Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Kommunikationsnetze

Virtualization and Software-Defined Control of Multilayer Flexible Optical Networks

Thomas Szyrkowiec, M.Sc.

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktoringenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr.-Ing. Norbert Hanik
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. Wolfgang Kellerer
	2.	Prof. Dr.-Ing. Thomas Bauschert

Die Dissertation wurde am 18.04.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 08.11.2018 angenommen.

Abstract

Today, optical networks are typically statically configured and provide customers with dedicated, fixed-bandwidth connections for long periods of times. Dynamic reconfigurability could benefit the infrastructure providers as well as their customers. Two promising concepts increasing the flexibility of computer networks are Software-Defined Networking (SDN) and Network Virtualization (NV). At its core, SDN describes the separation of the control and the data plane. It hands over the responsibility for forwarding decisions to a logically centralized controller. Even though optical networks follow a similar separation, they lack unified open interfaces, which are a corner stone of the SDN concept. NV follows a different approach that benefits from SDN without relying on it. NV enables multiple tenants to share a common infrastructure, thereby passing over partial control of the network to tenants. Common network abstractions for various technologies are one prerequisite. In the case of networks, abstraction removes details either for security reasons or to lighten the task of management by hiding complexity. Here, SDN interfaces are beneficial for controlling the hardware as well as virtual networks. Finally, flexible grids in Dense Wavelength Division Multiplexing (DWDM) transmission are an emerging technology improving the granularity in the assignment of spectral resources, with which the control and abstractions need to cope.
The definition of architectures and protocols for Software-Defined Optical Networks (SDONs) is currently an ongoing process and is far from being complete at the time being. This thesis investigates, through analytical methods and via experimental system implementation and evaluation, how optical networks with flexible DWDM grids can be transformed into SDONs in terms of dynamic lightpath provisioning and network virtualization. The analysis of existing models for the control of optical networks identified the need for a way of requesting virtual topologies from an infrastructure provider. A contribution of this thesis is the specification of a novel virtual topology intent interface. Later, an algorithm is described that covers all the intermediate steps from receiving the intent to exposing the resulting virtual network topology. One of the critical points in this transformation is the creation of virtual links. It maps physical link and spectrum resources to virtual links in the tenant's topology. In a two-step approach, a (shortest) path computation is required before spectrum is assigned. This thesis presents an extension of a shortest path algorithm based on precomputed information, which represents a trade-off between query time and memory space. The concepts are implemented and evaluated in a prototypical software called the Optical Virtualization Controller (OVC), which also provides a migration strategy for legacy networks. Finally, this thesis showcases the applicability of the developed solutions to use cases in the management and orchestration of multilayer networks.

Kurzfassung

Heutzutage werden Glasfasernetze überwiegend statisch betrieben. Neue Verbindungen werden selten hinzugefügt und sind anschließend über lange Zeiträume in Verwendung. Eine dynamische Administration würde sowohl dem Betreiber der Infrastruktur als auch dessen Kunden zugutekommen. Zwei vielversprechende Konzepte, die zu einer Flexibilisierung von Rechnernetzen führen können, sind Software-basierter Netzbetrieb (SDN) und Netzvirtualisierung (NV). Das Kernkonzept von SDN umfasst die Trennung der Steuerungsschicht von der Übertragungsschicht. Obwohl bei Glasfasernetzen diese Aufteilung inhärent ist, fehlen offene Schnittstellen, welche einen wichtigen Bestandteil von SDN darstellen. NV verfolgt einen anderen Ansatz, welcher zwar von SDN unabhängig ist, aber von dessen Konzepten profitiert. NV erlaubt es mehreren Kunden eine geteilte Netzinfrastruktur gemeinsam zu nutzen und gibt hierfür einen Teil der Steuerung an ebendiese weiter. Eine zentrale Voraussetzung sind gemeinsame Abstraktions-Modelle für die verfügbaren Technologie-Optionen. Diese Abstraktionen verstecken Details in Rechnernetzen entweder aus Sicherheitsgründen oder um die Konfigurationskomplexität zu reduzieren. Dafür sind offene SDN-Schnittstellen hilfreich, um eine einheitliche Darstellung und Steuerung der Hardware als auch virtueller Netze zu gewährleisten. Abschließend ist die Einführung von flexiblen Rastern in der Übertragung, die auf dichtem Wellenlängen-Multiplex (DWDM) beruht, zu erwähnen. Sie verbessern die Granularität für die Unterteilung des optischen Spektrums, womit allerdings auch die Steuerungskonzepte und Abstraktions-Modelle zurechtkommen müssen.
Die Definition einer Architektur für Software-basierte Glasfasernetze (SDONs) ist ein fortschreitender Prozess, der im Moment noch weit von einer Fertigstellung entfernt ist. Die vorliegende Arbeit untersucht durch analytische Methoden sowie experimentelle Implementierungen und deren Auswertung, wie Glasfasernetze mit flexiblen Rastern in SDONs überführt werden können, insbesondere wenn es um einen dynamischen Aufbau von Lichtpfaden und Netzvirtualisierung geht. Die durchgeführte Untersuchung der bestehenden Modelle für die Konfiguration von Glasfaser-Equipment identifiziert einen Missstand in Form eines fehlenden benutzerfreundlichen Konzepts zur Anforderung von virtuellen Topologien. Dieser Missstand wird in dieser Arbeit durch die Definition einer entsprechenden absichtsorientierten Schnittstelle behoben. Im weiteren Verlauf wird ein Algorithmus vorgestellt, der in der Lage ist, aus diesen Absichten eine virtuelle Topologie zu erzeugen und dem Kunden anzubieten. Ein kritischer Punkt dieses Prozesses ist die Erzeugung von virtuellen Kanten, welcher die Abbildung von physikalisch verfügbaren auf virtuelle Ressourcen darstellt. Bei einem zweistufigen Verfahren wird dazu zuerst ein (kürzester) Pfad für die anschließende Zuweisung des Spektrums berechnet. Ein wichtiger Beitrag dieser Arbeit in diesem Zusammenhang ist die Erweiterung eines Algorithmus zur Berechnung von kürzesten Pfaden, der auf vorberechneten Informatio-

nen basiert und damit einen Kompromiss zwischen Laufzeit und Speichermenge darstellt. Die Konzepte wurden in Form des Optical Virtualization Controllers (OVC), der zeitgleich eine Übergangslösung für Bestandsgeräte darstellt, implementiert und evaluiert. Abschließend zeigt die Ausarbeitung die Anwendbarkeit der Lösung auf das Management und die Orchestrierung von Netzen mit mehreren Schichten auf.

Acknowledgment

Since I cannot imagine any way of coming up with a complete list of people that need to be acknowledged and an incomplete list would offend the people I have forgotten, I acknowledge everyone who is reading this because you probably should be on that list.

Thanks!
Danke!
Dziękuje!

Projects

- The research has been partly funded by the German ministry for education and research (BMBF) under Grant 16BP12400 as part of the SASER ADVAntage-NET project.
- The research has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n.608528 in the project STRAUSS.
- The research has received funding under the European Union's Horizon 2020 research and innovation programme under grant agreement No 645127 (ACINO).
- Parts of this work have been performed in the framework of the CELTIC EUREKA project SENDATE-Secure-DCI (Project ID C2015/3-4), and it is partly funded by the German BMBF (Project ID 16KIS0477K).

Other

Icons partially courtesy of ADVA Optical Networking.

Contents

1. Introduction

Optical networks have been considered a conglomeration of large static pipes for a long time. New technologies like Software-Defined Networking (SDN), Network Virtualization (NV) and flexible Dense Wavelength Division Multiplexing (DWDM) grids have started changing that picture. SDN has triggered a paradigm shift in the overall network control. Its basic idea is to separate the control plane from the data plane and to hand over the forwarding decisions to a logically centralized controller. One goal is to give network operators a standardized interface for implementing forwarding behavior in Network Elements (NEs) and thereby providing opportunities for automation and efficiency improvement. Optical networks, which are responsible for high data rate transmission over long distances, are following a similar control separation by default. In optical networks, SDN is a driver for the definition and implementation of open interfaces for the control of optical network equipment. There, the goal is to develop unified interfaces and to increase the dynamicity of the network. NV is another promising area in networking. NV enables the sharing of a physical network between clients by creating virtual networks, which are then exposed to client controllers. This feature is not dependent on SDN, but NV benefits from open control interfaces. When defining virtual networks in the case of optical networks, the analog nature of optical networks needs to be taken into account. Finally, a new standard emerged introducing flexible DWDM grids. They allow for a more efficient use of spectral bandwidth and support higher data rates per channel. These grids need to be handled by (optical) network controllers.
All these recent developments point to a future with a more dynamic and cost-efficient operation of optical networks. Before this can become reality, preliminary work and adaptations need to be carried out. A fluid transition toward the new networking approach is the most likely path of action. In this chapter, the motivation for the thesis is established. The limitations of current optical networks are pointed out, the three mentioned technologies — SDN, NV and flexible DWDM grids — are explained in more detail and their applicability to the existing challenges is introduced. Finally, the contribution of this thesis is summarized by giving an overview of the following chapters and putting them into context.

1.1. Motivation

Optical networks are commonly deployed by incumbent operators, network service providers, Internet Service Providers (ISPs) and data center operators as well as businesses, banks or governmental institutions with their own infrastructure. They use lasers to transmit data through a fiber by modulating the emitted wavelength. One of the major advantages of optical networks is that they transfer huge amounts of data over a long

distance. Typically, current systems support 100 Gbit/s per wavelength. There are already modules available that achieve 200 Gbit/s per channel and the next generation will tackle 600 Gbit/s to 1 Tbit/s. Multiple wavelengths can be multiplexed on a single fiber using Wavelength Division Multiplexing (WDM), e.g., 80 or 96 channels with 50 GHz spacing in the C-band. Assuming an appropriate modulation format and a continuous amplification, the signal can be transferred up to a few thousand kilometers without leaving the optical domain. Staying in the optical domain eliminates the cost and delay induced by the conversion and processing on the electrical level. On the downside, the transmission is analog in nature and physical effects need to be taken into account, which adds complexity to the control and management of the network.

1.1.1. Limitations of Optical Networks

Optical networks are a rather small part of the whole network ecosystem. Through their capability to transport large amounts of data — in the order of multiple Tbit/s per fiber — they are a prominent part of networks with high capacity demands. Due to their special application area and inherent properties resulting from their analog nature, they suffer from a number of shortcomings.
One of the major limitations of today's optical transport networks is their rigid nature. Optical networks are often static with established connections being deployed for months or even years [Sim14]. One reason for the static nature is the complexity involved in provisioning. Often manual steps are needed for setting up or tearing down an optical connection. Even with the support of a Network Management System (NMS), which is not used in many cases, and remote access, the setup is still a lengthy process. Additionally, the lightpath needs to be configured hop-by-hop and the signal needs to be equalized to avoid interference with other neighboring signals. With the current mode of operation, it is not possible to request new connections from a provider within hours or minutes. This inflexibility prevents the optical network from reacting to fluctuations in demand, which are typical in today's traffic, e.g., due to time zones. Also, short-term requests for connectivity between endpoints cannot be satisfied. The low dynamicity is also caused by the fact that optical networks are planned for long time periods, i.e., few years [Muk06]. They do not only have to accommodate current demands but also forecast future requirements. The slow setup times and the long-term planning lead to overprovisioning and a low utilization of the available bandwidth. This means that transponders are operated at a low load to be able to mitigate high demands. Additionally, the interoperability of the NEs of different vendors is quite limited. On the data layer, the hardware implementations differ so that an interworking on that level is complicated. On the control layer, vendors use proprietary information models for standardized protocols and offer vendor specific NMSs for their devices. Therefore, similar network elements are grouped together and form vendor islands, which only interact at the edges. So far, optical networks relied on a rigid DWDM grid. It specifies a fixed amount of spectral bandwidth that can be used by a signal, e.g., 50 GHz. If a channel occupies less than the defined range, the remaining bandwidth is wasted. If a channel requires more spectrum, then multiple independent wavelengths, of which each includes

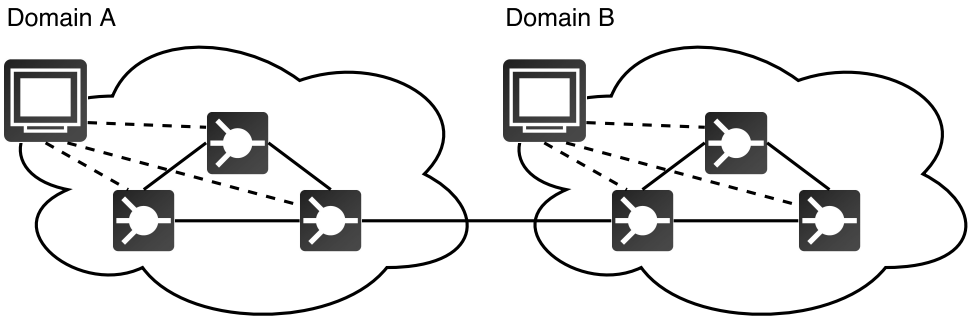


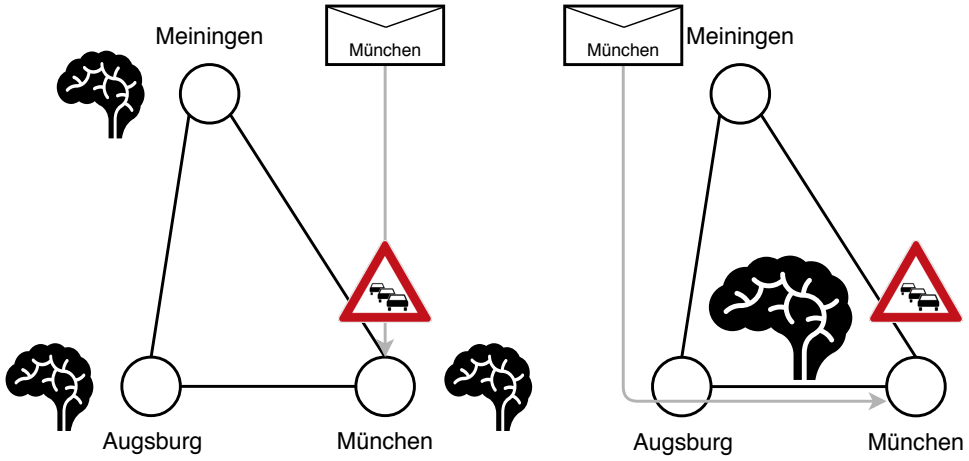
Figure 1.1.: Two domains managed by individual NMSs.

guard bands, have to be chosen and efficiency benefits from having a continuous part of the spectrum cannot be gained.

1.1.2. Problem Statement

In general, the control of optical equipment is a complex task and optical networks are rather static [Alv+17]. The presented limitations lead to the following challenges that need to be overcome: (i) Complex and vendor-specific control of optical equipment, (ii) no dynamic control of assigned network resources by the client, (iii) Open Systems Interconnection (OSI) layers are operated independently, (iv) inefficient use of spectrum due to fixed DWDM grids.
(i) The complexity of control results from heterogeneous and vendor-specific commercial systems and the analog nature of optical transmission. First of all, the equipment is modular, which leads to node constraints based on the hardware configuration. One example are fixed filters, which assign a fixed wavelength to a particular transponder. In addition, components from different vendors offer unique implementations and features in order to differentiate from each other. On a physical network level, the analog nature leads to network constraints, e.g., optical signal-to-noise ratio, crosstalk, and dispersion affect the maximum signal reach. They have to be taken into account before setting up services through the network. As a result, every vendor has his own closed NMS accessing the devices through proprietary interfaces, which leads to the aforementioned vendor islands. Typically, optical networks are divided into domains [GE13]. Domains group network elements by administrative unit, common characteristics, like technology and vendor, or control plane instance. These groups are then managed by an individual NMS for each domain (see Fig. 1.1). Even though the NMS is a centralized entity, it lacks standardized and open interfaces so far. Furthermore, unlike IP-networks, the optical domain most commonly deploys circuit switching. Existing techniques used in the OSI layer 2 & 3 are frame & packet oriented and not easily transferable to optical circuit switched networks, which represent a layer 0/1 technology. New protocol definitions and configuration models that go along with them need to deal with the inherent complexity. The required steps will cover an abstraction of device models and a unification of control interfaces for optical equipment.

(ii) A customer (of any kind) requesting connectivity through an optical network has only limited control over the provided resources and the connectivity. Usually, it is a lengthy process that is triggered by an inquiry for a connection between selected endpoints. Often, the network operator processes the request manually before it is installed. The installation involves a certain amount of manual configuration as well, depending on the deployed software and the available level of custom automation. At the end of this process, the customer receives a connection between the specified endpoints, which remains active for an extended time period. Today's requirements for connectivity services expect shorter delays between request and instantiation than days or weeks. For example in the field of broadcasting, connections between the central studio and event locations need to be set up on demand for the time of the event. After the event, the connection is no longer needed and can be torn down. Another example are data centers that want to react quickly to traffic peaks by activating additional bandwidth. The time required for setting up connections as well as their life span need to be reduced in order to allow a dynamic and efficient utilization of the network. One approach for this problem is to give the customer a restricted view of the network that he requested including the ability to control it. The customer can then react to changes on his own, without having to go through the network operator. Ideally this is done via open interfaces. Of course, safety precautions have to be in place to avoid interruptions of other traffic or to prevent the use of resources other than the assigned ones. To achieve this flexibility, mechanisms that enable the creation of network partitions and hand over the control to the customer need to be developed.
(iii) In transport networks, the packet layer (2/3) and the optical layer (0/1) are operated individually, sometimes even by different business units. These layers are split into separate domains and only the transition at the edge is considered in operations. This means aggregated packet streams enter the optical network at an entry point and leave it at an exit point. These packets use existing optical tunnels without dynamically adjusting connections in the underlying layer. One reason is that an optical connection is thought of as being a static pipe without a unified way of changing connectivity on demand. Therefore, routing decisions are made on the higher layers assuming the entry and exit points of the optical layer to be given. This approach does not allow for efficient management of the overall network resources. Without considering both domains in conjunction, the decision is based on partial information and does not take into account a joint configuration. For more resource-efficient decisions, common abstractions and a joint control need to be introduced.
(iv) The commonly deployed fixed DWDM grid limits the options for efficient use of the spectrum. Only for a small range of bit rates, fixed grids represent a good solution. For most bit rates, the grid is either wasting spectral bandwidth or it would be more efficient to get a larger piece of the spectrum instead of multiple independent wavelengths with individual guard bands. A recently standardized approach introduces the notion of flexible DWDM grids, which partition the spectrum into discrete units, i.e., slots. The size of a single slot is smaller than the channel spacing in a fixed grid and an arbitrary number of continuous slots can be chosen for a data channel. This leads to variable portions of the spectrum that can be selected, but at the same time the flexibility introduces



(a) Distributed control.

(b) Centralized control.

Figure 1.2.: Difference between a distributed and a centralized control using a mail delivery analogy. Road system: data plane, brains: control plane.

new issues like fragmentation, i.e., parts of the spectrum that effectively cannot be used. This evolution of optical networks needs to be considered by the upcoming generation of controllers and models.

1.2. State of the Art

The main drivers of change in this field that can help to overcome the presented shortcomings are SDN — Software-Defined Optical Networks (SDONs) in particular —, NV and flexible DWDM grids. SDN is a networking paradigm proposing the separation of control and data plane with a logically centralized controller including open interfaces. In SDONs, the concepts of SDN are extended toward optical networks. NV defines the creation of virtual topologies based on an underlying physical topology for exposing them to a client. This approach is facilitated by SDN but does not strictly depend on it. Finally, flexible DWDM grids introduce the flexibility to optimize the utilization of the optical spectrum.

1.2.1. Software-Defined (Optical) Networking

SDN is a networking approach that separates the control plane from the data plane. It introduces a (logically) centralized controller for the configuration of the forwarding devices. The approach will be explained using an analogy related to mail delivery. The difference between a centralized and a distributed control is explained using Fig. 1.2. The links represent a road system between the cities, which is mapped to the data plane, while the brains represent the control plane, which makes the forwarding decisions. We assume that we want to send a letter from Meiningen to München. In a delivery system with distributed control (Fig. 1.2a), the mail would be forwarded toward the destination

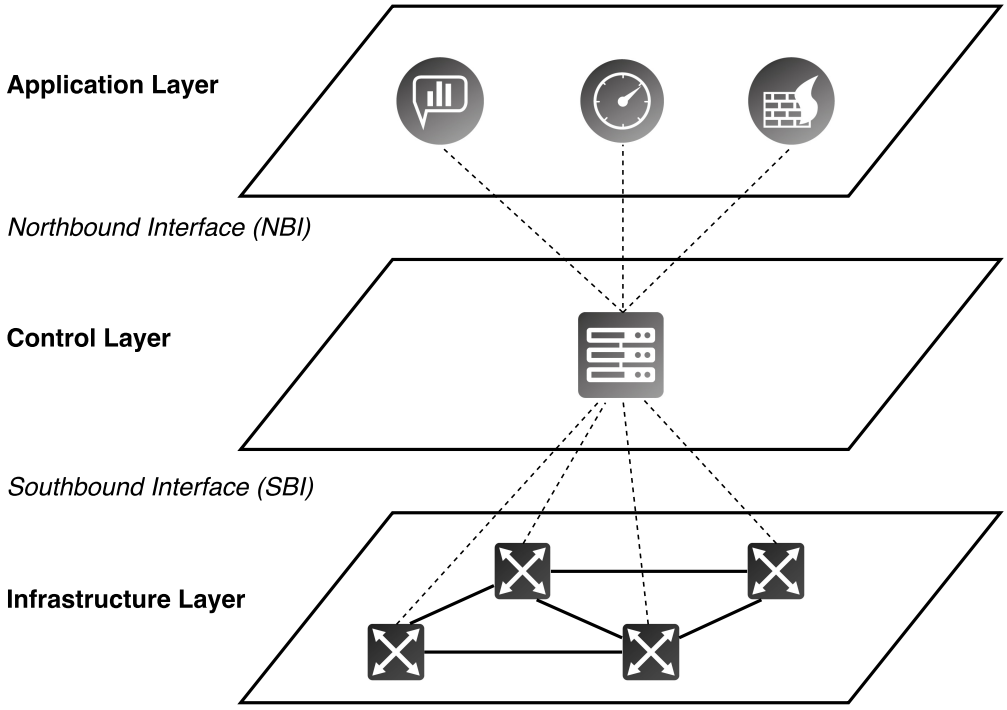


Figure 1.3.: General SDN concept (layers and interfaces).

based on local information (local brain), e.g., using the next hop along the shortest path toward the destination. Only limited information about the system as a whole is included in the decision, e.g., present traffic or free capacities along other paths might not be accessible. With a centralized location for the control with globally available knowledge about the system (central brain in Fig. 1.2b), traffic jams or construction sites on the shortest delivery path are taken into consideration and a better solution is found. This means that using the additional hop through Augsburg can lead to a better overall result according to an objective, like minimizing the delivery time.

A more formal definition is shown in Fig. 1.3 [Ope18c]. The infrastructure layer (or data plane) at the bottom comprises NEs that are forwarding traffic. In the middle sits the control layer that includes the logically centralized controller. In this context, it means that the controller acts as a centralized entity even though it might be distributed, e.g., across multiple servers and locations, due to scalability and performance reasons. The term Southbound Interface (SBI) is used for the interface between the controller and the infrastructure. The uppermost layer is the application layer. It includes applications that rely on information that are offered by the controller, e.g., firewall, metering or analytics. The interface between the controller and the applications is called Northbound Interface (NBI).

One of the initial drivers of SDN is the OpenFlow (OF) protocol [McK+08]. OF is a protocol for configuring forwarding devices that maintain a flow table. Early applications of OF focused on switches and routers. A flow table entry contains header fields,

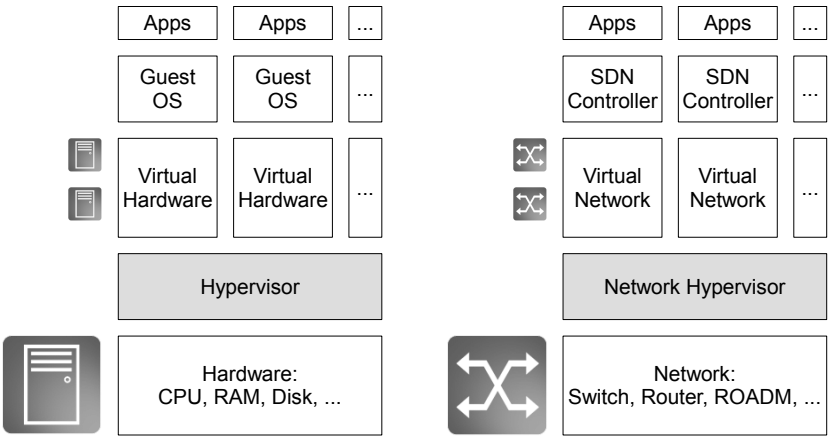


Figure 1.4.: Hypervisor in server and network virtualization.

actions and statistics. The header fields are used to match packets and assign them to flows. The actions of the matching rule are then applied to the packets of the respective flow. They range from selecting an output port to header manipulations. Additionally, statistics about packets are collected on a per rule basis as well as in an aggregated form. For packet-switched networks, OF is the emerging standard and multiple versions are maintained in parallel by the Open Networking Foundation (ONF) in their specification library [Ope18b].

No standardized Application Programming Interface (API) description is available for the NBI, which presents an access point for applications. Even though all controllers provide their own implementation of this API, most of them rely on Representational State Transfer (REST) protocols.

Transport SDN (T-SDN) is the extension of SDN to transport networks. A transport network comprises multiple layers, domains and vendors [Alv+17]. The available layers may span from Layer 0 (optical) to Layer 3 (packets) and the network is split into several domains — not only based on the layer but also based on technology and vendors. One part of transport networks are optical networks. Applying SDN to them leads to SDONs. Since optical networks naturally follow the main idea of a separated and centralized control, the SDN concept is easily applicable. Still, fundamental requirements for SDONs are common abstractions and interfaces. Apart from NMSs, specialized software tools such as scripts, need to be maintained by providers in order to automate the control of these networks. Today, diverse technologies and interfaces increase the complexity of this task. Therefore, the objective is to create common abstractions and interfaces similar to what OF offers for higher layer networks. Control protocols and models for optical networks are discussed in more detail in Chapter 2.

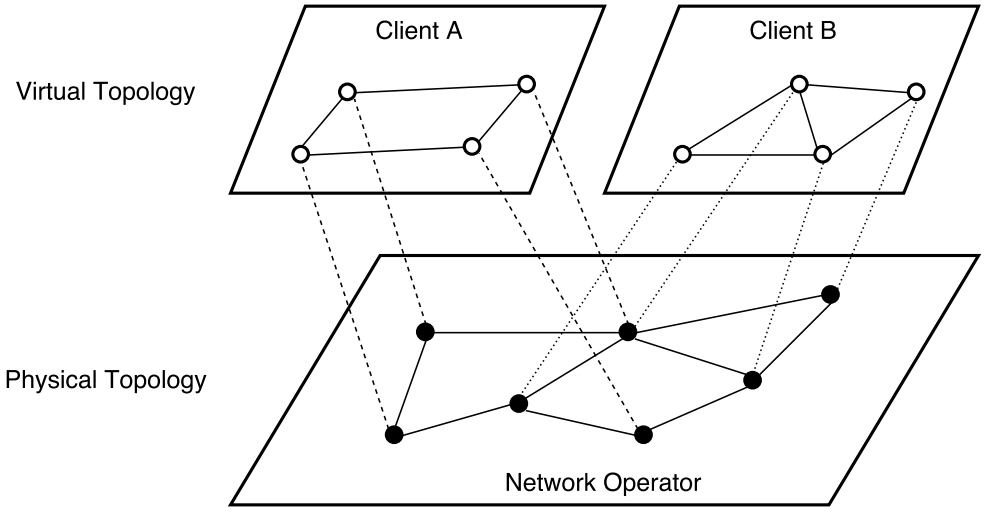


Figure 1.5.: Example for NV with two clients.

1.2.2. Network Virtualization

Generally speaking, *hypervisor* is not a new term and is well-known in the field of server virtualization (Fig. 1.4 left-hand side) where it is also called virtual machine monitor [Gol74]. This field abstracts hardware into virtual resources and exposes them in the form of virtual servers or machines. These virtual machines are then exposed to a guest Operating System (OS) that operates on top of those virtual resources. In general, the OS should not be able to detect any difference based on how it interacts with the hardware. The hypervisor is in some sense a mediation layer between the hardware resources and the virtual resources. It is also responsible for isolating individual virtual machines from each other. A similar concept can be applied to networks, where a network hypervisor abstracts physical network resources into virtual ones (Fig. 1.4 right-hand side). This concept is called NV and the virtualized network comprises switches, routers, Reconfigurable Optical Add-Drop Multiplexers (ROADMs), etc. These devices provide network resources of different kinds, e.g., packet processing capabilities, ports or wavelengths. The resources are shared among the clients that are using the network. The clients receive a virtual network, which is assigned a set of these resources. Like in a virtual server machine, the client is allowed to configure and use his virtual network according to his needs. The network hypervisor ensures client isolation and prevents any undesired interaction between them. Due to the standardized interfaces, SDN controllers represent a good option for the control of virtual networks. Similar to an OS, controllers should be able to operate a virtual network in the same way as any other network based on physical hardware.

Previously, NV was done by means of Virtual Local Area Networks (VLANs), Virtual Private Networks (VPNs), active and programmable networks and overlay networks [CB09; CB10; JP13]. They enable the creation of virtual partitions but they involve a proprietary configuration process for the hardware, e.g., handling of VLAN tags, or

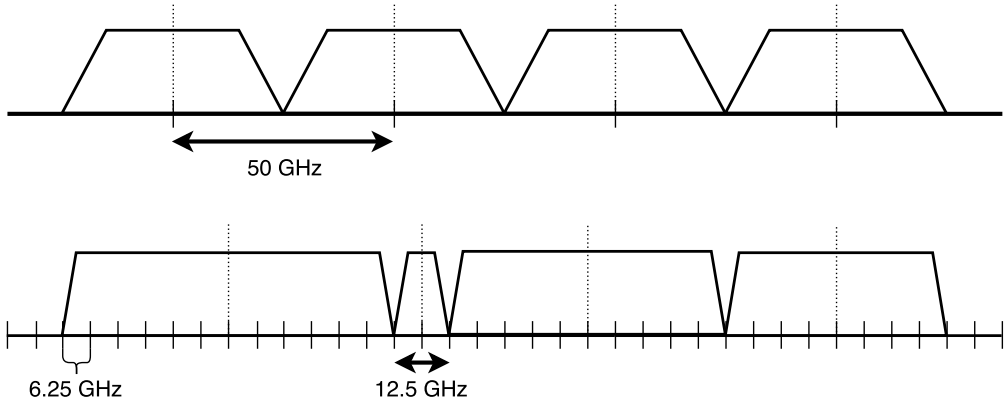


Figure 1.6.: Fixed grid with 50 GHz channel spacing (top), and flexible grid with 6.25 GHz central fequency spacing and 12.5 GHz slot width (bottom).

additional software, e.g., VPN server and clients. There are multiple reasons for deploying NV [Fis+13; JP13]. The most important one is sharing resources between clients. Shared infrastructure and resources require a strict separation that can be easier achieved with virtual networks. It also involves less overhead to assign resources to a client or to remove them. Additionally, the management of virtual devices is simplified through unified interfaces implemented in software. An example for a virtualized network is given in Fig. 1.5. It shows a network operator offering parts of his network to two clients. One possible mapping of the virtual topology to the physical one is indicated by the dashed (client A) and dotted lines (client B). It can be seen that both clients may share the same network elements and links. Therefore, the hypervisor needs to ensure that isolation is guaranteed.
In general, a user is allowed to request an arbitrary (virtual) network layout, which needs to be mapped to the available physical resources of the underlying network. The process is called Virtual Network Embedding (VNE) [Fis+13]. It comprises two tasks: the mapping of virtual links and the mapping of virtual nodes. The virtual links are not necessarily realized by a single physical link but can also be mapped to a path — a sequence of links — through the network. The virtual nodes are typically assigned to physical nodes, which can host multiple virtual counterparts. Both tasks can be either handled separately or jointly, which influences the result and the computational complexity. In optical networks, the link mapping needs to consider wavelength continuity constraints as well as physical layer impairments. Also, it is likely that the endpoints are predetermined and do not need to be mapped because of the client's hardware location.

1.2.3. Flexible DWDM Grids

Many optical networks are based on DWDM, which is a form of WDM. They typically use a fixed grid — defined by the International Telecommunication Union (ITU) in recommendation G.694.1 [ITU12] — with a common nominal central frequency spacing of 50 GHz or 100 GHz, which is anchored to 193.1 THz. Signals can only be transmitted

on one of these central frequencies in that grid, e.g, for 50 GHz the grid is defined by $193.1 + n * 0.05$ in THz where $n \in \mathbb{Z}$. An example is shown in the top part of Fig. 1.6. Due to this definition, the hardware is commonly fixed to one spacing and does not support signals spanning multiple central frequencies in this grid.
Assuming that there will not be a large deployment of new fibers in the next decade, the focus is on optimizing the currently available infrastructure [Zha+13d]. Spectral efficiency is one way of quantifying the progress, although the total bit rate and the reach need to be taken into account too. It is defined as bit/s/Hz, translating to the bit rate that can be transmitted per frequency unit. For efficient transmission of 400 Gbit/s and beyond, a fixed grid is no longer the best option [Ger+12; WLV13]. This means that by using a variable portion of the optical spectrum the spectral efficiency can be improved, even though it is also possible to transmit such bit rates using inverse multiplexing with a fixed grid. The ITU Telecommunication Standardization Sector (ITU-T) published the definition of the flexible DWDM grid in 2012 [ITU12]. It adds the concept of slots which have a nominal central frequency of $193.1 + n * 0.00625$ in THz and a slot width of $12.5 * m$ in GHz where $n \in \mathbb{Z}, m \in \mathbb{N}^*$. Any combination of n and m is allowed as long as slots do not overlap. An example is given in the bottom part of Fig. 1.6.
The benefit of a flexible grid is not only the finer granularity but also the ability to concatenate adjacent slots to form larger spectrum slices [WLV13]. The goals are to allow mixed bit rate signals and different modulation formats, while at the same time optimizing the spectral efficiency. On the one hand, a reduction of the required spectrum can be achieved for low bit rates. On the other hand, high-bit-rate signals, i.e., 400 Gbit/s and beyond, can utilize the spectrum in a more efficient way by being assigned a larger number of slots. Additional benefits can be gained from the use of higher order modulation formats like Quadrature Phase-Shift Keying (QPSK) or Quadrature Amplitude Modulation (QAM), e.g., 8-QAM or 16-QAM. The definition of flexible grids is backward compatible by choosing appropriate values for n and m. Even though the full definition allows for more flexibility, not all possible combinations have to be supported by the hardware, e.g., a reduction of slot widths and positions to even multiples is perfectly fine [ITU12]. One drawback of flexible grids is the fragmentation of the optical spectrum. Similar to hard disks, fragmentation describes unused fragments of spectrum which might be hard to use because of their size or continuity constraints for multi-hop connections [Ger+12; WLV13]. They occur when channels are torn down or added to the network and gaps are created between existing channels. There are two ways of dealing with this issue: either by applying defragmentation or by developing new Routing and Spectrum Assignments (RSAs) schemes. The first one can be achieved by hitless spectrum reallocation by tuning the central frequency without affecting the ongoing transmission. The latter one is derived from the Routing and Wavelength Assignment (RWA) for fixed grid networks. RSA is the process of finding a path through the network and assigning a suitable part of the spectrum to the channel. It is easier to implement than defragmentation but it leads to increased computational complexity.
A network deploying flexible transponders and grids is also called Elastic Optical Network (EON) [Ger+12; LV16]. The elasticity refers to the flexible spectrum assignment and the variable bit rate [Ger+12]. An elastic optical network offers multiple ways to

satisfy a demand. The appropriate modulation format as well as the Forward Error Correction (FEC) overhead is chosen according to the distance and the link characteristics. The application of superchannels, which bundle multiple carriers into a single channel, increases the flexibility even further [Zha+13d]. Bandwidth Variable Transponders (BVTs) are one important technology to take full advantage of the flexible DWDM grids [WLV13]. They are able to vary their bit rate by changing the modulation format or the baud rate [Sam+15]. By tuning these parameters for a given bit rate and reach, an operator is able to optimize the spectral efficiency. Sliceable BVTs (S-BVTs) are an extension of BVTs. They are able to support a number of virtual transponders that generate independent optical flows toward different destinations [Sam+15]. Technologies that are needed to enable EONs are Wavelength Selective Switches (WSSs) with a finer granularity, BVTs or even S-BVTs. The flexible control of these new technologies is an open issue [WLV13]. Many of the aforementioned technological advantages are not achievable without SDN [Zha+13d]. For example, a dynamic change of a bit rate or modulation format is impractical with current control mechanisms.

