic congestion and higher bandwidth connectivity. We address the following motivations in service provider networks and SDN; to our knowledge they have not been considered together in the literature. This led us to investigate the possibility of bringing survivability, as well as scheduling and calendaring of the Optical DCI:

- The primary motivation for this thesis is to help service providers build an ideal optimized solution, and enable time, bandwidth and Quality of Service (QoS) sensitive survivable optical interconnects for DCs that are geographically apart [4]. When service providers interconnect DC’s, various factors need to be considered, including traffic engineering, dynamic optical bypass, variable bandwidth packet links, load

balancing, differential treatment for different types of DCs, customized routing for each kind of customer, different wavelengths for different customers (shared or dedicated), spectrum management, topology and path computation and optical signal monitoring. Our action plan is to help service providers strike a balance of these features, with ensured continued bandwidth connectivity among the DCs through programmability, automation and orchestration.

- In addition to automation and orchestration, service providers should be able to create rapid, optimized bandwidth interconnection services with minimal cost. To cater to such service demands, they must do online packaging and scheduling of traffic demands in a way that ensures bandwidth connectivity for every DCI request [5]. Our scheme is to bring the orchestration and automation of SDN to DCI, in an extremely scalable, high bandwidth service provider network.

- Online packaging and scheduling would more effective and productive if service providers were aware of interconnection requests in advance. This pre-knowledge of demands would allow service providers to pack the bandwidth demands in a way that allows DCI to carry maximum service demands [5]. Our vision is to let service providers be prepared according to the calendar and sun-pattern, so they can deploy high bandwidth connectivity around the world.

- DCI [6-9] carries enormous traffic services with high bandwidth, and failures could lead to significant imbalance in DCs and their end customers or users [10]. We propose to introduce survivability in service provider software defined optical DCI, so traffic failures are imperceptible.

All these requirements are motivation for service providers to introduce SDN at the transport layer for interconnection, and to have their own policies and intelligence for optimized use of bandwidth and survivability at minimal cost. We aim to utilize the transport SDN capabilities of programmability, and propose an application driven SDN architecture for service providers optical interconnects.

1.3 Objective

The primary objective of this research is to introduce software programmability in the transport layer, to achieve, Software defined survivable optical DC interconnects for connections and migration of DCs in MAN/WAN or service provider. To accomplish such interconnections our work focuses on the industrial requirements of service providers to interconnect small, mid-size datacenters to large datacenters, with the aim of, fulfilling high bandwidth connectivity demand in an effective network design, flexible on-demand capacity allocation to the interconnects with pay per use capability.

In our work, we intend to deploy a smart business application for an application-driven service provider SDN architecture that can create a DCI with the following capabilities:

- Optimized use of bandwidth among DCs through effective packaging and scheduling;
- Follow the sun globally and according to calendared events, and thereby create high bandwidth interconnections in active hours and reduce the bandwidth in non-active hours for data replication, VM motion and other tasks;
- Deploy ‘time sensitive’, ‘bandwidth sensitive’ and ‘QoS sensitive’ DCI services by service provider;
- Reduce the CAPEX and OPEX for both service provider and its customers, by letting customers pay-per-use for bandwidth and type of connection;
- Effective traffic management, and restoration of interconnections upon failure by introducing survivability and programmability to the optical interconnects;
- Make service provider networks programmable, automated and orchestrated by effectively introducing SDN in such a way that DC network failures are imperceptible; and,
- Utilize OTN, the next generation optical transport protocol for service provider DCIs, in combination with SDN next generation networking technology.

1.4 Contribution

The important contributions of this thesis include the design and implementation of SDSOI architecture [11] for DC interconnections by service providers. We also simulated a DNS application and created REST API for it. DNS is designed for effective day night scheduling

activities using time and bandwidth sensitive intents to provide customized variable bandwidth at different times and for different DCs, and ensures survivability of these interconnects.

The proposed SDSOI architecture for service providers is designed with common north and south bound interfaces, which allows them to not only create, deploy, monitor and maintain any type of services, but also control them with the centralized intelligent control plane. SDSOI is an open source platform, with Open Flow (OF) as the southbound communication protocol and open northbound interfaces. The primary idea behind this open source architecture is to have open interfaces from applications to controller and controller to forwarding plane. This ensures support of any type of routing services for service providers, and maintains business continuity.

SDSOI is an application driven controller with built-in low and high-level plugins for basic functionality that can support various types of business applications, according to network requirements. Deploying such an application-driven approach optimizes resources and reduces costs, while providing high performance services and scalability. Applications are programmed with the business logics as per the network requirement and address increasing traffic demands and high-speed connectivity along with optimum use of bandwidth and minimal latency.

SDSOI is proposed with several unique applications, categorized as offline, online and third party. In this work we focus on the offline and online applications, with the aim of having optimize bandwidth usage by scheduling different bandwidths at different times of the day, and ensuring bandwidth continuity even inspite of failures by providing protection and restoration for paths, segments or links. They assure low latency, and can be easily provisioned in both protected and unprotected scenarios. The only difference between the two is protected interconnections have dedicated proactive paths, segments or links already programmed, whereas unprotected interconnections restores the interconnections via active neighbour discovery and liveliness monitoring of the optical gateways. Optical gateways inform the SDN controller about failures, signaling it to reroute the traffic using the next available shortest path.

DNS application is designed on an ONOS controller, and can install multiple types of intents between any pair of gateways with different bandwidths during scheduled loads of the network, and automatically revert back to the minimum default bandwidth setting. Supporting OTN standards, it delivers automatic bandwidth scheduling of interconnects, and rapid protection and restoration services. DNS is programmed to provide link and segment protection, as well as path

failure protection depending on the interconnects restoration time, priority, capacity and cost. DNS application in SDSOI architecture is evaluated under different topologies for restoration time, throughput, resource utilization and protection schemes and has higher performance than any of the existing scheduling activity application created by researchers.

SDSOI uniqueness is defined in terms of restoration and protection of the interconnects along with scheduling at the same bandwidth with the initial intended constraints. DNS application intents can be installed and viewed either in CLI, or by directly using our user-friendly application GUI. Topology dashboard on GUI and server script of ONOS tracks the application being installed, as well as the rapid protection and restoration service being installed on the interconnect.

1.5 Outline

The rest of the thesis is as follows:

- Chapter 2 describes the SDN concept and data center interconnects, and discusses related work done in optical SDN, WAN SDN and transport layer SDN by academic institutions and industry. We also explain SDN controls in hierarchical and centralized approaches, and briefly describe OTN standards. The chapter concludes with discussion regarding survivability in optical networks, and related work on using survivability for SDN.

- Chapter 3 explains the concept of SDN in terms of transport layers and service providers, and describes how Open Flow is extended for the SDSOI model. We detail the SDSOI architecture by focusing on each layer, discuss the application-oriented framework, algorithms for protection and list the advantages of day night scheduling application.

- Chapter 4 describes the simulations performed in the thesis, focusing on the environment setup and tools and technologies used, and briefly examine ONOS as the SDN controller. It includes the steps and procedures used to build the simulated application, and highlights application's intent creation.

- Chapter 5 extends the work of the previous chapter by performing various tests and analyzing the application in different scenarios. It includes performance testing in terms of packet monitoring, restoration time and interconnect utilization and capacity. Finally, we compare the results and efficiency of our work against other relevant research.

- Chapter 6 concludes this thesis by outlining potential work for the future.
- Appendix A briefly explains the traditional standardized optical transport protocol SONET/SDH, and outlines its limitations.
- Appendix B examines the next generation OTN optical communication protocol and its hierarchy, including a description of our extended TCM fields in OTN headers in OF protocol for SDSOI architecture.
- Appendix C describes test cases for DNS application in different topologies with diverse sets of source and destination pair gateways, under different protection and restoration schemes.
- Appendix D presents snapshots of the simulations performed in Chapter 4, and the results related to Chapter 5.

Chapter 2: State-of-the-Art

2.1 Introduction

This chapter discusses an important, topic-related survey used in this thesis. It includes work done by researchers in both academia and industry, and covers state-of-the-art technologies such as DCI, SDN, SONET, OTN and survivability of optical networks. Understanding these technologies, concepts, protocols, challenges and the research done so far is what led to the proposal of SDSOI architecture in Chapter 3. This survey helps the reader better understand the remaining chapters of the thesis.

2.2 Data Center Interconnects (DCI)

Optics has enormous potential for high bandwidth transmission, with experimental efficiency of 400GB/s to 1TB/s for a single fiber; DC's are interconnected using these fibers. They come in many forms and sizes with a variety of connectivity requirements, and can be tailored to enterprise, industry or public sector customers. Service providers offer a range of services to connect DCs, from managed and unmanaged interconnects, to guarantees to maintain their commitment to SLA [12]. While providing these services, service providers face two major obstacles: high-speed bandwidth and low latency.

The continuous increase in the growth of DCs across the world has helped Datacenter Interconnects (DCI) become the fastest growing application for optical networks in the global market [7]. DCI is a technology used to connect two or more DCs together over short, medium or long distances, with high-speed packet optical connectivity. With DCI, service providers can offer a wide range of optical DCI services, including carrier ethernet, dark fiber and managed wavelengths.

Carrier Ethernets are characterized by their use of high bandwidth Ethernet services, for internet access and communication between LANs. They use either microwave or fiber (SONET/SDH/DWDM/OTN) as the source medium for communication. Dark fibers are a committed and secure fiber service, with significant advantages. Managed DCI services use different wavelengths for different customers. They are managed by SPs, and maintain the committed SLA bandwidth and latency, even if service fails. Managed DCI services provide

several benefits to SPs, such as higher revenues and profit margins, enhanced business continuity and disaster recovery [13].

Service provider networks are typically composed of optical fibers and IP networks around the world. They are organized as a few central offices in different regions, and numerous other points of presence connected to the central offices to maximize the footprints and optimise reach. With the increase in the number of DCs, it has become difficult for service providers to manage them and offer high quality of service. Providing high QoS for SP requires centralized management and monitoring solutions that can control the interconnections, and provide optimized bandwidths for the DCs.

With more customers, migrating toward cloud interconnects and next generation networking infrastructure, SPs have more opportunity to increase revenues and drive profitability by using a centralized management system. The primary objective behind this is to provide DC customers, or smaller ISP customers, with more opportunity to build cost-effective, scalable, high-performance cloud interconnect services. Though building a flexible customer-centered platform is not easy, it can be made simpler by adopting SDN to transport layers for service providers, and making software defined optical DCI. This approach can provide highly scalable bandwidth to DCs and smaller ISPs, regardless if they are in private, virtual private, public or hybrid clouds, and can also offers more agility, flexibility and higher cost benefits. For these interconnections service providers advantage dark fibers, managed wavelength, Ethernet, OTN,IP/MPLS and others, can provide high revenues and margins, managed cloud interconnects and enhanced DCI services among DC and ISP customers.

These innovations, and the research that developed them, are now widely accepted in industries such as Nokia (formerly Alcatel Lucent), Cisco and others, and many have started incorporating them into their products and services. Nokia explains the recent market trends and DCI requirements in one of their strategic white papers [7], which defines industrial terms such as synchronous and asynchronous DCI, and considers methodologies using DCI dependence for DC needs. They also explain how DCI can be used optically for wavelength division multiplexing (WDM) in tiers 1 and 2, and how Ethernet and IP are used for DCI in tiers 3 and 4. This white paper also covers DCI evolution with respect to data center virtualization technologies like NVO and VXLAN, and how they are combined with existing routing technologies such as MPLS, BGP

and cloud-based WAN solutions. They also discuss how their DCI solution can be integrated with SDN and EVPN, and offer a simple, efficient, flexible and fragile way to automatically connect data centers in remote locations. Most of Nokia's access devices, such as 'on date', are capable of SDN solutions, and can deliver virtualization to unrestricted DCN within a DC. However, they have not yet extended SDN across WANs.

For maintaining business continuity services across the globe, large enterprises successfully and securely back up or replicate critical data and applications between multiple locations. DCI is an ideal way to address such demands, and is widely used. They are categorized by type of application, latency and networks, synchronous or asynchronous connectivity between the sites, and the distance between backup and primary sites.

Currently, many large enterprises with dedicated performance and security requirements use Optical WDM for DCI. However, due to recent advances in new technologies, demand for high bandwidth connections has increased significantly, and the traditional WDM/DWDM networks cannot support these bandwidth increases. High bandwidth interconnection requires an evolving DCI on a new optical standard protocol. Optical Transport Networks (OTN) is considered an ideal optical fit for such high bandwidth requirements. It is a next generation optical network protocol, and is growing well in the industry.

A combination of SDN and OTN for DCI could be a boon for SP, and help achieve high bandwidth managed DCI to customers. By adopting SDN in DCI for OTN, service providers will be able to deploy the most appropriate services for each customer, and every DC enterprises can share data, distribute the applications, maintain the workload balance and replicate or backup the data efficiently. The adoption of Software Defined DCI (SDDCI) by service providers would provide the following benefits:

- Scalable, flexible, direct connect and managed DCI services;
- High performance, scalable DCI services for DC and collaborations across the Campus;
- Scalable high-performance metro and long-haul DCI services to connect regional, national and global data centers; and,
- High DC resource utilization and increased operational efficiency, as well as reduced costs.

These benefits of DCI explain how high bandwidth connectivity with OTN networks using centralized SDN hierarchy, virtualization, effective data back up and VM migration is so useful. In the following sections, we discuss research already performed in SDN, SDON and SDN for DCI, and further explore some optical protocols and networking basics.

2.3 Software Defined Networking and Optical Interconnects

2.3.1 SDN basics

SDN is a leading networking paradigm, and it can address the limitations of traditional networking. It decouples vertical integration by separating the network control plane from the underlying data plane, thereby making the data plane simple forwarding hardware and uplifting the control plane to a logically centralized controller, thus simplifying network reconfiguration and policy enforcement. Communication between the data and control planes is via well-defined programming interfaces, and controller manages the data plane with Application Programming Interfaces (API) [14].

One of the most common South Bound APIs in SDN is Open Flow (OF) [14], [15], which is a southbound-based wire communication protocol that allows controller to configure the data plane using the simple packet forwarding rules (also called flows). OF architecture is comprised of three components: OF hardware, secure channel and controller. OF stores the flow rules in one or more flow tables or groups tables and uses a secured OF channel [16] to connect the external controller. The controller adds, updates and deletes the flow entries in the flow table, using OF protocol reactively and proactively. Each flow table in the hardware contains a set of flow entries consisting of match fields, priority, counters, instructions, timeout and cookies, as shown in Figure 2.1. Matching in the hardware for the flow entry begins at the first flow table, and continues to all the flow tables in the switch pipeline. If matching occurs, the instructions are executed while monitoring timeouts and cookies and updating the counters accordingly. Depending on the rules installed by the controller, the hardware can act as router, switch, firewall or any other role, including load balancer, traffic shaper or middle box [14].

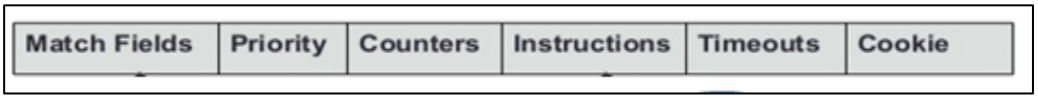


Figure 2.1 Flow entry in flow table.

Figure 2.2 presents a comprehensive survey of SDN, in which they cover the SDN infrastructure concept with a bottom-up, layered approach. It shows depth analysis of the hardware infrastructure, southbound and northbound interfaces, network virtualization layers, network operating systems, network programming languages and applications. Researchers in [14] also discussed cross layer problems such as debugging and troubleshooting the network.

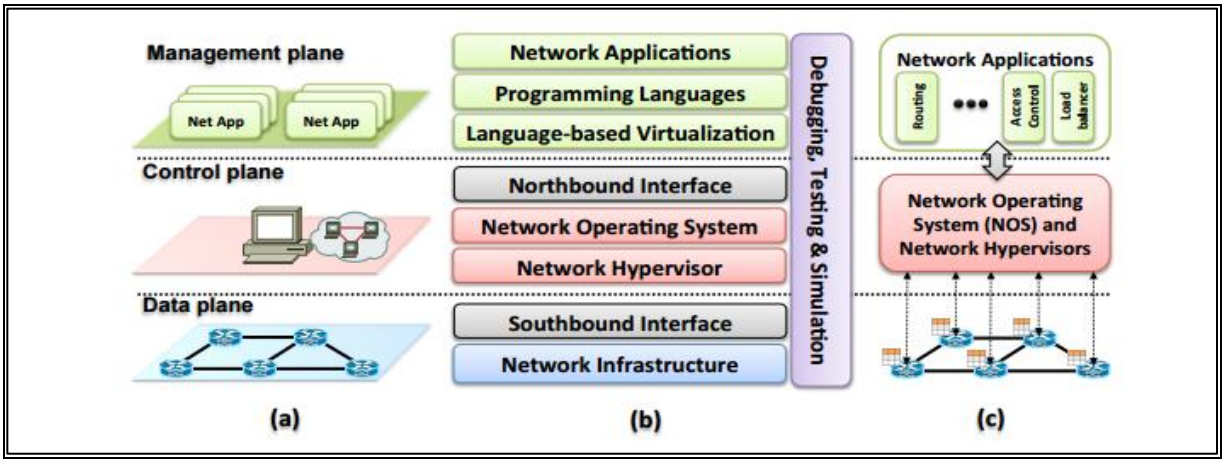


Figure 2.2 Trifold software defined network view in (a) Plane (b) Layers and (c) System Designed Architecture [14].

As mentioned above, OF is the widely-used networking protocol proposed by McKeown [16] to simplify network experiments on campus networks. This protocol is similar to other traditional routing protocols, and is based on forwarding via a flow table. ForCES, defined in [17], is also a southbound communication protocol, and is defined by separation of the control plane and the data plane for multi-controller and multiple network elements [16]. It provides an extended OF survey, and explains how OF can lead to network innovations. It even clarifies how OF enabled switches perform exceptionally well in smaller networks. To date, there has only been limited research on OF extension in WANs, and it has not yet been tested on real hardware in any industry. In addition to testing, researchers have not determined OF will be able to perform with scalability at the SPs layer.

2.3.2 Optical SDN

There have been a few studies to evaluate SDN workflow in the optical domain and the solutions to deploy SDN based multi-layer, multi-domain and multi-technology model to interconnect data centers. Authors in [18] presented a survey on software defined optical networks (SDON), and

explained how SDN can work in multilayer and multivendor situations. This study is an extension of SDON in the infrastructure layer using photonic communication. It explains the SDN control mechanism by describing how the SDN controller works with OF protocol, and how optical infrastructure layer components (i.e. transceivers, optical circuit and packet, and burst switches) are controlled. The authors also describe the insertion of an abstraction layer into the network elements, and the abstraction mechanism used by OF extensions for optical networks.

Along with the infrastructure layer, their work also discusses the control layer mechanism for virtualization in optical domains. Virtualization in optical layers is different for inside and outside the DC in the metro/core networks. SDON covers its usability in various business applications, including maintaining energy efficiency, QoS, fault tolerance, restoration, end-to-end QoS and video quality. In the latter part of the study, they describe the multi-layer, multi-domain orchestration for optical networks, highlight the open challenges to SDN in optical domains and outline their future directions.

Bhaumik et al. [19] present an overview of SDN and network virtualization concepts, and outline the principles for extending SDN and network virtualization in optical networking. Their work differs from [18] because they not only focus on the concept of optical SDN and its usability, but also the industrial effort done in this domain. They further discuss how generalized multi-protocol label switching (GMPLS) does not work in an SDN environment, and why OF is a perfect fit. They compare both these protocols, and ensure GMPLS is entirely covered by OF. The paper also covers the representative efforts done by the industries, and proves optical SDN is not just an academic idea, and will soon be commercialized.

Authors in [20] present a model of unified multi-layer architecture, with multiple controllers and a dynamic orchestra plane (DOP) for software defined multi-domain optical networks. It provides the overview of the access network challenges, such as the lack of support for on-demand modification of traffic transmission rules, limitations on the vendor-proprietary policies, rules and software, as well as how these can be addressed with SDN.

In [21], Zhang and Zhu propose an Integer Linear Programming (ILP) model, including several heuristics based on single-DC destination selection. They develop an optimal solution for small-scale problems, which explains how spectrum can be efficiently used for any type of cast in elastic optical inter-DC networks for SDN networks. They also demonstrate how their algorithm with

multi-DC destination selection in SDN is an excellent heuristic example of how optical interconnects can be used in inter-DC communication.

The authors in [22] explain how SDN can be used in optical transport technologies and cloud SPs to dynamically customize the infrastructure, and minimize capital and operational costs. They describe the control plane architecture in terms of integrated packet-optical networks using NOX as the SDN controller, and enable OF protocol on real hardware optical devices. They also verified the productivity of their architecture by performing cloud migration, and measuring the efficiency using optical SDN.

In [1], Chen and team discusses the optical transport network with an intelligent view of SDN, as well as the intelligence requirement gaps between current optical transport networks and the new trends of telecom applications using SDN. For this work, they studied research already conducted on building SDN/OF enabled optical transport network architectures and control plane platforms, and developed detailed algorithms and mechanisms to support and optimize optical transportation networks. [23] compares distributed and centralized control methods, and suggests an SDN application for a multilayer network. This application is designed so service providers and carriers can supply the required bandwidth between clouds and data centers, and virtualize the network. They extend their work in [24], and introduce a novel unified control plane for multi-domain and multi-transport networks, based on an SDN framework with OF as the protocol enabler.

In [25], Hong presents an efficient algorithm for mapping virtual optical network topology onto a physical multi-domain optical network. They focus on minimizing the total network link cost by dividing the problem into two. In the first part, they select an appropriate traffic analysis (TA) scheme, and in the second part they map virtual nodes and links to physical nodes in the various domains, and determine their corresponding physical link. Their research highlights how a candidate's location in the physical optical network can support efficient Virtual Optical Network (VON) planning and design for network operators.

Paper [26] applies the work done in [25], and proposes an auxiliary graph-based algorithm to map virtual infrastructures with survivability over T-SDN. This approach uses the modified Sourball algorithm to find parallel mapping of the primary and backup VIs over the physical substrate. In the algorithm, two physical nodes are found for a given virtual node. The graph is then constructed by adding a pair of auxiliary source/destination nodes and a set of auxiliary links, to connect

the auxiliary source/destination with the candidate physical nodes of the virtual nodes that belong to the virtual link. Their work with this algorithm improves the blocking performance of virtual infrastructure demands, as well as the traffic carrying capacity of the network.

[27] proposed a segment routing-based approach superimposed on SDN, to allow IP networks to benefit from these new features. Their controller core is based on MPLS for the IP layer, and Elastic Optical Network (EON) for the optical layer. Using SDN, a packet layer is superimposed on the optical layer, and both react efficiently when there are link changes in the optical layer to perform multi-layer communication. This maintains the resilience and reliability characteristics for link failures. For the optical link failures, when restoration has reduced the resource capacity the controller selectively overrides the MPLS infrastructure, and utilizes the remaining link capacities in another form. They also investigated the costs of the system in terms of the state of the network, the computational requirements of the controller and overriding the methods.

2.3.3 SDN in Optical WAN and transport layer - Industrial Efforts

Several researchers and industries are trying to transform the research of SDN in WANs and service provider layers, to develop solutions that use different types of platforms. Most leading companies have already launched their products for optical WAN communications, such as Fujitsu's Virtuora Controller, Cisco's transport SDN WAN solution and Nokia's Network Service Platform (NSP). These vendors are using TL1 as the major southbound communication platform for the controller to interact with the infrastructure layer. However, companies like Google use OF as the southbound protocol. They initially used the Greenwich technique, building half of their DC on OpenFlow (also known as the 'B4 SD-WAN solution) and the rest with traditional networking and later changed its entire DC architecture, connections and communications to the SDN model. Similar research has also been conducted at the academic level, with researchers developing their own ideas on how SDN can be used in the transport layer to improve existing infrastructures.

In this section, we focus on related work done by Google and few networking companies such as Ciena (blue planet), Nokia, Cisco and Fujitsu. First, we discuss the software defined Google WAN architecture. Their infrastructure layer switch hardware only forwards traffic, and does not have complex control software to interact with Network Control Server (NCS) hosting the OF Controller and Network Control Applications [28]. Google adopted this hybrid approach to extend

the support for existing routing protocols using their novel traffic engineering services, and later gave the control of their infrastructure hosting to NCS.

Cisco also developed solutions for SD-WAN services. For example, their IWAN solution is built with the SDN architectural approach [29], which enables network automation to allow virtual service models to deliver any application or service, regardless wherever the user is located. SD-WAN is also being extended to Cisco's other product range of application centric infrastructure controllers (APIC) and infrastructure layer access edge switches.

Fujitsu also uses SDN to create a programmable, automated network for optical layers [30], [31]. Their software suite product 'Virtuora' brings disparate systems and technologies together under a common management system. Virtuora is an open-source, open-platform controller that can easily be switched between Open Day Light (ODL) and Open Network Operating System (ONOS) controllers. It abstracts the underlying forwarding plane and delivers services corresponding to the abstractions, which improves the overall use of the network and reduces the delivery time. Virtuora [30] is a multilayer controller capable of IP, WDM and OTN networks with multivendor infrastructure layer approaches. However, for OTN they are restricted to transaction language 1 (TL1) only, and do not support OF.

In [31], they tested seamless service migration with Virtuora while optimizing the network for optically interconnected DCs. They also recently proposed a novel ILP-based approach to minimize connection disruptions while optimizing resource utilization for a re-optimization process. Their approach clarifies how an intelligent 'network re-optimizer' can be added to operators' SDN platforms and perform automatic network re-optimization, periodically or when triggered by service migration events. This approach defines the resource migration and connection migration patterns for service re-optimization, and interrelates them with the ILP algorithms it uses. Their evaluations are based on the results of the average number of disrupted connections per re-optimization, and its connection arrival rate.

Nokia's SDN WAN solution, called 'Network Service Platform (NSP)', guarantees service automation and network optimization, and dynamically ensures delivery [32]. NSP guarantees customers greatly reduced time for on-demand IP/optical network services by making optimal use of available assets. They commit to maintaining efficient on-going network services, and providing high network efficiency without manual intervention. To date, available NSP statistics and cases

have shown it is the most automated on-demand IP and Ethernet VPN service over integrated IP-optical networks with established dynamic label-switched path (LSP) [32]. Nokia is also well known in the global market for innovative product strategy for carrier SDN.

In addition, another group of researchers from Infinera and the Energy Sciences Network have proposed a transport SDN model that blends seamlessly with SDN within the DC [33]. In their work, they abstract the core transport node into the programmable OpenFlow virtual switch, and demonstrate its use for big data applications. They also explain the architecture and design of the open transport switch, and define its mode of operation in both explicit and implicit modes.

Authors in [34] present a hybrid electro/optic interconnect model to provide high performance computing for the DC. They describe their architecture as having two modes: it remains in the first mode until the system has limited prior knowledge of the applications running on it. In this mode, they perform periodic control process monitoring of the live traffic, and configure the optical network to fit the traffic pattern. Whenever there is a need for on-demand bandwidth, it automatically switches to the second mode of operation. Also, as per their architectural design, both modes can operate simultaneously. They use VLAN based forwarding instead of OF based forwarding, by making use of their own MEMS and ToR Ethernet switches.

2.3.4 SDN in Optical WAN and Transport Layers - Academia

As concentrated industrial research in software defined optical WAN and transport layers, much research has also been done by students and researchers. Academic research began with the investigation of bulk data transfer between datacenters, and proposing SDN transport for multi-stratum resource integration of SDN for inter-datacenter connects [35]. It then focused on cross-stratum optimization for ubiquitous datacenter interconnections [36], and continued to improve data transfer [8].

The authors in [35] describe their concept of using three distributed SDN controllers, one each for IP and optical layer and the third for the application layer. They use the controllers in multi-stratum resource integration resilience architecture (MSRIR), in which the controllers are physically and logically distributed. The communication flows from the OF based IP controller to the optical controller, continues to the application controller, and then flows back to the IP controller. This communication is based on the multi-path, global resource integrated resilience algorithm (GRIR),

interacts with the SD-OTN network and the optical controller, then continues to the application and IP controller. They verified their work using the same test bed as in [36], and proposed the joint optimization of the IP network, EON and datacenter application stratums and resources, and measured the end-to-end services. In both [35] and [36], they tested the performance of their proposal on the OpenFlow-based enhanced SDN (eSDN) control plane, according to parameters such as blocking probability, resilience, latency and resource utilization, and compared their work with other resilience algorithms.

In [37], Hui Yang proposed architecture for heterogeneous cross-stratums and multi-layer networking modes, and called it SDN for ubiquitous data center optical interconnection (SUDOI). This model implements cross-stratum optimization of applications and optical networks, to enhance the multiple layer resource integration in the ubiquitous datacenter optical interconnection. In this work, they investigated the interlinking procedure between user-access-oriented interconnection, multiple-layer resource integration, inter-data-center and intra-data-center service modes.

SUDOI architecture [37] is interesting and unique due to its different distributed controllers interacting with each other via their interfaces, computing different tasks and performing diverse duties. Their architectural model for datacenter interconnection consists of optical, transport, application and IP controllers. It starts with the optical access controller interacting with the access network optical network units (ONU) via a controller access interface, and continues to IP controller interacting via IP optical interface. The optical access controller processes the network information to the IP controller, and performs flow monitoring, flow scheduling and resource organization. The IP controller is built with resource integration control, flow table control and IP flow monitoring, and these modules only interact with the IP network in the optical transport network via controller network interface. Resource integration control of the IP controller interacts with the resource integration agent of the transport controller, and passes the information to spectrum control and monitor. They direct the flows into the optical transport network, using path computation and network abstraction plugins.

In [8], Xin et al. proposed Optimized Wide Area Network (OWAN), a unique traffic management system that optimizes wide-area bulk transfers with centralized joint control over both optical and network layers. It changes the network layer topology by reconfiguring the optical devices

according to network needs. Though most of the bulk transfers are not delay sensitive, however there are few services (e.g. HD-video) with deadlines, and must be sent at higher speeds to ensure success. Therefore, it is necessary to schedule these transfers to meet deadlines and SLA requirements. A similar approach is used in our DNS application, where scheduled bulk transfer combinations of smaller OPU, ODU's and ODU, are flex wrapped together in the interconnects container to perform end-to-end transfers.

OWAN design is proposed for tuple bulk transfer requests to a controller with a globalized view. Based on the available requests, the controller computes the optical circuit path and builds the topology with path computation. It computes the network states using a random neighbour algorithm, computes the energy and updates the network states. They evaluated their proposal with actual routers in a nine-node topology, and considered both deadline-limited and deadline-unlimited traffic. The results are measured in terms of performance to traffic load factors, as well as improvement of compilation time for internet, ISP and inter-DC networks.

2.3.5 Time and Latency Aware SDN

The State Key Laboratory of Information Photonics and Optical Communication, and Beijing University in collaboration with Huawei Technologies in China have done a lot of research on time aware service scheduling (TaSS), time enhanced service scheduling (TeSS) and latency aware service scheduling (LaSS). Researchers in [38] proposed a time-aware service scheduling strategy to allocate the network and datacenter resources, with scheduled flexible service time and bandwidth. In the model, they used different controllers for the transport and application layers, and evaluated the performance by measuring the blocking probability in the network.

Yang et al. [39], [40] worked on time aware OF based SDN controllers for inter and intra datacenter optical interconnection as well. The proposed model has different controllers for the application and transport layers, and uses cross stratum optimization of the network. Their work included how to modify datacenter services for specified fixed bandwidths, as well as providing services with less bandwidth. To address these objectives, they proposed a time aware SDN algorithm that extends the OF protocol in transport layers and uses OFPT_FEATURE_REQUEST and OFPT_FEATURE_REPLY, then evaluated their work in terms of blocking probability and OF Wireshark captures.

Yang [41] proposed a latency-aware software defined networking (LaSDN) architecture for OF based optical interconnects in intra-datacenter networks using latency-aware service scheduling (LaSS) strategy. This organizes the application according to the latency priority for the required QoS. Unlike our work, they focused on intra DC only, and did not consider inter DC. They evaluated their results on real open-flow hardware devices and specific distributed controllers for applications, IP and transport layers.

2.3.6 Centralized SDN Architecture

SDN controllers are divided into two major categories, centralized control and distributed control, based on the control maintenance of the network. They are equally efficient in both centralized and distributed architecture, and have their own pros and cons.

With centralized control, controller is the single entity that manages the forwarding devices and interacts with the applications. It has a global view of the entire network, and acts as the single point of failure. These controllers are designed in a very concurrent manner, to achieve the high throughput requirements of enterprise class customers or DCs, and are based on multi-threaded designs to explore the parallelism of multi-core computer architecture [14]. They can manage more than 20 million flows per second using large size computing nodes.

Centralized architecture is ideal for Overlay SDN solutions, as it can manage overlapping IP addresses between multiple tenants, and supports multi-path forwarding. They are the best fit for network virtualization, and supports an unlimited number of virtual networks in 24 bit addresses that are typically used by network overlays to identify VNs. They can identify slightly more than 16 million VN IDs. Centralized SDN technology evolved from research aimed at central control of forwarding. With centralized control and a global view of the entire topology, network addressing and connectivity is easier for end users.

2.3.7 Distributed SDN Architecture

Distributed SDN architecture is made up of multiple network operating systems with distributed control. Distributed control is defined as either a centralized cluster of various nodes, or a physically distributed set of elements. Centralized clusters of nodes are typically found in dense DCs and offer high throughput, while distributed sets of elements are used when there are different

types of logical and physical links. These clusters or elements interact with each other via east/west interfaces.

Distributed control has weak consistency semantics [14], and with them, every update on any distinct node must be eventually applied and updated to all the controller nodes. This implies that there may be interval off times, when different nodes could read different values for the same property. However, this also ensures that all the controller nodes read the most updated property value after the write operation [14]. Distributed controllers are fault tolerant; regardless of the failures they take on the responsibility of the failed node.

Cloud providers or service providers spread across a virtualization of multiple DCs interconnected regionally or internationally, require a hybrid approach of both centralized and as distributed architectures. In our proposed SDSOI model, we recommended a combination of centralized and distributed SDN controllers. The centralized controller provides multi-layer, multi-domain control across the IP-Optical IP-layer, and the distributed architecture gathers the information about the network status from distinct, geographically distributed DCs.

