Software Defined Networking For Optical Communications

Yuliya Verbishchuk
HBSc
117223897

Thesis submitted for the degree of
Master of Science

NATIONAL UNIVERSITY OF IRELAND, CORK

Faculty of Science

Department of Computer Science

University College Cork
Tyndall National Institute

December 2025

Supervisors: Dr. Fatima C. G. Gunning
Prof. Cormac J. Sreenan
Dr. Paul Gunning

Contents

I, Yuliya Verbishchuk, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere.

Yuliya Verbishchuk

Acknowledgements

I would like to thank the following people, without whom I would not have been able to complete this research, and without whom I would not have made it through my masters degree!

The photonics systems team at Tyndal National Institute, especially to my supervisor Dr. Fatima Gunning and Dr. Paul Gunning, whose insight and knowledge into the subject matter steered me through this research. Major thanks to Prof. Cormac Sreenan for reviewing and feedback on this project.

And special thanks to Natalia, Amandeep, Niamh, Sean and Shauna for providing guidance, support and feedback throughout this thesis.

Finally, I would like to thank my family for supporting me during the writing of this thesis.

Abstract

Network management has long relied on methods and tools conceived in the early days of the internet. The explosion in network usage due to the proliferation of mobile devices and round-the-clock internet access has revealed the inadequacy of these methods in providing fast and reliable services. Network growth struggles to keep pace with the ever-increasing demands of media consumption and mobile device usage. Consequently, the need for change is imperative to avoid vendor lock-in and introduce flexibility, reliability, and scalability to network management.

The emergence of Software-Defined Networking (SDN) has revolutionized the landscape of network architecture, transforming the way networks and data centers are constructed and managed at scale. SDN's fundamental principle of separating the control plane from the data plane, along with the ability to reprogram and reconfigure the data plane through a centralized controller, has reshaped network operations.

As the demand for network services continues to grow, service providers face the challenge of meeting customer requirements for fast, reliable, and scalable services. Conventional network management methods fall short in adapting to this evolving landscape, and this is where SDN has made inroads, particularly in data center networks.

In Telecommunications (TELCO) networks, a significant portion of the equipment is designed for specific functions, with a set-and-forget operational approach. The transition to incorporate SDN techniques necessitates a careful plan to enable the coexistence of legacy equipment with next-generation hardware and SDN capabilities. This transition allows network administrators and architects to design and manage networks with newfound flexibility, reliability, and scalability.

In this thesis, I show for the first time, that hardware and dissagrgation is possible all the way to the physical layer (Layer 1), including management of legacy disagregated devices along with SDN-enabled devices. At the time this reasearch started, there were very few efforts in integrating SDN controls with open APIs in the physical layer, for example with minimal efforts to introduce wavelength tunability (e.g. Opendaylight v4.0).

The research in this thesis demonstrates the use of NETCONF/YANG to monitor and reconfigure legacy dissagregated network devices within Dense Wavelength Division Multiplexing(DWDM) systems. It also explores the utilization of a SDN enabled optical switch to control the wavelength of transceivers. With an overall controll based on the Open Network Operating System (ONOS), employed for real-time control, monitoring, and on-demand device reconfiguration.

While SDN techniques have advanced considerably in the realm of packet-based forwarding devices, such as Layer 2 Ethernet switches within data centers, their application to optical switches and optical transceivers has lagged behind. This thesis also shows development of methods for controlling and routing optical carriers to output ports of a open software whitebox optical Re-configurable Optical Add-Drop Multiplexer (ROADM) device. This enabled greater flexibility in optical management and the potential for improvement of quality of service in telecommunications networks.

In summary, this thesis demonstrates, through experimental results, the feasibility of controlling a network including legacy disaggregated devices, using SDN tools.

List of Figures

Acronyms

AI	Artificial Intelligence.
API	Application Programming Interface.
ASCII	American Standard Code for Information Interchange.
ASE	Amplified Spontaneous Emission.
ASIC	Application-Specific Integrated Circuit.
BGP	Border Gateway Protocol.
BMC	Baseboard Management Controller.
CLI	Command Line Interface.
CPE	Customer Premises Equipment.
CPU	Central Processing Unit.
CW	Continuous Wave.
DAC	Direct Attach Copper.
DAL	Device Abstraction Layer.
EM	Electro-Magnetic.
EMS	Element Management System.
FCAPS	Fault Configuration Accounting Performance Security.
GPIB	General Purpose Interface Bus.
GUI	Graphical User Interface.
HAL	Hardware Abstraction Layer.
I2C	Inter-Integrated Circuit.

IP	Internet Protocol.
ISP	Internet Service Provider.
LC/UPC	Lucid Connector / Ultra Physical Contact.
LED	Light Emitting Diode.
MAC	Media Access Control.
MIB	Management Information Base.
MUX	Multiplexer.
NBI	Northbound Interface.
NETCONF	Network Configuration.
NIC	Network Interface Card.
NOC	Network Operations Center.
OCP	Open Compute Project.
ODM	Original Device/Design Manufacturers.
ODTN	Open Disagregated Transport Network.
OEM	Original Equipment Manufacturers.
OID	Object Identifier.
ONIE	Open Network Install Environment.
ONOS	Open Network Operating System.
OS	Operating System.
OSA	Optical Spectrum Analyzer.
OSC	Optical Supervisory Channel.
OSI	Open Systems Interconnection.
OSNR	Optical Signal to Noise Ratio.
OSPF	Open Shortest Path First.
OSS	Operational Support System.
P4	Programming Protocol-independent Packet Processors.
PISA	Protocol Independent Switch Architecture.
PPPoE	Point-to-Point Protocol over Ethernet.
QSFP28	Quad Small Form-Factor Pluggable 28.
RFC	Request For Comment.

RIP	Routing Information Protocol.
ROADM	Reconfigurable Optical Add Drop Multiplexer.
RPC	Remote Procedure Call.
SBI	South Bound Interface.
SDN	Software Defined Networking.
SFP+	Small Form-Factor Pluggable +.
SNMP	Simple Network Management Protocol.
SRAM	Static Random Access Memory.
SSH	Secure Shell.
TCAM	Ternary Content-Addressable Memory.
TCP	Transmission Control Protocol.
TELCO	Telecommunication Operators.
TLS	Transport Layer Security.
TSFP+	Tunable Small Form Factor Pluggable +.
VXLAN	Virtual Extensible Local Area Network.
WAN	Wide Area Network.
WIFI	Wireless Fidelity.
WSS	Wavelength Selective Switch.
XML	Extensible Markup Language.
YANG	Yet Another Generation.

Chapter 1

Introduction

1.1 Context

Many services such as browsing, emails, and e-commerce are hosted across the world by an interconnected mesh of data centres. Both data centres and the networks that connect them are invisible infrastructures often overlooked by users. These data centres form a Wide Area Network (WAN) that covers the entire planet as shown in Figure 1.1. This is the Internet. The Internet and networks are now ubiquitous tools used and needed by our society as a whole. End users connect to the Internet via Ethernet, Wireless Fidelity (WIFI) or cellular networks, physically using copper wires or optical fibre; or wireless respectively. The data centres and networks physically connect using the optical and electrical domains to deliver services to end users. Managing such a connected network is a very complex operation to ensure the reliable provision and continuation of services. We only notice this when the service is affected.

The physical equipment that forms the underlying network supports a mixture of electrical and optical transmission and switching. Data-modulated information is transferred onto electromagnetic waves of a particular frequency (or wavelength) and then transmitted between locations in the optical domain. The packet domain, which sits above the optical domain, organizes the data information such that each packet contains a header at its front. The header includes a source and destination addresses that are used to forward each packet closer to its destination. Decisions are made at locations along the way to forward every individual packet closer to its particular destination.

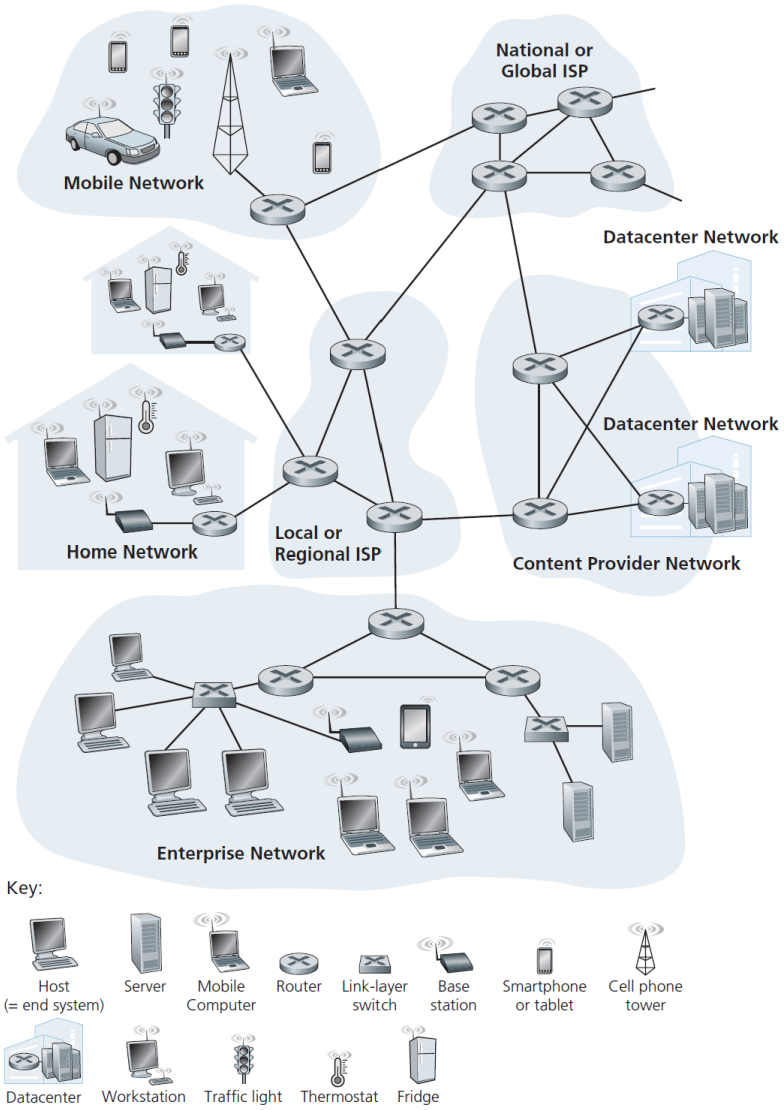

Figure 1.1: Illustration of a typical Internet connectivity. The Internet is a collection of many complex networks: Global, National, Regional, Metropolitan, Local, personal. It uses both fixed and mobile network infrastructure.[1]

The Open Systems Interconnection (OSI) protocol stack spans from the physical transmission medium (Layer 1) to the useful applications (Layer 7) that we use every day (e.g. WhatsApp). It will be addressed in further detail later in Chapters 3 and 4. However, this thesis will focus on the physical layer (Layer 1) formed from either photonic or electronic transmission media augmented by the data link and physical network layer.

The network elements (Switches, Routers) within the large, complex networks operated by the hyper-scalers (e.g. Google, Microsoft and Meta) and the telecommunications providers (e.g. Eircom, Telefonica and France Telecom) need to be managed remotely

and at scale. The suppliers (or vendors) of the network elements provide Application Programming Interface (API) for this purpose. Traditionally each vendor provides a proprietary API that can only be used for their network elements. This is commercially advantageous from the vendors' point-of-view because it ensures dependency by making it difficult to substitute the network elements they supply for alternatives from their rivals. The network operators call this practice "vendor lock-in" because it is difficult for them to easily substitute, deploy and operate competitive alternatives. As a result, a complex network containing network elements from several vendors has many incompatible APIs which make it very inflexible to operate.

A decade-and-a-half ago a group of researchers at Stanford University led by prof. Nick McKeown[11] set out to "disrupt" the commercial model of vendor lock-in by defining an open standard to manage both the control plane and the forwarding plane of network elements. They defined an open, programmable API that they called Openflow[12]. The Openflow API made it simpler for operators to program how Layer 2 data link frames (e.g. Ethernet) and Layer 3 network (e.g. IP) packets would 'flow openly' across a network comprising elements, sourced from many different vendors. Openflow could effectively replace many proprietary APIs with a single open API. It was very disruptive and it was resisted by the vendors because it was easier to substitute their products for those of rivals.

Although OpenFlow has been widely used, it was not the first adoption of open networking. However, it was the first practical and widely adopted implementation. Preceding efforts like Active Networks, ForCES, and PCE laid the foundation for its success.[13] As the most prominent innovation, OpenFlow drove the academic push that ultimately shaped what is now known as Software Defined Networking (SDN). SDN allowed researchers to perform experiments with commercial network elements by offering an open and centralized alternative to manage the control planes and the data planes across many proprietary network elements.

Openflow was the fundamental API of SDN [14] as it:

- allowed (or denied) the entry of frames and packets into a network of once proprietary under its control;
- explicitly define the path those packets or frames took across the network to enter, traverse and exit the network.

Fundamentally every network element has two components: the hardware, and the software that controls the hardware. SDN separated, or disaggregated, the hardware from the software. This was not welcomed by the vendors of network elements because it dis-

rupted their commercial models. It meant that the closed blackboxes with proprietary APIs that were sold by the commercial network element vendors, also called Original Equipment Manufacturers (OEM) could be substituted by open whiteboxes with open APIs that could be competitively sourced from Original Device/Design Manufacturers (ODM). This thesis will not explore the commercial implications of SDN. Instead, it explores the technical advantages that were made possible to extend or augment the use of SDN to not only control how data can be programmed to flow precisely across a packet network, but also how the frequency (or wavelength) of an electromagnetic carrier that supports the transmission of this data contained within the packets could also be independently programmed.

So, if SDN was the solution, what then was the original problem that it eventually contributed to solve? The original problem was explained over two decades ago in a workshop between the operators of complex networks and the vendors of the blackbox network elements that collectively formed this complexity. The output of that workshop was captured in RFC3535[15]. The RFC explained how the tools that were then available for the management and monitoring of complex networks were overwhelming. They caused lots of errors to be introduced during the configuration of the network elements. The operators explained what the major 'stresses' were that they had to contend with, but more importantly what they wanted:

- ease of use;
- a distinction between the data and statistics that were used to configure a network from the data and statistics used to operate the network e.g. SNMP was good for the latter, but not the former - configuration;
- how the network needed to be more easily configured as a whole and in its entirety, and not just as a collection of different, individual network elements.

The seven layer OSI model (or OSI 'stack') logically separates independent functions that serve to reliably deliver information between a sender and a receiver. The information that is exchanged might be the contents of an e-mail or a YouTube video or zoom call. Figure 1.2 shows both the seven vertical layers that every sending and receiving device must have. It also illustrates how each separate layer in the sending device connects with its peer layer in the receiving device.

This thesis is primarily focused on Layer 1, the physical layer; Layer 2, the data link layer; and Layer 3, the network layer of the OSI stack.

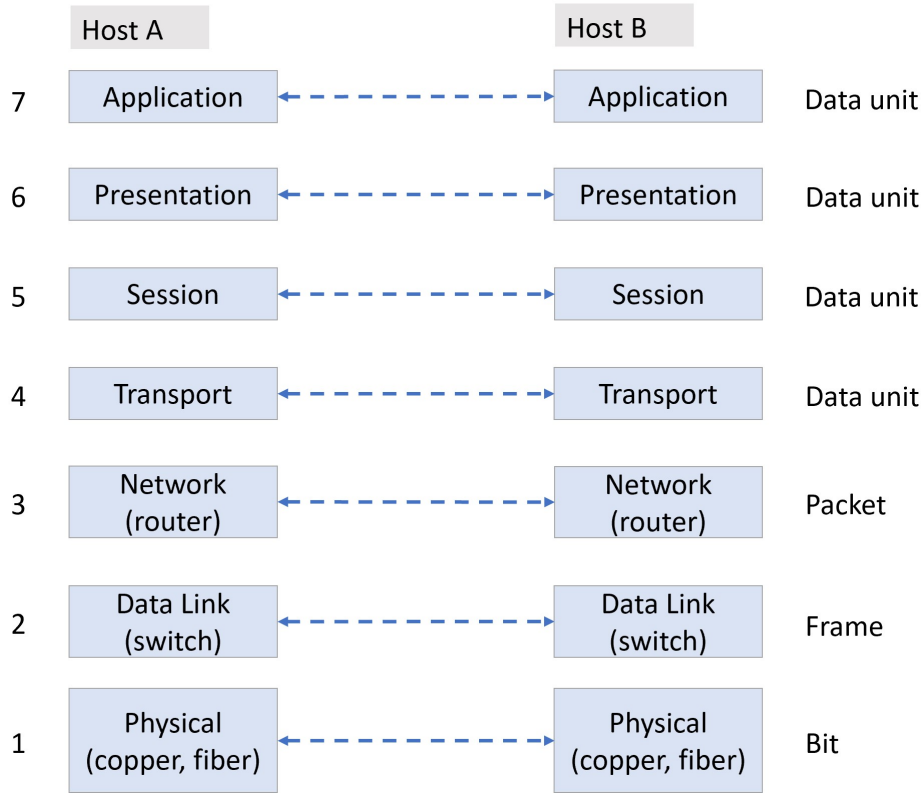

Figure 1.2: The seven layers of the OSI model, adopted from [2].

Layer 1 transmits data 'bits' (e.g. one's and zero's) between a sending and a receiving device. The transmission medium used was either optical fibre or copper wire/traces. The 'bits' are formed by modulating a property (e.g. amplitude or phase or frequency) of an Electro-Magnetic (EM) carrier.

Layer 2 orders the Layer 1 data 'bits' into a 'frame' that encapsulates the data from the layers above, i.e. Layer 3 to Layer 7 of the sending device. The frame has a header that includes the source address of the sending device and the destination address of the receiving device. The frame also has a 'tail' that verifies the integrity of the 'bits' that make the frame. By integrity we mean that the frame that is transmitted by the sending device is perfectly replicated at the receiving device. In this thesis, Ethernet was used, but open and programmable packet frame could also be used (e.g P4). Layer 2 is usually confined to the point-to-point transmission link that directly connects the sending device to the receiving device. A switch forwards frames, which has many Layer 2 ports, is used to forward frames across many point-to-point transmission links whenever the sending device and the receiving device are far apart. In subsequent chapters we'll describe how OpenFlow can be used to modify fields within the Layer 2 header of Ethernet frames. OpenFlow also provides the API that is used to 'push' rules down into the forwarding table of a Layer 2 switch to direct frames to leave from

a particular switch port.

Layer 3 orders the bits within the data section of the Layer 2 frame into packets. Each packet once again has a header. The header contains an Internet Protocol (IP) source address of the sending device and the IP destination address of the receiving device. A Layer 3 router has many ports that are used to 'route' or 'forward' Layer 3 packets from the sending device to the receiving device when they are very far apart across the Internet. We also can use Openflow to route Layer 3 packets. Typical structure of data link frame and network layer packet is shown in Figure 1.3.

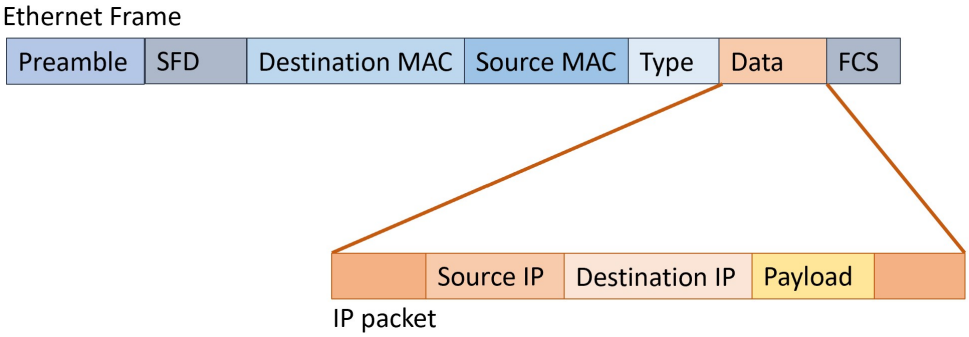

Figure 1.3: Network packet and data link frame structure.

Another transition challenge that operators face in moving to disaggregation architectures is one of timing. While network teams plan for future architectures, the product teams have service and feature requirements that must be met today. Competition does not allow operators to skip a cycle of important feature upgrades while waiting for the next generation of devices to arrive, and so the transition from old to new must also be managed with today's needs in mind.

1.2 Motivation

Networking research spans from the large wired networks that span the Earth, to the small networks that use WIFI within our home. The hyper-scalers like AWS, Google and Microsoft Azure, host services (e.g e-mail, video conferencing and video-on-demand) that are sold to customers. The services are carried over the networks that are provided by the network carrier who are in the main, Telecommunication Operators (TELCO)s.

For example, Cisco sells fixed network equipment to hyper-scalers, TELCOs and enterprises. Every few years the Cisco Annual Internet report [16] provides forecasting insight into the broad trends that will affect the hyper-scalers and the TELCOs alike. Unsurprisingly as a company whose job is to sell equipment, the latest edition [17]

predicted in 2020 that the growth will continue as shown in Figure 1.4. It offers some interesting, albeit self-serving, macro trends. These include:

- There will be 3.6 networked devices per person by 2023, up from 2.4 networked devices per capita in 2018.
- There will be 5.3 billion total Internet users (two-thirds of the global population) by 2023, up from 3.9 billion (51 percent of global population) in 2018.
- The average global fixed broadband speed will be 110 Mbps - of course not all users will have fixed broadband. Most developing communities will rely on wireless mobile connection for the final 'drop' to each person.

Another vendor Ericsson sells wireless network equipment to TELCOs. Their Ericsson global mobility calculator provided insight into the services that are carried by the bits that consume the bandwidth. They use 2020 as their benchmark and suggest shown in Figure 1.5 that:

- the average person bandwidth consumption will increase by a factor-of-four, from 9 GBytes-per-month in 2020 to 35 GBytes-per-month by 2026;
- the fractional breakdown of the bandwidth consumed will be dominated by video streaming (up from two-thirds in 2020 to four-fifths by 2026).

This intuitively feels about right. Most people are consuming video streaming services on wireless, mobile devices compared to smart home TVs connected to the fixed, access network. These predictions were true before the boom of Artificial Intelligence (AI) applications. Figure 1.4 illustrates the predictions of the growth of number of Internet users by 2023. At the time of submission of this thesis, these predictions have been achieved creating the need for agile networks. The trends in networking growth are evolving steadily, driven by the rapid rise of AI technologies. [18]

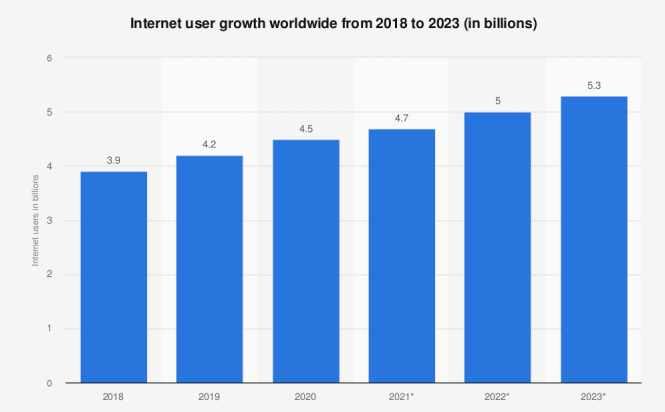

Figure 1.4: Worldwide Internet users growth from prediction by Cisco Annual Internet Report 2018-2023 [3]

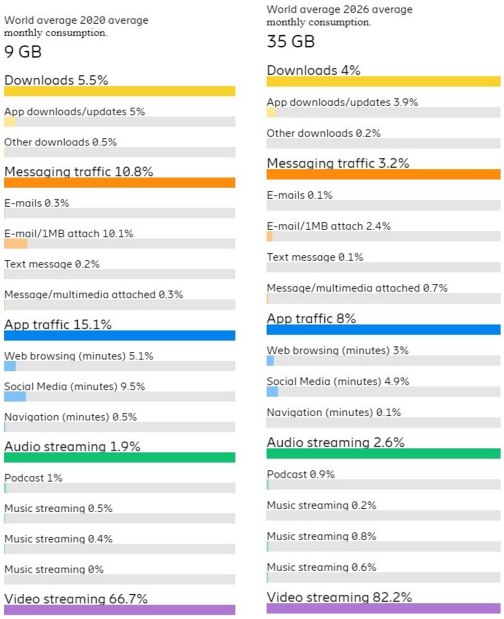

Figure 1.5: Worldwide Internet usage predictions from Ericson calculator[4]

It is a necessity for service providers to satisfy this demand and scale their networks. This is met by increasing the bandwidth capacity within the many individual network elements, like those shown in Figure 1.1, that form the end-to-end path between each persons mobile phone and the content providers. When the capacity of the network element is exhausted - then new equipment with greater scalability will be needed to replace existing, legacy network elements. This is a very expensive proposition. TELCO revenues are flat, or increasing modestly at best. Therefore to grow profits, costs must be cut. This means more automation and fewer people. It also means reducing the costs of the network elements that are sold by global suppliers like Cisco, Ciena, Nokia and Juniper. They are expensive, 'blackboxes' that aggregate, or combine, several software and hardware components that together provide particular network functions.

These network elements have traditionally been programmed with an API abstraction to perform certain rule-based tasks, for example take an ingress data-link frame from one port and forward it out a different egress port. It is important to make the distinction that open APIs are standardized, openly defined interfaces, while data plane programmability refers to the capability to dynamically reconfigure the data path. For example, a system can offer openness through standard APIs without necessarily providing full data plane programmability. The problem is that each supplier, to resist substitution by rival products, provides its own unique proprietary API. This makes it very complicated and very expensive to program the many network elements that are sourced from many suppliers. This was recognized two decades ago by the hyperscalers. They stopped buying blackbox network elements from the global equipment suppliers. They just could not maintain their profitability. So, instead, they began to design their own whitebox network elements. These deliberately separated, or disaggregated, the software and the hardware. They then used OpenAPIs that were put into the public domain that managed, controlled the forwarding of data flows that were mostly comprised of data link layer Ethernet frames, or network layer IP packets. Disagregating network elements enables more choice to identify the most suitable devices at the keenest price. It also unlocks the potential to design universal network elements that can scale to meet future demands [19].

The growth in demand increases the strain on the network resources, thus the current network management strategies and methods are being widely researched.

Disagregating network components enables better choices in terms of most suitable devices,and unlocks the potential to design a system to meet current and future demands.

Current research is investigating not only entire network architectures, but also at individual components/device level, which shows the need for disaggregation and centralized controls, enabling re-configuration of devices for flexible provision of resources as demand increases.

This thesis is based on the work to enable reconfigurability of disaggregated devices and instrumentation, managed by a single logical controller. This enables a potential new network architecture, which would be cost effective and very flexible from both client and line sides. This is thoroughly discussed in Chapters 3, 4 and 5.

At the start of the research for this thesis (in 2018), it was clear that there was no available path for implementation of disaggregated devices and software, with a few demonstrations of SDN in place [20], [21]. The benefit of a centralized software controls is the ability to reconfigure the network as needed, including both physical devices

and the data path. Most of devices that use Application-Specific Integrated Circuit (ASIC) are locked, because ASIC is not programmable.(ASICs are typically designed for specific function, and is one of main components of a network switch). However, this is now changed with the new devices being developed, for example from Barefoot Networks (now Intel). They designed a fully programmable switch, where the routing and forwarding rules can be openly programmed. This is exciting, as implementation of SDN becomes closer to reality.

However, quick replacement of equipment is not feasible, and in this thesis we also research the potential to disaggragate the physical equipment further, to enable incorporation of legacy devices to SDN.

The evolution of networking that is happening at the moment is calling for new ways to monitor, control and manage devices.

1.3 Problem Statement And Research Objectives

This thesis is concerned with moving data or information from one place to another by changing or manipulating some attributes of the bottom three layers of the OSI stack: Layer 3 - network layer (e.g. IP packets); Layer 2 - data link layer (e.g. Ethernet frames); Layer 1 - physical layer e.g. the electromagnetic (EM) carrier wavelength(nm), or equivalently, frequency(THz).

The aim of the thesis was to provide a basis for the manipulation, automation and orchestration of the:

- EM wavelength @ Layer 1;
- Ethernet header fields @ Layer 2;
- IP header fields @ Layer 3.

High-performance application specific integrated circuits (ASICs) have been commercially available for several decades that expedite Layer 2 (frame) and Layer 3 (packet) forwarding across a network. These network ASICs were traditionally fixed-function so they were somewhat inflexible in the tasks that they could perform (e.g. forwarding between an ingress and an egress port), without using an commercial equipment vendor proprietary API. Access to experiment with the ASICs was effectively locked - a very effective and regrettable barrier-to-entry to researchers and innovation. Nonetheless one PhD student - Martin Casado - was undaunted when he commenced his PhD, nearly two-decades ago to investigate if it was possible to exploit these ASICs within

commercial enterprise Ethernet switches that were formed a connected of network elements. Commercial switches relied on protocols like spanning-tree to ensure loop-free forwarding of Ethernet frames at Layer 2; or distributed routing protocols, like OSPF, to do the same at Layer 3. All to manipulate the forwarding, or flow, of protocol data units (PDUs). In [14], Martin Casado wanted much more control. He wanted specifically to:

- define a centralized controller with the ability to read from and write to the forwarding-table (MAC table or routing table) within a set of interconnected network elements - this required an Open API;
- provide a 'deny all' to L2 frames and/or L3 packets that were inbound from outside the set of network elements;
- allow "some" of these frames and packets to enter the network, subject to a defined policy;
- explicitly set the forwarding path that those frames or packets followed as they moved between the network elements.

and set provided fundamentals for what became the software defined (SDN) era of networking that continues to the present day.