1.2.4. Goals

Applying the concepts of SDONs, NV and flexible DWDM grids will change the landscape of optical networks. Foremost, the operation of optical networks is about to become more dynamic and flexible and it will bring network operators one step closer to an automated control and management of the whole network [Zha+13d].
SDN introduces a centralized control with open interfaces and promises network programmability for controllers and applications alike. Centralized control is already best practice in optical networks and does not require fundamental changes in the current approach. The standardization of interfaces is the next step toward the deployment of SDN controllers in the optical domain. The development of SBIs is progressing quickly and potential candidates for the long run are already available. As soon as an agreement is reached, vendors are likely to implement the agreed on interface in their devices and thereby enable the integration into the SDN environment. Network operators need a well-defined NBI to be able to automate tasks related to network control. So far, every available controller is offering its own NBI. Compared to the SBI, not much work is carried out on its definition and unification. In the end, it is important to unify both interfaces to avoid a multiplicity of definitions which would lead to similar challenges currently faced. Common abstractions for optical equipment are another important puzzle piece to enable interaction with SDN controllers. It is crucial for controllers to apply the same (or similar) model for the managed devices. Without common abstractions, it will be difficult to exchange the right information through the offered interfaces. There are already extensive models available for the management of optical equipment, e.g., ITU-T G.874.1 [ITU16], but they need to be abstracted to fit into the SDN view. Unified abstractions and interfaces lead to improved interoperability and control of heterogeneous equipment. Generic models identify a minimal set of functionality that needs to be supported by the devices and can be adequately used by the controllers too. There is still room left for extensions of the generic models to include additional features on both

ends. With such abstractions and interfaces in place, a joint control and management of multilayer networks is close to being reality. It gives the network operator the capability of further optimizing the network based on all available information.
The sharing of a common infrastructure through NV benefits the client as well as the network operator. It provides fully-controllable virtual topologies to the client that are created by the network operator. The client is able to control his portion of the network according to his requirements and can react to changes without explicitly involving the operator. Together with SDN controllers and open interfaces synergies are achieved. The client is able to use any compliant controller with the previously described benefits, in order to control his (virtual) network. Handing over the rights to the client is a major contribution to the dynamic control of optical networks. At the same time, the operator is able to optimize his operations by reducing the overhead for provisioning new connections. The operator's work is reduced to assigning and verifying the creation of virtual networks. With open interfaces in place, this task can be automated, giving the client full control over changes to his virtual network.
Both, SDN and NV, lead to a reduction of overprovisioning. Due to the improved dynamicity, the need for keeping active connections underutilized or on standby is reduced. Additional capacities can be activated or added as a timely reaction to changes in demand or for short term use.
On the one hand, flexible DWDM grids enable improvements of the spectral efficiency for traffic going through the optical network. The occupied bandwidth of the spectrum is optimized for the requested bit rate and reach while maintaining the transmission quality. On the other hand, they increase the complexity of the control task by introducing new parameters like number of slots, modulation format, baud rate, etc. The additional flexibility needs to be handled by the controller, which requires new algorithms that are able to include these additional parameters, e.g., in RSA. Also, the interface definitions and abstraction models have to be capable of representing this technology appropriately.
In summary, these are the goals that will be addressed by the presented paradigms, including their mapping to the problem statements listed in Sec. 1.1.2:

- dynamic and flexible (software-defined) control of optical networks {(i), (ii)}
- unification of southbound and northbound APIs {(i)}
- control and interoperability of equipment based on a generic model {(i)}
- automatic creation of virtual networks based on intents {(ii)}
- control and management of multilayer networks {(iii)}
- support for flexible DWDM grids {(iv)}
- reduction of overprovisioning {(i), (ii), (iii), (iv)}

1.3. Key Contributions

This thesis investigates the applicability of SDN and NV to optical networks. It involves the analysis of existing models and the identification of gaps in their definition, the creation of virtual networks, starting from intents, and the integration into a multilayer environment. The results are implemented in a new software platform and evaluated using commercial hardware. These are the main outcomes of this thesis:

- **Analysis of existing models for control and virtualization of optical networks:** A comprehensive analysis of the most prominent available models for optical networks is provided. This work investigates the capabilities of the models, offers a classification for them and identifies existing gaps in the current definitions. Their applicability to certain areas of control and virtualization of SDONs is shown.
- **Intent-based NBI for requesting virtual topologies:** An easy-to-use client interface for defining virtual topologies is identified as a gap in current models. An interface model description for closing that gap is proposed. It defines an interface that provides a way of requesting virtual topologies based on intents. Intents describe what the client wants without bothering him with technological details.
- **Shortest path algorithm extension:** Shortest path algorithms play a prominent role in optical networks for calculating routes through the network as part of RSA. An algorithm that partially precomputes the distances is extended to be able to react to changes in the network, like added and removed edges. Precomputation presents a trade-off between computation time and memory requirements.
- **Procedure for automatic creation of virtual networks:** Requests submitted through the proposed intent interface need to be processed in order to receive a virtual network. The presented procedure describes the intermediate steps needed between the intent submission and the usage of the virtual topology. It involves multiple tasks: processing the intent, assigning resources utilizing the shortest path algorithm and exposing the created topology.
- **Architecture for control and virtualization of optical networks:** A software architecture is introduced to enable the application of SDN and NV to currently deployed optical networks. It can be integrated into the existing environment and helps migrate the equipment that does not support the newly developed or experimental interfaces yet. It is implemented as part of this work.
- **Experimental evaluation of the presented approaches:** The developed model/interface, the shortest path algorithm and the architecture are evaluated based on a software implementation. Commercial hardware is used whenever possible to obtain a realistic evaluation that takes properties of the optical equipment into consideration. The results from the proof of concept implementation confirm the viability of the overall approach.

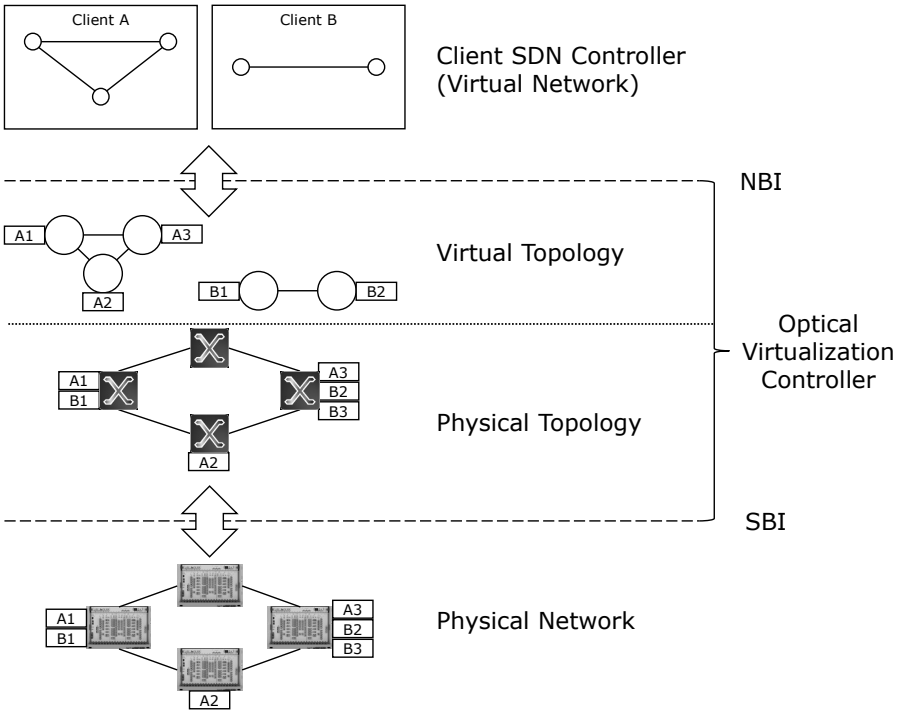


Figure 1.7.: General concept for applying SDN and NV to optical networks.

- **Verification of applicability to multilayer scenarios and orchestration:** Selected examples, in which optical networks are integrated into a multilayer orchestration context, are presented. Typical work flows like data center automation as well as forward-looking secure service deployment are explained. These proofs of concept verify that it is possible to combine the optical and the packet layer into a joint control with these new tools.

1.4. Conceptual Approach

The general concept used in this thesis for including optical networks in an SDN environment and exposing NV capabilities is presented in Fig. 1.7. A solution that can support the transition from proprietary control protocols for network equipment to open standardized interface is the use of a mediation layer. For current deployments a mediation layer translates existing protocols into SDN ones. In the future, the translation is no longer needed and the focus is only on the control, since the SBI interfaces are unified. It is represented by the software component Optical Virtualization Controller (OVC). The OVC combines two functionalities: it is the (SDN) domain controller for the underlying optical network and it provides the virtualization capabilities of a hypervisor. In its function as an SDN controller it is capable of communicating with the optical network equipment to execute control tasks. Examples are topology discovery, service instantiation and tear down of lightpaths. It is able to communicate directly with the devices,

to interact with a control plane or to go through an NMS. The right choice depends on the availability of the control functionality and the exact existing deployment. Based on the southbound communication, it is able to create a physical topology to capture all available information about the underlying network. The hypervisor capabilities establish a way of creating virtual topologies based on the physical representation. These can be used to expose virtual networks to clients or an orchestrator through an (SDN) NBI. Then the client is able to operate his virtual network with an SDN controller (mostly) without noticing a difference to a physical one. Ignoring the virtualization functionality, the OVC can be seen as a translation layer between closed interfaces and the SDN world. An illustrative example in Fig. 1.7 shows a simple scenario with a single physical network with four nodes. Three ports are assigned to each client as indicated by the prefixes A and B. Through an SBI the physical network is discovered and translated into a generic internal representation called physical topology. Based on that, the OVC is able to create virtual topologies that are exposed to the clients through an NBI. The NBI is not bound to any particular protocol or interface definition. Typically, an SDN protocol is chosen that is supported by the client controller. The controller is then able to control the virtual network like any physical network. The OVC is implemented as a proof of concept for the experimental evaluation of the presented results.

1.5. Structure of the Thesis

The remaining thesis is structured as follows:

- **Chapter 2** proposes an intent model for an NBI that facilitates the creation of virtual topologies and hereby complements the existing model ecosystem. First, this chapter introduces the control and modeling of optical networks. Then, details on optical equipment and how SDN is applied to this domain are given. The chapter evaluates the existing models for optical network management and identifies a gap for the creation of virtual topologies, which is closed by the proposed model/interface definition.

- **Chapter 3** introduces new extensions to a shortest path algorithm that uses precomputation and balances the computation time and the memory requirements. A comprehensive definition is provided for the extension of the update procedures for dynamic changes of the graph. This is followed by proofs for the most important properties of the algorithm. Besides the algorithm, NV is discussed in the context of optical networks, including the resource assignment. The chapter finishes with a description of a procedure for an automatic creation of virtual topologies.

- **Chapter 4** presents the developed OVC that combines the functionality of a controller with a hypervisor layer. The software architecture as well as design decisions are explained. The chapter specifies, how the concepts, presented in the previous parts of the thesis, fit into the architecture and how they are implemented. It concludes with an experimental performance evaluation of the software stack.

- **Chapter 5** provides information on successfully conducted proofs of concept that showcase the integration of optical networks into multilayer scenarios. Examples include the automation of data center workflows and in-flight encryption for secure services. These examples are explained from an architectural as well as an implementational point of view. Each use case concludes with an evaluation.

These chapters are followed by a conclusion, the appendices and all references used in this thesis.

2. Models for Control and Virtualization of Optical Network Resources

The raise of SDN is leading to a paradigm shift in network operations. The separation of control and data plane pushed the development of (logically) centralized controller entities. These (SDN) controllers are responsible for the configuration of the data (or forwarding) plane. In the case of optical networks, every vendor has his own centralized NMS with access to the devices through proprietary interfaces. Here, SDN promises a simplification and unification of network management for optical networks. It allows the automation of operational tasks, despite the highly diverse and vendor-specific commercial systems and the complexity and analog nature of optical transmission. Fundamental components for Software-Defined Optical Networks (SDONs) are common abstractions and control interfaces. Currently, a growing number of models for optical networks are available, claiming an open and vendor-agnostic management of optical equipment. Such open models will pave the way for the introduction of Network Virtualization (NV) to optical networks. Nevertheless, some challenges in software-defined network operations remain unsolved. This chapter introduces the northbound control of optical network resources by controllers, applications and orchestrators alike and compares the most important existing optical models. Then, a mapping of these models to the concept of an optical network hypervisor is established. Finally, an intent interface for creating virtual topologies is proposed that is incorporated into the existing model ecosystem. The chapter is based on results that have been published in [SAE15] and [SAK17].

2.1. Control of Optical Equipment

The preferred technology option for long distances and high bandwidth demands is optical fiber transmission. In most of the current SDN developments, circuit-switched optical transport networks and specifically lambda-switched WDM networks have been neglected and are only slowly gaining attention. One factor is the layer of operation: whereas switches and routers process packet traffic on higher layers in the OSI model (2+), Optical Circuit Switching (OCS) is a layer 0/1 technology which is transparent to higher layers and unaware of any packet information. This means that a physical connection is assumed to be present without the need to establish it first.
Optical networks typically comprise vendor islands. One reason for this fragmentation is that every island has its own proprietary control plane making routing and forwarding decisions based on its limited intra-domain knowledge. In addition, every equipment vendor has his own closed NMS running on top, accessing the devices through proprietary interfaces. Although the communication builds on top of standardized protocols, it

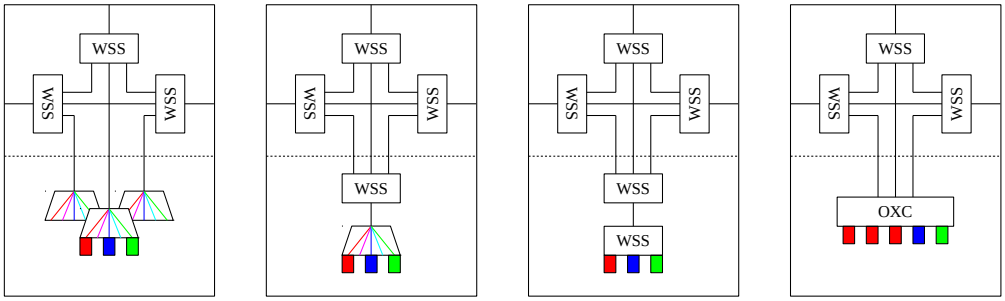


Figure 2.1.: Examples for ROADM structures: colored; directionless; directionless and colorless; directionless, colorless and contentionless [GE13].

contains vendor specific extensions. This leads to interoperability issues between different vendors, which equally apply to software and hardware components. Even though the NMS looks like a centralized controller and standardized models, e.g., ITU-T M.3000 [ITU00], are available, the NMS lacks open — southbound and northbound — interfaces so far. On the one hand, open northbound interfaces would allow the integration with applications and orchestrators. On the other hand, southbound communication toward the devices through standardized interfaces would enable the NMS to take over the role of a controller in multi-vendor scenarios. Unlike current Internet Protocol (IP)-based networks, the optical domain most commonly employs circuit switching. Therefore, packet-switching oriented protocols like OF are not naturally applicable. OF missed an opportunity in the beginning to include circuit-switched networks and has started a belated attempt to gain traction in this field with its optical extensions [Ope15]. Additionally, optical equipment is modular, which leads to node constraints based on the hardware configuration. The assumption that every (cross-) connection between input and output ports is possible does not suffice for a realistic model that will be adopted by vendors and operators. On a physical network level, the analog nature leads to network constraints, e.g., optical signal-to-noise ratio, crosstalk, and dispersion affect the maximum signal reach. They have to be taken into account before setting up services through the network and therefore need to be modeled correctly.

2.1.1. Reconfigurable Optical Add-Drop Multiplexers

ROADMs are one of the key elements for enabling a flexible optical network [Gri+10]. In general, a ROADM is connected to multiple fiber *directions*, which are also referred to as *degrees*, carrying a DWDM signal. The purpose of the ROADM is to flexibly direct incoming wavelengths to either another degree (*pass through* or *express*) or to a drop port and to add wavelengths coming from the node's client ports. The unit that allows for adding *wavelengths* (or *lambdas*) to an outgoing fiber and dropping them from

an incoming one is called *add/drop*. It developed from Optical Add-Drop Multiplexers (OADMs) with a fixed assignment for express and add/drop wavelengths. With wavelength blockers and later WSSs, the node structure became software controllable and thereby reconfigurable. A WSS is able to direct a wavelength from one of its input ports to the output port or to direct a wavelength from the input port to one of its output ports, if used in the opposite direction. Initially, ROADMs had two directions and were arranged in ring or linear structures. Today, much higher degrees can be achieved and the layout is more flexible.
There are three main properties that can be used to describe a ROADM [GE13; Gri+10]: colorless, directionless and contentionless. In simple setups, a fixed color is assigned to every add/drop port (Fig. 2.1a). Such configurations are called *colored*. Additionally, a separate add/drop is needed for every direction, making the configuration *directed*. Today, ROADMs are typically directed [Zha+13d]. This makes them inflexible due to the fixed assignment of transponders to directions, but wavelength contention can be avoided. By adding a WSS, which is connected to all degrees, the structure can be converted into a software-controlled *directionless* setup (Fig. 2.1b). This reduces the number of needed add/drop groups to service all degrees, although this setup introduces wavelength contention, which refers to the fact that every lambda can only be used once per add/drop group. By adding a second WSS (Fig. 2.1c), a *colorless* configuration can be achieved, always assuming that tunable lasers are used for transmitting. Any color that arrives at the lower WSS of the add/drop group can be directed toward any of the directions and vice versa by configuring the upper switch of the same group. One drawback is wavelength contention. With a *contentionless* extension of the ROADM, multiple copies of the same lambda can be added and dropped at the same group — as long as every wavelength is used only once per degree and other network constraints do not apply. This is typically achieved by means of an Optical Cross-Connect (OXC), which basically allows to direct any wavelength from an incoming port to any outgoing port. The exact realization of such an OXC is out of the scope of this thesis and can be found in literature [GE13]. This so-called Colorless, Directionless, Contentionless (CDC) ROADM solves this issue but is expensive and not scalable [Zha+13d].
For higher data rates, a flexible and programmable spectral bandwidth is beneficial [Gri+10]. For the migration toward flexible DWDM grids, it is not only required that the transponders support them but also the intermediate nodes, i.e., ROADMs [Zha+13d]. These flexible grid ROADMs have to be capable of redirecting any valid frequency band without introducing attenuation dips [GE13].

2.1.2. Optical Network Management Functions

An infrastructure provider, managing his network, needs different levels of granularity for his operations. On the one hand, calculating paths and installing services is preferably executed with a network-wide view, which hides low-level device details. On the other hand, device-specific settings and monitoring need a fine-granular node view with access to each device and its modules. For an RSA, a combination of both views might be needed to identify the internal node constraints as well as the network constraints.

Table 2.1.: Assignment of optical network management functions to scopes.

Network scope	Topology discovery, path computation, connectivity management, network monitoring, topology virtualization, multilayer operations
Node scope	Inventory discovery, device configuration, device monitoring

An overview of the management functions in optical networks and their mapping to a primary scope is given in Tab. 2.1.

From an SDN controller's point of view, the first and most important task is to establish a topology. This means discovering the devices and links of the managed infrastructure. Depending on the abstraction level, this may also include a detailed inventory discovery. It is a prerequisite for many tasks that rely on a network scope. One of them is path computation, which is performed on a topology representation of the network and needs to be aware of constraints. In a subsequent step, the computed paths are typically used to establish services or to optimize current assignments. Preferably, a service setup is executed on a network scope, defining the endpoints and a number of constraints, in a similar manner to an NMS. Nevertheless, a service setup can also be done on a per-device basis, leading to an increased complexity, due to the hop-by-hop setup, power balancing etc. The configuration of individual devices is also important for provisioning new equipment, changing fiber maps or defining port capabilities. In addition, monitoring is a major part of operations. On a device level, optical characteristics, like signal properties, can be captured and evaluated. If a controller or an NMS aggregates monitoring data from different sources, additional information and metrics with a network scope can be calculated, e.g., the location of a fiber break by correlating alarms.

The virtualization of networks is a field of growing interest and will become a more prominent part of network management (see Sec. 1.2.2). Virtual networks are defined by assigning virtual resources to their physical counterparts. Then, they are exposed to the respective clients, who are in control of their individual virtual resources. Likewise, (virtual) configuration and service requests need to be translated to hardware directives. An intuitive way of requesting virtual networks could be realized by an intent interface. An intent specifies what the user wants to achieve, without the need of being aware of how it is done. Even though the focus is on optical networks, other network layers at the edge are not ignored and the transitions, which are needed for efficient multilayer operations [Tom+14], are briefly considered. Finally, network management should be able to adapt to new technologies. One example for a current development are the standardized flexible DWDM grids (Sec. 1.2.3), which require different parameters for the spectrum assignment.

Depending on the scope, a subset of the presented management functions needs to be considered when designing new models and protocols, especially in the context of SDN.

2.1.3. SDN in the Context of Optical Equipment

An SDON is an optical network — OSI Layer 0 (photonic) and Layer 1 (Synchronous Optical Networking (SONET) / Synchronous Digital Hierarchy (SDH), Optical Transport Network (OTN)) — that applies SDN techniques. Such networks are mostly used for long-distance connections with high bandwidth demands, e.g., between data centers [Aus+14]. Service guarantees and dynamically changing traffic patterns lead to new requirements for these two layers [Kin+16]. The prospects of SDONs include an improved network utilization and a better multilayer and multi-domain control. Those objectives are accompanied by more agility in service provisioning and network optimization, which leads to lower operational costs and faster revenue recognition. Additionally, enabling optical network virtualization will allow multiple tenants (customers, applications, departments, etc.) to concurrently use shared network assets, in a fine-grained and spectrally efficient manner based on flexible WDM grids. Typical use cases have been described in detail in many places [ADH12; Aus+14; STR14]:

- Bandwidth on demand
- Virtual operators without own infrastructure
- Data center work flow automation, e.g.,
  - Storage migration and replication
  - Virtual machine migration
  - Distributed applications
- Multilayer interworking
- Network analysis and visualization
- Multi-vendor interoperability

The introduction of SDONs is the first step toward tackling these use cases. SDN is not a silver bullet solving all problems but it can facilitate some long overdue progress by improving network control, management and maintenance. Generalized Multi-Protocol Label Switching (GMPLS) control planes started moving toward SDN architectures with logically centralized controllers [Kin+16]. These SDN controllers collect information about the network, its topology and its state from all available sources, e.g., NMS and network elements. Based on this knowledge about the available resources, applications and network functions can be implemented on top of the controller. For that, standardized protocols between the controller and the network elements have to be in place first. The most promising protocols will crystallize based on the adoption by vendors and operators over the next years. SDN is able to improve the flexibility of optical networks by simplifying and automating the complex task of a manual connection setup through proprietary interfaces. While SDN improves the operations side, it misses features to allow a sharing of the infrastructure and giving direct access to tenants. This gap needs to be filled by other approaches that are discussed later in this thesis.

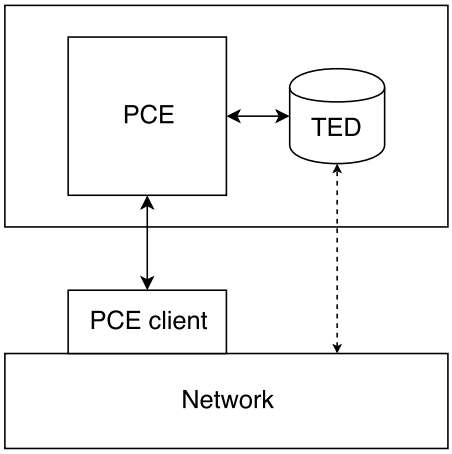


Figure 2.2.: Concept of a Path Computation Element.

The ONF tried to define an SDN architecture for transport networks [Vis+16]. They name two recursive concepts that simplify the description of transport networks. The first one is layering, which describes the utilization of an independent underlying layer — server layer — to transmit data of a client layer. The second one is the partitioning of networks into disjoint subnetworks. On top, they introduce the application plane as an additional component. For multi-domain scenarios, a top-level SDN controller, called orchestrator, is defined. In the case of multiple layers divided into separate domains per layer, a parent controller is coordinating the managed domains. For a single domain, the multilayer adaptations can be covered by one controller. Even though it is suggested to reuse flow identifiers to encode a center frequency and slot size, the concept of defining flows on a device level does not come natural for OCS.

One viable option for controlling optical networks and Elastic Optical Networks (EONs) is a Path Computation Element (PCE) [Dio17; VFA06]. It is an already available option for path calculations through a circuit-switched network that can be reused in the context of SDN. The PCE responds to path computation requests from clients with a Label Switched Path (LSP), satisfying the supplied constraints. To do so, the PCE queries a Traffic Engineering Database (TED), which contains information about the network status. The process of populating this database is out of the scope of this short overview. This general concept is visualized in Fig. 2.2. There are different incarnations of PCEs depending on the offered functionality. The most basic one is the stateless PCE, which only calculates paths based on the received constraints and the TED, which may be out of sync at the time of the computation. A stateful PCE [Cra+17a] additionally includes information about requested and existing LSPs. The last incarnation is an active stateful PCE [Cra+17b]. It provides the ability to initiate the setup of calculated paths. Therefore, the client no longer needs to trigger a setup in a subsequent step. A PCE by itself will not fulfill all SDN requirements, like topology dissemination, but it can be used as part of an architecture to compute paths through the managed network. One solution that embeds the PCE into a larger context is Application-Based Network

Operations (ABNO) [KF15; Kin+16]. Since ABNO is not widely deployed yet, a detailed description is omitted and the interested reader is referred to the referenced documents.

2.1.4. Control Protocols

All layers of an SDN architecture (see Fig. 1.3) depend on common protocols and interfaces. The ONF defines two generic interfaces: Control Data Plane Interface (CDPI) and Control Virtual Network Interface (CVNI) [Aus+14]. The CDPI is used for the direct communication between the provider's controller and his infrastructure, i.e., SBI. The CVNI defines the communication between the provider's controller and the client, i.e., NBI. Information exchanged through the CVNI describes a potentially virtualized view of the network. There is a number of protocols available which can be used for specific tasks. Most of the presented protocols are not limited to one of the interfaces, but they may be more suited for one of them.
Two examples for currently deployed protocols in optical networks are the Simple Network Management Protocol (SNMP) and GMPLS [GE13]. SNMP is defined by the Internet Engineering Task Force (IETF) and is used for the communication between network elements and the NMS. The protocol does not define which information is managed, but offers a way of defining it through Management Information Bases (MIBs). Control planes for Wavelength Switched Optical Networks (WSONs) usually utilize GMPLS and are typically separated from the data plane. It is a protocol suite that acts in a distributed way and covers three main tasks: routing, signaling and link management [Tom+14]. Examples for protocols are Open Shortest Path First with Traffic Engineering (OSPF-TE), Resource Reservation Protocol with Traffic Engineering (RSVP-TE) and Link Management Protocol (LMP) respectively. These days, it is commonly deployed in combination with a PCE, which takes over the path computation. Even though GMPLS follows the separation principle of SDN, it is not considered a viable option for SDN in general due to its complexity and the mainly distributed nature [DPM12; Pen+14]. In addition, the performance and the amount of communication has been shown to be worse with GMPLS than with an SDN approach [Gio+12; Zha+13e].
Coming from the field of packet switching, OF [McK+08] is the first protocol that emerged together with SDN, following the new concepts. OF is now governed by the ONF and it is briefly introduced in Sec. 1.2.1. One of its drawbacks is that it was originally designed to control packet-based equipment and is not easily usable with circuit-switched networks. Even though efforts have been made to integrate OF with circuit-switched networks in the beginning [Gud+10; Ji+14; Liu+11], it became clear that OF is not the best choice for this technology. Often, extensions have to be applied [Gud+10], or a mediation layer is required to make it applicable [EA12]. In version 1.4.0 [Ope13], OF introduced the notion of optical ports that represent the configuration of transceivers without considering node constraints. Further extensions are suggested in a separate description [Ope15]. Those attempts to promote OF for the optical domain did not succeed because other protocols had already filled this gap.
The network configuration protocol NETCONF [Enn+11] is a mechanism for applying Create, Read, Update and Delete (CRUD) operations to the configuration of network

devices. The available information is split into state and configuration data. While the former cannot be modified externally and only captures information about the state, the latter can be manipulated, assuming the client is authorized. All this data is stored in a datastore, which contains all available information about the network element. In general, NETCONF is an API that applications can use to interact with the datastore of every device. It uses Extensible Markup Language (XML) to represent and exchange data. The transmission itself is done through Remote Procedure Calls (RPCs) and is connection oriented. The RESTCONF protocol [BBW17] is a way of accessing a datastore that is compatible with NETCONF via Hypertext Transfer Protocol (HTTP). The intention is to allow web applications to perform a similar set of CRUD operations on the datastore. RESTCONF relies on independent (stateless) requests and allows an encoding of data in JavaScript Object Notation (JSON), in addition to XML.

Both protocols — NETCONF and RESTCONF — rely on YANG [Bjo15; Bjo16] models for the datastore. The models cover the configuration as well as state data. Additionally, available RPCs, which are operations with an input and an output, and notifications are defined. The data is organized in modules and submodules. Modules are allowed to import/include parts of other modules or submodules and extend existing data units by means of augmentation. Inside of modules, the data is organized hierarchically, building a tree-like structure. The nodes are assigned a name and either a single value or a set of children. Apart from built-in types, it is possible to provide own type definitions. Data structures like lists, which identify their elements by unique keys, are also included. Based on the model, the internal configuration of devices is described and a transport representation for protocols can be derived, which corresponds to an interface definition. Originally designed for devices, the models can also be used as an interface description for centralized network controllers, if modeled accordingly.

Some existing special-purpose protocols are also applicable to SDN. Two examples are Path Computation Element Protocol (PCEP) and BGP with Link-State (BGP-LS), although they are primarily defined for interacting with a PCE. PCEP [VR09] defines the communication between a path computation client and the PCE as well as between two PCEs. It is mainly used for path computation requests and the corresponding replies, which translate to LSPs. PCEP describes the messages and objects needed for these operations, while staying extensible for application specific requirements. A set of constraints can be included to restrict the selection of one or more LSPs. BGP-LS [Gre+16] is one option for collecting Traffic Engineering (TE) information from the network and sharing it. BGP-LS is based on the Border Gateway Protocol (BGP) and is well suited for filling a TED or a Link-State Database (LSDB).

There are various REST-based interfaces available northbound of the controller, which lead to a dependency between the application on top and the underlying controller. They triggered ongoing discussions on how to extract a unified set of directives for a common NBI. At the moment, the most promising approach seems to be using YANG models in combination with NETCONF and RESTCONF.

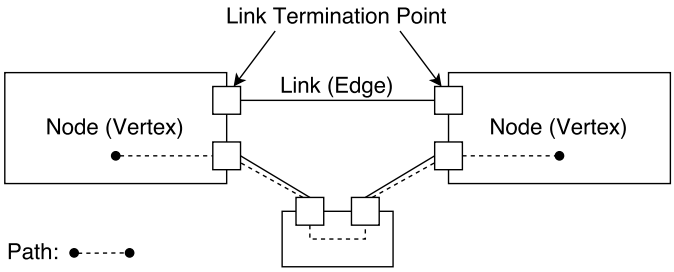


Figure 2.3.: Network terminology.

2.2. YANG Models for Optical Networks

Techniques for applying SDN to optical networks have been surveyed comprehensively in [Thy+16]. Thyagaturu et al. conclude that simplified management strategies are required, but they do not cover models for a centralized optical network management. Modeling optical network resources has been identified as an important research topic recently [Kin+16]. The modeling language YANG [Bjo15] facilitates the creation of new models and was already updated to version 1.1 [Bjo16]. It was originally defined as a data model for the configuration protocol NETCONF and inherently RESTCONF. YANG simplifies the creation of models that can be easily applied to protocols as well as code generation. All models have the goal of establishing an open unified network management for the base functionality of optical equipment, independent of vendors. The applicability of SDN to optical networks relies on such configuration and topology models in order to offer controllers for the optical domain or to include optical networks in a multilayer and multi-domain environment. For this reason, existing management models are ported to YANG, e.g., ITU-T G.874.1 [ITU16], and currently undergo a unification process, i.e., they are based on the ONF core model [LD15] and follow the same modeling guidelines. Extensions of these models provide the flexibility of adapting them to particular needs or technologies. This is done by means of augmentation — a mechanism provided by YANG. Operators expect to optimize their operations through unified models, assuming that controllers with a global view simplify network management and make it more dynamic and efficient through automation.

A variety of optical network models, covering diverse aspects of optical networks, has been published in recent years. They range from generic descriptions of optical network elements, in order to unify the configuration process, to network controller orchestration models, to allow the management of network-wide services [Vil+15b]. In this section, the most relevant models defined by standardization bodies and open industry alliances are surveyed and characterized. The focus is on specialized models that are trying to capture properties of optical networks and abstract them to a set of parameters — the least common denominator. An overview is given of the ones that currently receive the most attention among vendors, providers and operators. The selected models are ONF Transport API (TAPI), IETF TE Topology (TET), OpenConfig and OpenROADM. Before the analysis and evaluation of the models, the used network terminology is introduced.

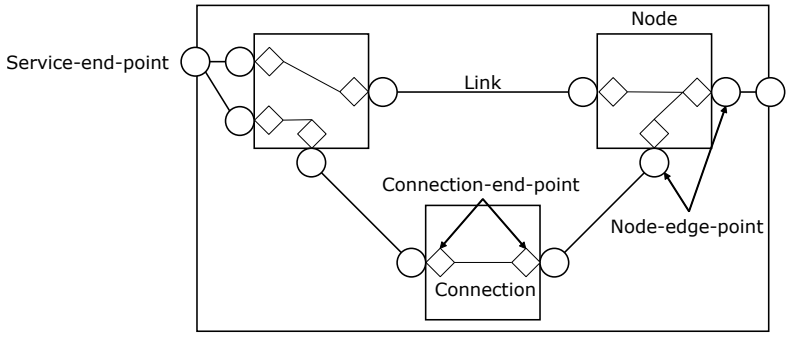


Figure 2.4.: TAPI main concepts and terminology.

Network Terminology

This section provides some general definitions of commonly used terms (see Fig. 2.3), which are mapped to specific definitions in the model comparison. First, a graph is defined as a set of vertices interconnected by edges. A network topology is a graph containing nodes (vertices) and links (edges). For the models, links are assumed unidirectional, if not stated otherwise. A generic node contains a number of Link Termination Points (LiTPs). LiTPs, also referred to as endpoints, represent attachment points for links. In general, they are not assigned to any particular layer, even though some models redefine them for special purposes and then assign them to layers. Links represent an adjacency between two LiTPs of two distinct nodes. A path lists the traversed nodes and links in a sorted order. The exact definition of a path differs in the presented models, e.g., its endpoint types. Virtual topologies, sometimes called abstracted or customized, refer to a network view, which is different from the underlying physical network topology (see Fig. 1.5). This is typically achieved by hiding details of the underlying network.