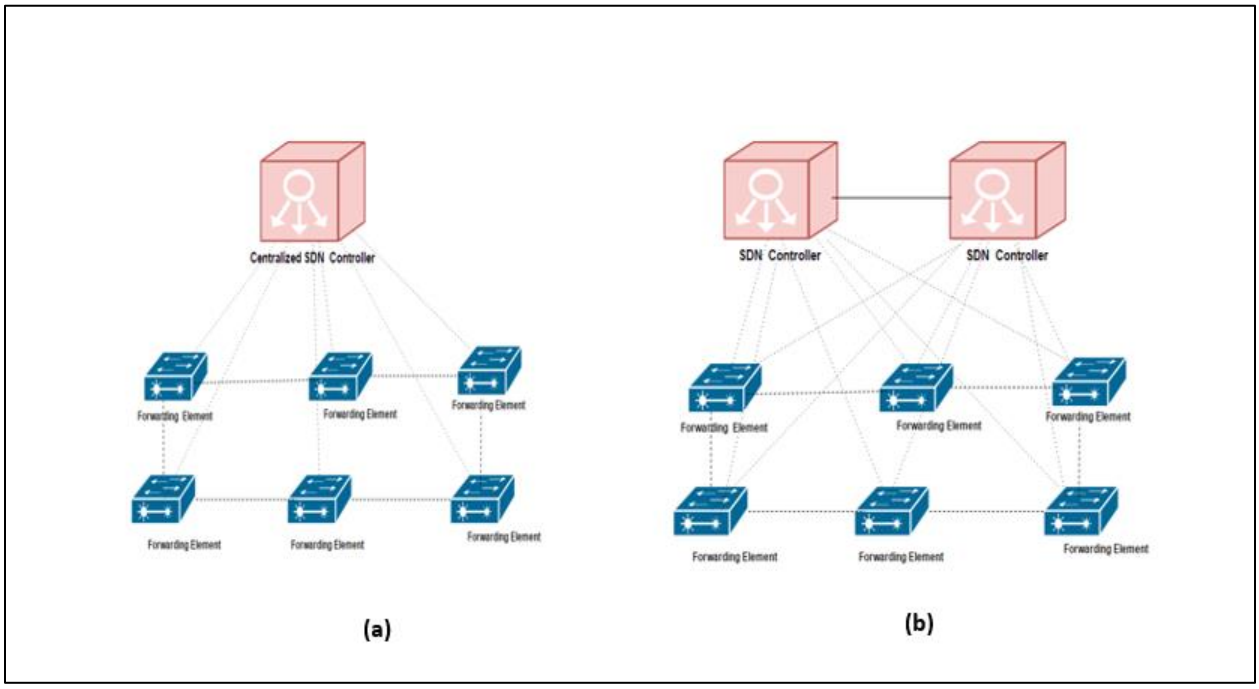


Figure 2.3 (a) Centralized SDN Control, (b) Distributed SDN Control

2.4 Optical Transport Network (OTN)

OTN [42] is the next generation optical transport technology used for high bandwidth transmission. It supports mixed rate client signals on higher rate wavelengths and switches, and its network element serves as a direct interface to DWDM systems for all wavelength range clients.

Non-OTN client circuits ranging from gigabit or higher rates of Ethernet, SONET/SDH (as described in appendix A) or video are wrapped, groomed and mapped together according to the OTN standard. A brief overview of OTN technology and its hierarchy is provided in Appendix B.

In traditional networks, each packet and optical layer has an individual administrator, database (DB) and network management system. Whenever a change or failure occurs in the network, the administrator must correlate the changes to the database, manage the recovery of each layer and perform advanced negotiations. To overcome the limitations of traditional networking, Fujitsu extended the OF protocol to support OTN in open networking foundations, allowing them to abstract the optical and packet network resources and process them in a unified manner [3] to achieve a single layer of failure. Extension of OF in the optical layer provides multi-layer control via a unified interface, and focuses on deploying rapid services with reduced CAPEX and OPEX.

2.5 Survivability in Optical Networks and SDN

Optical fibers have the potential to carry large volumes of data in a single wavelength channel, ranging from 40 GB/s to 400 GB/s. Managing high-speed critical data requires faster recovery from traffic failures, to ensure survivability of the network and maintain traffic continuity. With regard to protection and restoration, preserving data survivability requires recovery techniques based on some method of assignment and reservation of recovered resources. Survivability in optical layers is critical, as optical failures in lower layer can result in multiple failures at higher layers. To explain this, we first discuss the difference between restoration and recovery, since many readers think both terms mean the same thing.

Recovery of a network is defined as the process of enabling it to return to an operational state after a failure. Recovery can be applied at various levels in a network, and to forms such as path, link or segment. Path recovery means an entire path between the source and destination nodes is recovered, while with link recovery only the path or link that failed in the network is recovered

with an available alternate sub-path, and rest of the path remains the same. In the case of segment or section recovery, only the complete portion of the network is recovered. Recoveries are provisioned using backup paths whenever the active or working path fails. A working path is the active light path used for transporting data during normal operation, and a protection path is the new path defined to protect the working path when it fails. The protection path ensures that there is no compromise between the class of the service and the QoS.

Restoration is the process of establishing a backup path when the working path in the network fails. Restoration is a reactive strategy, in that it recovers affected traffic and quickly reinstates service continuity, whereas protection is a proactive strategy performed before a failure, to defend the network against any possible disruptions. Protection includes failure detection and localization, and has a signalling mechanism to notify the corresponding source element that a failure occurred. Restoration and protection are correlated by their actions for recovery. In traditional optical networking, protection schemes are classified as follows:

- 1+1 dedicated protection;
- 1:1 dedicated protection;
- 1: N shared protection;
- M: N shared protection; and,
- Shared mesh protection.

To maintain survivability in an optical network, the network capacity is generally overbuilt with respect to the average traffic volume of the users and support fluctuations in the traffic levels. Ideally, for survivability the protection path corresponding to the working path is built across the disjoint or distinct paths, so that when a failure occurs data is switched to the alternate protected path and the loss of service is imperceptible.

Traditional networks are provisioned with preplanned protection or restoration; that is, there is always an allocation of dedicated or shared paths available for them. Survivability in traditional networks requires time to create new services and address changes in the network and is considered an expensive option, because depending on the kind and type of connection created, corresponding bandwidth is always dedicated to the protection path, regardless of it is required or not.

However, with next generation networking, programmability, automation and orchestration is on board, and SDN controllers are smart enough to rapidly deploy service changes according to their programming, while also considering factors such as protection, latency and priority. When failure occurs protection paths gets activated, and they restore the traffic to maintain continued bandwidth services for both protected and unprotected paths. For protection in SDN, controllers are programmed to react proactively according to their installed policies, and they provide service quickly enough to make traffic failures unnoticeable. Even for unprotected paths in SDN, recovery time in the network is also short, since the SDN controller actively performs neighbour discovery, monitors the liveliness and rapidly (in ms) sends the failure information to the controller, and the controller instantly updates the flow table with new flow rules and routes the new traffic flow.

Academia and industry have recently begun to focus on survivability, failover and restoration in the SDON, in multilayer models as well as optical layers. Research is being conducted on dedicated protection in transport SDN (T-SDN), multilayer repair in elastic optical networks using SDN, 1-1 end-to-end protection using SDN multi-tier control planes [43], optimization of flow operations for multi-failure restoration scenarios and disaster recovery in T-SDN[44]. A brief overview of the work on optical protection and restoration follows.

For any rapid service change in a network, abstractions of the optical and packet layers should be effectively mapped to the virtual infrastructures on the physical hardware, and survivability is one of the biggest challenges for this mapping. Work done in [45] addresses mapping by dividing the problem into sub-problems of modulation selection and spectrum allocation in the process of provisioning optical channels to support virtual links while meeting optical layer constraints such as transmission reach and spectral continuity. They proposed parallel VI mapping (PAR) to provide dedicated protection for any single physical node or link failure.

PAR is designed using a modified Suurballe's algorithm [10], and is designed to find a parallel mapping of the primary and backup VIs. PAR offers dedicated protection against single physical node or link failures. The authors verified the PAR algorithm performance in transport SDN, in terms of blocking probability and traffic-carrying capacity with respect to the baseline sequential VI mapping approaches.

In [44], Blendin at el. propose the unique idea of gradually superimposing SDN for multi-layer repair in elastic optical networks. Currently, packet layers cannot make use of all the features

offered by modern optical communication. Thus, the authors suggest using a traditional network for modern optical communication, and SDN when survivability is required. For example, for a cut fiber in the network, an optical protection mechanism restores the original IP topology after a short transient interval. This optical restoration takes new, longer light paths, and can affect the capacity of the link, flexible optical transceivers utilize the remaining optical capacity efficiently by adapting to the network link capacity accordingly.

However, for certain cases it may not be possible to perform the optimum link capacity utilization because the packet layer cannot cope with fluctuating link capacities; this can cause a shutdown. To avoid such shutdowns, the authors in [44] proposed a segment routing-based approach with superimposed SDN that allow the IP network to benefit from the superimposing features. To verify this, they performed a gradual deployment of the system while keeping other proven resilient technologies and systems unchanged. This technique is ideal for deploying SDN, as it not only provides protection, but is also a complete multi-layer and multi-domain solution. Their approach is still being researched and tested in industry, as OF extension in optical layers is not as efficient as TL1 or other IP-Optical protocols.

Authors in [43] proposed a model for building an end-to-end, 1:1 protection scheme in a combined LRPON access and core network, based on a multi-tier control plane using SDN. They included an additional open flow component called an OF-relay on board the hardware OF switch [47]. The OF relay rapidly updates the local rules to the OF switch, and forwards the alarm to the higher layer control infrastructure. These open relays allow protection events to be relayed directly from working to secondary paths.

In addition to scheduling interconnections, this research focuses on applying proactive protection to ensure survivability of the optical interconnects that link datacenters together. We also investigate how we can use SDN to provide protection for unprotected networks.

Chapter 3: Architecture

3.1 Introduction

This chapter provides brief descriptions of the Transport SDN concept, and the extension of OF protocol for the optical transport layer in our proposed SDSOI model. It describes the complete architecture of the proposed model, including the infrastructure layer optical gateways, the SDSOI controller core, the unique application categorization and API descriptions and intents. Later in the chapter we describe our DNS application, which optically interconnects DCs via optical gateway networks by scheduling and protecting the traffic flow.

3.2 Transport SDN

Transport SDN is characterized as a highly scalable, flexible and programmable network model, with well-defined, technology agonistic, design functional entities, control and management [48] [49]. It is under the umbrella of the Open Network Foundation (ONF) Optical Transport Working Group (OTWG) [50], that describes its architecture and trends.

Transport SDN with centralized intelligence can be effectively achieved when the underlying network is fully programmable at every layer, and is capable of automation and orchestration of services. To provide centralized control in each layer, transport SDN extends the OF protocol to support the Layer 0 (photonic) and Layer 1 (SONET/SDH, OTN) networks with independent software and hardware development. This requires OF protocol extensions to support fibers, wavelengths, timeslots and packet headers, and to maintain the fundamental OF match and action tables across the multiple layers [51].

Transport SDN can abstract power and offer real-time Transport SDN network operations by allowing network controllers to bring up, re-route and tear down the optical links for physical plumbing of a wide area network; unlike traditional transport networks, where once the links are brought up they remain online throughout. These controllers abstract network power, and provide real-time, on-demand programmable bandwidth service models. The concept appears to be simple, but implementation is very complex due to the extremely large size.

Transport SDN networks have various applications, including being used by carriers' carriers for fast service provisioning when there are numerous vendors involved, in mobile backhauls with a

high volume of services and changing bandwidth and vendors, for datacenter interconnects for faster service provisioning, and for Global tier 1/2/3 networks with high numbers of services. The Transport SDN controller is a an intelligent device capable of handling the level of detail required, minimizing the latency of long haul transport links or optical interconnects, and optimal bandwidth management. A brief overview of transport SDN architecture and its operations is shown in Figure 3.1.

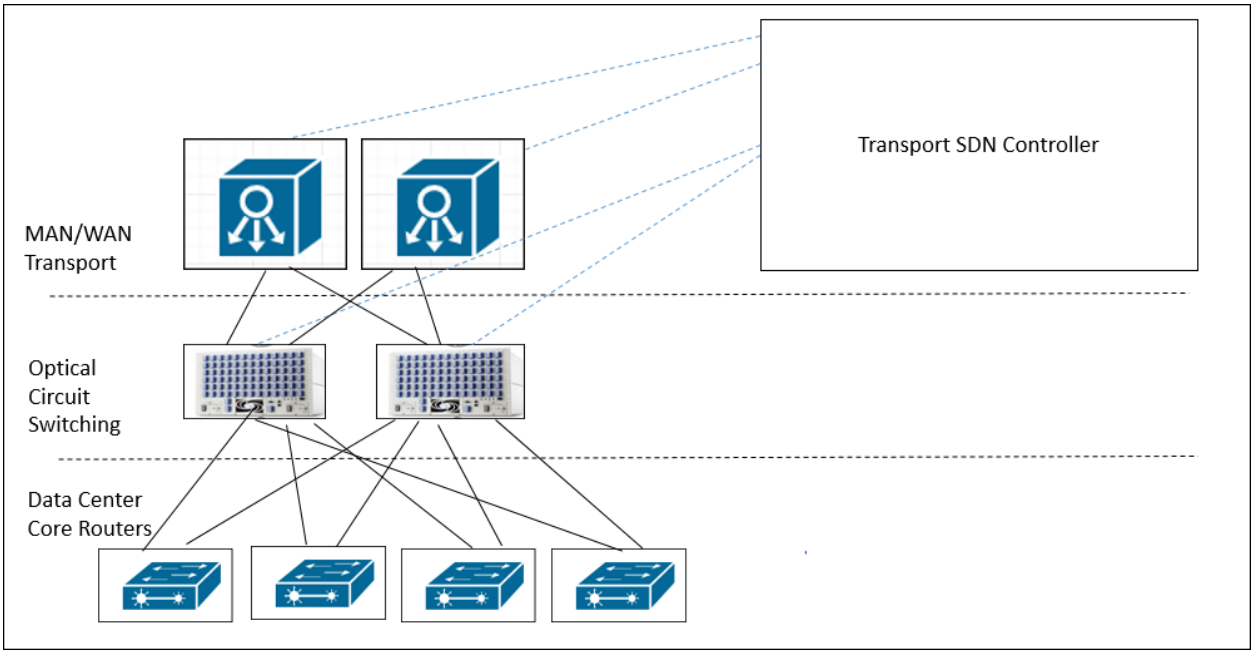


Figure 3.1 Transport SDN architecture.

3.3 Open Flow Extension

Until now, optical equipment in companies like Fujitsu, Nokia and Ciena are controlled by Transaction Languages 1 (TL1) protocol [52-56]. However, for multi-layer packet and circuit-switched networks, it is necessary to extend the OF protocol to every existing network element while maintaining their capabilities, and make them ready to use SDN in sync with other infrastructure devices. In the SDN era, maintaining multi-domain and multi-layer capabilities with SDN requires extending OF to every network element.

OF protocol, as described in Chapter 2, is a southbound protocol used by the SDN controller to communicate with the infrastructure layers via API. OF is the one of the first to provide commoditized abstraction between a centralized control layer and the underlying network to

virtualize the policy enforcements of applications. It works on the match/action strategy of the flow rules in the flow tables installed by the controller in Chapter 2. To extend the OF protocol to support OSI model Layers 0 and 1, it is necessary to match the optical parameters in the flow table, such as bandwidth, time-slot and wavelength. After matching these parameters, the transport layer controller can immediately take the actions necessary, including allocation of on-demand time slots and cross-connects between the DCs.

In the SDSOI model, we extended the OF protocol to support next generation OTN standard. As discussed in Appendix B, OTN consists of Optical Packet Units (OPU), Optical Data Units (ODU), Optical Transport Units (OTU) and Optical Channel Hierarchy (OCh) layers, so it is crucial to extend the OF protocol in the OTU and OCh overhead main parameters. We extend the OF protocol to support the match/action strategy in OPU, ODU, OTU and OCh overheads and their flow rules, and add optical parameters to the flow rules and flow tables.

The OF protocol in SDSOI is also extended by allocating and supporting Tandem Connection Monitoring (TCM) level fields in the flow table and flow rules. TCM in SDSOI extends the SDN controller monitoring of third party domains or sections of the interconnections. TCM in OTN and SDSOI helps install interconnect protection for both, whether within or passing through a domain. Enabling TCM fields in the flow rules from the controller to the optical gateways to be interconnected ensures survivability of the interconnects for failures while traffic is flowing from the service provider domain, or when it enters another intermediate provider domain.

Figure 3.2 explains the basics of the OF protocol used by the communication mechanism in the SDSOI model to exchange flow rules from the controller to the infrastructure layer. To initiate the process, the controller requests session establishment to the infrastructure layer or network of optical gateways, and the gateways reply with *Port and Features* information. This procedure is known as a ‘handshake’ between the controller and the gateways. Once the session is established, the controller requests discovery adjacency from the gateways, and updates the information it receives back. The controller then establishes the OTN connections for the optical gateways via *OFPFC_Add*. Every new request to the gateways from the DC is pushed to the controller with *OFPMP_FLOW Update Request*, and the controller replies with the *OFPMP_FLOW Update Reply*. The controller continues to send echo requests to the gateways and connected datacentres to ensure the devices below are ‘live’.

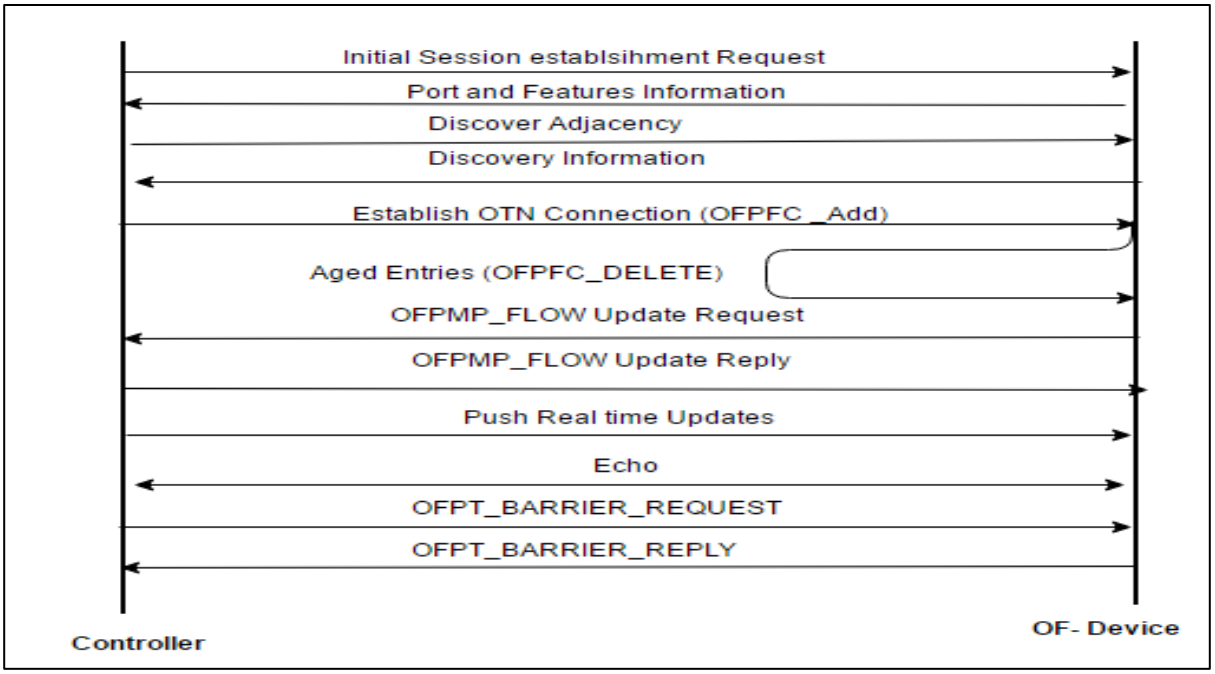


Figure 3.2 Session establishment and Flow mode in SDSOI OpenFlow Protocol.

SDSOI supports OF for on-demand counter changes and is built with the most stable *OF Version 1.3*, which includes the multi-part messages. For service providers, it is necessary that the SDN controller support encoding of message requests and replies of large volumes of data simultaneously; OF version 1.3 has this capability. Each sequence of OF multi-part messages carries one multi-part request or reply. These messages are primarily used for requesting statistics or state information from the switch. Requests and reply messages are exchanged between the controller and the gateways using *OFPT_MULTIPART_REQUEST* and *OFPT_MULTIPART_REPLY*. Multi-part requests and replies are defined in terms of a single structure, or as an array of statistics with the request or reply included in the body. Messages from multi-part requests or replies are interwoven with OF message types (requests or replies) and have distinct transaction ids, as shown in Section 5.2.1. We present our results in Chapter 5, and show how every interconnect request and message flow from optical gateways to the SDSOI controller, and vice versa, uses the following OF messages:

- OFPT_MULTIPART_REQUEST_OFMP_PORT_STATS,
- OFPT_MULTIPART_REQUEST_OFMP_FLOW,
- OFPT_MULTIPART_REQUEST_OFMP_TABLES,

- OFPT_MULTIPART_REPLY_OFMP_PORT_STATS,
- OFPT_MULTIPART_REPLY_OFMP_FLOW,
- OFPT_MULTIPART_REPLY_OFMP_TABLES.

3.4 Proposed Software Defined Survivable Optical Interconnects (SDSOI) for Data Centers

Most of the transport layer optical devices in today's world that carry the data, have their control and data plane planes decoupled from each other, though they still exist in the same chassis. Despite being in the same device they are configured but not programmed. Decoupling the control plane from forwarding has several benefits, however, academia and industry have questions: Will they be able to decouple the control from their existing devices and provide same level of features and support, or do they have to bring the new ones? Can the existing devices support OpenFlow? Will the new OpenFlow devices be able to support existing features?

With SDSOI, we aim to move this decoupling away from the chassis and into centralized control. In our architecture, this chassis is known as optical gateways and decoupled centralized control is known as an SDSOI controller. SDSOI architecture is modelled on the SDN control of transport layer network resources, which expose the SDSOI core to abstract network resources, and their status inside the DC via DCs respective SDN controllers and optical gateways. Like other SDN frameworks, SDSOI defines its network architecture where the network control is decoupled from forwarding plane and is directly programmable.

SDSOI architecture design considers several factors such as real time challenges, industrial needs, growing demands and scalability. It also addresses various DCs communication challenges, including on-demand bandwidth, VMware motion, data replication and bandwidth overprovisioning. SDSOI uses a pragmatic approach, and supports the highlights below:

- Network programmability that leverages the existing installed base/investments (this equipment is expensive, and new software updates can provide new features);
- Simplified multi-layer and multi-vendor control;
- End-to-end application awareness;
- Increased network efficiency;
- Commonality in heterogeneous infrastructure layer deployment;

- Dark fiber lease support;
- Multiple high bandwidth support (100G or more);
- Multi-layers abstracted to single end-to-end connections;
- Automation and orchestration management;
- Increased network utilization; and,
- Reactive SLA based pricing, features and support.

SDSOI is a concept that can be described as defining associations between gateways and interconnects in the network to the connected DC in the underlying network. It is a combination of centralized and distributed processes in which the SDSOI controller has centralized control of the distributed SDN controllers via intermediate optical gateways. Thus, the SDSOI core has centralized global control of the entire network across the globe, and it resides on top of the underlying DC networks with SDN control. These DC networks are interconnected to each other and the controller via a network of optical gateways (e.g. DCs communicate to each other via an east/west interface of optical networks).

Infrastructure layers are comprised of optical gateways that can be in the form of IP-Optical gateways, packet switches, ROADMs (Reconfigurable, Add and Drop Multiplexer), optical network gateways such as transponders, muxponders and routers, or combinations of all of these. The optical gateways connect the multilayer DCs with their SDN control. SDSOI architecture is a combination of both Overlay and Underlay SDN models, where the SDSOI controller core is part of the overlay architecture and simply forwards the data, without detailed inspection, by creating tunnels. That is the SDSOI controller initiates the interconnections without information about what is in the underlying network, or who owns it. However, the DC SDN controller connections via optical gateways are based on the underlying SDN model, so the DC underlay controllers have full access and control of the entire network in the DC, and have direct links to support multi-layer control from wavelength/OTN to MPLS/IP.

SDSOI is different than other service provider SDN architecture, due to it's already programmed and installed unique high level and low level inbuilt plugins within the core. These are the basic modules required for any typical DC interconnection. The SDSOI core's high and low-level plugin modules are shown in blue and yellow respectively in Figure 3.3.

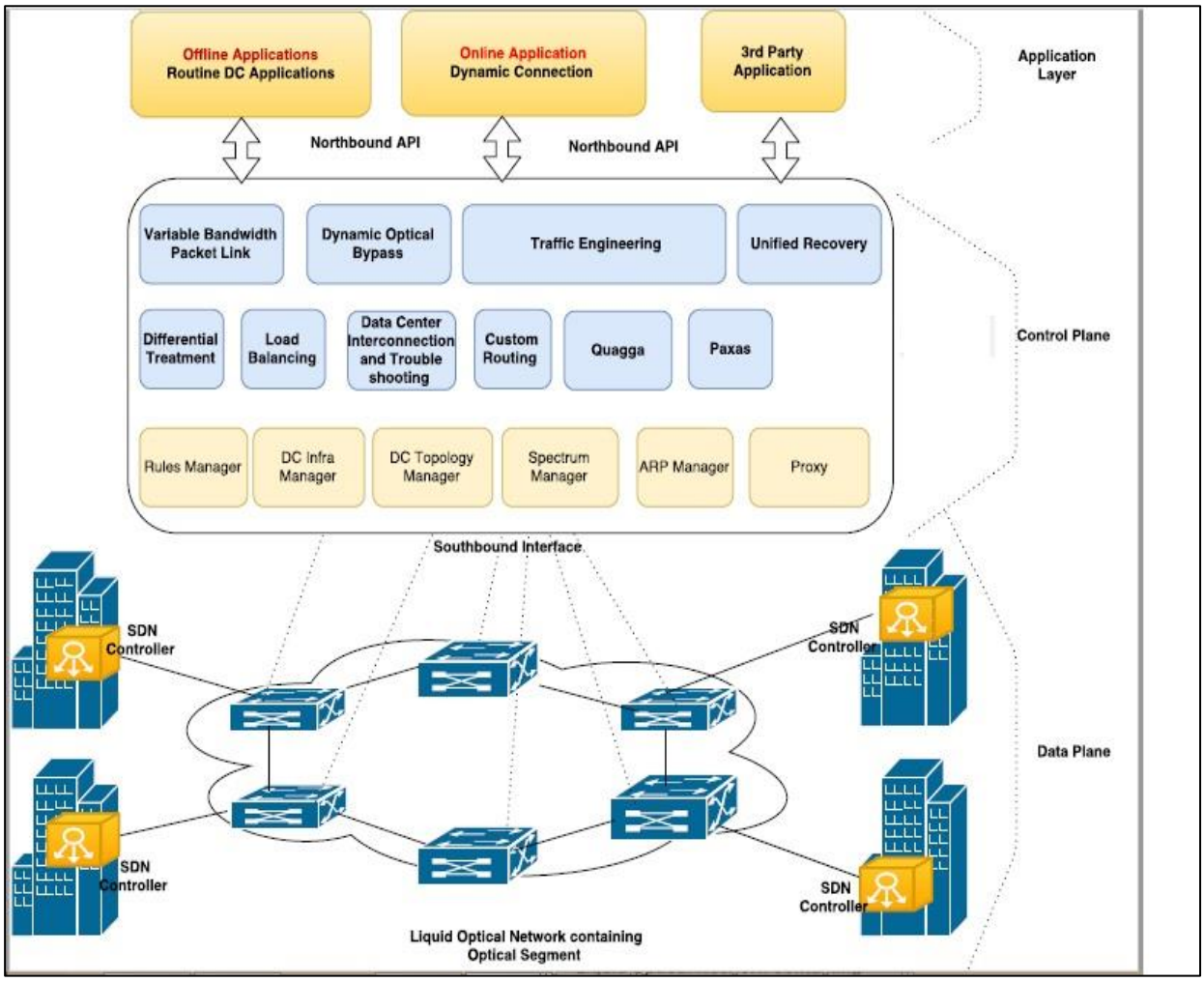


Figure 3.3 Proposed SDSOI Model Architecture.

Application layers in the SDSOI model represent clients, and communicate via north bound interfaces to the SDSOI controller core acting as a server. This client server model in the transport layer smoothens inside the DCs with SDN underlay model by virtualized switching and common management. It also mitigates outside the DC with the efficient circuit transport switching and on-demand, optimized bandwidth usage that are part of the overlay model.

SDSOI is an application driven architecture with an intelligent transport layer SDN control model. For the SDSOI controller to work effectively, smart business applications that supply it with policies and rules to define its behaviour are required. SDSOI models work in hierarchical client server models in which applications are the clients that interact with the SDSOI core or 'Black Box/Server'. The SDSOI core then performs the complex computations and installs the flows and flow table in the infrastructure layer. Infrastructure layer is made up of optical gateways and the underlying DC network connected to them, which can be physical or virtual.

We categorize SDSOI applications into three broad categories: offline or routine DC applications, online or dynamic applications and third party applications. Offline applications, also known as stable applications, are required by the DC for normal day-to-day operations. Examples of offline stable applications include Policy Enablement, day night work and bandwidth scheduling, Inventory and Performance Management, Configuration and Fragment Management and analysis. Online applications are known as dynamic applications, and are responsible for the creation of on-demand services when failures or changes occur. Dynamic applications include path calculation, backup service activation/restoration and network discovery. Third party applications are applications supported by the controller that are not programmed for the controller, though they support it. Third party applications include applications such as Microsoft Office, Confluence, Jira and many others.

The SDSOI controller, which usually resides in a cloud, and has the intelligence to know when to use which application, and among which DCs. There are cases when some DCs require only a particular type of application and others do not. For these cases, the SDSOI controller uses its inbuilt plugins and added programmable features to provide the required services to only those DCs. This process is similar to creating a Virtual LAN (VLAN) in traditional networks, where users with the same characteristics are grouped together and receive the same services. The only difference with the SDSOI approach is that the computation of the types of applications and services is done in a highly programmable, automatic and orchestrated fashion.

The SDSOI core interacts with optical gateways using southbound protocols and interfaces. As explained in Section 3.2, we extended the OF protocol in transport layers, rather than using TL1 or any other southbound communication protocol and restricted our work to OF as the southbound communication protocol. This allows our solution to work in multi-layer, multi-vendor open

devices or white boxes, and to maintain *openness*, *programmability*, *automation* and *orchestration* of the controller. We extended the OF by adding OTN overheads to the OF protocol.

The SDSOI core interacting with the optical gateways ensures delivery of the flow rules containing TCM fields (higher or lower) as well as other IP optical fields. Optical gateways match these fields in the flow rules, and take the corresponding step of forwarding the traffic. When optical gateways, cannot match the flow rule fields, such as TCM or any other, they pass the message back to the controller. Controller rediscovers the network by actively monitoring its network elements and those of its neighbors, and performs path computation according to the client's application intents. After the controller computes the necessary actions based on application design and policies, it resends the flows back to the optical gateways and the communication proceeds.

In the next few subsections, we focus on the Applications Layer and its categorization, Optical Gateways, and explain the key model of the SDSOI core.

3.4.1 Open Flow Optical Gateways

Infrastructure layers are comprised of optical gateways that can be in the form of IP-Optical gateways, packet switches, ROADMs (Reconfigurable, Add and Drop Multiplexer), optical network gateways such as transponders, muxponders and routers, or combinations of all and occasionally the basic control unit for all the cards or blades in the chassis. The optical gateways use different cards for DCI switching, routing and amplifying. They are connected to the core of the DC's SDN controller by layer 3 routers; these gateways are the OF enabled devices and are not related to TL1 protocol. They have a simple controller component inbuilt with the OF control channels and the data paths, and store the flow table entries in a pipeline.

Optical gateways are instructed and controlled by the SDSOI controller via open southbound interfaces, using the OF flow rules on how to function in the network and perform routing. Open optical gateways are designed as 'dumb devices', and support OF transport layer extension with features such as multi-part requests and replies, flow stats, ports and tables. These dumb forwarding elements are responsible for multiplexing OTN's different types of ODU, OPU demands into single interconnects, and transpond them accordingly. The SDSOI controller adds, updates and deletes the flow entries in the optical gateway flow tables, both reactively and proactively for unprotected and protected interconnections respectively. Flow entries consists of

match fields, counters and a set of instructions to apply to the matching packets. TCM is used for monitoring the OTN interconnects, and is added in conjunction with the match field in flow rules.

Optical gateways are used in the network to make optical external connections, and to interconnect the network of geographically distributed DCs using controller card ports. The controller card logical ports help model the control flow rules between the devices, and support the connection of multiple wavelengths. Physical or virtual links in real hardware or software network emulators (e.g. Mininet) describe how the gateway optical ports are connected to each other. Controller card ports use different datapath_ids to make them easily distinctive to the OF SDSOI controller. Optical gateway modules have Ports 2 to 5 available for interconnects, while Ports 1 and 6 are reserved to connect with the operating system tap ports.

Optical gateways support a wide range of applications other than DCI, including carrier Ethernet transport, mobile and broadband backhaul, multicast video and Cloud. They also support optical channel data unit-k switching, which has effective bandwidth management capabilities at sub-wavelength levels, and performs dynamic bandwidth provisioning across the OTN layer. This gives the optical gateways OF capable Optical Transmission System (OTS) lines, which are used by the controller for proactive protection support.

3.4.2 SDSOI Core

SDSOI is a programmable optical transport network controller that control optical gateway networks and properties without any compromise on kind of application support. It is a unified network controller with a centralized global view of DCs across a region, nation or the world. As explained in Chapter 2, SDN is about extracting the control of the networking devices; the SDSOI controller also extracts control from the open flow enabled forwarding optical gateways, and manages the flow control via southbound interfaces.

SDSOI is a multi-layer, multi-module and multi-vendor transport layer SDN controller, with various onboard features such as path computation, load management and load balancers as default plugin modules. These modules perform different tasks for computing and installing multi-layer end-to-end DC interconnections. Thus, the SDSOI controller acts as a centralized intelligent bridge to transport data across the gateways and corresponding DCs. The SDSOI core runs the algorithms to perform the analytics, automation and orchestration of installing the rules in the networks

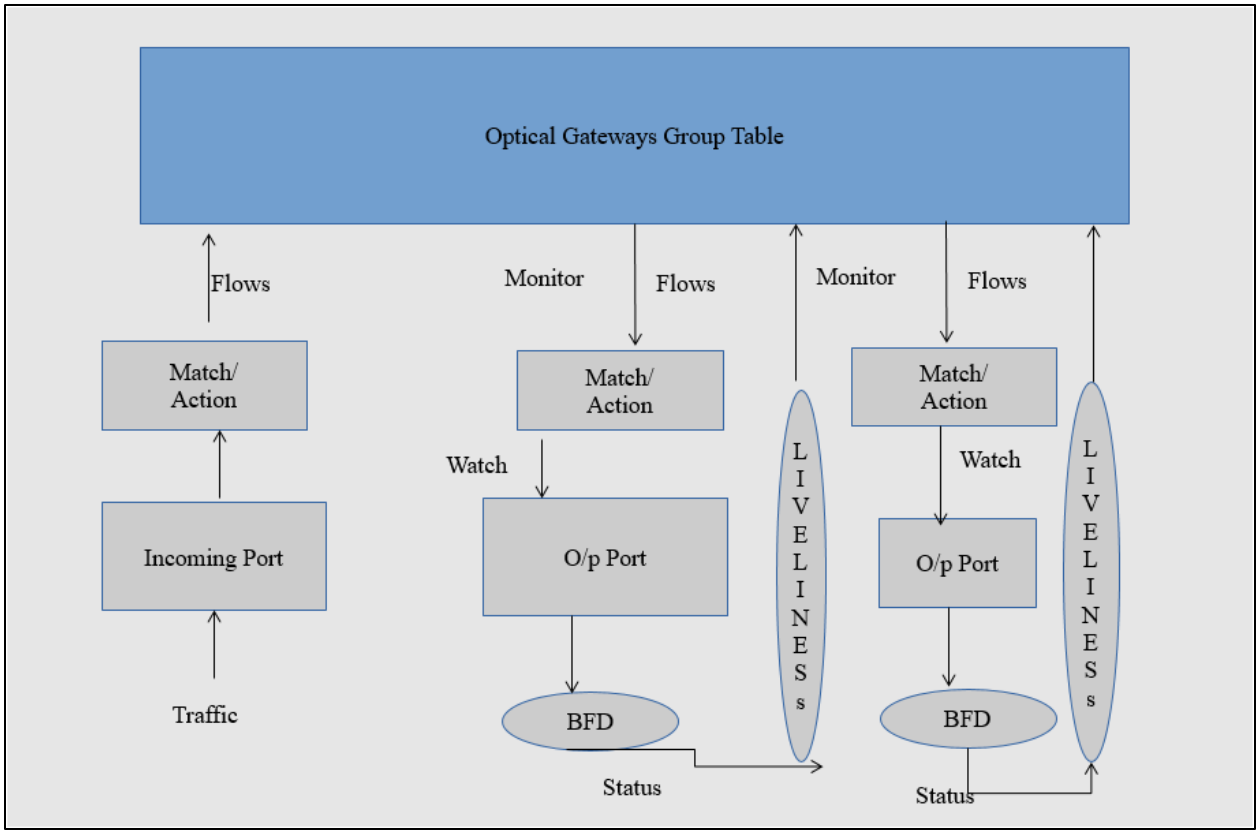


Figure 3.4 Path Failure Monitoring

SDSOI core layer programming enables the orchestration and automation of interconnection communication between the underlying datacenters, by doing the complex calculations (i.e. routing, naming, policy checks and declaration) and installing flow rules. The SDSOI core interacts with the management and business applications via open northbound interfaces, and with the gateways using open southbound interface. The controller is programmed and designed so it supports open protocols at both layers and is not restricted to a specific communication interface, which makes it multi-vendor and multi-domain compatible. By multi-domain, we mean whatever is happening inside the DC's SDN controller is a domain to the SDSOI controller and there are multiple such domains, which are independent of each other. SDSOI is designed using open interfaces in both the north and south layers, and can efficiently create DC interconnections globally or regionally, regardless of multi-vendor optical gateways or DC infrastructures.