Openflow became the defacto OpenAPI and protocol that issued from a centralised SDN controller. Although is was short-lived, see Figure 1.6,

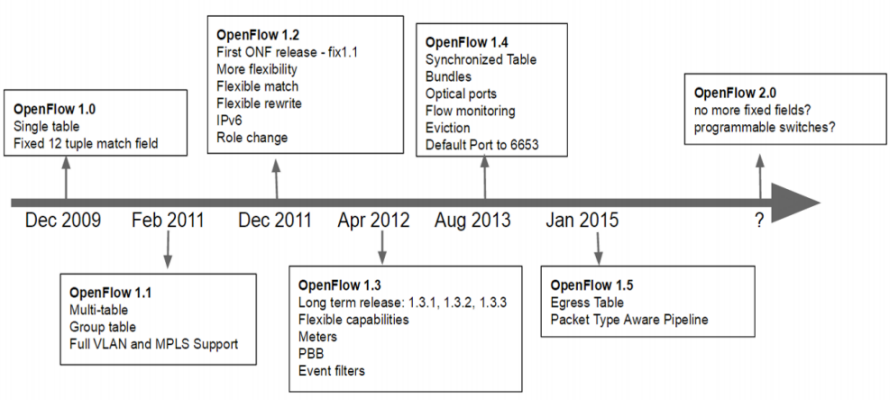

Figure 1.6: OpenFlow Version Timeline.

it was profoundly influential. It was succeeded by Programming Protocol Independent Packet Processors (P4) which might be considered as "Openflow 2.0". P4 maintained the open source nature of Openflow. It also allowed access to the registers a variety of target domains: virtual (e.g. BMV2); FPGA (e.g. NetSUME FPGA); ASIC (e.g.

Barefoot, now Intel, Tofino); using the P4 programming language,although the flexibility of first two not explored in this thesis.

In this thesis the known SDN developments in Layer 2 and Layer 3 of the OSI stack were harnessed. Much less developed was how SDN can be applied to the manipulation of optical parameters of the electromagnetic carrier at Layer 1. And this is the focus of this thesis. Here it will be shown that such parameters, specifically wavelength, can be "switched" or programmatically forwarded by an optical forwarding element called a reconfigurable optical add-drop multiplexer (ROADM). This required exploration of how the open network configuration and management protocol called Network Configuration (NETCONF) [22] could be used to READ from and WRITE to the parameters of a Yet Another Generation (YANG) data model [23] that, in turn, reflected the state of a small number of registers in a wavelength tunable, small form factor pluggable (SFP) transceiver. This shows the experimental implementations of this concept for the first time, to the best of our knowledge.

In this thesis we demonstrated the means to:

- use P4 to READ/WRITE packet header fields at Layer 2 and Layer 3 of the OSI stack. This will be the subject of Chapters 4.
- use NETCONF/YANG to READ/WRITE transceiver registers and the forwarding state of an Optical ROADM at Layer 1. This will be the subject of chapters 3 and 4.

We also outline the means for orchestrating these abilities concurrently using the open ONOS SDN controller, in Chapter 3.

And finally we summarize and outline some future work that might be undertaken in the future to progress this new and vibrant field of research.

1.4 Outline of the Thesis

Chapter 1 contextualized the research, challenges and opportunities. It presents a brief background and motivation behind the project and then presents the research questions addressed in this thesis.

Chapter 2 presents the summary state-of-the-art review of physical layer network management and SDN at the time of researching working on this experiment in 2018, including challenges with the technologies used and trends that are emerging within the topic since the beginning of this work. This chapter briefly introduces the technologies

and areas involved in this project and then reviews the relevant literature.

Chapter 3 looks into more details of the network management and how this work was structured to meet the set goals. It also deep dive into the architecture and internal details of the various components of the system that were used in this work.

Chapter 4 explains the new range of devices that are emerging with the big trends of software defined networking and Programming Protocol-independent Packet Processors (P4) language enabled. Also it, discusses the trends and the idea of open and disaggregated transport networks. In this chapter we discuss difference between optical switching and packet switching, and the devices used in those domains within the experiment and the technologies that make those devices so special.

Chapter 5 describes the experiment that was conducted during the project duration. Discusses its architecture and components, and results and findings of the experiment.

Chapter 6 concludes the thesis, talking about the challenges faced and contributions made to the state-of-the-art. It ends with a discussion on the future work in this direction an overview and discussion of the observations gathered through-out the project. This chapter places the research and findings into the overall big picture of SDN.

Outputs: Key publications for this work include:

- S. Ahearne, Y. Verbishchuk, C. Sreenan and F. Gunning, "Software Defined Control of Tunable Optical Transceivers Using NETCONF and YANG," 2018 European Conference on Networks and Communications (EuCNC), Ljubljana, Slovenia, 2018, pp. 1-86, doi: 10.1109/EuCNC.2018.8443188.[24]
- Y. Verbishchuk, C. Sreenan, and F. Gunning, "Configuration and monitoring of the optical physical layer using software-defined tools," in OSA Advanced Photonics Congress (AP) 2019. Optical Society of America, 2019, p. NeT1D.2. Available: http://opg.optica.org/abstract.cfm?URI=Networks-2019-NeT1D.2 [25]
- F. C. G. Gunning, A. Kaur, N. C. Estrada, J. Raulin, Y. Verbishchuk, and C. J. Sreenan, "Enabling High Capacity Flexible Optical Networks," 2021 International Conference on Optical Network Design and Modeling (ONDM), Gothenburg,Sweden,2021, pp.1-3, doi:10.23919/ONDM51796.2021.9492503. [26]
- J. Raulin, G. Davey, Y. Verbishchuk, A. Jeffries, C. J. Sreenan and F. C. Garcia Gunning, "A programmable Ethernet transport Packetponder using common compact form factor pluggable tunable transceivers to support novel DWDM architectures," 2023 Optical Fiber Communications Conference and Exhibition

(OFC), San Diego, CA, USA, 2023, pp. 1-3, doi: 10.1364/OFC.2023.Tu3D.6. [27]

- J. Raulin, G. Davey, Y. Verbishchuk, A. Jeffries, C. J. Sreenan, and F. C. G. Gunning, "Packetponder with open-source management system for future packet-optical networks," in Proceedings of the 2023 Networks Conference. Optical Publishing Group, 10 2023, p. NeM3B.3. [28]

Chapter 2

Fundamentals Review

2.1 Physical Layer Network Management

The International Organization for Standardization OSI Reference model[2] describes networking as "a series of protocol layers with a specific set of functions allocated to each layer", which structures and simplifies provision of resources and services. Each layer offers specific services to higher layers while shielding these layers from the details of how these services are implemented [29]. This is exemplified in Figure 2.1. In this thesis we concentrate on three bottom layers: physical layer (Layer 1), data link (Layer 2) and network layer (Layer 3), and their management.

Physical layer is responsible for the medium where the information, or bits, is travelling on, which includes the optical, electrical and wireless domains. The main functions implemented in the physical layer combine: representation of bits, determines the type of the signal used for transmitting the information, specifying the data rate, provides an interface such as port, and define how two or more devices are physically connected. Historically, network management of a physical layer only relied on setting up its physical parameters as above, including wavelength or carrier frequency for point-to-point transmission, upon configuration, with little room for flexibility. Monitoring becomes important in this case, checking for issues and potential component failures. Physical network layer is normally configured on a set-and-forget basis, with no further need for change, unless there is conflict or a problem. This view of "static physical layer" can result in obsolete management tools. Such static or semi-static nature of the physical layer is now an impediment to allow new photonic and electronic components to flexibly and efficiently provide for capacity and bandwidth when needed, and hence fulfill the fast growth of bandwidth hungry applications.

Data link layer is responsible for taking packets and adding relevant headers, and hence creating frames, ensuring successful safe delivery of those frames from a point to another over the medium. Data link layer is itself comprised of the two layers, Logical Link layer and Media Access Control layer. Logical Link is responsible for flow control of frames (the data units of information that are being sent) between each logical source and destination node; while media access control sub-layer deals with Media Access Control (MAC) address[30], which is a 12-digit hexadecimal number assigned to each device connected to the network. And is used to access physical network interface card on the device. Data Link layer is responsible for forwarding packets and frames.

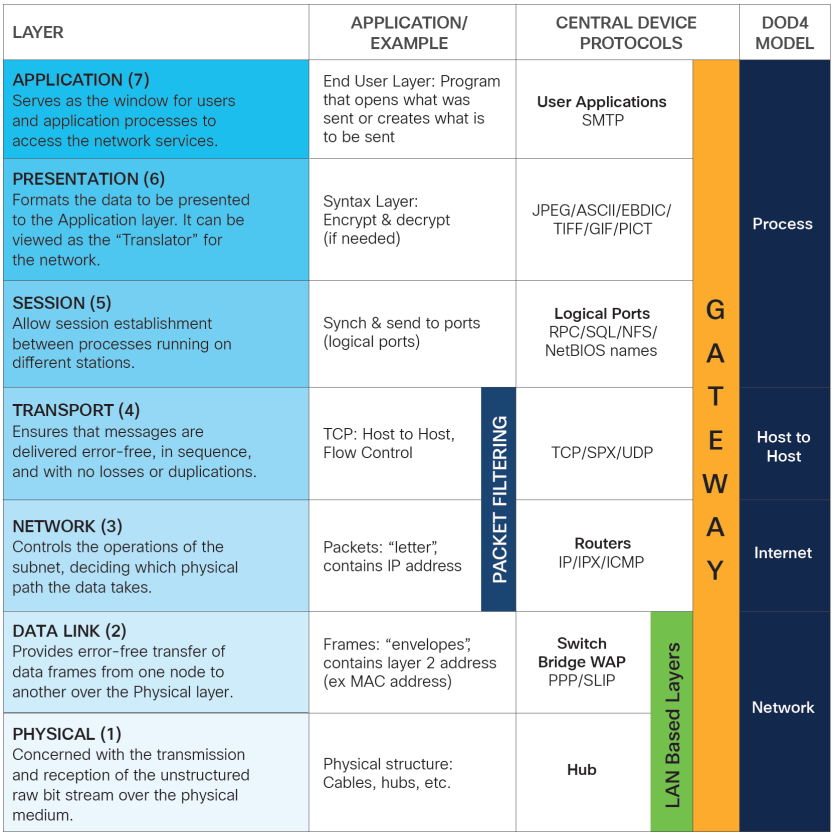

Figure 2.1: The seven layers of the OSI model as per Cisco [2].

Network layer provides a logical connection between different nodes and is responsible for breaking up the data into packets and then restructuring it at the arrival on the node. A Network layer adds the source and destination Internet Protocol (IP)[31] address to the header of the frame."Addressing" is used to identify the device on the Internet. Routing is the major function of the network layer, and it determines the best optimal path for fastest and cost-effective route from source to the destination on physical network.

Management of the functions of each layer is done on each device, via the use of Network

Interface Card (NIC). NIC provides the circuitry required to implement and work with different layers' standards and protocols. Each NIC represents a device, that is ready to prepare, transmit and control the flow of the data. Such as each device will have a flow table and will calculate the best path between different endpoints. It will keep a list of its nearest nodes and also will check the packets upon arrival. Figure 2.2 exemplifies how the different layers are triggered at different physical nodes in a communication network. For example, Host A sends a request or data to Host B. A routing protocol implemented on the routers in the path, will use algorithm to calculate best path and assign a path for packet delivery from Host A to Host B. [32]

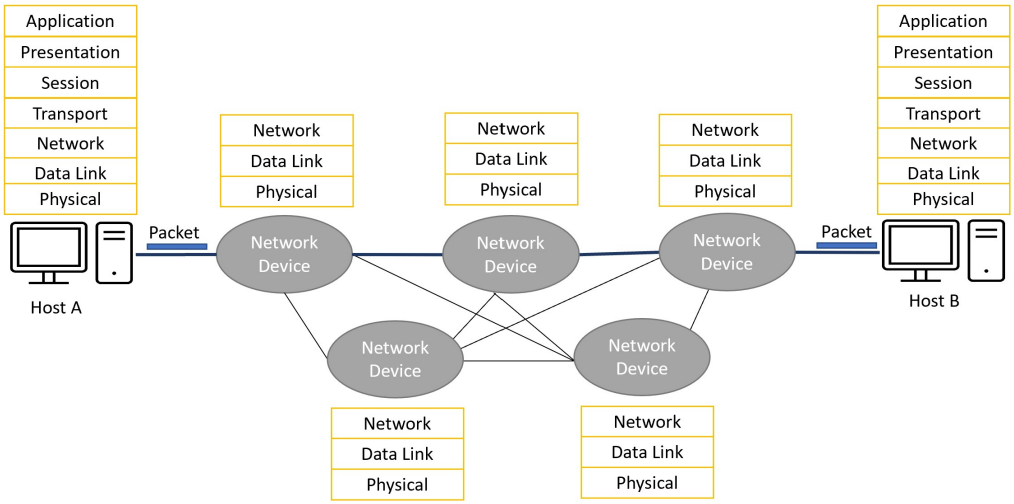

Figure 2.2: The host-to-host-system illustrates the layers and how they applied when using traditional approach.

The actual physical size of a communications network has grown significantly in the last decade. At first, it was sufficient to manage the network with an operator fixing issues that arise, as was with early days telephony. However, it has now grown, and management has become much more challenging, systems became more complex and device-locked.

Nowadays, physical layer consist of a multitude of devices. Such devices are switches, transponders and routers, which include elements from all domains electrical, optical and wireless. Management of the network is significantly more complex because of the number of devices and the constant demand that requires the service to be available 24/7/365.

With the growth of the network size, the management of the physical network becomes cumbersome. Many Internet Service Providers (ISP) outsource this task to the specialized companies, that specifically manage the physical network and its components. The challenges that are met by network managers nowadays include network security,

data breaches, constant user growth, regulations, constantly evolving threats, configurations, expanding network and more devices, cost and expenses, network performance, more traffic and more complexity, time spent troubleshooting, network health and network management solutions.

2.2 The network management of legacy devices and related issues

The Fault Configuration Accounting Performance Security (FCAPS) of an ISPs/TELCOs network is complex and expensive. TELCO and ISP networks use network hardware elements from a variety of vendors. Each vendor provides a proprietary, closed API to control and manage its closed, proprietary network hardware elements within the extent of its domain using the vendors specific element managements system (EMS). This acts as an effective barrier to the entry of alternative, rival vendors and prevents difficulties for ISPs/TELCOs to easily substitute hardware elements.

Traditional network management systems can be divided into several components that are shown in Figure 2.3:

- Managing server. This is a CPU-based hardware device that hosts the Element Management System (EMS) application in the Network Operations Center (NOC) of the Internet Service Provider (ISP) / TELCO. It is operated by trained network operations personnel. The managing server provides an overview summary of all the FCAPS activity on the network, including collection, processing, analysis, and dispatching of network management information and commands. This is the point where operations to configure, monitor and control the network device are initiated.
- Managed device. This is a network hardware element that hosts EMS application software on the managed network. It can be any device, that is network-connected, for example, switch, router, Customer Premises Equipment (CPE). The device itself will have many manageable components, for example, an Ethernet port, and configuration parameters for the hardware and software components, for example protocols such as Open Shortest Path First (OSPF) or Point-to-Point Protocol over Ethernet (PPPoE).
- Structured Data. Each managed device will have both static and dynamic data which represents its moment-by-moment ‘state’. There are a few types of data defined: (a)configurational data is a setting specifically defined by the device op-

erator, for example port speed; (b)operational data is related to device awareness such as list of neighboring devices, statistics that indicate the status of a network hardware element, including processing speed or temperature.

- Network management agent. This is the software application that supports communications and data exchange between a managed device and the managing server to happen, it supports FCAPS data exchange between the managed device that is under the control of the managing server.
- Network management protocol. It is a defined sets of rules running on the managed devices and allowing managing servers to query them via an agent living on the managed devices. In practice a typical end-to-end network consists of network hardware elements (e.g. managed devices) from many different vendors - each with a bespoke element manager. An Operational Support System (OSS) is then required that acts as the overall manager of the many, disparate element managers. This introduces massive additional complexity. It has been said that the management of an ISP/TELCO network is like changing the engine of a Boeing 747 at 30,000 feet halfway between Dublin and New York!

As new network hardware elements are introduced with their EMSs, it is still necessary to continue to run and maintain the legacy network elements and their EMSs. This is a very expensive proposition. It is helped by the use of the Simple Network Management Protocol (SNMP) which is a Layer 7 application, see Figure 2.1, that can gather data from the individual network hardware elements. The gathering or "reading" of FCAPS information is made possible with the SNMP protocol, but the way the data is structured using the SNMP Management Information Base (MIB) is vendor-specific and proprietary. In addition, SNMP has little support to configure or "write" data to the various registers and tables of network hardware elements. The "configuration API" of network hardware devices relies predominantly on the CLI which is proprietary to each vendor. A new approach to succeed SNMP was sought two decades ago, in June 2002, which is summarized in an "Overview of the 2002 IAB Network Management Workshop" [15]. It essentially deprecated the use of MIBs and recommended that a new approach was necessary. It took a further four years for a consensus to emerge around the adoption of the YANG data modeling language and the NETCONF protocol as the successor to SNMP and MIBS. We will defer discussion of the NETCONF/YANG approach to network management to section 2.3. First we need to outline the complementary activity around SDN.

Legacy devices can be understood as hardware that implements standardized protocols directly in specialized microprocessors within network devices. Physical hardware used

by network service providers are typically procured from one vendor for an entire end-to-end solution, for example from device manufacturers such as Cisco, Ciena or Broadcom etc. Those devices are configured with a specific set of software tools that tend to block access to underlying physical architecture by the vendor, and leaving only a small set of features that operators can use to configure devices. This means that the user has an option to configure a given telecom device or equipment, but does not have an option to reprogram it.

For a long time the physical layer devices didn't evolve as much, and changes that were happening were at network protocol "rule-set" to the ASIC.Though this process was very slow. For example, the Virtual Extensible Local Area Network (VXLAN) protocol [33] was first defined in 2010 by Cisco and VMware, however first hardware with features of this protocol was only available four years later. In those four years the operators had to create many workarounds to achieve the desired results faster.

The delay in incorporating new features in the merchant silicon was causing a lot of issues for many of the network suppliers as it was expensive and a not flexible solution. Meaning that the vendors provided a set of features that were not always lined up with the requirements of the operator. This would cause an operator to adapt and create multiple 'hacks' into implementing required functionality into their network. This, in turn, would increase the complexity of already a complex system. Hence, multiple large scale operators in the last decade have been working towards utilizing whiteboxes or baremetal devices in the large scale datacenters. With the advent of the availability of whiteboxes, this means that operators can now get a device from the ODM either completely blank (baremetal) and install a software of their choice on it, or else get whiteboxes which have an operating system already pre-loaded. Such approach allows them to program the devices with preferred software and be able to view it as whole 'fabric' rather then collection of individual devices. An example of such approach was performed by Google being one of the first adapters of this use case, such as Google B4 network[19], [21].

Many of the large-scale datacenter operators have been working with the SDN approach in their infrastructures. They were the first to separate the control plane from the individual devices and utilize open APIs to the underlying hardware complexity. The SDN controller uses modern server hardware, giving it more flexibility than conventional routers.

Other examples include the Open Compute Project [34], concentrates on creating a flexible hardware that is not vendor locked; and the TelecomInfra project (TIP) [35] is looking at

> " the development and deployment of open, disaggregated, and standards based technology solutions that deliver the high quality connectivity that the world needs – now and in the decades to come."

Along with the physical layer network architecture changes, network control and management tools also require changes. In the next section we discuss traditional network management tools and components.

Since network architectures evolves slowly, the methods to manage them tend to be based on well-known approaches and incorporation of new components with a small number of features added.

Diagram in Figure 2.3 shows the typical network management configuration, where managed devices are routers, or switches etc; management agent is the agent living on each device, communicates to the management server, sending data to it. In this configuration, the physical layer is completely independent from the layers above it and, in general, a device is the one usually monitored rather than an entire network.

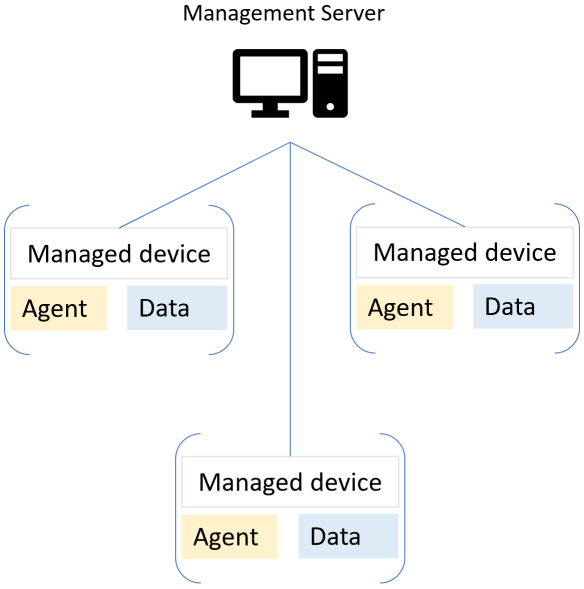

Figure 2.3: Network management configuration.

The main tools used by network operators to manage networks include CLI, SNMP[36] and more recently NETCONF/YANG combination.

CLI, or command line interface, allows a user to enter commands directly on the device. CLI commands are not identical on all devices and would depend on vendor software used. The CLI method of configuration has a limitations such as it is prone to human error and it does not allow for a roll back in case of a mistake. CLI lacks the ability to efficiently scale across multiple devices.

SNMP or Simple Network Monitoring Protocol, is an example of the common monitoring protocols used, so that operators can monitor network health and functionality. SNMP protocol is used to collect information from managed devices, that typically include routers, switches, servers, workstations, printers and others. The architecture of the SNMP protocol includes management interface and agent interface. Agents are allocated on instruments, and are polled by the management interface in the setup interval. This model allows the management system to quickly identify if the instrument is unhealthy or not functioning. However, the limitation is that the traffic will be affected and there is no easy and fast way to reroute the traffic from the broken instrument. This is because individual devices have to be reprogrammed to forward data to a different route. SNMP protocol utilises a MIB, a database of managed objects organized hierarchically that have Object Identifier (OID). The MIB is a database in form of American Standard Code for Information Interchange (ASCII) text file that is shared between an SNMP agent and an SNMP manager. The MIB is a collection of managed objects structured hierarchically in a tree structure. Managed objects come as scalar objects with a single instance or tabular objects with multiple instances (such as a table). Each managed object is given a OID to differentiate it from other objects. The OID is essentially a numerical tag that acts as an address. The format is as follows: 1.3.6.1.1.4.1.2682.1 . In a nutshell, SNMP agents collect performance data from the MIB located on the local device and then sends that data forward to a manager, where it can be viewed remotely with a network monitoring tool. SNMP limitations include the reduced bandwidth of the network, the SNMP protocols not always provide an update on the change, its not polling devices for updates so there is no way to actively discover failed devices; SNMP works only on SNMP-enabled devices.

With the rise of SDN, new methods to manage the network were being discussed. One of such discussions is explained in the document/standard RFC3535 [15], where network operators gather information on challenges and opportunities in standard tools used for managing a network, and what potential improvements could be worked on and appeared as new tools. And out of these discussions is where NETCONF protocol and YANG data modeling appeared, in order to work on the limitations introduced by SNMP, CLI and other tools, and provide more flexibility to the network operators. NETCONF/YANG opens up new opportunities in exploring software defined networks, and in particularly connectivity between Layers 1 to 3. NETCONF/YANG are discussed in depth in section 2.3.

2.3 NETCONF and YANG

Earlier in Section 2.2 we explained the FCAPS model of network management with SNMP for network monitoring. SNMP is now in its fourth decade of deployment for fault detection (via asynchronous "traps") and performance monitoring. Two decades ago the network management community was canvassed to find a replacement for SNMP with the many arguments - and future requirements - outlined in RFC3535 [15]. Many issues were identified including:

- SNMP for configuration proved unwieldy and,
- the management information bases (MIB) which is a represented by a dotted numeric values is typically vendor specific proprietary;
- Security was another concern then - and it is true that this is now largely been addressed by SNMPv3;
- Pull model where transient, aperiodic fault events can remain undetected because of a low polling rate.

As a result, the NETCONF protocol emerged in 2006 as Request For Comment (RFC) RFC4741 [37] with further refinements in RFC6241 [22]. In parallel, a consensus formed around the YANG data modelling language RFC6020 [23] that was updated in RFC7950 [38]. YANG is a standards based data modelling language written in English using Extensible Markup Language (XML). It can model the configuration and state data of network elements; configuration data is read/write (rw), whereas the state data is read-only (ro). YANG is complementary to NETCONF. The NETCONF protocol uses Remote Procedure Call (RPC) to exchange bi-directional information - configuration, state and telemetry data - between a NETCONF client (management plane) and a NETCONF server (network element). The combination of NETCONF and YANG have largely supplanted SNMP and are increasingly being deployed by hyper-scalers and TELCOs alike. The configuration, operational state and telemetry data of a network element can be precisely defined within structured YANG models. The models specify the data that is exchanged between the NETCONF client and server. Some data is essentially static and doesn't change much (e.g. the name of the hardware vendor of a network element); other data is more dynamic (e.g. the 'state' of a network interface, whether "up" or "down").

The architecture of the NETCONF protocol is summarized by the four layers shown in the Figure 2.4 in blue:

- Content layer (YANG defined) – this is human-readable, XML-formatted config-

uration data that reflects the YANG datamodel of the network device element.

- Operations layer (YANG defined) – these are actions to be invoked on the NETCONF server with RPC messages (e.g. get, get-config, edit-config).
- Notification layer - these are the RPC messages that are exchanged between the NETCONF client and NETCONF server (e.g. rpc, rpc-reply).
- Secure Transport layer – SSH Secure Shell (SSH) or Transport Layer Security (TLS) that run over Transmission Control Protocol (TCP) port 830.

NETCONF was designed to provide structure to install, change, delete device configuration. Operations are done with the use of Remote Procedure Call layer. NETCONF clearly defines the difference between operational data, which is read only, and configurational data, which can be changed.

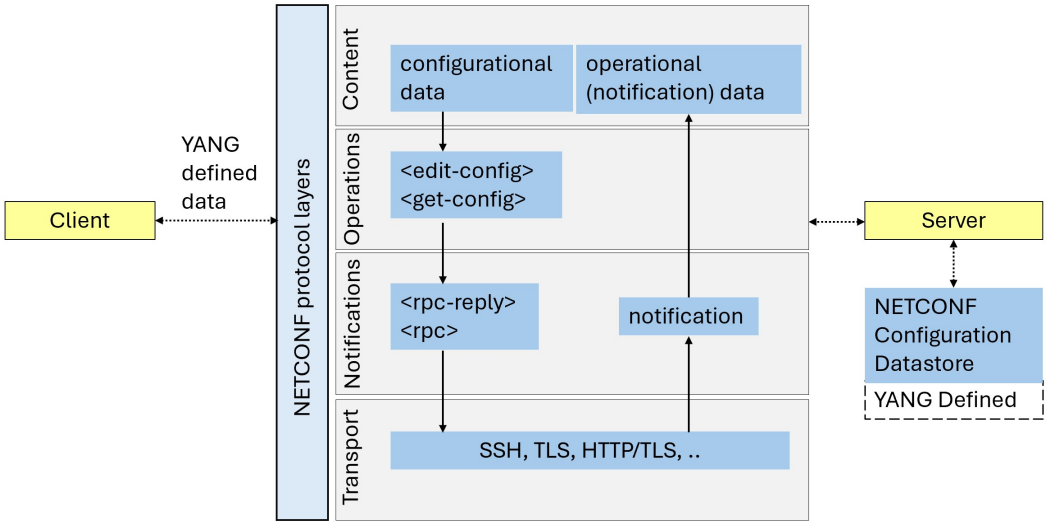

Figure 2.4: NETCONF layers showing communication between client(laptop) and server(ROADM). [5]

NETCONF utilizes concept of a datastore where the data is stored for access and storage. It can be implemented as flash memory, a database, using files or combination of both. There is a configurational datastore that contains initial device configuration, that is required to get a device from its initial default state into a desired operational state. An example of datastore is Sysrepo [39]. It is designed for UNIX/Linux applications and contains both configurational and operational datastore.

A network device element can maintain three configuration datastores that are different logical stores of data that can be managed by network devices using a protocol like NETCONF. These datastores provide an abstraction over where the data is stored and how it is accessed: (a)startup datastore is a base configuration state that is invoked after

booting the network device element; (b)running datastore is the committed, running configuration and state of the network device. It is typically an iteration of the startup configuration; (b)candidate datastore is a pending configuration which has yet to be "committed" to become the running configuration. Not all devices contain all three types of datastores.

NETCONF provides a RPC-based mechanism to facilitate communication between a client and a server. The client can be a script or application typically running as part of a network manager. The server is typically a network device [22]. However, the network manager may not have direct network access to the target network devices. For example, some target network devices may locate in a network with private addresses behind a network address translation device or a firewall. Thus, network manager cannot directly communicate with these target devices.

The NETCONF "hello" message to and from the device is triggered by the establishment of a connection and session to the device. The information for the establishment of the connection and the session to the device requires IP for that specific device and port number. Once the session is established the client and server are exchanging their YANG models to become aware of each others capabilities.

Operations layer is composed of a base protocol commands which are used to send/receive/retrieve configurations to/from NETCONF client. Capabilities are the extra features that can be defined and are optional. NETCONF is made to meet configuration management demands. NETCONF protocol is designed to be transactional, that allows to revert changes in case of any issues.

NETCONF utilizes YANG data modelling language for the modelling of configurational and state data of the instrument. Full syntax definition of the YANG language can be found in [23]. Previously scripting was used to manage multiple devices on the network, however there was no standardized methods therefore it was difficult to manage. YANG data models allow the user to standardize them. Both configuration data and state data can be modelled with YANG, providing a standardized way to describe devices. YANG data model is responsible for the content layer of the NETCONF protocol.

In Chapter 3 we will go into greater details on how we implemented the YANG models and the NETCONF protocol on a specific network element with a P4-programmable network element. Our motivation was to extend the functionality of the P4 programmable network element to include wavelength tune-ability using three discrete optical components. In this endeavour we used open applications that included Netopeer2 (a NETCONF server) and Sysrepo (a YANG-based datastore).

In the remaining section we'll provide some background on the network element and the discrete components that we report in this thesis.

2.4 SDN, its principles and key concepts

Two decades ago the hyper-scalers began to address issues of scale because of the enormous size of their warehouse-scale datacentres. They needed a much simpler and rational means of managing their network estate. So they developed open APIs that replaced SNMP and proprietary MIBs for FCAPS functionality within their networking estate. The hyper-scalers deployed open, disaggregated network hardware elements hosting open software components with open APIs. They have embraced the SDN approach for FCAPS across their infrastructures. Most notable has been Facebook/Meta. They have contributed through the Open Compute Project (OCP) [34] to place the open hardware specifications of the network hardware elements into the public domain. Moreover, they procure this equipment at scale which lowers the barrier-to-entry sufficiently for new, non-proprietary hardware and software vendors to enter the market to challenge the proprietary vendors. Google was another early-adopter of the SDN approach to network management. They openly described their endeavours such as the Google B4 network [19], that viewed their network estate view as an extended, distributed 'network fabric' rather then a collection of individual devices. Microsoft and Amazon Web Services were more opaque in their implementations with little information shared in the public domain.