2.2.1. ONF Transport API

The ONF defines an API for transport networks, called Transport API (TAPI) [Ope18a; Qia+16]. The ONF Core Information Model (CoreModel) [LD15], which describes a generic network model, serves as a starting point for this purpose. The TAPI exposes information that is relevant for applications and controllers. Applications may use this interface to control network resources whereas orchestrators and controller hierarchies may interact with underlying controllers through this API. Even though the model is originally created in Unified Modeling Language (UML), the provided toolchain allows a mapping to YANG for an easier integration with protocols like NETCONF and RESTCONF as well as code generation.
The topology is defined as a set of network resources that consists of nodes and links, including extended Logical Termination Point (LoTP) from the CoreModel (see Fig. 2.4). Three LoTP types are defined in the TAPI. *Node-edge-points* cover the inward facing aspects, i.e., forwarding capabilities. Being the entry and exit points of a node, the endpoints are responsible for encapsulation. *Service-end-points* are concerned with the

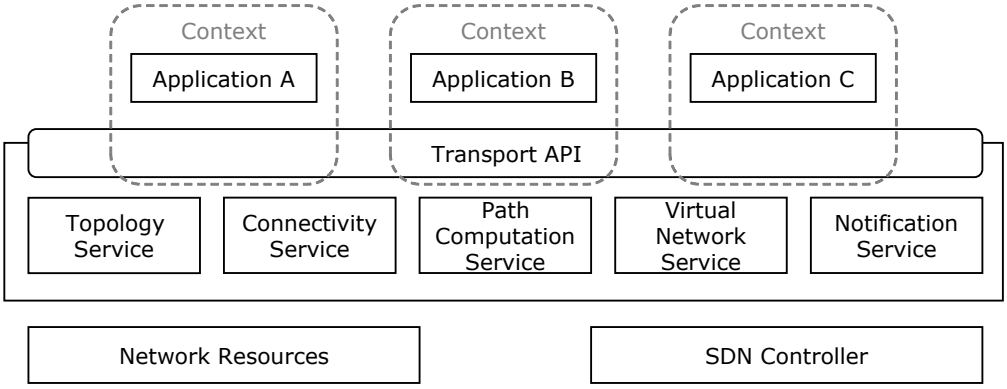


Figure 2.5.: Structure of the ONF TAPI.

outward facing aspects. They provide an abstracted view to the client and can map to multiple node-edge-points. *Connection-end-points* are the last type. They cover ingress and egress aspects of connections, which represent enabled forwarding capabilities, and have a client-server relationship with node-edge-points.
The TAPI covers a set of five control-plane functions and services (see Fig. 2.5). The *topology service* is concerned with the retrieval of information about topologies. Different granularities are defined, ranging from the whole topology down to individual node-edge-point details. The *connectivity service* is used to create and manage services between service-end-points. Connectivity is realized by underlying connections which represent forwarding behavior and are associated with connection-end-points. Connections can be built recursively and contain routes. At higher layers, a route is a list of connections in the underlying topology. At the lowest layer, they correspond to a switch matrix. The service interface does not necessarily require prior knowledge of the topology. The *path computation service's* main purpose is to compute and optimize point-to-point paths. The *virtual network service* allows the user to request virtual network topologies based on traffic constraints between pairs of service-end-points. They are implemented by reserved resources that can be controlled by the client. Finally, the *notification service* is one of the latest additions to the model. The notification types are currently limited to created and deleted objects and changed parameter values and states. Support for alarms and performance monitoring is scheduled for a future release. Any information exchanged through the TAPI is only valid in a particular context (see Fig. 2.5). It is shared between the (API) provider or server and its client.

2.2.2. IETF TE Topology

The IETF defines a number of different network and topology models. One prominent protocol, describing the topology of generic networks and extensible toward optical networks, is the TE Topology (TET). The TET is presented and explained following the hierarchical import dependencies summarized in Fig. 2.6. From top to bottom the following models are shown: a base network representation, a base network topology, the

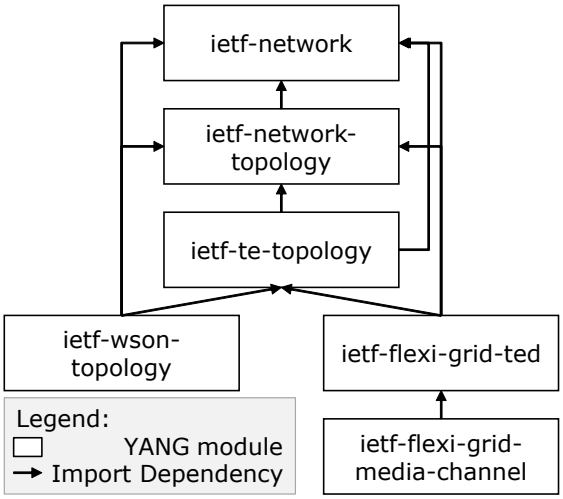


Figure 2.6.: Hierarchy of the considered IETF network and topology models.

TET and extensions for WSONs and flexible DWDM grids and media channels.

The *ietf-network* [Med+17] describes simple network hierarchies and their relation to each other. A network consists just of nodes and supporting networks, which correspond to underlay networks. The nodes are defined relative to a network and cannot be reused for multiple network instances. Nodes may have supporting nodes, which have to be part of one supporting network.
The *ietf-network-topology*, defined in the same draft [Med+17], extends the pure network model by adding termination points to the nodes and links that are interconnecting them. Therefore, topologies are supported while the previously defined hierarchies are preserved. Termination points and links may have supporting entities of the same type. Keeping the generic nature, there are no constraints on the particular implementation, e.g., physical port or logical port.
Both models need to be augmented for any technology-specific network or topology type. Some parts of the model are explicitly expected to be augmented by models implementing a particular technology, e.g., the network type attribute.

The TET defined in *ietf-te-topology* [Bee+16] describes one way of storing TE data in a TED. This representation supports path computation that is aware of physical constraints while being independent of any protocol. The vanilla TET defines a generic view of a given network that is discovered using any available technique or protocol. In general, the TET is the control-plane representation of the data-plane topology. The TET can have more than one data source and captures static attributes as well as dynamic ones that need to be updated regularly. The representation of a topology for a single provider may contain one or more layers corresponding to different network layers. The topology provider can create virtual overlay topologies for clients. In this context, a overlay topology is called *abstract* or *customized* TET and typically enriches the client's

native view by augmenting the provider's native topology. In the case of augmentation, the path computation is executed on top of the client's native TET which may contain customized views.
Every TET comprises TE nodes and links, which are terminated at one LiTP on each side. A TE node represents a fraction of one or multiple underlying nodes and is bound to a particular topology. Multilayer TE nodes can be decomposed into a client and server layer. A layer transition is represented by a link between those nodes — the transitional TE link. This kind of link represents a potential connectivity and is removed as soon as the connection is established. It is replaced by a TE link in the client layer topology. A Tunnel Termination Point (TTP) represents a transport service endpoint such as a WDM transponder. The accessibility between server LiTPs and TTPs is captured in the local link connectivity list, while the connectivity matrix shows valid interconnections between LiTPs. A calculated TE path based on this representation describes the nodes and links of a potential connection.
TETs can build hierarchies in which an underlay topology, a *native* topology at the lowest level, serves as a base for an overlay topology. Multiple overlay topologies can be built on top of the same underlay topology, e.g., views for different clients. Also, multiple underlay topologies can be the input for one overlay representation, e.g., views from different providers. A slight variation are topologies provided by multiple data sources which need to be merged before they can be used. The process of combining multiple views includes three main steps: identifying neighboring domains, renaming identifiers to make them unique and finally capturing layers. The TET can be used to represent and transport all described types of topologies as well as to manipulate abstract ones.
Notifications are an important part of topologies, therefore the model relies on a subscription and push mechanism for YANG datastores. The TET is also compatible with scheduling parameters and supports a set of optional features. One was presented before: the overlay/underlay relationship, which is particularly useful for vitualization purposes. Another feature is the template configuration that allows for a definition of default templates for TE objects. Since the TET description is technology-agnostic, any particular technology needs to augment the model to manage any technology-specific information.

To implement a network representation, the TET needs to be augmented with technology-specific details. Next, two models that are relevant for optical networks are presented. The first one focuses on the rigid DWDM grids and the second one addresses flexible frequency slots (see Sec. 1.2.3).
The *ietf-wson-topology* [Lee+17] adds a description of impairment-unaware WSONs. The goal is to support RWA. Important extensions are node types for fixed and reconfigurable optical nodes (ROADMs) and the augmented WSON connectivity matrix for describing available cross connections.
The flexi-grid extension (*ietf-flexi-grid-ted / media-channel*) is defined in [Mad+16]. It comprises two definitions: a TED definition for flexi-grid equipment and a media channel description for paths. The TED part defines three types of flexi-grid nodes: node, transponder and sliceable transponder. The node represents a wavelength switch based on a connectivity matrix. A transponder defines transmission parameters, e.g., modu-

lation format. The actual grid information is stored inside the links. A separate media channel description extends the TED representation by a path definition including the spectrum assignment.

2.2.3. OpenConfig

OpenConfig is a project by network operators to define model-driven network management interfaces, which are vendor-independent. The models try to capture configuration and state parameters. Additionally, performance monitoring is a central part of the models. Only the five models related to optical transport that are available in version 0.4.0 are considered [Ope18d]. Two of them cover general definitions and types, while the others correspond to amplifiers, terminal devices and wavelength routers.
The *openconfig-transport-type* model is a collection of types and identities for optical devices. The first common data elements are defined in *openconfig-transport-line-common*. Optical line ports are assigned a certain type, i.e., in, out, add or drop. The Optical Supervisory Channel (OSC) configuration captures the available OSC interfaces.
The *openconfig-optical-amplifier* model describes optical amplifiers that are deployed as part of the transport line system, e.g., Erbium Doped Fiber Amplifier (EDFA) and Raman amplifiers. The gain parameters, output power and mode of operation can be configured and the actual gain values and the input power are captured.
Terminal systems — client and line side — are defined in *openconfig-terminal-device*. The terminal system description is following the client-to-line direction, while the opposite direction is implicit. The physical port represents a physical, pluggable client port on the device with operational state and performance monitoring. Each physical port has one or more physical channels. Their main purpose is to allow individual monitoring of channels that build up the full port capacity. From a model perspective, the logical channel ingress defines the contributing transceiver and the corresponding physical channels. Logical channels are a grouping used for representing logical grooming elements. They are either assigned to another logical channel, in order to add another stage of multiplexing (or de-multiplexing), or to an optical channel that corresponds to the line side transmission and assigns a carrier with a wavelength or frequency. The two defined protocol types for logical channels are Ethernet and OTN. In general, the model assumes that the NMS will verify correct combinations of protocols.
Finally, the *openconfig-wavelength-router* model contains a definition for optical transport line system nodes or ROADMs. A wavelength router is defined as a configurable switching element. A media channel is described by a lower and upper frequency, therefore specifying a frequency band. This media channel is then assigned an input and output port, thereby defining the flow inside of the node.

2.2.4. OpenROADM

OpenROADM is an initiative defining a white-box model (version 1.1) for ROADM-based optical equipment that can be used by a control plane, i.e., OpenROADM controllers [Ope16; Ope18f]. It creates a network view that abstracts vendor-specific devices to

a generic device representation, which can be used for service instantiation or path computation. The modeling is closely coupled to the structure of a ROADM node, leading to two basic building blocks, direction/degree and add/drop group. A direction or degree is the block that is connected to the degree of another ROADM node on the one side and the internal structure on the other side. Add/drop groups are units that allow adding and dropping wavelengths between fibers and client ports. In the model, these client ports are grouped together based on Shared Risk Groups (SRGs). The connected client equipment, including transponders and routers, is called tail.
Degrees have two types of (logical) Termination Points (TPs): trail and connection TPs. The trail TPs are facing toward a degree of an adjacent ROADM and terminate Optical Multiplex Section (OMS) trails. Connection TPs, on the other hand, are used to connect add/drop ports and pass express traffic inside of a node.
An add/drop group is a construct consisting of WSSs, amplifiers and splitters/combiners for transmit and receive. Transponders are connected to add/drop ports and Connection Points (CPs) are facing the degrees. Add/drop groups are assigned one or more SRG depending on the hardware configuration, e.g., Colorless, Directionless (CD) or CDC. Every SRG contains one pair of CPs. In addition, various alarms and performance measurements are available at this level.
Logical connectivity links are a tool to represent the connectivity/cabling between building blocks. They cover external connectivity between degrees and the internal cabling between degrees and SRGs. Express links are used for transit traffic, internally passing from one degree to another (connection TPs). The connectivity map is based on input from planning. Wavelengths are represented as fan-out at the connection points. Each wavelength receives an own node. It multiplies the number of TPs by the number of available wavelengths per direction.
The path computation for a service can be performed between either tails or nodes. In the second case, appropriate SRGs and transponders (if available) are returned. The services are purely wavelength-based ignoring any other parameters. Only routing constraints may be applied to influence the outcome of the path computation, e.g., diversity, include, exclude.

2.2.5. Comparison and Evaluation

The models are first classified and then evaluated based on the management functions described in Sec. 2.1.2. The results are summarized in Tab. 2.2.
The applied classification is following the description in [BCM17]. It covers two dimensions: the module's abstraction layer and its origin type. The abstraction layer describes the scope of the module, i.e., network service or network element. Available origins are standardization, vendors or users. The TAPI and TET are standard models for network services. OpenROADM and OpenConfig, on the other hand, are models at the network-element level and are vendor-specific, according to the definition. Even though, OpenROADM can be seen as having a network scope to some extent. Three of the surveyed models are able to represent topologies. Only OpenConfig is lacking the notion of links or equivalent entities. This makes it unsuitable for path computation on

Table 2.2.: Evaluation of the selected optical models.

	Transport API	TE Topology	OpenConfig	OpenROADM
Module abstraction [BCM17]	Service	Service	Element	Element (+ Service)
Module origin [BCM17]	Standard	Standard	Vendor	Vendor
Topology representation	Yes	Yes	No	Yes
Path computation interface	Yes	No (TED for PCE)	No	No (TED for PCE)
Network wide operations (e.g., services)	Yes	Only with active PCE	No	Yes
Explicit device configuration	No	High-level	Yes	Yes
Notifications and performance monitoring	Notifications (alarms and PM planned)	Notifications	Yes	Yes
Network virtualization	Yes	Yes	No	No
Interworking with other layers	Predefined set	Through extensions	On a device basis	Transponders and pluggables
Flexible DWDM grids	No	Yes (Extension)	No	No

its own. While the TAPI provides an application interface for retrieving paths, TET and OpenROADM are representations that can be used to perform the actual computation. The calculated paths can be installed through the connectivity service of the TAPI or the service RPCs of OpenROADM. The TET can only be used in combination with an active stateful PCE to manage services. OpenConfig is not capable of such network wide operations without an additional entity that has knowledge about the topology. However, both device-oriented models, i.e., OpenConfig and OpenROADM, are able to configure individual network elements. The TET is limited to the information available for the path computation, whereas the TAPI focuses on network-wide operations but individual devices are out-of-scope. Notifications and performance monitoring are management tasks that the protocols centered on network elements support very well. Being intended for a TED representation, the TET does provide notification mechanisms about changes. The same is true for the TAPI, and further extensions toward alarms and performance monitoring are planned. Virtual topologies are only supported by the TAPI and TET.
At the edge of the optical network, an interworking with other layers is required. The TAPI defines a set of known signals, e.g., OTN and Ethernet. The generic TET requires extensions to support any technology specifics but is very flexible in using them. OpenConfig comprises models for many protocols, like IP, Ethernet, Multiprotocol Label Switching (MPLS), but the interworking takes place at an individual device level. Finally, OpenROADM supports transponders and pluggables at its edge. For external devices, additional information needs to be handed over to the controller.
The final point that is investigated is the consideration of current developments. Flexible DWDM grids are picked as an example that has been standardized but is not widely deployed yet. Most of the models do not provide any information on the applicability to flexible-grid scenarios. Therefore, the current definition and the needed adjustments are evaluated. In the TAPI, a very generic definition of a central frequency or wavelength is provided. This is not enough to describe flexible grids but it can be a starting point for augmentation. For the TET, a specialized extension is already available. In the case of OpenConfig, one frequency value per optical channel is available. Therefore, it is not capable of representing the new grids without changes. OpenROADM is focused on capturing the current generation of ROADMs. Especially the representation of wavelengths by individual connection points makes an adaptation very hard without major changes of the model.
Now, the conceptual approach presented in Sec. 1.4 is revisited. As a reminder, it introduced a mediation layer that allows applying SDN and NV to optical networks, while at same time providing a migration strategy for legacy equipment. The original concept (see Fig. 1.7) does not define any models and interface implementations. Since all the presented models have a particular application area in mind, they are mapped to this concept. The resulting assignment is shown in Fig. 2.7 (right-hand side). Starting from the top, the TAPI covers a wide range of interfaces needed for applications and controllers. This makes it a good fit for the NBI of the hypervisor, which requires support of network-wide operations. In general, the IETF TET is an extendable representation of networks for TEDs. This makes it applicable to the internal representation of the

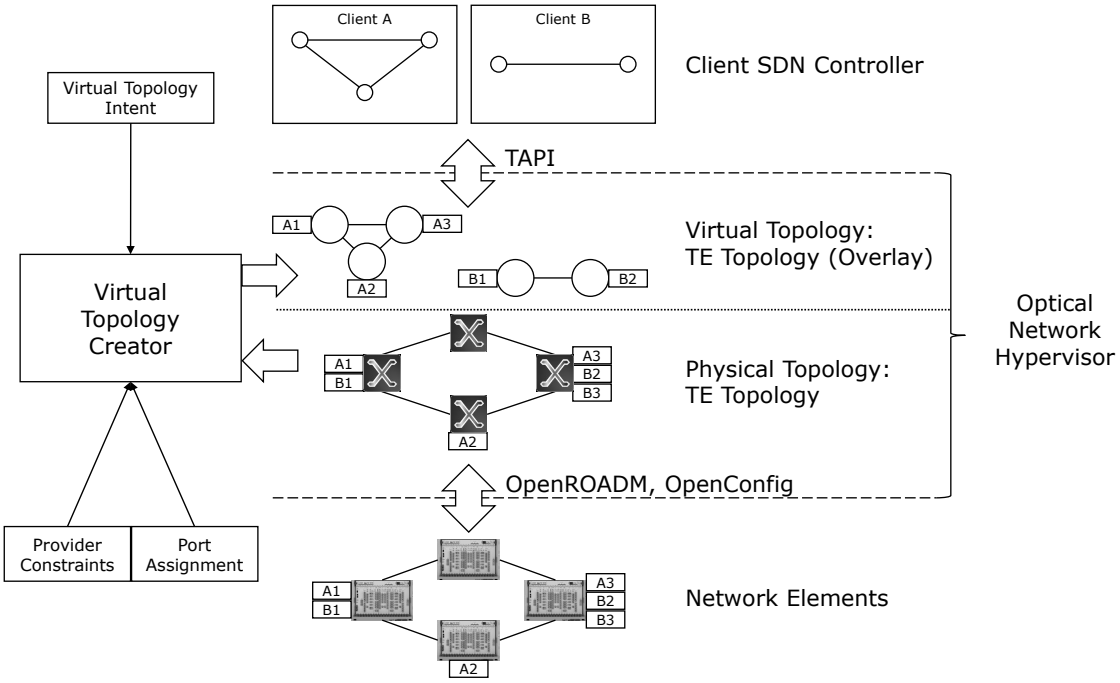


Figure 2.7.: Generic optical network hypervisor architecture with a virtual topology intent interface (left-hand side) and the model assignment (right-hand side).

hypervisor. The overlays are a good fit for creating virtual topologies for clients. The functionality that is usually added by an attached PCE is taken over by the hypervisor. For the configuration of the network elements, i.e., the SBI, OpenConfig and OpenROADM are possible solutions. The optical part of OpenConfig is still in an early stage and focuses on the configuration and monitoring of individual components of an optical transport line system. Unfortunately, many parts of the optical model are currently undefined. OpenROADM provides a device as well as a network view. It is very operations oriented, meaning all components have associated management parameters like physical location and vendor information.

Two of the presented models, i.e., TAPI and TET, allow a creation of virtual topologies. Chowdhury and Boutaba identify defining interfaces for network virtualization as one key research challenge [CB09; CB10]. So far, there is still no intuitive and user-friendly way for a customer or an application to define a topology by simply indicating his or her requirements (intent). Compared to network embedding, creating a virtual topology for optical networks, which can then be controlled by a client SDN controller, does not include an assignment of nodes in the network. The endpoints for potential connections are allocated to the customer by the infrastructure provider beforehand. The remaining task is to create links based on impairment-aware RSA in order to receive a topology that is exposed to the tenant.

2.3. Definition of Virtual Topologies

A way of defining virtual topologies is an important requirement for pushing the virtualization and slicing of optical networks for multiple tenants and scenarios. Current work at the IETF [Zha+16] mentions two ways of creating a virtual topology: the first one is a direct definition in one of the topology models and the other one is based on a traffic matrix, whose structure has not been defined yet. The first approach requires an administrator, most likely the provider, to explicitly specify the virtual topology following the used topology model. Such manual configuration processes are cumbersome and error-prone. The second approach is based on a traffic matrix estimating the needed bandwidth between endpoints. Based on this information, an appropriate topology will be generated, assuming the expected traffic is known beforehand. Nevertheless, this requires a standardized definition of the traffic matrix. The first approach is supported by the TET, and the TAPI follows an approach similar to the second one.

Some work on intent interfaces for network virtualization has been carried out recently. Cohen et al. suggest an intent-based network definition using directed graphs [Coh+13]. Those graphs are called network blueprints and capture the connectivity as well as the policies. In a blueprint, endpoints are grouped in policy domains (nodes) and connected via policies (links). No details about the available parameters, the assignment of nodes and an exchange model have been published. Policy domains bundle endpoints that need to be treated similarly regarding policies like intrusion detection or latency. The policies are applied when transitioning between domains. Due to the application of packet encapsulation, the architecture is only applicable to packet-switched networks. An exact list of the available parameters, the assignment of endpoints to domains and an interface model toward the client have not been published.

Han et al. introduce a platform that uses an intent interface for defining virtual networks in an SDN environment [Han+16]. Their goal is to automate the process of composing and embedding virtual networks. The platform's architecture follows a layered approach and combines the functionality of a hypervisor with an SDN controller. Three groups of virtual topology intents are introduced in the sense of high-level policies. A topology intent describes a connectivity between endpoints. Endpoint intents capture the requirements for relations between endpoints, like bandwidth. Finally, chain intents extend the previous one by chaining intermediate middle boxes and Virtual Network Functions (VNFs). The implementation is based on OpenVirteX [AlS+14] and Open Network Operating System (ONOS) [Ber+14; ONO18b], reusing its intent framework. Unfortunately, no details on the intents or their modeling is given. Since the slices are managed through the OF protocol, the proposed architecture is only applicable to switches.

Neither of the proposed ways of defining virtual networks has enough details for an adoption. This makes it hard to judge the ease of use, especially for the intent-based approaches. The presented work focuses mainly on packet-switched networks using encapsulation or the OF protocol and does not take into account OCS.

2.3.1. Model for an Intent-Based Definition of Virtual Networks

Current solutions for creating virtual topologies, like entering the network layout manually, are overly complex, error-prone and most of all cannot be easily defined and modified by the client. The customer should be able to express his topology request based on a set of simple intents. An intent specifies what the user wants to achieve without the need of being aware of how it is done. This interface is a missing piece in the management of SDONs . This work proposes an intent interface model to solve it [SAK17]. The intents that describe the high-level requirements defined by the customer are the main outcome of this chapter. Then, the resulting topology can be represented, exchanged and controlled based on existing optical models like the TET and the TAPI. The needed input for the virtual topology creation are the assigned client ports, provider constraints and the client's intents. The creator is shown on the left-hand side of Fig. 2.7. Provider constraints are out of the scope of this thesis but may constrain the visibility of network details. The created topologies can then be managed by the client's SDN controller.

The proposed interface model enables the user to define virtual topologies expressing his requirements in the form of intents. In Fig. 2.7, it corresponds to the virtual topology intent that is the northbound input to the virtual topology creator. The interface is defined in the modeling language YANG. Therefore, it can be easily translated into a protocol implementation based on NETCONF or RESTCONF. The full model that describes this interface that can be used for defining, requesting and manipulating virtual topologies can be found in App. B. Two main data entry points are available: `endpoints` and `topologies`.

The entry point `endpoints` and its children represent state that cannot be changed by the user. It contains a list of all `assigned-endpoints` that the client who is querying the interface is allowed to use. It is important to give the client a way to access this information as a starting point for his virtual topology requests. The assignment is read-only and is maintained by the provider based on existing agreements. The endpoints themselves are generic and can represent nodes, modules or ports, depending on the level of abstraction that is used for representing the topology. Additional information can be added by means of augmentation, e.g., user-friendly descriptions. From the point of view of virtual-topology creation, only a unique endpoint identifier is needed.

The configuration and therefore the creation is carried out through the data stored in `topologies` and the respective subtree. This top-level element contains a list of `installed-topologies` for the querying user. These topologies comprise an identifier (`topology-id`) as well as a list of `intents`. The uniqueness of the topology's identifier is enforced by the key property for the list in the YANG model. This is also true for the other identifiers. The individual intents combine a set of `endpoints` with requirements that need to be fulfilled by these endpoints. The available parameters include the `dedicated-bandwidth`, which is exclusively reserved, and the `flexible-bandwidth`, which is shared with others and might not always be available. Both are expressed in Mbit/s and thereby give the user an easy way to define a topology based on a bit rate instead of expecting him to be familiar with technology-dependent details, like flexible WDM grids and modulation formats. Additionally, parameters addressing properties

```
{
  "endpoints": {
    "assigned-endpoints": [
      {"endpoint-id": "A1"},
      {"endpoint-id": "A2"},
      {"endpoint-id": "A3"}
    ]
  },
  "topologies": {
    "installed-topologies": [
      {
        "topology-id": "Client A",
        "intents": [
          {
            "intent-id": "Intent A",
            "endpoints": ["A1", "A2", "A3"],
            "dedicated-bandwidth": 10000,
            "flexible-bandwidth": 5000,
            "minimum-paths": 2,
            "disjoint-paths": "link",
            "protection": false,
            "maximum-active-connections": 2,
            "satisfied": true
          }
        ]
      }
    ]
  }
}
```

Figure 2.8.: JSON representation of an example for a virtual topology configuration for "Client A" resulting in the depicted virtual topology in Fig. 2.7.

of the available paths, i.e., links in the virtual topology, are included. Those comprise the minimum number of parallel paths (`minimum-paths`) and their disjointness requirements (`disjoint-paths`), e.g., link or node disjoint. It is also possible to request an optical `protection`. Finally, the maximum number of parallel connections is included (`maximum-active-connections`). It defines how many connections may be active at any given point in time. This means that the links define potential connections but only a certain number of them can be provisioned by the user at the same time. Each intent contains a read-only flag (`satisfied`) that indicates if the virtual topology creator was able to satisfy all requirements of the intent. Based on a number of one or more such intent groups, a topology is created and exposed to the client's SDN controller, which is

then able to control the virtual network on demand. An example for assigned endpoints and a topology based on one intent is shown in Fig. 2.8. If the described YANG model was used to define a RESTCONF interface, the JSON representation would match a response to a request for the current configuration of the virtual topologies. The example corresponds to client A's topology in Fig. 2.7.
The proposed model complements the model landscape by providing a way of maintaining virtual topologies. While the existing models are used to control the network and expose (physical and virtual) topologies, the new intent interface offers capabilities for changing the mapping between topologies according to the client's needs. Being defined in YANG, the developed model can be easily translated to protocols.

Summary

This chapter evaluates existing models for the management of optical networks. An interface for manipulating virtual topologies is identified as a gap. Therefore, an intent model is proposed to offer a solution that makes it easy for clients to request, change and delete virtual topologies. The creation of virtual topologies from the received intents requires mapping physical resources to virtual ones. For the links, a calculation of paths through the network is required, which is the topic of the next chapter.

3. Virtualization of Optical Network Resources

Network Virtualization (NV) is gaining an increasing amount of attention in the field of networking. The idea is to split and share physical network resources between different tenants, while giving them control over their network slice, i.e., the part of the network assigned to them. To abstract from the underlying technology and to offer isolation between tenants, the network is exposed as a virtual network. Based on the interaction with an (SDN) controller, a virtual network is (mostly) indistinguishable from a physical network comprising hardware. Optical networks have different properties than packet-switched networks, e.g., the transmission is analog and they are typically circuit-switched. This needs to be taken into consideration by the virtualization process. Usually, a network hypervisor is used as a mediation layer between the physical network and the tenant's controller. Before a virtual network can be exposed, a resource assignment needs to be carried out. It starts with a request that is submitted by the tenant, describing the expected network. The associated requirements need to be mapped to the infrastructure. For optical networks, the node mapping is assumed to be given because the tenant's equipment is available at a predefined location. Only the links require a mapping, which can be achieved by means of Routing and Spectrum Assignment (RSA). In a two-step approach, the routing and the spectrum assignment are split into individual tasks. The first step, i.e., the routing, usually involves a shortest path computation. The naive procedure calculates a path for every link of a virtual topology request using one of the well-known algorithms, e.g., Dijkstra's algorithm. Since it can be expected that many requests (and links per request) have to be handled by the hypervisor, a more efficient solution is needed. Based on the shortest path the spectrum is assigned, which represents the second step. This concludes the mapping of virtual links before exposing the virtual topology to a tenant. This chapter covers NV in optical networks, proposes novel extensions for a shortest path algorithm and describes the process of virtual topology creation.

3.1. Network Virtualization

NV is seen as one of the important properties for next-generation networks [CB09; CB10]. A basic definition has been given in Sec. 1.2.2. Chowdhury and Boutaba define a virtual network as a set of virtual nodes and virtual links, which build up a virtual topology. This topology is a subset of the underlying physical infrastructure. Each virtual node is hosted by one physical node, whereas virtual links are paths through the physical topology, potentially passing multiple nodes in between. NV enables the sharing of

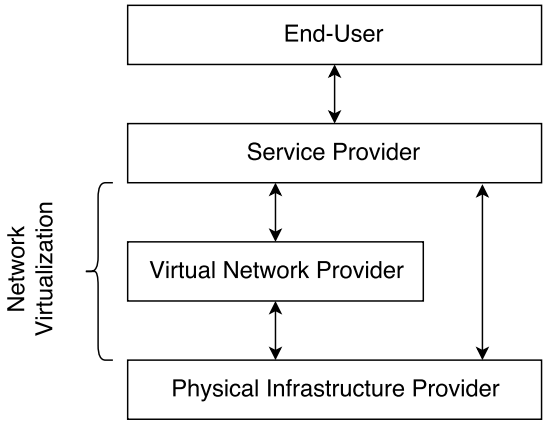


Figure 3.1.: Roles and dependencies in a virtualized network.

network resources between multiple tenants, including the recursive creation of virtual networks, i.e., the virtualization of virtual network resources. The introduction of NV also leads to new roles in the provider landscape [Sch+09]. A summary of the roles and dependencies is presented in Fig. 3.1. The original split between ISPs and Service Providers (SPs) is extended by specialized roles. The ISP is primarily responsible for providing access to the Internet based on own or leased infrastructure. Since the role of the ISP often includes multiple tasks, e.g., providing infrastructure and services, a clear naming convention is introduced. The Physical Infrastructure Provider (PIP) operates its own infrastructure and offers it to customers. The Virtual Network Provider (VNP) is a type of infrastructure provider without an own network. He relies on a leased infrastructure from a PIP or another VNP. The difference between a VNP and a PIP is that the VNP provides connectivity on top of a leased network and thereby, introduces an additional layer of indirection. Both network providers expose virtual resources through programmable interfaces. The SP offers a service of any kind based on infrastructure. His network may comprise slices provided by different sources, i.e., infrastructure providers. The SP then exposes services to the end-user, who terminates the service. Multiple provider roles might be covered by the same entity. NV especially plays a role between the PIP and his tenants as well as the VNP and the SP.

3.1.1. Network Hypervisor

For the realization of network virtualization, a concept similar to virtual machines can be applied to networks, where a hypervisor abstracts physical network resources into virtual ones. Previously available technologies that are related to virtualization in networks, e.g., VPN, MPLS or VLAN, provide link virtualization instead of network virtualization [4WA09]. Also, attempts have been made to virtualize hardware like routers in software and to deploy them on commodity servers [Egi+10; KG08], which differs from the original goal of virtualizing the network hardware itself. Fueled by SDN and its centralized control and unified interfaces, a new generation of hypervisors emerged. The first SDN

network hypervisor that abstracted packet switches is FlowVisor [She+09; She+10]. It is an intermediate layer between an OF controller and OF switches, intercepting messages and translating them according to the slice definition. The FlowVisor assigns a flowspace, which is defined by a number of OF headers, to every slice. It makes sure that these flowspaces do not interfere with each other and that the clients are isolated.
An extensive survey on hypervisors for SDN has been presented in [Ble+16]. The focus is on packet-oriented hypervisors for OF networks, even though a few special-purpose hypervisors are briefly mentioned. Blenk et al. identified three main network attributes that can be abstracted, i.e., topology, physical node resources and physical link resources. These three attributes form the basis, i.e., input, of the hypervisor. The survey also presents three types of network attributes that are relevant for isolation, i.e., the control plane, the data plane and the virtual SDN addressing. All three of them require different methods for ensuring that no unwanted interference between tenants occurs. Finally, a classification of the surveyed hypervisors is presented based on their architecture — centralized vs. distributed — and the execution platform they are running on. The platforms include general-purpose hardware, generic NEs as well as special NEs with included hypervisor functionality. This classification schema is also applicable to optical network hypervisors, even though some factors are less relevant, e.g., the addressing.
Most hypervisors do not consider optical networks and therefore, do not take into account optical transmission and its implications. Circuit switching and optical constraints have to be explicitly considered by NV in order to provide correct results. Examples are wavelength continuity, optical impairments and connectivity constraints [Guo14]. On a network scope, the level of abstraction is an important parameter [Aut+14; Guo14]. While (a part of) a network virtualized into a single switch is easy to expose, it might hide too many details of the underlying network. At the other extreme, a direct, i.e., one-to-one, mapping exposes the full complexity of the network, which needs to be handled by the controller. Intermediate virtual topologies present a trade-off between both extremes and are a suitable option in many cases. On a node level, optical layer constraints need to be captured in order to provide a meaningful topology [Guo14]. These constraints are related to internal configuration of the hardware and may limit the viable options, e.g., for the applicable wavelengths or directions. One of the first attempts of providing a NV mechanism, while capturing the impairments of an optical network, was done in [Azo+12]. This optical FlowVisor followed the FlowVisor approach that was used for packet switches and communicated through OF. A summary of other existing virtualization techniques and hypervisors for optical networks is available in [Thy+16].
Since OF is considered unfit for optical networks [Cas+18] and new protocol definitions are emerging (see Sec. 2.2), a discussion regarding the best protocol(s) for NV is ongoing. The general concept of a mediation layer still persists, only the OF protocol is no longer considered for the control of optical NEs and therefore, as the SBI of the hypervisor. Martinez et al. suggested to use the Control Orchestration Protocol (COP) [Mar+17], which has been an influence on the TAPI. The TAPI itself is also considered a viable option [Jan+16]. In addition, the previously presented TE Topology (TET) is capable of representing virtualized optical networks, even though it is not very prominent in

publications. In summary, more investigation is needed before an agreement can be reached. Only if a virtual network is defined and resources are assigned, it can be exposed through a hypervisor. These two topics are discussed next.

3.1.2. Defining Virtual Networks

Defining a virtual network is an indispensable step before it can be used by a client. A general approach for requesting a virtual network is to define its structure based on nodes, links and some resources offered by both, e.g., bit rate or wavelength. Typically, this is employed by Virtual Network Embedding (VNE), which is introduced in Sec. 1.2.2. The requested network description is handed over to the embedding algorithm, which tries to accommodate it in the existing network. This embedding algorithm maps two entities: nodes and links. The embedding itself is a computationally hard task (NP-hard), even if limited to assigning the links [Fis+13]. For smaller problem sets optimal solutions can be computed but for larger instances heuristics are applied. A general survey on methods and their classification is presented in [Fis+13]. It shortly discusses optical networks without going into details, acknowledging that physical layer impairments apply. Additionally, wavelength continuity constraints and transmission settings need to be considered. Attempts for including them in the VNE have been carried out [Pen+11]. Existing Integer Linear Programming (ILP) solutions for allocating Virtual Optical Networks (VONs) [Pag+12a; Pag+12b] are too slow to be applicable to network management. Several minutes for just a few nodes are not acceptable. Current work for optical networks mostly focuses on mapping nodes according to some heuristics [Wan+15; Zha+13c; Zhu+15; ZSB13] and uses a shortest path calculation for the links. Vilalta et al. compare a shortest path with a subsequent spectrum allocation with a spectrum-aware shortest path [Vil+14]. The spectrum-aware shortest path iterates all central frequencies, removes all edges that do not have enough slots for the respective central frequency and checks the shortest path on top of this pruned graph. Their results are not very detailed and the scalability of this approach is not verified. RWA in conjunction with VNE has been considered by [Zha+13b]. Even though the paper mostly focuses on the flexible assignment of nodes, it applies RWA based on a fixed shortest path in the mapping process. Thereby, it tries to reduce the number of wavelengths used.
For optical networks, it is unlikely that the nodes need to be mapped based on computational resources. The equipment is usually not directly connected to servers and it is mostly used to transfer big amounts of data between selected endpoints. Therefore, this thesis assumes that the endpoints are assigned to tenants, who can choose the kind of topology they need based on the assigned endpoints. This reduces the definition of a virtual network to the mapping of virtual links.