SDSOI is built with predefined inbuilt plugins known as high and low-level modules. These plugins are the mandatory software applications that are programmed for the controller to be used in routine DC's communications. High-level modules decompose the application policies and

objects, and send the traffic policies into several inbuilt plugin applications, such as variable bandwidth packet links, dynamic optical bypass, traffic engineering, unified recovery, differential treatment, load balancing, DCI and troubleshooting, SDSOI customized routing, Quagga and Paxas. These are the modules required by the service providers to create necessary automation and orchestration and to do computations to simplify the routing of traffic. They are the mandatory management-featured application modules for overlay transportation of traffic without the need to conduct detailed inspection of the packets. Packet inspection is actually never required, as the firewalls inside the DC infrastructure have already performed the security check and encryption, before passing it to the IP router and optical gateways for communication.

High-level modules include Quagga and Paxos, which assist with routing and fault tolerance. Paxos is used in SDSOI SDN environments with traditional fault tolerance algorithms, to ensure that when a link fails during traffic routing, gateways interact with the controller to determine the new master from the given gateways that allows the communication to proceed at same bandwidth speed. Control monitors the already computed gateway sub-path for failure, and responds with a new disjoint sub-path in the network. The new route change does not occur until all the intermediate switches of the path agree with the new path.

Quagga is a routing software suite plugin for the SDSOI core that implements other routing protocols, including OSPFv2, OSPFv3, RIP v1 and v2, RIP and BGP-4. It has a core daemon that performs abstraction of the forwarding devices, updates Quagga clients, implements the routing protocol and sends the updates.

High-level modules also interact with the inbuilt low-level modules of the core that manage the topology, links, flow rules, gateway sensing, neighbour discovery updates, ARP, RARP and proxies. These management modules inside a single server image core are responsible for interaction with the high-level core modules, and perform the overall computations in the network. The SDSOI core computes controls and manages the algorithms, as well as the coding and dependencies in the single script, and runs the log. Though the SDSOI core can be integrated with real deployment to the OF hardware gateways, in our work we used a virtual network emulator.

Service providers have large and complex infrastructures, with diverse demands for bandwidth, latency and services. For example, one customer might require a 100G wavelength dark fiber connection, while another only needs a 40G shared or dedicated bandwidth. It is also possible that

a DC needs 100G connectivity for one DC and 10G for another simultaneously. Thus, depending on customer requirements, data criticality and committed SLA, every interconnect can support different bandwidth needs at any time. Interconnecting high numbers of DC with high service demands requires applications plugins such as load balancing and DC gateway troubleshooting, to maintain business continuity while managing SLA demands of high bandwidth and low latency. SDSOI core also performs dynamic optical traffic bypass by directing flows to gateways' ROADMs, and instructing the transponder cards to bypass a specific OTN optical packet due to priority and latency.

SDSOI is designed to support proactive highly efficient protection and restoration of interconnections, while ensuring fault tolerance, topology updates, traffic analysis and characterization, link utilization and flow management in the network. This means that for every change or failure in the network, traffic must be modified and rerouted to a new path that has less or equal cost, low latency, reduced restoration time and the same bandwidth interconnection link so traffic failure is imperceptible to the DC. SDSOI is programmed to use its inbuilt algorithms to perform computation of routing in reduced time frames. The SDSOI core is designed for effective traffic management for service providers, traffic analysis and bandwidth, time and QoS aware scheduling in the network.

3.4.3 Application Layer

In addition to the above inbuilt plugins or modules in the core and NOS, SDSOI also supports business specific applications or management planes running on top of the core. This layer leverages the functions of the northbound interface to implement additional control and logic. The applications can be tailored according to business needs, and act as a secret sauce in SDSOI soup by defining the additional features and services required. They instruct the controller on what to do and when, and based on these rule inputs the controller to translates the policies into instructions for the forwarding plane using OF flow rules. The general concept of application interaction and high-level procedural behaviour, as well as component instruction, is shown in Figure 3.4.

Our SDSOI secret sauce is its unique applications, along with their categorization and how they are modelled to behave. In our architecture applications are categorized as:

- Online or Dynamic;

- Offline or Routine; and,
- Third party.

Online or dynamic applications deploy policies or rules in the network corresponding to changes in network demand. These include restoration, on-demand bandwidth provisioning, path calculation and backup service activation. They are used in the network according to the demands of the interconnects and the services being deployed. It is not necessary to have these applications activated and used for the creation of every service among the network elements, though their programmability does contribute to automatic service provisioning and maintenance of high SLA.

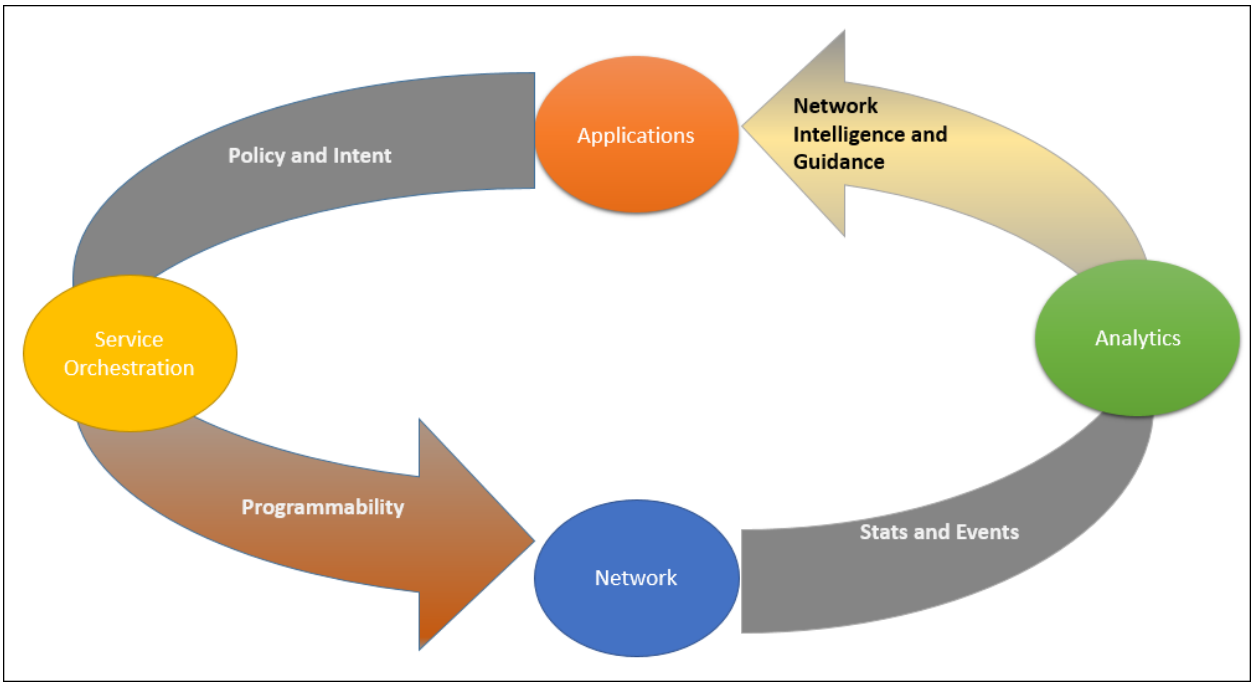


Figure 3.5 SDSOI architectural component interactions.

Offline or routine applications are involved with building the rules for routine DCI procedures, including effective management of bandwidths, scheduling different bandwidths at different times of the day to achieve bandwidth optimization, VM migration, data replication, gateway configuration and management, fragment management and inventory and performance management. These applications are continually being activated and used by the SDSOI core to deploy policies according to requirements, and for routine day-to-day communication when nothing in the network is unexpected or on-demand. These are the mandatory business applications that must be deployed by the service providers and used for DC communications.

Third party applications are fixed and static (e.g. Microsoft Office, Confluence, Jira) and may be required by some DCs and not others. These applications are hosted by cloud services, and may be needed for only few customers. SDSOI is smart enough to know which DC needs this-or-that application, and it thereby helps selected DCs make more effective decisions.

These applications are defined and simulated by the archetypes which we generate in the templates, and further programmed as per business needs. Once the code for an application is written, we build the application system using controller-specific build tools, then install and activate it. Procedures for building and creating your own business application is explained in Chapter 4.

The applications also create the policies that are implemented on the controller as Intents. The controller converts these policies to the topology-using rule for repeated communications, which becomes the rule for the optical gateways. Data planes or gateways follow the rules, and for every new request the gateways query the controller and request new flow rules, as explained in the OpenFlow section in the Section 3.2. We define Intents, and discuss how we use them in the SDSOI model in the next section.

3.4.4 Intents

SDSOI uses intent-based networking, with the intents defining the policies and actions required in the network. The controller responds to the policy intents [57], performs the complex object activities and converts them into flow rules to direct the gateway behaviour. Objects can be defined as constraints, network resources, criteria, instructions, and other aspects. SDSOI allows applications to create their own type of intents in the network, and send them to the controller. These intents carry a unique Intent ID, and are also tagged with the ID of the application used to create them.

Intents [58] are invariant, and do not change if links go down, servers crash, firmware is upgraded or the infrastructure changes. They remain the way they were installed until the objects they were built with crash, and if that happens they fail. Being invariant, the intents always remain in the installed state for the underlying network, which simplifies overall application development, testing and deployment in the network.

Another advantage of intents is they are not a protocol or a vendor, nor are they specific to a media type. They allow SDSOI to deploy a range of dissimilar solutions, thereby eliminating the need

for lengthy application changes and run time complexity. Intents are used to define the static action policies onto the controller, and continue to perform regardless if the network scales up or down. They also allow common resource allocation and management.

Intents have the inbuilt capability to resolve conflicts, as they communicate about *why* an action should be taken, not *how* it should be done. With its globalized and central view, the SDSOI controller determines actual or apparent conflicts in the network by fulfilling the cumulative intent of multiple-client services. Users or administrators define the intents, and composition and conflict monitors inspect the objects to identify any dependencies between them. As mentioned above, the intents are compiled down to a set of flow rule model objects in two stages:

1) Compilation of an intent into an installable intent called **IntentCompiler** [59]; and,
2) Compilation of an installable intent into FlowRuleBatchOperations by **IntentInstaller.**

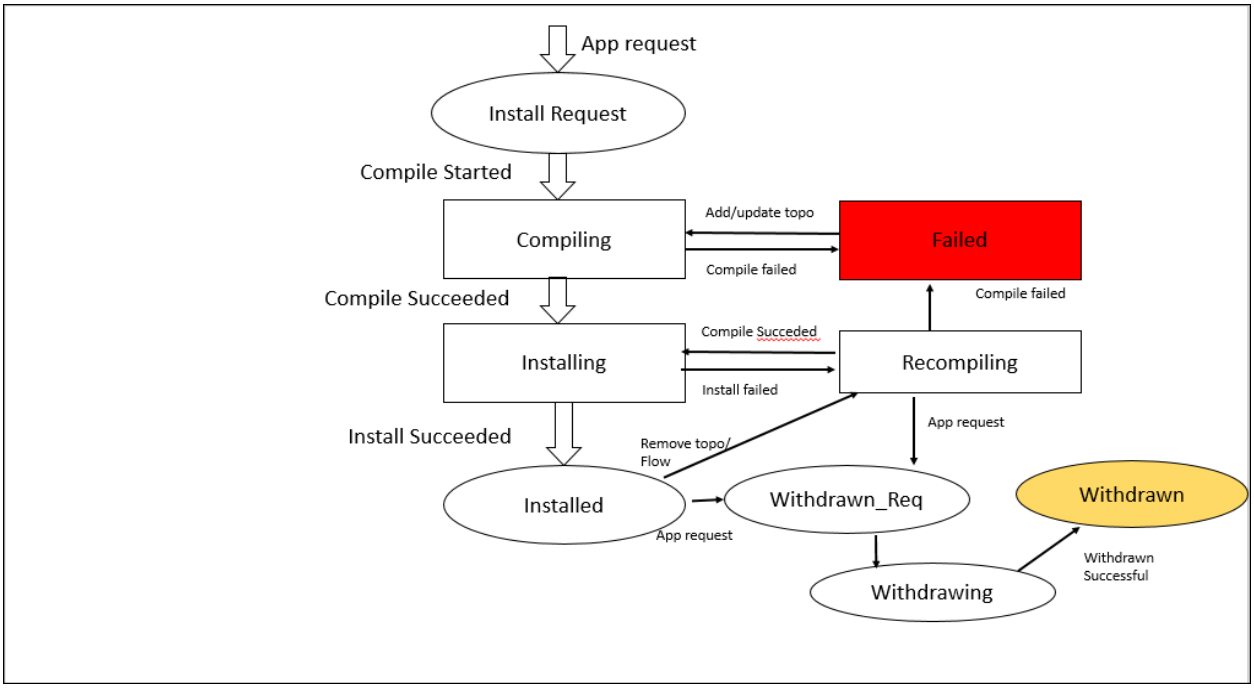


Figure 3.6 State Transition diagram for compilation of intents.

Figure 3.5 shows the intent lifecycle by the transitions it goes through, and details the stages before it is installed. As is evident in in the figure, not every application intent is installed in the network. Non-installable intents are only associated with IntentCompiler, whereas installable intents have IntentCompiler and IntentInstaller associations. Both are managed internally using **IntentManager** [59].

When we code our SDSOI applications, we write the application logic in Java programming language, which then is associated with the intent.java file of the application template. Thus, to make an application work in the network the application logic must create an application request to the controller, which uses various objects to create an immutable Intent Object. Different applications install different intents in the network, depending on network needs and the constraints they are built with. All the application intents can work simultaneously; they are created, managed, deployed, used and withdrawn in an interesting manner, and are beautifully designed in the architecture, as described below:

- Intent Manager manages the intent installer and compiler requests, decides which requests can be installed and converts them into installable format;
- Intent Store stores the intents in the database, and shares them back-and-forth with the intent manager as required;
- Intent Resource Scout monitors intent requests, and informs the installation worker to take the necessary actions; and,
- Intents Installation worker performs the actual installation and communication via REST API.

3.5 Day Night Scheduling Application

Day night scheduling is a smart business application that performs optimal resource allocation for the services, and supports varied bandwidth demands at different times of the day. It is designed with the objective of, maximizing the admitted demands, with high bandwidth connections at pay per use cost and focuses on the following goals:

- Efficient and optimum use of bandwidth;
- Optimized path calculations that consider minimal cost and capacity;
- Survivability of the bandwidths;
- Accommodate specific bandwidth demands (per customer needs);
- Efficient load sharing;
- Automatic downsizing of bandwidth when schedule expires;
- Maximized volume of served demands; and,
- High bandwidth connections.

Achieving above goals and making scheduling decisions for optimization is an NP hard problem, and to address it we decomposed our application solution into the following:

- Packaging and scheduling;
- Routing; and,
- Protection and restoration.

The primary aim of DNS application is to deploy the interconnection service that is a combination of the above three activities. Thus, for every interconnection service demand among DCs in the network, DNS application creates a policy to pack and schedule the service from the current gateway to the next sub-path gateway or the end destination, then instructs the controller to install the flow rules to perform the routing and implements protection of the interconnects. To execute and install the policies, SDSOI controller computes the optimal path between the source and destination DC, and ensures that the appropriate path computation is done. The controller uses various inbuilt and installed algorithms programmed to do the calculations, including *extended Dijkstra algorithm* to compute the shortest available route, and *K-shortest path* [72], [73] for proactive protection in the network. The concepts of the main features of a DNS are as follows:

3.5.1 Online Packaging and Scheduling

DNS applications package the traffic demands from any of the network gateways into a single ODU type package, which optimizes bandwidth usage and reduces overall interconnection cost. At the transport layer, the type of packaging of the traffic demands and DC interconnects help achieve optimal resource utilization. Any SDN controller can work more efficiently if the controller is already aware of the traffic demands of each DC, and they are prescheduled in the network. Prescheduling interconnect and bandwidth requests provides the controller with the interconnection demands it needs to manage, create and support, and for how long. With pre-knowledge of the connection requests, the SDSOI controller can effectively combine smaller OPU and ODU demands into bigger ODUs, which helps eliminate traffic loss and reduces interconnection costs. With this approach, the controller is already aware of the bandwidth demands of the DC, as well as the priority and latency. Thus, while computing the interconnection request the controller has full control of the optical gateway network and its demands, which allows it to compute

and install the interconnection request according to the defined constraints such as interconnections capacity, link priority and latency. The DNS application creates prescheduled intent requests in the network, which gets automatically activated at the scheduled time.

3.5.2 Routing

Routing is the second phase of a DNS application, when the application and controller are responsible for routing the traffic flow from the source DC gateway to the destination gateway, using flow rules and flow tables of OF communication protocol installed by the SDSOI controller. When the intents are actively installed in the network, routing of the traffic between the source and destination DC gateways begins automatically.

3.5.3 Protection and Restoration

The DNS application along with the packaging and scheduling of the interconnects to route the traffic according to the calendared demands, also supports, proactive protection and restoration of the traffic when failure occurs. Protection and restoration in a network is known as survivability, and is the third phase of the DNS application. With SDSOI, it is possible to apply survivability to the transport layer network using SDN and the OpenFlow protocol. Network survivability depends on the network's recovery time, which changes with every connection request and type. Our primary reason for bringing survivability to SDSOI DNS applications is to reduce the recovery time of transport layer SDN controllers with extremely high bandwidth DCI. Our work focuses on restoring and protecting the traffic in the event of failure, using the following protection schemes.

3.5.3.1 Link Protection and Restoration

Link based protection restores link failures in the network by rapidly installing a new alternate sub-path to replace the failed link. It does this by installing the protection path of only the link that failed in the network, and rest of the path is unchanged. Ideally, this reduces the need for crank back routing in the network, but when no alternate link is available it does crank back the traffic to the source node, to ensure traffic failure does not occur and business continuity is maintained. Link based protection and restoration depends on several factors, including link capacity

and cost, and the shortest available new alternate path for the link. Upon failure, the new alternate link sub-path is installed very quickly, thereby preventing the traffic going back to the source node. A pictorial depiction of link based protection and restoration is provided in Chapter 5, where we use an example to describe how link protection recovers from traffic failure by installing only an alternate sub-path that is available to that link.

3.5.3.2 Segment Protection and Restoration

Segment based protection and restoration restores the network traffic by rapidly installing new segment paths in the network to replace those that failed. To do this, the controller divides the network into multiple smaller segments according to geographical regions of presence, availability of hardware, type of bandwidth and capacity. Whenever an interconnect in a segment of the network fails, the source gateway of that segment restores the traffic by rapidly installing the new alternate sub-path for that segment, and restoring the traffic flow on this new protection segment.

Segment-based protection and restoration does not require signaling to the source node of the segment for traffic restoration, as the segment's source gateway, as well as the controller, actively perform live monitoring and scanning of all the gateways and interconnections in the segment. When a segment failure occurs, traffic is restored in new sub-paths toward the segment's destination. SDSOI does not need to do topology discovery again, since controller has already calculated multiple alternate new segment paths for that section.

With these types of protection and restoration schemes, failures in a segment do not require cranking back the traffic to the original source gateway of the network; it is simply rerouted between the segment's source and destination gateways. Thus, only the path for that segment is reinstalled and routed, and the rest of the traffic flow remains the same. Segment based protection for service providers is generally used in geographical areas or regions that are highly prone to failures, have multiple smaller ISP vendors , or use third party vendors. Segment protection is basically an improvement to path protection schemes. A pictorial depiction of segment based

protection and restoration is shown in Chapter 5, where, using an example of Google WAN topology, we describe how segment based protection solves traffic failure by installing a new segment path in only that segment, and rest of the path remains the same.

3.5.3.3 Path Protection and Restoration

Path based protection and restoration is responsible for restoring the network by rapidly installing an entirely new, disjoint interconnection path for the working interconnection. The paths installed with the path protection scheme are new and complete disjoint paths between the two gateways, pre-calculated by the controller when it discovers the topology. DNS applications are programmed to provide path protection, and they install a complete new disjoint backup protection path whenever more than one link in the network fails, or multiple links in multiple segment fail. If only one link or network segment fails, DNS typically restores them using the above two protection schemes.

The main purpose of path protection schemes is to restore traffic as soon as possible, by installing a disjoint path between the source and destination gateways. Disjoint path installation requires computation from the centralized controller with the topology information of the entire network. This is needed to ensure the new alternate paths can install the protection paths and restore the traffic flow.

Unlike the above two protection schemes, path protection traffic is cranked back to the source gateway and reinstalled in the network from the source gateway to the destination gateway. Though path protection does consume fewer resources in the network, it has higher restoration times compared to the other schemes. A pictorial depiction of path based protection and restoration is provided in Chapter 5, with the aid of an example of Google WAN topology. We also describe how path based protection recovers from traffic failure in SDSOI DNS applications by installing a completely new disjoint path.

Chapter 4: Simulation

4.1 Introduction

This chapter describes the simulation setup environment of ONOS as the SDN controller, and how a transport SDN is built with ONOS, including an overview, basic functionalities, and features of various tools and technologies used in this work for building the DNS application. It also details the procedures and steps required to build the DNS application, how intents are created and how the application functions. We also list various challenges encountered during the setup and development of the DNS application.

4.2 Environment Setup

This simulation framework is designed in a Linux environment. Linux comes in variety of flavours, depending on the Linux kernel and the GNU user space required. For the setup, we used a machine with 24 GB RAM, an Intel core i7 processor, 1 TB of hard disk space and open source Ubuntu 16.0.4 image. Ubuntu is a Debian-based OS widely used in networking and cloud servers.

While building the simulation framework, we installed an open source software development tool called ‘git’, with distributed version control platform. Git is widely known for pulling out various open source codes, and we cloned the ONOS source code (version 1.9) from git’s gerrit repository. ONOS is an open source SDN controller programmed in Java, and we installed Oracle Java 1.8 for development and testing of the Java programs, and to setup the Java home environment. The ONOS controller source code is imported in Intellij IDE for faster development and refactoring support, and a new project with a root *pom.xml* file is exported as *org.onosproject: maven* project inside the JDK 8 directory [60].

After completing the basic setup environment, we built the ONOS controller using ‘buck’, an open source build tool created by Facebook and further developed by Google. Buck builds all the dependency targets and SHA hashes of files, and avoids unnecessary processing by identifying whether the target artifacts require rebuilding or not, and increases the reproducibility of the controller builds. To build ONOS, we typically used one of the following commands:

```
$ONOS_ROOT/tools/build/onos-buck build onos --show-output
```

or

```
$buck build onos
```

ONOS needs to be rebuilt whenever changes are made in any of its directories, new code is written for it or any of the existing code is modified. Once the development environment is ready and there are no code changes in any directories, ONOS can run locally every time with the following simple command:

```
ONOS_ROOT/tools/build/onos-buck run onos-local -- clean debug
```

Once the entire ONOS project code is built successfully without errors, ONOS CLI is attached to the controller at another terminal, using the following command:

```
$ONOS_ROOT/tools/test/bin/onos localhost
```

Graphical user interface (GUI) for ONOS can be accessed using *http://ipaddress:8181/onos/ui*

4.3 Open Network Operating System (ONOS)

ONOS [61] is an open source protocol agonistic SDN controller, designed in a multi-module model and managed with OSGi bundles. It is comprised of several sets of sub-projects, each with their own source trees built independently in a hierarchical fashion. It uses Maven's concept of hierarchical POM file organization, where every component or sub-project in ONOS has its own pom.xml file. All these intermediate pom.xml files are linked to the parent pom.xml files of the ONOS project. The parent pom.xml file contains the shared dependencies and configurations of every sub-project pom.xml files. This amalgamation of intermediate and parent pom.xml files helps build ONOS in a completely independent manner.

ONOS is created by leveraging the Apache Karaf (modern and polymorphic container) feature. Karaf is a lightweight, powerful standalone container that supports a wide range of technologies and "runs anywhere" with Java, cloud and Docker images. It uses Apache Karaf as its OSGI framework, and enables the use of JAX-RS API for the development of REST API, which is used by the applications to communicate with the controller.

ONOS is a protocol aware network-facing module that interacts with an underlying network. It has a protocol-agonistic system core that tracks the information through run time server logs, and shares all information about the network state. Thus, whenever a new southbound protocol is added to the controller, the controller adds a new network facing module only against that protocol, and writes its API in the system. Figure 4.1 shows the outline of an ONOS architectural layer. ONOS supports its features in the form of services, comprised of multiple components that create a vertical slice through the tiers as the software stacks. Services can be defined as a collection of components or a subsystem. ONOS defines its set of primary services as Devices, Links, Hosts, Topology, Path and Flow rule, and packets and subsystems as tunnels, intents, Flow rule, drivers, device configuration and many others, as detailed in ONOS wiki [61].

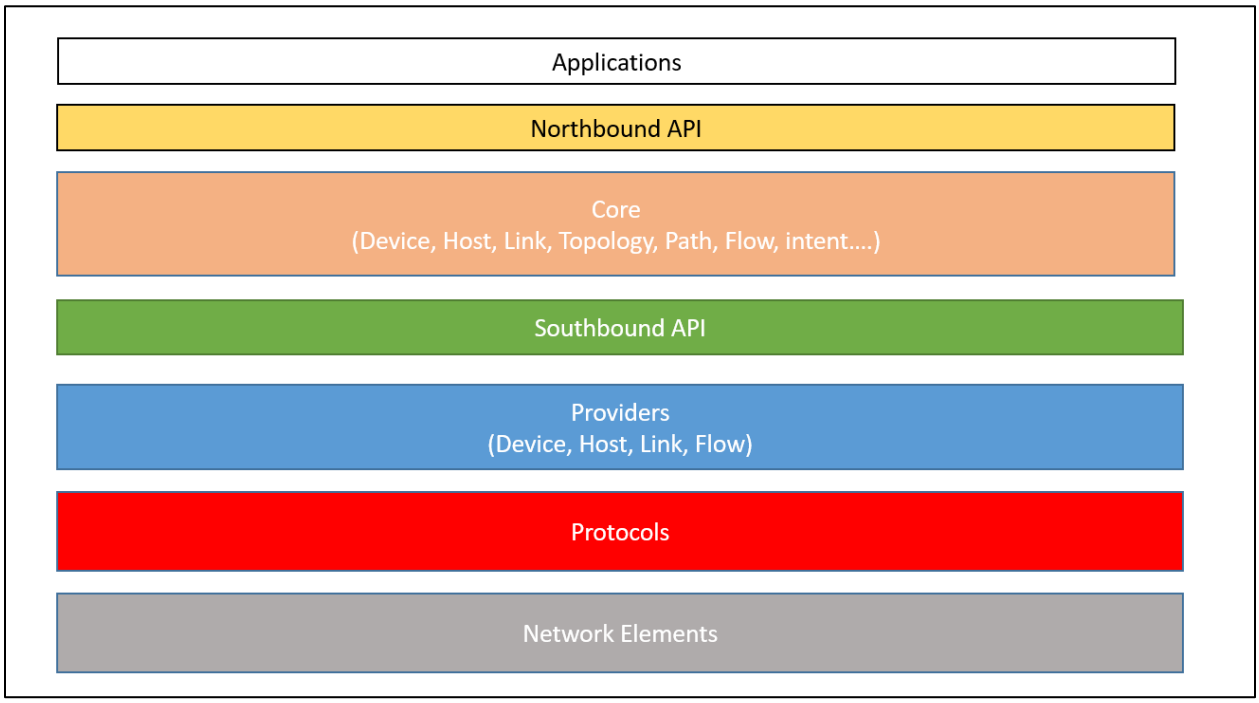


Figure 4.1 ONOS Architecture [61]

ONOS builds its protocol agonistic topology by using two complementary mechanisms: network discovery and configuration. It identifies the locations and properties of the network elements using the networking protocols, and proactively carries them out with automatic topology discovery when it is enabled. For example, whenever changes are made in the network topology whether it is the addition of a new device or the failure of a device, the controller is automatically aware of the change through its built-in topology discovery, and it adds it to the network view.

Further, depending on the requirements of the connection and protocol used, it adds information such as device capabilities, number of ports and others. Messages are conveyed to the controller via the handshake and session initiation by the southbound layer.

Configuration of the network refers to the technique of configuring or adding processes and features. These features are instructions from the application to the controller, which then adds, configures or modifies them as per the application attributes.

Device driven ONOS subsystems isolate the device specific codes to ensure they don't spread throughout the rest of the subsystems. The majority of these codes are either part of an unforeseeable future, or required minimally on a case-by-case basis. The subsystems provide the means to accommodate these cases, and allow the applications to interact with them as required through independent, well-defined device and protocol abstractions. Most of the infrastructure layer devices are upgraded and released in different cycles than the controller codes. This subsystem also allows asynchronous delivery and dynamic loading of device-specific code. With these unique attributes, drivers can be defined with unique names and the type of organization, with different sets of class behaviour. It can also inherit the unique behaviour of other drivers.

ONOS is well known as the distributed SDN operating system with a single controller comprised of many clusters, or different controllers interacting with each other via east-west interfaces and further interaction with the master SDN controller. Also, in certain cases one of the controllers in the distributed controller acts as the master controller, and directs the flow to rest of the controllers. This distributed and centralized approach provides high fault tolerance and resilience, even when the individual controller fails.

ONOS GUI [62] is a single-page web application with hash-navigation, comprising of client side with front ending languages like AngularJS, JavaScript libraries and modules like CSS files. The server-side of the GUI consists of Java classes and programs that create the GUI view and its interfaces with APIs, to provide the data for the views. The GUI webpage is served by the web server running an interface inside ONOS as a subproject. The core of the server side is exposed to GUI via *UIExtension* Service, which handles the UI view with '*css'* and *'js.html',* and generates the message handlers via *UIMessageHandlers* and request handlers.

As discussed, ONOS uses Apache Karaf with an OSGi framework to provide a platform for individual software components to declare certain parts configurable. This feature allows the developers to change the parameters in-flight, and tracks them every time the system is restarted. It comprises a subsystem for overlaying the OSGi mechanism, and manages the configurations in a distributed manner across the entire cluster.

An ONOS application layer is a separate subsystem that facilitates software delivery and management across all instances of the cluster. It uses distributed application management subsystems to manage the network control. These applications are built on top of the Apache Karaf feature, which is in turn built using OSGi bundles. The bundles and features allow activation and deactivation of the applications, which are a combination of OSGi bundles as a single feature.

ONOS SDN applications are designed according to business needs, and are further categorized as simple, multi-bundle, and multi-feature multi-bundle applications. Simple applications are built using single features with a single bundle, whereas multi-bundle applications are comprised of various OSGi bundles combined as a single feature. Multi-feature multi-bundle applications, as the name suggests, consist of many bundles combined to form a single feature, and many such features combined to form a single application. These applications provide the means to contain features definitions, and allow any type of OSGi to become a single artifact, which is stored in the form of ONOS Application Archive (.oar) file.

ONOS interacts with network elements or devices via providers. Providers are standalone ONOS applications based on the OSGi components, and are dynamically activated and deactivated in run time. Their primary task is to abstract the configuration, control and management of specific families of devices (e.g. OF, SNMP, Netconf). Providers execute requests from the core, and either install the OF flow rules for them, or set and get the Master information base (MIB) in SNMP, or select the port status in Netconf. The core of the controller originates the request from the provider manager, and the provider notifies the core about the events via providers service API or southbound API. This gives providers the capability to provide different types of support for each set of devices, which are configurable and programmable.

Providers are defined, as the same set of elements, and each are responsible for providing the control abstractions and environment sensing over the physical and logical infrastructure entities, such as devices, links and flow rules. Examples of providers include Alarm provider, Device

provider, flow rule provider, topology provider and tunnel provider. They are responsible for using provider API to notify the core about the discovery of an event, with detailed descriptions for as long as the method is exposed to the manager in the core. It also helps generate the signals, events and packets to the data plane.

ONOS also has protocol modules that contain the implementation features needed to communicate with infrastructure layers to implement the specific runtime control and management protocols. It is currently compatible with various protocols, including OpenFlow, NETCONF, SNMP, OVSDB and BGP. ONOS protocols are divided in two sub-modules: API and CTL. API contains the Java interfaces and classes, and CTL defines the specific implementation of the API. This helps build clean inter-modular dependencies, so administrators can rely on the API without the implementation of CTL.

4.4 Tools and Technologies Used

In this section, we describe various tools and technologies used while simulating the applications in the SDSOI architecture.

4.4.1 Mininet

Mininet [63] is an SDN network emulator that is widely used to create a network of virtual hosts, switches and links. It creates the Kernel and user-space that enables the OF switches and hosts to communicate over the simulated network. Mininet [63] executes process abstractions, and can easily implement process-based virtualization. It can run as many as 4,096 hosts and switches on a single OS kernel, and its coding is entirely written with the Python programming language. Mininet supports all versions of OF, and is widely used for research, development, learning, debugging, testing and prototyping.

Mininet has various advantages, such as faster boot time, higher bandwidth and greater scalability. It is an inexpensive, quickly reconfigurable, re-startable testbed, which can be installed on any Linux platform. With Mininet, sharing and reproducing network systems and experiments is relatively easy. Though it is mainly designed for OF networks, Mininet also supports legacy IP networks like Open vSwitch, Ethernet switching and IP routing[64].

4.4.2 Maven

We used Apache Maven for both software management, and as a comprehension tool. Maven [65] is a build automation tool to manage projects buillds, and we imported ONOS as a maven project by using its *POM.xml* file. Maven plugins are also used for packaging the ONOS applications, as well as generating the component configurations of resources. It specifically builds the *.oar* file for component configurations while creating the applications. ONOS applications can be easily rebuilt using the following simple Maven command in the CLI:

```
$ mvn clean install
```

4.4.3 Buck

Buck [66] is a system build tool developed by Facebook, and later acquired by google. Buck builds all project modules that consist of codes and resources, and supports a variety of languages. To build projects with Buck certain prerequisites must be installed, such as oracle JDK, python2, git, watchman and the newest Apache Ant. Buck is designed so that anything that will affect the output of the build rule must be defined as input of the build rule. It combines all the information into hash, and represents the total output.

Buck rules include a file named as BUCK, which defines the build rule (i.e. the procedure for producing an output file from the set of input files). Buck build rules use the built-in python functions in the built file, and use arguments such as name, deps and visibility. It can build multiple deliverables from a single repository.

In our work, we built ONOS every time we started it new or made changes to it, using the following buck command:

```
$ buck build onos
```

4.4.4 Intellij

Intellij is a Java integrated environment for software development. Intellij IDE offers code completion by analyzing the context, and helps navigate the code by jumping to a specific class, and it has options to fix inconsistencies via suggestions. ONOS application logic, codes and

modifications for application integration are written in Intellij. It also debugs ONOS source code by exporting the code with the debug option 'ONOS-Karaf clean' debug argument, and can show the values in the editor where the fields and variables are used.