The fundamental concept of SDN is to separate out the physical hardware from the operating system software of network elements. We take this for granted when it comes to PCs and servers: it is common to have server hardware supplied by an OEM like Dell or HP, but not the Operating System (OS) software. The OS is sourced separately from either Microsoft (e.g. Windows), or Ubuntu (e.g. Linux). But this is not the case with network elements where the physical hardware, the operating system software and the applications (e.g. routing protocols) are tightly integrated and combined into a proprietary product. This very much limits competition and stifles innovation.

The key principles of SDN include:

- Centralized Network Management. Each network element is connected to the centralized SDN controller which polls the element to extract usage requirements at any time, and re-configuring the device as required.
- Network Automation. This allows a network operator to automate tasks and allow services to be enabled on demand.

- Network wide topology view. Centralized control maintains a full view of a network as a whole. The SDN controller serves to retain and store this network topology.

There are also four key characteristics of SDN [40]:

- Flow-based forwarding: Packets can be forwarded based on a flow which equates to a unique combination of packet header fields (e.g. source address, destination IP address and port number) from with-in the link-layer/network-layer/-transport-layer headers. This is an improvement on the traditional routed approach where packets are forwarded based on the IP destination address the network-layer header field.
- Separation of data plane and control plane: The forwarding flow table of each network element is typically computed by its internal CPU. But SDNs separation of the physical hardware from the applications hosted by the operating system software means computations can be offloaded and distributed to independent remote, multicore CPUs within a centralized SDN controller. This allows the match-and-action forwarding logic to be centrally computed and then distributed to the many individual hardware elements.
- Network control functions: The centralized SDN controller (server) also monitors, in real-time, the "State" of the network. It preserves network state information such as flow tables, links, switches and hosts. This information then can be used by the applications for monitoring, programming and controlling underlying network elements.
- A programmable network: The applications running in the SDN controller affect the control plane. They can be used to perform several functions via open APIs. These functions include routing, authentication, load-balancing and end-to-end forwarding optimization across many network elements.

One of the first embodiments of an Open SDN controller was NoX [41]. Since then several alternative, open SDN controllers have emerged that include Ryu [42] , ONOS [43] and OpenDaylight [44]. The OpenDaylight controller and the ONOS controller have found considerable industry support. They are both open-source and are being developed in partnership with the Linux Foundation. The ONOS controller is discussed in more detail in section 2.6.

So far we have dealt with the forwarding plane. But the SDN approach can also encompasses the management and control planes. To appreciate this, Figure 2.5 outlines

a generic SDN architecture that shows the relationships between the forwarding plane, management plane and control plane in RFC 7426[6].

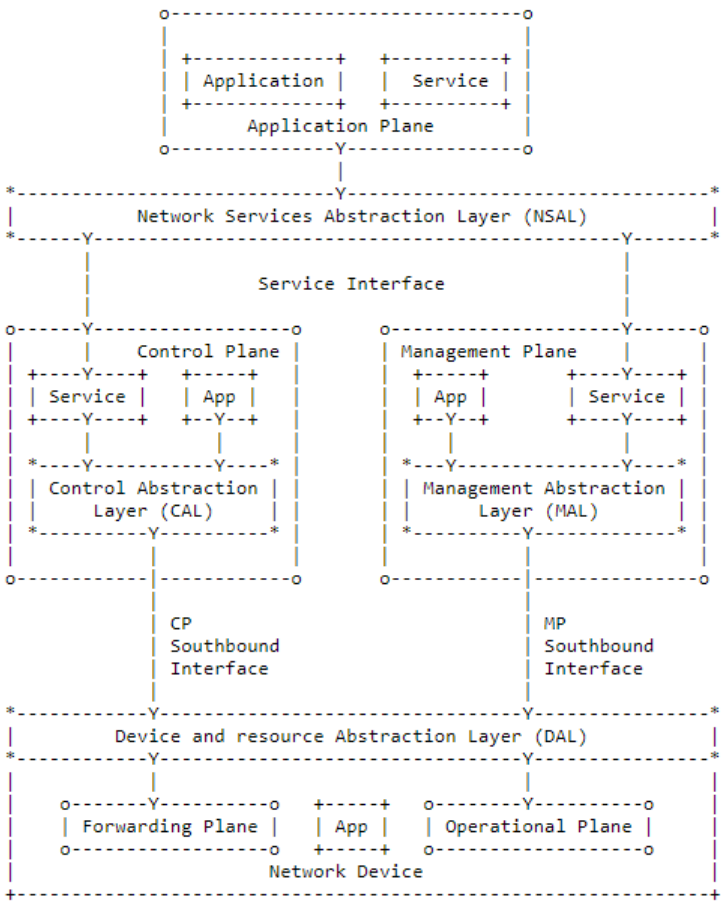

Figure 2.5: SDN layer architecture as illustrated in RFC 7426[6].

RFC 7426[6] defines a very useful taxonomy to distinguish between the different constituent parts that, when combined, embodies SDN.

These parts include the:

- Management plane - which is responsible for monitoring and maintaining network devices (network control functions: external to data-plane switches);
- Control plane - which is responsible for maintaining the forwarding function of the device (flow-based forwarding);
- Application plane - which is a collection of tools and software that can program network behaviour (a programmable network);
- Device and resources - which is the Device Abstraction Layer (DAL) or, in case of physical device, Hardware Abstraction Layer (HAL), is an abstraction for devices capabilities and characteristics;

- Service Interfaces - some of which are northbound and others southbound that pass information between the layers.

In the next section we explore these parts in more detail and stress the importance of well-defined northbound and southbound interfaces.

2.5 South-Bound Interface and North-Bound Interface in SDN

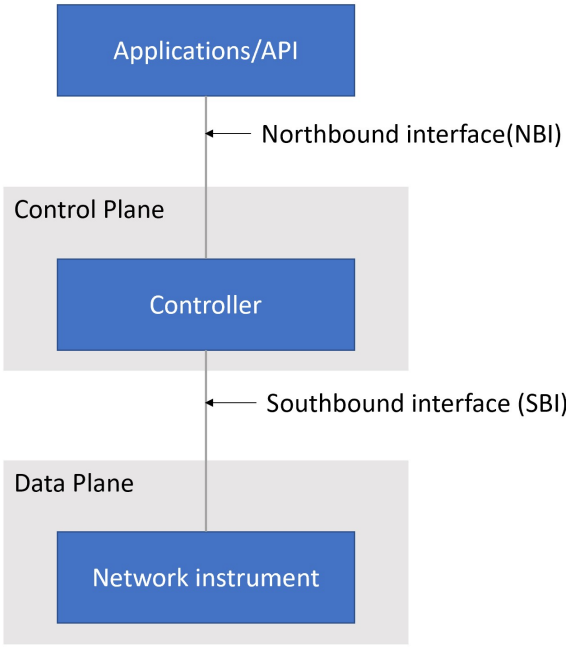

Figure 2.6: SBI-NBI-architecture.

The definition of several APIs is fundamental to the SDN architecture shown earlier in Figure 2.5. An API is a software interface that allows an application to give access to other applications by using pre-defined functions and data structures, using communication structures like illustrated in Figure 2.6. The APIs allow different layers of the architecture to exchange data, or information, between the layers (or planes) of the SDN model presented in Figure 2.5. The workings of the internal logic within each layer are essentially opaque and independent of the higher- or lower-layers. The Northbound Interface (NBI) is the API between one layer and the layer above; and the South Bound Interface (SBI) is the API between one layer and the layer below. Communication requests, flow between layers in both directions: southbound and northbound. Generally NBIs use API protocol libraries to communicate, while SBI APIs will expose the network topology that includes the network elements, the links between these elements, and the state of these links - up or down. The SDN controller has to communicate with network devices in order to program the data plane. This is done through the

southbound interface. This is not a physical interface but a software interface, often an API.

The northbound interface is used to access the SDN controller itself. This allows a network administrator to access the SDN controller to configure it or to retrieve information from it. This could be done through a Graphical User Interface (GUI) or CLI, but it also offers an API which allows other applications to have access to the SDN controller. It can be used to write scripts and automate network administration. For example, to list information from all network devices in the network; to show the status of all physical interfaces in the network; to add a new VLAN on all switches; to show the topology of the entire network; to automatically configure IP addresses, routing, and access-lists when a new virtual machine is created. The SDN controller is at the centre of the architecture. The main controller SBI is openflow that helps populate the flow entries of the forwarding plane of the network element. Openflow also specifies a protocol too to enable the exchange of data using transport layer security (TLS).

2.6 ONOS

The ONOS controller, shown in Figure 2.7, is probably the most widely deployed SDN controller. Internally, ONOS has several modular components that include: a database table that summarizes the connections and topology of the network elements that it controls and manages; and a path computation element that can run a suitable algorithm on the topology to optimize the path(s) between the network elements - subject to one or more constraints. Inter-process communications (IPCs) exchange information between the ONOS processes running on the modular components of the controller. Remote Procedure Calls (RPCs) are used to exchange information between processes that run between the ONOS controller and the processes that run on the network elements. The ONOS controller supports the OpenFlow protocol that forms the control plane SBI that programs the match/action tables of the forwarding plane of the network device ASIC (see Figure 2.5). ONOS also supports the management plane, SBI monitors devices state such as fan speed, cpu usage, interface statistics on sent and dropped packets on the network elements. The ONOS NBI is RESTful and provides API access to the modular components of the controller.

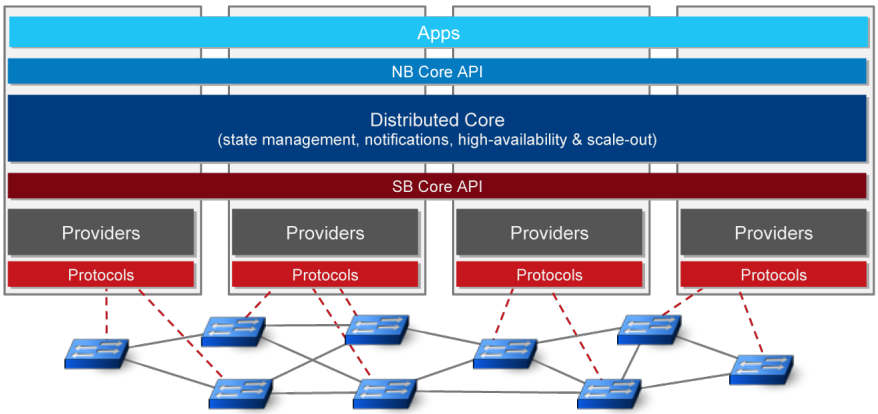

Figure 2.7: Architecture of ONOS controller. [7]

So we can summarize the main components of ONOS as:

- Apps - an operator software applications that can be plugged into the controller to perform certain procedures.

- NorthBound Core API - abstraction layer, exposing set of APIs to access network information, such as topology or devices list and more. Provides range of abstractions to the network via flow objective programming and intent-based programming.

- Distributed core - is responsible for presenting a logically centralized view of network state and logically centralized access to network control functions.

- SouthBound Core API - allows separation from the control side in distributed core layer and allows to directly communicate with the network device elements via the use of providers and protocols.

- Providers and protocols - providers interface with the network via protocol- specific libraries, and with the core via the Provider Service interface. Protocols that can be used include NETCONF, OVSDB, and even a traditional cli.

The flexibility of ONOS allows a user to create 'a provider' and utilize the protocol library to be able to communicate with the network device elements. This ability will form the main theme of the experimental work that will be described later in Chapter 4.

ONOS was designed for network operators by Open Networking Foundation in 2016. Since then ONOS has grown to be used in multitude of projects from virtualization of networks to the air traffic control networks.

2.7 Disaggregation/Stand-alone disaggregated components

Packet-level statistical multiplexing is a foundational principle of packet-switched networking, enabling multiple flows to share communication links dynamically at the level of individual packets. In contrast to circuit-switched systems, which reserve capacity regardless of demand, statistical multiplexing exploits the bursty and stochastic nature of traffic by transmitting packets opportunistically as they arrive, thereby improving link utilization and efficiency. This mechanism underpins today's Internet backbones, data center interconnects, mobile and 5G networks, and virtual private network (VPN) services, all of which depend on efficient aggregation of diverse and large-scale traffic. While inherently efficient, statistical multiplexing is best-effort in nature, creating challenges in guaranteeing quality-of-service (QoS) in heterogeneous environments. Emerging paradigms such as Software-Defined Networking (SDN) mitigate this limitation by introducing centralized programmability and fine-grained traffic control, enabling dynamic flow management, QoS enforcement, and network slicing across multiplexed streams.

Statistical multiplexing has long been considered a necessary element of network architecture to ensure performance, availability and capacity. Statistical multiplexing of the resources, used to be a common method to supply the demand and ensure enough resources available for the rapid jumps in bandwidth [45]. It is an expensive method to ensure availability of service and performance. This method increases the number of managed devices on the network that require constant monitoring.

The increasing adoption of disaggregated network devices further influences how statistical multiplexing is implemented and managed. On the one hand, disaggregation promotes cost efficiency, scalability, and rapid innovation by decoupling control and data planes, which aligns naturally with SDN-driven optimization of multiplexed traffic. On the other hand, the heterogeneous performance characteristics of disaggregated devices can introduce complexity in achieving consistent multiplexing efficiency and end-to-end QoS, as differences in buffer sizes, scheduling algorithms, and hardware pipelines may affect fairness and latency. Consequently, while statistical multiplexing will remain indispensable in future networks, its effectiveness will increasingly depend on how disaggregated infrastructures are orchestrated in accord with SDN to balance efficiency, performance consistency, and service guarantees.

Disaggregating the network optical devices is the term used nowadays by multiple parties to identify the process of separating optical line systems and complex optical devices. For example, disaggregating a transponders into separate components (such

as laser, attenuators, power meters etc.), each connected to the global system topology, which allows to monitor and troubleshoot each component individually. This method allows a network operator to mix and match devices based on the needs and requirements of their network. Vendor lock-in has been an issue for the network operators for many years as the device vendors were essentially the ones dictating what kind of features will be in the device, rather than network operators specifying what kind of feature-set they would like to implement in their network. Disaggregation of a network can be with regard to separating the functionality from the hardware itself to the software. This idea is described in [46].

Disaggregation of the optical systems allows the components to be separately sourced, resulting in large potential cost savings and increased scalability. This method allows any complex device to be broken up into separate components that can be configured and managed based on workload and demand. This approach has been widely supported by TELCOs and the Open and Disaggregated Transport Network initiative have been setup [47]. This group is researching disaggregated DWDM systems, including but not limited to transponders and Open Line Systems, amplifiers, multiplexers, all-optical switches and Reconfigurable Optical Add Drop Multiplexer (ROADM). Those are in the physical layer and are optical fibre based devices.

2.8 SDN in the field of optical networks

Optical network provides better transmission capacity than networking in the electrical domain. The challenges with managing optical domain include optical path computations, noise level management, power management etc. SDN establishment in the current networks is ongoing and many different works have been performed towards implementing different components into the network ecosystem.

SDN methods have been well established in packet domain, and its presence in the optical domain is growing. For example OpenRoadm multi-source agreement has been established to create a standard for configuring ROADMs with a strong emphasis on interoperability in multi-vendor environments [48]. During the experimental phase of this thesis, OpenRoadm was only starting, with a few issues that were resolved with Open Networking Foundation.

Certain parts of optical networking have been investigated in depth, those include path computation elements, point-to-point routing and wavelength assignment prior to provisioning, however the operations such as setting up lightpaths, tuning of amplifiers, switches and power settings have not been well documented or is just starting to appear

by the start of this research.

Many experiments were reported with the use of SDN tools and optical network components such one experiment is described in [49], where the team used ONOS controller to connect to and communicate with ROADMs from different vendors utilizing NETCONF protocol. Another work where SDN was implemented in WAN network is described in [40].

Similar approach is being investigated by the Open Disagregated Transport Network (ODTN) project, the goal of which is to use disaggregated components in optical domain to mix and match the devices and be able to reconfigure them using the open tools such as NETCONF.[47]

In [50] various operators express their interest in SDN and its benefits it brings to the way optical network is being run, and to include whitebox devices in the network.

2.9 Summary

The challenge in today's networking is to flexibly manage it end-to-end, both electrical and optical domains, expanding the capacity and bandwidth to satisfy the constant increase in data demand. The rise of SDN allowed network to be researched and improved using different software tools, it created an opportunity to manage network centrally and allowed to separate control plane from operations. SDN allows to view a full topology without dividing it into separate domains. The challenges and further opportunities of SDN networks are discussed in Chapter 3.

Chapter 3

Network Management

3.1 Centralized vs distributed network control

Network management is a set of procedures aimed towards reducing network downtime, improving availability, testing and analyzing the state of the network. The majority of functions on the physical layer network are performed on a device-by-device basis, for example, setting the wavelength of a particular transceiver is assigned as a set-and-forget status in current networks, being difficult to adapt or change it from management level. This approach is challenging, causing an inefficient resource allocation, increasing network scaling issues, unpredictable failures, poor quality of service etc. The earlier network management systems relied on physical connections to a mainframe computer. Later, this evolved into connection to many computers/servers, so the evolution of network management was not as fast as the growth of the network size itself[51].

Due to the fast growth of the network size, the operators got into a state where the IP and the optical layers do not share a common control plane, where the operator is forced to maintain two disparate systems. A single proprietary system locks the operator into a single vendor, impacting the carrier's freedom for upgrades or implementing the best cost effective solution, without affecting the entire network configuration.

The main network layer functions can be divided into three categories, which is equivalent to Figure 2.5 from section 2.4: SDN, principles and key concepts:

- Forwarding (data plane) - moving packets, including devices wavelength assignment.
- Control (control plane) - best route path allocation and calculation.
- Maintenance (operations plane) - maintaining devices state, such as cpu usage,

temperatures and fan optimization etc.

By splitting network functions into the data, control and operations plane it provides a better overview and alignment of the entire network architecture, as suggested in context of SDN in RFC7426[6].

3.1.1 Forwarding plane

Forwarding plane (data plane) is responsible for receiving the packets on input interfaces, analyzing them to determine where to forward to, and what path to use, or else if the packet has to be dropped. In, current networks the forwarding function is built in the device itself (e.g. switch, router). However, typically, devices share the resources for other tasks, such as calculating routing tables. Forwarding of a packet is complex procedure, as depending on security policies, packets can specify its destination or require encapsulation of certain protocols or encryption, which require time and resources. An SDN approach allows the forwarding function to remain within the device, while other functions, such as calculating of the routing table, can be performed by the central controller. The central controller then optimizes the resource allocation and release the devices for packet forwarding.

3.1.2 Control plane

Current distributed IP networks utilize control plane protocols such as Open Shortest Path First (OSPF), Border Gateway Protocol (BGP) and Routing Information Protocol (RIP) for both control and forwarding of traffic on the network. Those are dynamic routing protocols that have functions setup on each individual device. SDN separates control plane (routing) from individual network device to a specific controller for network management (i.e. from many individual devices into one device that takes care of many). An example of a comparison of the traditional models versus SDN proposed model is shown in Figure 3.1.

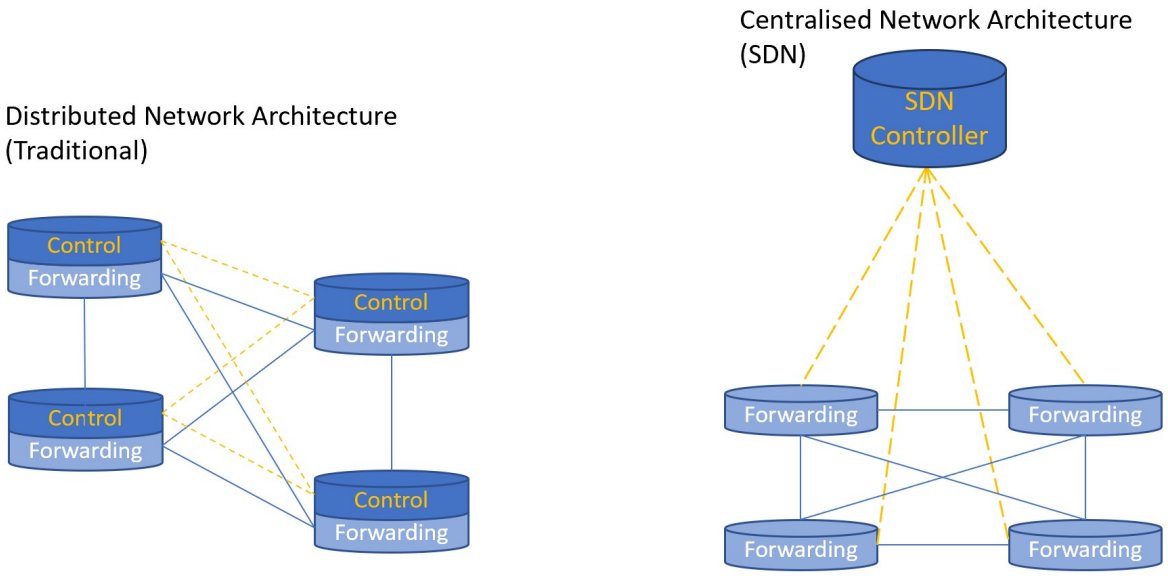

Figure 3.1: Traditional network management architecture vs SDN proposed.

In a distributed network architecture, configuration of settings is performed on each device, which in large scale, management on any update can be difficult and time consuming. As an example, the routing table can be calculated per device and then shared between routers, each device has to complete this operation at certain intervals to ensure up-to-date and consistent routing table is set. It has to include updates such as device "unreachable", meaning the route is not possible any more. Within a centralized architecture this operation can be performed by a central controller and then forwarded to the routers. The central controller would have a better knowledge of the available devices in the network and hence better changes for opening further available paths.

3.1.3 Management plane

Management plane is referred to administrative tasks, such as those related to the device's configuration, monitoring, and maintenance. This is typically the interface through which network administrators or operators interact with the device to configure its settings, update firmware, and monitor its status. This includes configuring devices themselves to be compliant with protocols used, for example SSH/NETCONF/SNMP.

3.1.4 Operations plane

The Operations plane of the device is responsible for overall functioning of the device itself, such as monitoring temperature levels, fan speeds and other components, en-

suring that the device can work as expected. Management plane tools should query the devices for the state of components (or sensors), for example temperature or fan speed. With the help of centralized controller it should be possible to facilitate device maintenance such as questioning device temperatures and fan speeds to ensure the best settings are configured to benefit the device effectiveness and avoid failures caused by overheating. Ability of the sensors components to be controlled by software provide the desired flexibility in terms of device operations, as operators should be able to modify device settings from central point of control. This means accessing the devices registers for monitoring and control.

The SDN approach proposes a centralized control to forwarding and routing decisions of the network. Maintenance of the devices can be accomplished with the help of central controller and network wide topology view. Devices can be pinged/contacted during specified intervals to confirm they are up and available to work. Devices will receive a "heartbeat" signal from the controller and respond to it. If not responding for a set timeout limit, the device is set as unavailable and a new routing table have to be computed, causing a full reconfiguration of all other devices in the network.

The main issues faced by network managers when an issue arises/service is closed are the ability to find the root problem and the time required to fix it, in order to restore services. A physical network issue (i.e. cable issues, device problems), can be easily identified, however, issues with protocols deployed on devices (and other software components) can be cumbersome to be identified in large scale network. Device maintenance requires constant state information exchange to ensure optimal health and temperatures to avoid overheating. These issues are amplified as network sizes and demand increases. Separating distributed routing and packet forwarding allows one to move from per-device control to centralized control, as proposed by SDN.

The main difference that makes SDN so much more appealing to network operators is the ability to write an application that uses REST-full APIs, on top the SDN controller, and utilize NETCONF or other protocols to deliver the application functionality. It is the ability to write an application on top of the SDN controller is what makes it so much different then CLI. The CLI approach requires manual intervention for configuration changes, which can lead to delays in adapting to network changes and increased operational overhead. Where as software approach allows to dynamically adjust network configurations and routing based on real-time traffic and application needs.

3.1.5 Open Central Controller

In many cases, network growth and the complexity of network architectures are poorly understood by the network managers. They require a fully exposed multi-layer topology from Layer 1 to 3. Hence, the centralized controllers should be able to abstract such a network topology for these different layers.

The ability to scale, test and reconfigure the network from a central point of control brings great benefits. Configuring devices via central point of control can potentially allow the creation of a fully automated network, as well as bringing the possibility to perform constant statistics and telemetry polling. Those functions can be achieved by using NETCONF protocol, for example. The NETCONF protocol provides capabilities of rolling out a configuration edit on multiple devices at once. This is so called "transaction capability", which allows to apply configurations to multiple devices at once and in case of a failure of one device, rollback to the previous working state. ONOS is a potential controller that can exploit such properties.

Arguably, central point of control is seen as a point of failure for a network. To avoid this, ONOS is using a distributed controller technique in which a cluster of controllers are deployed and interconnected together; they share information and communicating using a "Gossip" protocol.

3.1.6 Disaggregated devices

Device disaggregation and network disaggregation are complementary architectural principles that together enable fully open and programmable networks. Device disaggregation breaks apart the hardware and software components within an individual network element, and could allow operators to select merchant-silicon forwarding hardware, independent operating systems, and pluggable optical modules rather than relying on vertically-integrated vendor platforms. Network disaggregation, enabled by SDN, extends this principle across the entire network by separating the centralized control plane from distributed data-plane devices, enabling consistent provisioning and policy control over multi-vendor infrastructure. When combined, these approaches allow each device to be independently optimized and upgraded while still being orchestrated under a unified control framework. This creates an environment where operators can introduce innovation at both the device level (e.g., switching ASICs, optical modules, and control software) and the network level (e.g., centralized traffic engineering, automation, and open APIs), while maintaining interoperability and operational coherence. In essence, device disaggregation provides modular building blocks, and SDN-driven network disaggregation provides the system-wide intelligence to coor-

dinate them.

Centralized control of the physical layer (Layer 1) has been researched since 2000s, however, with limited capabilities.[13], [52].

Optical devices such as ROADMs and routers are often still operated as coarse-grained units. While modern systems increasingly offer open APIs and multiple parallel control sessions, these interfaces manipulate vendor-defined abstractions — for example, wavelength switching and power management are typically applied per channel or band, and router programmability remains constrained to standard packet forwarding and queueing functions. The limitation therefore lies not in the absence of programmability, but in the lack of direct fine-grained access to physical-layer parameters (e.g., per-channel laser tuning or modulation adaptation). Enabling deeper device disaggregation could provide such level of control, offering greater flexibility and more network management adaptability.

This thesis shows how truly disaggregated devices (such as a power monitor/ power meter, laser etc) and a ROADM can be centrally and simultaneously controlled. Disaggregating those devices enables further controls, creating an opportunity to explore automation of the network. For example, being able to control the components depending on the traffic load allows us to reduce the power consumption, such as when the traffic load is small and fan rotations can be reduced, or else increase dynamically the channel bandwidth to accommodate to larger loads. This is the sort of flexibility that service providers are looking for to create a fully programmable network.

3.2 Management of disaggregated devices

3.2.1 GPIB connection and Linux interface

In this thesis the devices used have common physical connection such as General Purpose Interface Bus (GPIB)[53] or USB. Many of the legacy devices are only capable to communicate with vendor provided software and have limited abilities to be interconnected or managed by other software. Here, devices used were ILX 8210H power meter, Agilent laser N7741A and 8164b Keysight Lightwave Measurement System (attenuator), supported by GPIB connection and protocol, as shown in Figure 3.2. These are typical units that are minituarized and inserted into transcievers. The goal here was to show the potential to include new specific adaptive functionalities to the central control to facilitate better network health state awareness, and monitoring of configuration status.

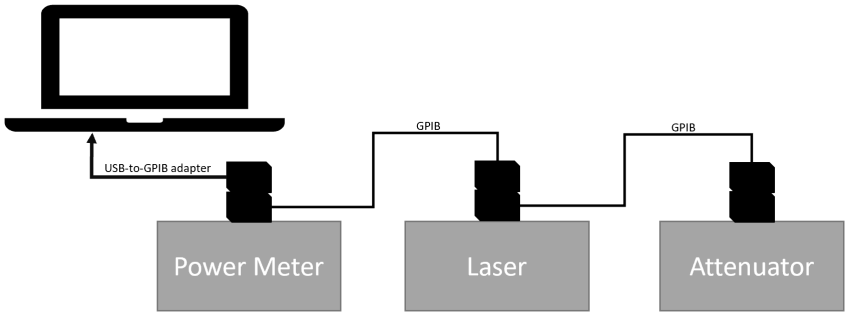

Figure 3.2: Typical GPIB devices link set up with a serial topology, the architecture used in this thesis.

The devices used in the thesis are manufactured by different vendors. Typically, each vendor provide a software that will allow an operator to connect to device, set and reset it, as well as querying for the devices health.

In order to control a physical device, one must physically connect it to a computer or server (through a cable/WIFI/Ethernet etc.), requiring a software driver to enable communications.

Figure 3.3 is showing a USB-to-GPIB converter device used as a connection adapter between the devices and a laptop. The connection protocol GPIB together with the GPIB-to-USB converter, permits the physical communication with one device at a time. A GPIB address number must be assigned to device via the device's driver in order to control each individual device, otherwise it cannot identify the devices connected.

Figure 3.3: Agilent 82357a USB-to-GPIB converter used in this thesis.

We used publicly available drivers that were compatible with the Linux operating system (e.g Ubuntu 16.04), using drivers such as "linux-gpib", which were expanded to include new functions specific to each device.

The first step was to find the correct software driver for each of the devices. GPIB is a general purpose interface bus that allows one to connect up to 20 devices in a chain, individually communicating with each device defined by a specific set of commands. The commands set is divided in two parts: first is device specific depending on the

device capabilities, and second is standard commands defined by GPIB, those include read, write, open and others. The second standard set of commands can be varied in different devices in terms of command syntax.