3.1.3. Assigning Optical Resources to Virtual Entities

Virtual networks always rely on a mapping to an underlying network, irrespective of its physical or virtual nature. The resources of the underlying network are mapped to

their respective virtual resource. There are two main (physical) resources that need to be assigned to their virtual counterparts: nodes and links. The nodes can be further partitioned, e.g., into ports. Since this thesis assumes that in optical networks, nodes and ports are preassigned to tenants, the mapping of these resources is not further examined. Therefore, only virtual links need to be mapped. Virtual links correspond to paths through the network and need to be assigned a wavelength or a part of the spectrum. This is done by either RWA in the case of fixed grids or RSA for flexible DWDM grids. In most cases, wavelength conversion is not available because of the cost and limited blocking reduction [ZJM00]. For this reason, the wavelength or spectrum continuity constraint has to be taken into account. To reduce the complexity of this task, the process can be split into two stages: finding one or more valid routes and assigning a wavelength or a portion of the spectrum.
Natural options for the first step are fixed routing, fixed alternate routing and adaptive routing [Hua+12; ZJM00]. Fixed routing, e.g., shortest-path, leads to higher blocking probabilities because it only considers one path and ignores the current state of the network. Fixed alternate routing uses an ordered list with a predefined number of paths, e.g., link- or node-disjoint paths, and uses the first path that allows for a valid wavelength assignment. A way of obtaining the shortest k disjoint paths is described in [Bha97]. Two variations are explained that either result in node or link disjoint paths. Adaptive routing takes the state of the network into account, i.e., the existing connections and the occupied spectrum. This also means that more information needs to be exposed by the NMS or the control plane.
Zang et al. state that the routing is more important than the wavelength assignment for RWA [ZJM00]. None of their presented wavelength assignment heuristics, such as first-fit, random or more elaborate ones, leads to noticeable improvements. This means that using a simple heuristic — complexity wise — such as first-fit does not lead to high penalties regarding blocking probability. RSA examples for one-step and two-step approaches are shown in [Wan+11]. The authors of [Li+14] present a heuristic for VNE in optical networks based on k-shortest paths that tries to minimize the number of used wavelengths. By doing so, they show that the acceptance ratio improves and that a growing number of k is beneficial up to some point. This thesis only considers dynamic RSA, i.e., requests arrive at one point in time and may disappear after a time period. Additionally, the two-step methodology is applied: first a shortest path is calculated and then the spectrum is assigned.

3.2. Shortest Path Algorithms for Networks

Application areas for shortest distance or path calculations include road networks, web graphs, biological networks, computer networks, ranked keyword searches, databases and social networks. Sometimes, only the distance is relevant, e.g., to determine the closeness of a node in a social network. Other cases, such as navigation through road networks, require the whole path. The path is also relevant for computer networks, in order to route traffic.

A few basic definitions are needed for further discussion of the algorithms. Let $G = (V, E)$ be a graph with vertices V and edges E. The number of vertices is denoted as $n = |V|$ and the number of edges as $m = |E|$.
The default approach for conducting a shortest path calculation between two nodes is running Breadth-First Search (BFS) or Dijkstra's algorithm [Dij59] from one of them until the other node is reached. The main difference is that BFS calculates a shortest path based on the least hops, i.e., it is only applicable to graphs with uniform edge weight, whereas Dijkstra takes into account weights assigned to edges. This leads to a difference in the implementation and computational complexity. For BFS, it is sufficient to use a simple list for queuing nodes because the nodes are by definition inserted in the right order. Since Dijkstra processes the weights of edges, nodes in the queue may reduce their distance and move further to the front. A priority queue is needed to keep track of the correct ordering. The choice of the queue reflects in the computational complexity: while the BFS has a complexity of $\mathcal{O}(n + m)$, Dijkstra with a Fibonacci heap leads to $\mathcal{O}(n \log n + m)$. The advantages of both shortest path algorithms are a simple implementation and that no memory is needed to store additional information. The drawback is that every query requires a computation from scratch, even if it was calculated before. Depending on the number of queries, the repeated recalculation leads to a substantial computational overhead. In addition, the query time increases with a growing network size.
One naive way of reducing the query time is precomputing All-Pairs Shortest Paths (APSP). In this case, the shortest paths between all nodes are available and can be queried in constant time $\mathcal{O}(1)$ by performing a lookup. On the downside, precomputing APSP is computationally complex and results in high memory requirements for storing the distances. For small networks, precomputation is a viable option to improve the query processing time while the memory is still manageable. For larger networks, it leads to excessive requirements on the hardware side.
A graph representation that is related to shortest paths are Shortest-Path Trees (SPTs). They are a type of spanning tree, which forms a subgraph of G containing all vertices of G [WC14]. An SPT is rooted in a source and connects "all nodes such that the sum of the edge lengths from the source to each node is minimized" by selecting a set of edges from G [WC14]. An SPT can be computed using Dijkstra's algorithm and represents a solution to the Single-Source Shortest Paths (SSSP) problem. SPTs can be applied to different use cases, e.g., to improve the speed of a bidirectional BFS [HAK16].
There are two categories of shortest-path distance computations: exact methods and approximations. An approximation provides an upper bound on the exact distance and is applied in order to reduce complexity or save memory compared to exact methods. One group of algorithms approximates the distance based on landmarks. They try to estimate the distance between two nodes based on precomputed paths connected to the landmarks. One example for a landmark-based estimation for shortest-path distances has been presented in [Tre+11]. It uses SPTs rooted in landmarks as a starting point for the estimation and improves the result by taking into account common ancestors as well as shortcuts. An update procedure after insertion and deletion of edges is also explained. These procedures are needed to update the SPTs. In practical terms, approximations

are not a good approach for shortest path calculations in (optical) networks. Usually, those methods just provide a distance and this is not sufficient for a connection setup. Even if the path was reconstructed it might be much longer, especially for nodes which are close to each other. Using such a path would potentially waste resources. For this reason, the thesis focuses on exact methods.
Partial precomputation takes the middle ground between Dijkstra and APSP. It improves the query times while at the same time keeping the memory overhead reasonable. The result of the precomputation is any kind of helper data structure, which then improves the response time for queries. Of course, this computation needs some additional time beforehand. Two examples for this class are tree decomposition and labeling methods. The first uses a divide and conquer approach and decomposes the graph into a tree. It then precomputes shortest paths for parts of the tree. Queried distances are calculated based on these partial results. The second one creates labels, which contain distances to particular nodes. Based on these distances, the shortest path can always be computed because it is part of the labels by construction. Both approaches are discussed in more detail next.

3.2.1. Tree Decomposition

An approach for indexing graphs based on tree decomposition and answering shortest-path queries is presented in [Wei10]. The tree decomposition is used for creating the index, while bottom-up operations are applied to find the shortest paths. A tree decomposition describes the transformation of a graph into a tree representation. After the decomposition, the tree consists of nodes, which are called *bags* and contain vertices and edges from the original graph. There are three important properties that need to be fulfilled by this representation: (i) every vertex needs to be at least in one bag, (ii) every edge including both endpoints needs to be at least in one bag and (iii) bags that contain the same vertex need to form a connected subtree. Every vertex can be present in multiple bags as long as the last rule is not violated. Common vertices in neighboring bags along the tree represent transitions between these bags. APSP between the vertices of every bag is precomputed. Paths between vertices in different bags are calculated on demand by traversing the tree. This approach offers a parameter for balancing the trade-off between query time and index size. The initial formulation only supports static data and no weighted edges are mentioned [Wei10].
An extension of this algorithm is presented in [RLL12] as part of an idea for a shared graph library that can be used inside SDN controllers to provide all graph-related algorithms. The extension adds support for dynamicity in the network by allowing the insertion and removal of vertices and edges. This extension makes the algorithm applicable to SDN networks that are dynamic in nature. In general, the algorithms of the proposed library are expected to work with physical and virtual network graphs.
Another shortest path algorithm that applies tree decompositions and aims at SDN controllers as an application area is presented in [Xu+16]. It supports weighted edges but needs to run the full preprocessing after every update. The algorithm improves the time for preprocessing compared to previously available tree decompositon and labeling

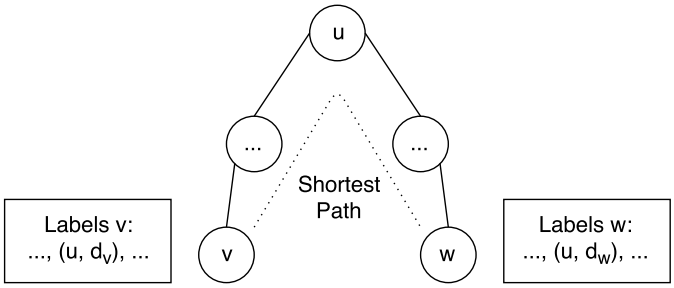


Figure 3.2.: The concept of 2-hop labels for a shortest path between v and w passing through u.

methods. Additionally, it covers batch processing of queries by exploiting tree overlaps in the calculation.

3.2.2. Labeling Methods

2-hop labels "are based on 2-hop covers of the shortest paths" [Coh+02; Coh+03]. They can be used to calculate the reachability, distance or paths in directed or undirected graphs yielding exact or approximate results. 2-hop labels define shortest paths through a label at both endpoints that defines a common intermediate hop. Fig. 3.2 is used as an example to explain the general idea. Let us assume that a shortest path between the nodes v and w needs to be calculated. First, the labels of both nodes are scanned for common hops. Further, we assume that both nodes contain a label for node u, which results in the lowest total distance $d_v + d_w$. This means that the shortest path between v and w is passing through u and has the calculated distance.
Hierarchical hub labeling is one approach that follows the 2-hop label definition [Abr+12]. Hubs are another name for known hops that belong to a node and are part of the label. The hierarchy relates to the relationship between nodes based on their availability as a hub of another node. This relationship creates a ranking of the nodes and leads to an order. The ordering is then provided as an input to the calculation of the labeling which only contains the highest-ranked node for each shortest path. An extension of this scheme is proposed in [Del+14]. The most important improvements comprise finding a better ordering efficiently and compressing the labeling.
Pruned Landmark Labeling (PLL) is an algorithm based on 2-hop labels that is introduced in [AIY13] together with a bit-parallel labeling method. PLL is an index-based approach for accelerating the query time for exact distance queries between arbitrary nodes in the graph. Like in the original definition of 2-hop labels, every node maintains a set of labels. Each label contains a hop and the respective distance. The construction of the labeling guarantees that the labels of two nodes contain a common hop along one shortest path [AIY13]. Therefore, by comparing the total distance of common hops, i.e., nodes that are part of the intersection of both label sets, the minimal distance along the shortest path can be found. One of the key characteristics contributing to the algorithm's scalability is the pruning during the BFS run for every node. The pruning

reduces the requirements for computation time and memory space compared to a full APSP. It stops exploring the graph at vertices that can be already reached via a shorter or equally long path that is available in the labeling. The presented bit-parallel labeling enables the calculation of multiple labels in parallel but it relies on bit-level parallelism This feature is not available in all programming languages, and is just applicable to unweighted graphs. In summary, the focus of PLL is on computing shortest distances in static undirected graphs. Akiba et al. provide hints on extensions, in order to support the shortest path as an output as well as weighted and directed graphs [AIY13]. Results show that PLL outperforms state-of-the-art methods based on distance labeling and tree decomposition, and it is applicable to very large data sets with millions of vertices and edges [AIY13].
An extension of PLL to road networks is presented in [Aki+14]. It defines so-called highways, which describe main routes going through a (road) network. Their classification relies on the length of an edge and its travel speed. Then, the labels are calculated taking into account the highways. This approach could be mapped to the length and capacity of links in computer networks but it is not further investigated in this thesis.
Akiba et al. propose two extensions to PLL [AIY14]. One allows adding new nodes and edges to the graph and the other one adds the capability of answering historical queries. Without the update extension it was necessary to rerun the preprocessing after every change, which takes considerably more time. The algorithm only adds new label entries and updates distances where needed. This avoids processing more vertices than necessary to provide correct responses. After an update, the size of the labeling is usually no longer minimal. This extension is also compatible with the previously described bit-parallel labeling. Historical queries answer shortest-path queries for the past and enable queries for points in time when changes occurred. The extension toward adding nodes and edges forms the basis of the algorithm that is presented in this chapter. Historical queries however are not relevant for computer networks.

3.2.3. k-Shortest Paths

In the context of optical networks, the k-shortest paths can be used to provide multiple potential paths to the spectrum assignment of the RSA, thereby reducing the blocking probability. Not all the presented approaches are applicable to the computation of the k-shortest paths. A naive implementation of Dijkstra's algorithm visits every vertex up to k times. This means it has a computational complexity of $\mathcal{O}((n \log n + m) * k)$. Even the improvements published by Eppstein only reduce it to $\mathcal{O}(n \log n + m + k)$ [Epp98]. Both algorithms are too slow to be applied to large graphs with many incoming queries. Therefore, approaches taking advantage of precomputation are a better option. Using tree decomposition, it is not clear how to tackle k-shortest paths.
The first indexing method calculating the top-k distances or k-shortest paths is an adaptation of PLL [Aki+15]. It is based on the idea of a 2-hop cover and only considers static graphs, like the initial definition of PLL. Due to the consideration of loops, one of the main challenges here is the correct count of paths. The algorithm not only needs to keep track of intermediate nodes but also of loops that are available at every node. The

Table 3.1.: Notation for the algorithms' description.

$G = (V, E)$	A graph with a set of vertices V and a set of edges E
v	A vertex $v \in V$
v_k	The root of the k-th Dijkstra run
d_{uv}	The distance between vertex u and v
w_{uv}	The weight of the edge between u and v
p_v	The parent/predecessor of vertex v along a tree or a path
l	A label represented by $(v, v_k, d_{v_k v}, p)$
$vertex(l)$	The labeled vertex v of a label l
$root(l)$	The root v_k of a label l
$dist(l)$	The distance $d_{v_k v}$ of a label l
$parent(l)$	The parent p of a label l
L	The labeling (or index), i.e., all labels for all vertices
L_k	The labeling after the k-th Dijkstra run
$L[v]$	All labels for a vertex v
$L(v)$	All root vertices for a vertex v
$L[v][v_k]$	The label of vertex v for the hop/root v_k
$N(v)$	The set of neighbors of vertex v
$order(V)$	A set of vertices V sorted by a defined ordering (e.g. degree)
PQ	Priority queue that sorts the entries by key, i.e., distance d

first step is to calculate up to k loop labels for every node. After this step, the pruned BFS calculates the labels for the top-k distances. To support weighted graphs, BFS needs to be replaced by Dijkstra's algorithm. The presented results indicate that the query time can be substantially improved with this technique, while keeping the index size reasonable, fitting into the memory of modern computers [Aki+15]. A hierarchical and dynamic approach for maintaining a k-all-path cover [AYM16], which contains all paths of length k, is not efficiently applicable to k-shortest paths. The main difference is that the former one refers to the exact length of paths and the latter one to the number of shortest paths.

3.3. Extensions to Pruned Landmark Labeling

The shortest path algorithm used as a starting point for this work is PLL. It represents a trade-off between the required precomputation — time and memory — and the response times for queries. The original definition does not support changes in the graph [AIY13]. It is only applicable to a static graph, or a full preprocessing is required in case an edge or vertex is added or removed. In order to avoid the expensive precomputation, an approach is needed to react to changes in the underlying graph by applying updates to the labeling. The update procedure should reestablish the correctness, while being less costly than a precomputation. An extension supporting the insertion of new edges

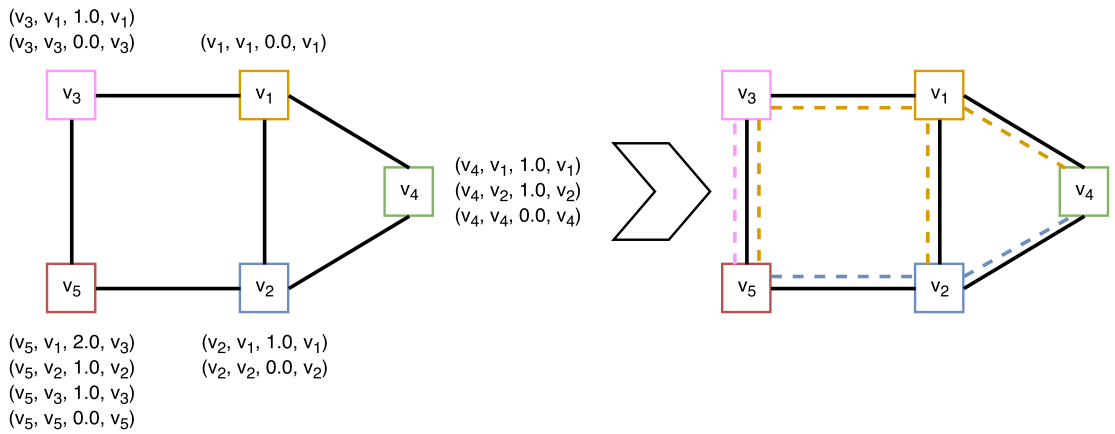


Figure 3.3.: Example of a labeling and its mapping to a colored SPT representation.

has been presented in [AIY14]. However, the algorithm neglects the cleanup of obsolete entries, i.e., the size is no longer minimal (with respect to the given ordering) and the number of labels increases with every update. These entries force the user to execute the preprocessing again, when the index size exceeds a certain threshold. Work on the removal of edges is not available currently.
For the upcoming description of the algorithms and the representation in figures a common terminology is required. The notation that will be used throughout this section is summarized in Tab. 3.1.
One way of representing SSSP rooted in a given source are SPTs. If they support edge updates in the form of increased or decreased weights and the insertion and deletion of edges, they are said to be fully dynamic. The structure of the labels calculated by PLL is similar to SPTs. In fact, the labels for a given root represent a pruned SPT that only contains entries relevant for the calculation of the shortest path. In this work, SPTs are used to visualize the labeling calculated by PLL and available algorithms for SPTs [FMN00] are adapted for maintaining fully dynamic networks. An example for the representation that is used throughout this chapter is shown in Fig. 3.3. On the left-hand side, a representation listing the labels that are available at each node is shown. If we want to calculate a shortest path between v_5 and v_4, both lists of labels are scanned for common hops, i.e., root vertices. v_1 and v_2 are available in the lists of both vertices. The first vertex results in a distance of 3.0 by summing both distances toward v_1, and the second one sums up to 2.0. Therefore, the shortest path between v_5 and v_4 passes through v_2 and has a length of 2.0. In order to make the drawing clearer and more concise, the pruned SPT representation on the right-hand side is used. Every vertex that contains a label from a given root vertex is connected to the colored root vertex by edges of the same color, i.e., every SPT corresponds to a color. For example, v_5 has a label for hop v_2 (blue node). For this reason, v_5 is connected to v_2 by a blue dashed line.
In this work, a way of supporting edge insertions and deletions for PLL is introduced that keeps the labeling correct and minimal. The main challenge after an insertion of a new

Algorithm 3.1: PrunedDijkstra(v_k)

```
   Data: Graph G, labeling L_{k-1}, root vertex v_k
   Result: labeling L_k
 1 PQ ← priority queue with one element (0,v_k);
   /* P is a tentative shortest path map that contains distances and
      parents (along v_k's SPT) for all v ∈ V.                        */
 2 P[v_k] ← (0, none) P[v] ← (∞, none) ∀v ∈ V \ {v_k};
 3 L_k[v] ← L_{k-1}[v] ∀v ∈ V;
 4 while PQ is not empty do
 5 |   u ← PQ.RemoveMin();
 6 |   (d_{v_k u}, p_u) ← P[u];
 7 |   if query(v_k, u, L_{k-1}) ≤ d_{v_k u} then continue;
 8 |   L_k[u] ← L_k[u] ∪ {(u, v_k, d_{v_k u}, p_u)};
 9 |   foreach w ∈ N(u) do
10 |   |   (d_{v_k w}, p_w) ← P[w];
11 |   |   d_update ← d_{v_k u} + w_uw;
12 |   |   if d_update < d_{v_k w} then
13 |   |   |   P[w] ← (d_update, u);
14 |   |   |   PQ.DecreaseOrInsert(d_update, w);
15 |   |   end
16 |   end
17 end
```

edge is the removal of entries that would have been pruned if the new shorter path existed from the start. In contrast, a deletion of an edge may only lead to an increase in distance for the shortest paths. Therefore, all affected nodes have to be identified and invalid entries have to be removed in order to compute correct results based on the labeling. In addition, pruned branches may need to be resumed because a shorter path that led to a pruning before is no longer available. This thesis assumes that (optical) networks are mapped to undirected graphs with edge weights because a bidirectional communication is required and the weight/length of edges has an influence on the chosen path. Further, the initial node degree is used for the ordering throughout this work. This means that nodes with a higher degree get a smaller index and are processed earlier. Next, the base algorithm for creating the labeling as well as the update procedures including helper functions are introduced.

3.3.1. Base Algorithm - Pruned Dijkstra

The base algorithm for the preprocessing that implements Dijkstra's algorithm with pruning, from now on called "pruned Dijkstra", is presented in Alg. 3.1. In a nutshell, Dijkstra's algorithm is run from every node and it is stopped, i.e., pruned, whenever an existing pair of labels provides an equal or shorter path. Pruning means that a

particular branch of the search tree is cut off because the new path does not provide any improvement. The implementation is an adaptation of the pruned BFS taking into account the hints for links with weights and for the retrieval of paths (instead of distances) from [AIY13; AIY14]. A partially applicable example of a pruned Dijkstra has been presented in [AY16]. It uses different data structures and does not store all the required information to reconstruct paths. Additionally, the pruned BFS's definition has been fixed because the presented pseudocode [AIY13; AIY14] contains mistakes: (i) the wrong index is used (line 10: $L'_{k-1} \rightarrow L'_k$), (ii) the wrong distance is accessed (line 10: $P[v_k] \rightarrow P[u]$) and (iii) an undefined node is used for searching neighbors (line 11: $N_G(v) \rightarrow N_G(u)$).
The definition of a label follows the suggestion of [AIY13] and includes the parent along the path, so that not just the shortest distance but also the shortest path can be queried. A label $L[v][v_k]$ is a quadruple of the form: $(v, v_k, d_{v_k v}, p_v)$, where v is the labeled vertex, v_k is the hop vertex and the root of the pruned SPT, d_{uv} is the distance between the labeled vertex and the root and p_v is the parent node along the path toward the root. The vertex v that this label is attached to is only included for convenience reasons. Given two nodes with a common label, the shortest path can be reconstructed by starting from both endpoints and following the parent entries toward the common root.
Alg. 3.1 summarizes the preprocessing for a single root. The algorithm needs to be repeated for all vertices $v \in V$ according to the ordering. In more detail, v_k is the root of the k-th pruned Dijkstra run and L_{k-1} comprises the labels that have been computed by previous runs. First, the priority queue `PQ`, which is sorted by distance representing the key, and the tentative shortest path map P are initialized in lines 1 to 2. The latter contains the tentative distances to the root and the respective parent vertex along the path to v_k for the current k. Additionally, the labels that have been computed so far are included in the result of this step because the labeling is built gradually. The while-loop processes all elements of the priority queue until it is empty. In line 5, the element with the minimum distance is retrieved from the priority queue. If the queried distance between v_k and u based on the labels L_{k-1} is lower than or equal to the stored value in P, further processing is omitted for this element. This branch of the if-statement corresponds to an existing path with equal or shorter distance. Else, we found a new shortest path and need to add a label to node u's list and process its neighbors. For every neighbor w, whose distance is reduced by the new path, we update the stored distance and add it to the queue or decrease its key, i.e, reduce the stored distance, in case it was already available in the queue. After the execution is finished, the updated labels are available in L_k. The proof of correctness was already provided by Akiba et al. [AIY13].

3.3.2. Adding Edges

An insertion of new edges has been presented in [AIY14]. Akiba et al. resumed their pruned BFS for all labels that are available on both sides of the new edge. The idea is that only labels that are present on either side of the new edge might use it for a shorter path. All other labels have been pruned before and never reached the new

Algorithm 3.2: edgeAdded(e)

```
   Data: Graph G with E ∪ {e}, labeling L, new edge e from a to b
   Result: Updated minimal labeling L for the new graph G
 1 R ← ∅ ;                                  /* vertices with reduced distance */
 2 foreach v_k ∈ order(L(a) ∪ L(b)) do
 3 |   if v_k ∈ L(a) then
 4 |   |   R ← R ∪ resumePrunedDijkstra((b, v_k, d_{v_k a} + w_ab, a));
 5 |   end
 6 |   if v_k ∈ L(b) then
 7 |   |   R ← R ∪ resumePrunedDijkstra((a, v_k, d_{v_k b} + w_ab, b));
 8 |   end
 9 end
10 foreach r ∈ R do pruneUpdatedTree(r);
11 foreach r ∈ R do removePrunedLabels(r);
```

edge. Therefore, we need to continue the labels located at the edge's endpoints on the opposite side and check if they result in new labels. By applying this procedure, the correct labeling is restored since PLL always chooses the pair of labels with the shortest distance. This approach has one major drawback: labels that are no longer used are still kept and lead to a growing size of the labeling. Therefore, the minimality of the original labeling is not maintained. This thesis presents extensions to the algorithm that make it applicable to Dijkstra and restore the minimal size of the labeling through a cleanup procedure. Even though it increases the running time, it leads to space savings and is a requirement for the proposed deletion of edges.

The handling of an inserted edge is presented in Alg. 3.2, in which the selection of labels and `resumePrunedDijkstra` are based on [AIY14]. It is assumed that the algorithm has access to the graph `G` that already contains the new edge and the existing labeling `L` that was correct and minimal before e was added. The new edge e is inserted between vertices a and b. First, the set `R` is initialized. At the end, it contains all nodes whose distance has been reduced throughout the runs of `resumePrunedDijkstra`. These nodes are relevant for the subsequent cleanup procedure. The pruned Dijkstra is resumed for all labels that are available on both sides of the new edge, i.e., $L(a)$ and $L(b)$. In line 4 new labels passing through the edge from a to b are resumed, while in line 7 the opposite direction is handled. The procedure `resumePrunedDijkstra` is called with the new label that potentially needs to be resumed. It contains the labeled node, the root v_k, the distance through the new edge and the parent on the opposite side. The result of this procedure is a set of vertices that decreased their distance toward the root. It is merged with the already stored vertices in set `R`. After finishing `resumePrunedDijkstra` for all available labels, the vertices in `R` need to undergo a two-step cleanup. First, `pruneUpdatedTree` is run to prune the tree rooted in r for every element of `R`. Second, all vertices are checked by `removePrunedLabels` for labels that are no longer needed. This refers to entries that would not have been added if the shorter path was available from the start. As a result

Procedure resumePrunedDijkstra(l)

Data: Graph G, labeling L, label $l = (v, v_k, d_{v_k v}, p_v)$
Result: Updated labeling L based on l, vertices with reduced distance R

```
1  PQ ← priority queue with one element enqueued by distance (d_{v_k v}, v, p);
2  R ← ∅;
3  while PQ is not empty do
4  |  (d_{v_k u}, u, p_u) ← PQ.RemoveMin();
5  |  if restrictedQuery(v_k, u, k) ≤ d_{v_k u} then continue;
6  |  L[u][v_k] ← (u, v_k, d_{v_k u}, p_u);
7  |  R ← R ∪ {u};
8  |  foreach w ∈ N(u) do
9  |  |  PQ.InsertByDistance(d_{v_k u} + w_{uw}, w, u);
10 |  end
11 end
12 return R;
```

of this algorithm, `L` is again a correct and minimal labeling for the graph with the new edge e.

Resume Pruned Dijkstra

The pruned Dijkstra is resumed for all labels that are available on both sides of the new edge e, i.e., at node a and b. The algorithm is based on the resumed BFS presented in [AIY14]. Differences include the use of Dijkstra, the definition of the labels and that the nodes with a reduced distance are collected. In general, the idea is that labels might have used the new edge if it was there from the start. Therefore, they are continued on the other side of the new edge and added wherever needed until the search is stopped, i.e., pruned, or the whole graph has been walked. Before going into detail, a helper method `restrictedQuery(a,b,k)` needs to be introduced. It queries the current labeling `L` for the shortest distance between `a` and `b` using only labels that belong to a node with a position in the ordering smaller than or equal to `k`.
The complete procedure is defined in `resumePrunedDijkstra`. The algorithm needs the graph `G` and the labeling `L` to process the label that is passed as an argument. This label contains the labeled vertex v, the root of the tree v_k, the distance $d_{v_k v}$ between v_k and v and the parent p_v. A subset of values from this label is used for the initial element of the priority queue `PQ`. They are enqueued according to the associated distance. `R` is initialized and collects all vertices, whose distance is reduced. Then, the while-loop is repeated until the queue is empty. The minimum element is removed from the queue and the distance $d_{v_k u}$, the labeled vertex u and its parent p_u are extracted. If there is a shorter or equal path resulting from a query that is restricted to vertices with a smaller index than k (line 5), we prune this label and skip further processing for it. Else, we found a shorter path and need to store the new label in $L[u][v_k]$. It is either added or

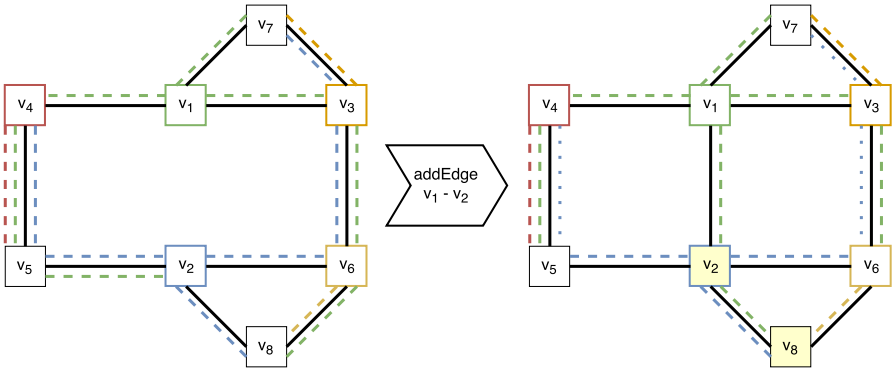


Figure 3.4.: Example for a labeling, in which the two highlighted nodes changed their distance and multiple labels belonging to v_2 need to be pruned (dotted lines).

replaces the existing label. This is safe because it provides a shorter path and only one label per root is maintained for the shortest path computation. Additionally, the node u is added to `R` because its distance has been reduced. Then, the neighbors w of vertex u are iterated and inserted into the queue. It is possible that multiple labels for the same node are available in the queue. Since the distance of a new path is compared to a query answered by the existing labels, a shorter path cannot be overwritten by a longer one. As soon as the queue is empty, the set R is returned.
After the execution of `resumePrunedDijkstra`, all labels for the root v_k passing through the new edge have been updated. If this is done for all labels on both sides of the edge, future queries will return the correct shortest path [AIY14]. However, the labeling might no longer be minimal and therefore, contains unnecessary labels. These obsolete labels existed before and are no longer needed for the computation of the shortest path. The labeling will be cleaned up in the next two steps. For this reason, all nodes with a reduced distance have to be collected and returned by `resumePrunedDijkstra`.

Prune Updated Tree

The first step of the cleanup is concerned with pruning trees, i.e., (pruned) SPTs, whose root node v_k has at least one new or updated label l_{new} that leads to a shorter path than the one available before the edge was added. This label l_{new} may introduce a path to a node v_{cut} of v_k's tree that is shorter than the initially available path, when the tree was calculated. As a result, this shorter path may lead to pruning at v_{cut} that needs to be applied now. The basic idea is to walk the tree and check if the labels of the tree need to be pruned because of a shorter path along a different tree. This only applies to shorter paths that are spanned by trees that occur earlier in the ordering.
An example for this case is shown in Fig. 3.4. On the left-hand side, we see the original graph and the labeling visualized by the dashed lines indicating the pruned SPTs. The ordering of the vertices corresponds to their index. Then, an edge between node v_1 and v_2 is inserted. This results in a recalculation of the labels by `resumePrunedDijkstra`. The highlighted vertices, v_2 and v_8, have reduced their distance toward v_1 (new green

Procedure pruneUpdatedTree(v_k)

```
   Data: Graph G, labeling L, root v_k
   Result: Labeling L with a correctly pruned tree for v_k
 1 Q ← queue with one element (v_k);
 2 while Q is not empty do
 3 |   u ← Q.Dequeue();
 4 |   if restrictedQuery(v_k, u, k − 1) ≤ dist(L[u][v_k]) then
 5 |   |   L[u] ← L[u] \ {L[u][v_k]}
 6 |   end
 7 |   foreach w ∈ N(u) do
 8 |   |   if u = parent(L[w][v_k]) then
 9 |   |   |   Q.Enqueue(w);
10 |   |   end
11 |   end
12 end
```

dashed lines). Therefore, both vertices need to be processed by the cleanup procedure. In the first step, `pruneUpdatedTree` removes three labels which are represented by the dotted lines. In particular, these entries are located at node v_3, v_4 and v_7 and belong to the tree rooted in v_2. The reason for this is that the new v_1 label at v_2 provides shorter or equal distances to the affected nodes and it is processed before v_2.

The procedure walking the tree from the root vertex and pruning labels is called `pruneUpdatedTree`. It needs access to the graph `G` and the currently non-minimal labeling `L`. A root node v_k is handed over as a parameter to the procedure and it is the starting point for the algorithm. A queue (first in, first out) is initialized with this root node. At this point, it is not mandatory to use a priority queue since we are walking an existing tree and the order, in which the nodes are traversed, is not important as long as all nodes are visited at least once. For every node in the queue the distance is queried by exclusively using labels that are smaller than k. The retrieved distance is compared to the one stored in label $L[u][v_k]$ (line 4). If the query returns a smaller or equal value, the label needs to be removed because it would have been pruned in the original search. Remember that a label for root node v_k is only added to u if it provides a shorter path than any available label of the labeling L_{k-1}. Finally, all children need to be added to the queue, i.e., neighbors w that have a label $L[w][v_k]$ with parent u. At the end of this procedure, all unnecessary labels that are part of the tree rooted in v_k have been pruned. From the definition, it can be seen that labels are pruned (pruneUpdatedTree line 4) if and only if they had been skipped by the original pruned Dikjkstra (Alg. 3.1 line 7). This approach automatically prunes the whole subtree because a new label that causes a pruning at any node of the tree must have been propagated along the subtree by `resumePrunedDijkstra`, since it provides a shorter path toward the root. This finalizes the first step of the cleanup process.

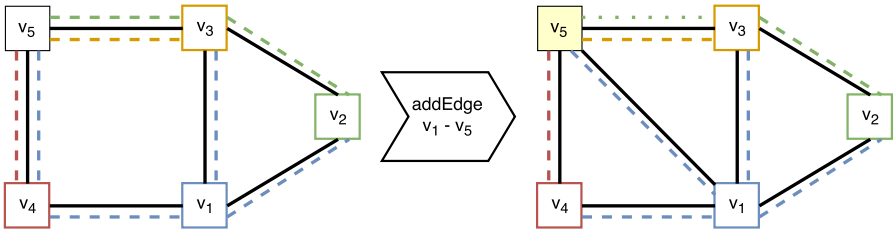


Figure 3.5.: Example for a labeling, where the label for v_2 at vertex v_5 needs to be pruned after the insertion of the new edge (dotted line).

Procedure removePrunedLabels(v)

Data: Graph G, labeling L, vertex v
Result: Labeling L without pruned entries at v

```
1 foreach l ∈ L[v] do
2 |   v_k ← root(l);
3 |   if restrictedQuery(v_k, v, k − 1) ≤ dist(l) then
4 |   |   L[v] ← L[v] \ {l};
5 |   end
6 end
```

Remove Pruned Labels

The second step of the cleanup inspects vertices that changed their distance for labels that are no longer needed because a shorter path was introduced by a new or updated label. This can only be the case if the new label belongs to an SPT spanned by a vertex that was processed earlier according to the ordering. The cleanup is carried out for all vertices whose distance has been reduced. It removes obsolete labels belonging to trees of root vertices that did not change any label. Therefore, the corresponding tree was not traversed in the first step. These labels must be removed from the labeling in order to achieve a minimal size again.

An example for this case is given in Fig. 3.5. On the left-hand side, we see the original graph with a vertex ordering given by the vertices' subscript and the labeling visualized by the dashed lines showing the pruned SPTs. An edge between v_1 and v_5 is inserted into the graph. After `resumePrunedDijkstra` is finished, only the highlighted node v_5 has reduced its distance (right-hand side of Fig. 3.5). Since it does not span a tree, the first step of the cleanup does not lead to any changes. Now, we can see that there is a label for v_2 that is no longer needed because the node v_1 with a lower index in the ordering provides an equidistant path. This entry for v_2 at v_5 can be pruned (dotted line) and is removed from the labeling by `removePrunedLabels`.

The second step of the cleanup is covered by `removePrunedLabels`. It is applied to all nodes whose distances changed and iterates all available labels at these nodes (line 1). Then, the distance provided by the labels with an index that is smaller than the label's root v_k is queried. In case a smaller or equal path is available, the label is removed from

the labeling. It follows the same argumentation as for the first step. Only labels that are shorter than the previously existing ones are part of the labeling. After this step is finished for all nodes, the labeling is minimal again with regard to the given ordering. It corresponds to the labeling that would be calculated by the preprocessing for the same graph with the same ordering. The overhead introduced by the cleanup procedure is evaluated experimentally in the next chapter.