4.4.5 REST API

REpresentational State Transfer (REST) is not a protocol [67], but a software architectural pattern for creating the interface, and a means to exchange information between the application and the controller without including the structure of the information. REST [68] is analogous to transport protocols for payloads of any format, such as 'ssh' or 'telnet'. It is an API that allows clients to perform read/write operations on the data of the server. These HTTP methods are known as *CRUD* (Create, Read, Update, Delete), as described below. REST is very simple API approach and is applicable in both XML and JSON languages.

- **Create –** This method is used whenever we create something new for the server, including adding a new device, and creating its IP address, hosts and name. It implies we need to construct these attributes and send them attached to a HTTP **POST** request.
- **Read** – This is used whenever we need to manipulate any object on a server. We send a HTTP **GET** request to the API/devices, and it responds with a payload of the full list of devices.
- **Update** – This method is used when we need to update the devices in the network. We send a fully updated payload with the HTTP **PUT** method.
- **Delete –** This is used to delete the devices in the network, using the HTTP **DELETE** method.

4.4.6 Ganglia

In this work, we used Ganglia as a scalable distributed monitoring system for high performance computing of the ONOS grid, and to evaluate the performance of SDSOI architecture. Ganglia[69] is an open source project that leverages XML for data representation, XDR for compact, portable data transport, and the RRD tool for data storage and visualization. Gmond is installed on an ONOS controller to monitor the performance of application intents, as well as the traffic flow in the network in terms of flow rules. Gmetad collects the data from other gmetad and gmond sources, and stores their state in the disk using indexed round-robin databases. Ganglia web front-end data is stored on the disk, and expresses the gmetad data on a graphical web interface using PHP. ONOS already has the ganglia plugin installed as the default application for monitoring the performance.

4.4.7 Collectd

Collectd [70] is another free, open source project that runs in the background and collects system and application performance. It provides a means to store the values in variety of ways. In our work, Collectd gathers the metrics from the ONOS server operating system, logs the files and their output on the devices, then stores the information on the network. It generates the statistics of the OF devices used in Mininet, and computes the performance analysis in terms of flow rules and flows sent by the controller. Collectd includes features and metrics optimization and supports numerous plugins, including C, Java and python, making it very compatible with servers. Collectd is based on a data push model; that is, data is collected via the API and pushed to the server to support networking, multicast, unicasts and proxy operations. It also encrypts network traffic, and performs active automatic discovery.

4.4.8 Wireshark

Wireshark [71] is an open source network packet-analysis tool that captures the network packets and displays them in detail. It is widely used by administrators to troubleshoot the network, by engineers to examine network security, and by developers and researchers to bug their protocols and assess their performance. It can capture the live packets from different network media types, the selection of which depends on the type of operating system being used. Wireshark measures all types of physical interfaces in Linux. We used Wireshark to measure OF messages and network packets on Mininet, with ONOS as the SDN controller.

In this work, SDN applications are designed on an ONOS framework, using all the above-mentioned tools and technologies. Their features and roles in building an application and integrating with the ONOS platform are explained, to ensure the reader has a basic understanding before reading the next section.

4.5 Testbed Setup and Simulation

In this section, we demonstrate how to create an SDN application in ONOS, and demonstrate its path from application layer to ONOS core and infrastructure layer.

4.5.1 Creating a ONOS Application

This section details how to create the template for an OF based DNS application, which is responsible for interacting with the network elements via network-controlled device. The application instructs the controller with policies on how to function during each scenario and each request. The DNS application directs the service providers controller to interconnect the data centers using the next generation OTN network, and interconnects them on the requested type of bandwidth (ODU).

To write this application we created an ONOS OSGi bundle project using a Maven archetype, which is an example of a multi-bundle as a single feature. As discussed in Section 4.4.2, Maven is a build automation build tool that generates its own project dependency files (e.g. pom.xml, app.xml) in the project specified directory, and produces the templates for all the project dependencies. We created this application project directly under the home directory with the following command:

```
mvn archetype:generate -DarchetypeGroupId=org.onosproject -
DarchetypeArtifactId=onos-bundle-archetype -DarchetypeVersion=1.10.0-
SNAPSHOT
```

Following the command, the application procedure interviews the developer for information in fields such as version, name, group ID and artifact ID. Using these inputs, it automatically creates the application's required dependency files (e.g. pom.xml, app.xml) in the source folder of the template project, and builds the rest of the application dependencies. The business logic of the application is written using Java as the programming language. The 'pom.xml' file that was generated in this step is the heart of the application, with logic similar to the pom.xml file of the ONOS controller that we imported while cloning the ONOS source code. Figure 4.2 shows the pom.xml file for DNS application.

```
    ~ limitations under the License.

 -->
-<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
 -<parent>
    <groupId>org.onosproject</groupId>
    <artifactId>onos-dependencies</artifactId>
    <version>1.8.0</version>
    <!-- parent is remote -->
  </parent>
  <artifactId>day-night-scheduling</artifactId>
  <version>1.1.0-SNAPSHOT</version>
  <packaging>bundle</packaging>
  <description>Day night scheduling</description>
 -<properties>
    <onos.version>1.10.0-SNAPSHOT</onos.version>
    <onos.app.name>org.daynight.scheduling</onos.app.name>
    <api.version>1.0.0</api.version>
    <onos.app.origin>ON.Lab</onos.app.origin>
    <onos.app.title>Day night scheduling</onos.app.title>
    <onos.app.category>Utility</onos.app.category>
    <api.description>Day night scheduling</api.description>
    <onos.app.url>http://onosproject.org</onos.app.url>
    <api.title>Day night scheduling</api.title>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <api.package>org.onosproject.daynightscheduling</api.package>
    <web.context>/onos/day-night-scheduling</web.context>
  </properties>
 -<dependencies>
  -<dependency>
     <groupId>org.onosproject</groupId>
     <artifactId>onlab-rest</artifactId>
     <version>${onos.version}</version>
   </dependency>
  -<dependency>
     <groupId>org.glassfish.jersey.containers</groupId>
     <artifactId>jersey-container-servlet</artifactId>
   </dependency>
  -<dependency>
     <groupId>org.osgi</groupId>
     <artifactId>org.osgi.core</artifactId>
   </dependency>
  -<!--
     Note: version intentionally not in sync with ${project.version}.
             ONOS no longer publishes shaded thirdparty bundle.
   -->
```

Figure 4.2 pom.xml for "org.onosproject.daynightscheduling" application

4.5.2 Programming the application

In Section 4.5.1, all the dependencies for designing an ONOS project application are created. This section covers the process and procedures for writing the application logic in the src folder of the application project, which was generated with Java as the programming language as discussed above. To write the application, we imported various existing Java classes and wrote a few new classes, including paths, intents, topology constraints, traffic treatment, constraints, bandwidth, protection and network elements and their link classes.

A DNS application is programmed to deploy a scheduled bandwidth interconnect intent capable of protection and restoration. While programming the logic for the application's intent, we focussed on the online packaging and scheduling of BW, while considering latency and the priority of the requests. Priorities are not bandwidth sensitive in the program logic of DNS application; that is, higher bandwidth connection does not imply higher priority, and vice versa.

Bandwidth protection and traffic restoration are programmed and designed so that the DNS application deploys the same bandwidth interconnection, with exactly the same constraints on the newly created protected interconnection as on the working path. This is done to ensure there is no compromise in customers' SLA, and network failures are imperceptible.

As discussed in Chapter 3, applications guide the SDSOI controller on how to create a scheduled optical interconnection, protect the interconnects and restore the traffic. Protection and restoration in DNS is designed for three protection schemes: link, path and segment. These schemes use the k shortest path algorithm [72], [73] where the controller is programmed to use the next shortest path available. The code is designed to provide proactive protection for the paths, segments and links failures with the topology updates. The controller calculates an alternate available path, link and section. The primary objectives of this application are as follows:

- Different bandwidths for different timeslots during the day;
- Priority of the interconnections by service providers;
- Latency of the interconnects, as there could be services with lower bandwidth demands that still require low latency (e.g. dedicated point-to-point connections);
- Proactive protection for links, paths and segments;
- Monitoring of third party patches;

- Automatic withdrawal of the existing interconnects and decreasing them to reduce bandwidth; and,
- Automation and orchestration of interconnection and service creation.

4.5.3 Disjoint Path Discovery and Protection

Computing disjoint paths in the algorithms is a requirement for every kind of protection in the network to bring survivability. In this section we will describe the basic algorithm that are programmed to compute disjoint protection paths for the scheduled DCI service. For this, we have extended the ONOS dijkstra algorithm with Bhandari's algorithm in two phases:

- **Phase 1 – Disjoint path discovery**
  It computes the shortest and disjoint path amongst the gateways using ONOS SDSOI code logic and schedules the bandwidth between the DC at this disjoint path.
- **Phase 2 – Protection path discovery**
  For each protected optical interconnect created by the DNS intents in the network, SDSOI computes a shortest protection path, link or segment as per the intent and object specification. To discover this protection scheme, extension is made to the existing Dijkstra algorithm of ONOS controller along with some protection java classes.

Phase 1 is applied to discover the paths V1 and V2 using Dijkstra algorithm. If a simple sub path, Q2 exists for any of the link, segment or path that failed, protection for Q1 is guaranteed as explained in Algorithm1. For each path, link or node in the network graph, controller automatically calculates the alternate sub path available, and SDSOI core rapidly restores it.

Algorithm 1 (discovers shortest path)

- Phase 1 -- Discover shortest path between V1 and V2 in G
- Step 1 Discover Shortest Path

  (a) Execute Revised Dijkstra Algorithm (G, V1, V2) ⟶ Q1

  (b) If Q1 exists

(c) Go to Step2

(d) Else

(e) Stop Algorithm

Algorithm 2 (discovers disjoint path)

- Phase 1- Discover disjoint path pair between v1 and v2 in G
- Step 2 - Discover disjoint path pair

(a) Execute Bhandari's algorithm (G, v1 , v2 , Q 1 , s) =⇒ Q 2 , P 1 , P 2

(b) If P 2 exists:

(c) Set P P (v1) = P 2

(d) Else:

(e) Stop algorithm

(f) If L Q 1 > 1:

(g) Go to Phase 2

(h) Else:

(i) Return {Q 1 , P P (v1)}

Pseudo code of Step 2 is given in Algorithm 2 where LQ1 represents the number of hops in path Q1 and s is the requested survivability. With Q1 and the required survivability s (link or switch failure protection), simple path Q2 is discovered and disjoint paths P1 and P2 are constructed using Bhandari's algorithm. When no disjoint path pair is discovered, the algorithm terminates

Algorithm 3 (discovers protection path)

- Phase 2 - Discover protection paths for Vi in G

(a) Remove Q1 from G

(b) For Vi in Q 1, where i != 1, 2 :

(c) Execute Revised Dijkstra algorithm (G, Q1, Q 2,s) =⇒ P P (Vi)

(d) Return {Q 1, P P (Vi)}

In phase 2 the protection paths for protected paths in the shortest path Q1 are computed. In Algorithm 3, a global overview of phase 2 is given in pseudo code, where m is the requested minimization setting and PP(Vi) denotes the disjoint path protection for vertex i.

4.5.4 REST API Creation

Writing the Java code and designing the application to interact with the controller, requires an interface that is both programmable and capable of interacting with the controller as well as passing the polices to it. This interface is known as REST API or Northbound API, as discussed in Chapter 2 and Section 4.4.5. For DNS to interact with the controller and build the policies, we designed 'PUT, POST and DELETE' HTTP request methods. These interact with the SDSOI server as REST API, and are built for DNS applications (see *http://localhost:8181/onos/v1/docs* when DNS is activated). If the SDSOI controller is running and the application is activated, HTTP "POST" request creates the web resource for the triggered scheduled bandwidth intent. It is designed with the same input parameters as those on the webpage form of the DNS application in GUI; that is, it contains the source and destination DC gateway addresses, the optical gateway port numbers at both ends, the start and end times of the schedule of the intent, primary and default bandwidth, survivability and latency. Figure 4.3 shows the REST API docs created for the DNS application.

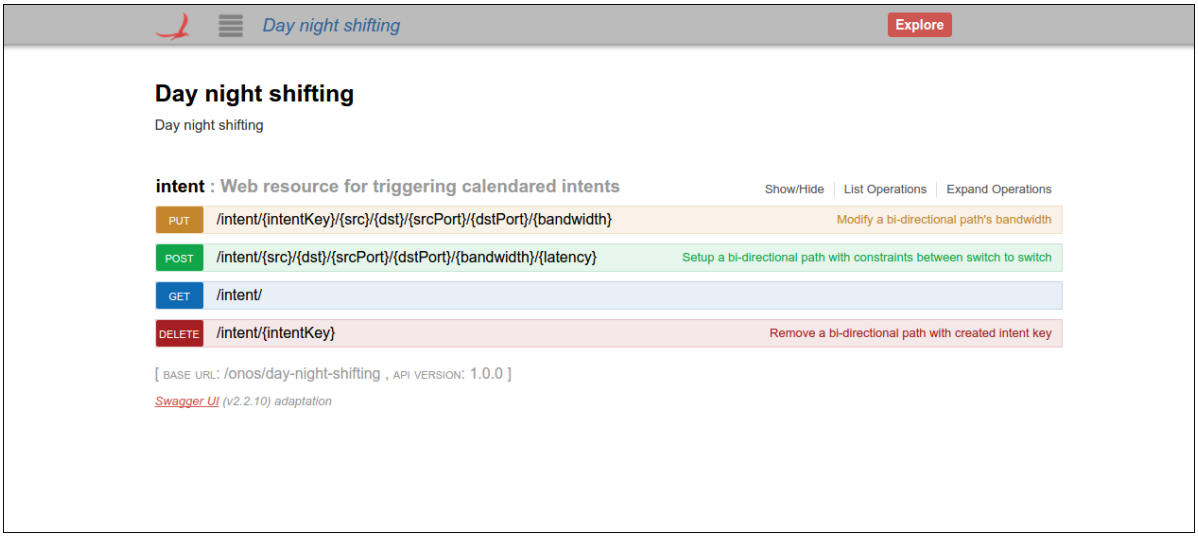


Figure 4.3(a) REST API high level view for Day Night Scheduling Application

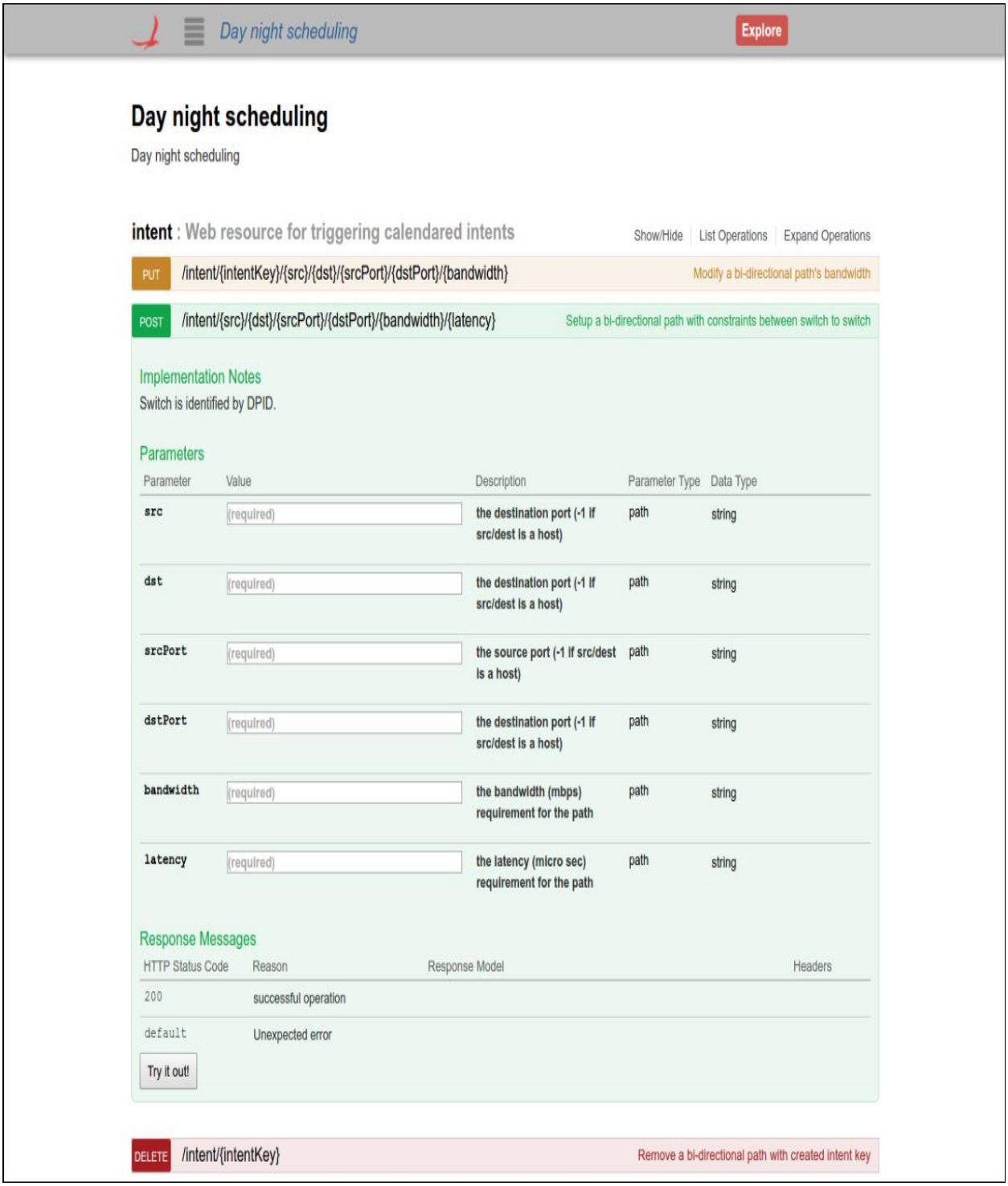


Figure 4.3(b) REST API low level view for Day Night Scheduling Application

4.5.5 Building an Application

Once the application is successfully programmed with accurate and desired logic, its pom.xml file is edited and the onos.app.name property settings are changed to onos.daynightscheduling.app. When the application code is successfully created and the pom.xml file is edited to the desired settings, the application is built using the build tools mentioned in the previous section. For DNS, we used Maven as the build tool, with the following CLI command:

```
"mvn clean install"
```

When the application is successfully built, it automatically creates the .oar and jar file bundles in the target folder of the application directory. To test the application's incorporation of the required changes several times we also used the following CLI command, which helps skip some of the tests and build more quickly.

```
"mvn clean install –Dmaven.test.skip=true"
```

Figure 4.4 shows the creation of the .jar and .oar files for the DNS application in the target folder, once the application is built. If the .oar file is generated it means the application has deployed successfully. We can further test the application by importing, installing and activating it on ONOS GUI, or ONOS CLI with few commands.

Figure 4.4 – "oar" and "jar" bundle of Day Night Scheduling Application.

4.5.6 Installing, activating and deactivating the application

After the application is successfully built, it can be installed using either the GUI or ONOS CLI. We use following command to install the DNS application:

```
onos-app ipaddress install target/daynightscheduling-app-1.10-SNAPSHOAT.oar
```

DNS application can be easily activated and deactivated through ONOS GUI directly, or from ONOS CLI with the following commands:

```
onos > app activate org.daynightscheduling
```

```
onos > app deactivate org.daynightscheduling
```

The following command is used to reinstall the DNS application:

```
onos-app ipaddress reinstall org.daynightscheduling target/ daynightscheduling-app-1.10-SNAPSHOT.oar
```

4.5.7 Intent Creation

As discussed in Chapter 3, applications express their network control preferences in the form of policies rather than mechanisms. These policy-based directives for controllers are known as intents. Whenever an application is created intents are requested, compiled and installed as described in the intent lifecycle in Figure 3.6.

Once intents are successfully installed, they alter optical gateways network behaviour to create interconnects, tunnel the links that are provisioned, install the flow rules on the devices or optical gateways, install the wavelengths to be served and much more. Intents are finally compiled down to the controller to install the flow rules in the network, which allows the optical gateways to process the network demands. Intents can be installed or withdrawn or failed, depending on the application design and issues they encounter. Figure 4.5 depicts the creation of all three types of intents for day night scheduling applications on GUI, which can also be seen in the ONOS server log.

The intents in Figure 4.5 failed because we did not create them either with the correct source and destination address or correct datacenter port numbers. It is also possible that these ports had no host attached. Intents are withdrawn when the time sensitive duration and bandwidth that they

were created for ends successfully. SDSOI then automatically activates and installs new intents of default bandwidth corresponding to the withdrawn intents, which is the one of the primary objectives of this application: to downsize the bandwidth whenever traffic load decreases.

Intents only get installed when they are built with the appropriate program logic code, REST APIs are created for them, the MAC address and port number of the gateways are correct, the application is working and installed, and it is compatible with the southbound interfaces and network gateways. With the state of intents being created, inbuilt FlowRuleBatch operations are handed off to the FlowRuleServices of the OpenFlow version, and FlowRuleprovider provides the Flow Rule to the devices.

4.5.8 Random Real-World Topology

The DNS application is tested on the Google WAN topology with twelve optical gateways, each corresponding to each of the DC and the most referenced Pan-European (Pan-EUR) test network comprised of thirty-two optical gateways that correspond to same number of DCs. Topology views of both the topologies on ONOS GUI are shown in Appendix D, and Figures D.1 and D.2.

4.5.9 Scheduling the bandwidth and protecting the interconnects

When application is successfully coded and built, we create the bandwidth and protection policies via intents from the GUI. To do this, we add a widget on the GUI called Scheduled Bandwidth, which opens the form as shown in Figure 4.6. We enter the values of the source and destination DC addresses and the port numbers in the form, and populate the required bandwidth, latency and priority for the interconnection. Service providers' SDN administrator can decide if they want to protect the interconnects proactively when they fail, or keep them unprotected and wait for the controller to be informed by optical gateways and neighbor discovery, and then create new flow rules. We also need to populate the default-downsized bandwidth in scheduled bandwidth form to let controller withdraw the existing intent and install new intent and interconnection with the default bandwidth. Default bandwidth varies with customer or DC, and their usage.

As described above, with all the tabs populated in the form we then activate the create function, and at the scheduled time the intent is installed. Intents can be reviewed using the Intents tab in the ONOS GUI, as well as in the ONOS CLI. Application performance is tested on intents installed for hours, days and weeks. They immediately switch from the working interconnect path to the

Figure 4.5 Intent generation in Day Night Scheduling Application

protected interconnect path whenever failure occurs, and automatically switches the port in the gateways if required for protection path.

The DNS application build is tested under different pairs of gateways. For every failure in the interconnect path, the required protected path is installed and the intents remain in the installed state as designed. The intents are tested and evaluated for different protection schemes in Chapter 5, and flow rules are monitored accordingly.

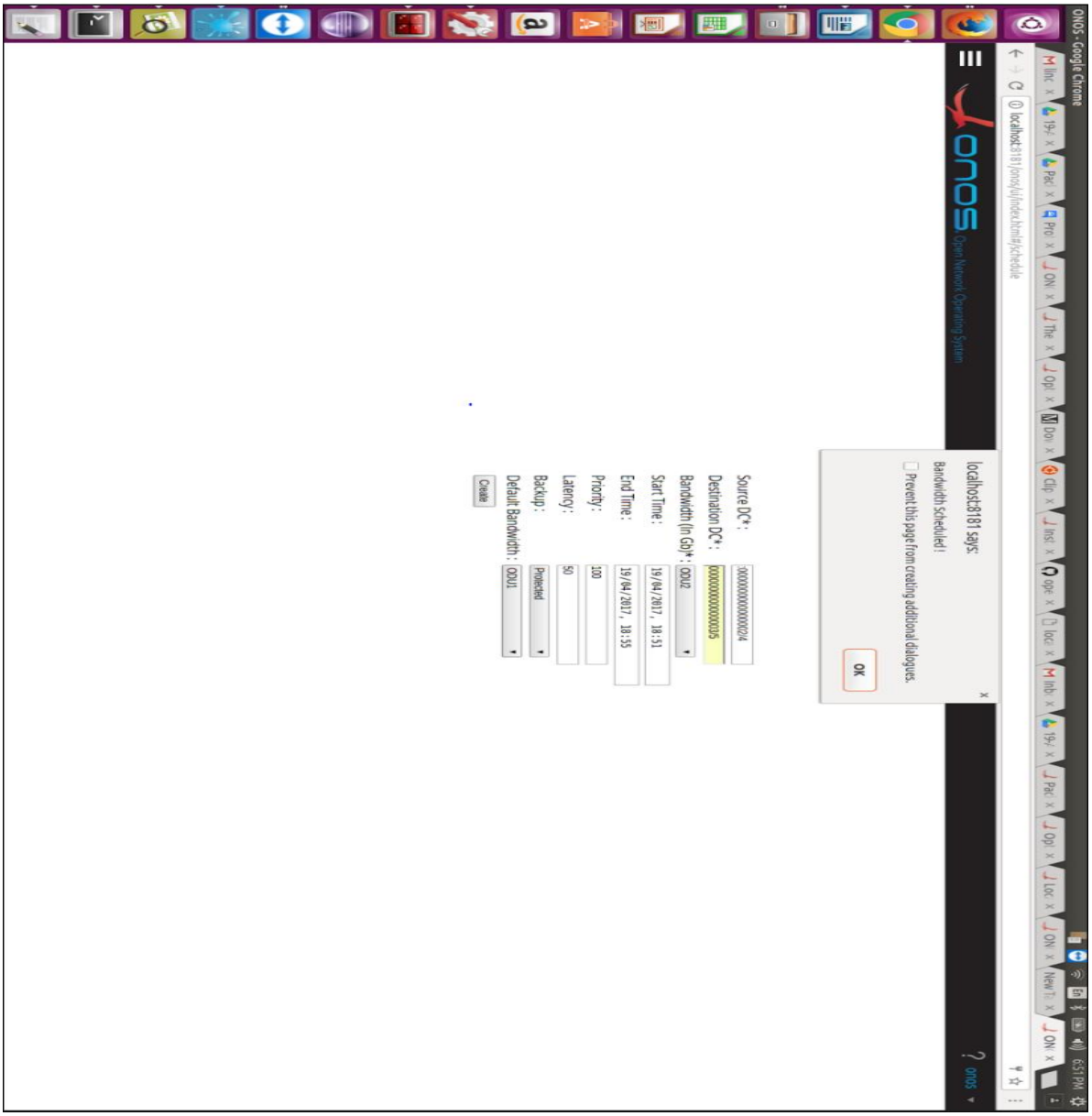


Figure 4.6 Scheduled bandwidth form and successful creation of Intent

When the scheduled time is reached and the intents are installed, clicking on their topology reveals the shortest interconnection between the source and destination datacenters. This shortest path is

calculated with the extended Dijkstra algorithm and k shortest path. In Figure 4.7, the red double line depicts the successful creation of interconnect intents for proactively protected scheduled bandwidth. For a pictorial view of scheduled interconnect being installed between source and destination nodes with the ONOS GUI, refer to Figure D.3.

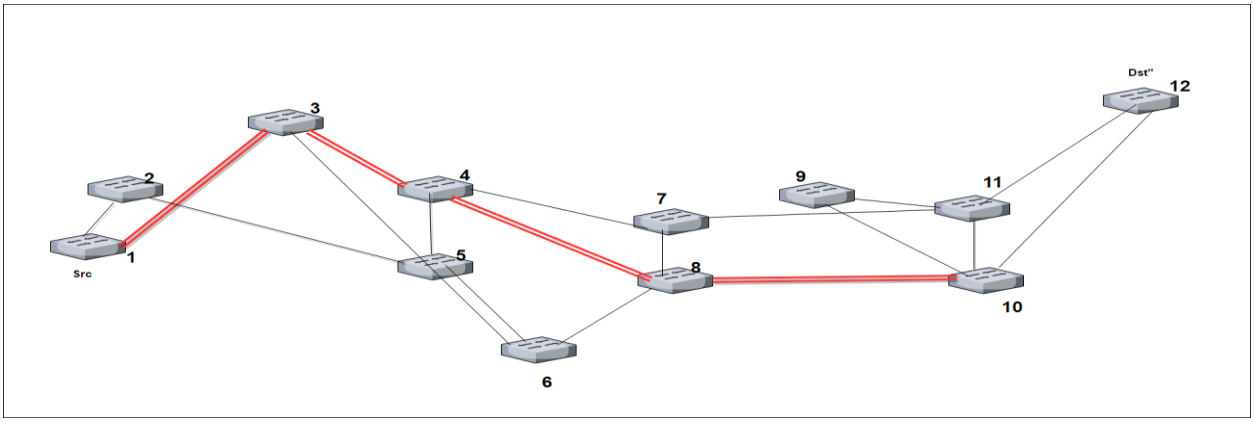
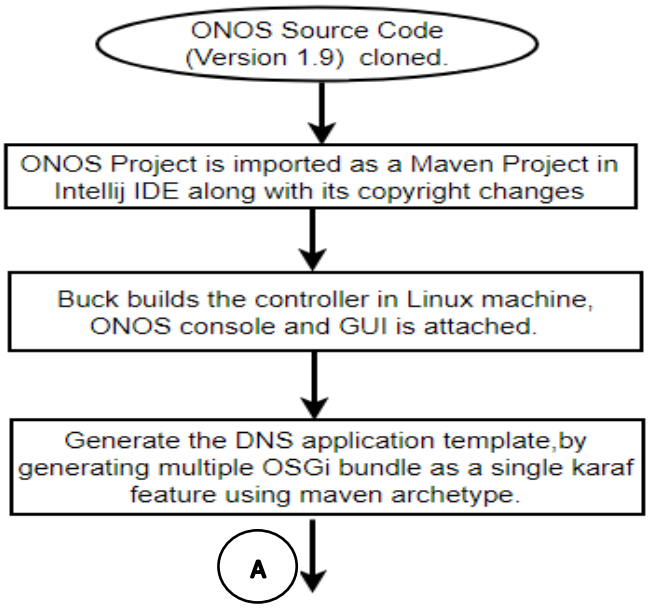
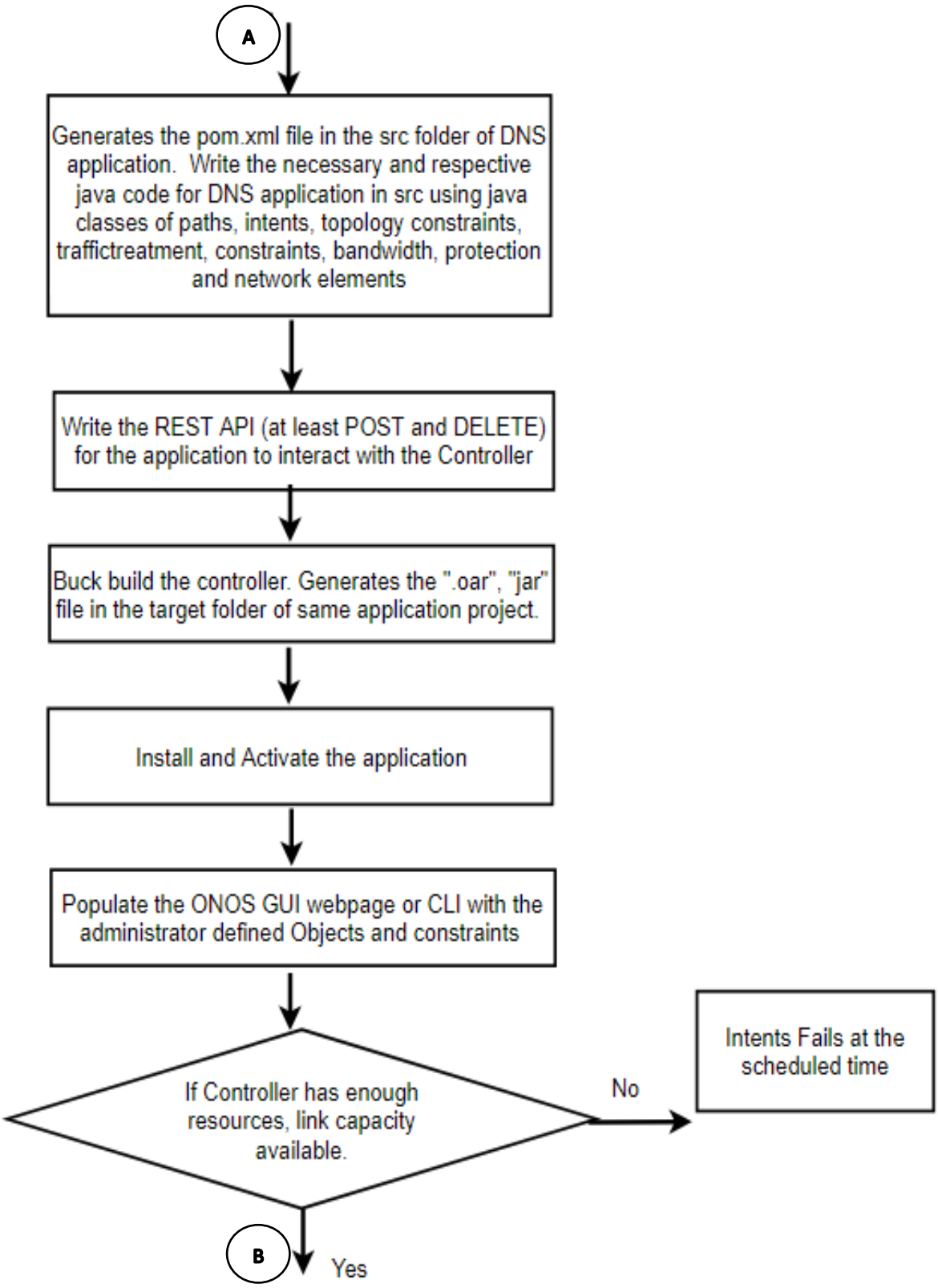


Figure 4.7 Successful protected interconnect intent.

4.5.10 Flow Chart for Simulation of DNS Application

In this section, we explain with the summary of steps in this simulation for the application with the help of a flow chart, as shown in Figure 4.8.




A
Generates the pom.xml file in the src folder of DNS application. Write the necessary and respective java code for DNS application in src using java classes of paths, intents, topology constraints, traffictreatment, constraints, bandwidth, protection and network elements
Write the REST API (at least POST and DELETE) for the application to interact with the Controller
Buck build the controller. Generates the ".oar", "jar" file in the target folder of same application project.
Install and Activate the application
Populate the ONOS GUI webpage or CLI with the administrator defined Objects and constraints
If Controller has enough resources, link capacity available.
No
Intents Fails at the scheduled time
B
Yes

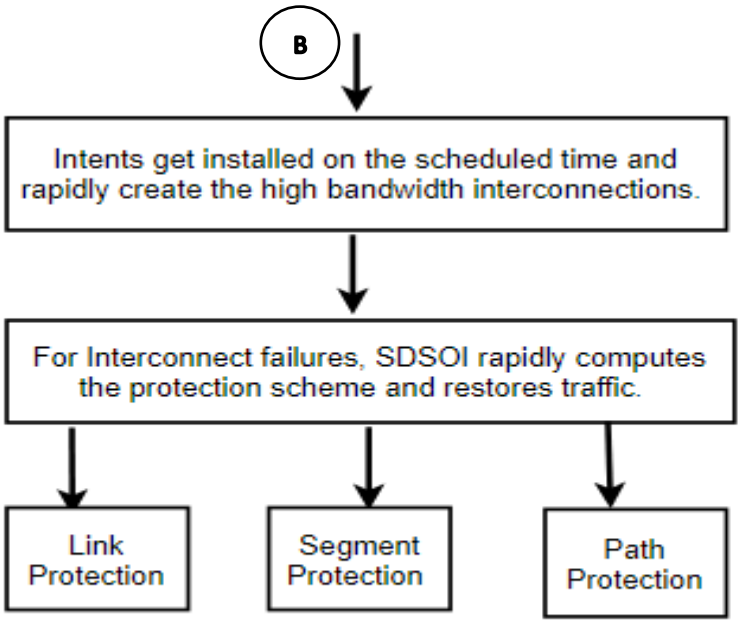


Figure 4.8 Flowchart representation of simulation of DNS

Chapter: 5 Result and Analysis

In this chapter, we describe the results of the proposed SDSOI-based architecture for the DNS application, and compute its efficiency. DNS is the external business application of SDSOI, as it determines the scheduled path and survivability of end-to-end DC communications. The survivability of these interconnections is evaluated under three protection schemes: link protection, path protection and section protection. To experimentally evaluate the efficiency and performance of our simulated application and proposed SDSOI controller architecture, we created the master SDN ONOS controller in the basic Linux machine, and connected it to the two Virtual Machines (VMs) acting as the DCs with the ONOS SDN controller installed.