Although finding drivers was straightforward, setting up the connections within a configuration was a lot harder. The first issue we encountered was that the driver was over five years old, and it wasn't compatible with the current kernel versions running on Ubuntu 16.04, which was the latest version of operating system used at the time for this thesis. The solution to this problem was to install an older versions of the kernel.

Once the connection to the devices was established the next step was to ensure that devices could be configured via such a method. The GPIB protocol provided two test programs for communications. The specific tests were useful in order to enable us to create our own line code, utilizing GPIB-library functions in "C", enabling communications and reconfiguration of each device within a C-programme code. The "C" program was required, as it was an essential step in order to write a device specific code or make sure that the syntax of the commands is specific to each of the devices. This step allows us to be able to reconfigure the device dynamically.

GPIB protocol has been widely used for instruments since 1978 and has a set of defined functions an operator can use to manage the devices. The details of the protocol are described in IEEE-488 standard document[54]. The functions of the protocol have been described in a code libraries for both Windows and Linux operating systems. The Linux variant of the GPIB package is available to download and use from a free and open source code repository[55]. This package contains drivers and modules that allow to enable specific set of instruments working with Linux kernel.

The management laptop used in the experiment run a Windows Operating system, however the software tools used in this work for the experiment were designed for a Linux/Unix based operating systems. This requirement has resulted in the use of a tool Virtual Box software, which allowed to setup and run virtual machines with Linux Ubuntu v16.04 operating system. All further steps performed and software programs written were done so for on a Ubuntu operating system, unless specified otherwise.

Linux-GPIB package allowed to reuse defined functions, in order to set up a communication between the legacy instruments and a laptop. As was specified in Chapter 3, section 3.2 the package was used only to enable communication with the GPIB-to-USB converter Agilent 8235B which was used as a communication bus.

Connecting the Agilent 8235B GPIB-to-USB converter to the laptop involved several steps to properly mount the physical device onto the operating system, as per steps,

shown in Figure 3.4.

```
sudo depmod -a      #scan through modules
sudo modprobe gpib_common
sudo modprobe agilent_82357a
#plug in usb to connect gpib to laptop
sudo lsusb
sudo fxload -D /dev/bus/usb/002/003 -I gpib_firmware-2008-08-10/agilent_82357a/82357a_fw.hex
sudo chmod 666 /dev/gpib0
check if usb is connected, if not connect (virtual box setting)
lsusb
sudo gpib_config
sudo ibtest
```

Figure 3.4: Commands used to mount the device to the Ubuntu virtual machine.

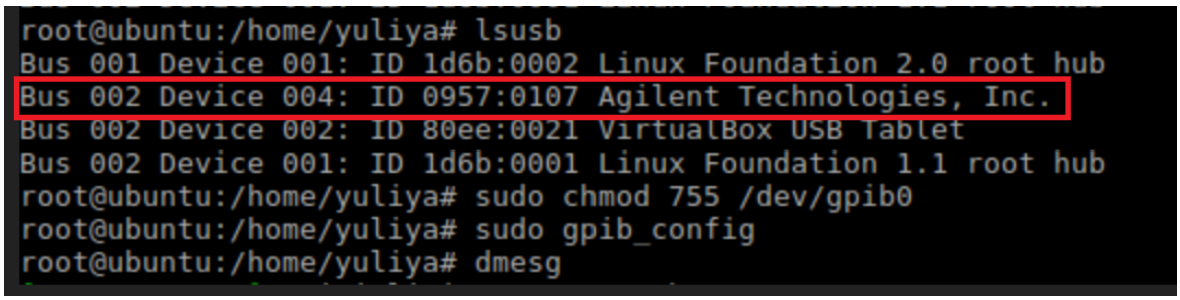

Figure 3.5: "lsusb" command output showing details of device connected to the Ubuntu virtual machine, in this case, the attenuator labelled Agilent technologies Inc.

Figure 3.5 shows the device is physically connected and recognized by the operating system. The steps include loading specific modules into the Ubuntu system, ensuring the device was properly mounted, and verifying that the correct permissions were set. These actions facilitated the physical connection of the USB to the laptop and its subsequent mounting to the virtual machine via VirtualBox tools. The "depmod" command scans the modules in the directory "/lib/modules/version" and identifies dependencies, which are stored in the "modules.dep" file. The "modprobe" command then adds the necessary modules to the Linux kernel, while the "fxload" program is used to download firmware to USB devices. Finally, the "lsusb" command scans and lists all connected USB devices.

The package also creates a file in "/etc/gpib.conf" which needs to be edited to specify the connector that is being used. The contents of "gpib.conf" file with parts that needs to be edited to specify which device we want to connect to are shown in Figure 3.6, the setting needed is primary address of interface.

```
interface {
        minor = 0       /* board index, minor = 0 uses /dev/gpib0, minor = 1 uses /dev/gpib1, etc. */
        board_type = "agilent_82357a"   /* type of interface board being used */
        name = "violet" /* optional name, allows you to get a board descriptor using ibfind() */
        pad = 0 /* primary address of interface             */
        sad = 0 /* secondary address of interface           */
        timeout = T3s   /* timeout for commands */
        eos = 0x0a      /* EOS Byte, 0xa is newline and 0xd is carriage return */
        set-reos = yes  /* Terminate read if EOS */
        set-bin = no    /* Compare EOS 8-bit */
        set-xeos = no   /* Assert EOI whenever EOS byte is sent */
        set-eot = yes   /* Assert EOI with last byte on writes */
        master = yes    /* interface board is system controller */
```

Figure 3.6: Contents of gpib.conf file that needed to be edited to connect to Agilent 8235a usb-to-gpib adapter.

Once these steps were completed, a connection to the device could be established, allowing communication through GPIB package commands such as “ibtest”. An example of connecting to the GPIB device using the “ibtest” command is shown in Figure 3.7

It is important to notice that the legacy instruments involved in our setup were never originally designed to be managed via modern open source software tools such as Sysrepo, YANG and NETCONF protocol. This was achieved by writing a software driver in “C” for each of the physical devices to enable control using Sysrepo and NETCONF.

```
root@ubuntu:/home/yuliya# gpib_config
root@ubuntu:/home/yuliya# ibtest
Do you wish to open a (d)evice or an interface (b)oard?
        (you probably want to open a device): d
enter primary gpib address for device you wish to open [0-30]: 1
trying to open pad = 1 on /dev/gpib0 ...
You can:
        w(a)it for an event
        write (c)ommand bytes to bus (system controller only)
        send (d)evice clear (device only)
        change remote (e)nable line (system controller only)
        (g)o to standby (release ATN line, system controller only)
        send (i)nterface clear (system controller only)
        ta(k)e control (assert ATN line, system controller only)
        get bus (l)ine status (board only)
        go to local (m)ode
        change end (o)f transmission configuration
        (q)uit
        (r)ead string
        perform (s)erial poll (device only)
        change (t)imeout on io operations
        request ser(v)ice (board only)
        (w)rite data string
: w
enter a string to send to your device: *IDN?
sending string: *IDN?

gpib status is:
ibsta = 0xc100  < ERR TIMO CMPL >
iberr= 14
EBUS 14: Bus error

ibcnt = 0
```

Figure 3.7: Connecting to a device via GPIB package, using “ibtest” command. The connection was established successfully but the command to identify the device has failed.

A key difference between NETCONF and other management protocols is that NETCONF is built around the idea of a transaction-based configuration model. The NETCONF specification provides for some optional device capabilities aimed at assisting operators with the life cycle of configuring a network device, such as rolling back a configuration upon an error. Unfortunately, not all network devices support such capabilities, but the protocol was built to make it easier to discover what kind of capabilities a network device can support.

The data that is contained within many NETCONF operations is not within the scope of the protocol itself. For the most part, NETCONF treats the data within the “config” element as opaque data, and makes the assumption that the endpoints are able to make sense of this data. That is where YANG comes into the picture. It is describing the device settings and features, configurable and non-configurable data, that is then used inside the “config” element of the protocol. Another reason to use NETCONF and YANG is the ability to be autonomous from a vendor dependency. Meaning that we

do not depend on vendor proprietary management tools.

As the experiment includes legacy devices that are GPIB protocol compatible and not NETCONF and YANG compatible during the research for this thesis we had to investigate and design a potential method to adapt instruments to understand NETCONF and YANG. For this reason, the Sysrepo tool was introduced. Sysrepo is defined as a YANG-based configuration and operational state datastore for Unix/Linux applications. Applications can use Sysrepo to store their configuration modeled by provided YANG model instead of using flat configuration files for example. Sysrepo can ensure data consistency of the data stored in the datastore and enforce data constraints defined by YANG model[39].

The high level architecture of the Sysrepo and how it fits in with legacy devices is shown in Figure 3.8. Sysrepo is a software component in between devices and a controller. It composed of a plugin that is used by devices to communicate with the Sysrepo engine and library.This setup allows to communicate between a laptop and three individual disaggregated hardware instruments without using a vendor provided software. The goal of this stage was to set and reset instrument settings using NETCONF, YANG and individual device drivers using GPIB. When the connection was successfully established, and the devices could be reconfigured using GPIB protocol via Ubuntu OS, continuing to working on designing device drivers to communicate with ONOS.

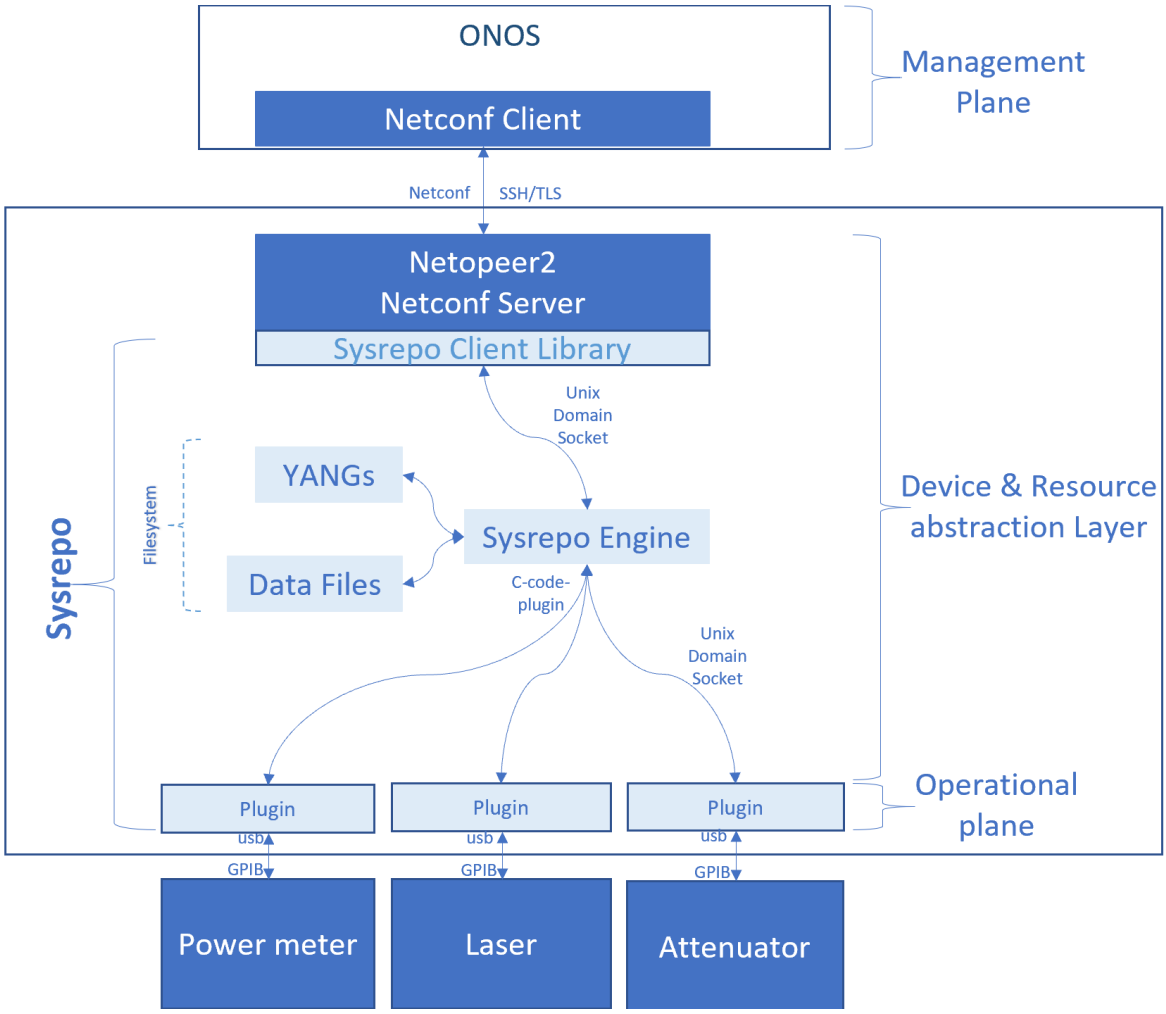

Figure 3.8: The connection schematic between legacy components, Sysrepo and Netopeer2

3.2.2 Control plane of disaggregated devices

In the context of the control plane for disaggregated devices, we would refer to the ability of the device to perform its allocated function such as (a) an attenuator to provide an optical attenuation of the light power within the optical path; and (b) an optical power meter to measure the power at the ROADM line output of the optical fibre link. This will become clearer in Chapter 5 with the full experimental setup and results. Those functions are typically turned on by pressing buttons on the devices themselves, or else utilizing the vendor provided software. Such devices do not have an agent that constantly queries information on the health and functionality for each. In this thesis, such functionality was here enabled by extending the YANG models capabilities and creating drivers to query the devices functions.

3.2.3 Operations plane of disaggregated devices

The operations plane is referring to the device physical health such as state on/off, overheating, fibres connected etc. In terms of the operations plane of the devices, the devices are equipped with the warning mechanisms that would issue an error if device is not in the desired working state. This information is typically displayed on the device management software system that is provided by the vendor. One of the use cases of this thesis, as example, was to use the operations plane to switch a laser on and off; or to change/reduce the output power of the laser by increasing the attenuation. This was achieved by including such functions in the devices' YANG model and driver. To the best of our knowledge, the proposal shown here in this thesis was the first demonstration of the use of SDN and YANG models to disaggregated, legacy devices as such.

Management plane of the disaggregated devices would be tightly coupled with operations plane, and following the same example in the thesis, to measure a power or to attenuate.

NETCONF protocol, as in section 2.3, was the chosen protocol to configure the devices. It addresses shortcomings of existing practices and protocols, as it allows configuration of network devices through standard APIs. The other reason this was a protocol of choice is because the components needed to emulate a YANG datastore and NETCONF server, that was available at the time of this work being done was very limited. Combination of Sysrepo and Netopeer2 were the most advanced tools available for such use and were designed as open source tools compatible with Linux operating system as those tools were designed specifically for use with NETCONF and YANG. Sysrepo is a key tool to store the bespoke YANG models, as the legacy devices did not have the ability to self-store them. Netopeer2 was then used to emulate a NETCONF server for this case. More details in the next section.

3.2.4 Implementations of NETCONF SBI for disaggregated devices

NETCONF is a connection oriented client/server protocol which means it has to have both (client and server) in order to establish communication between them. NETCONF was designed for transporting data using SSH/TLS protocol [37], as described in section 2.3, and in detail:

- NETCONF server is the agent that resides inside the network element and accepts incoming requests from NETCONF clients; acts upon those requests; and provides a reply.

- NETCONF client is related to the management stations or the client application-s/interfaces used to access and manage the NETCONF capabilities of the device.

In this thesis, the architecture of communication setup between the client and server is shown in Figure 3.9. NETCONF client is part of the ONOS controller, that consists of ONOS internal NETCONF driver and a driver written for each of the devices the connection is being setup to. This was written in java programming language as this is the language the ONOS controller is written in.

NETCONF server comprises of a combination of components such as Sysrepo YANG datastore and Netopeer2 server. Netopeer2 is a set of tools that utilise "libnetconf" library to connect to NETCONF devices, and it is using Sysrepo as its datastore for YANG models.

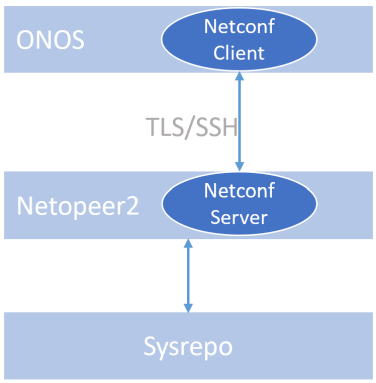

Figure 3.9: Network client and server communication within the proposed architecture for this thesis.

Setting NETCONF for disaggregated devices involved setting up Netopeer2 server and ensuring it was successfully connected with Sysrepo datastore. An example of the setup is shown in Figure 3.10. The image illustrates five different terminal windows, each running separate processes. As is shown in the Figure 3.10, terminal 1 is a sysrepo-daemon, terminal 2 is sysrepo-plugind, terminal 3 is netopeer2-server, terminal 4 is netopeer2-cli and terminal 5 is ONOS cli.

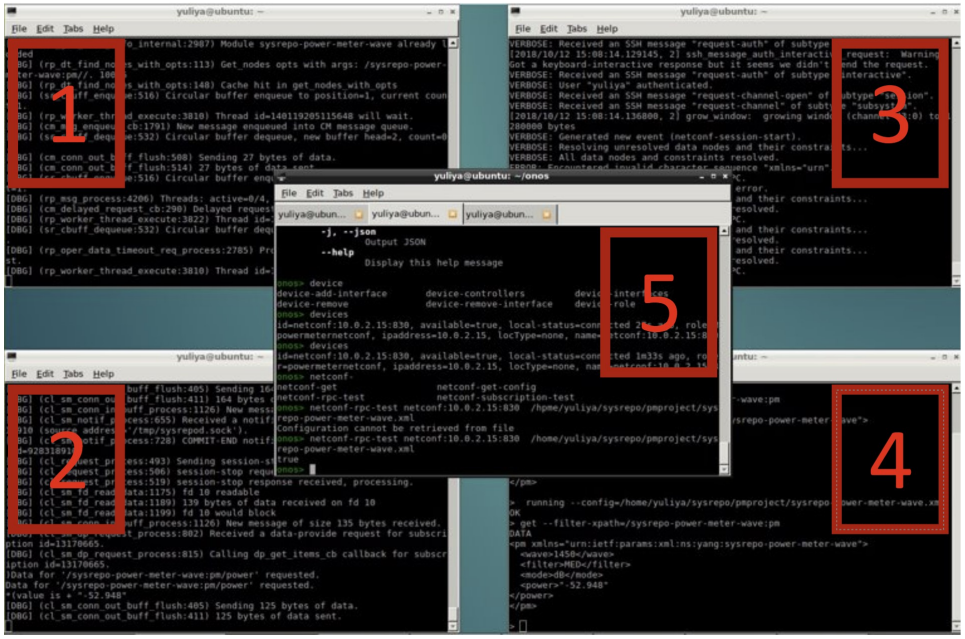

Figure 3.10: Screenshot showing all components started, such as sysrepo-deamon(1), sysrepo-plugind(2), netopeer2-server(3), netopeer2-cli(4) and onos(5), connected with presentation of power meter settings in netopeer2-cli(4).

Once this step was accomplished, the Netopeer2-cli was utilized to ensure that the connections to devices were established; and the cli allowed us to reconfigure these devices. It is important to notice that the usage of Syrepo and Netopeer2 has been shown for other software tools such as DHCP server [56] [24], however, our approach is the first demonstration of connecting disaggregated devices with these tools together with an ONOS controller.

As NETCONF is just a protocol to transport data, the main consideration was to ensure it can establish connections and communicate between the server and client. The steps taken to establish connection are detailed below.

The first step in this experiment involved connecting all devices physically and ensuring that they can communicate with management laptop. This was completed with help of GPIB protocol and GPIB-to-USB adapter as explained in section 3.2.

The approach and steps described in this thesis were followed in this order to ensure each device can communicate with the software used, and allow to test the setup at each step, avoiding errors and misconfigurations.

To setup Netopeer2, Sysrepo had to be successfully installed and compiled. The detailed steps to install and compile are available on official Github repository [39] and in the Sysrepo documentation repository [8]. Official documentation includes descriptions and explanations of features and functions available in the Sysrepo and how they can

be utilised. Sysrepo also provides a guide for plugin development [57], which guides a user how to integrate and work with Sysrepo.

Once Sysrepo is up and running, the next step is to setup Netopeer2 server. The steps to install and compile Netopeer2 is provided in the Github repository [58]. The important step for Netopeer2 to work, was creating an SSH key on the system for secure setup of an SSH session between Sysrepo and Netopeer2.

To establish connection between a client and server in this thesis, ONOS had to be notified of a device available to connect to. ONOS contained a file in the directory "ONOS-ROOT/tools/test/configs/netconf-cfg.json", this file contains device specific information such as IP address and port number, user credentials, protocol used and a driver name that were utilized during connection for identification purposes.

Communication between Netopeer2 and Sysrepo is built in by design. Sysrepo provides APIs for applications to read updated attributes received from NETCONF client to server. The key is to start the processes in a specific order, to allow each component to establish a connection to each other. This follows as 1.sysrepo-deamon; 2.sysrepo-plugind; 3.netopeer2-server; 4.netopeer2-cli.

Using Netopeer2-cli, we then established a NETCONF session by running "connect" command similar to example shown below, with details of the device we are trying to connect to. Where "connect" is the command to be run, "–login" flag was used to login with username "user1", password was prompted by the client and connect to the server which in this case was "localhost" indicating that the server was running on the same machine as the client.

```
connect --login user1 localhost
```

At this point the software tools Netopeer2 and Sysrepo were setup, and NETCONF session was established to a device. Next step was to configure the device, and for this bespoke YANG models needed to be created for each connected device.

3.2.5 Implementations of YANG data models for disaggregated devices

YANG stands for Yet Another Next Generation, YANG is a data modeling language used to model configuration data, state data, Remote Procedure Calls, and notifications for network management protocols[38].

YANG data model is hierarchical structure of a data, that is divided into modules or sections. The main sections of the YANG data model are header statements, revision statements, and definition statements. The header statements section describe the

model and give information about the section itself, while the revision statements give information about the history of the model, and the definition statements are the body of the section where the data model is defined. The header statement and revision statement, were the least complicated part, as one can define the title of the module, version and description. Since this was a first YANG model written for the legacy devices, the name, version and description was given based on device specifications and datasheets..

The definition statements can define different types of data such as integer or a string, specify the length and if it is a write only or read-write data type. This part was more complex as it involved testing the devices functions and defining what types of data were used, for example write-only or read-write etc.

For example, a power meter measures the optical power at the output of an optical fibre (or optical transmission line). Power is represented in its units in 'mW' or in 'dBm', so a user have to define both or the preferred unit used. The power meter data can only be read, but the measuring units can be changed (written to).

YANG defines configurational data and state data. Configurational data is read-write type; state data is read-only type.

The YANG model is then uploaded into a Sysrepo datastore. This is required for the Netopeer2 to be able to read it and compare the payload. This makes devices "NETCONF compatible". The YANG data models are used by Netopeer2 to read the functions that devices are capable of, for example "wave" or "power" in power meter. Function "wave" is equivalent to GPIB call defined for a power meter, in this case it means "Set wavelength for calibrating detector response".

Commands to upload the files into datastore are as follows:

```
sysrepoctl --install -yang=/home/yuliya/sysrepo/laser/laser-
    model.yang
sysrepocfg -import=/home/yuliya/sysrepo/laser/laser-config.xml
    --datastore=startup -format=xml laser-model
```

The first command installs the YANG model into the Sysrepo datastore, making the data model available for use (such as validating configuration data or generating configuration-related operations like read or update operations). The second command imports configuration data from an XML file (laser-config.xml) into the Sysrepo datastore, associating it with the "laser-model" YANG model and specifying that the configuration should be stored in the startup datastore in XML format.

The process of testing the model is further completed by creating an instance of the YANG model. This instance represents the specific configuration or state of a device, which is used when exchanging data.

For our setup we incorporated a tool called Netopeer2. Netopeer2 is a server that implements network configuration management based on the NETCONF protocol[58]. Netopeer2 is a set of NETCONF tools build on the "libyang" and "libnetconf2" libraries. It allows operators to connect to their NETCONF-enabled devices as well as developers to allow control their devices via NETCONF. The Netopeer2 server uses Sysrepo as a NETCONF datastore implementation. Figure 3.11 shows the architecture of Netopeer2. The netopeer-cli is accessing configuration data storage of a device via SSH/TLS over the network. In other words, it communicates with a NETCONF enabled device using SSH and accesses its configuration datastore. NETCONF Server: This is the core part of Netopeer2, which listens for NETCONF requests from clients, processes those requests, and interacts with network devices to apply or retrieve configurations.

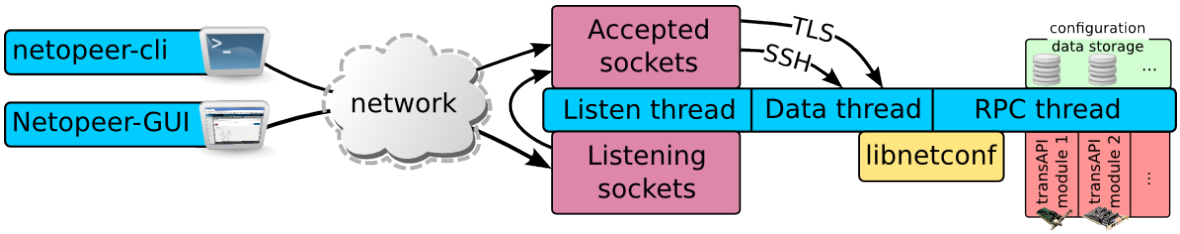

Figure 3.11: Netopeer architecture diagram[8].

Netopeer2 command example that is used to reconfigure devise settings is shown below. This command specifies editing configuration in running datastore of the device. The command below utilises the instance of a YANG model in XML representation.

```
edit-config --target running --config=/home/yuliya/sysrepo/lms
    /sysrepo-power-meter-wave.xml
```

Datastores are a fundamental concept binding the data models written in the YANG data modeling language to network management protocols such as NETCONF and RESTCONF. A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof. A datastore maps to an instantiated YANG data tree [59]. This makes a call to the "C" driver for the device, and calls a write function to be performed on the device. Reaches to datastore and reads YANG model, then connects to device via a driver and initiates a write operation on the device, allowing it to perform

a configuration change or update.

Netopeer2 and Sysrepo are connecting using private key. The command to generate private key is

{sudo ssh-keygen -t rsa -N "" -f /etc/ssh/ssh-host-rsa-key}.

Netopeer2 has a directory "keystored" that needs to be compiled before server and CLI are compiled. This directory contains keys that are used for communication between Sysrepo and Netopeer2. Sysrepo is using subscription mechanism to connect to other processes running, it creates a session and allows Netopeer2 to connect to its API [60]. After installation, server has a default startup configuration which enables SSH connections on all the interfaces on the designated NETCONF SSH port 830. To connect to the server on "localhost" Netopeer2 CLI can be used.

Next step plugged in the device driver into Sysrepo by copying drivers into "plugins" directory in Sysrepo. This is done by using "cmake" tool to add to code, reference of the driver directory as shown in Figure 3.12.

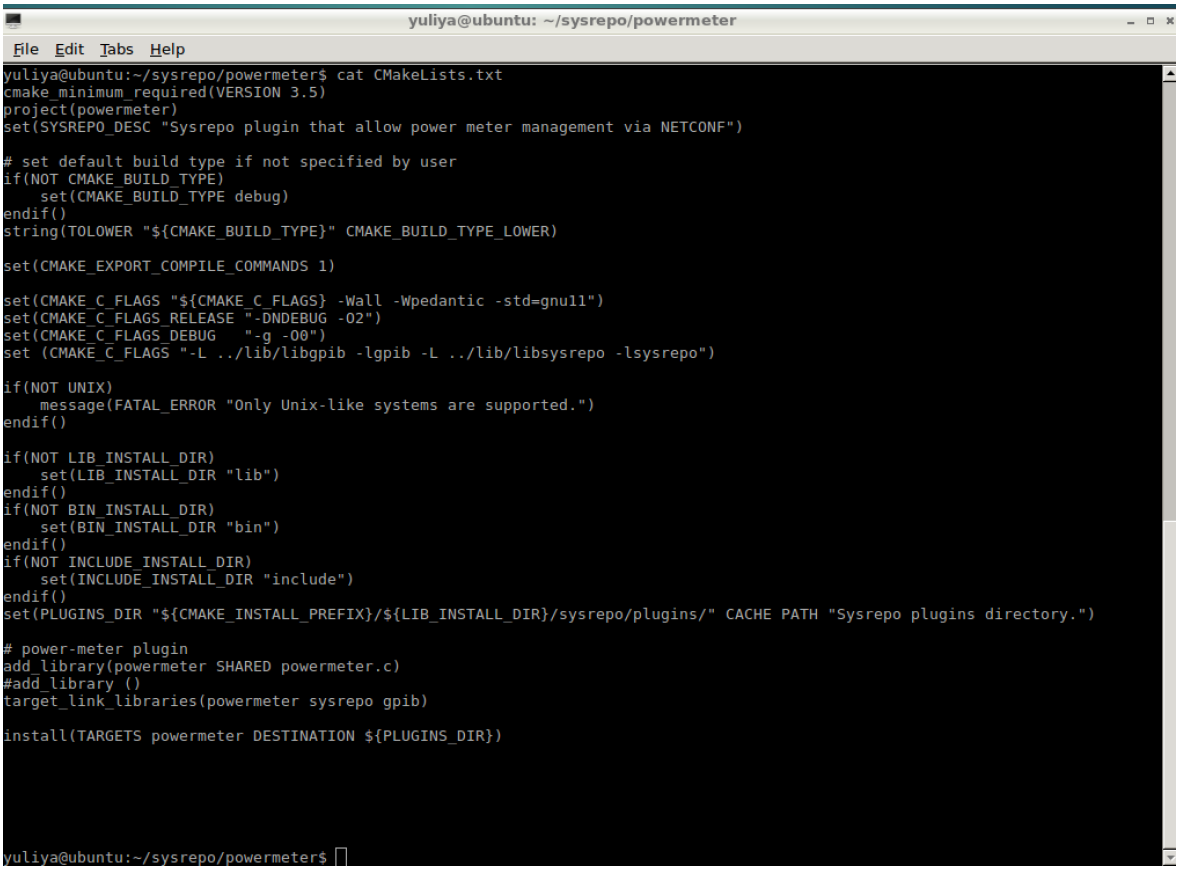

Figure 3.12: The file showing how the Sysrepo was connected to drivers written for each of the legacy devices.