Proof of Correctness

We assume that the original labeling that is available before Alg. 3.2 is applied is correct and minimal. The goal is to show that after the algorithm finishes, the labeling is still correct and minimal. In particular, the focus is on the newly introduced cleanup procedure. The following lemma provided by [AIY14] is reused for the proof:

Lemma 1. *If the distance between two nodes decreased, then the new shortest path between them passes through the new edge.*

In order to prove the correctness of the cleanup approach, it is necessary to show that there is no label left that is not needed for the calculation of the correct shortest path. For the sake of clarity, the labels are reduced to tuples, comprising the root vertex and the respective distance, in this proof. First, we show that only one of the two labels that lead to a new shortest path is new.

Lemma 2. *Only exactly one of the labels that lead to a new shortest path between two vertices and result in an existing label being removed has been added to one of the endpoints of the shortest path.*

Proof. We assume that there exists a label $(v_k, d_{kl}) \in L[v_l]$ $(k < l)$ that needs to be pruned because of a new shortest path between v_k and v_l going through v_j $(j \neq k)$ based on the labels $(v_j, d_{jk}) \in L[v_k]$ and $(v_j, d_{jl}) \in L[v_l]$. If none of the labels was added, then the original labeling was not minimal, which contradicts the initial assumption. If both labels for v_j were added due to the edge insertion, then a new shorter path exists between v_j and v_k as well as v_j and v_l. According to Lemma 1, this means that both new shortest paths pass through the added edge. For this reason, one of the root nodes v_i with $j < i \leq k$ would have labeled the path without passing through the new edge twice before those labels were added. Therefore, the new labels cannot be the reason for the pruning, since another shorter path, without the detour through the new edge, existed before. This contradicts the assumption that the labels (v_j, d_{jk}) and (v_j, d_{jl}) result in a pruning of label (v_k, d_{kl}). □

From Lemma 2 it follows that a new shortest path that leads to a pruning of a label can only be introduced by a new label at one of its endpoints. This finding is used to prove the minimality of the resulting labeling by contradiction.

Theorem 1. *After running* `pruneUpdatedTree` *and* `removePrunedLabels`*, there is no label left that needs to be deleted.*

Proof. Let us assume that there exists a label $(v_k, d_{kl}) \in L[v_l]$ $(k < l)$ that needs to be removed because it should have been pruned. This means that there exists a hop v_j that provides a shorter path with the labels $(v_j, d_{jk}) \in L[v_k]$ and $(v_j, d_{jl}) \in L[v_l]$. Since the label (v_k, d_{kl}) needs to be pruned because of the label (v_j, d_{jl}), it follows that $j < k$ and $d_{kl} \geq d_{jk} + d_{jl}$. Otherwise, the assumption is contradicted that the label needs to be pruned. Using Lemma 2, we know that either (v_j, d_{jk}) or (v_j, d_{jl}) is new or updated. If (v_j, d_{jk}) is new, then the new shortest path passes through v_j. The old path cannot have passed through v_j because $j < k$. For this reason, the label (v_k, d_{kl}) cannot have existed without violating the minimality. Therefore, the distance between v_k and v_j changed and v_k received a new label (v_j, d_{jk}). As a result, `pruneUpdatedTree` would have walked the tree and deleted the label (v_k, d_{kl}). This contradicts the assumption about the existence of (v_k, d_{kl}). If (v_j, d_{jl}) is new, then (v_k, d_{kl}) would have been pruned by `removePrunedLabels` because by adding a new label, the distance of v_l to v_j changed. This is again contradicting the assumption that the label exists. Following from those contradictions, there cannot be any label that needs to be removed after running the two steps of the cleanup. □

Since the cleanup only removes labels that are not needed, the correctness of `resumePrunedDijkstra` is maintained. Together with Theorem 1 the conclusion is that the labeling is correct and minimal at the end of Alg. 3.2.

3.3.3. Removing Edges

Support for a dynamic removal of edges from the graph in PLL has not been presented yet. Akiba et al. mention that in many cases, a removal of edges is not needed and the complexity outweighs its benefits [AIY14]. However, in computer networks, a failure of a link or port can lead to edges that need to be removed, at least until the problem is solved. Work about SPTs proposes solutions for the (addition and) removal of edges [CY09; FMN00]. These methods are also partially applicable to PLL. The labels created by PLL correspond to a partial (or pruned) SPT. The main difference is that PLL prunes labels (SPTs) if a path with an equal or shorter distance is already available. Two phases are required after an edge is removed from the graph in order to achieve a correct and minimal labeling. First, labels that traversed this edge need to be removed or reattached to another path with the same distance. Second, existing labels need to be continued, if the reason for pruning, i.e., another shorter path, is no longer given. This work introduces an algorithm that is able to handle edge removals for PLL while keeping the labeling correct and minimal.
The processing that is required after an edge is removed is presented in Alg. 3.3. Access to the graph `G` without the removed edge and the labeling `L` is needed for the removal process. The edge e between a and b that is no longer part of the graph is handed over as a parameter. A set `C` storing vertices whose distance changed is initialized first. In case of an edge removal, we are only interested in the labels that are available on both sides of the edge and therefore, their SPT potentially traverses the removed edge. All labels that have been pruned before and are not available on both sides are not

Algorithm 3.3: edgeRemoved(e)

```
   Data: Graph G with E \ e, labeling L, removed edge e from a to b
   Result: Updated minimal labeling L for the new graph G
 1 C ← ∅ ;                          /* vertices with changed distance */
   /* Phase I: remove or reattach labels                              */
 2 foreach v_k ∈ L(a) ∩ L(b) do
 3     if Tree of v_k passing through e from a to b then
 4         C ← C ∪ removeOrReattachSubtree(L[b][v_k ]);
 5     else if Tree of v_k passing through e from b to a then
 6         C ← C ∪ removeOrReattachSubtree(L[a][v_k ]);
 7     end
       /* else v_k not passing through e → no processing              */
 8 end
   /* Phase II: resume previously pruned labels                       */
 9 ResumeLabels ← findResumeLabels(C);
10 foreach w ∈ order(C ∪ ResumeLabels.keys) do
11     if w ∈ C then
12         extendUpdatedTree(w);
13     else
14         foreach l ∈ ResumeLabels[w] do
15             resumePrunedDijkstra(l);
16         end
17     end
18 end
```

affected by the removal. If the tree is passing from a to b through e then its subtree needs to be removed (or reattached) starting from the label $L[b][v_k]$. If it passes in the opposite direction, i.e., from b to a, we start with label $L[a][v_k]$. Otherwise, a label is available on both sides but the tree itself does not traverse e and therefore remains unaffected by the removal. In the first two cases `removeOrReattachSubtree` is called for the respective label. The changed vertices are aggregated until all labels have been processed and the loop finishes. Now, `findResumeLabels` is called for the vertices in `C`. The procedure searches for all labels that potentially need to be resumed and returns them grouped by their root vertex. "Resumed" means that for these labels, Dijkstra's algorithm needs to be continued at the spot at which it has been pruned before. All the affected (root) vertices need to be handled to reestablish a correct and minimal labeling. First, an ordered union of the vertices whose distance changed and the keys for the labels that need to be resumed is created. The keys correspond to the root vertices that the `ResumeLabels` are grouped by. Then, for each vertex either the whole tree is checked for pruned branches that need to be resumed or a set of selected labels is inspected and resumed if needed. If the vertex is part of `C`, then the whole tree needs to be scanned by `extendUpdatedTree`. Individual labels stored in `ResumeLabels` are omitted, since

Procedure removeOrReattachSubtree(l)

Data: Graph G, labeling L, label l to start from
Result: Labeling L with handled subtree of l, redVertices with removed labels

```
1  redVertices ← ∅;                                   /* all vertices white */
2  PQ ← priority queue with one element enqueued by distance (l);
3  while PQ is not empty do
4  |  l_min ← PQ.RemoveMin();
5  |  (u, v_k, d_{v_k u}, p) ← l_min ;
6  |  L[u] ← L[u] \ {l_min};
7  |  if dist(L[w][v_k]) + w_wu = d_{v_k u} s.t. w ∈ N(u) ∧ w ∉ redVertices then
8  |  |  L[u] ← L[u] ∪ {(u, v_k, d_{v_k u}, w)};                  /* pink vertex */
9  |  else
10 |  |  redVertices ← redVertices ∪ {u};                        /* red vertex */
11 |  |  foreach w ∈ N(u) s.t. parent(L[w][v_k]) = u do
12 |  |  |  PQ.InsertByDistance (L[w][v_k]);
13 |  |  end
14 |  end
15 end
16 return redVertices;
```

they are implicitly covered by scanning the whole tree. In the other case, only the labels that are stored in `ResumeLabels` are evaluated using `resumePrunedDijkstra`. After the algorithm finishes, the labeling `L` is correct and minimal for the graph `G` without edge e. It corresponds to the same labeling that a preprocessing applied to the new graph with the same ordering would calculate.

Remove or Reattach Subtree

The `removeOrReattachSubtree` procedure adapts an approach for SPTs, shown in [FMN00]. It classifies vertices into three categories: (i) not affected by the removal, (ii) the parent changed but the distance is the same, and (iii) the parent and the distance changed. Following the classification of vertices by color presented in [FMN00], these types are assigned the colors white (i), pink (ii) and red (iii), respectively. In the procedure description, the red vertices are collected in the set `redVertices`. In line 2, a priority queue is initialized with the label l that represents the root of the subtree. The while-loop is repeated until the queue is empty. The minimum label is extracted to l_{min} and removed from the labeling because the existing label used the deleted edge and is no longer valid. Next, we have to classify the vertex. If a neighbor exists that provides the same distance and is not itself a red vertex, then we found a pink vertex (line 7). This vertex can be simply reattached to this particular neighbor, and all its children are unaffected, i.e., white. Pruning does not need to be considered in this case because the distance did not change and we assume that the initial labeling was minimal. Else, the

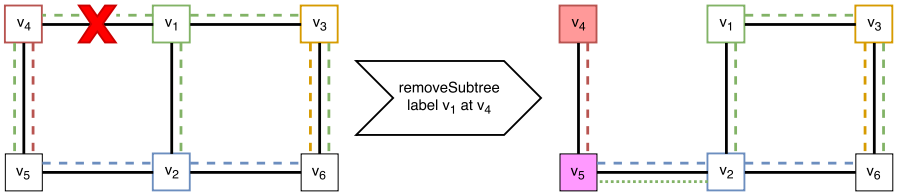


Figure 3.6.: Example for a subtree removal, in which v_5 (pink vertex) and v_4 (red vertex) are affected by the removed edge.

vertex cannot be reattached to the currently processed SPT and lost a label. Therefore, it changed the parent and the distance and represents a red vertex, which needs to be handled later on. Since the distance changed, all children of this vertex need to be considered. They are traversed (line 11) and added to the priority queue by distance. Finally, after the whole subtree has been checked, the `redVertices`, i.e., vertices who changed their distance, are returned. At the end, the labeling does not contain any invalid entries that belong to the subtree spanned by l. After this procedure is finished for all labels, the labeling does not contain any labels traversing the removed edge, but it is still incorrect because of missing labels that need to be added by phase II.
An example for `removeOrReattachSubtree` is given in Fig. 3.6. On the left-hand side, the graph is shown in its initial state with the index of the nodes representing their ordering. The edge between v_1 and v_4 is removed and `removeOrReattachSubtree` is run for the label of root vertex v_1 at node v_4. As a result of this process, v_4 is colored red because it cannot be reattached and lost its parent, and the distance changed. On the other hand, v_5 is colored pink because we are able to reattach it to the SPT of v_1 via v_2 while keeping the distance (dotted line on the right-hand side). This example only shows an intermediate result on the way to restoring a correct labeling.

Find Labels to be Resumed

In addition to vertices that changed their distance, we also need to consider labels that have been pruned before and need to be continued now. This applies to labels that were pruned because of a shorter path that no longer exists because it used the removed edge. These labels are exclusively located at neighbors of vertices whose distance changed, i.e., increased. For this purpose, we need to check if any neighboring labels need to be (potentially) resumed and collect them.
This is done by the procedure `findResumeLabels`. It takes a graph `G`, a labeling `L` and a set of nodes whose distance changed. The set `R` of labels that need to be resumed is initialized first by assigning the empty set to every root vertex. Then we iterate all labels l_w of all neighbors w of the nodes v_r that have been provided as input. To make it clearer, all nodes whose distance changed are processed one after another. We need to look at their neighbors w and the labels l_w that are available at w. From these labels, we extract the root node v_k. If the distance from v_k to v_r using the label l_w including the edge weight w_{wv_r} is lower than the one that can be queried, then this label potentially needs to be resumed. The query considers only nodes with an index up to k. In other

Procedure findResumeLabels(C)

Data: Graph G, labeling L, label l, vertices with changed distance C
Result: Set R of labels that need to be resumed grouped by root vertex

```
1  R[v] ← ∅ ∀v ∈ V;
2  foreach v_r ∈ C do
3  |  foreach w ∈ N(v_r) do
4  |  |  foreach l_w ∈ L[w] do
5  |  |  |  v_k ← root(l_w);
6  |  |  |  if dist(l_w) + w_{wv_r} < restrictedQuery(v_k, v_r, k) then
7  |  |  |  |  R[v_k] ← (v_r, v_k, dist(l_w) + w_{wv_r}, w);
8  |  |  |  end
9  |  |  end
10 |  end
11 end
12 return R;
```

words, we check if the label was pruned because of a shortest path that is no longer available in the labeling, i.e., removed as part of phase I. These labels that potentially need to be resumed are collected in R and returned after processing all input nodes. At this point, we are not able to tell which of the labels will be resumed in the end. Due to the ordering that is applied later on, other resumed trees or labels may lead to them being pruned.

Extend Updated Tree

Trees rooted in vertices whose distance increased need to be checked for branches that have been pruned before but need to be resumed now. This happens if a label that was previously available at the root provided a shorter path to a node that is part of the tree and caused the Dijkstra to prune the tree at this node before. This constellation is fixed by `extendUpdatedTree`, which walks the tree from a given root v_k and checks all pruned ends if they need to be continued. It is similar to the pruned Dijkstra (Alg. 3.1) but it exploits information about the existing tree, e.g., children that are already available do not need to be recalculated. The procedure has access to the graph G and the labeling L and takes a root vertex v_k as a parameter. The own label of the root vertex is enqueued as the first element of the priority queue, which is looped until no further element is available. In each iteration of the loop, the minimal label l is extracted from the queue. If there is no label available so far, l is added to the labeling (line 5). In case a label already exists, we need to verify that it is the same as the stored one, because there could be multiple labels for the same vertex in the queue (line 7). Only the matching labels need to be further processed to avoid redundant calculations. The neighbors of these nodes are iterated next. If the neighbor is a child of u along the SPT of v_k, it is added to the queue, so that we keep walking the tree. Else, a new label is only created

Procedure extendUpdatedTree(r)

```
   Data: Graph G, labeling L, root vertex v_k
   Result: Labeling L with correct labels for the SPT of v_k
 1 PQ ← priority queue with one element enqueued by distance (L [v_k][v_k]);
 2 while PQ is not empty do
 3 |   l ← PQ.RemoveMin();
 4 |   (u, v_k, d_{v_k u}, p_u) ← l;
 5 |   if L[u][v_k] = empty then
 6 |   |   L[u] ← L[u] ∪ {l};
 7 |   else if L[u][v_k] ≠ l then continue;
 8 |   foreach w ∈ N(u) do
 9 |   |   if parent(L[w][v_k]) = u then
10 |   |   |   PQ.InsertByDistance(L[w][v_k]);
11 |   |   else if d_{v_k u} + w_{uw} < restrictedQuery(v_k, w, k) then
12 |   |   |   PQ.InsertByDistance((w, v_k, d_{v_k u} + w_{uw}, u));
13 |   |   end
14 |   end
15 end
```

and inserted into the priority queue if a shorter distance than the one that is available through roots with an index up to k is found. This corresponds to previously pruned labels that are hereby resumed. After the whole tree has been walked and all new labels (if any) have been pruned, the correct labeling for v_k is restored in L.

Proof of Correctness

Before the edge is removed, the labeling is assumed to be correct and minimal. After phase I is finished, all invalid labels are either reattached or removed. This means that there are no labels that use the deleted edge, even though there might exist labels that need to be added by phase II in order to achieve a correct labeling again. After all invalid subtrees have been removed, phase II processes all labels that are potentially affected by the distance changes and need to be resumed.

Lemma 3. *If the distance between two nodes increased, then the old shortest path between them passed through the removed edge.*

In other words, Lemma 3 summarizes the obvious fact that an increased distance can only be triggered by a shortest path that used the removed edge. It is used to show that only one of the labels defining a shortest path that is no longer valid results in the insertion of a new label. To improve the readability, the labels have been reduced to the root vertex because the remaining three components are irrelevant for the proof.

Lemma 4. *A removed label for a root vertex that leads to new shortest path between two vertices and results in a new label being added can only have been removed from exactly one of the endpoints.*

Proof. We assume that there exists a label $(v_k) \in L[v_l]$ $(k < l)$ that needs to be added because the old shortest path between v_k and v_l going through v_j $(j \neq k)$ based on the labels $(v_j) \in L[v_k]$ and $(v_j) \in L[v_l]$ no longer exists. If none of the labels was removed, then the assumption that the path no longer exists is violated and leads to a contradiction. If both labels were removed, Lemma 3 states that the shortest paths between v_l and v_k as well as between v_l and v_j passed through the removed edge. Therefore, one of the root vertices v_i, so that $j < i \leq k$, would have labeled the path between v_k and v_l without passing through the removed edge twice. So either there is a label for v_i at either of the nodes that has been removed or the label (v_k) existed before. Both contradict the assumption that the labels (v_j) and (v_j) are the reason for (v_k) not being available. □

Based on the result of Lemma 4, we will now prove that after applying phase II of the insertion, no label is missing.

Theorem 2. *After finishing phase II of Alg. 3.3, there is no label missing that needs to be added.*

Proof. We assume that there exists a label that is missing for root node v_k at node v_l $(k < l)$ and it provides a shorter path between both nodes. Since the label is missing, it must have been pruned somewhere along the path from v_k to v_l. Without loss of generality, we assume that v_i is the last node that contains the label along this path and v_{i+1} is its successor without the label. Two cases need to be considered now: (i) v_{i+1} changed its distance or (ii) v_{i+1} did not change its distance. In case (i), v_{i+1} changed its distance, the vertex is processed and its neighbors' labels are extended by`resumePrunedDijkstra`. This includes v_k because it provides a shorter path according to the assumption. Therefore, the procedure also reaches v_l and the missing label leads to a contradiction. In case (ii), the distance of v_{i+1} did not change. This means that the tree of v_k was cut at v_{i+1} because a label for a root vertex v_j $(j < k)$ provided a shorter path between v_k and v_{i+1}. Following the assumption that v_k is missing at v_l, the labels for v_j no longer provide the shortest path and, according to Lemma 4, must have been removed at v_k or v_{i+1}. If the label was removed at v_{i+1} then case (i) is applicable. If it was removed at v_k,`extendUpdatedTree` would be applied and the new label for v_k would be added to v_l, which contradicts the assumption that the missing label for v_k at v_l exists. □

Theorem 2 shows that all required labels are added by phase II and therefore, the labeling is correct. Since we removed all invalid entries in phase I and the methods used to add missing labels are applying the same approach as Alg. 3.1, the result of Alg. 3.3 is a correct and minimal labeling.

3.4. Automatic Creation of Virtual Optical Networks

In this section, a workflow for the automatic creation of VONs is defined. It demonstrates how the results of the thesis can be combined into a software component that creates

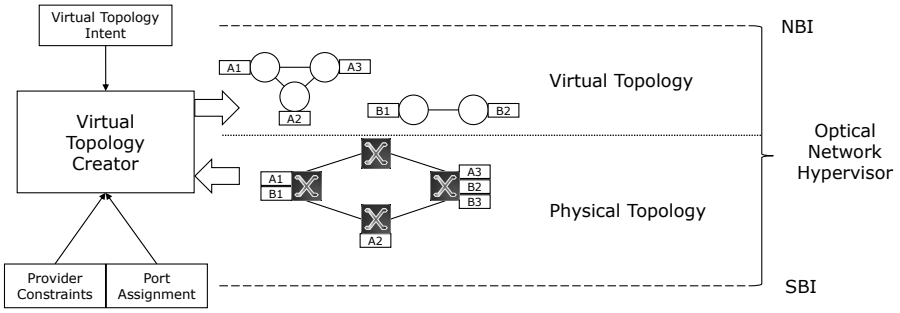


Figure 3.7.: Overview of virtual topology creation.

Algorithm 3.4: Virtual topology creation based on a set of intents.

Data: Set of Intents I
Result: Virtual links satisfying I

```
1 virtualLinks ← ∅ ;
2 foreach intent i ∈ I do
3 |   constraints ← Preprocessing(i);
4 |   paths ← ShortestPaths(constraints);
5 |   virtualLinks ← virtualLinks ∪ AssignSpectrum(constraints, paths);
6 end
7 return CreateTopology(virtualLinks);
```

virtual topologies up to the point when it is ready to be exposed to the client. An overview and the larger context are given in Fig. 3.7. The virtual topology creator is the software component that builds a virtual topology form intents and the physical topology. Additional input are provider constraints and the port assignment. Both are not further discussed in this thesis. The virtual topology intents are submitted by the client and need to be processed by the creator. The expected output is a virtual topology that satisfies the intents. This is a multi-step process that is explained next.

All steps are summarized in Alg. 3.4. The processing starts from a set of intents that are submitted by the user specifying his high-level requirements for the topology. These intents need to be taken into consideration by the algorithm. They are preprocessed and transformed into requirements that can be evaluated by the algorithm. This step mostly consists of extracting `constraints` for the creation of the topology. In the next step, resources need to be assigned to every intent, i.e., lightpaths through the network. The nodes and ports are assigned by the network operator beforehand. For the subsequent RSA, a two-step approach is chosen. The first step calculates one or more shortest `paths` and the second step assigns an appropriate part of the spectrum. The result of the assignment is collected in the set `virtualLinks`. Finally, the virtual links and their corresponding lightpaths through the network are merged into a (virtual) topology.

3.4.1. Preprocessing the Intents

The intents that are submitted to this algorithm are high-level requirements, defining what is expected but not how it is achieved. They hide implementational details from the client, while giving more freedom to the network provider with regard to the exact realization. An interface definition for the intents, which describes the input, has been shown in Sec. 2.3.1. One request may consist of a set of intents, which bundle endpoints and connection requirements between them. These requirements need to be translated into a representation, i.e., constraints, that the algorithm can use to create the correct number of virtual links with the expected properties. The preprocessing should exploit optimization options for every intent, e.g., reuse links and spectrum for multiple virtual links if only one of them is allowed to be active at any given time. Further synergies between intents of a request are out of the scope of this thesis. The output of this stage are constraints that define the involved endpoints for the virtual links as well as any interdependencies between them that need to be considered for the assignment.

3.4.2. Assignment of Network Resources

After the constraints have been extracted from the intents, the algorithm can start processing them. For each intent group, a number of virtual links needs to be created in order to satisfy the constraints. The available endpoints are by definition fixed because they are preassigned to clients. The remaining task is to create virtual links, which correspond to a path through the network and need a spectrum assignment. This corresponds to an RSA in optical networks. A two-step approach is commonly chosen in order to reduce the complexity. In the first step, a number of paths, e.g., shortest or disjoint paths, is calculated. The second step tries to accommodate the required spectrum on one of those paths, usually starting with the shortest one to save resources. The two steps are explained in more detail next.

Calculation of Shortest Paths

Before the spectrum can be assigned in the two-step approach, a number of viable paths needs to be calculated. A new computation for every request is not an efficient approach because it increases the response time. Precalculating APSP is expensive in two ways: the time it takes to compute all paths and the memory needed to store them. For small networks, this approach might be applicable, but it does not scale for large networks. Partial precomputation covers some middle ground between both. It precalculates helper information that requires an acceptable amount of time and memory. With this information, requests can be handled faster. This approach represents a good compromise because intents require multiple queries for shortest paths and several clients need to be served in parallel. Since the network topology does not change very often in optical networks, the precomputation amortizes across a number of requests. Nevertheless, the following changes of the physical network topology should be covered: 1. edge insertion (optionally: weight decrease), 2. edge deletion (optionally: weight increase), 3. vertex insertion and 4. vertex deletion. The edge insertion and deletion are

important features for optical networks to cope with failures and to offer flexibility for new deployments. Weight changes are usually not very prominent in this field because they are closely related to the link length. Vertex insertion and deletion are not very important scenarios neither because optical networks are planned for long periods with only minor changes in between.
This thesis presents extensions to PLL that cover a subset of these operations. Edge insertions and deletions are explained earlier in this chapter. Weight manipulations are not explicitly defined. They could be achieved by increasing/decreasing the weight and applying similar cleanup procedures to deleting/adding an edge. For now, a deletion of the edge with the old weight and an insertion with the new weight represent a workaround. Vertex insertion or removal is not available as a single operation. However, it can be split into multiple edge changes. An insertion of a vertex can be represented by adding an unconnected node to the graph, only with its own label, and then inserting the attached edges. A deletion works in the opposite direction. First, all attached links are removed from the graph and then the unconnected vertex can be simply deleted.
The presented algorithm provides one shortest path. Multiple paths are assumed to reduce the blocking probability for requests. Ramamurthy and Mukherjee conclude that an alternate-route is more beneficial than wavelength conversion if the number of routes is less than the edge connectivity [RM02]. Additionally, fixed-alternate routing converges toward adaptive-shortest-cost routing for a growing number of alternate routes. Therefore, further work in the direction of an adaptation for k-shortest paths might yield better results. It has been shown that PLL is able to compute k-shortest paths [Aki+15]. This is achieved by visiting the same node multiple times. Here, the main task would be to remove loops and adapt the cleanup procedures to k paths.

Spectrum Assignment

The spectrum assignment itself is less critical than the selection of the route. Even simple assignment methods like first-fit do not result in much worse assignments. The spectrum-assignment process iterates all calculated paths and finishes as soon as the first valid assignment is found. If none of the provided paths has enough free spectrum, the assignment fails and the intent is rejected. The required spectrum can be calculated based on a simple lookup table that provides values for the path length and the requested bit rate. Limitations of the hardware may apply and need to be considered by this step.

3.4.3. Output of the Virtual Optical Network

The final step merges all calculated virtual links into a single virtual topology. This happens for individual intents as well as for the complete set of virtual links. Optimizations may be applicable but are not considered here. The VON is then maintained inside of the hypervisor and exposed via an appropriate NBI (see Chapter 2). The client is then able to control his VON with an SDN controller. Any connection setups across virtual links are translated into the calculated paths in the underlying physical network. The virtualization layer also needs to verify that the restrictions of the requested topology

are met, e.g., maximum number of parallel connections. In case the topology needs to be adjusted, the creation process can be rerun with new or updated intents.

Summary

This chapter identifies the computation of virtual links as the most relevant task for creating VONs. In order to assign the links, a (shortest) path needs to be calculated and spectrum needs to be assigned. A shortest path algorithm based on precomputation is introduced and newly developed extensions to support the addition and removal of edges are defined. Finally, a complete workflow for the creation of VONs is presented. The next chapter shows how the algorithm can be included in the OVC and evaluates its performance.

4. Optical Virtualization Controller

The introduction of two concepts — SDN and NV — affects packet-switched networks as well as optical networks that rely on circuit switching. The first concept refers to network elements that can be programmed by SDN controllers and the second facilitates the sharing of resources between network tenants. A hypervisor is an abstraction unit that virtualizes the existing hardware. In contrast to OF hypervisors for switches, optical network hypervisors are uncommon and mostly of theoretical nature. With the introduction of open protocols, created from a common subset of the individual capabilities, a unified control and virtualization becomes more likely. The ultimate goal is multilayer network operation. As long as these control protocols are neither in place nor exposed by the hardware, the virtualization layer needs to be able to communicate southbound with the hardware through proprietary device interfaces. It must also expose open or experimental protocols toward northbound clients. At the same time, this approach presents a migration strategy that can be used to enable virtualization and SDN control for legacy equipment. With open protocol descriptions becoming available, the virtualization layer can adopt these interfaces to talk to the hardware in a vendor-independent manner.
This chapter introduces a new distributable Optical Virtualization Controller (OVC) that supports control of the network as well as virtualization. The control is mostly related to the translation of protocols and building an SDN adaptation layer for proprietary protocols, while exposing open protocols to SDN controllers. The virtualization enables to split a network into slices and distribute them to clients. First, the OVC is introduced by presenting the general architecture and the design choices. Then, the performance is evaluated. This final step examines the shortest path algorithm, the delay introduced by the virtualization layer and deployment options for its components.

4.1. Architecture

This section introduces the architecture of the OVC — an actor-based implementation of a unified controller and hypervisor for optical networks. An embedding in the general context and the main functional blocks are summarized in Fig. 4.1. The OVC is designed to fully virtualize optical networks based on lambda switching and supports the whole spectrum from a single network element representation to a direct mapping. The mapping also includes subsets of the network that are assigned to the respective clients. Like many other hypervisors, the OVC acts as a proxy or mediation layer between the controllers and the hardware in the physical network. It is built up of actors, which are programmatic base units that contain an internal state and communicate with each other through messages. They are a convenient way of handling concurrency and improving the performance by parallelizing sequential tasks. Since the message exchange can take

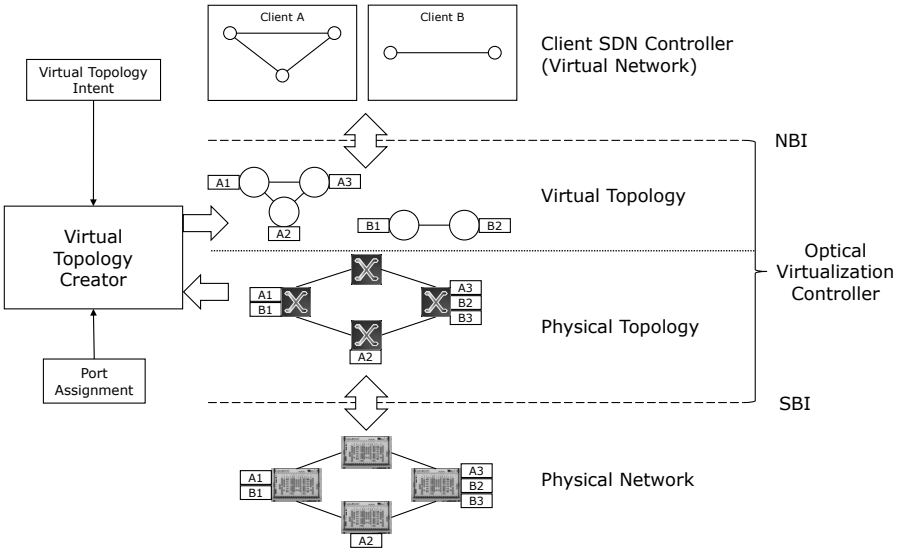


Figure 4.1.: General architecture of the OVC.

place using a network, a high flexibility of deploying the components is achieved, leading to a potentially distributed system. The OVC is able to slice and control an optical network and is built following a layered architecture. The layers cover functional blocks and are also inspired by the actor hierarchy, which describes a supervision dependency between the actors. North- and southbound communication utilizes REST interfaces and accesses resources through HTTP. Southbound an open device interface is used. Northbound a number of open (and mostly experimental) protocols is exposed toward a controller or an orchestrator.

4.1.1. Layers

The OVC follows a layered architecture built around functional blocks. A functional block is a group of actors that have a particular purpose that is independent of neighboring layers. The four (logical) layers (see Fig. 4.2) — from top to bottom — are:

Northbound Interface This layer includes different implementations for the communication toward clients' controllers or orchestrators. The NBI offers open protocol descriptions, which are preferably standardized, and exposes the topology together with the control of the network through them. It represents an entry point to the OVC from the outside world, usually a controller or an orchestrator.

Virtual Network This layer manages the representation of every client's custom slice, which is controlled by the client through one NBI. The abstraction model decides on the exposed details and ranges from a single node abstraction to a direct mapping, which has implications on the granularity of control. The client is able to retrieve the assigned portion of the network and control it according to the agreement.

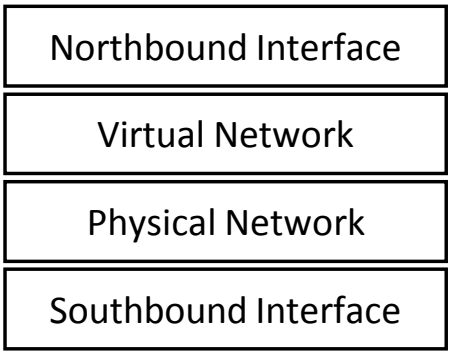


Figure 4.2.: Logical layers of the OVC.

Physical Network This layer collects information about all domains and resources managed by the hypervisor. The network elements are disseminated through the underlying SBI and stored in a generic representation. This enables the control of different network elements and technologies without unnecessary details. The work is mainly concerned with optical equipment even though other layers could be included too.

Southbound Interface This layer comprises implementations for specific interfaces to communicate with the hardware, directly or through a mediator, such as an NMS or a domain controller. The available hardware and resources are abstracted and a generalized representation is reported upward. As soon as standardized hardware interfaces are available, a single SBI implementation can talk to hardware of different vendors.

Most of the presented layers group many actors that are responsible for subtasks. The communication between layers is carried out exclusively through messages. The handled messages at the border of a layer represent an interface in the classical sense.

4.1.2. Topologies

The logical units, i.e., layers, are not limited to a single instance or actor, e.g., NBI instances for different customers. This fact is reflected in the topology representations shown in Fig. 4.3. The boxes represent topology instances consisting themselves of one or more actors. Every layer uses a different representation that is best-suited for the task at hand. At the SBI, it makes a difference if a single device or a whole network domain is managed. For a single device, the focus is on providing a proper abstraction to the upper layer. In this case, it is superfluous to capture and represent topology information except for neighboring nodes. This is not true for a network domain, which needs to capture the topology in order to allow the upper layer a correct representation of the managed network.
The physical infrastructure is the heart of the controller that contains all the available (abstracted) information that has been collected by the SBIs. The OVC is only aware of the whole topology at this part of the architecture. This layer needs to build a topology from the input by aggregating all incoming data from the underlying layer. The physical

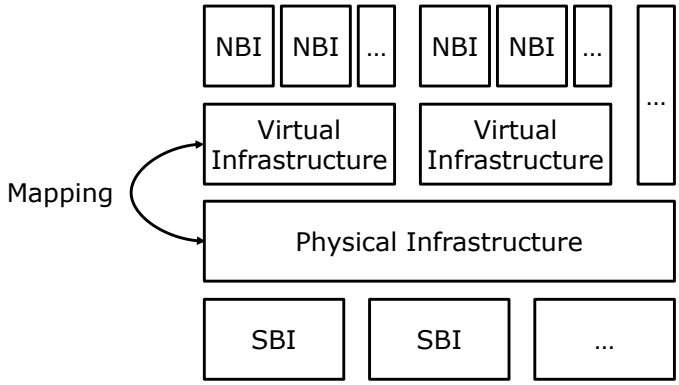


Figure 4.3.: Topology instances at different layers.

representation forms the basis for a virtualization of the available resources. It is used to calculate paths through the network that are mapped to virtual links.
The virtual infrastructure comprises network slices that are subsets of the physical infrastructure with regard to network elements, links, ports etc. There is an assignment of the aforementioned resources to every client. Based on these resources and the requirements of the client, defined by an intent, a virtual topology is created. This virtual network is then exposed to the client. The assigned resources can be either exclusive or shared. In the first case, they are reserved and in the second case, they may be consumed by others too. The client has full control over the exposed virtual infrastructure, according to the agreement or contract in place.
At the uppermost layer, the (virtual) topology is exposed toward the client through one of the supported protocols. The controller or orchestrator is able to interact with the OVC through this interface. Therefore, at least one NBI needs to be chosen and exposed. It is possible to use multiple interfaces by design. This can be useful if a specialized interface for different tasks is needed, e.g., for topology dissemination. The NBI abstracts the protocol details from the virtual representation and allows the client to choose the most appropriate one.

4.2. Design Choices

The architecture is the underlying blueprint for the OVC. To implement the presented architecture, multiple decisions had to be made about the design. These range from high-level decisions like the right programming framework to details about the right protocol and algorithm. In this section, the most important ones are introduced and explained.