We considered these virtual machines as a pair of geographically separate, distinct DCs. They are connected to each other and to the master SDSOI (service provider) controller. The DNS application is installed on the base machine's SDSOI controller, and has both online and offline application capabilities. Due to hardware limitations, we tested the SDSOI setup with only two VMs or DCs, running with distributed control to a hierarchical SDSOI controller.

To verify the performance of SDSOI and DNS in larger topologies, we used Mininet as the network emulator and considered devices in the Mininet to be optical gateways connected to the DCs SDN controller. Using Mininet as an optical gateway topology emulator, we applied Google WAN and PAN European as reference networks of the optical gateways.

We observed the flow rules that get installed and exchanged from the SDSOI controller to the optical gateways via Wireshark (a packet capture tool), and a control plane monitoring application that is an integrated ONOS plugin already developed as an application in ONOS default applications. Controller efficiency is also measured with a third-party performance-monitoring tool, which observes the performance related to the controller's CPU usage, metrics and 'Collectd'. At the end of the chapter, we analysed and compared our results with others' work in transport or service provider's SDN controller performance, in terms of latency, bandwidth and restoration time.

5.1 Evaluation of Day Night Scheduling Application

In this section, we illustrate the performance of DNS applications in Google WAN topology. We evaluate the performance from an administrator perspective, and illustrate the scheduling and protection performed by the intents created between any pair of the optical gateways.

5.1.1 Day Night Scheduling Application

Day night scheduling is a smart business application, which performs optimal resource allocation and utilization of the interconnect services. It supports scheduled and varied bandwidth demands, as well as protection for different time slots during the day. The objective is to maximize the admitted demands with high bandwidth connections at pay-per-use costing. DNS application has the following goals:

- efficient packaging of OPU's and small ODUs to larger ODUs;
- efficient and optimal use of bandwidth;
- optimized path calculation with minimal cost with available capacity;
- survivability of the bandwidth and interconnects;
- automatic downsizing of bandwidth when scheduled time expires;
- maximizing the volume of demands served; and,
- high bandwidth connection.

We achieved these objectives using the simulation setup described in Chapter 4. In this section, we assess the testing of the application, and highlight some results based on scheduling and protection.

For an administrator to create an interconnection between any two DCs and their optical gateways, they must define their needs in terms of interconnection policies. These policies contain the rules of how an interconnect should behave with the built-in constraints. Therefore, when we schedule the bandwidth between the gateways it creates 'intents', which are translations of applications policies and rules and specify the interconnection behaviour to the controller. The controller uses its intelligence to perform the intensive processing and computing, and installs the interconnection. The intents are automatically installed at the scheduled time via their respective optical gateway network, and they interact based on the constraints defined by that object. The objects are input

parameters defined by the administrator, and include latency, priority, type of ODU and protected or unprotected connections. The type of protection and restoration scheme to be deployed upon failure is computed by the controller, using the logic of coded algorithms and calculations for parameters such as link capacity, cost and path availability. We also demonstrate an example in this chapter for intent creation between optical gateways 1 and 12 connected to their respective DCs, and the scheduling of bandwidth between them. Moreover, we further explain how the paths are scheduled, and verify their protection and restoration.

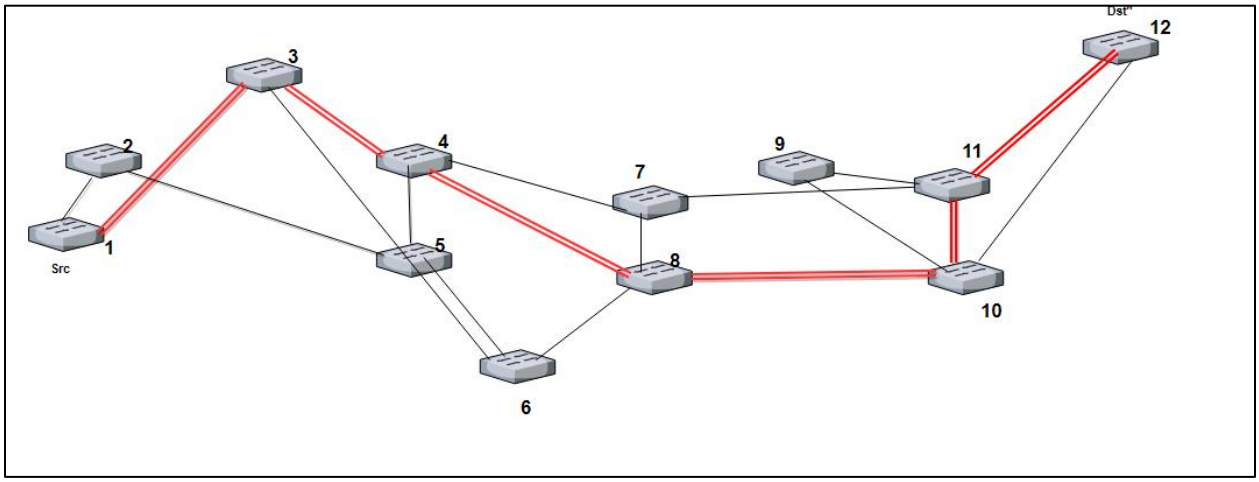


Figure 5.1 Route computed by scheduled bandwidth intent interconnect for DNS application.

Figure 5.1 shows the scheduled bandwidth interconnections between DC1 and DC 12 when the intent is installed. These interconnections are programmed to deploy whenever the scheduled time is reached. Intents are pre-scheduled proactively to make optimal use of bandwidth globally, following the concept and pattern of Sun (i.e. more bandwidth in daytime and reduced at night). Intents are installed only when all the tabs in the application form of the GUI are populated correctly, or when correct CLI commands are entered with all the objects in them defined. The GUI displays the following message after successful creation of intent and bandwidth: "localhost8181 says bandwidth scheduled" which lets administrator know that the policy is successfully installed and sent to the controller. These intents are installed if they are created correctly, otherwise they fail. Intents generally fail when they do not get the appropriate input parameters (e.g. port numbers), or the correct optical gateway numbers because their addresses are long and in hexadecimal format; unlike the graphical display values on the GUI. Although ONOS

allows the devices to be renamed, their device ID or MAC address remains in the same hexadecimal format.

SDSOI can deploy multiple and different bandwidth interconnections among different sets of gateways at any time, limited by the interconnect capacity. For testing and verification of this application, we created multiple intents for the same time between different source and destination nodes. When all the intents are installed, we found that many of the interconnections have common paths between them. For testing purposes, we programmed and assumed that the total capacity of each fiber interconnect was 500GBs, and once that is exhausted intents will fail to be compiled. DNS application was tested with various sets of source and destination gateways and in different topologies, as demonstrated in the test cases designed for the application in Appendix C. In the next section, we discuss the basic recovery scheme and procedures.

5.1.2 Link Protection in Day Night Scheduling Application

For link-based protection and recovery, once the topology discovery is complete, the controller automatically computes the backup sub-path for each interconnect link in the topology; a sub-path is defined as the path between two direct ends of interconnects. Whenever an interconnect fails, traffic is redirected to the backup sub-path of that interconnect and the rest of the path does not change. Figure 5.2 shows an example of link protection and recovery by the SDSOI controller. In the example, the flow rules are installed by the controller to a bi-directional path from gateways1-3-6-8-10-11-12 for a scheduled interconnection between DC 1 and DC 12, as shown in Figure 5.1. When an interconnect link in the traffic flow between DC1 and DC12 breaks between gateways 10-11, the SDSOI controller automatically restores communication by redirecting the traffic for that link to an alternate sub-path of gateways 10-9-11, as shown by the dotted red line in Figure 5.2.

In the event of link failures, the SDSOI controller performs complex computations, and then applies the traffic and flow rules to an alternate sub-path for the failed link. We observed rapid installation of a new sub-path for that link, and traffic being redirected to the new sub-path was instantly added to the script log of the server and the ONOS GUI. Examples of these iterations in Google WAN and PAN European topology with different sets of source and destination addresses are shown in Appendix C and Figures C.2, C.6 and C.10, and further explained in their corresponding sections.

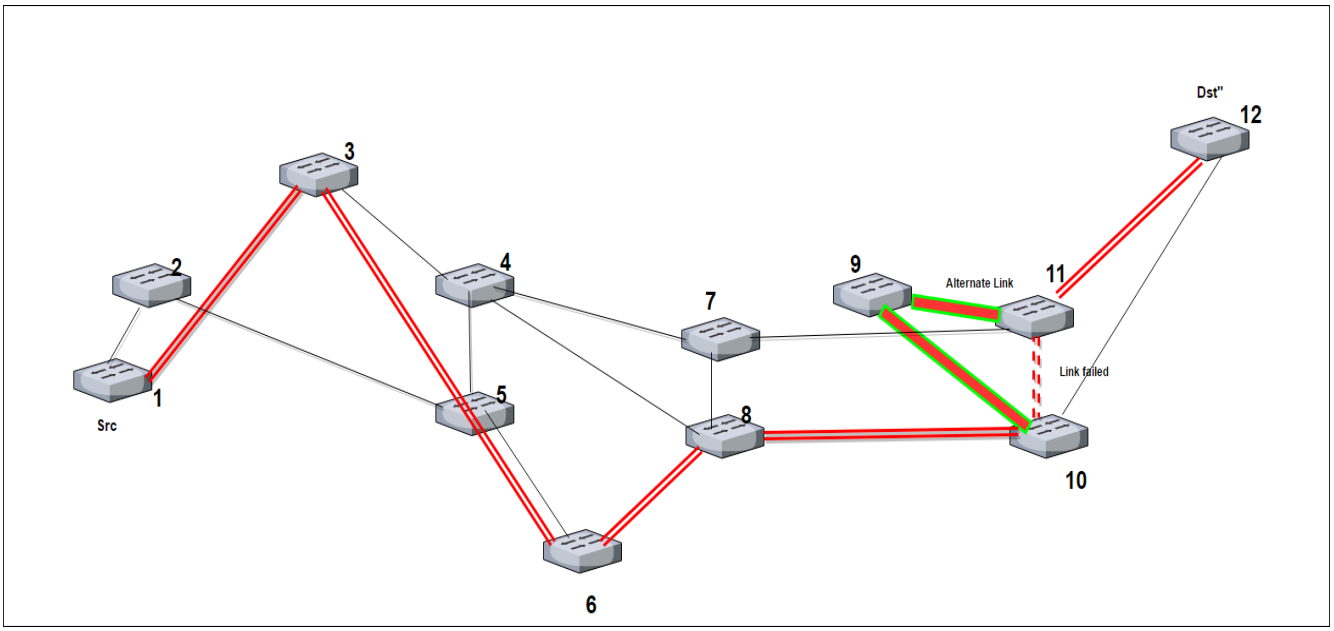


Figure 5.2 Routing behaviour of the scheduled interconnect for link failure and restoration in the DNS application.

5.1.3 Segment Protection in DNS Applications

Service providers generally build and use segment (i.e. a group of links) protection and restoration for sections in the geographical areas that are highly prone to failure. To perform segment-based protection and restoration in the network, the SDSOI controller computes the alternate protection sub-path for each segment in the network, as described in Chapter 3.

Segment-based, alternatively protected sub-paths are installed by the active intents of the interconnection, using idle timers in the flow rules. The controller immediately sends the update messages to the backup segment to be installed, and informs the incoming traffic flow for the change in path. It also updates the gateway idle timers to prevent automatic flow removal. Figure 5.2 depicts an example of segment protection in which the working path of the interconnection carries the traffic from the source gateway DC1 to the destination gateway DC12, via gateways 1-3-6-8-10-11-12. We assumed that the optical gateway topology is divided into three segments, as shown in Table 5.1. In segment 2, when links 3-6 and 6-8 in the network fail, SDSOI immediately installs new backup sub-path 3-5-6-8 to replace the working path 3-4-7-8, and restores the traffic flow to this backup protection sub-path. This sub-path is already computed as the disjoint sub-path

by the controller for that segment, as shown by the solid red and green interconnections in Figure 5.2.

Segment protection and restoration by DNS is iterated for various segments of several network topologies, under different pairs of source and destination nodes and with different types of bandwidth and latency demands. A few cases and examples of segment failure and restoration of traffic by alternate segments are shown in the second part of all cases of Appendix C, and depicted in Figures C.3, C.7 and C.11.

	Optical Gateways					
Segment 1	1	2				
Segment 2	3	4	5	6	7	8
Segment 3	9	10	11	12		

Table 5.1 Segments considered in the evaluation of Google WAN topology.

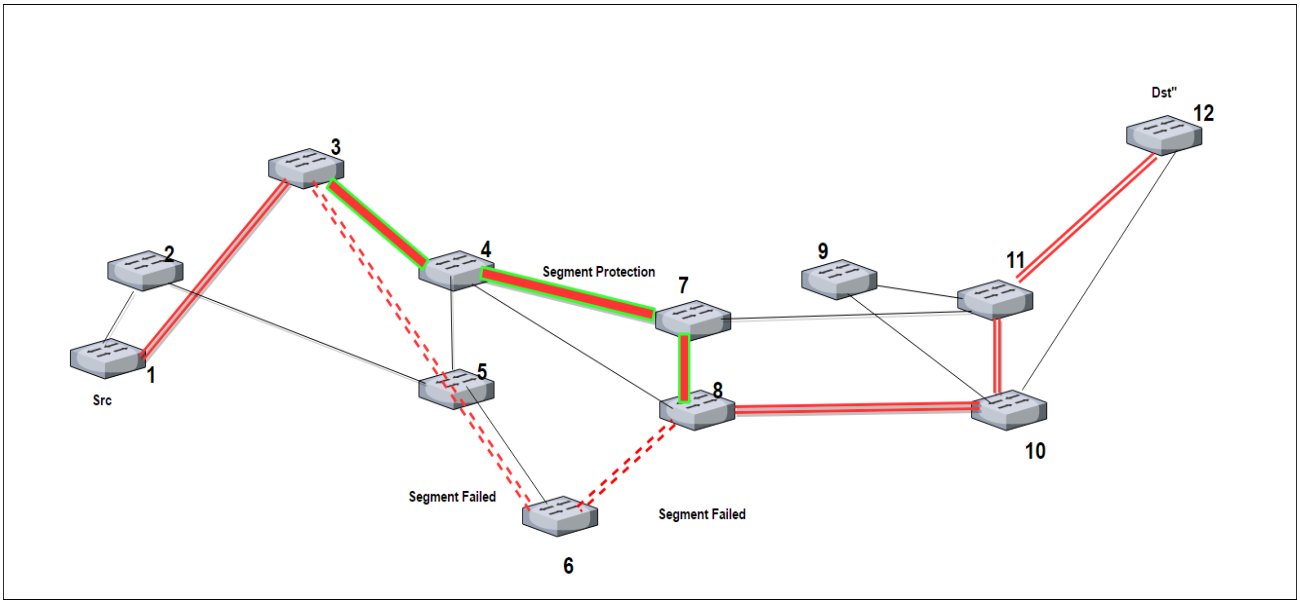


Figure 5.3 Routing behaviour of the Scheduled interconnect for segment failure and restoration in the DNS application.

5.1.4 Path Protection in the DNS Application

To perform path-based protection and restoration, the SDSOI controller calculates the disjoint backup protection paths for all intents that are created. The computation of this new protection path requires that the SDSOI controller to have global and centralized views of the entire network,

including precise topology information for all the gateways and the link capacity of each interconnection. Path-based protection and restoration for all interconnection intents are calculated using both the disjoint path Dijkstra algorithm, extended for simulation in this work, and the k shortest path algorithm. When synchronized, these two algorithms install the next best possible and shortest protection path, with exactly the same constraints as the working path.

Compared to the two protection and restoration schemes above, upon failures in path-based protection and restoration traffic crankbacks to the source gateway, from where it is re-rerouted to the destination gateway using same constraints as the working path, but on a new disjoint path. This type of protection does not significantly increase the overall cost for the interconnection, as it utilizes approximately the same amount of resources as the working path. However, it greatly increases the traffic restoration time, as the SDSOI controller re-computes the complete new path with the same constraints and objects.

Figure 5.4 shows an example of path-based protection and restoration computed by an SDSOI controller for DNS applications. We used the same scheduled intent interconnection as in Figure 5.1, where traffic is flowing from DC1 to DC12 via optical gateways 1-3-6-8-10-11-12. For this interconnection, we assumed links 1-3, 8-10 and 10-11 in the working path fail, as shown by dotted red lines in Figure 5.4. To restore this interconnection and maintain the services and traffic flow, the controller immediately installs a new, identical protection path via gateways 1-2-5-4-7-11-12, and restores the traffic to it as shown in Figure 5.4. The path-based protection and recovery of the DNS application is iterated in several topologies, under various combinations of pairs of source and destination gateways. A few of the test cases for path protection in Google WAN and PAN European topology, as well as different pairs of source and destination gateways, are shown in Appendix D, and depicted in Figures C.4, C.8 and C.12.

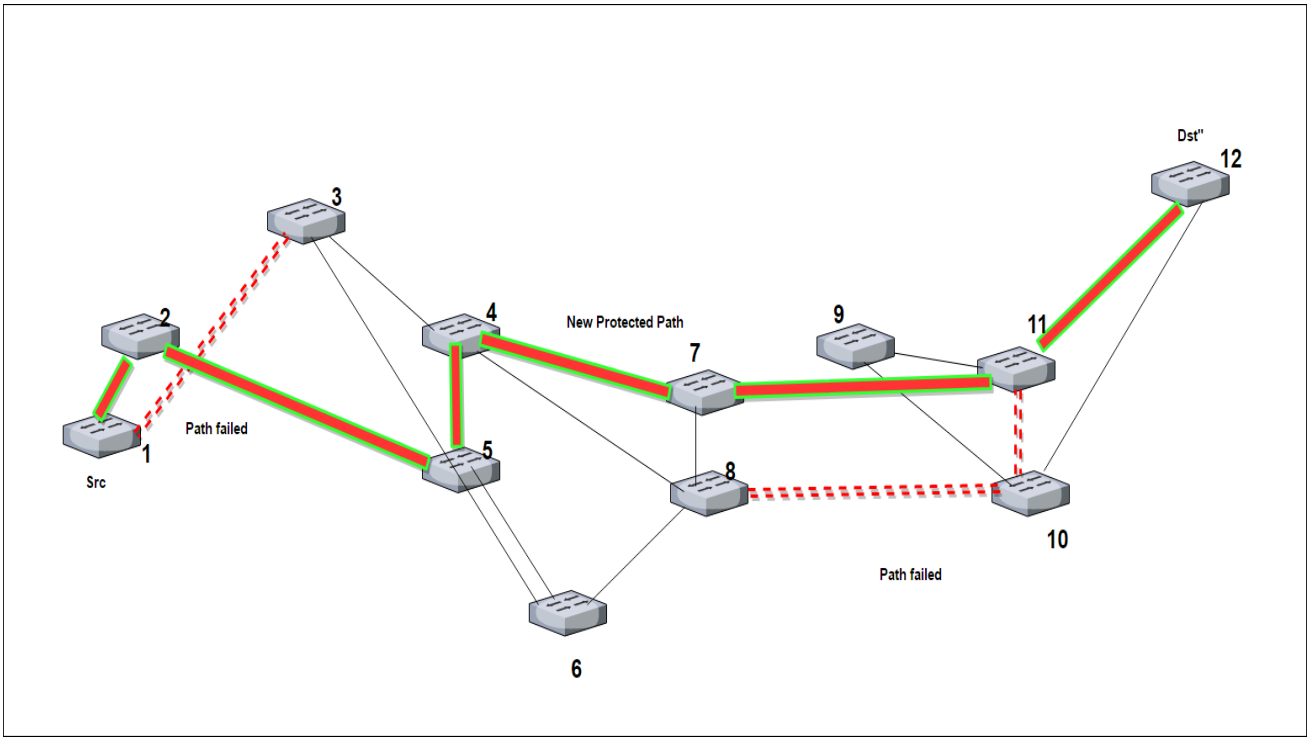


Figure 5.4 Routing behaviour of the scheduled interconnect for path failure and restoration in the DNS application.

5.2 Performance Analysis of DNS Application

In this section, we demonstrate the performance and results of the DNS application, and observe the network packet behaviour when creating the DCI. We also investigate the behaviour of restoration and protection upon interconnect failures, in terms of new flow rules being installed for the following restoration schemes:

- Link Protection;
- Segment Protection; and,
- Path Protection.

5.2.1 Packet Capture with Wireshark

Our application creates intents in the network that define the policies and rules subjected to various constraints, and describes the behaviour of interconnection in the network. At the start of the scheduled time, the controller automatically installs the intents, and defines the flow rules for that interconnection. OF protocol defines the flow rules among the gateways, and stores them in the

flow tables. We use OF Version 1.3, which is compatible with the IP and optical layer, and has great potential to handle large volumes of data. SDSOI defines these flow rules with flags in the request and reply fields.

To implement the intent installation, the controller initiates the packet sending process among all the gateways used in the interconnection. These gateways reply with empty-bodied messages and acknowledgments back to the controller to start the packet transfer. The controller then sends and installs the flow rules to all the gateways and its interconnections, using OFPT_PACKET_OUT. Whenever there is a new demand to a gateway it includes an OFPT_PACKET_IN message back to the controller, which checks the policies and rules installed by the application for such arequest, and performs the heavy computation and calculation. It then installs the flow rules for that demand using OFPT_PACKET_OUT. This initial conversation is repeated between all the DC gateways and controllers, as shown in Appendix D and Figure D.2 on a Wireshark capture. Every packet that is exchanged in the form of an OF message contains the flow rules of type, length, buffer and action, as explained in Chapter 2. With Wireshark, one can observe this in the packet capture description, which determines if it was an ONOS (SDSOI) discovery or not, and captures the gateway ID created for this request.

Figure D.2 shows the capture of how OF messages are exchanged by the controller and gateways in the early phase of intent installation. When the SDSOI controller receives DC demands or requests via optical gateways topology, it sends the OFPT_PACKET_OUT messages to the gateways and the interconnections for the traffic flow. Once the connections are established, the controller sends the queries to the active DC and gateways of that intent, to provide the statistics of the groups of DC that are present, their descriptions in terms of BW type and connection, the latency and other factors. For these controller exchanges, the requests and replies are in the form of OF messages, such as OFPMP _GROUP and OFPMP _GROUP_DESC, and they also inquire about OFPMP METER requests.

5.2.2 Results and Discussion

This section examines the results and analysis of the tests performed in the SDSOI DNS application under different topologies, with various sets of source and destination gateways and under different types of protection and restoration schemes. As discussed above, we observed

different protection and restoration schemes, how long it takes an interconnection to be installed, and computed the recovery time of the interconnections under different protection schemes for failures. We measured the installation and restoration times of the interconnections, in terms of how long it takes the flow rules and open flow messages to be installed using the Wireshark packet capture tool.

When interconnection is established and any link, path or segment in the network fails, the intents remain in the installed state and do not fail. For link and segment protection, traffic does not return to the source gateways for a new path request; rather it automatically switches to an available, already calculated disjoint alternate link and section of the same capacity. However, for path protection traffic returns to the source node, and takes a new available alternate disjoint path route with same bandwidth capacity. This is shown by the link, path and segment protection curves in Figure 5.5.

Figure 5.5 shows the average interconnection and recovery time taken by an intent that is installed and initiates the traffic flow under different protection schemes. The graphs are based on the average results of numerous iterations (30 for each protection scheme) performed for each topology, with various combinations of source and destination gateways and different ODU types and latencies. DNS interconnection time in Figure 5.5 represents the average time taken by that topology for the scheduled connection to be established. The link, segment and path protections plotted in Figure 5.5 show the restoration time taken by the SDSOI controller to re-establish the traffic flow in the respective topologies when failure occurs.

As discussed above, protection built by the DNS application for scheduled interconnections help reduce restoration time in the network. Although the recovery time in any of the protection schemes is slightly higher than the scheduled intent interconnection time, it is much less than the time required for the interconnection to be re-established. As this protection is proactively built by the controller from DNS intents, and the amount of bandwidth utilization via each gateway and interconnection is already known since everything is prescheduled, it takes less time for the interconnections to be restored if they fail. Link protection takes the least time of all the protection schemes, as it only switches the traffic to an alternate link between the gateways of same capacity, while rest of the path is unaffected. Segment protection takes slightly longer since the traffic reflows from the failed segment to a new alternate segment sub-path. Path protection takes the

longest, as the traffic must be redirected to a new disjoint path. Faster restoration with the same QoS and bandwidth is very important at service provider levels, as it helps maintain the committed SLA's to the customer, and network failures become imperceptible. Thus, DNS application reduces the latency time in the network, and provides faster restoration. Another advantage is, unlike traditional protection and restoration mechanisms, it does not reserve the bandwidth for each interconnection in advance.

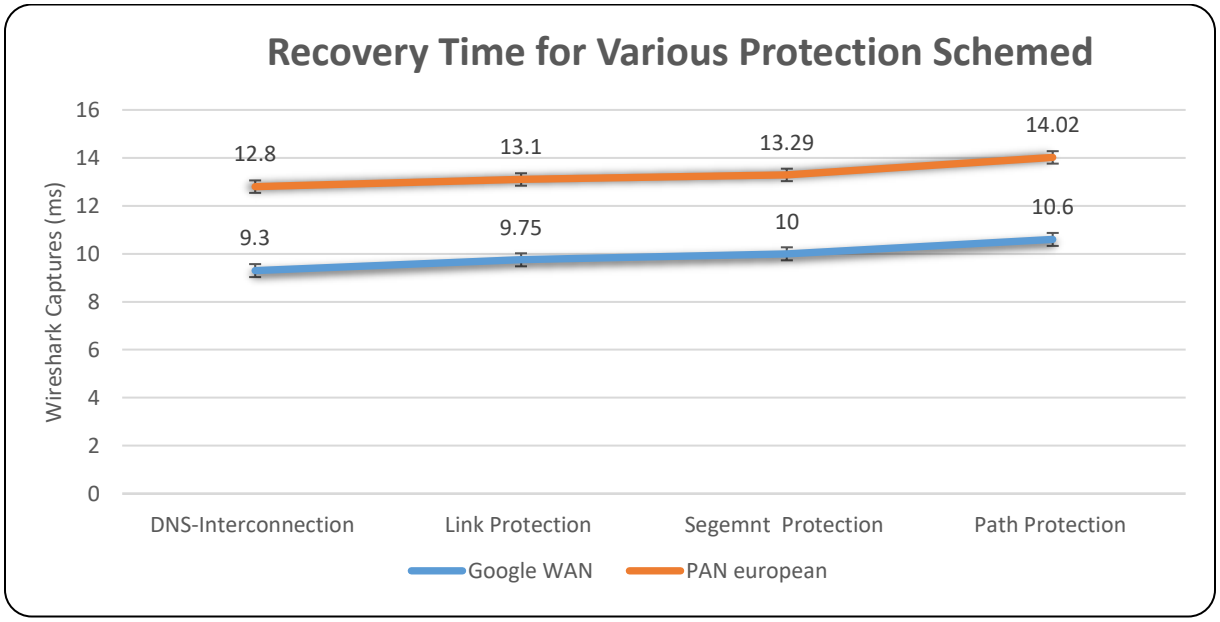


Figure 5.5 Recovery time of SDSOI interconnects for various protection schemes.

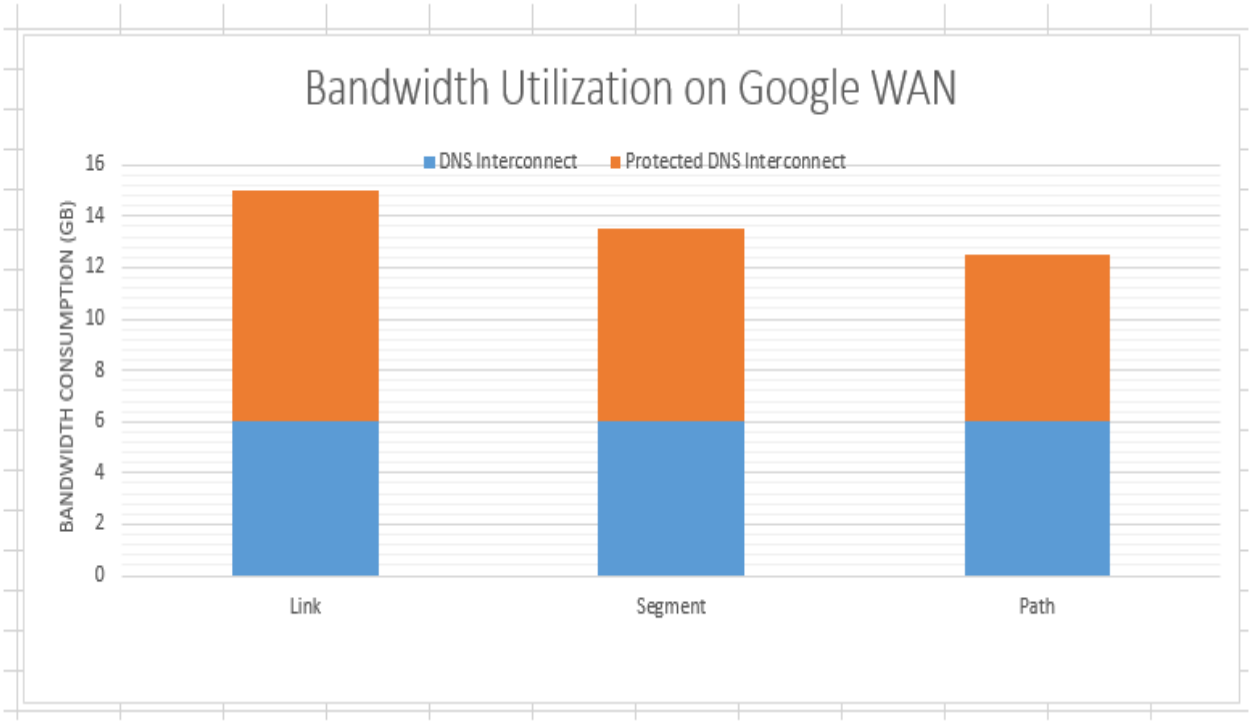
When failures occur, SDSOI commits the same bandwidth restoration, and to achieve the same bandwidth interconnection it takes the new available disjoint path, link or section, depending on the availability and type of protection that is built and the kind of support offered by the controller to that interconnection. To maintain this commitment, the controller consumes extra resources; that is, it uses other links in the network that are of same bandwidth capacity with same traffic, constraints and object qualities.

We also calculated the efficiency of our application in terms of extra bandwidth and resource utilization in the network for each of the protection schemes plotted by the graphs in Figures 5.6(a) and 5.6 (b) for Google WAN and PAN European topology respectively. These graphs show the average statistics of the iterations performed for the path that are consumed by the application when interconnections are scheduled in blue, while modifications in the new path and resource utilization by the controller upon restoration of traffic with the protection scheme is depicted in

orange. These modifications and reinstallations of the path always use additional resources. Figures 5.6(a) and 5.6 (b) shows how average resource utilization is highest for link protection that has the lowest restoration time of the schemes. However, this increase is considerably smaller for path and segment protection, and they have higher restoration times.

The graphs are plotted on the percentage increase of resource consumption (bandwidth and links) in each type of protection scheme, compared to the resource utilization of the original scheduled interconnection. For each protection scheme, approximately 40 iterations were performed to calculate the change in resource utilization when failure occurs, for different sets of source and destination gateways. The highest percentage of resource consumption increase is with link protection, as this available alternative link for a failed link has more intermediate links. These intermediate links are also installed with the same QoS, bandwidth and latency, thereby increasing their total bandwidth utilization. In contrast, for path and segment protection there is only a slight increase in resource consumption, as the protection path that replaces the working path has about same number of links and almost same bandwidth utilization. Though there is more bandwidth capacity available in the network with both the path and segment protection schemes the restoration times are slightly higher, as the SDSOI core takes more time (in ms) to recompile, execute and install the disjoint available route.

Figure 5.7 plots the efficiency of the controller in terms of average percentage of traffic throughput transmitted by the controller and received by the optical gateways. This is measured using the ganglia Gmond for optical gateways, and the Gmetad for the controller for test cases and iterations performed in 48 hours of continuous traffic flow. The graph clearly shows that with scheduling and pre-awareness of traffic demands, SDSOI packs all the ODU and OPU demands more efficiently, and maximises traffic flow and service continuity.



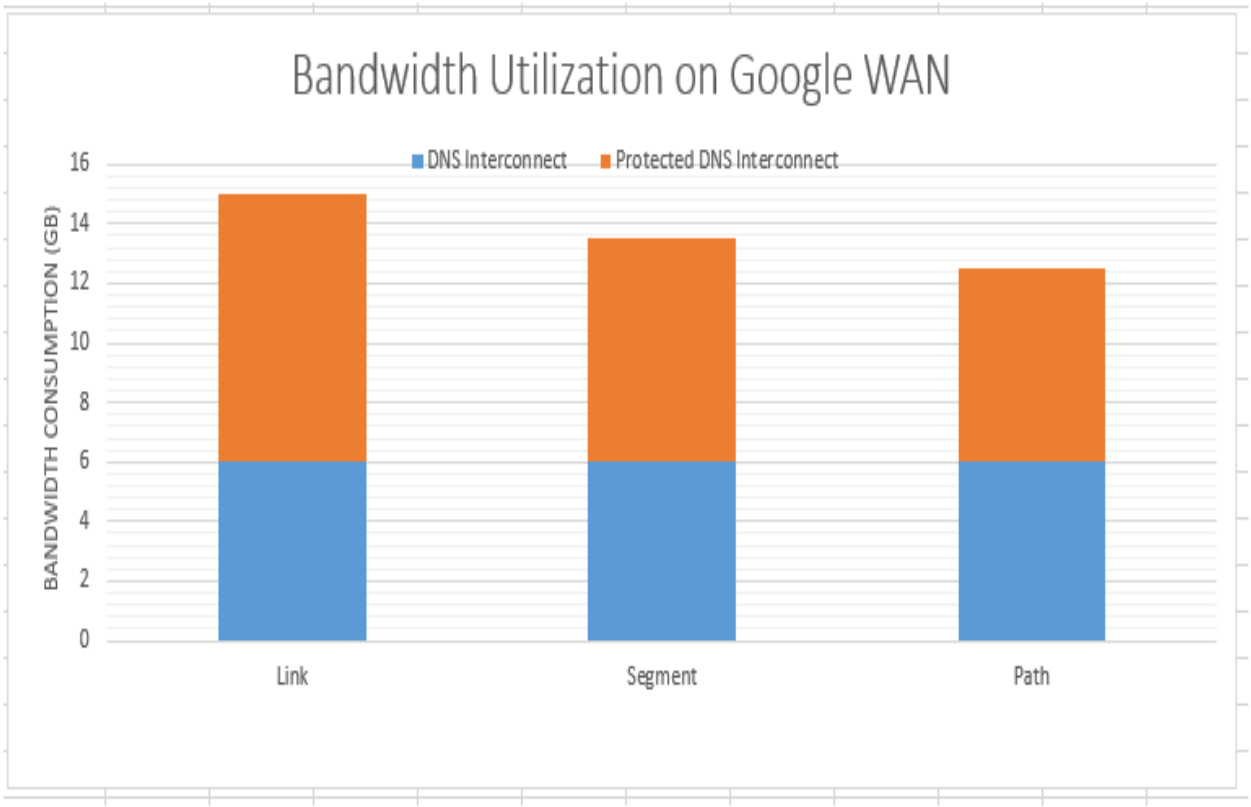


Figure 5.6 Bandwidth Consumption of DNS interconnects for Link, Path and Segment Protection in (a) Google WAN topology (b) PAN European Topology.