Figure 3.13 illustrates how all Sysrepo components are connected and started. Those include sysrepo-deamon, sysrepod, netopeer2-deamon, netopeer2-cli.

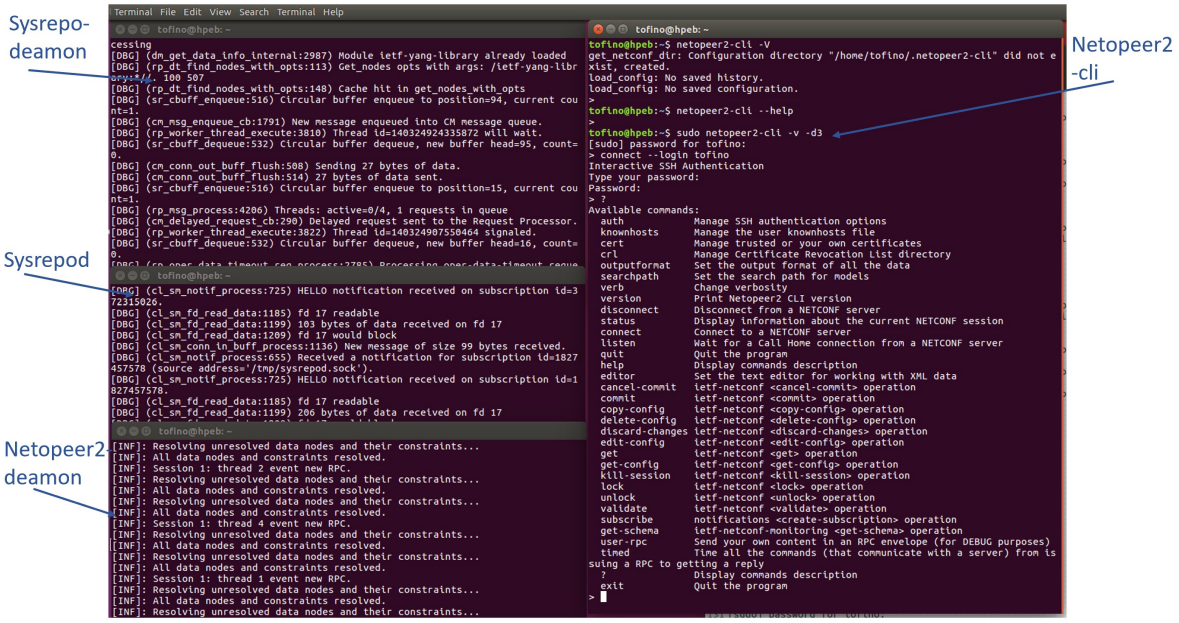

Figure 3.13: All Sysrepo components started.

Figure 3.14 shows an example of commands issued in netopeer2-cli to configure disaggregated devices.

```
>
>
> edit-config --help
edit-config [--help] --target running|candidate --config[=<file>] [--defop merge|replace|none] [--test set|test-only|test-
then-set] [--error stop|continue|rollback]
> edit-config --target running --config=/home/tofino/sysrepo/laser/laser-config.xml
OK
> edit-config --target running --config=/home/tofino/sysrepo/lms/lms-config.xml
OK
> edit-config --target running --config=/home/tofino/sysrepo/powermeter/powermeter-config.xml
OK
>
```

Figure 3.14: Applying configuration to devices connected via Sysrepo.

The precise YANG data models for each of the physical devices used are shown in Figures 3.15, 3.16 and 3.17 . In Figure 3.15, it shows the YANG data model for a power meter (ILX8210H), where the measurement wavelength could be set to a number between 850 to 1600nm; the acquisition time filter could be set to slow, medium or fast, and the power could be captured in dBm or as ratio in dB. In Figure 3.16 YANG model for Agilent N7714A showing the following leaf elements were defined: state to set laser on or off; the unit of laser source (dBm or W); wave to set the wavelength value; and laser-power to set a power. Finally, Figure 3.17, the YANG model for a different manufacturer power sensor and attenuator (Agilent 81635A and 81571A) are

shown. The interesting fact here is that these two units ("slaves") were within the same chassis rack ("master"), and hence the YANG model had to reproduce the command series from the chassis to the unit, and in transceivers. In the attenuator section of the model defined were wave and attenuation units, representing the value to set the wavelength and to attenuate the signal respectively.

```
module sysrepo-power-meter-wave
{
  yang-version 1;
  namespace "urn:ietf:params:xml:ns:yang:sysrepo-power-meter-wave";
  prefix sysrepo-power-meter-wave;
  description "Power Meter ILX8210H YANG Module";
  container pm
  {
    leaf wave {
    type uint16;
    description "Sets the wavelength in a range from 850 to 1650 nm ";
    }

    leaf filter {
    description "Power meter filter speed (slow, fast, medium)";
    type string;
    }

    leaf mode {
    description "Power meter units mode (dB, dBm";
    type string;
    }

    leaf power {
    type string;
    description "Shows the current power from the device display, measures accurately over 80 dB dynamic range from +30
    dBm to -50 dBm.";
    config "false";
    }
  }
}
```

Figure 3.15: YANG model for ILX8210H powermeter. Listing container and leaf elements of the model.

```
module laser-model{
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:laser-model";
  prefix laser-model;
  description "Laser Agilent N7714A YANG Module";

  container laser {
    leaf laser-state{
        type uint8;
        description "Determines if the laser is on or off of the source, 0 Disabled, 1 Enabled";
        }
    leaf unit {
        type uint8;
        description "Determines the unit of laser source N7714A, 1=dBm, 0=W";
        }
    leaf wave {
        type string;
        description "Sets the wavelength of the source";
        }
    leaf laser-power {
        type string;
        description "Determines the power of the laser";
        }
    }
}
```

Figure 3.16: YANG model for Agilent N7714A laser listing container and leaf elements of the model.

```
module lms-model{
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:lms-model";
  prefix lms-model;
  description "Power Sensor 81635A YANG Module, Attenuator 81571A YANG Module";

  container powersensor {
    leaf unit-master {
        type uint8;
        description "Determines the unit value of master channel 81635A, 0=dBm, 1=W";
        }
    leaf wave-master {
        type string;
        description "Determines the wavelength value of master 81635A, MIN = 800nm, MAX=1650nm, DEF=1225nm(This is not
        the preset (*RST) default value but is half the sum of, the minimum programmable value and the maximum
        programmable value)";
        }
    leaf averagetime-master {
        type string;
        description "Determines the averaging time value of 81635A, if unit is not specified its in seconds by default";
        }
    leaf master-power {
        type string;
        config false;
        description "Determines the power value of 81635A master channel";
        }
    leaf wave-slave {
        type string;
        description "Determines the wavelength value of slave 81635A, MIN = 800nm, MAX=1650nm, DEF=1225nm(This is not
        the preset (*RST) default value but is half the sum of, the minimum programmable value and the maximum
        programmable value)";
        }
    leaf unit-slave {
        type uint8;
        description "Determines the unit value of slave channel 81635A, 0=dBm, 1=W";
        }
    leaf slave-power {
        type string;
        config false;
        description "Determines the power value of 81635A slave channel";
        }
    }

 container attenuator
    {
    leaf attenuation {
        type uint16;
        description "Determines the attenuation value of 81571A, Min=2dB, Max=62dB, Def=2dB";
        }
    leaf wave {
        type string;
        description "Determines the wavelength value of 81571A, Min=1200nm, Max=1700nm, Def=1550nm";
        }
    }
}
```

Figure 3.17: YANG model for Agilent Power Sensor 81635A and Attenuator 81571A. Listing container and leaf elements of the model.

The above Figures illustrate how a YANG model is structured,but in order to use the settings defined in the models it requires an XML format. NETCONF protocol uses XML for encoding data in its Content Layer. Therefore the YANG instance being passed to the device via NETCONF is defined in XML format. An example of a simple XML definition is shown in Figure 3.18. For example, if a user needs to change the wavelength in 3.18, it requires a change to be specified in the XML file, which is then pushed by the NETCONF server.

```
<powersensor xmlns="urn:ietf:params:xml:ns:yang:lms-model" >
    <unit-master>0</unit-master>
    <wave-master>1530nm</wave-master>
    <averagetime-master>2s</averagetime-master>
    <wave-slave>1550nm</wave-slave>
    <unit-slave>1</unit-slave>
</powersensor >

<attenuator xmlns="urn:ietf:params:xml:ns:yang:lms-model" >
    <attenuation>3</attenuation>
    <wave>+1.5300000E-006</wave>
</attenuator>
```

Figure 3.18: XML representation of data to set in LMS.

3.3 Software Defined Networking for Optical Communications

ONOS controller, as described in section 2.6, is designed for network management. The main features of the controller include distributed core to achieve scale and high availability; policy based northbound API; support for different southbound protocols; and modular software.

ONOS enable network wide topology view of connected devices, and communicate with devices via protocols such as NETCONF, RESTCONF, OpenDaylight. In this thesis we concentrated on using NETCONF as protocol for communications with devices. This is because NETCONF offers a standardized, secure, and efficient way to manage device configurations.

In ONOS, the terms, provider (Figure 2.7) is an API that interfaces with the network via protocol-specific libraries, in our case it was NETCONF protocol library, with distributed core via provider service interface components. ONOS allows to create an application that would provide a logic to manipulate and configure devices connected.

ONOS controller is already designed to control and configure multiple network elements such as switches for packet domain and some ROADMs for optical domain. However, the functionality for managing optical domain instruments in ONOS v13.3 that was used in this thesis was limited. In this work, we connected passive devices of an optical network such as laser, power meter and attenuator, to illustrate the full network architecture.

3.4 Connecting disaggregated devices to ONOS controller

Connection of a device to ONOS is a straight forward process detailed in the official documentation [43] and its architecture [7]. To connect a device to ONOS, one needs to have credentials for the user that is trying to connect, such as username and password, and port number on which to connect. This will notify ONOS of a device, and ensure it is shown in the topology once successfully connected. However, ONOS will not know what functions and capabilities the devices have. In this case, a driver is needed to create an understanding between ONOS and a device, and to let ONOS know what functions each device has. For this reason, and in this thesis, drivers were written for each device that was connected to ONOS[1]. The process requires an operator to send a 'POST' request with the details of the device to ONOS, as the controller does not do an active scan to identify potential new connections. In order to realize this, a file called "netcfg.json" in directory "ONOS-HOME/tools/config/" and edit the file to contain device specific details such as IP address, port number, protocol used, and user credentials.

To connect devices to ONOS that do not have a port number or IP address, creating a virtual Ethernet interface with an IP address was necessary to simulate multiple IP addresses on that interface. This would allow to connect the devices to controller and communicate with them. ONOS requires all connected devices to have an IP address and port number. However, not all disaggregated legacy devices had such identifiers. Here we created virtual interfaces (e.g IP address) to simulate Ethernet interfaces to emulate such connections.

Typically a "POST" command to connect a device to ONOS controller will look as shown in Figure 3.19.

```
sudo curl --user "onos:rocks" --header "Content-Type: application/json"
--request POST http://localhost:8181/onos/v1/network/configuration -d
@netconf-cfg.json
```

Figure 3.19: A "curl" command used to 'POST' the device details to ONOS, to set up a connection.

Content of a netcfg.json file is shown in Figure 3.20 using NETCONF protocol and specifying port 830 to connect to as this is standard port for this protocol.

[1]Please see Appendix B for drivers

```
root@ubuntu:/home/yuliya/onos/tools/test/configs# cat netconf-cfg.json
{
  "devices": {
    "netconf:127.0.0.2:830": {
      "netconf": {
        "ip": "127.0.0.2",
        "port": 830,
        "username": "yuliya",
        "password": "yuliya"
      },
      "basic": {
        "driver": "laser"
      }
    }
  }
}
root@ubuntu:/home/yuliya/onos/tools/test/configs#
```

Figure 3.20: Contents and structure of netconf-cfg.json file.

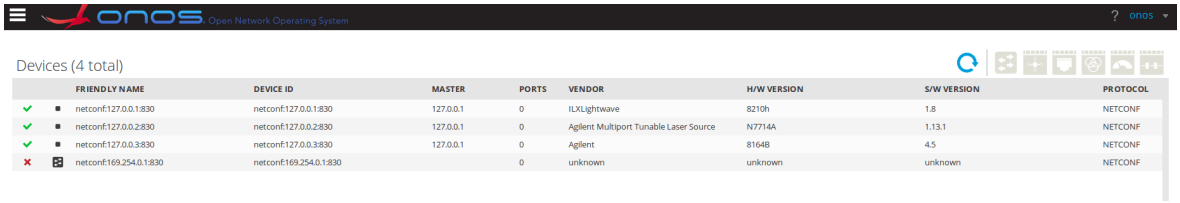

Figure 3.21: ONOS user interface view of connected devices with device descriptions.

Figure 3.22: ONOS CLI view of connected devices with device descriptions.

The controller itself does not contain any functionality to edit device configurations. This had to be defined in the drivers written in "C" language for GPIB specific commands; and in the drivers created in ONOS for each device was written in "Java" language. The ONOS driver had a definition of a NETCONF session which allowed us to setup a connection session with the device. In version ONOS-13.3 which was used for this thesis, the user was required to write a driver to make ONOS aware what device was being connected so that NETCONF capability exchange can occur to make ONOS aware of what were devices capabilities, features and functions it had.

ONOS communicates with Netopeer2 using "libnetconf" library [61], it is a 'C' language library created to facilitate development of NETCONF server and client. It provides basic functions to connect NETCONF client and server to each other via SSH, to send

and receive NETCONF messages and to store and work with the configuration data in a datastore (in this thesis, it was a Sysrepo datastore).

3.5 Summary

This chapter described how in this thesis we enabled many legacy and dissagregated instruments to be setup, including the protocols to be used, and how the configurations for devices can be presented in a YANG model view. In next Chapter, we will show how to integrate the SDN-enabled devices and how they work with NETCONF and P4 language. We describe the topology view of the connections in detail.

The work described in this chapter is detailed in the following outputs:

- Y. Verbishchuk, C. Sreenan, and F. Gunning, "Configuration and monitoring of the optical physical layer using software-defined tools," in OSA Advanced Photonics Congress (AP) 2019 (IPR, Networks, NOMA, SPPCom, PVLED). Optical Society of America, 2019, p. NeT1D.2., Available: http://opg.optica.org/abstract.cfm?URI=Networks-2019-NeT1D.2 [25]

Chapter 4

SDN for Optical Networks

(SDN enabled devices/SDN in physical layer/SDN in packet layer)

4.1 Layer 1 - physical layer (optical)

4.1.1 ROADM

Traditionally, network switches were supplied by vendors like Juniper or Cisco and came preloaded with proprietary operating systems such as JunOS or IOS. This meant operators were purchasing not just the hardware but also a bundled operating system. These proprietary systems offered limited functionalities for the operator's use and restricted any modification or customization. For instance, operators could not adjust device-specific settings like fan speed or other hardware parameters.

With the emergence of Software-Defined Networking (SDN), many device manufacturers and network providers began exploring more open hardware alternatives to the traditional approach. This shift led to the growing popularity of devices such as whitebox, britebox, and bare-metal switches[50]. The appeal lies in the flexibility these devices offer, as they enable network operators to customize the hardware and software to meet their specific needs. This flexibility is made possible by utilizing open software, which can be programmed by the operator and integrated with SDN tools to unlock its full potential, as described previously in chapter 2.

Whitebox switches, for instance, typically come with a pre-installed operating system, but they are designed to be open and reconfigurable. This separation between hardware and operating system allows operators to replace or modify the software as needed. In contrast, bare-metal switches are standard commodity switches equipped with a

bootloader, such as the Open Network Install Environment (ONIE)[62], which enables the installation of an operator's preferred operating system. Britebox switches, short for branded whitebox, are produced by original design manufacturers ODMs. and feature a branded bezel, as seen with vendors like Dell. These devices use standardized hardware and serve as cost-effective building blocks for networks.

A common choice for operating systems in SDN and whitebox switches is Linux-based systems. These systems provide access to a wide array of open-source and free Linux tools, allowing administrators to tailor devices to their specific needs while maintaining strong defenses against cybersecurity threats.

In this thesis, a whitebox Lumentum ROADM was used as part of an emulation of an optical network. A ROADM (re-configurable optical add drop multiplexer) has the ability of dropping and adding optical wavelength channels to the optical fibre backbone. This is achieved through the use of a Wavelength Selective Switch (WSS) module. A ROADM is a layer 1 device and is part of a DWDM optical transport network systems. The control of the Lumentum ROADM was enabled by NETCONF protocol. In this thesis, OpenRoadm was in its infancy and not available for the experiments carried out at the time. Hence, NETCONF protocol was used for device configuration. Newer ROADM models have pre-loaded OpenRoadm technology for configuration and management.[63]

The main components of a ROADM are a multiplexer, demultiplexer and amplifiers to equalise optical powers. Wavelength division multiplexing is a technology which multiplexes a number of optical carrier signals (of different wavelenghts) onto a single optical fiber. This technique enables bidirectional communications over one strand of fiber, as well as multiplication of capacity. Demultiplexing is a technique that works oppositely to multiplexing, were it separates wavelengths into separate fibre signals by "colour". This technique allows to add or drop individual or multiple wavelengths carrying data from a transport fiber without the need to convert the signals to electronics and back again to optical signals as shown in Figure 4.1. ROADM allows to utilize C-band of an infrared spectrum, which is within 1530 - 1570nm[64]. This is a common range used in installed systems today, with research exploring extending wavelengths beyond the C-band.

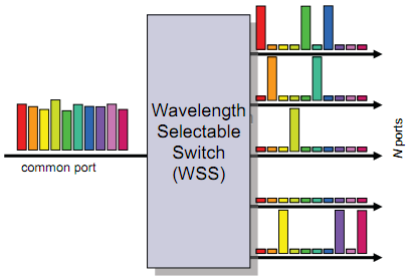

Figure 4.1: An illustration of wavelength selective switch work principle.

In addition to the mux, demux, WSS and amplifiers, the Lumentum ROADM used in this experiment also contained components such as Optical channel monitor, Line IN and Line Out ports, booster and amplifiers. The Optical Channel Monitor (OCM) is used to measure the power of individual channels, while Variable Optical Attenuators (VOAs) adjust the relative channel powers. Since VOAs can only reduce, not restore, signal strength, they do not compensate for loss directly. Instead, in combination with optical amplifiers, OCMs and VOAs are used to balance and equalize power levels across channels in order to maintain signal quality and system stability. Those components allow monitoring and dynamic balancing of optical power for all channels across the network and a detailed architecture of the Lumentum ROADM used in thesis is shown in Figure 4.2.

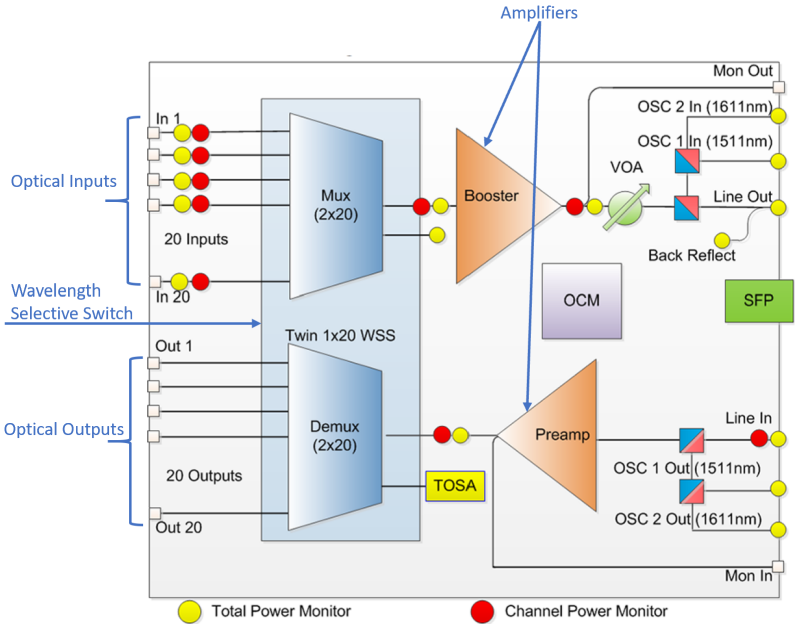

Figure 4.2: ROADM architecture from official device documentation.[9]

An example of the functionality of a wavelength-selective switch is illustrated in 4.1. This device takes multiple optical wavelengths at its input and separates them into distinct single-wavelength channels. With the capability to configure wavelength spacing

as fine as 6.25 GHz, it enables the aggregation of multiple wavelengths onto a single optical carrier (fiber), optimizing the utilization of available bandwidth.

The actual physical ROADM box contains a set of optical inputs and outputs, those are labeled on the figure 4.2 with numbers from 1 to 20 inputs and outputs, as shown in Figure 4.3. All optical connections used on the ROADM-20 are optical plugin connectors, arranged in duplex pairs. Connectors are organized in two rows across the face plate. The connections do not have an internal definition of port but rather it is a pair of Input and Output, that allows wavelengths to be transmitted on different channels[65].

The channels arriving at the Line-In port are within the C-band spectrum, as previously described, with channel bandwidths ranging from 12.5 GHz to 100 GHz in most current deployed systems. The ROADM employed in our experiment can configure channel bandwidths within these limits, supporting configurable channel widths starting from 6.25 GHz up to the entire C-band. However, as of 2024, ongoing research is exploring ways to extend channel bandwidths to 400 GHz, paving the way for even greater data-carrying capacity.

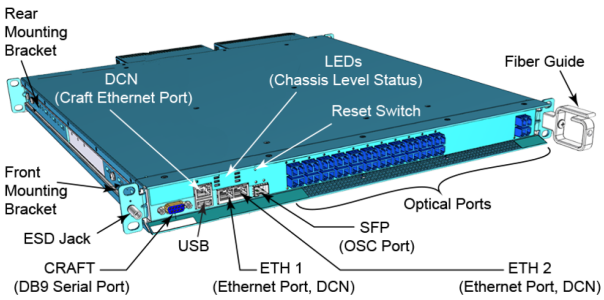

Figure 4.3: ROADM ports

Experimentally, the optical monitor port of ROADM was connected to Optical Spectrum Analyzer (OSA), which allowed to monitor the signal wavelength and configure correct channel. An example of a number of spectra output from the ROADM captured with the OSA is shown in Figure 4.4, the input port can receive the whole C-band and selectively add/drop, bypass or block any channel selected.

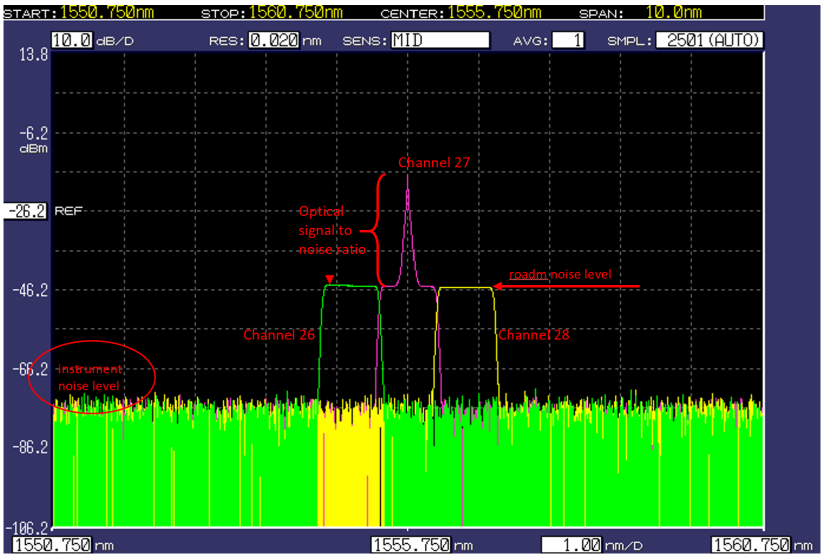

Figure 4.4: Optical Spectra of 3 channels, X-axis is wavelength, Y-axis is optical power density.

In order to understand the capabilities of the ROADM, a simple test was carried out by launching a Continuous Wave (CW) laser light signal, centered at 1555.75nm on channel 27 from [64], to the LINE-IN of the ROADM. Normally in an optical network, up to 96 wavelength channels would be launched to LINE-IN of the ROADM, but this simple test allows us to understand the ROADMs operations. In addition, the OSA was connected to the MON-OUT after the second amplifier (Booster amplifier in Figure 4.2) in order to check all different optical parameters. For example, letting channel 27 through, or only allowing channel 26 or 28 through, even though they both had no signal input. So the spectrum of the input signal is shown in 4.4 (pink trace),confirming its center wavelegth at 1555.75nm. The presice optical power value is not really reliant on it from monitoring point. However, the important aspect to observe is the Optical Signal to Noise Ratio (OSNR) indicated in the picture of about 30dB. Note, however, that the noise level seem to be $<\sim$ 66.2 dBm and that the OSNR was calculated with a reference noise at -46.2dBm instead. This is due to the fact that the monitoring optical point is at the output of the booster amplifier in 4.2, which adds in-band (i.e. under signal) and out-of-band Amplified Spontaneous Emission (ASE). This was confirmed when dropping channel 27 (which contained signal) and adding channel 26(green channel on 4.4) on 28 (yellow channel on 4.4) both with no signal but still showing the ASE effects from the optical amplifier. The OSNR is a critical parameter in optical networks that depicts the quality of a signal end its relation to the ability of this signal reaching its destination with integrity.

To set up a ROADM for this experiment, we have a utilized vendor provided manage-

ment GUI. This allowed us to have full view of the components that can be configured. A view of a GUI configuration for a channel is represented in Figure 4.5. The same configuration is represented in XML format that is used by YANG and NETCONF. The main settings used were name of a connection, start, end and middle frequency to set the correct channel, and attenuation to test the quality of the service. Those settings were modified during the experiment and will be explained in more details in Chapter 5.

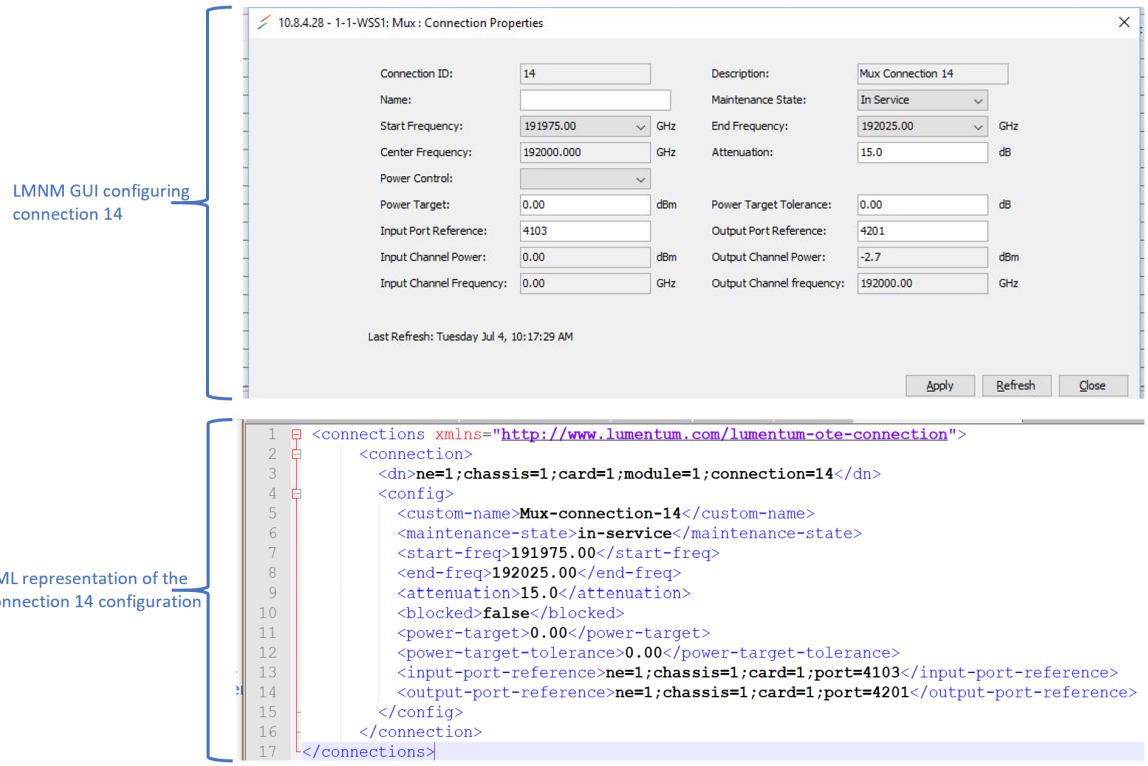

Figure 4.5: Lumentum Multi Node Management Interface (top) connection configuration vs same configuration in xml representation (bottom).

The Lumentum ROADM used in the experiment was NETCONF-enabled and contained predefined YANG models. NETCONF-enabled means that the device itself contains a NETCONF server running as part of its software components. This was an easier implementation than the devices from Chapter 3, where the ROADM is SDN-enabled, while the legacy disaggregated devices had to be made SDN-enabled through Sysrepo, Netopeer2, YANG and NETCONF. (See section 3.2)

The Lumentum ROADM YANG models enabled configuration and control of many functions. Of note and used within this thesis were NETCONF control interface, Amplifier operating modes (Constant power mode, Constant gain mode), all Optical Supervisory Channel (OSC) ports and Multiplexer (MUX) inputs.

To connect the ROADM to a laptop, a serial DB9 cable was used, with Ethernet connection, which allowed to plugin into the Ethernet port on the laptop. The ROADM had a set IP address, username and password, which allowed easy connection, and the user had a choice of protocol and port to be chosen. In our case it was a NETCONF protocol and standard NETCONF port 830.

We also have utilized Netopeer2 to connect to the ROADM and, this allowed us to reconfigure the device using the YANG module instead. Netopeer2 enabled us to have pre-configured multiple YANG module instances (for example, have an XML file ready for configuring channels on the mux and demux) and utilise those to reconfigure the device on a cli instead of GUI.

An example of configuring the device via Netopeer2 is shown in Figure 4.6.