4.2.1. Mediator Approach

Hypervisors are typically deployed as a mediation layer. One additional reason for the mediator approach is the legacy equipment. To introduce open interfaces to these

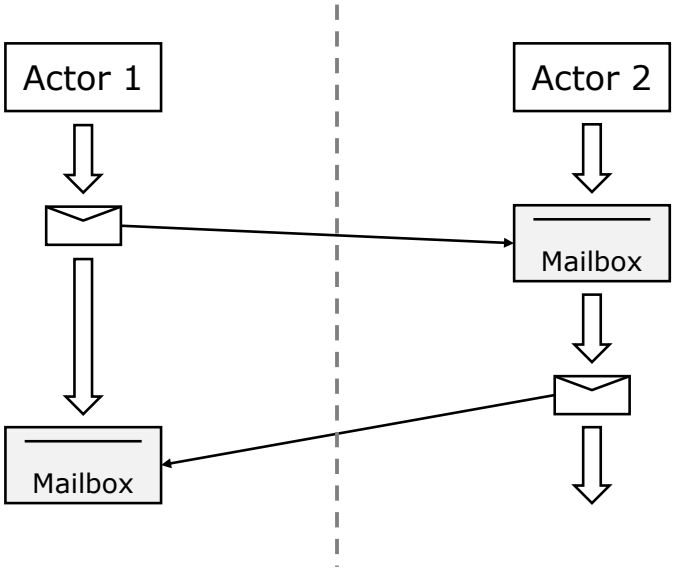


Figure 4.4.: Interaction and message exchange between actors.

network elements, a translation layer is needed. This limitation might be solved by the next generation of equipment that supports open interfaces by default. Still, it is likely that a domain controller will expose standardized interfaces with a network scope to hide the complexity of configuring an optical network and its elements. Therefore, it is important to be aware that the optical network requires some time to apply changes. Ongoing configuration tasks need to be taken into account when handling new incoming requests.

4.2.2. Actor Model

Actors are a model for concurrent programming and can be seen as building blocks for software applications. Each actor itself runs sequential code. In contrast, the set of actors that form an application is executed in parallel, i.e., a parallelization of sequential tasks. The actor model has been adopted by programming languages like Erlang and is also available as an extension through libraries, e.g., akka for Scala and Java [Lig18]. In the akka library, an actor has a state, an exchangeable behavior, one mailbox and optionally children. Actors communicate with each other only through messages and are otherwise isolated. This is illustrated in Fig. 4.4. The internal state should exclusively be accessed and changed through messages. For this reason, all activity is triggered by messages. The behavior of an actor decides how an actor reacts to a particular message and may be different depending on the state. All actors have their own mailbox, which allows message exchange between actors who know each others reference, which roughly corresponds to an address. Actors are structured hierarchically with an arbitrary degree per node. It means that every actor has exactly one parent and can have multiple children. This is necessary to have a chain of command and a responsible entity for handling failures of

children or subtrees. Due to their decoupled existence, actors allow for a distributed deployment as long as the message exchange can take place, e.g., through a network. This enables a horizontal scaling and migrations to improve access times. It means that arbitrary actors or subtrees can be deployed on separate machines, sending messages through the network. Each layer is a potential candidate for deployment on a separate machine. By reducing the granularity, each layer can be subdivided into smaller portions that, again, can be distributed. This gives the OVC the ability to be very flexible and (horizontally) scalable.

4.2.3. Extensibility

Extensibility is a characteristic of a software that enables it to be extended by providing a custom implementation for one of its existing components. Typically, extensibility is achieved by providing programmatical interfaces that need to be implemented. Applying the actor model, this is handled differently. Compared to a typical interface in Java, no methods are defined that have to be implemented, but a number of messages has to be accepted, handled and replied to. This way, the NBI and the SBI can be easily extended by following the messaging contract, which allows for the integration of future protocols. Throughout this thesis, a number of protocols for the NBI has been implemented, verifying the extensibility. The SBI supports only one implementation so far.

Northbound Interface

Northbound, an interface for controlling the assigned slices is required. In packet-switched networks, a prominent protocol — with limited applicability to circuit switching — is OF. Common choices for transport networks such as PCEP, BGP-LS, NETCONF and RESTCONF are introduced in Sec. 2.1.4. The following protocols have been implemented and are part of the OVC.

Early on, OF 1.0 and 1.3 were adopted. The goal was to use existing SDN controllers, such as Floodlight [Pro18] or OpenDaylight [Ope18e], that were mostly limited to OF. Initially, to expose optical networks, a single switch was used. In this representation, the whole network is abstracted into a single big switch covering all available (client) ports. Port-based flows are directly translated into lightpaths between the corresponding client ports. When the flow is removed, the lightpath is torn down. A direct mapping results in a more detailed representation. Each optical network element is mapped to a switch, which includes the client and the line ports. A topology is created by announcing links between the line ports, according to the network topology. It is then necessary to correlate flows on individual switches, to create a lightpath. This can be achieved, e.g., by using the cookie field for attaching an identifier. By concatenating these flows, the source and destination ports can be identified and a setup is triggered. These workarounds confirm that OF is not very well suited for the control of an optical network.

Next, the Control Orchestration Protocol (COP) was added to the available interfaces. It is an open source model, developed in the European project STRAUSS [STR16] and

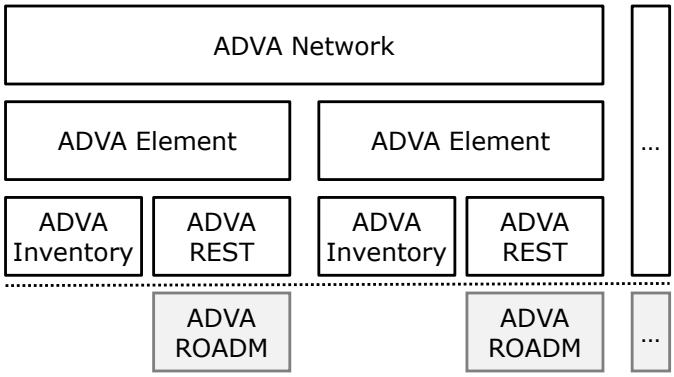


Figure 4.5.: Structure of the SBI controlling the ADVA equipment.

is an early approach for orchestrating networks [Vil+15b]. COP is defined as a YANG model that can be easily used in combination with NETCONF or RESTCONF and provides a basic tooling for code generation. The model definition abstracts the functions that an SDN controller needs to orchestrate networks employing heterogeneous control plane paradigms and data plane technologies. Models for topology dissemination, service management and path computation have been published. For simplicity reasons and because the available toolchain provided classes and stubs for a REST implementation, a RESTCONF-like protocol was chosen. COP is one available NBI of the OVC for exposing the topology as well as triggering the lightpath setup and tear down.
The TAPI is an effort by the ONF [Qia+16] that uses concepts developed in COP. It is the third protocol that is exposed northbound. The TAPI is more mature than COP and offers additional and extended interfaces, e.g., for notifications, path computation and virtual networks. In order to receive more flexible code stubs than the ones generated by the provided toolchain, a plug-in for `pyang` was developed. The changes and extensions are described in App. A. In summary, the YANG models are provided as input and the generated output comprises Java beans and code stubs for a Java implementation with akka.
The final addition to the set of NBIs is the virtual topology interface (App. B) introduced in Sec. 2.3.1. It can be called by clients to manage their slices. The interface is capable of providing virtual topologies based on the intents submitted by the client. It enables the user to modify existing virtual networks and remove topologies that are no longer needed. The created topologies are exposed through any available protocol for the NBI.

Southbound Interface

Only one SBI was implemented throughout this work. It is an open but proprietary REST interface for ADVA devices, ROADMs in particular. This interface manages a network section that is under the control of a single control plane. A simplified overview is given in Fig. 4.5. The figure represents the logical components rather than individual actors. An ADVA network captures all devices that are part of the controlled network. It comprises one or more ADVA elements and the links interconnecting them. The links

provide information about adjacencies between elements. Network elements maintain two things: the inventory and the REST device interface. The inventory contains all the information about the present hardware and its capabilities. Due to the modular structure of a ROADM, components like modules or plugs need to be captured. The device discovery as well as the configuration are carried out through the device interface. It is the gateway for the communication with the hardware.
Southbound extensions for other protocols can easily be added by implementing the messages that are expected by the physical infrastructure. In the foreseeable future, common protocols for optical equipment will be standardized, e.g., OpenConfig. Then, the proprietary actor stack can be replaced by a vendor-independent one that follows the description of the standardized interface.

4.2.4. Graphs and Algorithms

Generally speaking, all network topologies can be represented by a graph. From the topology instances presented in Fig. 4.3, the virtual and physical infrastructure rely on graphs, while the NBI and the SBI use simplified representations. To avoid implementing a graph representation from scratch, the Java library JGraphT [NC17] was chosen. It is a library that provides a graph representation, algorithms to operate on top of them and visualization tools. It is based on Java 8 and receives regular updates. The included algorithms cover many areas like cliques, cycles, shortest paths and spanning trees. These algorithm classes are implementing interfaces that group them by functionality, e.g., shortest path algorithms. A graph representation is used to manage the physical as well as the virtual topology. The main difference is the abstraction level of nodes. While mapping the nodes of the physical layer to the virtual layer is straight forward, the virtual links are mapped to paths through the physical network. This is enabled by a shortest path computation that utilizes the offered interfaces for algorithms. This choice makes it easy to integrate already existing algorithms that support this interface or to apply new ones. For example, default implementations can be compared to newly-developed algorithms without major changes in the code. The PLL implementation based on the description in Sec. 3.3 was implemented that way.

4.3. Performance Evaluation

This section evaluates the performance of the prototypical implementation of the OVC. To be more precise, it focuses on selected aspects, which represent the most important capabilities of the OVC. The three main parts are: the shortest path algorithm, the delay of the mediation layer and a distributed deployment / migration of OVC components. The shortest path algorithm is evaluated based on a number of representative optical networks and compared to Dijkstra's algorithm. The delay introduced by the mediation layer is measured in a local testbed with physical equipment for typical tasks. Finally, a distributed deployment spanning two geographically dispersed testbeds is considered, including a migration between both locations.

Network	Nodes	Links	Avg. Degree
cost266	37	57	3.1
dfn-bwin	10	45	9
geant	22	36	3.3
germany50	50	88	3.5
giul39	39	172	8.8
janos-us-ca	39	122	6.3
nobel-eu	28	41	2.9
nobel-germany	17	26	3.1
polska	12	18	3
sun	27	102	7.6
ta2	65	108	3.3
zib54	54	81	3
p2p-Gnutella04	10876	39994	7.4

Table 4.1.: Properties of selected reference telecommunication networks [Zus06] and one large computer network [LK14].

4.3.1. Evaluation of Shortest Path Algorithm

The shortest path algorithm is part of the virtual topology creator (see Fig. 4.1). It is used to calculate paths between nodes in the network. These paths correspond to virtual links and are exposed to the client as part of his virtual topology. In this regard, a major contribution was making the algorithm capable of coping with change in the network. After edges are added or removed, the developed cleanup procedure needs to update the tables that contain the precomputed hop values. Without this procedure a removal of edges was impossible. These extensions are the focus of the evaluation but first a baseline is established. This is achieved by assessing the preprocessing and the shortest path calculation and comparing the run time to Dijkstra's algorithm, which represents the default choice in controllers, for individual queries. Second, the edge removal and the cleanup procedures are compared. Both are the focus of the algorithm's evaluation. The first step creates a baseline to be able to estimate the factor that is introduced by the prototypical implementation of the existing base algorithm. Since the prototype has been developed in Java and only limited optimizations have been applied, the run times are considerably higher than the ones presented by the original authors. However, the goal of this thesis is to present a proof of concept in order to verify that the developed cleanup procedures only contribute a limited amount of overhead. The absolute time values can be significantly reduced by an optimized implementation in a programming language like C++. Nevertheless, the presented results give an indication on the viability of the approach. For the evaluation a number of reference telecommunication networks and one larger computer network are used. Their properties are summarized in Tab. 4.1. The networks cover typical sizes for optical networks as well as one larger network in order to estimate the algorithm's scalability.

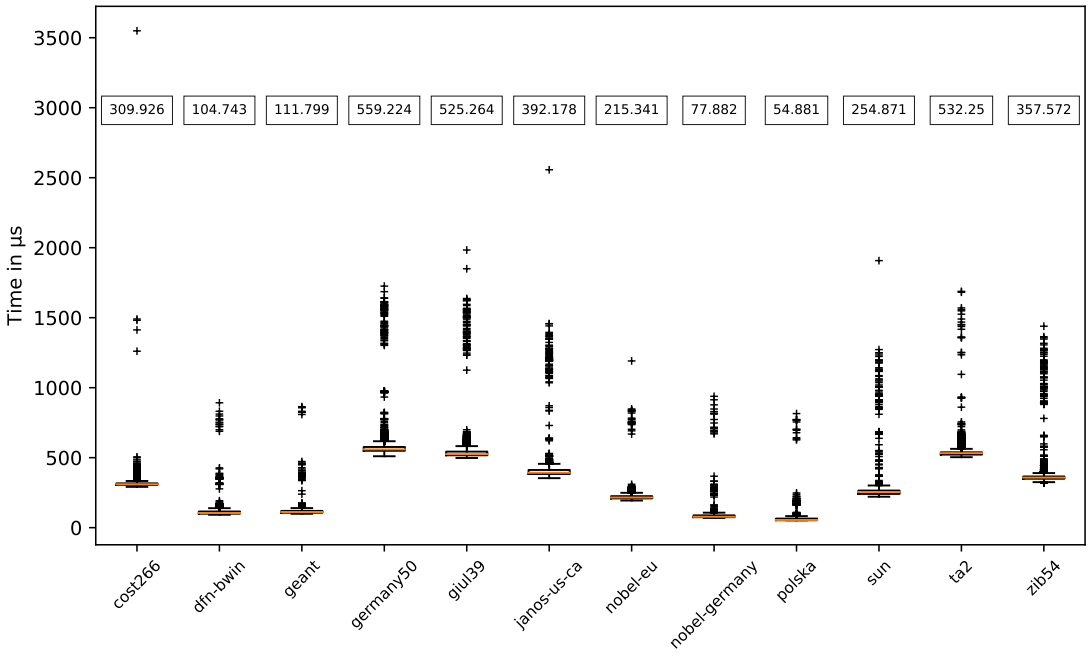


Figure 4.6.: Time for preprocessing each reference network.

The machines used to run these experiments, if not stated otherwise, were HP ProDesk 400 G2 Mini PCs equipped with an Intel Core i5-6500T @ 2.50 GHz (turbo frequency 3.10 GHz) and 16 GB of RAM (DDR4-2133 MHz). They were exclusively used for the measurements to avoid any interference by other users or processes. Each machine was running an Ubuntu Server 16.04.4 LTS 64-Bit Linux for the OS. The Oracle Java Virtual Machine (JVM) version used throughout the measurements is 1.8 (64-Bit, Update 161).

To get meaningful results, the JVM is provided with a warm-up time by discarding the first measurements — roughly around 10 %. The number of runs mentioned throughout this section always excludes the warm-up time. Mainly box plots are used to present the measured results. The bottom of the box represents the first quartile and the top of the box corresponds to the third quartile. The median is marked by an orange line and its actual value is captured in a box (per column of the graph). It has been chosen as an indicator because it is less sensitive to outliers than a mean value. The end of the upper and lower whisker represents the highest and the lowest datum within 1.5 times the interquartile range. Outliers are marked by crosses. Extreme outliers are assumed to be peculiarities of the JVM and could be removed by a programming language that is not relying on a virtual environment and automatic garbage collection.

Preprocessing and Shortest Path Calculation

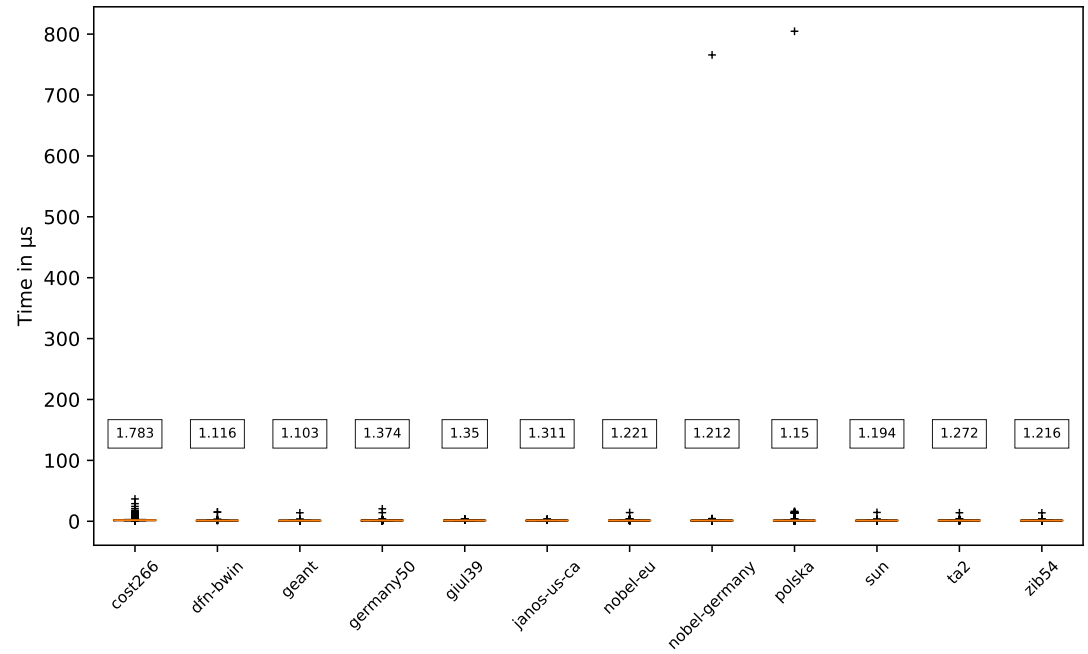
The first step of the performance evaluation is establishing a baseline by evaluating the preprocessing time and the query time. They are compared to Dijkstra's algorithm, which performs a new calculation for every query and represents the default choice in most controllers. First, the reference networks are tested and afterwards one large computer network is benchmarked.

The preprocessing times for the reference networks are shown in Fig. 4.6. The x-axis lists the processed networks, while the y-axis indicates the time in µs. 10 000 runs per network have been conducted for the presented results. The median values range from 54.9 µs to 559.2 µs. In general, large networks, i.e., with a larger number of nodes and links, lead to higher preprocessing times. The reason is that one search using Dijkstra's algorithm is started from every node, even if it is pruned immediately, and more links lead to more options that need to be explored during the search. Therefore, the lowest preprocessing times are achieved for `polska` and `nobel-germany` — two networks with a small number of nodes and links. The highest times are accounted to `germany50`, `giul39` and `ta2`. This is an expected result because of the high number of nodes and links. By comparing `germany50` (50 nodes, 88 links) and `janos-us-ca` (39 nodes, 122 links) it can be seen that more links and thereby a higher average node degree leads to a better run time than a higher number of nodes with a lower degree. This indicates that the number of nodes has more influence on the preprocessing's run time than the number of links.

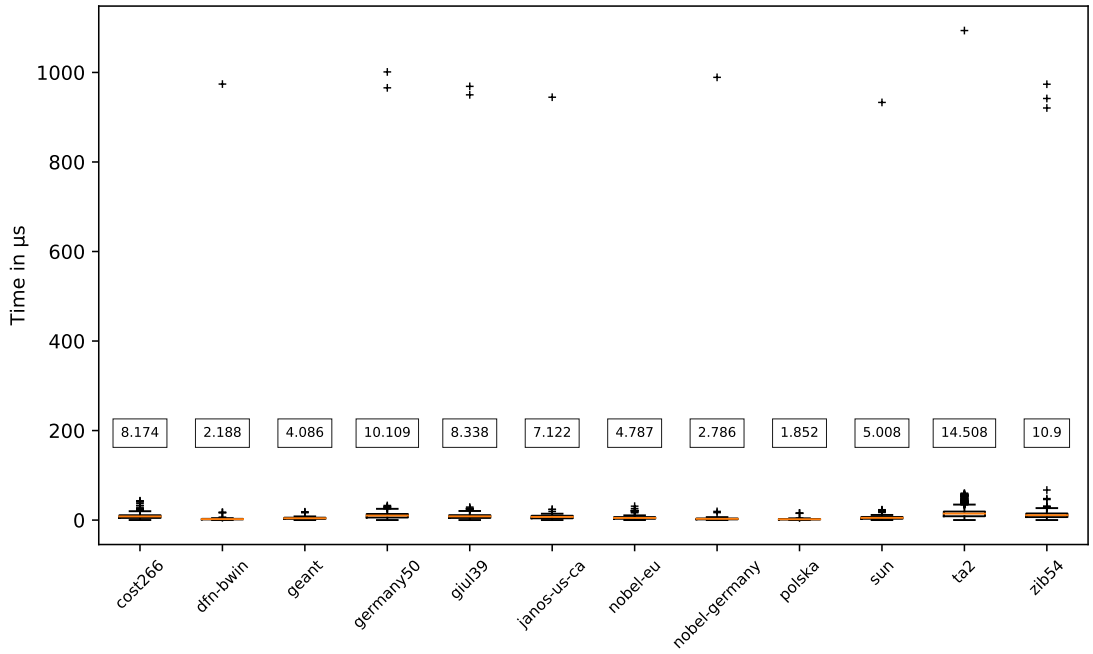
Next, the query time for each graph is evaluated, i.e., the time that is needed to answer a shortest path query between two random endpoints in the network. The nodes have been varied for every query and a total of 10 000 queries has been conducted. Fig. 4.7 summarizes the results in the form of box plots. The x-axis lists the tested networks and the y-axis shows the time in µs for every query.

Fig. 4.7a covers the measurements using PLL for the queries, which requires a preceding preprocessing. It can be seen that the median values are all below 2 µs for the reference networks. In general, it takes longer to answer queries for larger graphs than for smaller graphs. The size is not the only factor playing a role. The query time mostly depends on the number of labels per node, since calculating a shortest path translates to finding a common hop with the lowest total distance (see Sec. 3.2.2). However, the number of labels is not only influenced by the number of nodes and links but also dependent on the structure of the network. For example selecting central nodes, through which many shortest paths pass and that are processed early on, can reduce the size considerably. The exact relationship between the structure of the network and the number of labels is out of scope. It is one explanation why `cost266` has the highest query time of all reference networks, even though it neither has the most nodes nor the most links. A similar argument can be used to explain, why `geant` is faster than `dfn-bwin`. The outliers are assumed to be connected to the JVM. The times could be improved by adopting memory- and speed-optimized techniques for "join" operations in data bases.

In order to receive a base for the evaluation, Dijkstra's algorithm is used for every query. In contrast to PLL, this approach does not require any preprocessing and is executed



(a) Using PLL queries based on precomputation.



(b) Using Dijkstra's algorithm for every query.

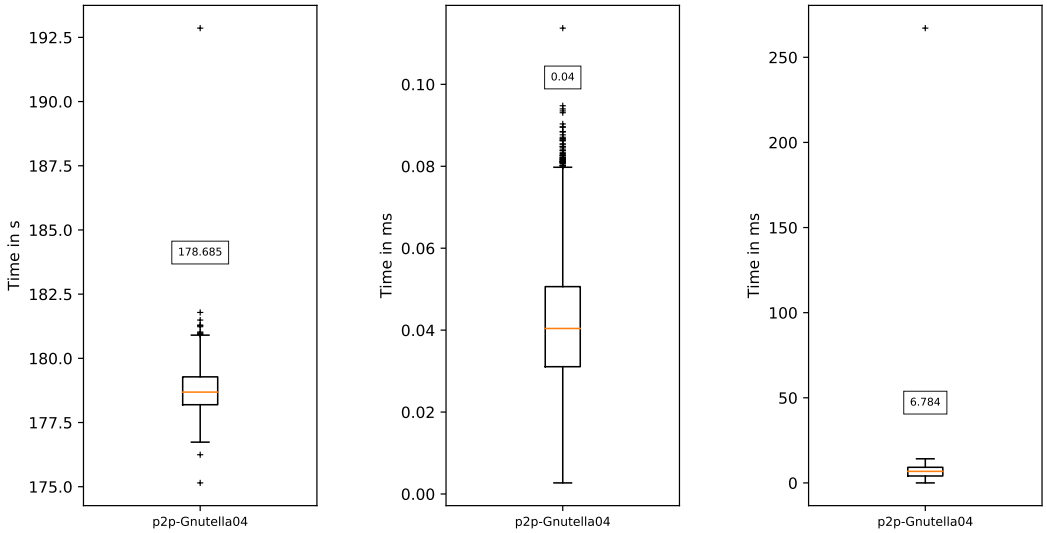
Figure 4.7.: Time for running queries between two nodes in the reference networks.

Table 4.2.: Summary of measurements for reference networks (time in µs) and number of queries for break-even point.

	cost266	dfn-bwin	geant	germany50	giul39
Dijkstra query time	8.174	2.188	4.086	10.109	8.338
PLL query time	1.783	1.116	1.103	1.374	1.35
Query time difference	6.391	1.072	2.983	8.735	6.988
Preprocessing time	309.926	104.743	111.799	559.224	525.264
Break even (# queries)	49	98	38	65	76

janos-us-ca	nobel-eu	nobel-germany	polska	sun	ta2	zib54
7.122	4.787	2.786	1.852	5.008	14.508	10.9
1.311	1.221	1.212	1.15	1.194	1.272	1.216
5.811	3.566	1.574	0.702	3.814	13.236	9.684
392.178	215.341	77.882	54.881	254.87	532.25	357.572
68	61	50	79	67	41	37

without any prior knowledge, except for the current state of the graph. The results are shown in Fig. 4.7b. Here, the size of the network heavily affects the measured time for the Dijkstra search. Big networks like `ta2`, `zib54` and `germany50` have the highest median with more than 10 µs. The only network with a median below 2 µs is the small `polska` network due to its small overall size. Due to Dijkstra's nature, the run time mostly depends on the network's size.
The results achieved by PLL need to be put into perspective by comparing them to Dijkstra's algorithm. Tab. 4.2 summarizes the captured median values and offers additional insights. The table shows the times needed to answer queries using PLL and Dijkstra. Both have been presented in Fig. 4.7 before. The time difference between them is shown in the third row, capturing the time advantage of using PLL. The last two rows list the preprocessing times and the number of queries required to break even, i.e., compensate the preprocessing. First of all, it can be seen that the implementation of PLL is faster than using Dijkstra for all reference networks. The absolute time improvement is higher for larger networks, e.g., `ta2` or `zib54`. Timewise, this means an improvement of 13.236 µs and 9.684 µs. These values translate to savings of about 91 % and 89 %, respectively. However, for very small networks like `polska` the benefit is just around 0.702 µs or 38 %. The time difference divided by the processing time decides how many queries it takes before a break-even point is reached. For the selected reference networks, the break-even point is reached after 37 to 98 queries. Smaller networks tend to needing more queries due to the lower time difference, while larger networks reach this point earlier. Some previously described network properties, e.g., the node degree, implicitly play a role in this regard. Keeping in mind that a virtual topology consists of multiple nodes and links and every link results in at least one query, it can be safely assumed that the savings will quickly outweigh Dijkstra's initial advantage.



(a) Preprocessing. (b) PLL queries. (c) Dijkstra queries.

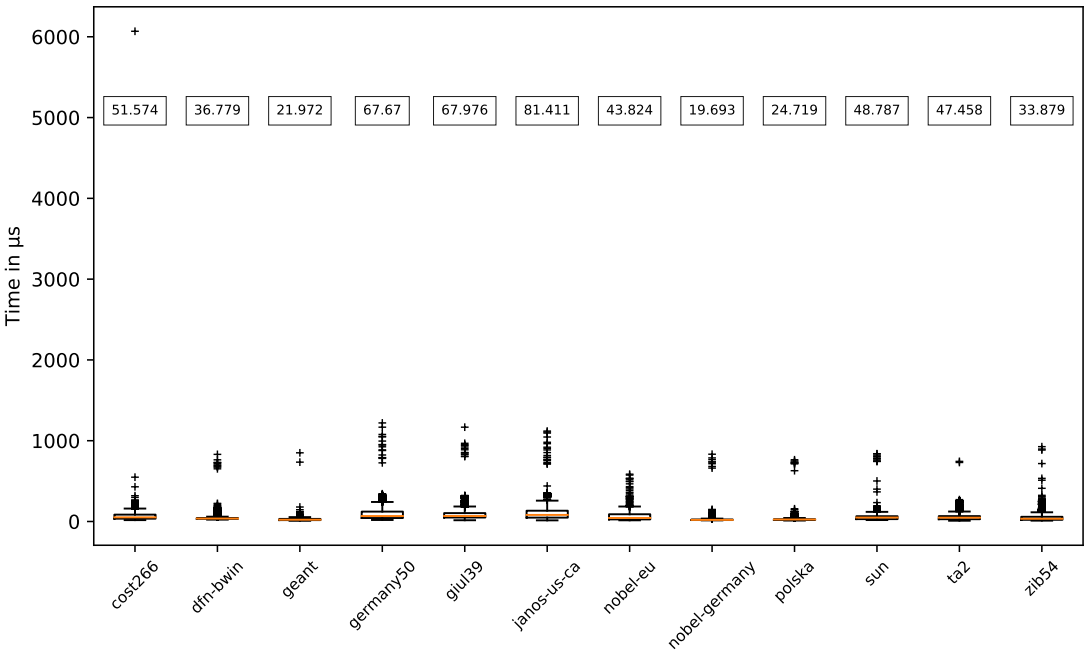
Figure 4.8.: Time for preprocessing and shortest path queries for `p2p-Gnutella04`.

The benefits of PLL for the reference networks, which represent rather small graph instances, have been demonstrated. Therefore, one could argue that precomputing APSP is even faster because a query corresponds to a simple lookup. The clear drawback is the memory space that is needed to store these values. For small networks, this might not be an issue, depending on the used hardware, but as the network size grows, this approach becomes impractical. Evaluating the memory size is out of scope for this work and some data is available in the original work [AIY13]. Nevertheless, in order to prove the scalability of the chosen approach, a larger network needs to be considered. For this purpose, the `p2p-Gnutella04` network is chosen with more than 10 000 nodes and slightly less than 40 000 edges (see Tab. 4.1). For this graph, the results based on 1250 runs for the preprocessing and 10 000 runs for the queries are summarized in Fig. 4.8. The preprocessing for a graph of this scale takes roughly 3 min (Fig. 4.8a). Afterwards the queries are reduced to 40 µs (Fig. 4.8b), while a Dijkstra query takes 6.784 ms (Fig. 4.8c). This means that the time difference is 6.744 ms, which corresponds to 99.4 %. With these savings per query, it takes about 26 500 queries to break even after performing the preprocessing. Considering the size of the graph, especially the number of nodes, it seems reasonable that the value is reached quickly. For example, this number corresponds to queries from every node to only three other nodes in the graph. Since every link can host many channels, even the same pairs might be queried multiple times. Taking into account that a full preprocessing is only needed once in the beginning and the query times are notably improved, the approach is also applicable to larger graphs. To get a feeling for the factor that can be gained by an optimized implementation, the results of this thesis are compared to the numbers presented by Akiba et al. [AIY13]. They evaluated a bigger Gnutella network with 63 000 nodes and 148 000 links. It takes

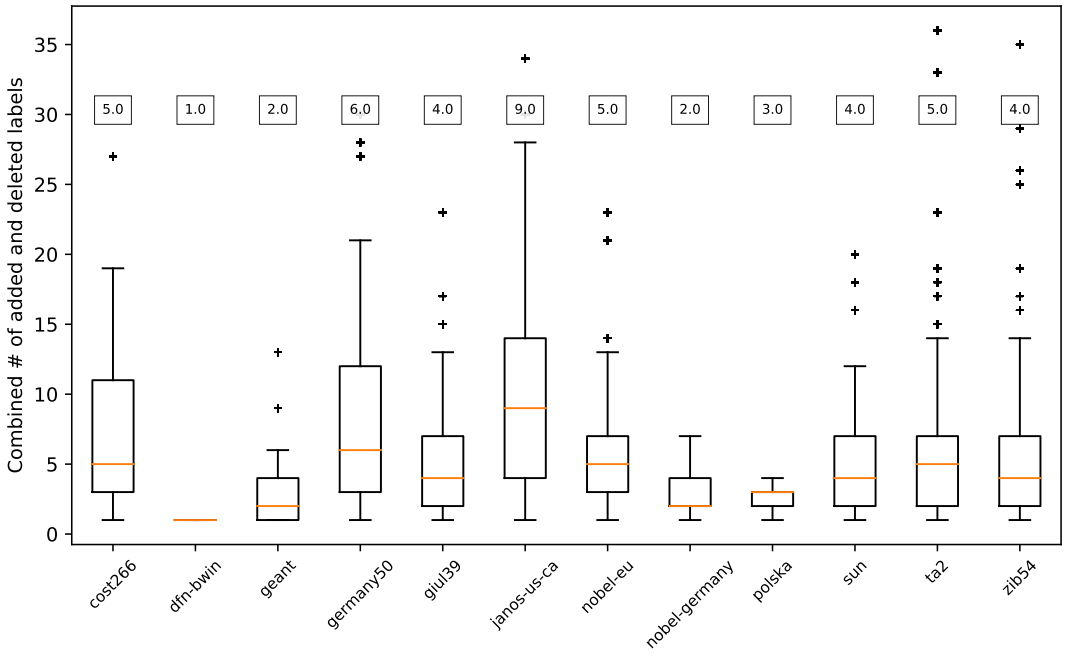
them 53 s to preprocess the network and 5.2 µs to answer queries. Disregarding the difference in size for the moment, we can assume that at least a factor of 3 can be gained in the preprocessing stage and a factor of 7 is achievable in query time. The speedup for preprocessing can be treated as a lower bound because the number of nodes is 6 times higher than the one in `p2p-Gnutella04`. Also, the number of links is 3 times higher. From the results in this thesis, it can be seen that the preprocessing depends on the network size and therefore, the gain for the smaller `p2p-Gnutella04` network should be higher. For these reasons, improvements of one order of magnitude by optimizing the code seem realistic. The growth of the query time is harder to predict but at least their presented measurement gives a rough idea of the achievable improvements.

Adding Edges

In this subsection, the contribution of this thesis to the PLL algorithm is evaluated. The presented extensions are needed in order to support dynamic changes of the graph. In particular, the contribution comprises the cleanup of obsolete entries after an edge has been added to the graph as well as the handling of an edge deletion including the cleanup. The presented measurements were captured by doing one preprocessing and then removing a random edge and re-adding it again. Every run captured one removal and insertion of an edge. After that, a new edge was selected and the procedure was repeated. One graph captures the time needed to finish the operation on the edge and another one summarizes the number of operations applied to the labeling, i.e., the number of added and deleted labels in order to re-establish a correct labeling.
Fig. 4.9 shows the run times (in µs) and the number of operations on the labeling for the reference networks. The data was collected over 10 000 runs. It can be seen that the times range from 19.693 µs to 81.411 µs (see Fig. 4.9a). The values here are influenced by two major factors: the (average) degree of the nodes and the number of operations. A high degree means that many neighboring nodes have to be visited even if no updates are required in this direction. The number of operations is an indicator of how many entries and nodes are affected by the update, i.e., each change requires to check the neighbors for updates that need to be propagated. A higher number of operations results in a longer cleanup and total processing time. The number of operations, namely the sum of added and removed labels, is depicted in Fig. 4.9b. It can be seen that `janos-us-ca` needs the longest time to finish the process of adding an edge. This result is a combination of applying many updates to the labeling as well as having a high average degree. The influence of a high node degree can be seen by comparing `germany50` and `guil39` or `sun` and `ta2`. Both pairs have approximately the same run time. While `germany50` and `ta2` require more operations, `guil39` and `sun` need a lower number of updates. Nevertheless, a notably higher node degree leads to similar run times. One extreme example in this regard is `dfn-bwin`. It is a fully meshed network that only needs one operation but due to the high node degree, it needs longer to finish than `nobel-germany`, which has more nodes and operations but a lower average degree. In the end, the labeling itself, which in turn depends on the chosen ordering, influences the number of labels that need to be changed.



(a) Run time comparison for adding an edge.



(b) Number of changes in the labeling for adding an edge.

Figure 4.9.: Results for adding edges to the reference networks (with cleanup).

Table 4.3.: Time for adding an edge to reference networks, the speedup factor compared to a full preprocessing and the number of queries to break even.