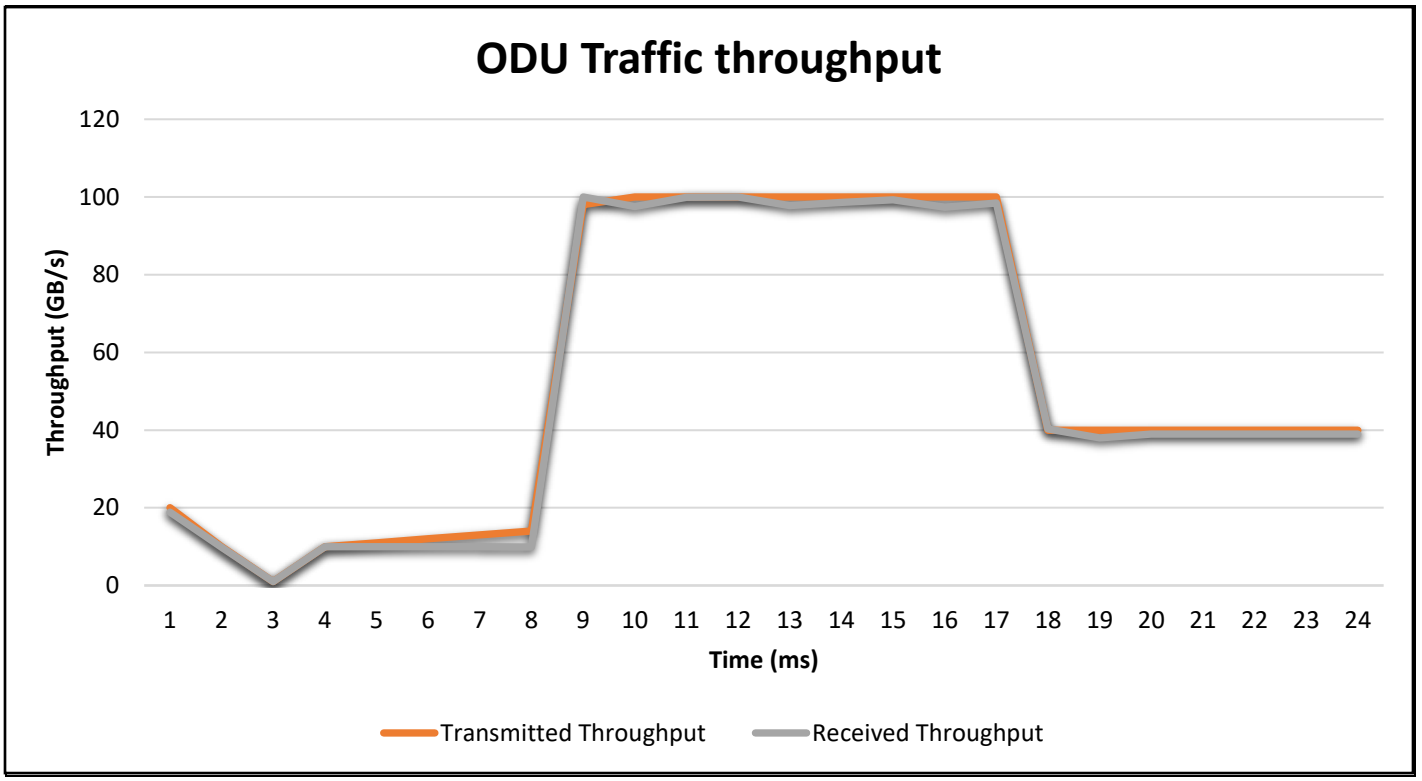


Figure 5.7 ODU traffic throughput monitoring.

Figure 5.8 displays a graphical overview of multiple intents being installed in the topology, with different bandwidths for different timeslots in the day between some pairs of DC's. To explain this, we assumed that the day is divided into three timeslots: morning, afternoon and evening. As these DC's are geographically apart and in different time zones around the globe they can take advantage of the zones and follow the sun, which allows us to provide all locations with extremely high bandwidths. We can clarify Figure 5.8 with an example: If we assume that DCI service 1 is in timeslot 1, they perform data and VM replication between the source and destination DC using 40G of bandwidth. In timeslot 2, the active hours of operation and connectivity use 100G bandwidth, while the timeslot 3 bandwidth is downsized to 10G as they only maintain the basic operations and connectivity. With this automatic change of bandwidth by intents for any interconnection, service providers can optimize their bandwidth usage and save the DC cost, since they use the "pay what you use" model. Similarly, the remaining bandwidth is used by other interconnect services in other parts of the world. Figure 5.8 also provides an overview of how efficiently the total 150 G interconnect capacity is used by the controller with shrinking and

scheduled bandwidth, and how we could optimize the use of high bandwidths across different global time zones.

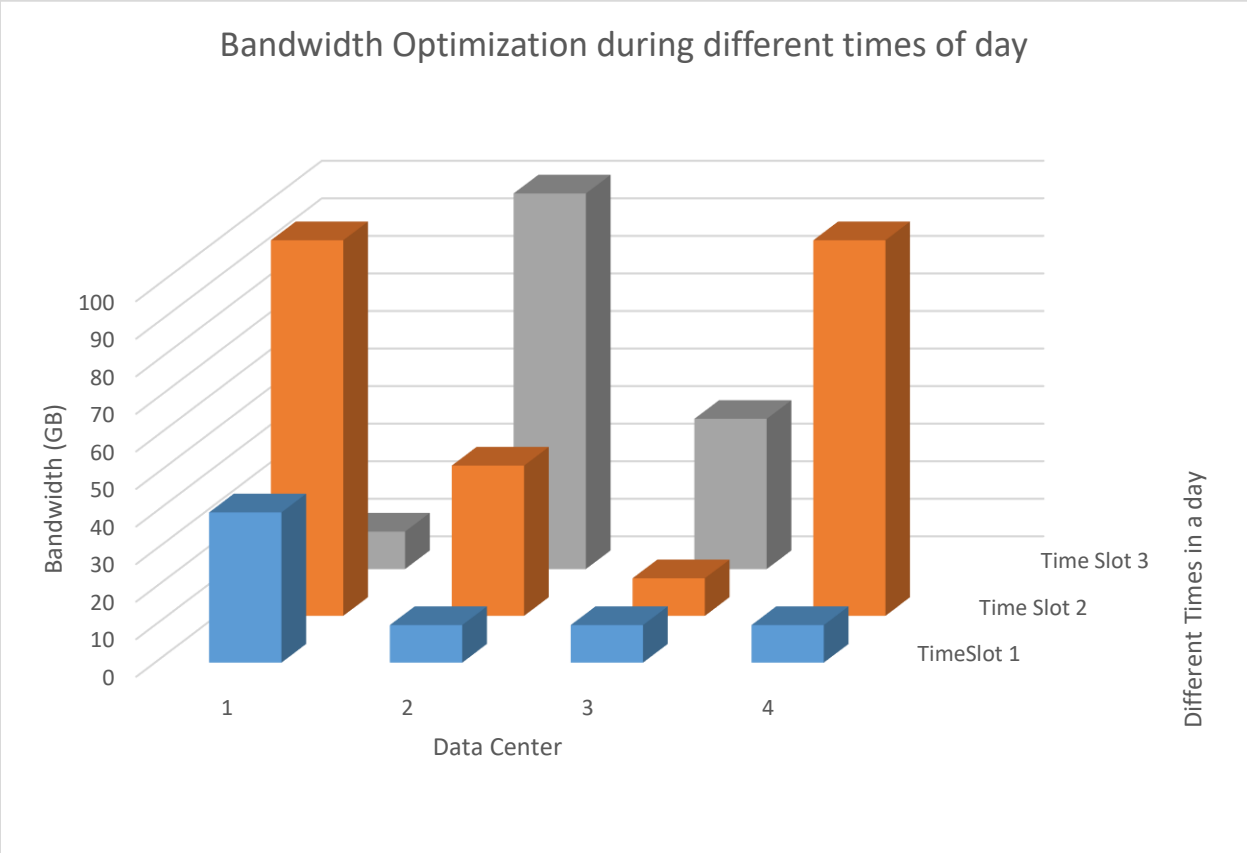


Figure 5.8 Effective Day Night Scheduling and Bandwidth management amongst the Optical Gateways

We compared the above results of DNS applications deployed in the SDSOI model to the work done in [35]. In SDSOI, with a single multilayer controller working as an application and transport controller, we can provision optical DC interconnects in Google WAN topology on an average as low as 9.3ms of provisioning or installation time. Results of the relevant work done by the researchers in [35] with different distributed application and transport controllers, show that it takes 11.3ms to provision the interconnect in the same topology. This indicates that our model is superior, as even for failure in the network it takes only maximum of 10.06ms, significantly less than the 11.3ms required by [35] to establish the connection. In addition, we have protection and restoration services on-board our application, which reduces the risk of failure and has higher quality potential than LaSS [35].

Chapter 6: Conclusions and Future Work

In this chapter, we review the main contributions of the thesis, and discusses future opportunities.

6.1 Conclusions

Innovation in Software Defined Networking (SDN) has successfully made its way into Next Generation Networking and development environments. The frameworks proposed in this thesis, and in extensive related work, show how SDN for transport providers can meet the high bandwidth requirements of Data Centers (DC) and their optical interconnects, and further advance automation and orchestration in service provider networks.

We propose a SDSOI model for transport service providers, with scheduled policy-based bandwidth connectivity for effective protection and restoration, and load sharing of interconnects in the event of failure. This architectural solution is proposed for global transport providers with geographically distributed DC connectivity, as it will help them provide high bandwidth connectivity across the globe. This solution provides high bandwidth connectivity, reduce latency, ensure higher QoS, schedule optimized bandwidths and lower restoration time for interconnections between geographically separate DCs.

Unlike traditional optical interconnects that require predefined protection (shared or dedicated) to restore a signal when failures occur, SDSOI provides proactive protection of interconnect services for failures without the need of advanced bandwidth or path reservation. SDSOI provides protection and continues routing even if intent for the route is unprotected. It performs active and live monitoring of network elements, and uses neighbour discovery to make network failures so imperceptible that end customers don't realize a failure occurred.

The SDSOI controller core is designed with some unique predefined mandatory plugins as part of its architecture. The plugins perform the primary connectivity operations, and are actively involved in computation, monitoring, load balancing, packaging and management of the interconnect, in such a manner that higher bandwidth needs are sufficient along as well as flexibility.

SDSOI is not only unique because it has inbuilt mandatory plugins for end-to-end DC connections and troubleshooting; it also has a smarter way of categorizing applications to handle ever-changing business demands. The SDSOI core, also known as the 'black box', performs the complex

computation task for interconnecting geographically distributed DCs while physically residing in a cloud. It provides high-end interconnects with GB and TB of OTN bandwidth and lower latency. The core is specialized for optimum bandwidth utilization by assigning prescheduled bandwidths at different times of the day, and downsizing them to different levels when the scheduled time expires. During the scheduling activity, the original scheduled intent is automatically withdrawn from the network, and the new intent is installed with the required reduced bandwidth defined by the administrator while creating the intent.

SDSOI is a multi-layer, multi-domain and multi-vendor global controller designed with open interfaces in both the northbound and southbound layers, and it is independent of any type of protocol for any layer. The proposed controller uses open interfaces so it can support any type of optical gateways, as well as DC SDN controllers. Since we are connecting the DC at global level using open interfaces, there is no limitation on the protocols used to connect them, and industries do not require the same vendor optical gateways and SDN controller.

To further demonstrate its capabilities and efficiency, we simulated a policy-based application for the SDSOI controller that we call ‘Day Night Scheduling’. This allows administrators to create immutable policy intents that can be tailored to the needs of each interconnect. These intents are highly time sensitive, and are automatically installed at the scheduled time to start delivery of the intended services for which they were created. The intents produce policy-based rules on the core of SDSOI based on objects such as topology, constraints, active and default bandwidths, protection and cost. When successfully coded and built, they are automatically installed in the network and perform the bandwidth connection services. The intents continue to provide the defined services, even when there are modifications in the network topology, or there is a version or software update for gateways.

Once installed in the network the intents define the policies for the controller, which then does the computations according to various factors such as priority, latency, bandwidth utilization, minimum cost computation, shortest available free path, restoration time and type of restoration. After considering these factors, the SDSOI core calculates and installs the optimal end-to-end interconnect routing for the source and destination datacenters, in the form of intents. Multiple intents designed for different purposes can be installed simultaneously for different applications.

Intents and applications are interrelated via unique application and intent IDs, and an application can install multiple intents in a network that are identified by their specific Key ID.

Though the experimental SDSOI architecture was built using ONOS as the SDN controller, it can be used with any available open source SDN controller platform, and can be open-swappable if required. Proposed SDSOI architecture efficiency is evaluated in terms of the DNS application, built with multiple features. SDSOI and DNS performance is evaluated according to intent creation, type of protection provided, restoration time, kind of bandwidth and resource utilization by proactive protection provisioning in the network. Controller efficiency is also measured in terms of multiple simultaneous ODU's from a single gateway at a particular time, and by the controller's intent with standing capacity (of multiple 40G and 100G).

The SDSOI experimental setup is a perfect example of combining hierarchical and distributed SDN controllers, using Overlay and Underlay SDN model respectively. With the overlay model, a hierarchical global SDSOI controller creates the interconnection for the geographically distributed and virtualized SDN DC controllers, using packet optical gateways. The SDN controller inside the DC follows the underlay SDN architecture, and performs deep packet inspection and routing. As SDSOI is a globalized controller, it performs diverse bandwidth scheduling for different times of day, and follows the sun pattern to provide high speed, low latency interconnections globally. It automatically reduces the bandwidth level once the active time is over, and uses the default bandwidth for basic connectivity, VM and data migration and replication. It will be a great benefit for industry, as it creates a protected high-speed, end-to-end connection with a pay-per-use advantage. SDSOI is an ideal model for low cost, low latency and high speed uninterrupted consistent connections.

6.2 Future Work

Though the SDSOI model includes most of the inbuilt, high and low-level plugins in its core, still there is still room for others. An important plugin that should be added to the transport layer is ‘neutron’, an open-stack networking module that can make the transport layer compatible with open stack.

Third party applications are not necessarily required by all DCs, and can be added and tested as part of the application layer. We simulated only one application from each of the offline and online categories; the remaining applications such as Inventory and Performance Management, Configurations and Fragment Management and Analysis, Monitoring and Alarms can be added to complete the solution design of SDSOI with all the proposed applications. Researchers can also add their own applications in both the online and offline categories.

Due to a lack of real available hardware, we conducted our entire simulation on open-source simulators, controllers and emulators, and did not research the architecture of optical gateways in detail. In the future, researchers can work to replicate the base code of DNS applications and controllers on real live hardware controllers and gateways. We also see great promise for those interested in designing optical gateways as white boxes that are open-flow compatible. These would be similar to the Nokia/Alcatel Lucent ‘Photonic Service Switches (PSS 1830)’.

In addition to the above future work, there are opportunities to reduce restoration time and resource consumption for different protection schemes for path and segment protection. Researchers can also research how to maximize the overall received throughput and add more features in the next generation OTN SDN.

References

[1] X. Chen and Y. Zhang "Intelligence on Optical Transport SDN" International Journal of Computer and Communication Engineering, Vol. 4, No. 1, January 2015.

[2] D. King, et al. "The Software Defined Transport Network: Fundamentals, Findings and Futures", DOI 978-1-5090-1467-5/16, ICTON, Trento, Italy 2016.

[3] S. Yamashita et al. "Extension of OpenFlow Protocol to Support Optical Transport Network, and Its Implementation", IEEE, ISBN: 978-1-4799-8928-7/15, CSCN, Tokyo, Japan 2015.

[4] A. Gupta, et al. "Mesa: A geo-replicated online data warehouse for Google's advertising system", 59(7):117–125, Communications of the ACM, June, Hangzhou, China 2016.

[5] L. Gkatzikis, et al. "Bandwidth Calendaring: Dynamic Services Scheduling over Software Defined Networks", DOI: 10.1109/ICC.2016.7510888, IEEE ICC, Kuala Lumpur, Malaysia, 2016.

[6] X. Zhao, et al. "The prospect of inter-data-center optical networks" IEEE Communications Magazine, 51(9):32–38, 2013.

[7] Alcatel Lucent: white paper "Data Center Interconnect Market trends and Requirements", PTR2014107802EN (October) , Copyright © 2014 Alcatel-Lucent.

[8] X. Jin, et al. "Optimizing Bulk Transfers with Software-Defined Optical WAN" SIGCOMM, 978-1-4503-4193-6, 10.1145/2934872.2934904, Florianopolis, Brazil, August 2016.

[9] M. Noormohammadpour "DC Cast: Efficient Point to Multipoint Transfers across Datacenters" 9th Conference USENIX Workshop on Hot Topics in Cloud Computing, arXiv: 1707.02096v1, Santa Clara, CA, Jul 2017.

[10] Z. Ye, et al. "Survivable virtual infrastructure mapping over transport software-defined networks (T-SDN)" in Proc. OFC, San Francisco, CA, USA, 2014.

[11] S. Chandna, N. Naas, H. Mouftah, "Software Defined Survivable Optical Interconnects for Data Centers", ICTON, Girona, Catalonia, July 2017.

[12] Light Wave: white paper "Editorial Guide: Data Center Interconnects Trends" copyright 2016 by Pennwell Corporation.

[13] D. Karthi; G. Das "WMRD net: An Optical Data Center Interconnect" OFC/NFOEC, INSPEC No. 13582347,Anaheim, CA, USA, June 2013.

[14] D. Kreutz, et al "Software-Defined Networking: A Comprehensive Survey" Proceedings of IEEE, DOI: 10.1109/JPROC.2014.2371999, December 2014.

[15] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," IEEE Commun. Surv. & Tut., vol. 16, no. 4, pp. 2181–2206, 4th Qu. 2014.

[16] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: A survey," IEEE Commun. Surv. & Tut., vol. 16, no. 1, pp. 493–512, 2014.

[17] R. Alvizu and G. Maier, "Can open flow make transport networks smarter and dynamic? an overview on transport SDN," in Proc. IEEE Int. Conf. on SaCoNeT,Vilanova i la Geltru, Spain Jun. 2014.

[18] A. Thyagaturu, et al. "Software Defined Optical Networks (SDONs): A Comprehensive Survey" arXiv: 1511.04376v3 [cs.NI] July 2016.

[19] P. Bhaumik, S. Zhang, P. Chowdhury, S.-S. Lee, J. H. Lee, and B. Mukherjee, "Software-defined optical networks (SDONs): A survey," Photonic Netw. Commun., vol. 28, no. 1, pp. 4–18, 2014.

[20] H.Yang, Y. Cui and J. Zhang "Unified Multi-Layer among Software Defined Multi-Domain Optical Networks (Invited)" Electronics 2015, 4, 329-338; doi:10.3390/electronics4020329, Basel, Switzerland., June 2015

[21] L. Zhang, Z. Zhu "Spectrum-efficient anycast in elastic optical inter-DC networks" Optical switching and Networking 14 (2014) 250-259, Honolulu, USA, 2014.

[22] M. Channegowda, R. Nejabati, and D. Simeonidou "Software-Defined Optical Networks Technology and Infrastructure: Enabling Software-Defined Optical Network Operations [Invited]" A274 J. OPT. COMMUN. NETW. /VOL. 5, NO. 10/OCTOBER 2013.

[23]S. Gringeri, N. Bitar, and T. J. Xia, "Extending software defined network principles to include optical transport," IEEE Communications Magazine, vol. 51, pp. 32-40, Mar. 2013.

[24] M. Channegowda, et al. "Experimental demonstration of an OpenFlow based software-defined optical network employing packet, fixed and flexible DWDM grid technologies on an international multidomain testbed," Optic Express, vol. 21, pp. 5487-5498, Mar. 2013.

[25] S. J. Hong, J. P. Jue, and Q. Zhang, "Effective virtual optical network embedding based on topology aggregation in multi-domain optical networks," presented at the Optical Fiber Communication Conference, San Francisco, California, March 2014.

[26] Z. Ye, A. N. Patel, and P. N. Ji, "Survivable virtual infrastructure mapping over transport software-defined networks (T-SDN)," presented at the Optical Fiber Communication Conference, San Francisco, California, March 2014.

[27] J Blendin et al "Enabling Efficient Multi-Layer Repair in Elastic Optical Networks by Gradually Superimposing SDN" CNSM, 10.1109/CNSM.2016.7818407, Montreal, QC, Canada January 2017.

[28] S. Jain et al. "B4: Experience with a Globally-Deployed Software Defined WAN" SIGCOMM'13, August 12–16, 2013.

[29] Z. Kerravala, white paper "Cisco Intelligent WAN Is the Foundation for the SoftwareDefined WAN", ZK Research, September 2015.

[30] Fujitsu "SDN and the Future of Service Provider Networks" whitepaper © Copyright 2013 Fujitsu Network Communications Inc, 2013.

[31] Y. Takita "Towards seamless service migration in network re-optimization for optically interconnected datacenters", Optical switching and networking, Elsevier, March 2016.

[32] Nokia “Carrier SDN-enabled dynamic enterprise services” whitepaper © Copyright 2016 web, (dated 31 May2017).

[33] A. Sadasivarao, “Bursting Data Between Data Centers: Case for Transport SDN” 2013 IEEE 21st Annual Symposium on High-Performance Interconnects, San Jose, CA, USA, August 2013.

[34] K. Christodoulopoulos et al “Accelerating HPC Workloads with Dynamic Adaptation of a Software-Defined Hybrid Electronic/Optical Interconnect” ©2014 Optical Society of America, San Francisco, CA, USA, March 2014.

[35] H. Yang, et al. “Performance evaluation of Multi-stratum resource integrated resillence for software defined inter-data center interconnect” J. OPTICS EXPRESS 13384, 5(10), A240–A248 (2013), 2013, Vol. 23, No. 10, DOI:10.1364/OE.23.013384, May 2015.

[36] H. Yang, et al. “Multi-stratum resource integration for OpenFlow-based data center interconnect [Invited],” J. Opt. Commun. Netw. 5(10), A240–A248 (2013), 2013

[37] H. Yang, et al. “SUDOI: software defined networking for ubiquitous data center optical interconnection” IEEE communication magazine, 15775280, 10.1109/MCOM.2016.7402266, Feb 2016.

[38] H. Yang, et al. “Performance evaluation of time-aware enhanced software defined networking (TeSDN) for elastic data center optical interconnection” Vol.22, No.15, DOI:10.1364/OE.22.017630 | OPTICS EXPRESS 17630, July 2014.

[39] H. Yang, et al “Experimental demonstration of time aware software defined networking for open flow bases intra-datacenter optical interconnects" Optical Fiber Technology, Elsevier, http://dx.doi.org/10.1016/j.yofte.2013.12.003, Atlanta, GA, USA, December 2013.

[40] H. Yang, et al. “Time-aware Software Defined Networking for OpenFlow-based Datacenter Optical Networks" Network Protocols and Algorithm 10.5296/npa.v6i4.5922, November 2014.

[41] H. Yang, et al “Experimental demonstration of Latency aware software defined networking for open flow bases intra-datacenter optical interconnects" ICT 2013, 10.1049/ic.2013.0199,Beijing, China, August 2013.

[42] V. Hutcheon “OTN to Enable Flexible Networks” Optical society of America, OSA/OFC/NFOEC, Los Angeles, CA, USA, 2011.

[43] F. Slyne, “Design and experimental test of 1:1 End-to-End Protection for LR-PON using an SDN multi-tier Control Plane”, ECOC, Cannes, France,, 2014.

[44] S. S. Savas, et al. "Disaster-Resilient Control Plane Design and Mapping in Software-Defined Networks” IEEE 16th International Conference on High Performance Switching and Routing, Budapest, Hungary, 2015.

[45] Z. Ye, et al. “Survivable Virtual Infrastructure Mapping With Dedicated Protection in Transport Software-Defined Networks”, J. OPT. COMMUN. NETW, VOL. 7, NO. 2, February 2015.

[46] J Blendin, et al. "Enabling Efficient Multi-Layer Repair in Elastic Optical Networks by Gradually Superimposing SDN" CNSM (international conference), 10.1109/CNSM.2016.7818407, Montreal, QC, Canada January 2017.

[47] Kitsuwan, et al. “A novel Protection Design for OpenFlow-based Networks”, Graz, Austria, ICTON, 2014.

[48] C.Janz et al. "Emerging Transport SDN Architecture and Use Cases" IEEE Communications Magazine, DOI 0163-6804/16, October 2016.

[49] T. Tsuritani and X. Cao "Transport SDN for Integrating Multi-optical Technology Networks” OFC, DOI: 978-1-943580-07-1/16, Anaheim, CA, USA, March 2016.

[50] Open Network Foundation "OpenFlow-enabled Transport SDN" white paper May 2014.

[51] Open Network Foundation "Optical Transport Protocol Extension" white paper March 2015.

[52] Open Network Foundation "OpenFlow-enabled Transport SDN" white paper May 2014.

[53] Open Network Foundation "Optical Transport Protocol Extension" white paper March 2015.

[54] Fujitsu Virtuora WDM Control Suite "Software-Defined Applications that Automate and Control Layer 0/1 WDM Services" 2015.

[55] Light wave "Ciena makes WAN SDN play with Agility software portfolio" webpage, July 2014, dated 3 June 2017.

[56] Fujitsu "SDN and the Future of Service Provider" copyright Fujitsu Network Telecommunication, white paper, dated 3 June 2017.

[57] Y. Han et al. "An intent-based network virtualization platform for SDN" Network and Service Management (CNSM), 2016, DOI: 10.1109/CNSM.2016.7818446, Montreal, QC, Canada, January 2017.

[58]M. Pham, D.B. Hoang "SDN applications - The intent-based Northbound Interface realisation for extended applications" Net Soft Conference and Workshops (Net Soft), 2016 IEEE, DOI: 10.1109/NETSOFT.2016.7502469, Seoul, South Korea, July 2016.

[59]ONOS wiki [Online] Available: "https://wiki.onosproject.org/display/ONOS/Intent Framework" [Accessed 10 June 2017].

[60]ONOS wiki [Online] Available: "https://wiki.onosproject.org/display/ONOS/ Developer+Quick+Start" [Accessed 10 June 2017].

[61]ONOS wiki [Online] Available: "https://wiki.onosproject.org/display/ONOS/ System+Component" [Accessed 10 June 2017].

[62]ONOS wiki [Online] Available: "https://wiki.onosproject.org/display/ONOS/ Web+UI+Architecture" [Accessed 10 June 2017].

[63] F. Keti, S. Askar "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments" Intelligent Systems, Modelling and Simulation (ISMS), 2015, DOI: 10.1109/ISMS.2015.46, Kuala Lumpur, Malaysia, October 2015.

[64] R. Barrett et al " Dynamic Traffic Diversion in SDN: testbed vs Mininet" Computing, Networking and Communications (ICNC), 2017, DOI: 10.1109/ICCNC.2017.7876121, Santa Clara, CA, USA, March 2017.

[65] Apache Maven Project [Online]. Available: "https://maven.apache.org/archetype/maven-archetype-plugin/examples/create-multi-module-project.html" version 3.0.1, last published 08 april 2017, [Accessed: 11 June 2017].

[66] Buck wiki [Online]. Available: "https://buckbuild.com/" © Copyright Facebook, 2013, [Accessed: 11 June 2017].

[67] W. Zhou, L. Li , W. Chou " SDN Northbound REST API with Efficient Caches" web services (ICWS), 2014, DOI: 10.1109/ICWS.2014.46, Anchorage, AK, USA July 2014.

[68] W. Zhou, L. Li, W. Chou "REST API Design Patterns for SDN Northbound API" Advanced Information Networking and Applications Workshops (WAINA), 2014, DOI: 10.1109/WAINA.2014.153, Victoria, BC, Canada ,May 2014.

[69] Ganglia Monitoring System [Online]. Available: "http://ganglia.sourceforge.net/" [Accessed: 11 June 2017].

[70] Collectd–The system statistics collection daemon [Online]. Available: "https://collectd.org/" [Accessed: 11 June 2017].

[71] Wireshark wiki [Online] Available: "https://www.wireshark.org/tools/" [Accessed 10 June 2017].

[72]K shortest path, GitHub [Online] Available: "https://github.com/opennetworkinglab/onos/blob/master/core/api/src/main/java/org/onosproject/net/topology/PathService.java" [Accessed 10 June 2017].

[73]K shortest path, [Online] Available: "http://api.onosproject.org/1.5.0/org/onlab/graph/class-use/GraphPathSearch.html" [Accessed 10 June 2017].

[74] C. Xie, et al. "Multi-Point Ethernet over Next-Generation SONET/SDH" , IEEE ICC, DOI**:** 10.1109/ICC.2009.5199207, Dresden, Germany, June 2009.

[75] A. Hamad et al "SONET over Ethernet" Annual Global Online Conference on Information and Computer Technology. DOI 10.1109/GOCICT.2015.24, Louisville, KY, USA , 2015.

[76] Series G: Transmission systems and media, Digital Systems and Networks "G.709 Synchronous Multiplexing Structure" [Accessed 10 June 2017].

[77] Timothy P. Walker "Optical Transport Network (OTN) Tutorial" ITU, [Accessed 10 June 2017].

[78] Series G: Transmission systems and media, Digital Systems and Networks "G.872 Architecture of Optical transport networks" ITU-T, [Accessed 10 June 2017].

[79] Series G: Transmission systems and media, Digital Systems and Networks "G.805 Generic functional architecture of transport networks" ITU-T, [Accessed 10 June 2017].

Appendix A: SONET/SDH

The SONET (Synchronous optical network) and SDH (Synchronous Digital Hierarchy) [74] are optical standard protocols developed for synchronous transfer of multiple digital bit streams over optical fibres using lasers or coherent light from LEDs (light-emitting diodes). SONET was majorly used in North America whereas SDH was matured in Europe and the rest of the world. This standard was developed for the telephone networks to provide long-term solution. Bell Core initially proposed it. They defined the rates, format, physical layer, network element architectural features as well as network operational criteria. They gained widespread popularity in very early phase and were accepted as the data communication model. They allow efficient bandwidth usage and protection and therefore became perfect fit for the physical layer of the OSI data network model.

SONET over Ethernet was developed as a transmission technology and it is used to carry SONET frames directly on the Ethernet. This technology is fully compatible with the Ethernet standard IEEE 802.3 and transmits the packets as per SONET headers instead of learning MAC addresses of the Source and Destination. Every Ethernet frame that has to be transmitted is inserted into the SONET payload, and from there they are further encapsulated into SONET's data frame and are forwarded on the SONET segments [75]. For every transmission in this technology, frames get logically divided into two subparts: SONET and Ethernet. SONET part handles SONET switching and the Ethernet part is processed by the SONET over the Ethernet side. Operators started using this technology for a while and came up with the conclusion of complicated and expensive mapping scheme. To overcome these limitations, optical standards were evolved further as WDM and then to OTN. OTN is an ITU-T standard; and it provides better bit rate efficiency, resiliency and management with huge capacity and high bandwidths.

Appendix B Optical Transport Network

B.1 Optical Transport Network

Over the past few years, optical transport architectures have migrated from SONET/SDH networks to WDM architectures. WDM systems allow carriers to tap into enormous capacity of data by carrying multiple wavelengths over a single fiber. WDM brings a lot of cost saving for carriers, as compared to SONET because the cost of deploying single-channel SONET networks or overlay networks for each service type or technology is relatively on higher side. With early WDM platforms, many of the SONET capabilities that we took for granted in optical standards, such as performance monitoring, fault detection and isolation, standard multiplexing scheme, and standard communication channels, were either missing or implemented in a proprietary manner by each WDM equipment vendor. To overcome these feature limitations of WDM as well as to cater to the increasing demand of high bandwidth support, Optical Transport Network (OTN) was developed by ITU-T G.709 [76].

This OTN standard is designed to add the same type of performance monitoring, fault detection, communication channels, and multiplexing hierarchy for WDM wavelengths as those, which existed for single channel SONET signals. The four primary goals of OTN are to:

- Provide a universal container to support any kind of traffic type.
- Provide enhanced OAM for wavelengths.
- Provide a standard multiplexing hierarchy.
- Enable end-to-end optical transport transparency of customer traffic.

Optical networks primary functionalities are to provide support for transportation, multiplexing, routing, supervision and survivability of client signals, which are processed predominantly in the photonic domain. OTN (similar to SONET and SDH) performs all of these functionalities; however, it has more of the management features imbibed in it as compared to overheads.

The OTN has evolved with more efficient convergence of the SONET/SDH, new data services, reduced TDM transport complexity and related cost. It is designed for the effective mapping of different protocols and rates into the same uplink pipe to provide a high bandwidth and lower cost.

It enables greater signal transmission to longer distances and less regeneration rates because of its advantages such as Forward Error correction (FEC).

The OTN incorporates optimized overhead for the efficient mapping of various protocols like LAN, Ethernet, SONET/SDH, Video and SAN signals over carrier WDM networks. It does this by wrapping each of these protocols in a completely transparent client signal over flexible containers with the optimized usage of optical wavelength resources in the network as shown in Figure B.1.

B.2 Tandem Connection Monitoring

SONET/SDH divides their monitoring into section, line and path monitoring. However, there occurred a problem when SONET tried to send the signal through Carrier's Carrier where they were supposed to monitor the segment of the path that has passed in another carrier's network. This process of monitoring the section of the path, which has another vendors segment in them, is called as Tandem Connection Monitoring (TCM) [77]. TCM exists as a layer between Line Monitoring and Path Monitoring. G.709 [76] defines six TCM standards for the Optical Transport Network. These standards are defined as follows: TCM1 is used to monitor the Quality of Service (QoS) in general whereas TCM2 is used by the first operator to monitor the end-to-end QoS. TCM3 is used to monitor the domains for Intra domain monitoring and TCM 4 is used by operator B for protection monitoring of the signal. TCM5 and TCM6 are OEM defined standards and OEM's informs the service providers that they are keeping this particular TCM for monitoring purposes.

TCM also supports the monitoring of ODUk connections for the following network applications:

- Optical channel tandem connection monitoring (e.g., fault localization or verification of delivered quality of service).
- Optical UNI to UNI tandem connection monitoring.
- Optical NNI to NNI tandem connection monitoring.
- Sublayer monitoring for linear 1+1, 1:1 and 1:N optical channel subnetwork connection protection switching for determining the signal fail and signal degrade conditions.

- Optical channel tandem connection monitoring for the purpose of detecting a signal failure or signal degrade condition, to initiate automatic restoration of the connection during fault and error conditions in the network.

Our research mainly uses the last two application of TCM to develop a programmable and automated SDN application for the optical interconnections of DCs.