In summary, we have two options of setting up the device for our usecase. Option 1 is to use netopeer2 for device management, or option 2, which uses a vendor provided Multi-Node Manager application (GUI). This works by connecting Ethernet cable between a ROADM and a laptop that runs a GUI application and using NETCONF protocol to connect to a device. Using netopeer2 for management of the ROADM is simpler in comparison to setting up of the legacy components, as ROADMs already have built in store for YANG models. However, in this experiment we have used the ROADM management GUI.

A ROADM supports both candidate, running and startup datastore. ROADM datastore already has all YANG modules stored and has defined internal software that performs changes requested by a NETCONF command. Therefore for ROADMs we only need to use Netopeer2 to connect and reset settings as required. To connect ROADM to Netopeer2 the command was used as shown in Figure 4.6.

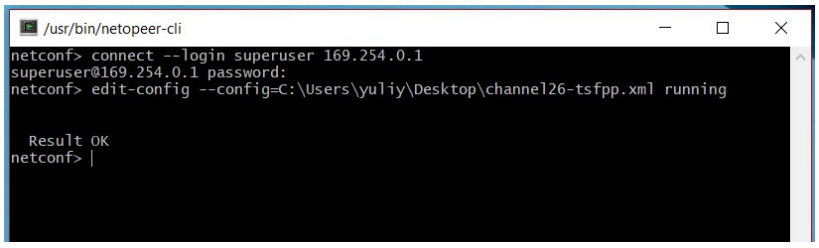

Figure 4.6: Connecting ROADM to netopeer2. The picture shows command to login to the netopeer and then sending edit-config command with an XML configuration definition to be committed to running datastore.

The ROADM connects to Netopeer2 using SSH, the default protocol employed by NETCONF for secure communication. The NETCONF has a list of commands that

allows to set, reset and monitor devices. Once connected one can retrieve or change settings of the device by using edit-config command and providing instance of a YANG model in an XML format. In this thesis, a LMNM GUI tool was utilized for two main reasons: (a) Netopeer2 in our experiment relies on Sysrepo as its datastore, and (b) the ROADM has its own built-in datastore. Access to the ROADM was possible through two methods: the LMNM GUI or the CLI. For this work, a NETCONF session was executed using the LMNM GUI. A NETCONF session establishes a connection between the network configuration application and the network device. The YANG models deployed on the ROADM Whitebox are a combination of Lumentum-specific models and standard models, as defined in the YANG RFC standard [38].

Once we connected to the ROADM via Netopeer2, user can run a command "status" to see the device capabilities (capability exchange) and its available YANG modules as is showing in Figure 4.7.

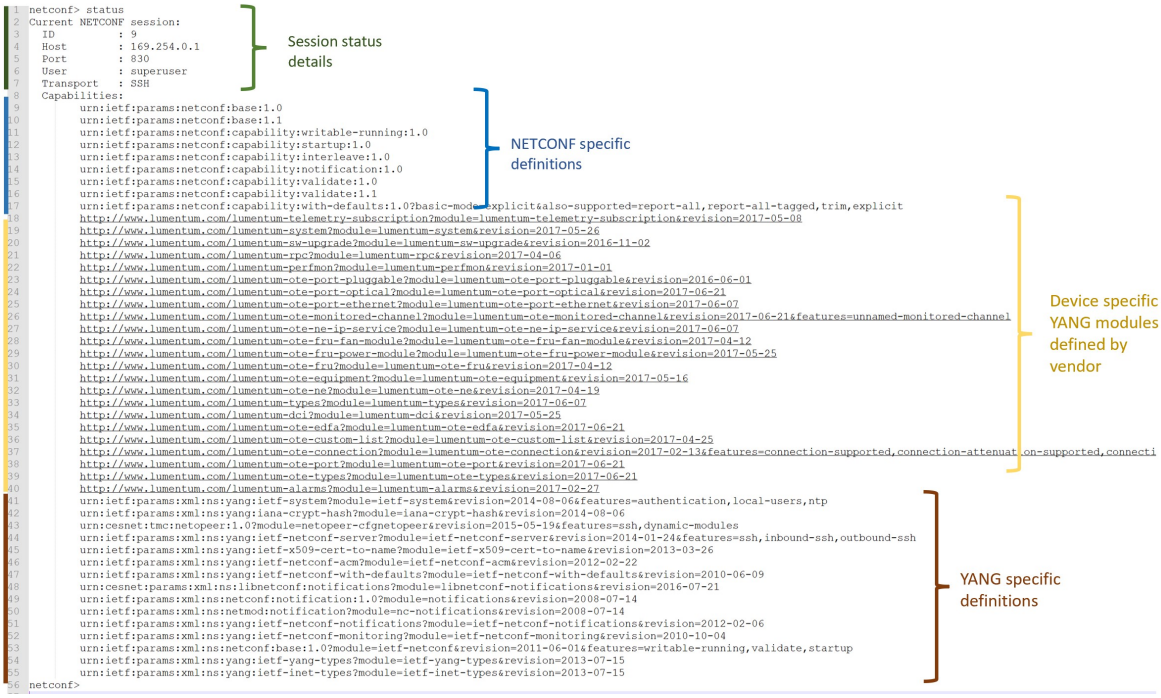

Figure 4.7: ROADM connection to netopeer2, showing output of a status command listing all installed YANG modules on the device.

4.1.2 Open and Disaggregated Transport Networks

Optical network domain is undergoing a lot of changes in order to achieve SDN capabilities and be more flexible in terms of management of the systems. Historically optical transport networks have been designed with proprietary systems, where a vendor not

only provided network equipment hardware, but also locked-in software for management and control, as described in Chapter 3. A ROADM, for example, together with transponders, amplifiers, lasers, power meters, attenuators make up the physical layer infrastructure of equipment hardware for operations of the network. Those systems traditionally are working on the basis of set and forget configuration, tied to a vendor specified software, making re-configuration difficult and expensive. Moreover, this is of a disadvantage for many of the network owners/operators, as there is no flexibility to mix and match hardware and software from different vendors, enabling gradual network updates, reducing capex. The demand for change is growing, and some of the main points raised by operators for a future network include open and standard vendor-neutral APIs, separation of a transponder behavior from control of the line systems , modular and production ready format. This has brought attention to separation of the device dependency on vendor proprietary software and disaggregation of terminal equipment.

The flaws of current optical transport networks have been reviewed and addressed by collaboration of vendors and network providers in ODTN [66] project. Open and Disaggregated Transport Network (ODTN) project is an operator-led initiative to build data center interconnects using disaggregated optical equipment, open and common standards, and open source software.

At the time of starting this thesis in 2017, there was a significant interest in opening systems, disaggregating equipment to software, and creating orchestration tools for better management. It is within this context that the work on this thesis bases on implementing open management and control systems of disaggregated devices that are commonly used in the network, in particularly legacy devices. The legacy devices that were used in this thesis are shown in table 4.1 below and described in more detail in chapter 3:

Table 4.1: List of Devices and their Specifications

Device type	Vendor	Model	Control Interface
Power Meter	ILX	8210h	GPIB
Laser	Keysight	N7744x	GPIB
Attenuator	Agilent	8164A/B	GPIB

Those devices are General Purpose Interface Bus (GPIB) compatible and connect to a management machine via Agilent 82357b GPIB-To-USB converter.

In optical networks, various instruments are utilized to ensure optimal system performance. A power meter is used to measure the total optical power in fiber optic equipment, such as at the end of an optical fiber connector or the output power from

a laser. A laser, acting as a light source, serves not only for regular operation but also for testing purposes. In the context of this thesis, an external tunable laser was used as a test light source, scanning through different wavelengths to simulate interference (or noise) with the signals from pluggable devices.(more on this in Chapter 5)

An attenuator is a device used to reduce the optical power of a signal. Excessive power can saturate the optical receiver, leading to degraded performance. By adjusting the optical power, the attenuator ensures the best performance of the fiber optic system. This is especially useful for controlling the optical power at various stages, such as before a detector or receiver, to prevent saturation. Flexible ROADMs often integrate internal attenuators for this purpose. In this thesis, an external attenuator was used to demonstrate the potential for disaggregation and full control over the optical signal.

To control the disaggregated devices, they had to be physically connected via the GPIB interface, as described in Chapter 3. Consequently, the instruments listed in table 4.1 required the use of the GPIB connection, which facilitated the development of a driver in 'C' code to manage them. An extensive description of how to connect the disaggregated legacy devices depicted in chapter 3.

4.2 Layer 2 – 7(packets)

4.2.1 SDN enabled devices

Since the use of fixed function switches, not much was changing in the ways the network was build and managed. Fixed function means the device is capable of a certain functionality and if there is a request for a feature that is not part of devices current function set, it potentially may take years to be incorporated into the device itself in current systems. As this feature would not be able to be installed on a current device, but instead a completely new device would need to be designed to include such request/feature.

This kind of approach meant that if there is something specific that a network operator would like to use in their setup, they may have to do a workaround to implement the desired outcome partially, or else they may have to replace the hardware to achieve the desired outcome. This kind of complexity of devices and the inability to use software upgrade rather than hardware has been resolved by a network programmable switches. An example of such device is the Barefoot Tofino(now Intel) switch [67]. This is a breakthrough in the technology as it allows the flexibility of the software and ability to fully program the network switch.

A P4-programmable switch differs from a traditional switch in two essential ways:

- The data plane functionality is not fixed in advance but is defined by a P4 program. The data plane is configured at initialization time to implement the functionality described by the P4 program and has no built-in knowledge of existing network protocols.
- The control plane communicates with the data plane using the same channels as in a fixed-function device, but the set of tables and other objects in the data plane are no longer fixed, since they are defined by a P4 program. The P4 compiler generates the API that the control plane uses to communicate with the data plane[68].
- The Barefoot Tofino switch differs from conventional Ethernet switches with SFP+ ports primarily through its fully programmable data plane, enabled by P4. Unlike fixed-function switches that only support standard L2/L3 operations, Tofino allows custom packet parsing, matching, and actions, enabling new protocols, in-network telemetry, and application-specific processing at line rate. It also provides flexible multi-speed port configurations, advanced flow-level monitoring, and fine-grained traffic control beyond standard QoS features. These capabilities make Tofino particularly suitable for SDN integration, experimental networking, and next-generation deployments where programmability, visibility, and protocol innovation are required.

The most interesting component of the switch was Tofino ASIC from Barefoot (now Intel) which utilizes "Match Action Pipeline" architecture, now called Protocol Independent Switch Architecture (PISA)[69] for packet processing. The device was the worlds fastest programmable switch in 2016 with processing speed of 6.5Tb/s.

The front panel of the Barefoot Tofino used in this thesis is presented in Figure 4.8 and the architecture of a device is shown in Figure 4.9

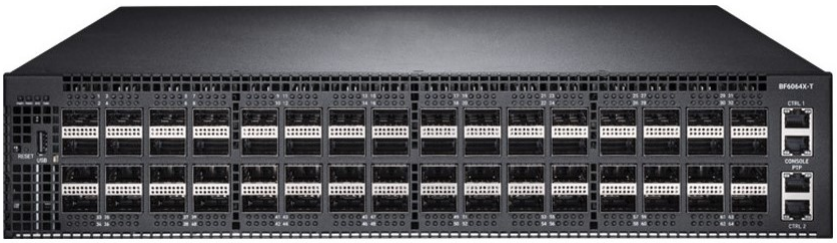

Figure 4.8: Tofino front panel view.

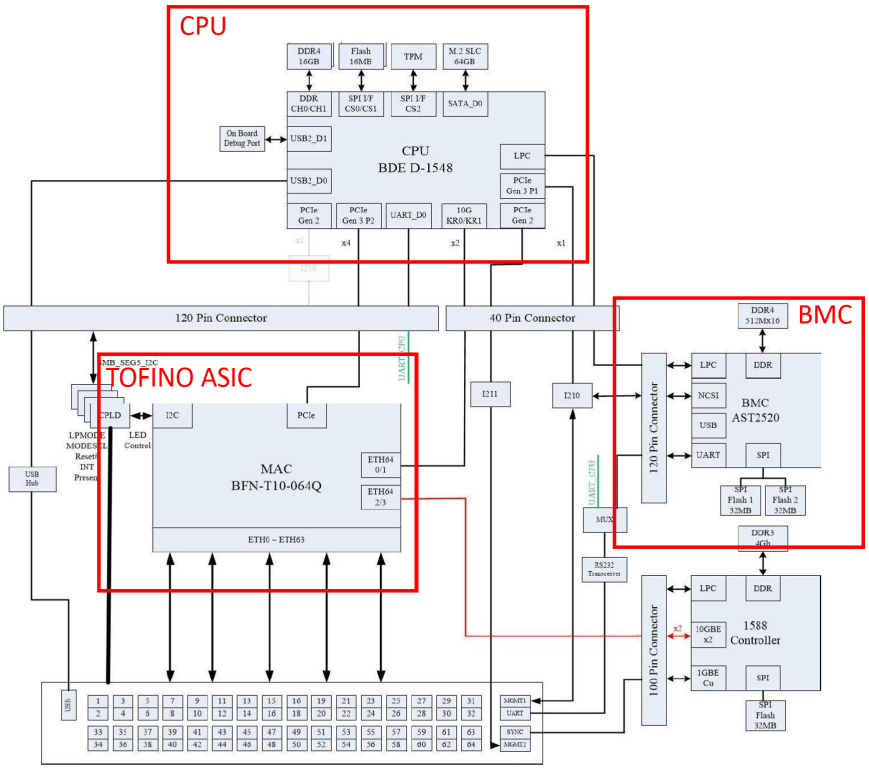

Figure 4.9: Tofino block diagram, specifying main components[10].

The main components of the device are Central Processing Unit (CPU), Baseboard Management Controller (BMC) and ASIC. CPU is taking care of the operating system running on the device, in this case it was Ubuntu 16.04 LTS, and its job is to ensure that all components run as expected.

A BMC is a specialized service processor that monitors the physical state of a network server or other hardware devices using sensors and communicating with the system administrator through an independent connection.

The Tofino ASIC is an application specific integrated circuit, or a microchip, designed for a specific application. In case of network switches, ASIC is designed to quickly do switching decisions based on certain criteria. Traditionally in Layer 2 switches, it is based on MAC address, the more ports device has, the more the processing ASIC has to do. Tofino has 65 100GbE QSFP28 ports which support 100GE applications in a compact two-rack unit form factor. Each port is capable to divide into 4 lines, capable of supporting 10Gbs/25Gbs/50Gbs/100Gbs speed each. More details shortly.

PISA is an architecture for high-speed programmable packet forwarding; it is a way of letting users program a network for themselves, without loss of performance, using the programming language P4. The device is capable of reviewing and manipulating a

frame/packet header, depending on a criteria specified by a P4 program. Therefore it is possible to have full access to all header fields, add or remove, or read any of those as needed. The device is protocol independent because it does not understand protocols until it gets programmed to. The architecture of a pipeline is shown in Figure 4.10.

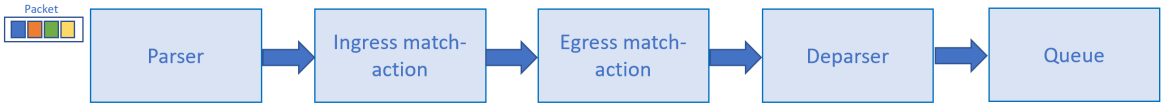

Figure 4.10: Match action pipeline architecture diagram.

The pipeline includes parser that is receiving a packet and scans it for headers. Ingress match action is a mix of Ternary Content-Addressable Memory (TCAM) and Static Random Access Memory (SRAM) that matches headers that are in the packet to the operation, and then forwards them onto egress match action where packet is being matched for the action to be performed on it, for example drop or send. Deparser then recombine data from the packet back into each packet, before storing it in buffer or in a queue. One of the components of ASIC in Barefoot switch is the P4 compiler, which generates the API that the control plane uses to communicate with the data plane.

The switch comes with a ONIE bootloader [62], that was utilized to install Ubuntu 16.04 LTS server operating system. The second component that was required was the software development environment that was loaded on top of the operating system. The steps to install software development environment were provided by the vendor along with the installer. Once those steps have been completed, user can write a P4 program and load it onto the device, run a compiler to compile it and then test it.

The device had multiple components that can be reconfigured by user via CLI. An example of such is ports: in order for a port to work, a user has to switch to port management CLI, then create the port and enable it; with each port being configured individually. An example of port setup is shown on Figure 4.11, with ports 1-4 added.

```
bf-sde> pm
bf-sde.pm> port-add 1/0 10G none
bf-sde.pm> port-add 2/0 10G none
bf-sde.pm> port-add 3/0 10G none
bf-sde.pm> port-add 4/0 10G none
bf-sde.pm> port-enb 1/0
bf-sde.pm> port-enb 2/0
bf-sde.pm> port-enb 3/0
bf-sde.pm> port-enb 4/0
bf-sde.pm> show
-----+---+---+----+------+----+---+---+---+---------------+---------------+-
PORT |MAC |D_P|P/PT|SPEED |FEC |RDY|ADM|OPR|FRAMES RX      |FRAMES TX      |E
-----+---+---+----+------+----+---+---+---+---------------+---------------+-
1/0  | 3/0| 48|0/48| 10G  |NONE|YES|ENB|DWN|              0|              0|
2/0  | 2/0| 52|0/52| 10G  |NONE|YES|ENB|UP |              0|              0|
3/0  |63/0|444|3/60| 10G  |NONE|YES|ENB|DWN|              0|              0|
4/0  |62/0|440|3/56| 10G  |NONE|YES|ENB|DWN|              0|              0|
bf-sde.pm>
```

Figure 4.11: An example of adding ports on the port management cli in Tofino.

Once a port is added and enabled, and if the port is physically connected to a compatible source, the port status changes to "UP". A user can then specify the speed of the port at the time of its creation.

Tofino ports are Quad Small Form-Factor Pluggable 28 (QSFP28), designed for 100Gbps applications. It offers four channels of high-speed differential signals with data rates ranging from 25 Gbps up to 100 Gbps. QSFP28 can support 4x10G, 4x25G and 2x50G breakout connections, or 1x100G depending on the transceiver type that is used. The Small Form-Factor Pluggable + (SFP+)10G tunable transceivers[70] that were used in our experimental setup, are shown in figure 4.12. SFP+ 10G transceivers from Finisar[71] were connected on a Spirent packet tester and a Tofino switch. The form-factor required for the Tofino port is QSFP28 but, with a Mellanox Adapter[72] that was used in this setup, standard SFP transceivers can be used as shown in 4.12. The cables for QSFP28 form factor were Direct Attach Copper (DAC) passive cable, however in the experiment we used Lucid Connector / Ultra Physical Contact (LC/UPC) connectors and fibre optical cables.

Figure 4.12: Transceivers, Mellanox adapter and components used for connection between Tofino switch, Lumentum ROADM and Spirent packet tester.

The Lumentum (formerly JDSU) tunable transceiver is a device that converts an incoming electrical signal into a specific optical wavelength, enabling dynamic wavelength selection and efficient signal transmission over optical networks. The transceiver itself requires to be configured to accept specific wavelength. In other words, a channel has to be set with a specified centre wavelength. This will allow the transceiver to specify what channel the incoming signal have to be converted to.

The Tofino switch internal configuration contains 8 multiplexers that are connecting to the cages where the pluggable transceivers are inserted. The diagram of the switch ports connections to internal Inter-Integrated Circuit (I2C) multiplexers is shown in Figure 4.13. This is important because it defines the specific I2C registers for each port to over see the wavelength configuration of the tunable transceiver.

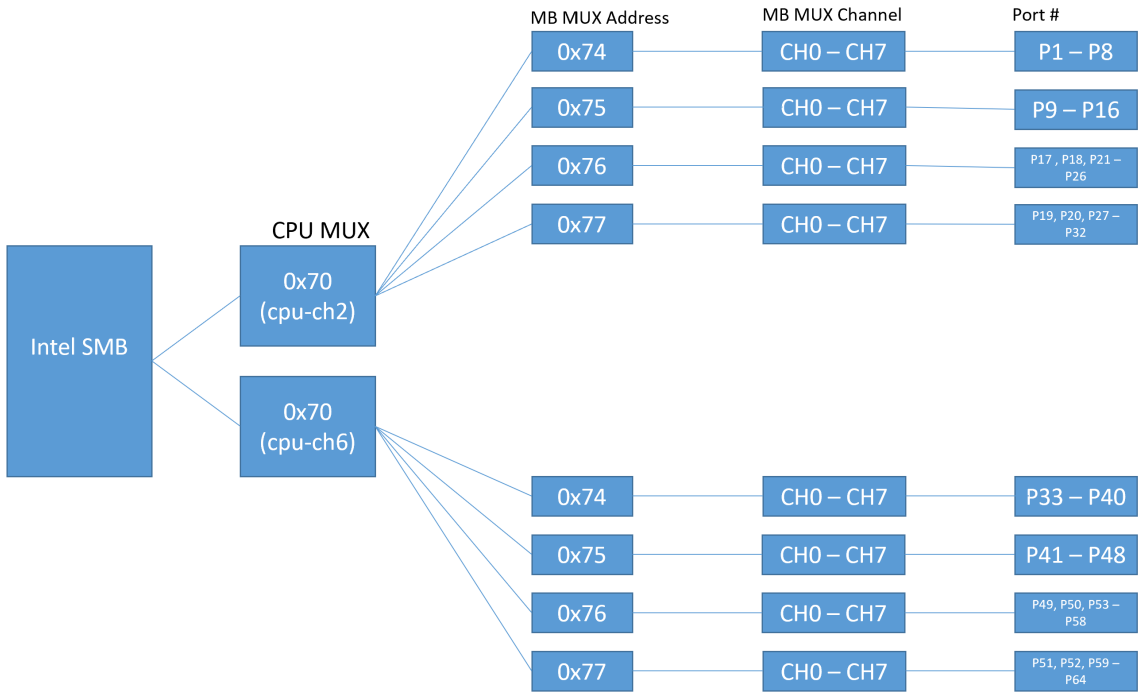

Figure 4.13: Tofino I2C port connections responsible for control of the QSFP28 ports.

To connect via I2C to a port, the commands "sudo i2cset -y 0 0x70 0x40 (for 0x70 (ch6))" and "sudo i2cset -y 0 0x70 0x4 (for 0x70 (ch2))" are required in the Tofino cli itself.

In Figure 4.14 we can see the output of a I2C management view from Tofino switch CLI. The first part is running "modprobe" commands to install I2C drivers; then determine to which port. Access is required, so it runs "i2cset -y 0 0x70 0x4" to access the first 4 multiplexers shown in Figure 4.13 or "i2cset -y 0 0x70 0x40" for second set of multiplexers shown in the same image. Command "i2cdetect -l" allows to scan the interface for devices connected to it. In this case we observe "i2c-0" device. The next command "i2cdetect -y 0" allows us to write to the device, without the interaction from it. We can see in this case that registers "0x50" and "0x51" are occupied, and available to review. Running the command "sudo i2cdump -y 0 0x50" allows us to see the first part of the details of the transceiver (i.e. first page). We also see the serial number, part number and vendor name. To select and view a second page, we run the command "sudo i2cdump -y 0 0x51", on page 2 the register "0x91" is responsible for channel setting of the device. We can observe that it is set to "0x1b", which is a hexadecimal value for channel number 27. This way the signal that arrives to this transceiver is then transmitted over to ROADM via channel 27 with a specific wavelength set as alligned with section 2.1.

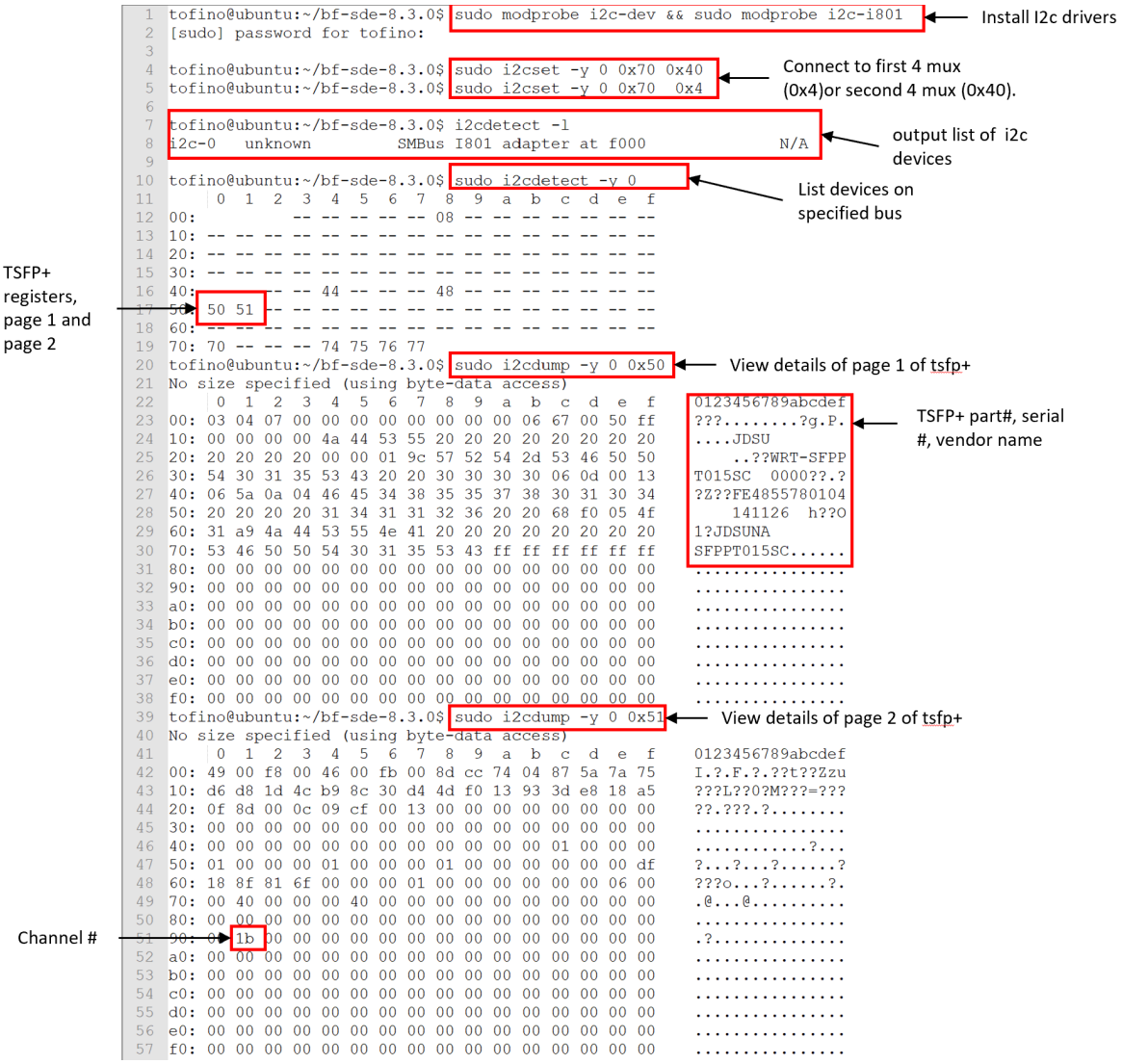

Figure 4.14: Tunable SFP+ information from Tofino cli.

Figure 4.15 reflects the same settings as in the I2C, but in a vendor provided GUI. The limitation of using GUI is that it works with a specific Gryphon board that controls the Tunable Small Form Factor Pluggable + (TSFP+), therefore one is tied to the hardware component to control the transceiver. In this figure, we can observe that we have a few settings that are configurable. Among those are wavelength and channel number. If the transceiver does not match the channel that is set on the ROADM (receiving end), it will drop the signal.

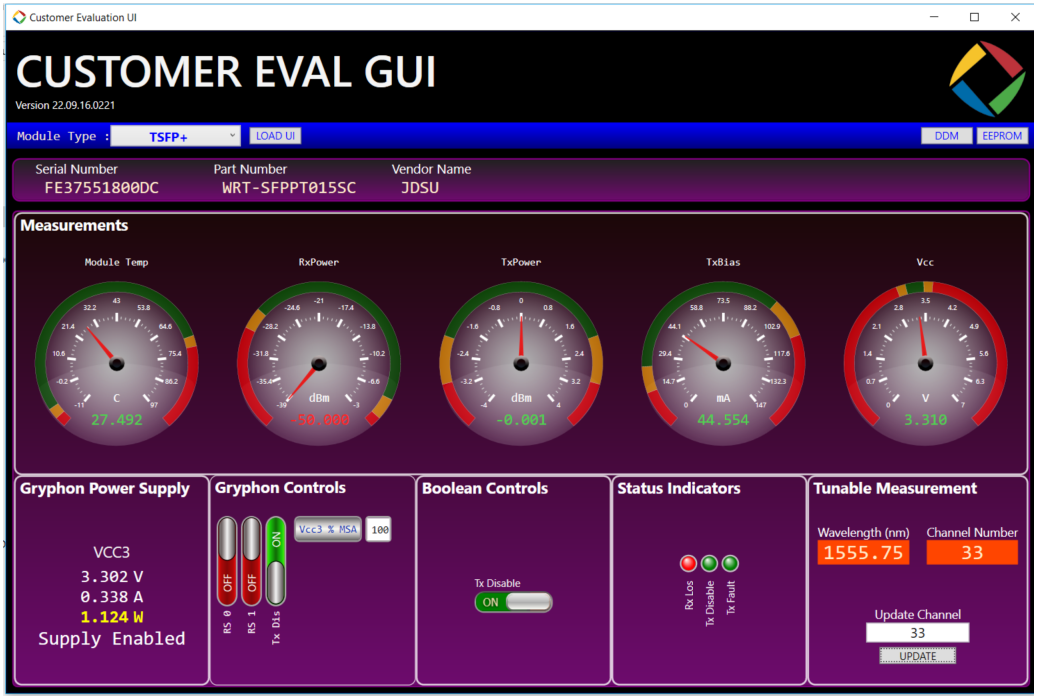

Figure 4.15: TSFP+ vendor provided GUI view, allows to set a channel and wavelength.

Once we confirm that the channel is set correctly and our link is up, we can physically observe that the link has been setup correctly by looking at the Light Emitting Diode (LED)s at each of the ports on Tofino. There are 4 LEDs that light up with the colors that are on when the link is up,and it will reflect which bandwidth is set up, for example 10Gb is purple.