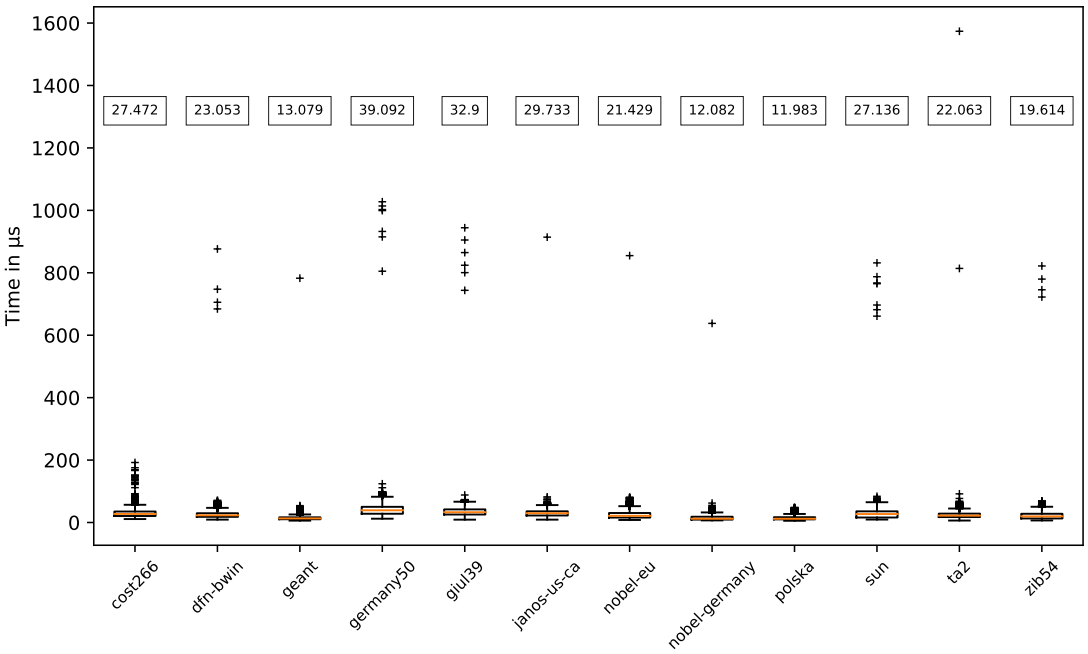
	cost266	dfn-bwin	geant	germany50	giul39
Adding an edge (in µs)	51.574	36.779	21.972	67.67	67.976
Speedup factor (Preprocessing)	6	2.8	5.1	8.3	7.7
Break even (# queries)	9	35	8	8	10

janos-us-ca	nobel-eu	nobel-germany	polska	sun	ta2	zib54
81.411	43.824	19.693	24.719	48.787	47.458	33.879
4.8	4.9	4	2.2	5.2	11.2	10.6
15	13	13	36	13	4	4

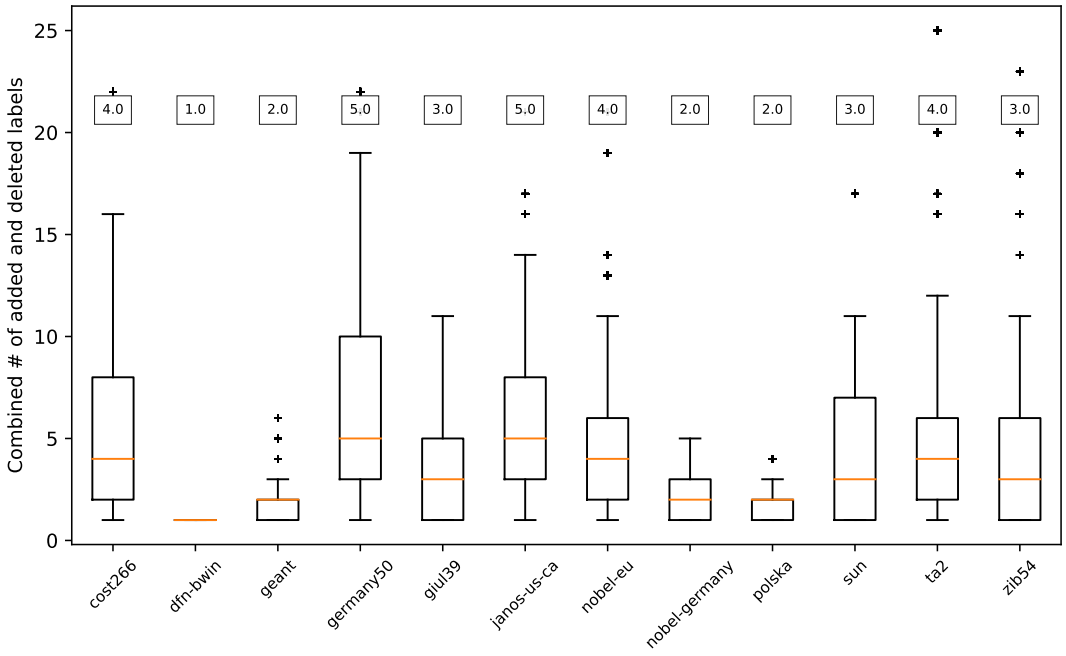
Next, the improvements compared to a full preprocessing are examined. The numbers are presented in Tab. 4.3. It is obvious that adding an edge and cleaning up is faster than applying a full preprocessing. The speedup factors range from 2.2 to 11.2. The larger a network the bigger the speedup factor because most often, the changes are just affecting a limited region. That is the reason why the smallest networks experience the lowest speedup, e.g., `polska`, while big networks such as `ta2` benefit the most. Because of all speedup factors being larger than 1, the break-even point occurs earlier than with executing a preprocessing. It can be reached as soon as after 4 queries or take up to 36 queries for the reference networks. These results demonstrate the benefits of a dynamic update: the correct labeling is re-established noticeably faster than with a new preprocessing.

As a last step, the overhead introduced by the cleanup procedure is quantified. As described in Sec. 3.3, an algorithm for adding edges was already available but it was in need of an extension that supports a cleanup. Therefore, the same implementation with a deactivated cleanup procedure is used to identify the overhead. The results for adding an edge without a subsequent cleanup are shown in Fig. 4.10. The median run times in Fig. 4.10a range from 11.983 µs to 39.092 µs. All timing results are of course lower because the cleanup is omitted. They are more connected to the size of the network than their counterparts with a cleanup. This is an effect caused by (resuming) Dijkstra's algorithm that is used to update the labeling. Hence, small networks are handled faster than large networks. Nevertheless, the time also depends on the number of updated entries because an update at a node results in all its neighbors being checked even if they are pruned. In general, the number of changes, which is shown in Fig. 4.10b, is reduced due to the cleanup. As expected, the cleanup removes obsolete labels from the labeling. Thereby, the number of changes is lower without a cleanup. This can be seen by comparing both box plots (with and without cleanup), even though the median is not always affected.

Even more interesting is the overhead that is introduced by the cleanup procedure, which is captured in Tab. 4.4. The absolute value varies between 7.611 µs and 51.678 µs. Since



(a) Run time comparison for adding an edge without cleanup.



(b) Number of changes in the labeling for adding an edge without cleanup.

Figure 4.10.: Results for adding edges to the reference networks (without cleanup).

Table 4.4.: Impact of the cleanup procedure on adding edges (time in µs).

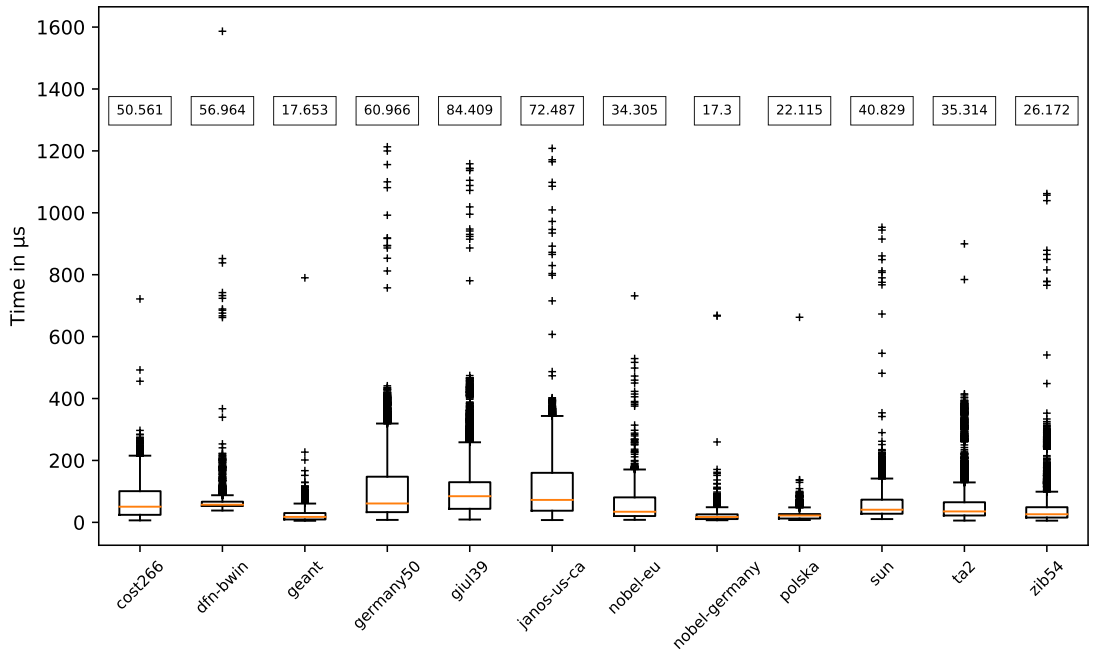
	cost266	dfn-bwin	geant	germany50	giul39
Adding an edge (no cleanup)	27.472	23.053	13.079	39.092	32.9
Cleanup time overhead	24.102	13.726	8.893	28.578	35.08
Slowdown factor for cleanup	1.9	1.6	1.7	1.8	2.1

janos-us-ca	nobel-eu	nobel-germany	polska	sun	ta2	zib54
29.733	21.429	12.082	11.983	27.136	22.063	19.614
51.678	22.395	7.611	12.736	21.651	25.395	14.265
2.8	2.1	1.7	2.1	1.8	2.2	1.8

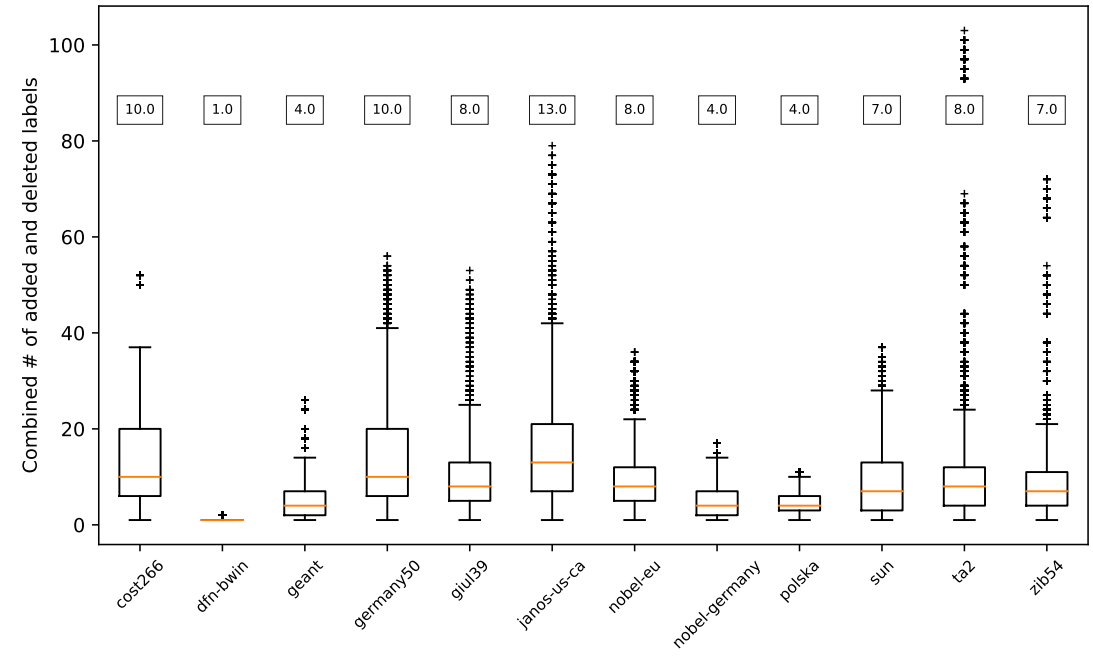
the run time for adding an edge also varies, it is not easy to classify the results. By calculating the slowdown factor caused by the cleanup, i.e., how many times longer it takes to finish the addition, the picture gets clearer. It can be seen that in most cases, the factor fluctuates around 2. One higher deviation can be observed for `janos-us-ca` with a factor of 2.8. This is related to the fact that for this particular network structure and the chosen ordering, the cleanup needs to remove more labels than in any other case. The median increased from 5 to 9, which means that the number of changes almost doubled. All in all, the cleanup on average only doubles the time needed for an addition. Compared to the benefits that can be gained by not having to run a new preprocessing and being able to remove edges, this slowdown seems still acceptable.

Removing Edges

Removing edges is an extension of PLL that was not available before and makes the algorithm fully dynamic with regard to edges. One reason why this feature was previously missing is that after adding an edge, a cleanup is required for the proposed removal to work. In addition, it is not only necessary to remove no longer valid labels after removing an edge but also to add missing ones, in order to maintain a correct labeling.
The measurements are captured in Fig. 4.11. Again, 10 000 runs have been conducted. The timing results are shown in Fig. 4.11a. The x-axis lists the reference networks and the y-axis shows the measured time in µs. Fig. 4.11b summarizes the number of changes that occurred in the labeling, when an edge was removed. The median values for the removal time range from 17.3 µs to 84.409 µs. Small networks such as `nobel-germany`, `geant` and `polska` experience the fastest cleanup times. These three have a similar amount of changes in the labeling. The differences result from different network structures, which lead to varying numbers of labels that are processed but not removed, thus not resulting in an operation. On the other hand, larger networks, e.g., `giul39` or `janos-us-ca`, need more time for the removal. The median number of changes is much higher for these networks. The outliers indicate that many entries in the labeling are affected in some cases.



(a) Run time comparison for removing an edge.



(b) Number of changes in the labeling for removing an edge.

Figure 4.11.: Results for removing edges from the reference networks (with cleanup).

Table 4.5.: Time for removing an edge from reference networks, the speedup factor compared to a full preprocessing and the number of queries to break even.

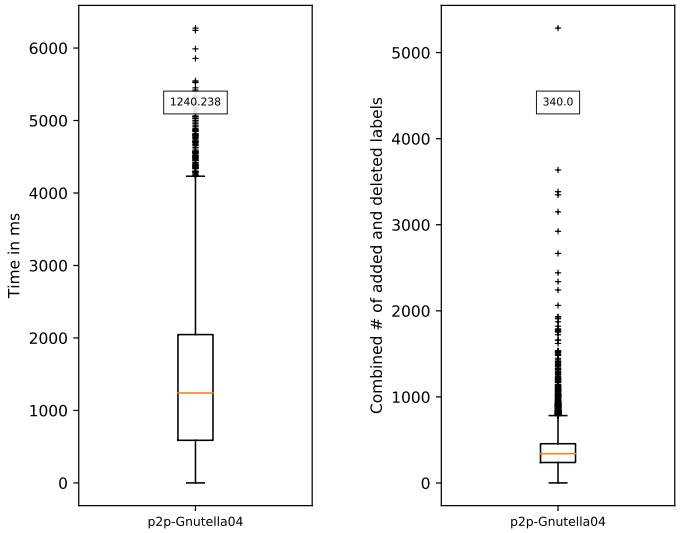
	cost266	dfn-bwin	geant	germany50	giul39
Removing an edge (in µs)	50.561	56.964	17.653	60.966	84.41
Speedup factor (Preprocessing)	6.1	1.8	6.3	9.2	6.2
Break even (# queries)	8	54	6	7	13

janos-us-ca	nobel-eu	nobel-germany	polska	sun	ta2	zib54
72.487	34.305	17.3	22.115	40.829	35.314	26.172
5.4	6.3	4.5	2.5	6.2	15.1	13.7
13	10	11	32	11	3	3

Tab. 4.5 gives an overview of the results for removing an edge from the reference networks, the speedup factor compared to a full preprocessing and the break-even point for this procedure compared to using Dijkstra. The speedup factors range from 1.8 to 15.1. This indicates a behavior similar to adding an edge. The larger the network the higher the speedup because often, only a part of the network needs to be updated. For `dfn-bwin`, the factor is small due to the high node degrees in a fully-meshed network. From the larger networks, `ta2`, `zib54` and `germany50` experience the largest speedup. This observation is related to the rather low average node degree around 3. For two of them this leads to a break even after as early as 3 queries. The network with the smallest speedup factor requires 54 queries. Finally, it can be stated that the removal procedure is faster than a preprocessing for all networks and more beneficial for larger ones.
Comparing the addition and removal of edges, there are two cases in which adding an edge is faster than removing it, i.e., `dfn-bwin` and `giul39`. From their properties, it is likely that adding is faster than removing for networks with a high average node degree. This observation requires further investigation and is deferred to future work.

Scalability

In order to evaluate the scalability of the developed extensions, adding and removing edges is also applied to the `p2p-Gnutella04` network. The measured time for adding an edge as well as the required operations on the labeling are summarized in Fig. 4.12 (10 000 runs). Fig. 4.12a shows that it takes 1240.238 ms (median) to add an edge to the graph. This means that the procedure is 144 times faster than doing a full preprocessing, and a break-even point is reached after 184 queries. The number of changes that need to be applied to the labeling has a median value of 340 (see Fig. 4.12b). Compared to the number of nodes and edges, these changes only concern a small amount of the labels, even though higher values can occur as indicated by the outliers.
The results for removing edges are captured in Fig. 4.13. It summarizes the time and the number of operations on the labeling (10 000 runs). Fig. 4.13a indicates that the median



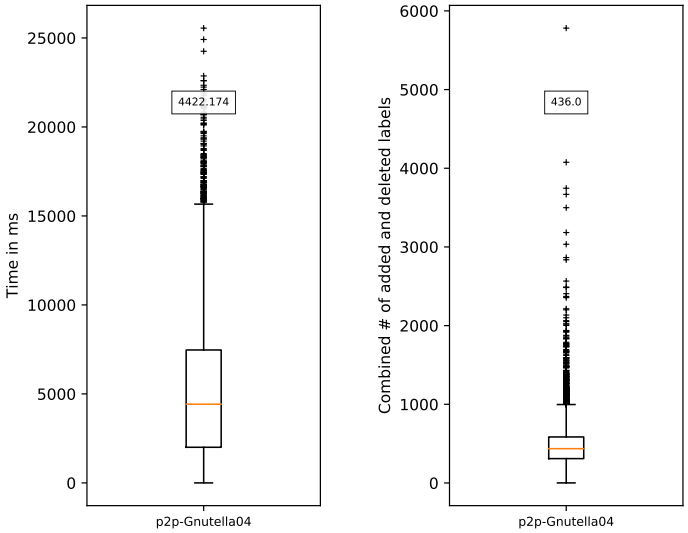
(a) Time in ms. (b) Number of operations.

Figure 4.12.: Results for adding an edge to the Gnutella graph.

value for removing an edge is 4422.174 ms. Compared to a full preprocessing, it is still more than 40 times faster and it takes 656 queries to reach a break-even point. Also in this case, the outliers indicate that higher values can occur leading to worse results. The deletion requires 436 operations (see Fig. 4.13b), which is more than for the addition.
To conclude the scalability evaluation, we can see that in all cases, the extensions providing a cleanup and the ability to add and remove edges are faster than a full preprocessing. Also, they considerably reduce the number of queries needed to break even compared to a preprocessing and thereby provide benefits to using Dijkstra's algorithm. Considering the previously estimated factors for a non-optimized implementation, it can be assumed that the run time for adding and removing edges in a network of this size can be reduced to seconds or even milliseconds. Since queries are typically the dominating operation in a network, the presented results verify that the extensions are well-suited for this application area.

4.3.2. Mediation Layer Delay

This section evaluates the delay introduced by the virtualization layer, i.e., the OVC. The measurements are performed using the same hardware as for the algorithm's evaluation in Sec. 4.3.1. Due to the exclusive use and the low load throughout the measurements, the obtained results provide reliable insights on the introduced delay by the OVC and its layers. The whole software stack of the OVC is deployed on one machine. Two use cases are investigated: installing and deleting a lightpath. On the left-hand side of Fig. 4.14, the total delay of the OVC (in µs) is shown. It shows that the median value for installing a lightpath is 3.5 times higher than the one for deleting a lightpath. This



(a) Time in ms. (b) Number of operations.

Figure 4.13.: Results for removing an edge from the Gnutella graph.

means in numbers that the installation takes 682.55 µs and the deletion 191.25 µs. The differences are examined by assigning portions of the total run time to the layers of the OVC (see Sec. 4.1.1).
The break down for installing a lightpath is captured in the upper right box plot of Fig. 4.14. It shows the time in µs that it takes to process the request at every layer. The processing of the connection request is started at the NBI. This layer takes 47.16 µs for parsing the JSON content of the request. Next, it is handed over to the virtual topology, which is responsible for finding the requested endpoints and sending the mapped request to the physical topology. The step completes in 38.857 µs before the physical topology needs to find the network elements and the designated SBI. After 35.628 µs, the request is forwarded and the southbound implementation takes over. This is the most complex part of the processing because it involves all the preparations for configuring the devices. Apart from finding the correct modules and ports, it involves the creation of new objects for maintaining the tunnel and channels. Only after these preparations are finished, the first message is sent to the device, which concludes the measurement.
Deleting a lightpath follows a similar order of events and the measured times are shown in the lower right box plot of Fig. 4.14. The NBI only needs 13.067 µs to finish the processing because the request does not contain a body that needs to be parsed. The identifier of the service that should be deleted is encoded in the Uniform Resource Locator (URL). Following the northbound processing, the request is routed through the virtual topology as well as the physical topology, which takes 40.578 µs and 36.564 µs, respectively. These two layers have very similar processing times for both cases, i.e., installing and deleting a lightpath. The slightly higher times for the deletion are related to the lookup and verification of the service at each layer. Finally, the SBI processes

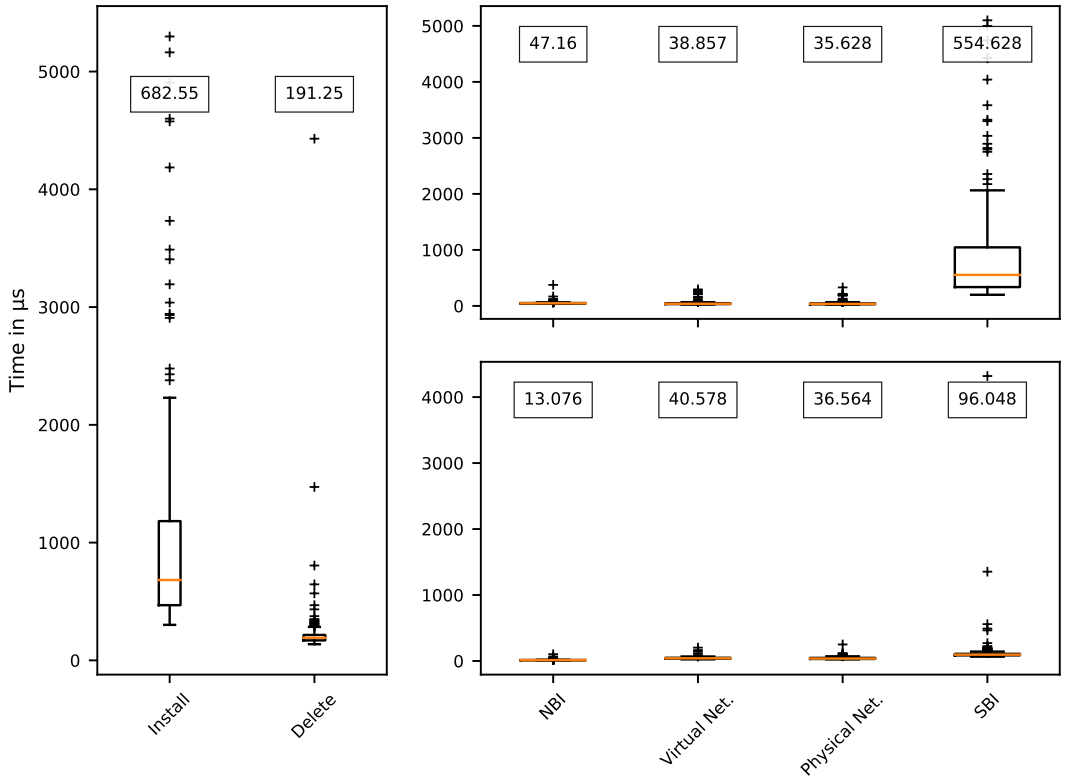


Figure 4.14.: Total delay introduced by the OVC for installing and deleting a lightpath as well as a break down to the four layers.

the request and triggers the right actors needed for the deletion. The tear down starts by deactivating the channels first. Therefore, the channel actors are responsible for the first outgoing message, which concludes the measurement. The processing at this layer takes 96.048 µs. The big difference, compared to the installation, results from the already available actors and objects, which do not need to be initialized. This heavily contributes to the shorter overall run time.

Currently, one typical way of achieving either an installation or a deletion of a lightpath is using a Network Management System. We ignore the option of configuring the devices manually, which is done for simple point-to-point connectivity, because it is even more cumbersome. With an NMS, an operator needs to select the proper endpoints, choose the connection parameters and trigger the setup. In more detail, he selects the devices, the modules and the ports that represent the endpoints of the lightpath. The correct transport protocol settings are one example for connection parameters. Finally, the setup can be triggered. From this description it can be seen that depending on the network size and the experience of the operator, this task takes at least tens of seconds. If the lightpath needs to be ordered first, the maximum setup times are even higher

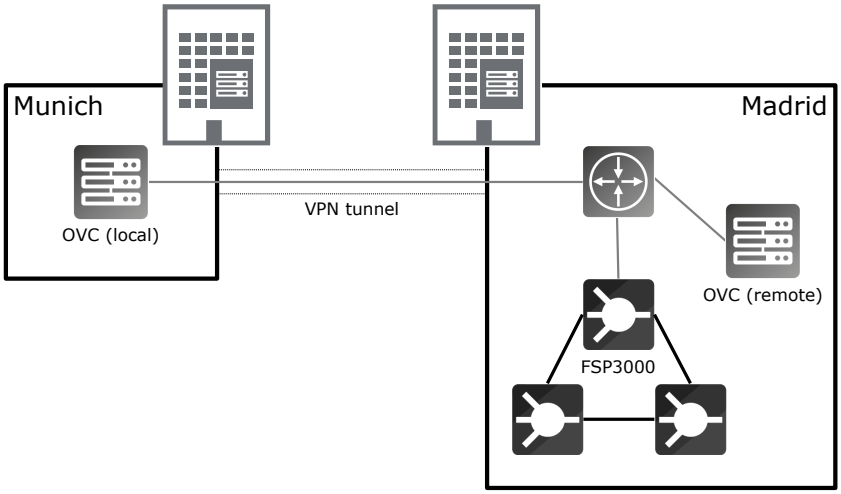


Figure 4.15.: Total delay introduced by the OVC for installing and deleting a lightpath as well as a break down to the different layers.

[Faw+04]. Giving the control to the user and automating the setup and tear down avoids this bottleneck. The user selects the endpoints from a smaller set of assigned endpoints and sends his request to the operator. If custom values are not specifically allowed and requested by the client, default values defined by the operator can be used for the connection parameters. This automates the setup procedure and improves the request times significantly. The time needed is below 1 ms, which is much faster than seconds. Taking into account the time it takes to set up or tear down the connection itself, the delay introduced by the OVC can be safely neglected.

4.3.3. Distribution of OVC Components

The OVC offers the flexibility of being deployed on multiple server machines including remote deployments interconnected by a network. This property is enabled by akka and the actor model. Due to the message-based exchange, it is possible to deploy individual actors at any JVM independent of its location. In addition, akka provides all features needed in order to communicate through a network. Using remote locations for deploying OVC components remotely can be done from the start of an actor's lifetime or later on by means of a migration.
In order to evaluate a distributed deployment, a distributed testbed is used. It comprises an optical network and a virtual server in Madrid as well as one machine in Munich (see Fig. 4.15). Both locations are connected through a VPN tunnel. The machine in Munich (local) is running the main parts of the OVC, while the server in Madrid (remote) is used for remotely deploying actors. The local machine is able to access the optical equipment through the tunnel. The round-trip time between the two locations was measured to be roughly 45 ms based on 500 pings minus 12 lost packets. The machine in Munich is the previously described computer used for the previous measurements in this section.

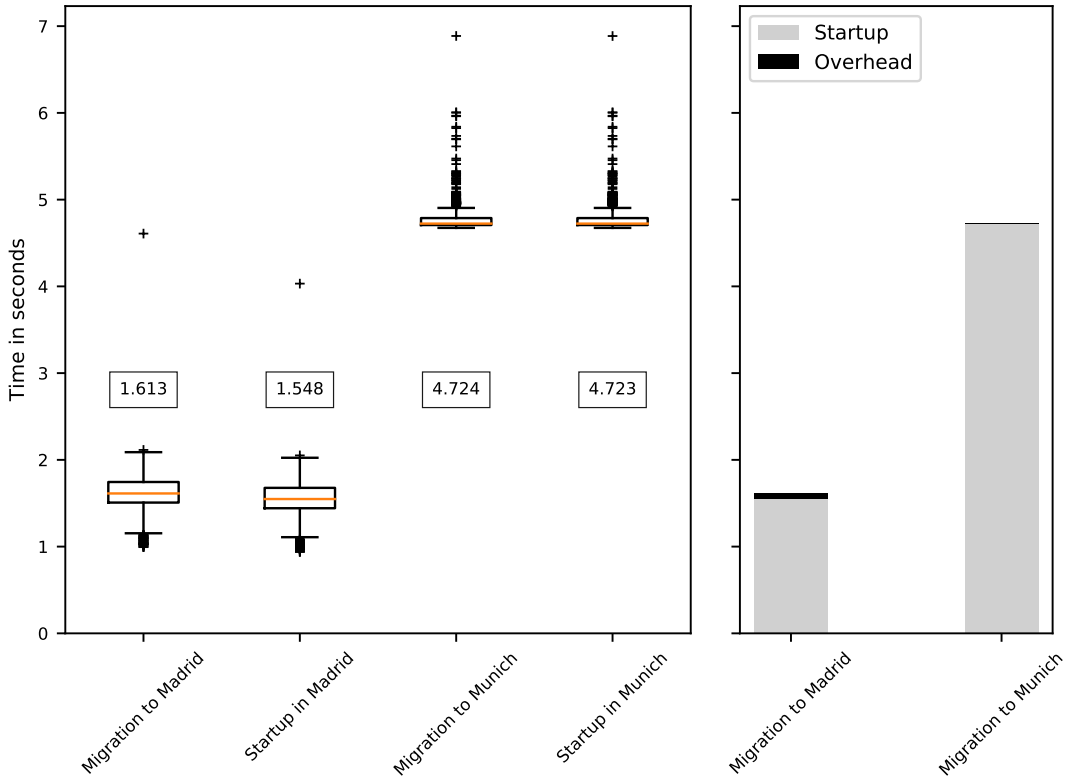


Figure 4.16.: Comparison of migration and startup for a migration to Munich and Madrid and the relationship between the startup and the migration overhead.

A virtual machine is used in Madrid equipped with 2 cores of an Intel Xeon CPU E5-2660 @ 2.20 GHz and 2 GB of RAM. It used Ubuntu 16.04.4 LTS 32-Bit for the OS. Two scenarios are considered in this setup: a migration of an SBI actor and a remote deployment of all SBI actors.

Migration

A migration procedure can be used to migrate single actors or parts of an actor hierarchy to a remote location. Results for migrations of an individual SBI network element in the distributed testbed are shown in Fig. 4.16. On the left-hand side, the startup time and the migration time are shown for each direction, i.e., Munich to Madrid and vice versa. The startup time includes the initialization of the actor as well as a full update of the relevant inventory information. In addition, the full migration time contains the overhead of creating the actor and transferring data, if needed, for a remote instantiation. From Munich to Madrid, the migration time (median) is 1.613 s. 1.548 s of the migration time is used by the startup time. This means that about 65 ms are overhead of the

migration. The opposite direction, i.e., from Madrid to Munich, results in higher values. The full migration takes 4.724 s, while 4.723 s are just part of the startup. Therefore, the migration overhead toward the main machine is only about 1 ms. This ratio is summarized for both directions on the right-hand side of Fig. 4.16. To explain the difference in startup time, we need to take a closer look at the startup procedure. The main task of the actors is to update their knowledge of the managed devices. This means, discovering the configuration, updating the status and creating new actors for subcomponents. This task requires communication with the device. Being in Madrid and thereby close to the devices improves the communication delay and only an abstracted view of the device needs to be sent on to the main machine in Munich. This results in a time improvement by a factor of almost three. The Overhead on the other hand is higher for a migration to a remote location. The reason is that information needed for the actor's instantiation needs to be exchanged between the locations. For a local creation of an actor, the overhead is minimal.
There is not only an improvement in time but also in the amount of data that needs to be exchanged between both locations. For one exemplary device, the data needed to be transferred between Munich and Madrid is 75 kB for the local deployment and 30 kB for the remote one. The migration saves 60 % of the data. Since an update is not only required in the beginning but also to synchronize the state throughout the lifetime, a migration is beneficial for such equipment deployments.

Distributed Deployment

Next, the influence of deploying parts of the SBI in a remote location is evaluated. In particular, the effect on the installation and tear down time of a lightpath is examined. The same setup as shown in Fig. 4.15 is used. The only difference to the previous measurement is that all SBI network elements are deployed at their final location from the start. Three scenarios are considered: a centralized deployment of all components (in Munich), all SBI network element actors are deployed remotely (in Madrid) and finally the tunnel actor as well as the SBI network element actors are deployed remotely. 1000 runs have been conducted per scenario and operation, i.e., installing and tearing down a lightpath. The time is measured starting with an incoming request at the NBI and it is stopped when the NBI receives the confirmation that the task was successfully completed. The collected results for a lightpath with a single hop are shown in Fig. 4.17.
The results for installing a lightpath are shown on the left-hand side of Fig. 4.17. For a centralized setup, we can see that it takes 9.486 s (median) to install a lightpath. If all three SBI element actors are moved to the remote location, the time is reduced to 9.253 s. Moving also the actor responsible for the tunnel to Madrid reduces the time even further to 9.161 s. By deploying all actors related to the network elements and the tunnel remotely, savings of more than 300 ms can be achieved. However, most of the time spent for the lightpath setup is actually related to physical properties of the optical network, e.g., power equalization.
Numbers for deleting a lightpath are shown on the right-hand side of Fig. 4.17. For a centralized setup, forming the baseline, the deletion takes 7.594 s. In general, the time it

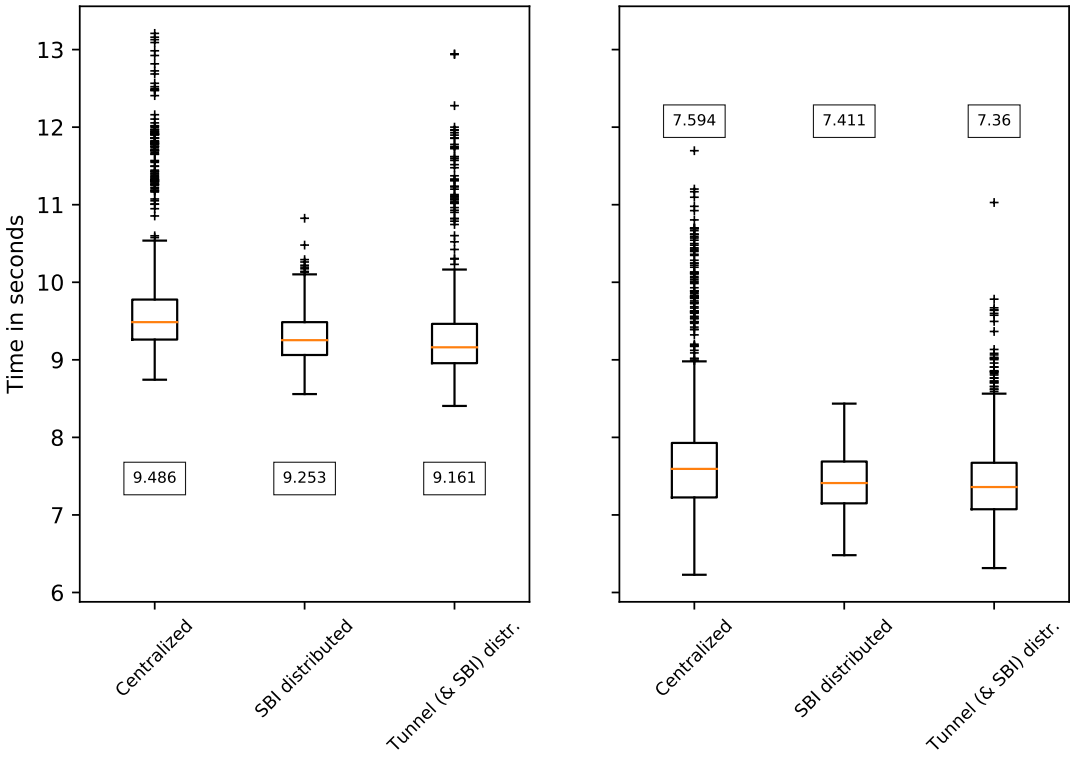


Figure 4.17.: Timing results for different deployments when installing (left-hand side) and tearing down (right-hand side) a lightpath.

takes to tear down a lightpath is lower than the time it takes to establish it. Moving the SBI element actors to the remote location, the time is reduced to 7.411 s. If the tunnel actor is moved too, the time is further reduced to 7.36 s. This means that more than 200 ms can be saved by moving these components to the remote location. Again, most of the time is related to the characteristics of the optical equipment, e.g., deactivating the tunnel and powering down the transceivers. It has to be mentioned that the equipment is not optimized for fast power-on and power-down.