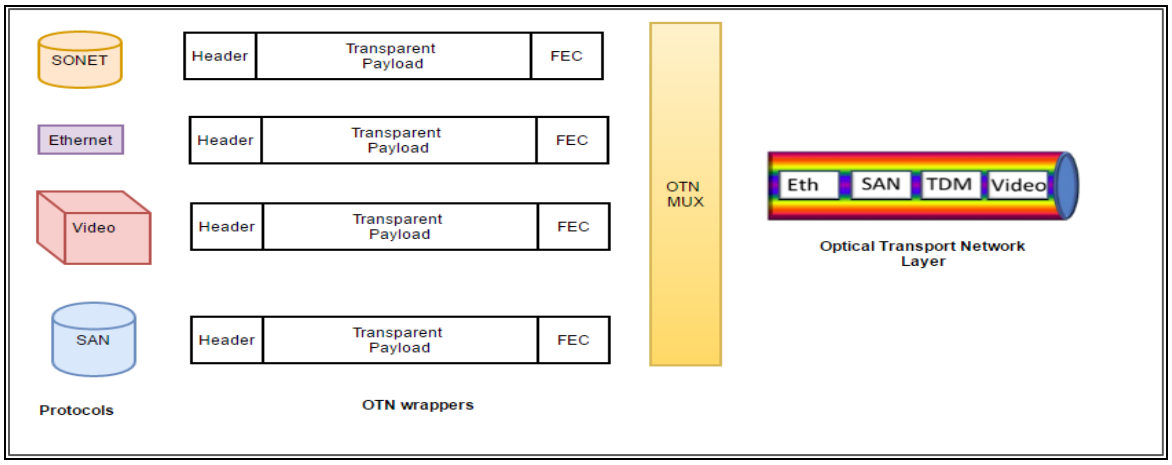


Figure B.1 OTN wrapper consisting of different kind of services.

B.3 OTN Hierarchy

OTN defines an Optical Transport Hierarchy (OTH) for multi-wavelength optical networks [77]. The OTH is recently developed as a transport technology by ITU G.872 [78]. Various other standards fall under the "OTN" Umbrella. The primary standard under the OTN umbrella is G.709 i.e. Interfaces for the OTN. Other standards include G.798 (characteristics for OTN), G.8251 (Control of Jitter OTN) and G.872 that defines the OTN architecture, which comprises of Optical Channel (OCh), Optical Multiplex Section (OMS), and Optical Transmission Section (OTS). The motivation of making optical transport architecture as three-layer architecture is described below:

- **Optical channel layer network**

  This layer provides end-to-end networking of optical channels for transparently conveying client information of varying formats. To accomplish this end-to-end networking, OCh possesses capabilities like the optical channel rearrangement for flexible network routing, channel overhead, which ensures the integrity of the optical channel adapted information, and optical channel supervisory functions for enabling network level operations and management functions.

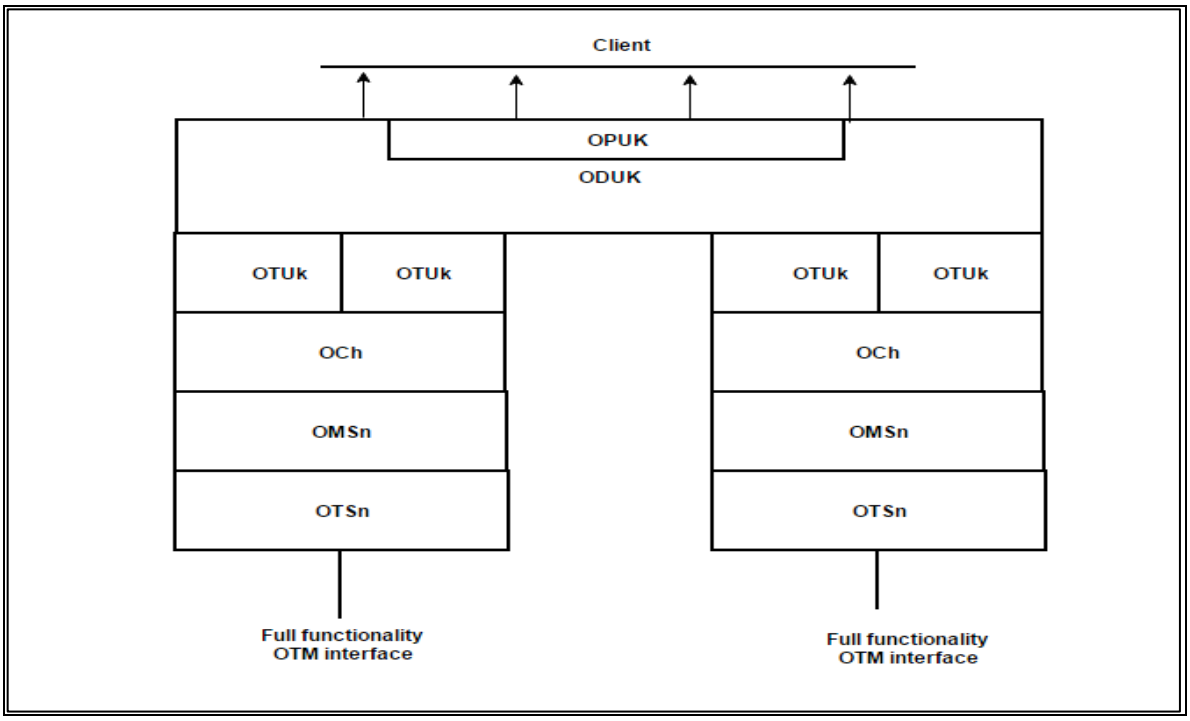


Figure B.2 Optical Transport Network (OTN) Hierarchy.

- **Optical Multiplex Section (OMS)**

This layer is responsible for the networking of multi-wavelength optical signal. It includes the optical multiplex section overhead for ensuring the integrity of the multi-wavelength optical information and optical multiplex section supervisory functions for enabling the section level operations and management such as the survivability of the multiplex section.

- **Optical Transmission Section (OTS)**

This layer deals with the capability of transmitting the optical signal on various single mode or multimode fibres. It includes an optical transmission section supervisory function to enable section level operations and management such as transmission section survivability.

Appendix C Evaluation of Day-Night Scheduling Application in Real World Topology

This appendix describes the test cases for evaluation of DNS application in Google WAN and PAN European topology. We have tested this application among different sets of source and destination gateways and for different values of bandwidths and latency under both proactive and non-proactive protection. To perform this testing, basic input values are user defined as explained in Section 4.4.8.

C.1 Day Night Scheduling DC1 to DC12 with 10G of bandwidth connectivity for Google WAN topology

- Before populating any values in the DNS application form, ensure that DNS Application is installed and activated.
- For this interconnection, we populate the tabs in schedule bandwidth form with Source DC as 0000000000000001 and Destination DC as 000000000000000c, enter the date and time of the desired scheduled interconnect, populate the primary BW as ODU2 and downsized or default bandwidth when schedule expires as ODU1. Please note that we have chosen the primary value as ODU 2 as in this test case we are trying to create the 10G connectivity.
- Once the above step is completed, hit the create tab and the intent will be created.
- This intent will appear automatically in the GUI intents page and the ONOS CLI when the scheduled time is reached
- As shown in Figure C.1 with yellow dotted line, intent is installed in the network and geographically distributed DC1 and DC12 are now interconnected with each other at 10G protected connectivity.

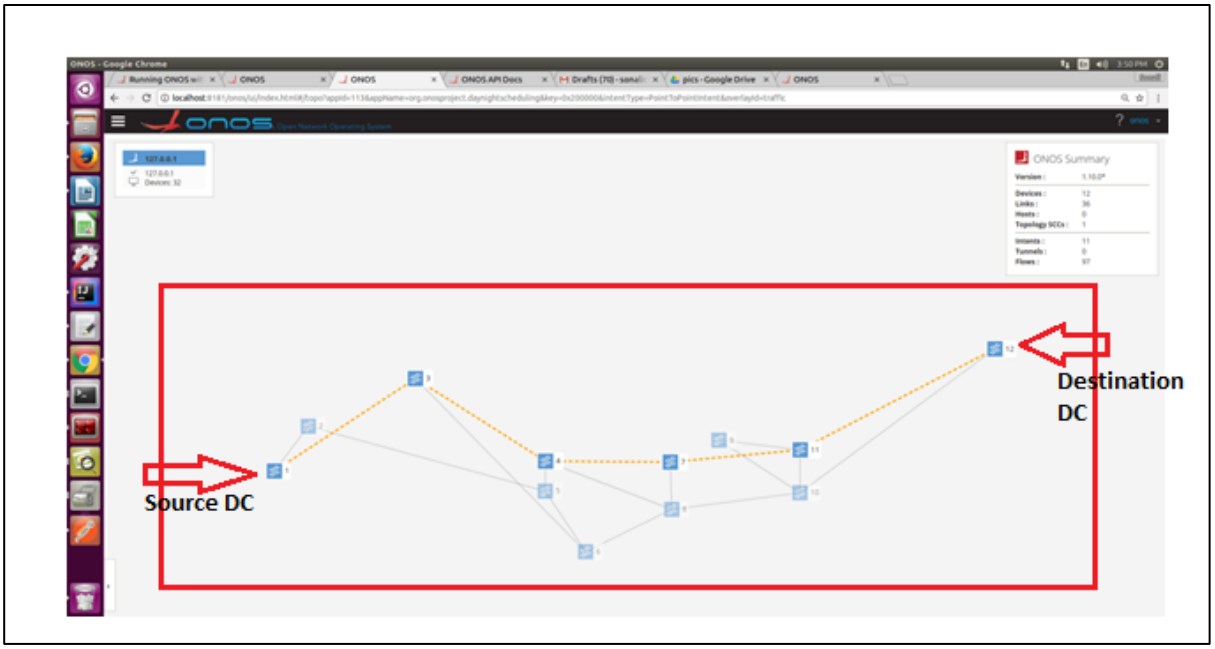


Figure C.1 Optical Interconnect between DC1 and DC12 created by the DNS Application in SDSOI for Google WAN topology.

- To verify the protection schemes built by DNS application for DCI, we performed some manual failures in and observed the protection behaviour as followed.
- In Figure C.1, when we manually failed interconnect between gateway4 and gateway7; we observed that SDSOI controller immediately restored the traffic with the same bandwidth to interconnection between gateways 4, 8, 7. Thus when a particular link failed, only that link got replaced with an alternate sub path as shown in Figure C2.
- With this link protection and restoration scheme, the controller neither had to reroute entire traffic in the network nor it crank backs the traffic on Source node 1 and only the failed link got replaced with the new sub path.
- Link protection saves the overall cost of traffic retransmission and has reduced restoration time as only the failed link got replaced and rest of the path remained same.

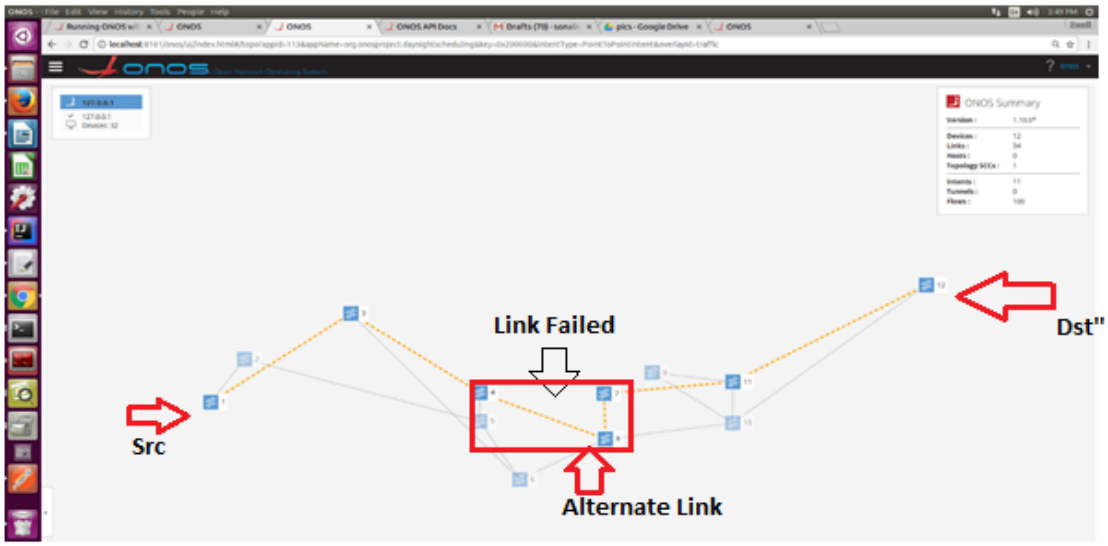


Figure C.2 Link Protection for interconnect between DC4 and DC7 created by the DNS Application in SDSOI for Google WAN topology.

- In Figure C.1, let's assume our optical gateway network is divided into following three segments:

Segment	Optical Gateway No.					
1	1	2				
2	3	4	5	6	7	8
3	9	10	11			

Table C.1 Segments considered in case 1 for Google WAN Topology.

- For segment protection in Segment 2, we manually fail two of interconnects i.e. interconnect 3, 4 and interconnect 4, 7. Failure of these two interconnects makes the entire segment down.

- With DNS segment protection and restoration, SDSOI controller immediately installs the pre-computed new available alternate path in segment 2 via interconnection 3-6-7 instead of the previously installed interconnection 3, 4, 7 and restores the traffic on this new alternate path.

- Thus when more than an interconnect fails in any of the highly sensitive segment of the network, SDSOI immediately computes the new route of interconnects for that segment and lets rest of the connection remains same.
- Segment protection for this case is shown in Figure C.3.
- These segments are actively monitored by the TCM fields of the OTN headers as well as by the segments source gateway

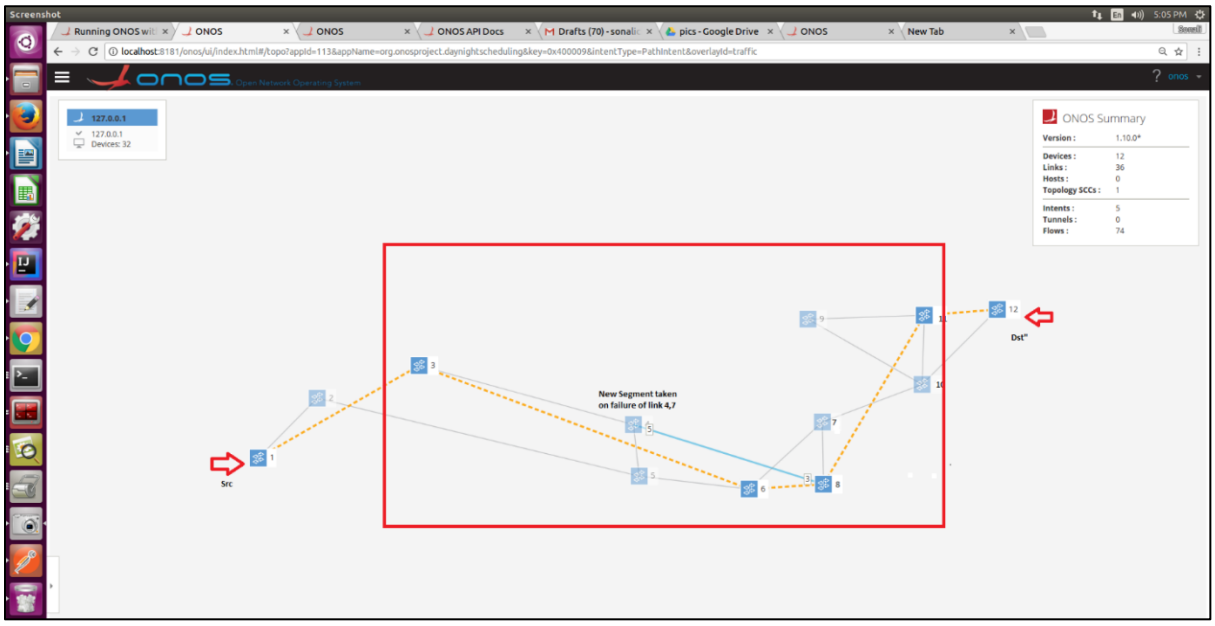


Figure C.3 Segment protection for Segment 2 (DC 3- DC8) created by the DNS Application in SDSOI for Google WAN topology.

- For Figure C.1, when more link fails in the overall path of different segments or segment protection has higher resource utilization value and more restoration time, SDSOI automatically computes the new shortest disjoint protected path with the same bandwidth capacity and constraints.
- Path protection assures that the committed SLA demands of the service providers are maintained when overall transmission cost or restoration time by link and segment protection are higher.
- Figure C.4 depicts the example of path protection and restoration, when interconnections between gateway 4 - gateway 7 and gateway 11 – gateway 12 in

the working path fails. SDSOI immediately installs the new and disjoint protected path via gateway's interconnection 1-3-4-8-7-10-9-11-12.

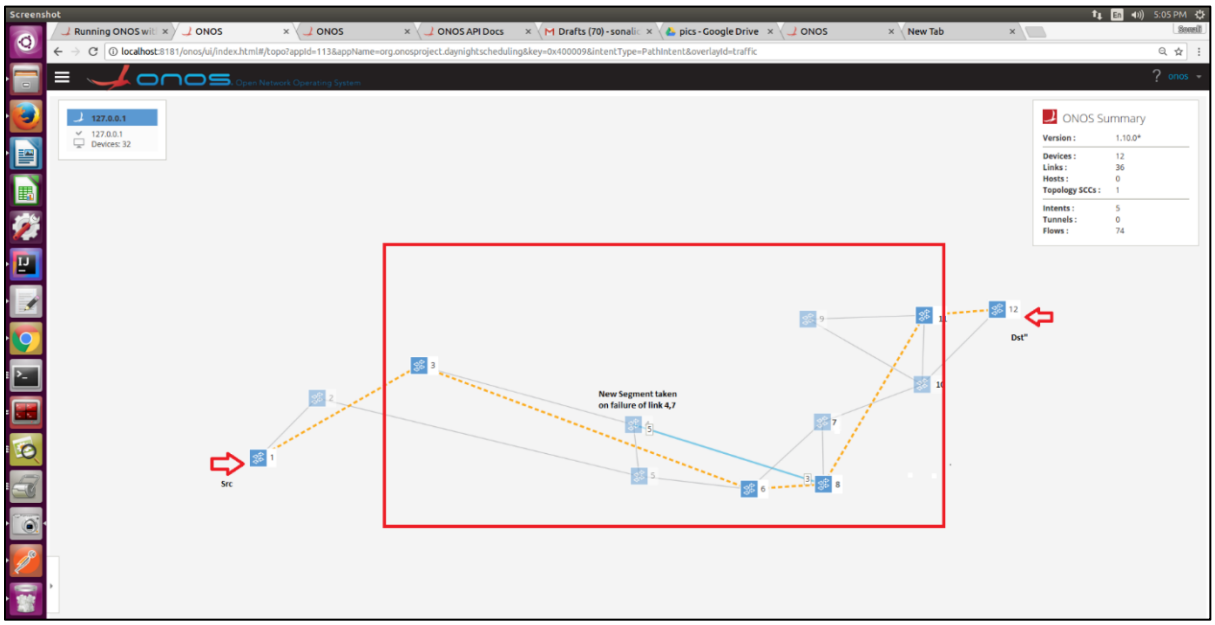


Figure C.4 Path protection for Interconnection between DC 1 and DC 12 created by DNS Application in SDSOI for Google WAN topology.

C.2 Day Night Scheduling between DC23 to DC25 with 100G of bandwidth Connectivity in PAN European topology

- Ensure to install and activate the DNS application either from the application tab of GUI or from CLI
- For this interconnection, populate the tabs in schedule bandwidth form with Source DC as 0000000000000017 and Destination DC as 0000000000000019, enter the scheduled time of the interconnect as well as populate the primary BW as ODU4 (assumed) and the downsized or default bandwidth as ODU2 (assumed)
- Hit the create tab and it will display the message box on the GUI "localhost 8181 says bandwidth scheduled"
- This intent will appear automatically in the GUI intents page and the ONOS CLI when the scheduled time is reached
- As shown in Figure C.5 with yellow dotted line, intent is installed in the network and geographically apart DC23 and DC25 gateways are now interconnected to each

other at 100G of administrator defined bandwidth and latency as per the customer's SLA.

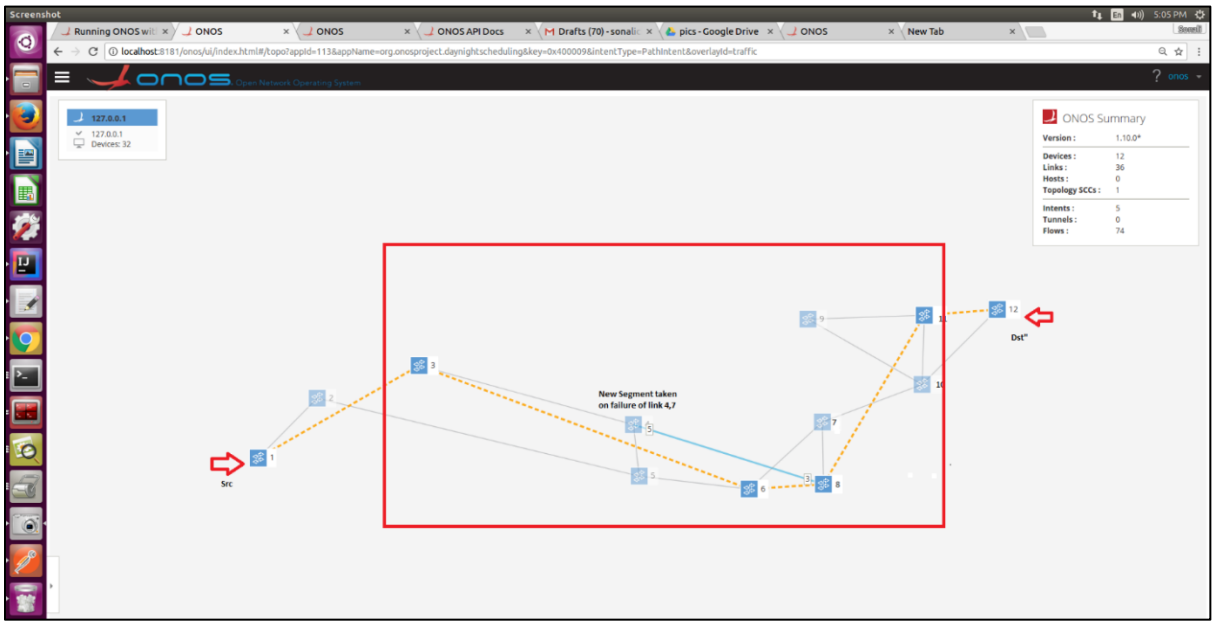


Figure C.5 Optical Interconnect between DC23 and D25 created by DNS Application in SDSOI for PAN European topology.

- In Figure C.5 for the intent, that creates the connection between DC 23 and DC 25 gateways, when we manually fail the interconnect between DC 18 and DC 16 gateways, SDSOI controller automatically creates and installs the new alternate sub path between them.
- In this case, for link protection, the new path is installed between the gateways 18-3-14-2-16 and the traffic failure between gateway 18 and 16 is restored on it.
- In Figure C.5, let's assume our network is divided into following segments as shown in Table C.2.

Segment	Optical Gateways No.								
1	4	5	11	15	32				
2	1	13	16		21	23	28	30	31
3	2	4	12	14	17	20	24		
4	3	6	7	9	18	19	22	27	
5	8	10	19	25	26	29			

Table C.2 Segments considered in case 2 for PAN European Topology.

- For verifying segment protection in this intent, we manually fail two of the interconnects of segment 4, interconnect 6-9 and interconnect 6-18 and to restore the traffic flow in the network on failure, SDSOI controller immediately created the new disjoint alternate segment route as the protection path via gateways 26-7-3-18-16.
- SDSOI controller immediately installs the new flow rules among all the gateways of that segment and updates the flow tables however rest of the connection and flow tables remains same as shown in Figure C.7.
- Similarly, while interconnecting DC 23 and DC 25 in Figure C.5, if more numbers of links fail in the overall path of different or same segment or if segment protection has higher resource utilization value and more restoration time, SDSOI automatically computes the new shortest disjoint protected path with the same bandwidth capacity and constraints.
- The new protected paths are the disjoint paths from the existing working path and are computed using Dijkstra algorithm, which is extended in this thesis.
- The new protection path in this case of interconnection between DC 23 and DC 25 is shown in Figure C.8 below.

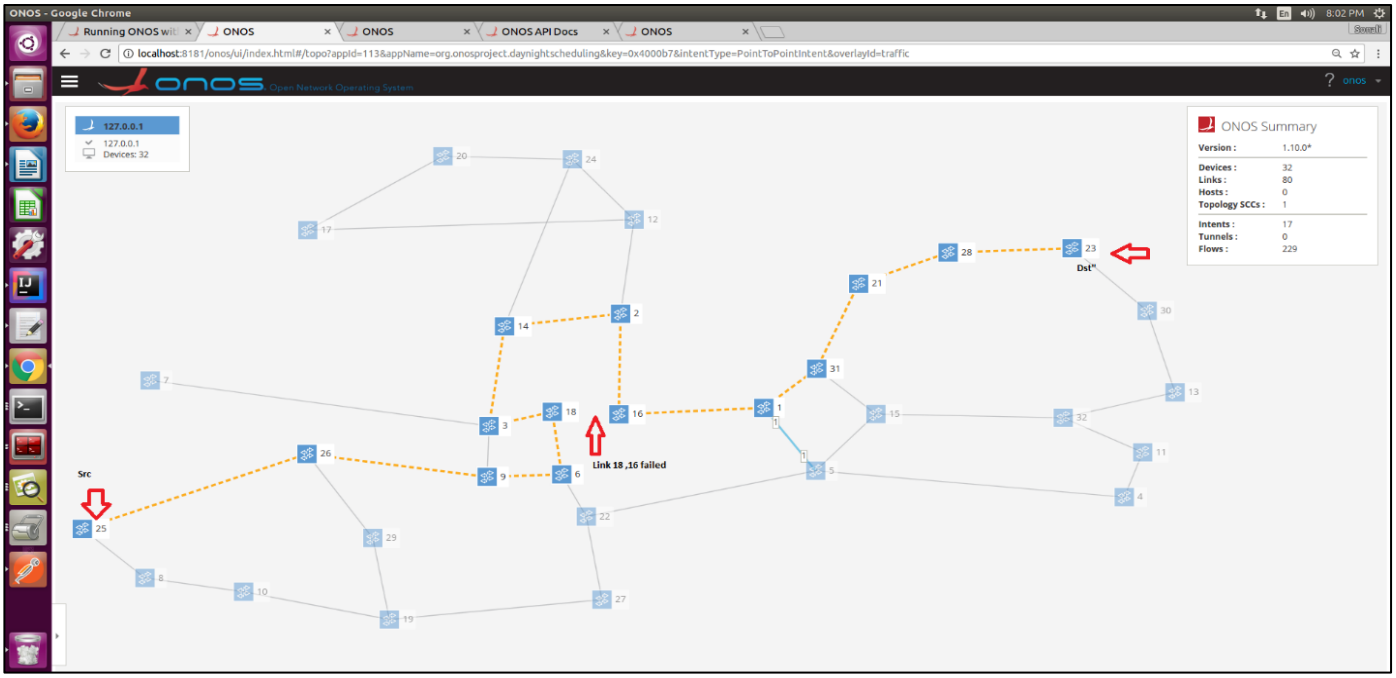


Figure C.6 Link Protection for interconnect between DC23 and DC25 created by the DNS Application in SDSOI for PAN European topology.

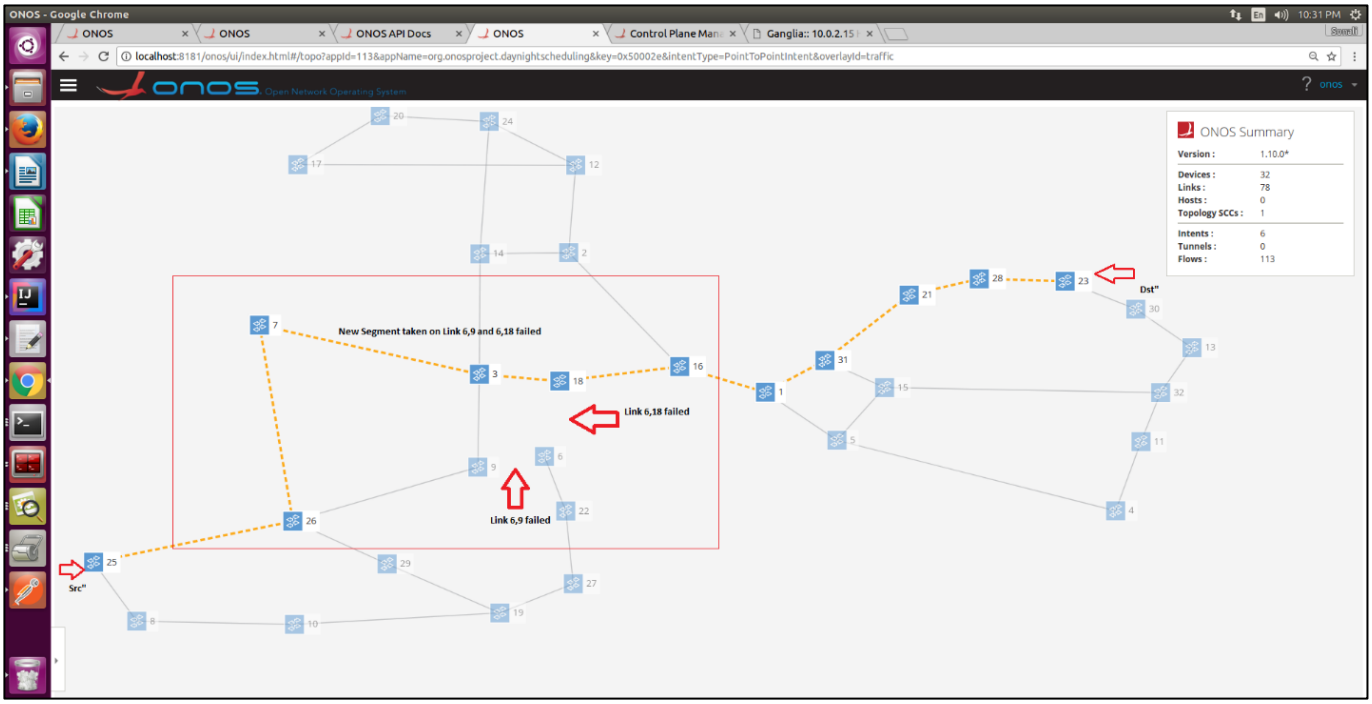


Figure C.7 Segment Protection for segment 4 interconnects between DC23 and DC25 created by the DNS Application in SDSOI for PAN European topology.

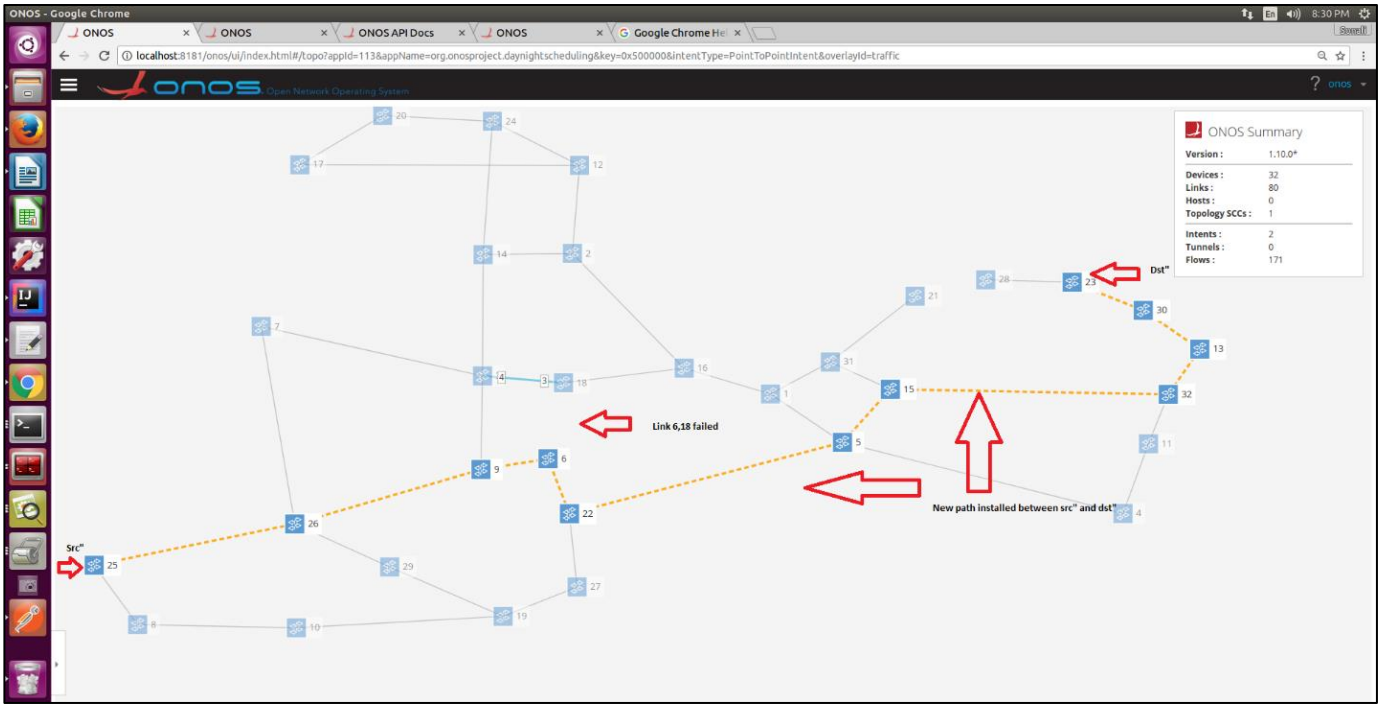


Figure C.8 Protection path created by DNS Intents for connecting DC 23 and 25 in SDSOI for PAN European Topology.