4.2.2 Programming Protocol-Independent Packet Processors (P4)

P4 is a domain specific language designed to program the data plane of programmable switches. One of the device compatible with P4 is Barefoot Tofino ASIC. P4 is a descriptive language that provides flexibility to the operator, as it is designed to be used on variety of software and hardware devices. It allows the operator to define how and where packets are travelling through the device. The main components of P4 language are parser, ingress control flow, egress control flow and deparser. From a user point of view, the required items are P4 program and metadata table.

The goal of our bespoke P4 program is to ensure that if a frame comes into port A, the switch is aware that it must send it out via port B and vice versa. It is a simple routing table definition. In a P4 code, this is shown in Figure 4.16.

```
/* -*- P4_14 -*- */
#ifdef __TARGET_TOFINO__
#include <tofino/constants.p4>
#include <tofino/intrinsic_metadata.p4>
#include <tofino/primitives.p4>
#else
#error This program is intended to compile for Tofino P4 architecture only
#endif

/*******  H E A D E R S  **********/
header_type ethernet_t {
        fields {
          dstAddr : 48;
          srcAddr : 48;
          etherType : 16;
        }
}

/****  M E T A D A T A  *******/

/****  P A R S E R ******/
parser start{
        return  ingress;
        }
/** I N G R E S S   P R O C E S S I N G  **/
action forward(port) {
        modify_field(ig_intr_md_for_tm.ucast_egress_port, port);
    }

table ing_port {
    reads {
        ig_intr_md.ingress_port : exact;
  }
    actions {
        forward;
    }
        size : 256;
}
control ingress {
        apply(ing_port);
}
/* E G R E S S   P R O C E S S I N G  */
control egress {
}
```

Figure 4.16: P4 Code that was written for the experiment.

The code in the figure has the following components:

- The “Include” statements define the core library and specific modules required for this use case. For example “Intrinsic-metadata” is a metadata provided by the architecture associated with each packet (e.g. the input port where a packet has been received).
- “Header” contains packet definition fields and can be expanded to include specific fields of network protocols. In this thesis the specifics of a type of a packet wasn’t

important, therefore it was left with basic definitions.

- "Metadata" is intermediate data generated during execution of a P4 program.
- "Parser" describe the permitted sequences of headers within received packets, how to identify those header sequences, and the headers and fields to extract from packets. In our case, it simply returned the input packet value.
- "Ingress processing" block defines the functions how the data has to be modified and/or processed. In our code, we specified that the field needs to be modified, any egress port needs to have a value of variable 'port'. This variable is specified in the table 'ing-port' entry, defined in table, and shown how this table is populated in Figure 4.17. We also specify an action, this is a simple forwarding action to send the packet out to its destination via specified ports.
- "Egress processing" can choose to drop the packet instead of sending it to the port chosen earlier, but it cannot change the choice to a different port. Ingress code is the most common place in a P4 program where the output port(s) are defined.

Figure 4.17 illustrates the table entries we made on the Tofino switch. The entries here are specifying that anything coming into port 48 will egress to ("action port") port 52 or 444.

```
1 pd ing_port add_entry forward ig_intr_md_ingress_port 48 action_port 52
2 pd ing_port add_entry forward ig_intr_md_ingress_port 48 action_port 444
```

Figure 4.17: Tofino table entries, where each ingress port has matching egress port specified

4.2.3 Summary

In this Chapter, we detailed the description of devices connections in both optical and physical domain. We described the architecture of the communication setup between physical instruments and software components. We described both novelty equipment used and legacy instruments. Chapter 5 will bring together the development from chapters 3 and 4 into an experimental set up to validate the proposed architecture of this thesis.

The work described in this chapter is detailed in the following outputs:

- S. Ahearne, Y. Verbishchuk, C. Sreenan and F. Gunning, "Software Defined Control of Tunable Optical Transceivers Using NETCONF and YANG," 2018

European Conference on Networks and Communications (EuCNC), Ljubljana, Slovenia, 2018, pp. 1-86, doi: 10.1109/EuCNC.2018.8443188.[24]

- Y. Verbishchuk, C. Sreenan, and F. Gunning, "Configuration and monitoring of the optical physical layer using software-defined tools," in OSA Advanced Photonics Congress (AP) 2019 (IPR, Networks, NOMA, SPPCom, PVLED). Optical Society of America, 2019, p. NeT1D.2. Available: http://opg.optica.org/abstract.cfm?URI=Networks-2019-NeT1D.2 [25]

Chapter 5

Practical Demonstration and Evaluation of Network Control

5.1 Prototyping network system control

Currently, telecommunications networks contain both electrical and optical equipment to provide smooth, reliable and fast data transmission for long and short distances, as introduced in Chapter 3. An example of such architecture can be seen in Figure 5.1. The physical equipment (optical/electrical) in the network has limitations in respect to scalability, component complexity, data loss, bandwidth, data rates etc. In this thesis, one of the main objectives is to simplify the network architecture from Figure 5.1, merging functionalities of the opto-electronic equipment, providing for reconfigurability end-to-end, and potentially minimizing both latency and costs. An Ethernet packet/frame or data transfer from layer 3 to layer 1, normally requires a transponder, or a router to merge to the fast optical lane, converting and converging data streams, through modulated optical WDM carriers. Both domains work well with the use of an intermediary device to allow for translation of a signal from electrical to optical, and vice-versa. This intermediary device is normally a transponder. A transponder accepts grey optical signals from the packet domain, which then converts them into colored optical channels (or carrier signals), and converges other signals which are then sent to the line side of the optical transport network, typically through a ROADM. Transponders are expensive devices, and they will generally work in tandem with equipment from the same vendor, and work on a "set and forget" basis. This means that the device is configured with a static set of rules that are not dynamically reconfigured.

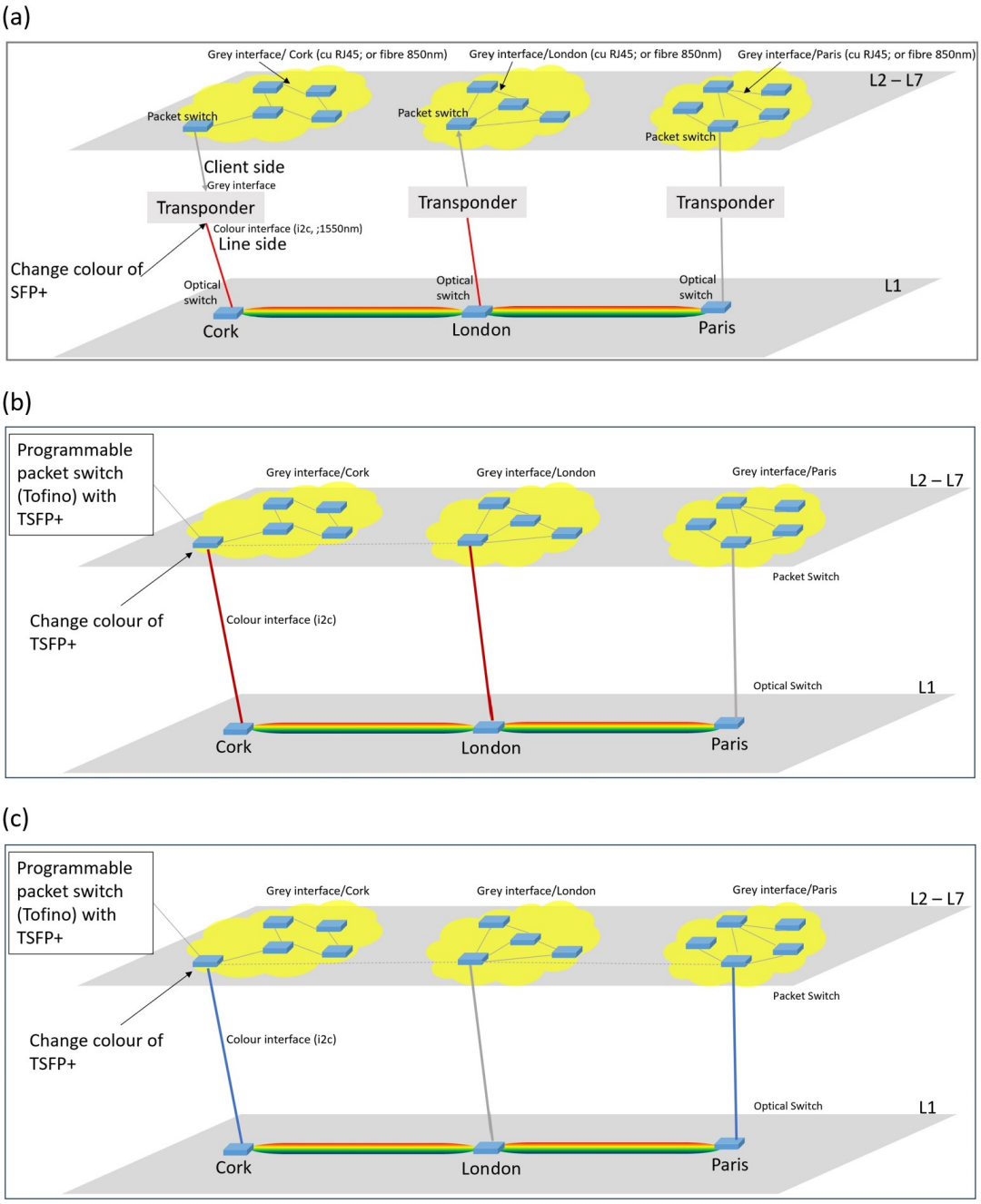

Figure 5.1: Proposed network concept. This figure (a) is a simplified example of a typical transport network, for example, from Cork to Paris via London. From (c) to (b) the wavelength of the transceiver in the programmable switch changed to accommodate scenarios such as wavelength contention. In the experiment, Ethernet switches and transponders were replaced with a programmable switch accommodating tunable transceivers.

Figure 5.1 illustrates the proposed concept, where we replace the common switches and transponders with a programmable switch that accommodates tunable transceivers. We call it a packetponder [27]. In this scenario, the client interface remains unchanged, either with "grey" optical or colored optical links, but it provides flexibility on the node configuration and wavelength assignment.

In this thesis we propose a simplification of the physical network from Figure 5.1a to Figure 5.1b, where the transponder is removed and the packet switch from Figure 5.1a is replaced with a programmable switch that accommodates high bandwidth tunable transceivers of common form factor. Aggregation from client side is still through the switch, utilizing RJ45 or grey optics or even colored optics, to a QSFP28 (or SFP+, as per Chapter 4 in our experiment). However, this same switch with high capacity transceivers can be connected to an add-port from a ROADM with tunable transceiver. Figure 5.1c shows the potential to change the same transceiver wavelength to reuse the signal to a different destination, as an example.

5.2 Experimental demonstration

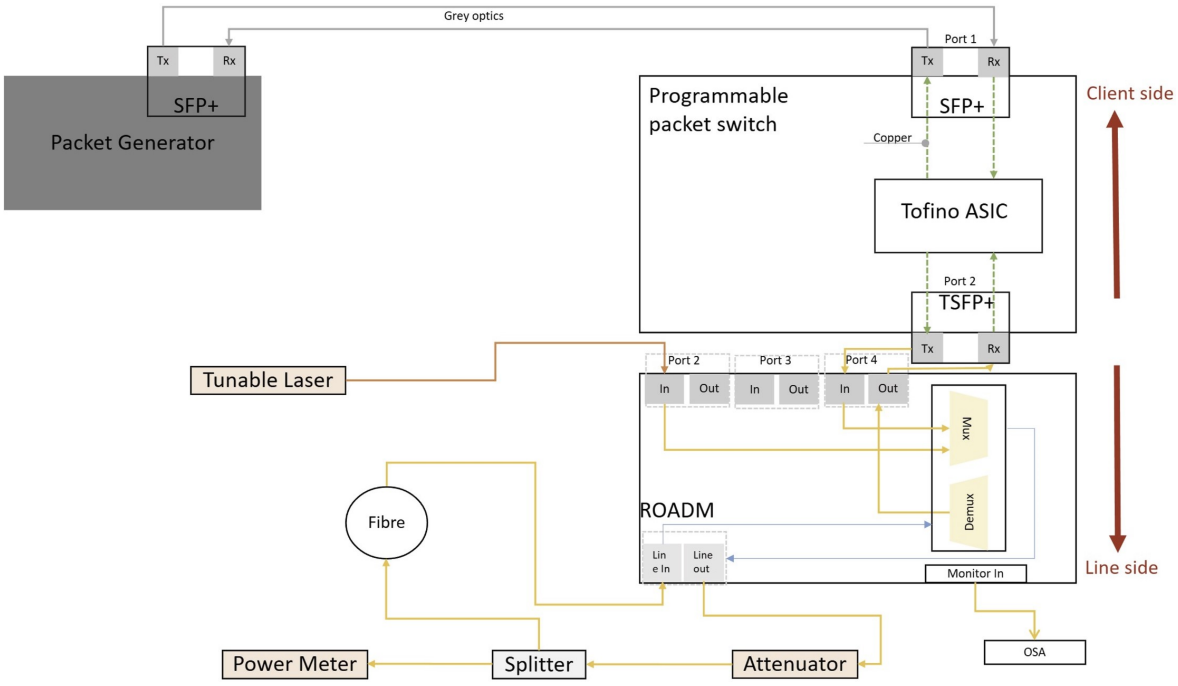

Figure 5.2: Experimental demonstration of the proposed network.

Figure 5.2 shows the actual experimental setup used to emulate the proposed scheme from Figure 5.1b and c.

In order to emulate data stream from the client interface, a packet generator with a

Table 5.1: Table of components used in the experiment.

Instrument	Vendor and Part Number
Packet generator	Spirent Test Center SPT-2000A-HS (chassis CV-10G-S2)
SFP+ transceiver	Finisar FTLX8571D3BCL
Mellanox adapter	Mellanox SFP to QSFP28 adapter, MAM1Q00A-QSA28
Tunable SFP+	JDSU WRT-SFPPT015SC
ROADM	Lumentum WB-RDM-3200-AC

transceiver at fixed wavelength of 1555.75nm was used and linked to a similar SFP+ transceiver via a Mellanox adapter to port 1 of a programmable packet switch (Tofino). For these transceivers the data rate was 10Gb/s. As with any other switch, the data is converted from optics to electronics (opto-electronic conversion), and in our case routed to port 2 via the Tofino ASIC switch fabric. Port 2 also contains another tunable SFP+ transceiver as described in table 5.1. The operation in the ASIC is as follows: when the data signal reaches port 1 in the Tofino switch, its ASIC then has to forward the data according to its routing table that was defined in a P4 program, as described in Chapter 4. The program simply states that, if anything comes into port 1 it then forward its packets via port 2. The behaviour described for the Tofino ASIC differs from standard L2/L3 switching in that the forwarding logic is not hard-coded into the device but is instead defined by a bespoke P4 program. Traditional switches and routers implement fixed pipelines where functions such as VLAN tagging, IP forwarding, and access control are pre-determined by the vendor and only configurable within predefined limits. In contrast, P4 allows the programmer to redefine the parsing, matching, and action stages of the pipeline itself, enabling support for new protocols, customized header formats, or non-standard processing behaviours that cannot be achieved with conventional devices. For example, while a traditional switch can only manipulate VLAN tags in the way its firmware allows, a P4-programmed device can define entirely new tagging schemes or metadata fields for use in traffic engineering or network slicing. Thus, the innovation lies not in reproducing existing L2/L3 capabilities, but in providing the programmability to extend, modify, or replace them according to experimental or application-specific requirements, so providing a true packet over WDM. The Tofino switch has to forward the signal strictly following to the defined routing rules in this example. This scheme replaces the need of a transponder, specially that high capacity switches and tunable transceivers are constantly appearing in the market, with significant innovations for 400G+ per switch port. (Recent work shows that coherent pluggable modules (e.g., 400ZR/ Open ZR+) are extending the reach well beyond the short-reach 80–120 km domain, indicating that the gap between switch-port optics and full ROADM/transponder systems is narrowing [73], [74].) The transceiver

from port 2 is then connected to the ROADM add-drop port 4. Simultaneously, in order to emulate congestion and noisy transmission, a single wavelength CW laser was connected to add-drop port 2 of the ROADM, and multiplexed with the signal under test from add-drop port 4 (MUX) within the ROADM circuitry. The output of the multiplexed signals from the ROADM circuitry (line out) was then transmitted over fibre (80km SMF-28), while emulating further power through an optical "attenuator". A 90/10 splitter was used after the attenuator to monitor the optical power launched to the fibre. In a real scenario, a second ROADM, switch and tunable transceiver would be needed to receive the signal at the destination. However, due to the limited resources, the output of the fibre was looped back to the same ROADM (line in), where the signals were de-multiplexed and detected all the way back to the packet generator, where frame or packet loss could be monitored.

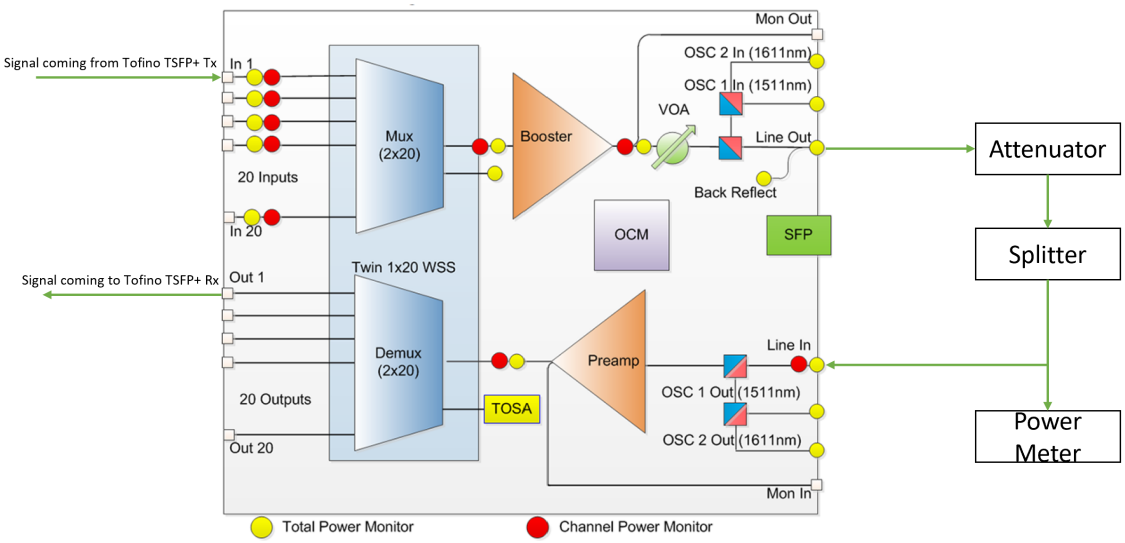

Figure 5.3: Schematic diagram of the ROADM as in Figure 4.2, adding the optical carriers and experimental instruments.

Experimentally, we can monitor for the incoming data from the packet generator at port 1 of the Tofinos own CLI. Figure 5.4 shown as example of the data monitoring, in port 1/0 (top) with no frames received, and in port 1/0 (bottom) with 1305038 frames received once the packet generator started sending data. In this Figure, we can also observe port 2/0 behavior/environment where the ASIC switches the packet from port 1/0 (received) to port 2/0 (transmitted) and also when the packets arrive back at the fibre transceiver (port 2/0 received). This is a useful monitoring tool for port statistics health check.

```
bf-sde> show
PORT    MAC   D_P      P/PT     SPEED   FEC    RDY    ADM    OPR    FRAMES RX      FRAMES TX    E
-------------------------------------------------------------------------------------------------
1/0     3/0   48       0/48     10G     NONE   YES    ENB    UP            0              0
2/0     2/0   52       0/52     10G     NONE   YES    ENB    UP            0              0
bf-sde> show
PORT    MAC   D_P      P/PT     SPEED   FEC    RDY    ADM    OPR    FRAMES RX      FRAMES TX    E
-------------------------------------------------------------------------------------------------
1/0     3/0   48       0/48     10G     NONE   YES    ENB    UP      1305038        1109043    *
2/0     2/0   52       0/52     10G     NONE   YES    ENB    UP      1109585        1301067    *
bf-sde>
```

Figure 5.4: Tofino CLI showing frames received and transmitted on both ports.

Using the packet generator GUI, we can monitor number of frames sent and received, and other statistics. Figure 5.5 shows a table view of the packet generator GUI with the total counts for frames sent to Tofino and received back. In this case 14 million frames in one minute were sent from "client side" to "line side", and thus were all received without errors.

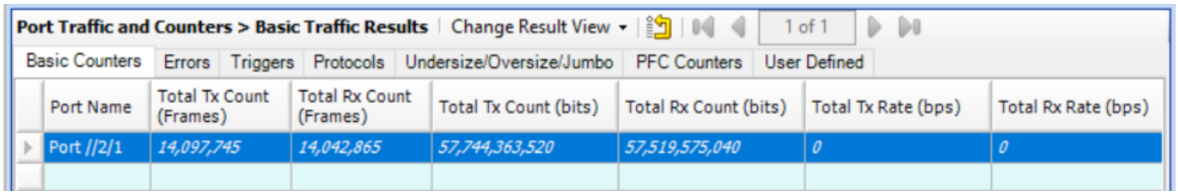

Figure 5.5: Spirent GUI showing traffic transmitted and received (Tx and Rx).

5.3 Full wavelength tuning reconfiguration - Packetponder

The final experiment to test the proposed architecture was to tune the SFP+ on the line side, having the client side wavelength static. Figure 5.8 shows the Tofino CLI port management showing the frames count per port. As before, port 1/0 is the client side; port 2/0 is the line side. Port 1/0 transceiver wavelength remain the same, Figure 5.8 (top) shows ports when no signal is transmitted from the client side. Figure 5.8 (middle) shows as per Figure 5.4 all frames received at port 1/0 from the client side, transmitted through port 2/0 to the line side, received at port 2/0 from line side and sent back to port 1/0 - client side with minimal frame loss (< 10000). Figure 5.8 (bottom) shows the loss when the wavelength of the transceiver in port 2/0 was changed to 1555nm. It shows no penalty in terms of frame loss rate, all data was received at the client side, even though the client side wavelength remained the same. This is a huge achievement, showing that it is possible to have a static configuration from client side but flexible at line side, opening up the possibility for efficient bandwidth allocation,

which is critical for today's networks. This was possible with an online and live data streams running through the system and a simple software control command that changed the wavelength of the transceiver, and the center wavelength of the ROADM simultaneously.

```
bf-sde> port-stats-clr 2/0
bf-sde> port-stats-clr 1/0
bf-sde> show

PORT    IMAC/PI/SPEED  IFEC    IRDY    AIMD    ORI    FRAMES RX      FRAMES TX    E
-------------------------------------------------------------------------------
1/0     481/0/10G      NONE    YES     ENABLED  UP           0              0     E
2/0     521/0/10G      NONE    YES     ENABLED  UP           0              0     E

PORT    IMAC/PI/SPEED  IFEC    IRDY    AIMD    ORI    FRAMES RX      FRAMES TX    E
-------------------------------------------------------------------------------
1/0     481/0/10G      NONE    YES     ENABLED  UP    240363531     237759561    E
2/0     521/0/10G      NONE    YES     ENABLED  UP    239816331     239897414    E

PORT    IMAC/PI/SPEED  IFEC    IRDY    AIMD    ORI    FRAMES RX      FRAMES TX    E
-------------------------------------------------------------------------------
1/0     481/0/10G      NONE    YES     ENABLED  UP    240363531     237759561    E
2/0     521/0/10G      NONE    YES     ENABLED  UP    239816331     239897414    E

bf-sde>
```

Figure 5.6: Tofino CLI showing traffic transmitted frames and received frames on both ports.

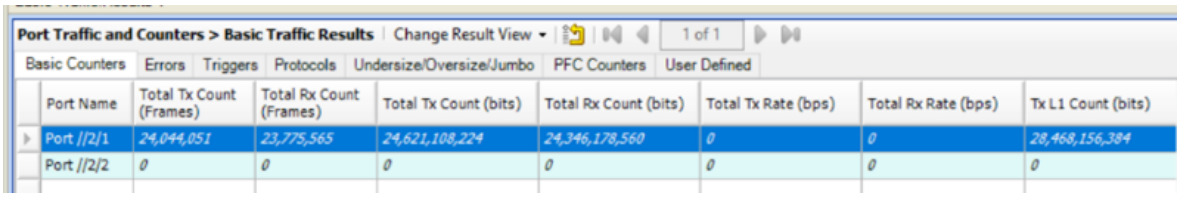

Figure 5.7: Spirent showing results of a transmitted and received frames with offset channel in ROADM.

Further evidence is shown in Figure 5.9 from Spirent GUI, where the Tx frames in the client side and the Rx frames (client side but travel through line side when wavelength change) shows virtually no frame loss.

A more advanced demonstration of the concept was shown at OFC 2023 [27], by colleagues at Tyndall, where better tools were then available for total reconfiguration, we called it a Packetponder.

```
bf-sde> port-stats-clr 2/0
bf-sde> port-stats-clr 1/0
bf-sde> show

PORT    IMAC/PI/SPEED  IFEC    IRDY    AIMD    ORI    FRAMES RX      FRAMES TX    E
-------------------------------------------------------------------------------
1/0     481/0/10G      NONE    YES     ENABLED  UP           0              0     E
2/0     521/0/10G      NONE    YES     ENABLED  UP           0              0     E

PORT    IMAC/PI/SPEED  IFEC    IRDY    AIMD    ORI    FRAMES RX      FRAMES TX    E
-------------------------------------------------------------------------------
1/0     481/0/10G      NONE    YES     ENABLED  UP     240363531      237759561    E
2/0     521/0/10G      NONE    YES     ENABLED  UP     239816331      239897414    E

PORT    IMAC/PI/SPEED  IFEC    IRDY    AIMD    ORI    FRAMES RX      FRAMES TX    E
-------------------------------------------------------------------------------
1/0     481/0/10G      NONE    YES     ENABLED  UP     240363531      237759561    E
2/0     521/0/10G      NONE    YES     ENABLED  UP     239816331      239897414    E

bf-sde>
```

Figure 5.8: Tofino CLI showing traffic transmitted frames and received frames on both ports.

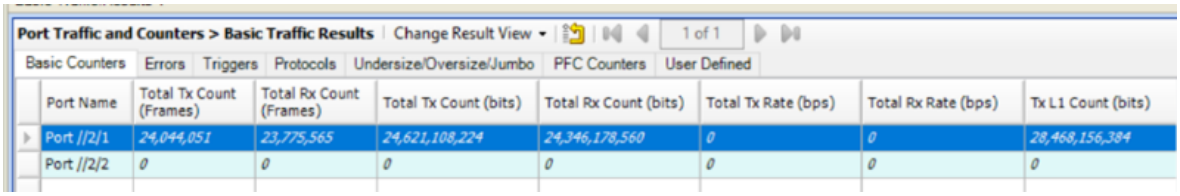

Figure 5.9: Spirent showing results of a transmitted and received frames with offset channel in ROADM.

Further evidence is shown in Figure 5.9 from Spirent GUI, where the Tx frames in the client side and the Rx frames (client side but travel through line side when wavelength change) shows virtually no frame loss.

We can further observe the difference in signals with a 0dB or 19dB attenuation in Figure 5.10, by observing the spectra. The spectrum shows wavelength (x-axis) and optical power density (y-axis) indicating the peak wavelength at 1555.75nm as expected. The "pink" trace shows the signal with no attenuation, and the yellow trace with a 19dB attenuation. Note that not only the signal power (centre wavelength power) is reduced when adding 19dB of attenuation, but also unintended consequence was that the OSNR has also changed from 32dB to 10dB, reducing the gain of the transmission link. Note that 19dB attenuation is equivalent of 95 km of standard single mode fibre (0.2 dB/km attenuation).

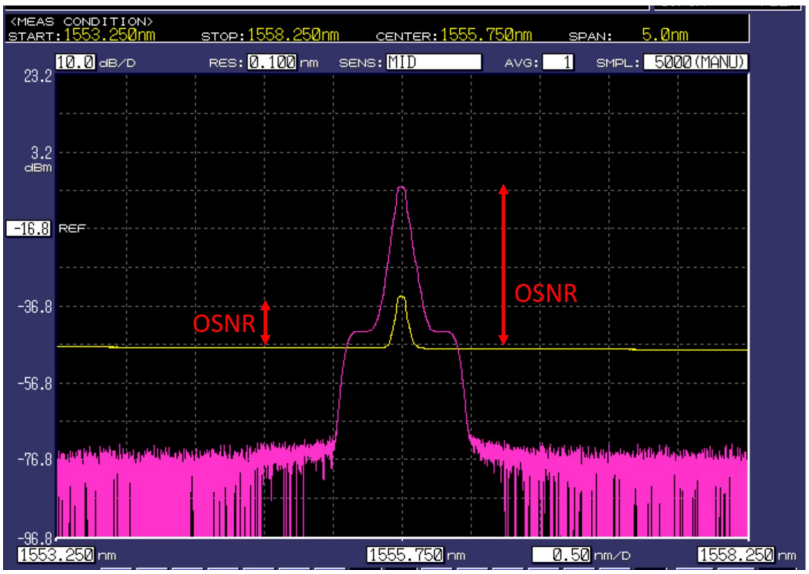

Figure 5.10: Spectra of the signal with 0dB attenuation (pink), and 19dB attenuation (yellow).

Here in this experiment, we ensured that the line-side equipment was managed centrally via ONOS, while the packet frame transmission used the packet generator proprietory software (of course, in the future, open platforms would be of benefit, but for the exercise in this thesis, we are showing that disaggregated optical devices can be controlled / managed centrally). The laser, power meter and attenuator were connected via ONOS, as explained in Chapter 3, and its topology is shown in Figure 5.11 and also through the ONOS CLI as in Figure 5.12.

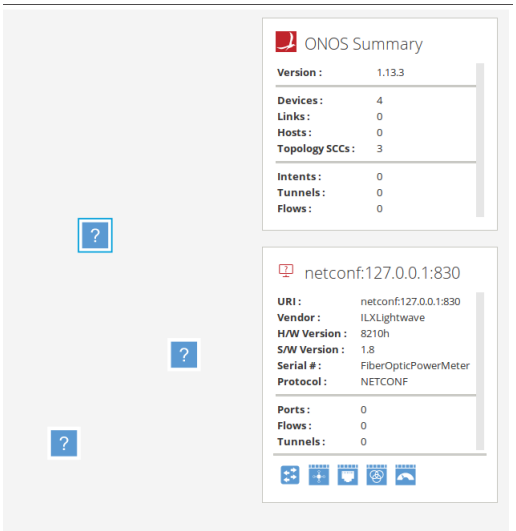

Figure 5.11: Topology view of devices connected power meter laser and attenuator to ONOS controller. The question mark icons are the 3 devices connected. However as they are new and unknown type to the controller they are presented as question mark icons.

```
onos> devices
id=netconf:127.0.0.1:830, available=true, local-status=connected 2s ago, role=MASTER, type=OTHER, mfr=ILXLightwave, hw=8210h, sw=1.8,
serial=FiberOpticPowerMeter, chassis=0, driver=powermeternetconf, ipaddress=127.0.0.1, locType=none, name=netconf:127.0.0.1:830, port=830,
protocol=NETCONF
id=netconf:127.0.0.2:830, available=true, local-status=connected 36s ago, role=MASTER, type=OTHER, mfr=Agilent Multiport Tunable Laser
Source, hw=N7714A, sw=1.13.1, serial=Multiport Tunable Laser Source, chassis=0, driver=laser, ipaddress=127.0.0.2, locType=none,
name=netconf:127.0.0.2:830, port=830, protocol=NETCONF
id=netconf:127.0.0.3:830, available=true, local-status=connected 3m19s ago, role=MASTER, type=OTHER, mfr=Agilent, hw=8164B, sw=4.5,
serial=Lightwave Measurement System, chassis=0, driver=lms, ipaddress=127.0.0.3, locType=none, name=netconf:127.0.0.3:830, port=830,
protocol=NETCONF
```

Figure 5.12: ONOS CLI output of command "devices",showing list of connected devices such as power meter, laser and Lightwave Measurement System part of which is attenuator used in this experiment.

At the time this experiment was carried out in (2017-2019), ONOS was not advanced enough to detect the active connections in the ROADM and we had to set those connections up in ONOS manually. With regard to power meter, laser and attenuator those devices were connected and managed as per Chapter 3, utilizing Netopeer2, Sysrepo and ONOS.

For the purpose of this experiment, the process to ensure communications with the instrumentation were the following:

- ROADM; via vendor provided Lumentum Multi Node Management GUI was used to make changes to the settings of ROADM.
- Tofino; via Putty CLI, allowed access to the Tofino to monitor port statistics and compiling and running P4 program to set routing rules for the routing table.
- Attenuator, power meter, laser; via Sysrepo and Netopeer2 communicated with ONOS, for the monitoring of the network and performance testing.

5.4 Quality of Experience - power impairments validation

As the proposed experiment of an emulated network was working as expected, we were able to make measurements in terms of frame loss or frame loss ratios, which are quantitative measures of the quality of the signal. Frame loss ratio is defined as a ratio, expressed as a percentage of the number of frames not delivered, divided by the total number of frames during time interval T, where the number of frames not delivered is the difference between the number of frames arriving at the ingress interface flow point to be delivered to the egress interface flow point and the number of frames delivered at the egress interface flow point in a point-to-point interface connection or multi-point interface connectivity [75].

As the result of this emulation we discovered that attenuation set to 18dB, as shown in Figure 5.13 is causing the signal to lose frames which is consistent with industry

standards for 10Gbit/s signal and 80km of standard single mode optical fibers. This number is also consistent with the TSFP+ technical specifications.

Figure 5.14 illustrates the collected results of transmitted and received frames for both the Tofino and the Spirent packet switch.

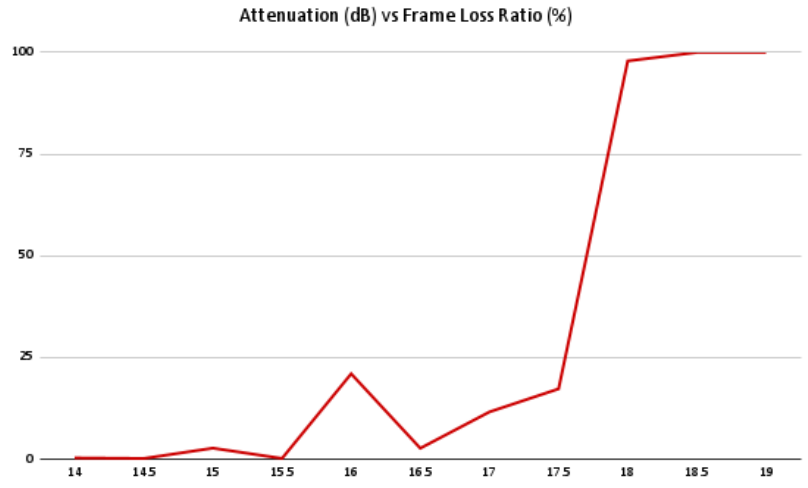

Figure 5.13: Attenuation vs Frame Loss Ratio, Frame size = 512 bytes, Time = 1 Min

Acceptable packet loss is dependent on what data is sent. Typically, real-time applications demonstrate significant quality loss due to small amounts of packet loss. But, in cases of file transmissions, like web pages or downloads, dropped packets are typically recalled, and packet loss rates can be higher than real-time applications without showing noticeable quality loss.

	Tofino				Spirent			
Attenuation (dB)	Port 1Tx	Port 1 Rx	Port 2 Tx	Port 2 Rx	Port Tx	Port Rx	Frames Lost	Frame Loss Ratio %
14.5	14,070,287	14,096,588	14,071,567	14,071,500	14,097,745	14,070,287	27,458	0.195
15	13,712,697	14,096,880	14,077,119	14,060,437	14,097,745	13,712,697	385,048	2.731
15.5	14,068,625	14,097,545	14,092,370	14,091,263	14,097,745	14,068,625	29,120	0.207
16	1,135,467	14,095,499	14,045,927	13,897,120	14,097,745	11,135,467	2,962,278	21.012
16.5	13,716,279	14,097,118	14,083,370	14,066,407	14,097,745	13,716,279	381,466	2.706
17	12,456,953	14,097,080	14,081,592	14,004,691	14,097,745	12,456,953	1,640,792	11.639
17.5	11,658,189	14,095,743	14,052,289	13,937,304	14,097,745	11,658,189	2,439,556	17.305
18	296,530	14,097,142	948,287	898,652	14,097,745	296,530	13,801,215	97.897
18.5	2	14,097,417	139	116	14,097,745	2	14,097,743	100.000
19	-	14,097,147	76	58	14,097,745	-	14,097,745	100.000

Figure 5.14: Comparative table of frame loss ratio vs attenuation from the Tofino CLI monitoring and Spirent packet generator measurements.

5.5 Additional system components

The experiment shown here required significant studies, engagement and development of new tools/architectures from Layer 1 to Layer 3, many of which were not well

developed or not even existed. For this reason, it created many challenges which had to be overcome in order to see the successful outcome as in Figure 5.14.

The first challenge was related to the disaggregated/legacy devices and their physical port connections. The details of overcoming this issue , assisted with Sysrepo/Netopeer2/ONOS is extensively discussed in Chapter 3. Another challenge was a multitude of operating systems used to work with. Spirent Test center (packet generator) is a device that has been around for long time, and required Windows 7 operating system to be run on, while all other software were running in a virtual machine running Linux Ubuntu 16.04 operating system. As a result, we required the use of two management laptops, one to control Spirent packet generator and the second laptop to control all other components. Another important aspect was the fact that, at the time, Tofino ports form factor was QSFP28, and we had to use Mellanox adapters in order to match the port form factor with SFP+ transceivers. At the time the experimental analyses of this thesis were carried out, there was no commercially available tunable QSFP28, and this was the reason why tunable SFP+ transceivers were used instead.

While designing and planning the experimental setup, we used several tools to allow us to monitor and observe. The experiment, allowing us to detect any issues or failures. These included the Spirent GUI, OSA, ROADM GUI, Tofino CLI, ONOS CLI etc., which in the future a single platform for control and monitoring would be ideal.

5.6 Summary

We have successfully emulated a client side and line side network in a lab and demonstrated the possibility of a network being controlled with SDN tools. We demonstrated the use of tunable SFP+ with a programmable packet switch shown the potential for our packetponder to manage bandwidth allocation. This experiment demonstrated the use of SDN tools to control and manage legacy dissagregated devices along with new cutting edge devices to create a functional network transmission.

The work described in this chapter is detailed in the following outputs:

- Y. Verbishchuk, C. Sreenan, and F. Gunning, “Configuration and monitoring of the optical physical layer using software-defined tools,” in OSA Advanced Photonics Congress (AP) 2019 (IPR, Networks, NOMA, SPPCom, PVLED). Optical Society of America, 2019, p. NeT1D.2. Available: http://opg.optica.org/abstract.cfm?URI=Networks-2019-NeT1D.2 [25]
- F. C. G. Gunning, A. Kaur, N. C. Estrada, J. Raulin, Y. Verbishchuk and C.

J. Sreenan, "Enabling High Capacity Flexible Optical Networks," 2021 International Conference on Optical Network Design and Modeling (ONDM), Gothenburg,Sweden, 2021, pp.1-3, doi:10.23919/ONDM51796.2021.9492503.[26]

- J. Raulin, G. Davey, Y. Verbishchuk, A. Jeffries, C. J. Sreenan and F. C. Garcia Gunning, "A programmable Ethernet transport Packetponder using common compact form factor pluggable tunable transceivers to support novel DWDM architectures," 2023 Optical Fiber Communications Conference and Exhibition (OFC), San Diego, CA, USA, 2023, pp. 1-3, doi: 10.1364/OFC.2023.Tu3D.6. [27]
- 2023, J. Raulin, G. Davey, Y. Verbishchuk, A. Jeffries, C. J. Sreenan, and F. C. G. Gunning, "Packetponder with open-source management system for future packet-optical networks," in Proceedings of the 2023 Networks Conference. Optical Publishing Group, 10 2023, p. NeM3B.3. [28]

Chapter 6

Conclusion

6.1 Conclusion

The primary motivation behind this thesis was to leverage the power of SDN for the management and control of legacy networking elements, alongside the integration of next-generation networking tools. Our objective was to span across the Physical, Data Link, and Networking Layers of the OSI stack, enabling the manipulation and management of wavelengths and packets based on various criteria trough optics over IP.

Throughout this work, we have demonstrated the effective utilization of SDN tools for the management of legacy devices and their seamless integration into a network that spans both optical and electrical domains. This demonstration was exemplified through practical testing with a programmable switch, with packet forwarding rules provided by the P4 language.

Chapter 3 shown us a comprehensive demonstration on how to implement SDN tools with NETCONF and YANG for non-standard devices, with detailed insights into the architectural setup of software connections. Challenges encountered during the software development phase were presented, including the unavailability of some ONOS elements. Additionally, we addressed the challenge of manipulating legacy devices using software, which we resolved by creating a "C"-drivers for each of the devices. This work showcased the ability to control legacy devices using new SDN techniques, marking a significant step toward the separation of control and management planes within network devices.

Chapter 4 shown us the details how SDN operates well with devices already SDN-enabled and added optical elements to it.

Chapter 5 shown us how the proposed architecture could work in reality as we concluded experimentally a network comprising both client-side components that remain static and line-side components that offer the flexibility for modification. The results obtained serve as a testament to the effectiveness of the combination of tools and methods employed, which enable the same level of availability and quality of service as observed in current optical networks. We shown for the first time the potential for centrally managing wavelength tuning for better bandwidth allocation without compromising the client-side interface.

These findings highlight the potential of the methods used for controlling simultaneously both packets and wavelengths within the network. Notably, they underscore the feasibility of disaggregating each network component, allowing for the creation of a network that can be tailored to meet specific requirements and demands. Moreover, this approach is not only feasible but also cost-effective, as it accommodates the utilization of both next-generation and well-established common network instruments.

In essence, the results obtained in Chapter 5 validate the viability of the project's goals and demonstrate the potential for a network infrastructure that is highly adaptable, customizable, and aligns with the evolving needs of modern network environments. This conclusion reiterates the project's core objectives of enhancing network flexibility and efficiency through the strategic utilization of SDN and related tools and methods.

At the outset of this project, our ambitions were high, with the belief that the complete separation of software from hardware was attainable. However, it became evident that this endeavor would involve significant complexity, especially considering the state of tools available in 2018 when this work commenced. The challenges faced during this journey included the absence of software drivers for certain devices and the need to craft bespoke drivers for each device. The transition from proprietary/closed systems to open/disaggregated forwarding control planes is a formidable challenge that continues to drive innovation.

Our work exemplified the potential to transition from legacy equipment to next-generation hardware, ensuring that networks can evolve while providing uninterrupted service. This endeavor was achieved through the use of open-source tools, offering network operators the flexibility to adapt and rewrite software to suit their specific requirements.

This project has laid the foundation for a future in which the seamless coexistence of legacy and cutting-edge technology is not only possible but essential. It is a testament to the power of SDN, open-source tools, and the resilience of network operators in the

face of evolving network demands to be explored, such as 3R regeneration and provisioning of more available paths in a network. This work set the begging of promising changes in the future optical networks.

6.2 Future Work

In summary, SDN brings a new level of control, flexibility, and efficiency to optical physical communication networks. It helps operators manage complex optical resources, adapt to changing traffic patterns, and optimize network performance, ultimately leading to better service quality and improved resource utilization.

The work presented only opened a small window into uniting the software with a network legacy instruments. The continuation would involve opening a network optical components such as a switch to be fully controlled by a software. Continuation of this work would involve learning and understanding of the software components and architecture of an optical switch. Since Tofino is a fully P4 enabled switch it already allows an operator to reconfigure it as desired, but going down in the depth of it and having access to all of the devices components provides more flexibility. The integration of a tunable transceiver with a programmable switch and the SDN control architecture have applications beyond wavelength allocation, and has further experiments in this field were shown the potential for other network programs.

6.3 Contributions and publications

The key publications from this work, as shown in each chapter as well, were:

The publications below during the time this masters work was done up to 2019.

- S. Ahearne, Y. Verbishchuk, C. Sreenan and F. Gunning, “Software Defined Control of Tunable Optical Transceivers Using NETCONF and YANG” 2018 European Conference on Networks and Communications (EuCNC), Ljubljana, Slovenia, 2018, pp. 1-86, doi: 10.1109/EuCNC.2018.8443188.[24]
- Y. Verbishchuk, C. Sreenan, and F. Gunning, “Configuration and Monitoring of the Optical Physical Layer Using Software-Defined Tools” in OSA Advanced Photonics Congress (AP) 2019 (IPR, Networks, NOMA, SPPCom, PVLED), OSA Technical Digest (Optica Publishing Group, 2019), paper NeT1D.2.[25]
- F.C. Garcia Gunning, A. Kaur, N.C. Estrada, J. Raulin, Y. Verbishchuk and C.J. Sreenan,“Enabling High Capacity Flexible Optical Networks” 2021 International

Conference on Optical Network Design and Modeling(ONDM) Gothenburg, Sweden, doi:10.23919/ONDM51796.2021.9492503,2021. [26]

Multiple discussions were held on open source platforms such as ONOS discussion boards at the time of this work being researched. An example of such discussion thread can be viewed at following source https://groups.google.com/a/onosproject.org/g/onos-discuss/c/k7Xncdeugr0/m/emznuSNSCAAJ?pli=1.

Furthermore, list of public conversations are below

- https://github.com/CESNET/netopeer2/issues/438
- https://github.com/sysrepo/sysrepo/issues/1315
- https://github.com/CESNET/Netopeer2/issues/276
- https://github.com/CESNET/Netopeer2/issues/340

and the continuation of this work was also published in the following papers:

- J. Raulin, G. Davey, Y. Verbishchuk, A. Jeffries, C. J. Sreenan and F. C. Garcia Gunning, "A programmable Ethernet transport Packetponder using common compact form factor pluggable tunable transceivers to support novel DWDM architectures", 2023 Optical Fiber Communications Conference and Exhibition (OFC), San Diego, CA, USA, 2023, pp. 1-3, doi: 10.1364/OFC.2023.Tu3D.6.[27]
- J. Raulin, G. Davey, Y. Verbishchuk, A. Jeffries, C.J. Sreenan, and F.C. Garcia Gunning, "Packetponder with open-source management system for future packet-optical networks", in Advanced Photonics Congress 2023, Technical Digest Series (Optica Publishing Group,2023), paper NeM3B.3.[28]

References

[1] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, 7th ed. Boston, MA: Pearson, 2020.

[2] C. Systems, "Osi model reference chart." [Online]. Available: https://learningnetwork.cisco.com/s/article/osi-model-reference-chart

[3] Cisco Systems, "Cisco annual internet report: Executive perspectives," https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html, 2023.

[4] Ericsson, "Ericsson mobility report november 2022," 2022.

[5] S. Popić, M. Vuleta, P. Cvjetkovic, and B. Todorovic, "Secure topology detection in software-defined networking with network configuration protocol and link layer discovery protocol," 11 2020.

[6] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426, Jan. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7426

[7] Open Networking Foundation, "Onos," 2024. [Online]. Available: https://opennetworking.org/onos/

[8] Netopeer Project, "Sysrepo repository documentation." [Online]. Available: https://netopeer.liberouter.org/doc/sysrepo/master/html/

[9] Lumentum Operations LLC, ROADM-20 Whitebox Product Guide, 2017. [Online]. Available: https://www.lumentum.com/en/products/dci-roadm-graybox

[10] Open Compute Project, "Bf6460x-t switch specifications v2," 2021. [Online]. Available: https://www.opencompute.org/documents/210216-bf6460x-t-switch-specifications-v2-pdf-1

[11] N. McKeown, "Nick mckeown's papers," 2023. [Online]. Available: http://yuba.stanford.edu/~nickm/papers.html

[12] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks."

[13] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 87–98, April 2014. [Online]. Available: https://doi.org/10.1145/2602204.2602219

[14] M. Casado, "Architectural support for security management in enterprise networks: A dissertation submitted to the department of computer science and the committee on graduate studies of stanford university in partial fulfillment of the requirements for the degree of doctor of philosophy," Ph.D. dissertation, Stanford University, Stanford, California, 2007.

[15] J. Schoenwaelder, "Network working group," 2003. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3535

[16] C. V. N. Index, "The zettabyte era—trends and analysis, 2015–2020: White paper," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-481360.html, July 2016.

[17] C. Annual and I. Report, "White paper cisco public," 2018. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[18] Cisco Systems, "2024 global networking trends report," Cisco Systems, Tech. Rep., 2024, accessed: June 16, 2024. [Online]. Available: https://www.cisco.com/c/dam/global/en˙uk/solutions/enterprise-networks/2024-global-networking-trends.pdf

[19] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally deployed software defined wan," in Proceedings of the ACM SIGCOMM Conference. Hong Kong, China: ACM, 2013, pp. 3–14. [Online]. Available: http://cseweb.ucsd.edu/~vahdat/papers/b4-sigcomm13.pdf

[20] V. Lopez, J. M. G. Josa, V. Uceda, F. Slyne, M. Ruffini, R. Vilalta, A. Mayoral, R. Muñoz, R. Casellas, and R. Martínez, "End-to-end service orchestration from access to backbone," Journal of Optical Communications and Networking, vol. 9, no. 6, pp. B137–B147, 6 2017.

[21] Google, "Inter-datacenter wan with centralized te using sdn and openflow background," 2012. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/02/cs-googlesdn.pdf

[22] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," Internet Engineering Task Force (IETF), Request for Comments RFC 6241, 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6241

[23] M. Björklund, "Yang - a data modeling language for the network configuration protocol (netconf)," RFC 6020, Oct 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc6020

[24] S. Ahearne, Y. Verbishchuk, C. Sreenan, and F. Gunning, "Software defined control of tunable optical transceivers using NETCONF and YANG," in 2018 European Conference on Networks and Communications (EuCNC). IEEE, June 2018, pp. 81–86.

[25] Y. Verbishchuk, C. Sreenan, and F. Gunning, "Configuration and monitoring of the optical physical layer using software-defined tools," in OSA Advanced Photonics Congress (AP) 2019 (IPR, Networks, NOMA, SPPCom, PVLED). Optical Society of America, 2019, p. NeT1D.2. [Online]. Available: http://opg.optica.org/abstract.cfm?URI=Networks-2019-NeT1D.2

[26] F. C. G. Gunning, A. Kaur, N. C. Estrada, J. Raulin, Y. Verbishchuk, and C. J. Sreenan, "Enabling high capacity flexible optical networks," in Proceedings of the 25th International Conference on Optical Network Design and Modelling (ONDM 2021). Institute of Electrical and Electronics Engineers Inc., 6 2021.

[27] J. Raulin, "A programmable ethernet transport packetponder using common compact form factor pluggable tunable transceivers to support novel dwdm architectures," in Optical Fiber Communications Conference and Exhibition (OFC) 2023. Optical Society of America (OSA), 2023. [Online]. Available: https://doi.org/10.1364/OFC.2023.Tu3D.6

[28] J. Raulin, G. Davey, Y. Verbishchuk, A. Jeffries, C. J. Sreenan, and F. C. G. Gunning, "Packetponder with open-source management system for future packet-optical networks," in Proceedings of the 2023 Networks Conference. Optica Publishing Group, 10 2023, p. NeM3B.3.

[29] H. Zimmermann, "Osi reference model-the iso model of architecture for open sys-

tems interconnection," IEEE Transactions on Communications, vol. 28, pp. 425–432, 1980.

[30] I. of Electrical, E. Engineers, I. C. S. L. S. Committee, and I. S. Board, IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture. Institute of Electrical and Electronics Engineers, 2002.

[31] D. J. D. Touch, "Updated Specification of the IPv4 ID Field," RFC 6864, Feb. 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc6864

[32] D. Liu, B. Barber, and L. DiGrande, "Chapter 5 - routing protocols: Rip, ripv2, igrp, eigrp, ospf," in Cisco CCNA/CCENT Exam 640-802, 640-822, 640-816 Preparation Kit, D. Liu, B. Barber, and L. DiGrande, Eds. Boston: Syngress, 2009, pp. 169–196. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781597493062000099

[33] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," RFC 7348, Aug. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7348

[34] Open Compute Project, "About the open compute project." [Online]. Available: https://www.opencompute.org/about

[35] Telecom Infra Project, "Telecom Infra Project Official Website." [Online]. Available: https://telecominfraproject.com/

[36] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple network management protocol (snmp)," RFC 1157, 1990. [Online]. Available: https://www.rfc-editor.org/info/rfc1157

[37] R. Enns, "NETCONF Configuration Protocol," Internet Engineering Task Force (IETF), Request for Comments RFC 4741, 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4741

[38] M. Björklund, "The YANG 1.1 Data Modeling Language," Internet Engineering Task Force (IETF), Request for Comments RFC 7950, 2016. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7950

[39] The Sysrepo Project, "Sysrepo," available at: https://github.com/sysrepo/sysrepo,.

[40] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and

S. Uhlig, "Software-defined networking: A comprehensive survey," Proceedings of the IEEE, June 2014. [Online]. Available: http://arxiv.org/abs/1406.0440

[41] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. Mckeown, and S. Shenker, "Nox: Towards an operating system for networks," NOX Project, 2008. [Online]. Available: http://www.noxrepo.org

[42] A. Albu-Salih, "Performance evaluation of ryu controller in software defined networks," Journal of Al-Qadisiyah for Computer Science and Mathematics, vol. 14, p. 1, 02 2022. [Online]. Available: https://ryu-sdn.org/

[43] ONOS Project, "Onos," 2024. [Online]. Available: https://wiki.onosproject.org/display/ONOS/ONOS

[44] OpenDaylight Project, "Opendaylight," 2024. [Online]. Available: https://www.opendaylight.org

[45] A. Drakos, T. Orphanoudakis, C. Politi, and A. Stavdas, "Statistical traffic multiplexing with service guarantees over optical core networks," in 2010 International Conference on Data Communication Networking (DCNET), 2010, pp. 1–5.

[46] M. Ruffini and D. C. Kilper, "From central office cloudification to optical network disaggregation," in IEEE Photonics Society Summer Topicals Meeting Series, SUM 2018. Institute of Electrical and Electronics Engineers Inc., September 2018, pp. 153–154.

[47] M. D. Leenheer, Y. Higuchi, and G. Parulkar, "An open controller for the disaggregated optical network," in Proceedings of the 22nd Conference on Optical Network Design and Modelling (ONDM 2018). Institute of Electrical and Electronics Engineers Inc., June 2018, pp. 230–233.

[48] J. Kundrat, J. Vojtech, P. Skoda, R. Vohnout, J. Radil, and O. Havlis, "Yang/netconf roadm: Evolving open dwdm toward sdn applications," Journal of Lightwave Technology, vol. 36, pp. 3105–3114, 8 2018.

[49] P. Castoldi, A. Giorgetti, A. Sgambelluri, G. Cecchetti, A. Ruscelli, N. Sambo, F. Paolucci, A. Morea, I. Cerutti, and F. Cugini, "Optical white box: Modeling and implementation," in Proceedings of the International Conference on Transparent Optical Networks (ICTON). IEEE, July 2018, pp. 1–4.

[50] E. Riccardi, P. Gunning, O. G. de Dios, M. Quagliotti, V. Lopez, and A. Lord, "An operator view on the introduction of white boxes into optical networks," Journal of Lightwave Technology, vol. 36, no. 15, pp. 3062–3072, August 2018.

[51] A. Lord, A. Soppera, and A. Jacquet, "The impact of capacity growth in national telecommunications networks," Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 374, no. 2062, p. 20140431, 2016. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2014.0431

[52] M. Ruffini and F. Slyne, "Moving the network to the cloud: The cloud central office revolution and its implications for the optical layer," Journal of Lightwave Technology, vol. 37, pp. 1706–1716, April 2019.

[53] I. Instrumentation, M. S. T. C. 8, A. N. S. Institute, I. of Electrical, E. Engineers, and I.-S. S. Board, IEEE standard for higher performance protocol for the standard digital interface for programmable instrumentation. Institute of Electrical and Electronics Engineers, 2003.

[54] IEEE Standards Association, "Ieee standard digital interface for programmable instrumentation (revision of ansi/ieee 488-1975, includes supplement ieee std 488a-1980)," ANSI/IEEE Std 488-1978, vol. 1, no. 84, pp. 1–84, 1978. [Online]. Available: https://doi.org/10.1109/IEEESTD.1978.7425098

[55] Linux GPIB Project, "Linux gpib - linux gpib driver and application for ieee-488 bus," 2017. [Online]. Available: https://linux-gpib.sourceforge.io/

[56] ISC Blog, "Ietf hackathon in berlin: Kea and yang/netconf." [Online]. Available: https://www.isc.org/blogs/ietf-hackathon-in-berlin-kea-and-yangnetconf/

[57] Sysrepo Project, "Sysrepo plugin development." [Online]. Available: https://www.sysrepo.org/plugins/documentation/

[58] CESNET, "Netopeer2 repository," 2024. [Online]. Available: https://github.com/CESNET/netopeer2

[59] L. Geng, C. A. Williams, S. B. Stok, and S. Shankar, "Revised datastore architecture for yang modules," https://tools.ietf.org/id/draft-ietf-netmod-revised-datastores-08, 2018.

[60] Netopeer Project, "Sysrepo - Connection and Session Management," 2020, available at: https://netopeer.liberouter.org/doc/sysrepo/master/html/conn˙sess.html.

[61] CESNET, "libnetconf." [Online]. Available: https://github.com/CESNET/libnetconf

[62] Open Network Install Environment, "ONIE: Open Network Install Environment." [Online]. Available: http://onie.org/

[63] Open ROADM Multi-Source Agreement (MSA), "Open ROADM: Enabling Interoperability in Optical Networks." [Online]. Available: http://openroadm.org/

[64] International Telecommunication Union, "Spectral Grids for WDM Applications: DWDM Frequency Grid," ITU-T, Recommendation G.694.1, 2020. [Online]. Available: https://www.itu.int/rec/T-REC-G.694.1/en

[65] International Telecommunication Union (ITU), "ITU-T Recommendation G.694.1: Spectral Grids for WDM Applications – DWDM Frequency Grid," 2003, available at: https://www.itu.int/rec/T-REC-G.694.1/,.

[66] O. N. Foundation, "Open disaggregated transport network (odtn)," 2020. [Online]. Available: https://opennetworking.org/odtn/

[67] I. Corporation, "Intel Tofino: Intelligent Fabric Processors," 2019, available at: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html,.

[68] P4 Language Consortium, "P4 Language Specification Version 1.0.0," available at: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html,.

[69] N. McKeown, D. E. Culler, R. Mahajan, S. Katti, D. A. Maltz, and D. A. Wetherall, "Rmt: A flexible and efficient transport architecture for multicasting," 2013.

[70] L. O. LLC, "Tunable sfp optical transceiver with limiting electrical interface," available at: https://www.lumentum.com/en/products/tunable-sfp-optical-transceiver-limiting-electrical-interface,.

[71] Optcore, "Ftlx8571d3bcl-fns - 10g sfp+ optical transceiver," available at: https://www.optcore.net/product/ftlx8571d3bcl-fns/,.

[72] M. Technologies, "Mellanox mam1q00a-qsa - sfp to qsfp28 adapter," https://shop.fiber24.net/FOSF-ME-MAM1Q00A-QSA/en.

[73] Effect Photonics, "A story of standards: From 400zr and open roadm to openzr+," https://effectphotonics.com/insights/a-story-of-standards-from-400zr-and-open-roadm-to-openzr/, 2023, accessed: 2025-11-12.

[74] Cisco Systems, Inc., "Growing the network with 400 gbps coherent pluggable optics," https://www.cisco.com/c/dam/en/us/solutions/collateral/service-

provider/routed-optical-networking/white-paper-sp-400g-coherent-optics.pdf, Cisco Systems, Tech. Rep., 2022, accessed: 2025-11-12.

[75] I. T. U. (ITU), "Itu-t recommendation g.8013: Ethernet transport network requirements," 2015, available at: https://www.itu.int/rec/T-REC-G.8013/en,.

.1 Appendix A: Code and files repository

The code files for this project is available at the following link:

- Public repository https://drive.google.com/drive/folders/1vvH3y8uQgd2LhUb3tj99GDc4FwI6LYSA?usp=sharing

This includes the thesis pdf document along with drivers code that was written during this thesis. Images used in the thesis as well as output publications are available for review at the repository.