C.3 Day Night Scheduling between DC19 to DC32 with 40G of bandwidth Connectivity in PAN European topology

- Ensure to install and activate the DNS application from the application tab of GUI or from CLI
- For this interconnection example, populate the tabs in schedule bandwidth form with source DC address as 0000000000000013 and destination DC address as 000000000000001f, enter the scheduled time of the interconnect for which intent has to be installed and populate the primary BW field as ODU3 (assumed in this case) and the downsized or default bandwidth as ODU1 (assumed in this case)
- Hit the create tab and it will display the message box on the GUI “localhost 8181 says bandwidth scheduled”
- This intent will appear automatically in the GUI intents page and the ONOS CLI when the scheduled time is reached

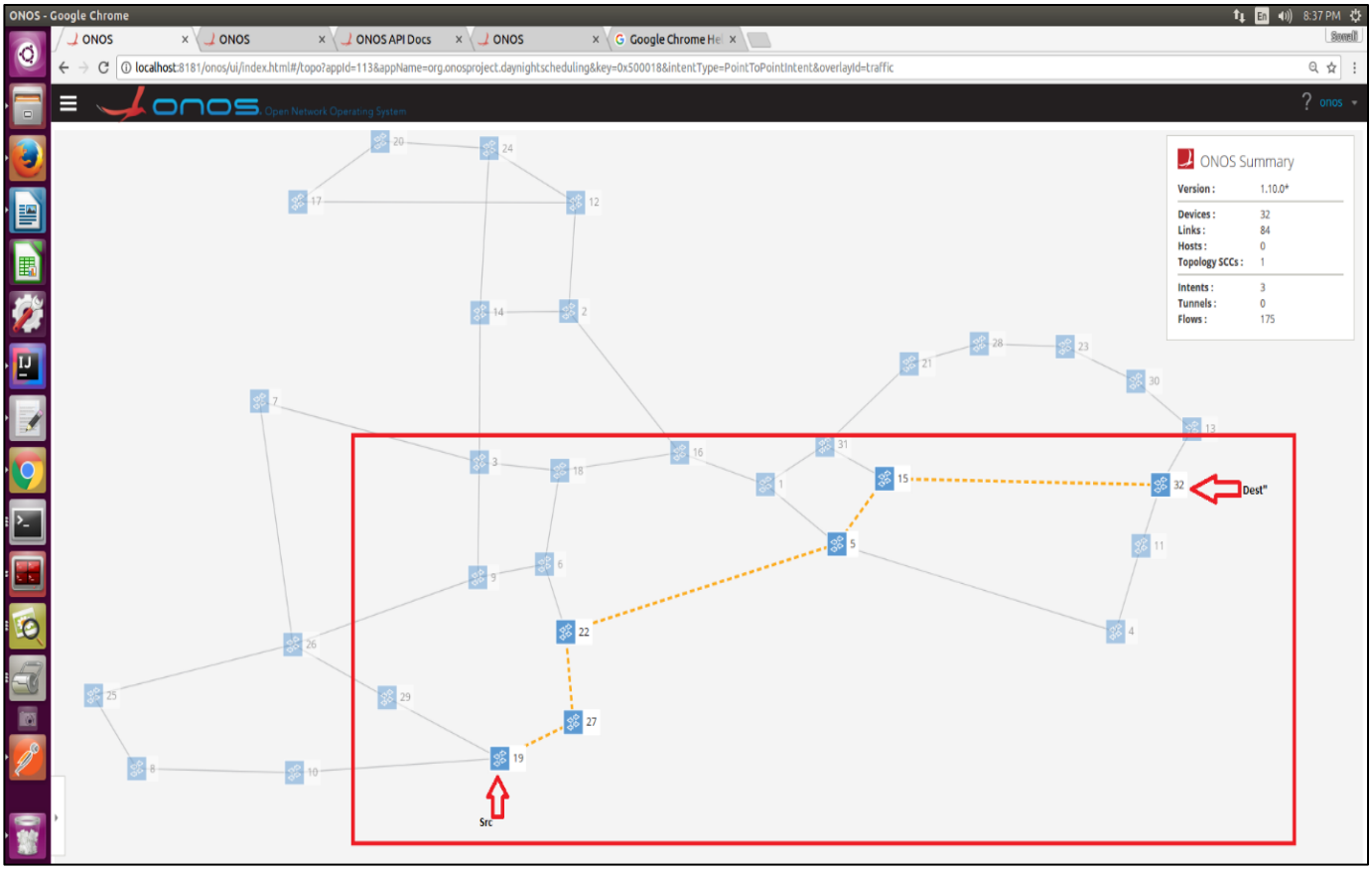


Figure C.9 Optical Interconnect between DC19 and D32 created by the DNS Application in SDSOI for PAN European topology.

- As shown in Figure C.9 with yellow dotted line, intent is installed in the network and DC19 and DC32 are now interconnected with each other at 40G of administrator defined default bandwidth and latency as per the customer's SLA.
- For this intent connection between DC 19 and DC 32, whenever any particular link in this interconnection fails, the SDSOI controller proactively and instantly installs the new low cost, shortest route link between them. This new link or interconnect is the replacement for the existing failed link and rest of the path or route for the interconnection remains the same with minimum cost and restoration time.
- As shown in Figure C.10, when the link connection between DC 15 and DC 32 gateways fails, it takes the new alternate route from gateway 15-31-21-28-23-30-

13-32. This new alternate link consumes higher no of resources in the network and uses more bandwidth or wavelength.

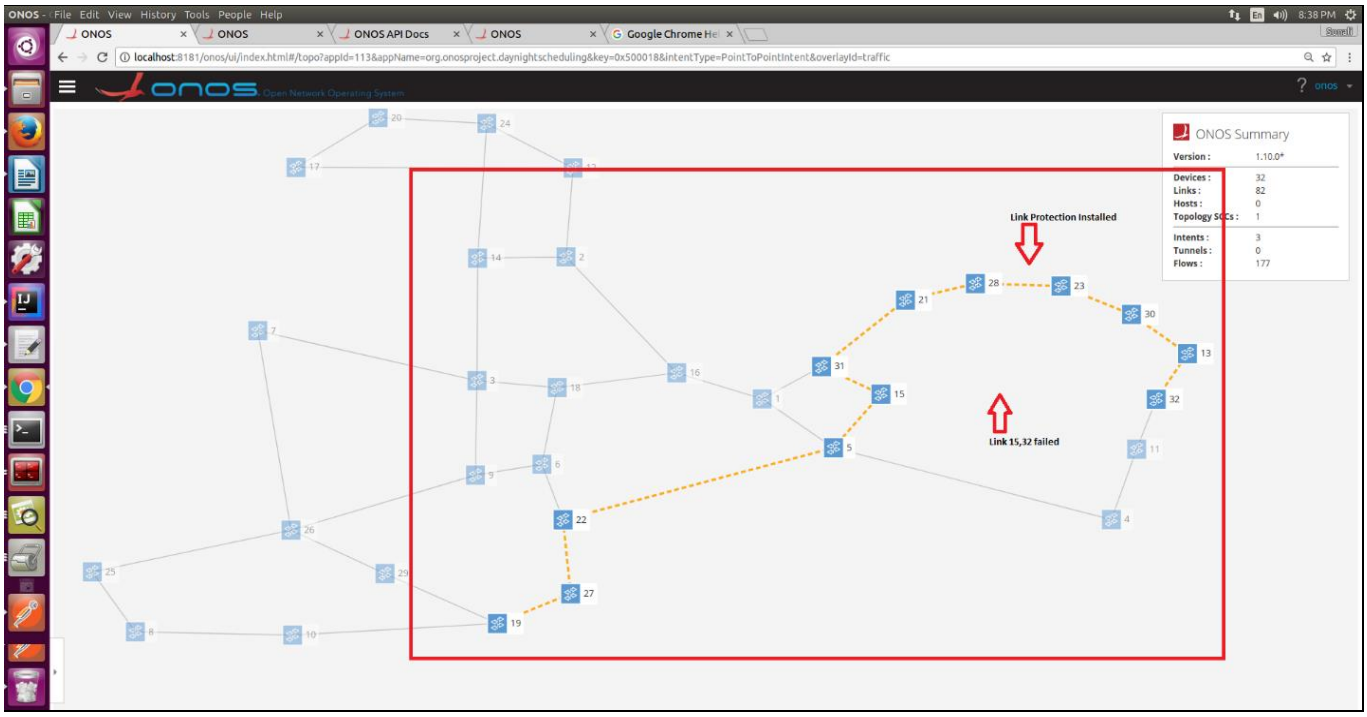


Figure C.10 Link Protection for interconnect between DC 19 and DC 32 created by DNS Application in SDSOI for PAN European topology.

- For segment protection in this example, let's assume our network is divided into the following segments:

Segment	DC / Optical Gateways No								
1	4	11	13	23	28	30	31	32	
2	8	10	19	25	26	29	7	3	9
3	2	4	12	14	17	20	24		
4	1	5	6	15	16	18	19	22	27

Table C.3 Segments considered in case 3 for PAN European Topology.

- To verify the segment protection in this case, lets manually fail two of the interconnects in segment 4, interconnect between gateway 1 and 22 and interconnect between gateways 5 and 15.

- SDSOI controller immediately installs the new protected path for segment 4 routed from gateway 6-18-16-1-31-15. For this new protection path, controller installs the new flow rules, messages to all the gateways in this segment and updates their flow tables accordingly and rest of the connection remain same as shown in Figure C.11.
- Similarly, while interconnecting DC 19 and DC 32, when more than one link or segment fails or if the cost of restoring the traffic on existing working path increases tremendously due to multiple failures, the SDSOI controller proactively generates the new protection path between the DCs. This new protected path has disjoint paths from the existing working path and uses extended Dijkstra Algorithm for computation of this new protection path. The protection path installed by the intent for DC interconnects between DC 19 and DC 32 is shown in Figure C.12 below.

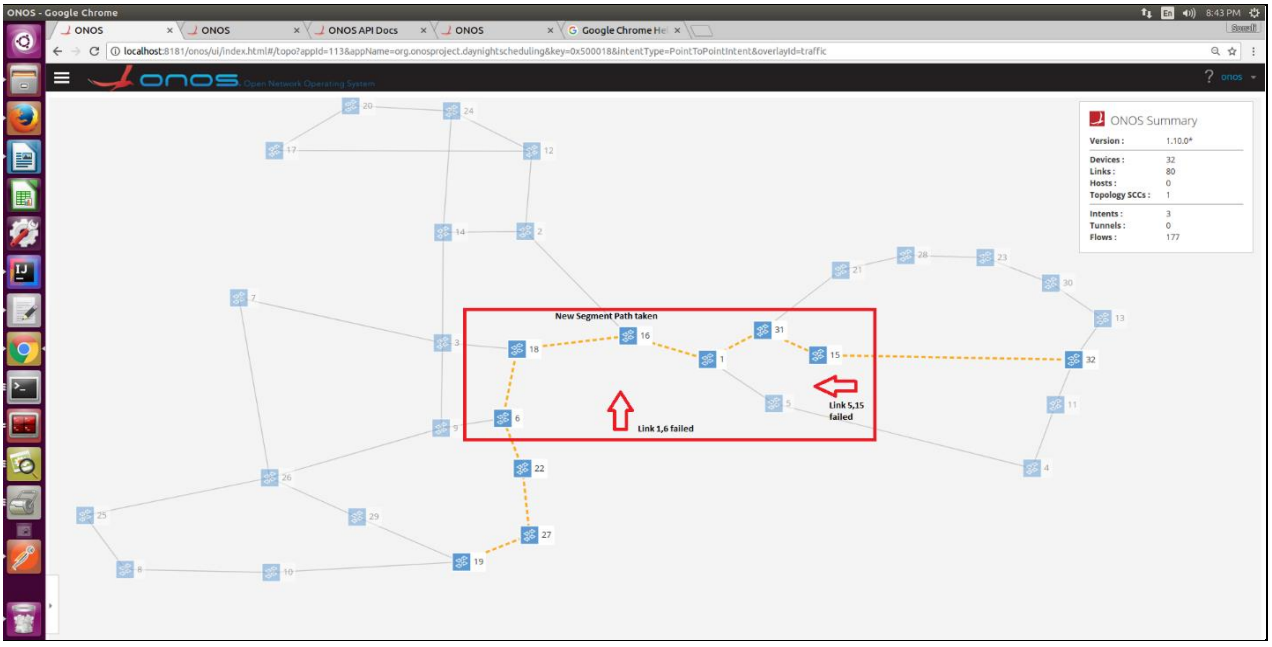


Figure C.11 Segment Protection for segment 4 interconnects between DC19 and DC32 created by the DNS Application in SDSOI for PAN European topology.

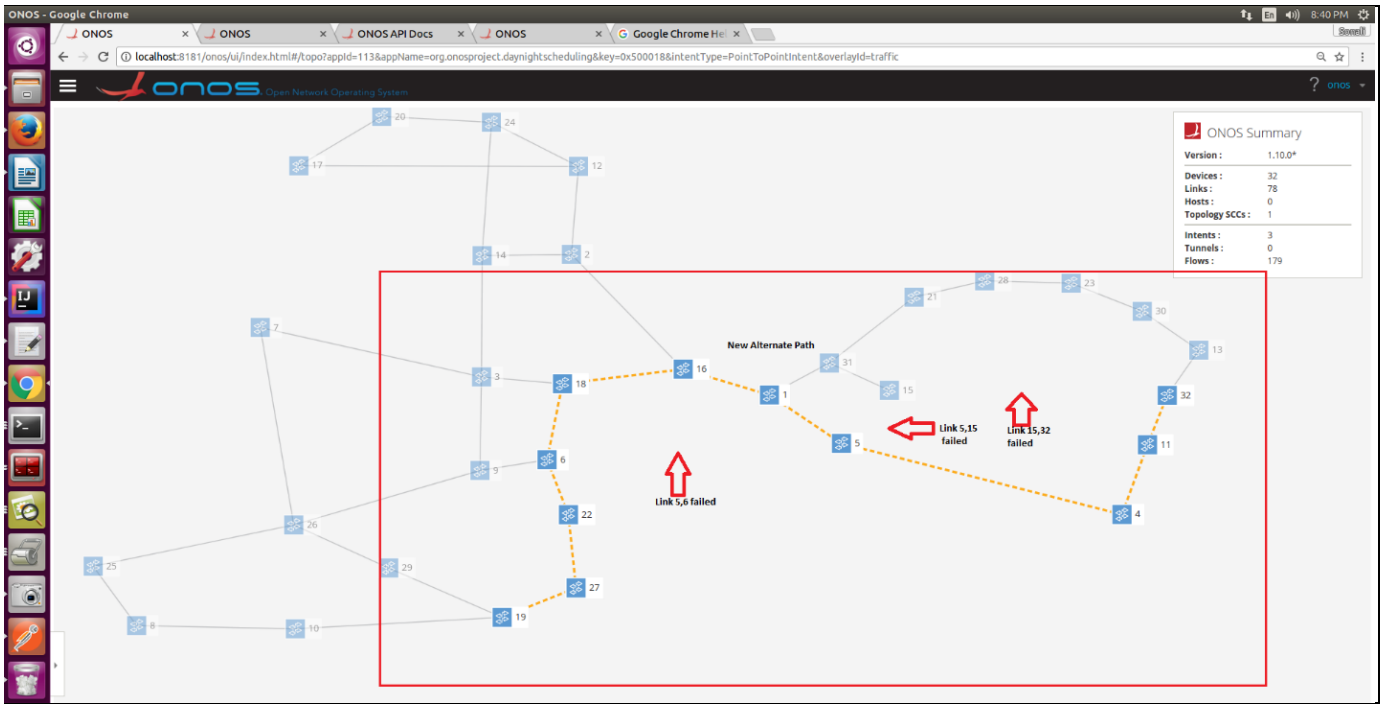


Figure C.12 Protection path created by DNS Intents for connecting DC 19 and 32 in SDSOI for PAN European Topology.

Appendix D Snapshots

The following are the snapshots referred in this thesis.

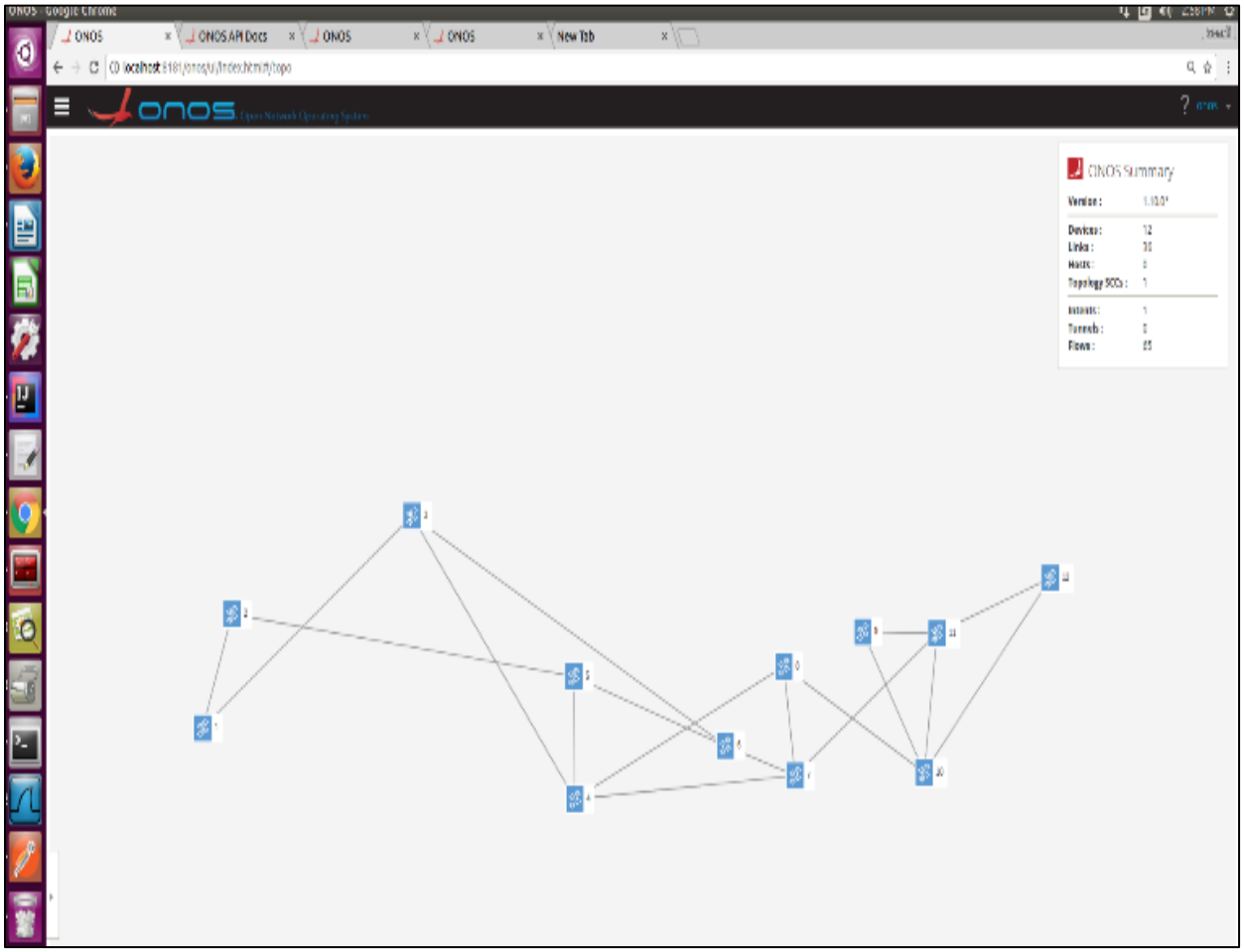


Figure D.1 Google WAN topology on ONOS GUI.

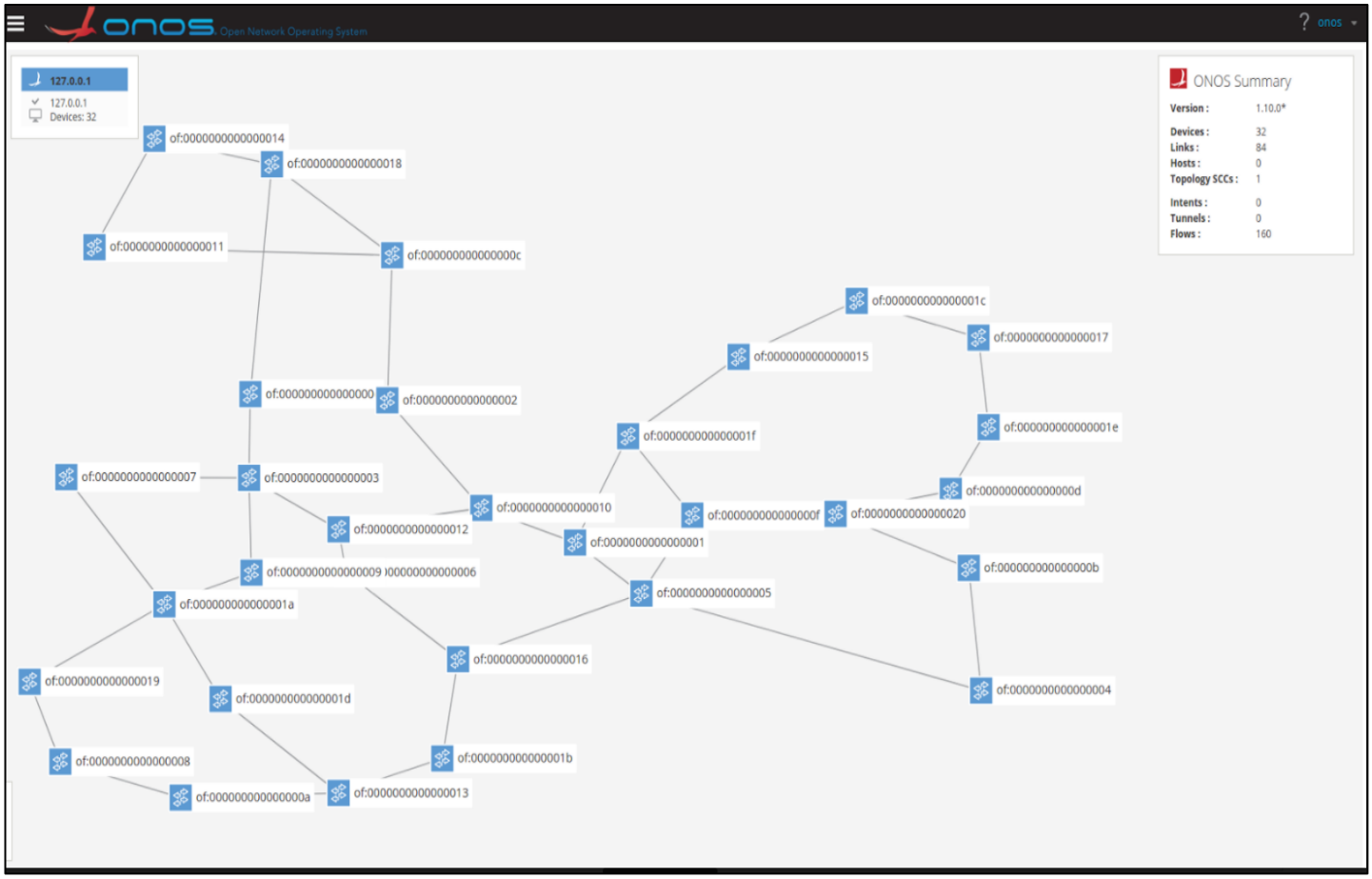


Figure D.2 PAN European topology on ONOS GUI.

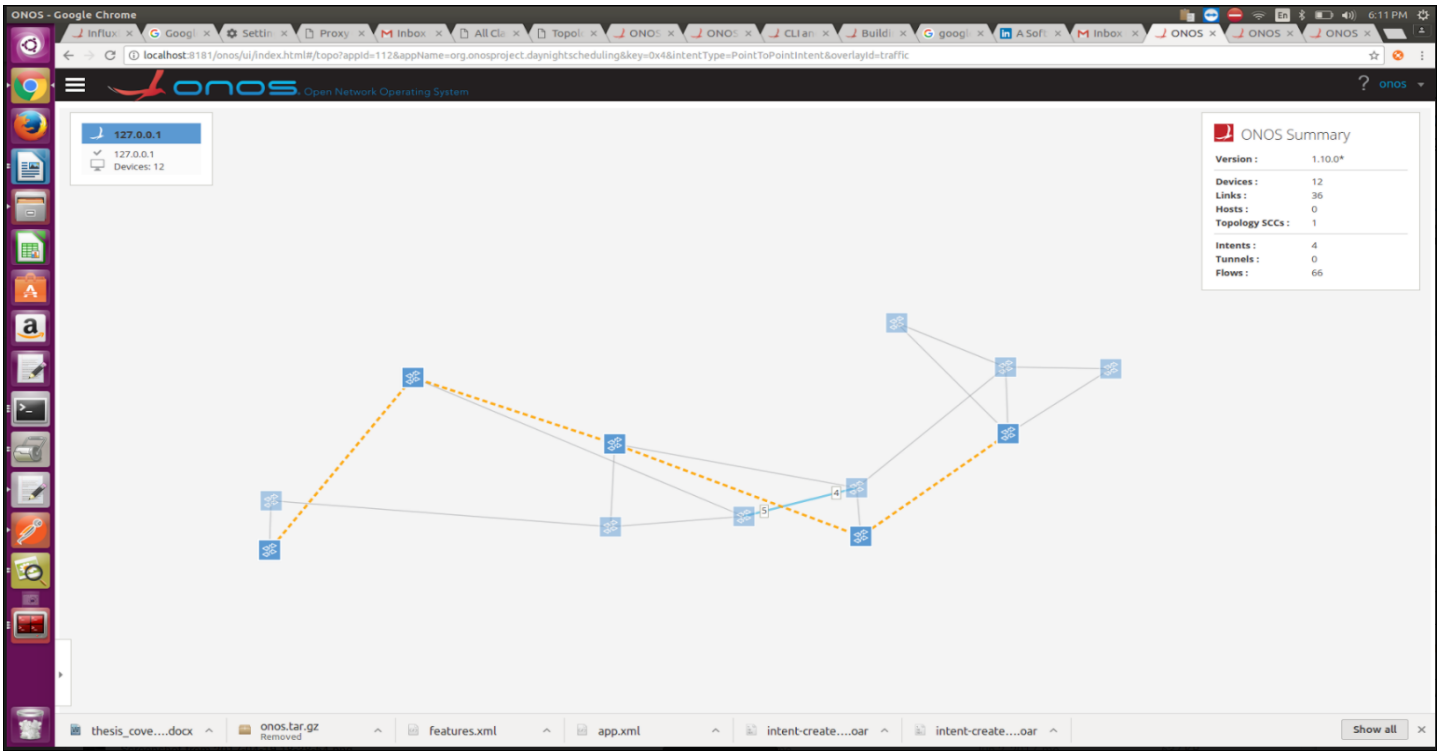


Figure D.3 Successful protected interconnect intent.

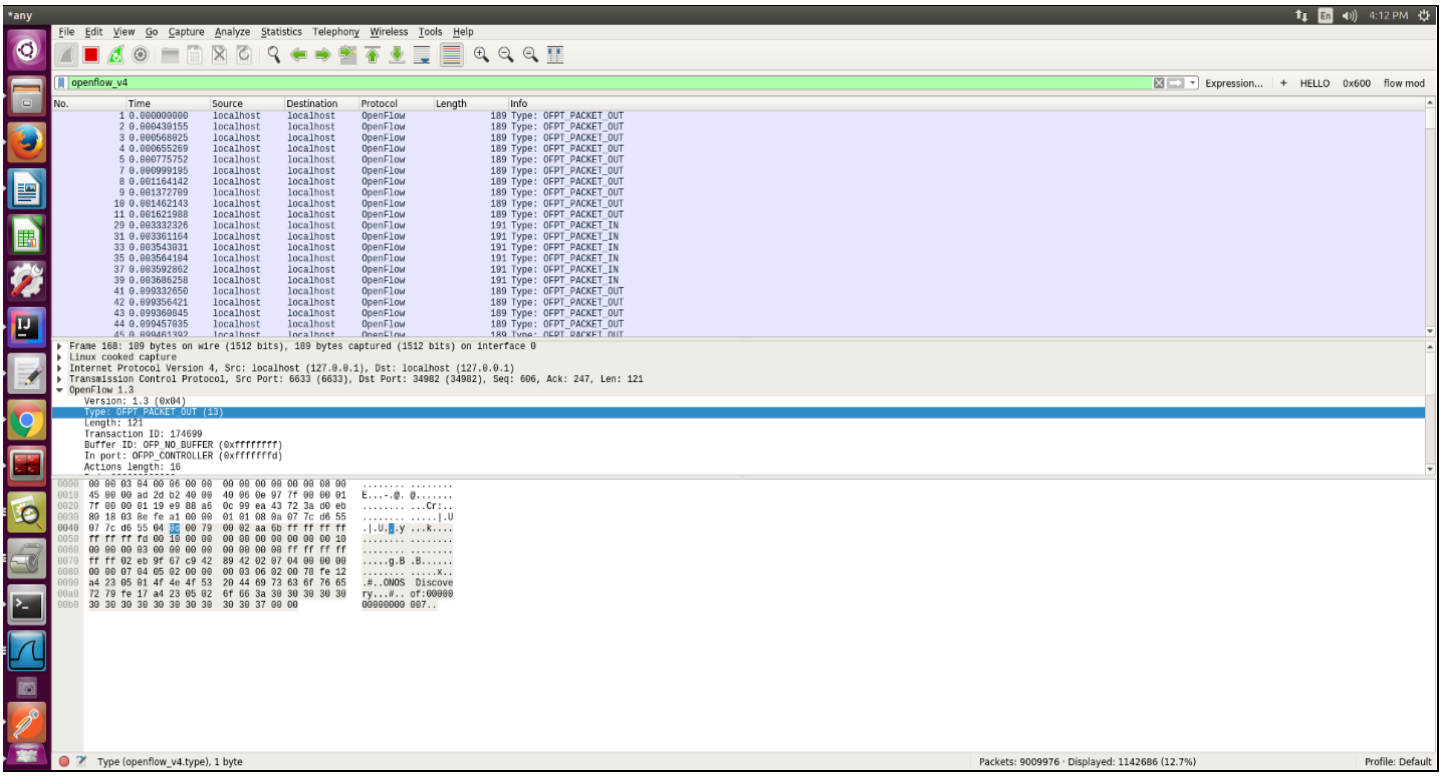


Figure D.4 Capture of Wireshark Flow Initiation for DNS intents.

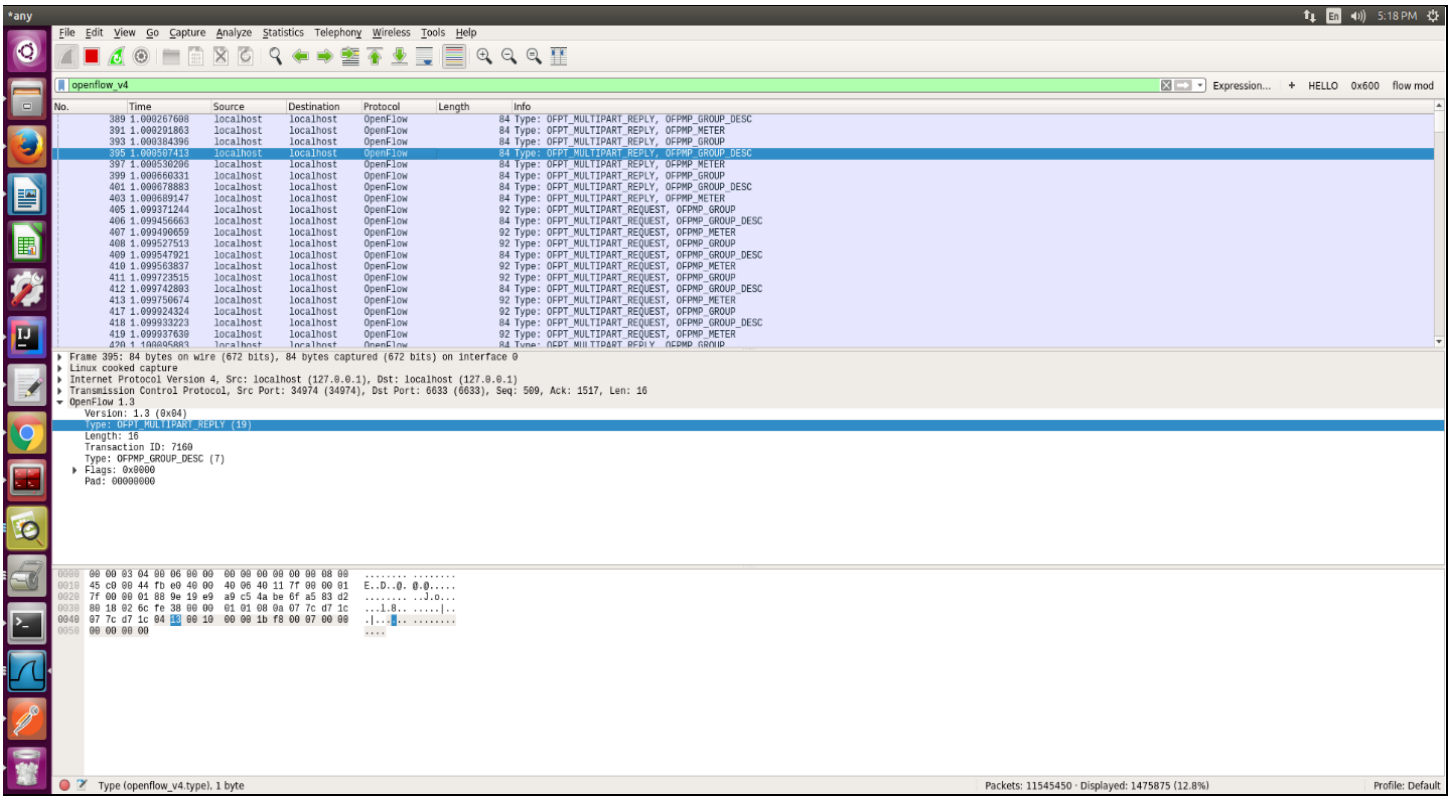


Figure D.5 Capture of Wireshark extending the request for FLOW GROUPS, DESCRIPTION and METERS.

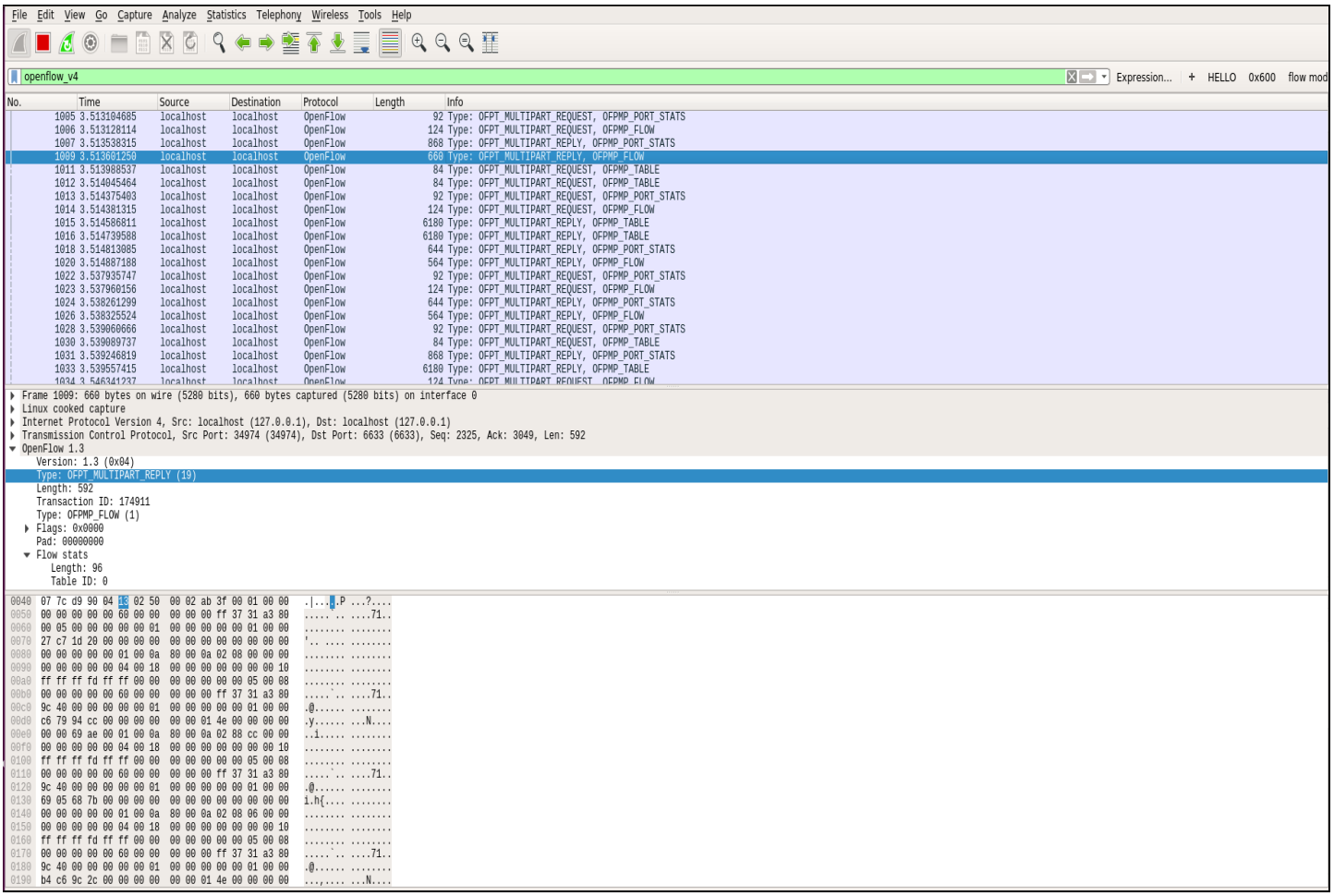


Figure D.6 Wireshark captures for Interconnects created by Intents of the DNS application.