One observation in both scenarios is that the measurement for a distributed SBI contains fewer outliers than the other two. Since the VPN passes through the Internet and the lab in Madrid is also used for other activities, it is hard to predict the best time for the measurements, which need to run for an extended amount of time. It is likely that the time chosen for this particular measurement, i.e., Sunday night, was less busy. Nevertheless, the measurements are still representative because it appears more likely to experience interference during operations. The difference in traffic is additionally mitigated by the median value, which is less sensitive to outliers than a mean value.

Another remark is that many of these steps require a polling to check if a configuration change was successful. The reason for this is that so far, only the REST interface is

used, which does not provide a way of pushing information to the OVC in its current implementation. Therefore, the polling interval was set to 100 ms. This can be improved with technologies that are able to push data like WebSockets or Server-Sent Events. Alternatively, registering for existing SNMP traps could be another solution.
Most of the savings result from the fact that the connection between the sites has to be used less often. All accesses that no longer require passing through the VPN tunnel are improved regarding the introduced delay. This also means for locations with a higher delay that the savings will also increase, but that is not the only benefit. Due to the deployment on different machines, a horizontal scaling can be achieved. Even though, further investigation on the deployment of other layers or actors is needed. Some actors are especially critical in the sense that they store the state of the infrastructure, i.e., physical and virtual. They represent bottlenecks and probably need to be scaled differently. This will require synchronization between actors that share a common task or duplicate data.

Summary

The OVC is one approach to expose control to the user and provide virtualization capabilities to the operator. This chapter validated the mediator approach and applying actors to achieve performance and flexibility. It has been verified that the algorithmic extensions to PLL are applicable to the optical domain and can be easily scaled to larger networks with further optimizations. The delay introduced by the virtualization layer has been found to be negligible, especially taking into account the optical layer. Finally, it has been shown that different deployment options for the components of the OVC are beneficial for scenarios with distant locations. Having proved the standalone viability of the approach, the next step is to include it in a larger context.

5. Including the Optical Layer in Multilayer Networks

Multilayer in the context of this thesis refers to multiple network layers of the OSI model. In particular, multilayer is often used interchangeably with IP-over-optical, which are IP connections utilizing an underlying optical network. This can be implemented, e.g., by IP routers that are connected to transponders via gray interfaces. If every layer is operated individually, then the optical layer offers connectivity between selected endpoints based on the (maximum) bandwidth without considering other service requirements. On the other hand, the IP layer treats the optical connection as a static big pipe between Layer 3 (L3) devices. This restricted view does not allow for optimization techniques that need to be aware of both layers at the same time. Better decisions and more efficient operations are only possible if an overall view is embraced.
This chapter shows how SDN-enabled optical networks can be included in multilayer environments and discusses the benefits of joint control. First, multilayer networks are introduced and a selection of application areas is presented. Then, two scenarios are discussed: a data-center automation and the automatic creation of secure services. The first showcases the migration of legacy equipment to the SDN world, while the second explores unified control and new opportunities for application-oriented networking. Both scenarios highlight advantages of joint multilayer operations and each is validated by a proof of concept with commercial hardware.

5.1. Multilayer Networks

In general, multilayer networks span multiple layers between L0 and L3. A common case for optical networks is IP-over-optical. Currently, network planning in this kind of networks does not jointly consider both layers. The upper layer is just used to provide information on the demands that need to be satisfied by the lower layer. Even though there are practical ways for including multilayer information in planning [Aut16], typically a two-step process is applied. This process handles the IP layer first and performs an independent step for the optical layer afterwards [Lop+17a]. Originally, the capacity required by applications was magnitudes smaller than the granularity provided by the optical network. Therefore, it was — and often still is — necessary to groom multiple IP connections into a single optical connection. Because of the ever-increasing bandwidth requirements of applications and the advent of EONs introducing finer granularity, multilayer planning is of growing interest, especially in combination with application awareness [Lop+17a]. Without considering the IP and optical layer in conjunction, the network operator misses out on Capital Expenditure (CAPEX) savings resulting from

multilayer optimizations, especially for EONs in the optical layer [Kle12]. Further savings can be achieved through the inclusion of the control plane in planning (including protection) [IGL13a]. This idea can be extended to an integrated management and control architecture based on a PCE [IGL13b]. An evaluation of architectures for planning a protection, comparing a single layer approach to a multilayer procedure, indicates significant CAPEX savings when considering both layers. Finally, managing multiple layers also influences the IP routing that needs to consider costs for taking an optical lightpath, e.g., hop- or distance-based [Pal+14].
Multilayer resilience — covering protection and restoration — is another prominent showcase for the importance of interaction between layers [Sch12]. Protection relies on resources that are reserved beforehand. They are either shared between multiple connections or dedicated to one connection. In contrast, restoration tries to find and instantiate a solution after a failure occurred. It can only use resources that are free at the time of the failure. For this reason, protection provides faster recovery times than restoration. In general, there are different approaches for recovery in IP-over-optical networks [Pic+06]: (I) single-layer recovery, (II) static multilayer recovery and (III) dynamic multilayer recovery. The first case (I) considers only a single layer, e.g., the bottom or the top layer. This may lead to recovery problems or increased complexity. If multiple layers are considered, an uncoordinated or a sequential approach can be applied. The first one allows every layer to apply countermeasures on its own, while the second one is escalating either from bottom to top or the other way around. An uncoordinated recovery, triggered at both the optical and the IP layer, may lead to slow convergence [Sim14]. A coordinated approach can be used to introduce a fixed structure, e.g., from bottom to top. Then, only if the first layer fails to recover, the next one is triggered. Multilayer recovery not only needs to coordinate the individual steps but also to assign resources. In the static case (II), resources are preassigned to every connection and activated in case of failure to maintain the original topology. The dynamic techniques (III) allow for the setup of new connections by using the control plane of the underlying network. Applying multilayer recovery leads to a more efficient recovery with regard to time and resources. These benefits can extend beyond the network itself. By increasing the allowed maximum time to repair through multilayer restoration, the number of service teams needed for repairs can be reduced, which leads to savings in Operating Expenditure (OPEX) [Cru+14].
Multilayer interaction is also important for the implementation of application-centric networking [GLS15]. In-operation optimizations are improved by including application awareness compared to current bandwidth-driven approaches [Lop+17a]. Applications not only rely on a particular bandwidth but may also have other service requirements, such as latency. Service requirements define which properties need to be met by the network in order to satisfy the request. Based on this, an application-aware framework for a dynamic multilayer resource allocation and optimization can be defined calculating tailored solutions for the requested service requirements [Lop+17a]. In this context, application-aware planning in multilayer networks comprises multilayer planning, optical resilience and multilayer reroute [Lop+17a]. These building blocks can be applied incrementally in order to maximize savings.

Most benefits can only be utilized if EONs become available and foremost optical networks are integrated into the overall SDN control framework. The development and deployment of new technologies is out of the scope of this thesis. The SDN integration, on the other hand, is a central topic. To validate the migration toward SDONs and show opportunities that arise from multilayer operations, two scenarios are discussed in detail next.

5.2. Data-Center Automation

This section summarizes the first SDN-based orchestration of data-center and optical network resources over field-installed optical fiber using exclusively Open-Source Software (OSS). It presents an intermediate step on the way to SDONs. The proof of concept demonstrates a migration path for legacy equipment, in which a mediation layer, called the Optical Network Controller (ONC), provides an OF interface to an SDN controller. The validation focuses on the orchestration of elastic data-center and inter-data-center transport network resources. For this scenario, OpenStack and OF are used to control commercially available equipment and to allow network automation. Programmatic control enables a data-center operator to dynamically request optical lightpaths from a transport network provider to accommodate rapid changes of inter-data-center workflows, such as the migration of virtual machine clusters. The results indicate that provisioning times are reduced to tens of seconds compared to typical provisioning times of days or weeks, even though OF is not a natural option for optical networks. This section is based on results published in [Szy+13] and [Szy+14b].

5.2.1. Data-Center Use Case

Currently, optical transport networks are statically configured and provide customers with dedicated, fixed bandwidth connections for extended time durations (years). Data-center operators are well-versed in exploiting dynamic compute and storage resources through virtualization. They anticipate the same flexibility from their data-center network. Network operators appreciate an opportunity to utilize automation and would like to apply virtualization techniques to provide customers with dedicated, fixed bandwidth connections for shorter time periods. A prerequisite is a dynamic, reconfigurable optical transport network that permits the release of idle statically configured connections in order to redistribute bandwidth, flatten peak network load and increase network utilization. Clearly, a mechanism to orchestrate cloud resources within and optical transport resources between data centers is required [EA13; GBX13; Mcd13].
OpenStack [Ope18g] is the leading open-source data-center orchestration platform used within data centers operated by cloud service providers and large enterprises alike. It comprises an expanding collection of independent service modules that include the management and dynamic orchestration of virtualized compute (nova), storage (cinder & swift) and networking (neutron) resources hosted on hardware servers. Neutron was conceived to automate virtualized Layer 2 (L2) & L3 connectivity services between Virtual Machines (VMs) and the underlying physical servers within a data center. Neutron

plug-ins provide various options for the control of the network, e.g., through an SDN controller. Indeed, publications at that time have reported experiments using OpenStack with OF-enabled L2 switches where the Wide Area Network (WAN) between data center locations consisted of fixed optical connections [Miz+12] or reported the orchestration of data-center and optical-transport resources, but were based on proprietary software and confined to a laboratory environment [Zha+13a].
Typical workflows inside data centers include storage migration, virtual machine migration, active-active storage replication and distributed applications [ADH12]. Storage migration involves either the backup or transfer of data for future usage, e.g., to reduce access time. VMs with live workflows are typically migrated because of the need for more computational resources that can be met elsewhere. Active-active storage replication ensures the synchronization and coherence of replicated data across different locations. Finally, distributed applications require communication between virtual machines hosted on physically separate, dispersed hardware servers. These workflows highlight the importance of having a reliable and elastic connectivity between data centers deployed across multiple geographical locations. This interconnection is most often provided by an infrastructure provider. To maximize the utilization of the network, the infrastructure provider might offer virtual slices to his customers, e.g., data-center operators, who can increase and decrease the bandwidth or redirect the communication channel to another data center.
End-to-end workflow automation offers many benefits. It reduces the lead time for tenants and data-center operators to access innovative services across a geographically distributed pool of resources. Automation also removes the barriers imposed by statically configured optical transport to better match the usage of all available resources. On the one hand, a data-center operator could accommodate variable bandwidth demands and temporary increases within workflows by closely tracking the aggregated requirements of his tenants using flexible bandwidth. On the other hand, he could be provided access to a fixed quantity of bandwidth that can be moved between his demarcation points based on workload. The data-center operator is then able to control his network through an SDN controller to maintain the packet forwarding state. Currently, the optical network requires an Optical Network Controller (ONC), which is a mediation layer, to establish lightpaths between data centers. The ONC exposes an SDN interface and effectively acts as a domain controller and a hypervisor for the optical network. This approach exposes an appropriate level of control over the resources northbound to a data-center orchestrator. To evaluate the benefits of SDN-controlled automation, a demonstrator is introduced.

5.2.2. Demonstrator Setup

The setup focuses on the interaction between the data-center orchestrator and the transport network. The ONC presents the optical ring topology, consisting of three ROADMs, as one (virtual) switch to the Floodlight OF controller. By applying port-based flow modifications (flow mods) to this virtual switch, a lightpath setup or teardown is triggered in the optical network. This enables OpenStack to create lightpaths on demand

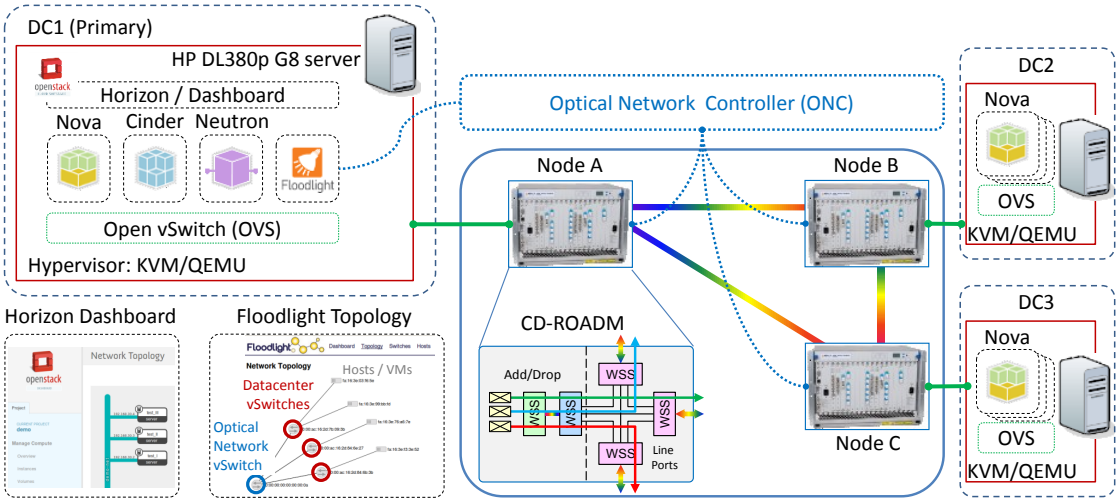


Figure 5.1.: Hardware and software components of the demonstrator.

Table 5.1.: Optical paths between the three data centers chosen by the control plane.

Requested connectivity	Optical path assignment
DC1 to DC2	Node A ↔ Node B
DC1 to DC3	Node A ↔ Node B ↔ Node C
DC2 to DC3	Node B ↔ Node A ↔ Node C

through an interface offered by Floodlight. A number of subsequent steps in the optical layer are handled by the ONC and hidden from the orchestrator.

Fig. 5.1 outlines the main components of the demonstrator. Three discrete data centers (DC1, DC2 & DC3) were emulated with HP DL380p G8 servers. All three servers ran Open vSwitch (OVS) 1.4.0 and the nova compute component, which provides the ability to instantiate multiple virtual machines. The server representing DC1 was nominated as the primary server that in addition hosted the OpenStack cloud management application and the Floodlight OF controller.

The inter-data-center network comprised three ADVA FSP3000 colorless ROADM nodes interconnected to form a ring using field-installed, ducted optical fiber within BT's network. A 10GbE client-side port of each ROADM was connected to a GbE Network Interface Card (NIC) in the server of the data center via an ADVA FSP150CC-XG210 demarcation/aggregation device (not shown in the figure). Two of the ROADMs were directionless (Node A and Node C) and one node was directed (Node B). Tab. 5.1 shows the optical paths chosen by the control plane for this setup.

The measurements were conducted with the configuration described in [Szy+14b]. The installed OS was Ubuntu server 12.04 LTS (3.5.0-41-generic kernel) due to the more stable and extensive driver support compared to release 13.04, which was reported in [Szy+13] before. OpenStack 2013.1.3 (Grizzly release) from the Ubuntu Cloud Archive

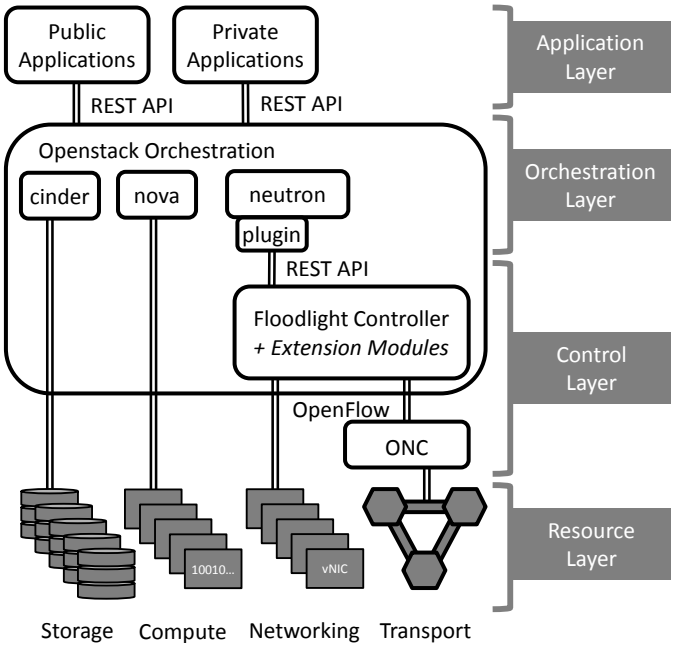


Figure 5.2.: Layers and logical components of the demonstrator.

was used. QEMU/KVM was deployed on each server as the virtualization environment together with OVS 1.4.0, enabling the creation and interconnection of the VMs created by OpenStack. An out-of-band management network was used for controlling the OpenStack cloud. A Dynamic Host Configuration Protocol (DHCP) relay was installed at DC2 and DC3 to distribute IP addresses via this management network when no optical path was available through the data network.

Logical Components

Fig. 5.2 summarizes four layers with the main logical components present in the demonstrator. From top to bottom, the *Application Layer*, which includes the OpenStack Horizon dashboard, uses a REST API to invoke services through OpenStack. The *Orchestration Layer*, which offers an NBI to applications, provides access to storage and compute pools on the servers via OpenStack cinder and nova, respectively. The *Control Layer* comprises the Floodlight controller and the ONC. Floodlight is accessed through a REST API via the Virtual Network Filter that acts as the network backend for OpenStack's neutron plugin. Additional extension modules that are incorporated into Floodlight access ADVA's ONC. The ONC acts as an OF virtual switch, which, in turn, communicates with each FSP3000 node via SNMP. The *Resource Layer* comprises four resource pools: compute, storage, networking (emulated by virtual NICs and switches) and optical transport. The Floodlight controller discovers the devices and inventory automatically via Link Layer Discovery Protocol (LLDP) for OVSs and the attached VM instances running on the server. Forwarding-plane connectivity of the FSP3000

nodes was advertised via GMPLS and polled (using SNMP) by the ONC. A Link Inserter application provided the static connectivity mapping (patch cabling) between the client ports of the FSP3000 nodes and the L2 network elements via a REST API or configuration file. At the same time, it advertised the connectivity to Floodlight. The compute and storage inventory pools were accessible via the default OpenStack Horizon dashboard.

Steps of the Measurement Procedure

The measurements for a flexible bandwidth assignment recorded the time needed for the setup of a lightpath. Before each measurement, all preexisting connections were torn down. The measurements started with the creation of a flow using the REST interface. A traffic source — one VM — continuously sent 100 User Datagram Protocol (UDP) packets per second. As soon as the first packet arrived at the receiver, the measurements concluded and the elapsed time was captured. The reception of the first UDP packet verifies the instantiation of the lightpath and indicates that connectivity has been established at the transport layer. This test was repeated 100 times to obtain the average setup time.
To measure a steering of the available bandwidth, which corresponds to changing one endpoint of a lightpath, two traffic sources and one destination (receiver) were chosen. The receiver triggered a redirection of the lightpath. Each measurement comprised three components: 1. the deletion of any existing connection, 2. the creation of a lightpath between the new source and the destination and 3. finally the waiting time for the first UDP packet to be received to confirm the data flow. This was followed by a measurement of the switch over in the opposite direction. The senders followed the same description and packet creation rate as above. Again, each measurement was repeated 100 times.

5.2.3. Measurements of Data-Center Workflows

The measurements are divided into three sections: I. flexible bandwidth II. steered bandwidth and III. storage migration.
The first set of measurements examined the time required for flexible bandwidth (I.). Due to the limitations of the software and hardware back then, it was simply modeled as a lightpath setup. Measurements commenced with a request issued by the user for a new bidirectional connection between two data centers, e.g., DC1 $\leftrightarrow$ DC2. An optical connection was established following this request and the time was stopped when the first packet arrived at the receiver's side. The setup time mostly depended on the number of hops along the route that was chosen for the lightpath by the control plane and included all the intermediate steps following the request. The frequency distribution of the recorded time values is shown on the left-hand side of Fig. 5.3. For DC1 $\leftrightarrow$ DC2 the direct one-hop path was chosen according to Tab. 5.1. 15.8 s were measured on average for the connection setup. The optical path for DC1 $\leftrightarrow$ DC3 and DC2 $\leftrightarrow$ DC3 included an additional optical hop, i.e., two hops in total. This increased the mean setup time to 24.3 s and 25.2 s respectively. The additional hop imposed a penalty of roughly 8 s to

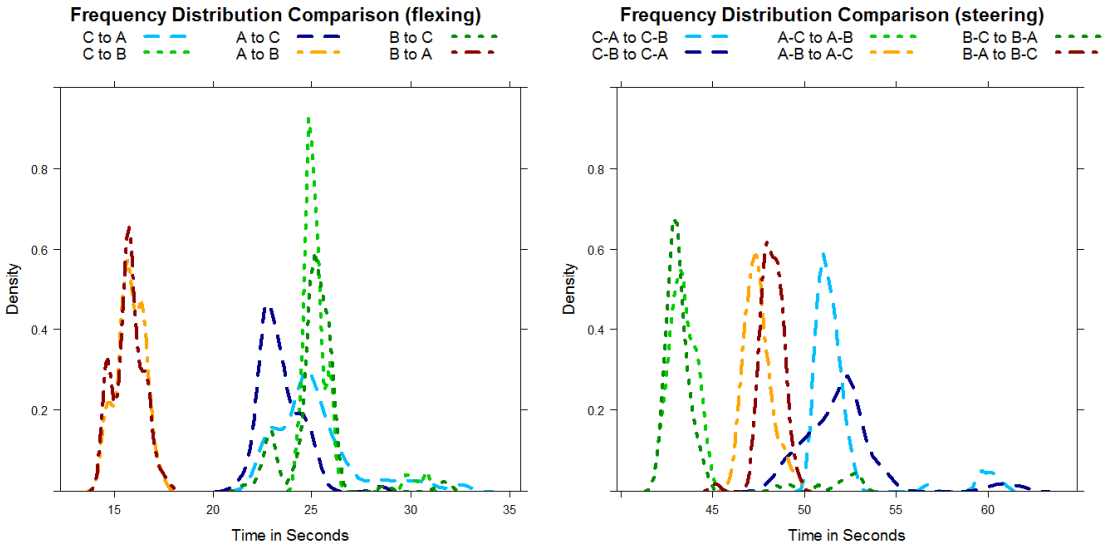


Figure 5.3.: Frequency distribution for bandwidth flexing (left-hand side) and steering (right-hand side).

10 s. The differences in the group of two-hop cases can be traced back to different fiber and hardware properties on the two routes.

The second set of measurements evaluated the steering of connections (II.) in order to reroute a connection and reuse resources. Measurements commenced with the request for a teardown of the existing connection, immediately followed by a request to establish a lightpath to the other data center while keeping one endpoint fixed. Each run ended with the reception of the first incoming packet from the remote data center. Fig. 5.3 (right-hand side) illustrates that the results can be grouped into three regions, each one containing two peaked distributions. The regions are discussed from left to right. Region 1 (green curves) was centered around 44 s and exhibits the fastest toggle times. The reason is that a one-hop optical path can be set up more rapidly than a two-hop path. Region 2 (brown curves) was centered at 48 s. The main difference here was that a one-hop optical path had to be torn down, while a new optical path with two hops was created. This led to slightly slower setup times, which, in turn, caused an additional delay of 4 s compared to region 1. Region 3 (blue curves) covers a two-hop teardown and a two-hop setup. It resulted in the slowest times and accounted for a mean of 52 s.

At last, a data transfer between two data centers, representing a storage migration (III.), was investigated. The identified steps for simulating this transfer are visualized by the flowchart on the left-hand side in Fig. 5.4. With the source VM running in DC1, the actual workflow started with the creation of a second VM in DC2. This was followed by a request for a lightpath. 100 GiB of data were transferred using the Linux shell tool `nc`. The lightpath was released upon conclusion of the data transfer and the source VM was terminated afterwards. A `wireshark` representation of selected packets captured with `tcpdump` throughout the measurement are shown on the right-hand side of Fig. 5.4.

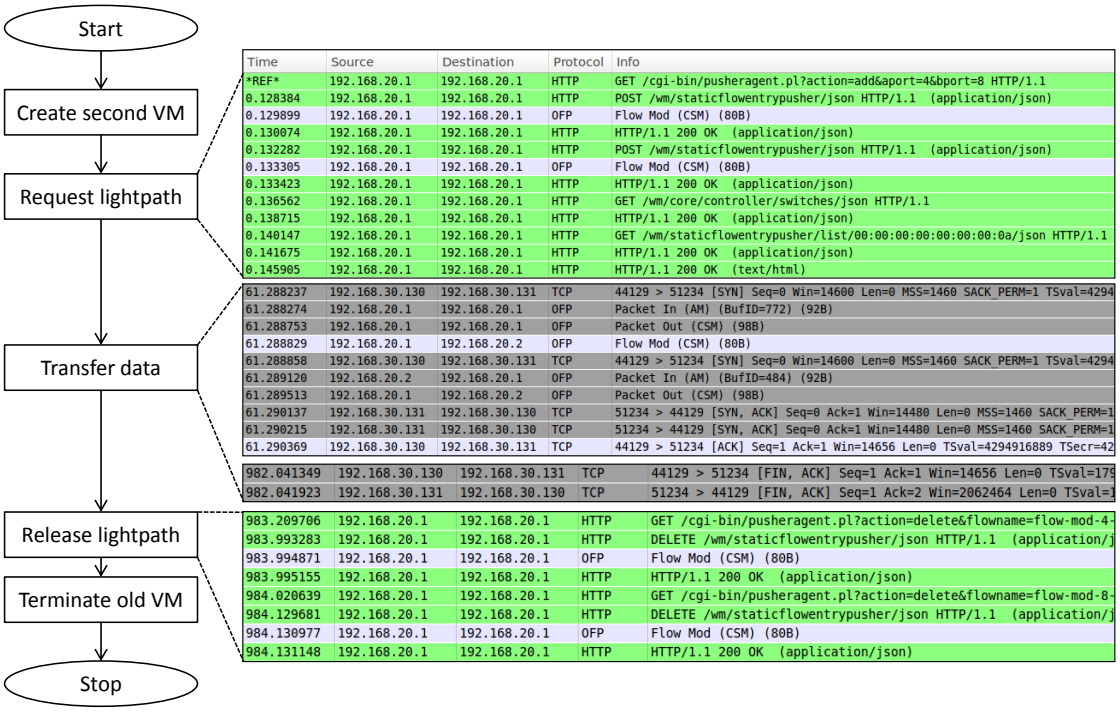


Figure 5.4.: Flowchart for data transfer and `wireshark` trace.

The first box summarizes the setup of the lightpath. The user interface (`pusherAgent`) is called via an HTTP-request and its timestamp is taken as reference point for the subsequent packets. It triggers Floodlight through the `staticflowentrypusher` to send the flow mod to the optical virtual switch. The message is confirmed in line 4. This process is repeated because a lightpath corresponds to flows in both directions and the second request ends in line 7. The user interface updates its information in the next lines until the initial request is confirmed by the last HTTP-message. The second box shows the start of the data transmission. It starts with the Transmission Control Protocol (TCP) `[SYN]` request to initialize the transfer. This leads to a packet-in at the first OVS and is answered by the controller with a packet-out. Additionally, a flow mod is sent to the OVS at the second node. The packet reaches the second node in the next lines and triggers another packet-in and packet-out. Then the TCP connection is confirmed by the other side with a `[SYN,ACK]`. The transmission starts with the following `[ACK]` message. The end of the transmission is captured in the third box. The transmission in this exemplary recording took 921 s and the TCP connection is torn down at the end by `[FIN,ACK]`. After this, the user interface is used once again for deleting the flows and thereby tearing down the lightpath. The steps are shown in the fourth box. There are two calls to the `pusherAgent` interface, one for every direction. The script sends a delete request to Floodlight's `staticflowentrypusher`, which results in a flow mod in order to delete the flow. The measurements revealed that 16 min 24 s were needed to complete the data migration. As expected, a high proportion (approximately 93 %) of the elapsed

time, namely 15 min 21 s, was spent exclusively on the data transfer, limited only by the GbE NICs used in the experiments.

Assessment

The emulated data migration shows that for transferring large amounts of data, it can be very attractive to acquire additional bandwidth for a limited time. The improvement will be more pronounced if higher-performance NICs with greater data throughput are used. Additionally, with the possibility of flexible and steerable bandwidth, a reduced amount of equipment can be used more efficiently and the utilization of underused fiber resources can be increased. In the case of OpenStack, a solution for including SDN-controlled optical networks, e.g., for VM migration, is essential. It leads to fast and flexible lightpath setups, enabling the dynamic usage of the available bandwidth capacity. With high bandwidth-on-demand services new revenue streams become possible for data-center operators. Nevertheless, this was an early demonstration and the used OF protocol has major drawbacks, if applied to OCS. Flows need to be set up (and removed) in both directions in order to satisfy the forwarding concept of OF controllers. The single switch representation is very abstracted and hides details about the chosen path. Even if the topology is extended to multiple switches, a correlation between single flows on different devices needs to be derived and optical constraints cannot be represented. Therefore, the next scenario demonstrates recent developments in the field of SDONs.

5.3. Automatic Intent-Based Secure Service Creation

Growing traffic demands and increasing security awareness are driving the need for secure services. Current solutions require manual configuration and deployment based on the customer's requirements. Applications need a way of describing requirements directed at an underlying network in a technology-agnostic manner. In future SDN-controlled IP-over-optical networks, service requirements should be considered to provision services tailored to the application's needs and to optimize network resource usage. This section presents the concept of an automatic selection of the appropriate encryption layer in a multilayer — IP, Ethernet and optical — network based on requirements that are expressed by an application through intents. Intents define the application's requirements, e.g., bandwidth, without specifying the actual realization. The goal is to move the decision complexity away from the application requesting the service toward the orchestrator, which also acts as an SDN controller for all devices. The controller is responsible for receiving an application's intent and translating it into network requirements, evaluating the related trade-offs and (technological) constraints, and eventually provisioning a secure service that can be used by the application. This concept is experimentally validated with an implementation using an open-source SDN controller, which is aware of the available layers, and commercial SDN-enabled hardware. This network equipment is controlled using an open interface, i.e., TAPI with extensions. The section is based on the results published in [Szy+16], [Cha+17] and [Szy+18].

5.3.1. Secure Services

Service providers used to deploy separate networks to host enterprise and end customer services. However, such an approach is not scalable in economic terms as the infrastructure is duplicated and underutilized. Current deployments are based on unified networks, which means that all traffic flows share the same infrastructure. Additionally, companies have to contend with higher risk and potential costs associated with data breaches of $3.6 million on average [IBM17]. As a result, it is critical for businesses to deploy solutions to secure their distributed and potentially virtualized network or cloud infrastructure. Encryption is a key component in this regard and responsible for ensuring the confidentiality of a communication between two trusted endpoints. Network encryption is one crucial element for data transfer over an untrusted public infrastructure. End-to-end encryption is flexible and independent of the underlying network infrastructure, making it relatively easy to deploy. One example is the widespread utilization of Hypertext Transfer Protocol Secure (HTTPS) for Internet traffic, mainly driven by the over-the-top providers. This flexibility comes at the cost of higher processing requirements on both — server and client — communication endpoints, which consequently induces more latency and reduces the throughput of the network.

Given the large number of communication protocols, a deployment of specialized mechanisms for each individual protocol is not feasible and in-flight encryption is used as a standard mechanism to secure these protocols. In-flight encryption is applied to traffic on one of the lower layers of the OSI model, i.e., physical (Layer 1 (L1)), data link (L2) and network layer (L3). Protocol solutions operating at these network layers (e.g., Internet Protocol Security (IPsec), Media Access Control Security (MACsec), physical layer) have inherent technical (e.g., cost of deployment, latency, effective throughput) and cost trade-offs. In-flight encryption assumes that protocols that do not support security mechanisms will be encapsulated into one of these protocols. When performing such tasks at the network level, requirements on endpoint capabilities and processing complexity are relaxed. Moreover, this allows the network operators to optimize the cost per bit for encrypting traffic between two remote sites. Network service providers typically deploy infrastructure with multiple, potentially vendor-specific, choices for in-flight encryption. Manually evaluating the technical, cost and security trade-offs between the possible solutions for an application, requesting a secure connectivity service from the infrastructure, poses a significant overhead. Furthermore, applications are interested in satisfying their own requirements, which commonly do not match the priorities of the network operator: while applications are concerned about bandwidth, latency, availability, security, etc., management systems are optimized to minimize the usage of resources, energy consumption and so on. In theory, applications should be able to clearly define their needs without getting into details on how the service is created, i.e., via an intent. The selection of the best encryption mechanism for the applications is a feature that the network must provide in order to optimize the application's experience, while providing the most cost-effective solutions on the operator's side. This is called a Secure Transmission as a Service [Lop+16b]. Throughout this section, the term "in-flight encryption" describes the process of applying a technology to encrypt traffic. The term "secure ser-

Table 5.2.: Overview of encryption layer properties.

Requirement	IPsec (L3)	MACsec (L2)	Physical (L1)
Latency	high	medium	low
Data Throughput	low	medium	line speed (no overhead)
Protocol Transparency	low	medium	high
Flexible Encrypted Payload Size	restricted	restricted	1G – 100G
End-to-End Compatibility	IP only	L2 only	OTN or SONET/SDH
Flexibility (Meshed)	high	low	medium

vice" is used in the context of a connection that is transferring traffic that needs to be secured by some kind of (in-flight) encryption mechanism.
Current research on security in SDN is focusing on controllers and the communication between the control and data plane [DK15; Sco15; SOS13]. Even though the data plane is also considered in isolation, the work is mostly limited to OF switches [KRV13]. Therefore, it is concerned with preserving the integrity of flow tables and flow rules [SOS13]. Eavesdropping is a general threat in computer networks and it applies to the electrical [SZ16] as well as the optical domain [Fur+14]. A natural and effective countermeasure is encryption. In OF networks, encryption of packets may result in matching issues because some headers are no longer accessible. This section proposes an architecture for enabling the intent-based creation of secure services, providing encryption, in multilayer environments.

5.3.2. Encryption mechanisms

The main purpose of an encryption mechanism is to protect the user data against eavesdropping and therefore ensure its confidentiality. The choice of the encryption mechanism depends on the requirements of the application. This section summarizes the general idea of in-flight encryption and compares various properties of physical layer encryption, MACsec and IPsec. They are summarized in Tab. 5.2. These properties are used by the orchestrator to assign an appropriate encryption mechanism to the requested secure service. The configuration of an encrypted connection is a multi-step process. First, the two endpoints need to be authenticated, which means identifying the other side as the expected communication partner by a pre-shared key, username/password based authentication or using certificates. The encryption mechanism needs two cryptographic primitives: a symmetric-key algorithm and a key exchange protocol. A symmetric-key algorithm is used to provide data confidentiality. It uses the same key for encryption and decryption. The most popular symmetric-key algorithm today is Advanced Encryption Standard (AES), which is also used in the experiments. A key exchange protocol, on the other hand, is based on public-key cryptography and is used to establish a symmetric session key over a public channel for the AES encrypted communication. Note that the symmetric key is regularly refreshed in order to limit the amount of data that is encrypted with the same key.

ADVA's physical-layer encryption is a point-to-point L1 encryption over the optical fiber. The hardware encrypts the Optical Channel Data Unit (ODU) payload before inserting it into the Optical Channel Transport Unit (OTU) frame, which is decrypted at the other side. Because the payload of an OTN frame is encrypted with no additional overhead, it offers a high protocol transparency. This approach provides the lowest latency and highest throughput, i.e., line speed, among the presented mechanisms. The bulk data encryption is typically done by a symmetric-key algorithm, such as AES, and the session key can be established by a key exchange protocol, such as authenticated Diffie-Hellman (DH) key exchange. The drawback of physical layer encryption is that it is bound to custom hardware at both endpoints of the infrastructure and the setup time roughly corresponds to a lightpath provisioning.
MACsec is a security protocol for Ethernet links (L2) and enables secure communication between neighboring nodes. Each packet is encrypted using symmetric key cryptography so that the communication cannot be monitored or altered while traversing the link. The symmetric key is established by a higher-level protocol that is part of Institute of Electrical and Electronics Engineers (IEEE) Std 802.1X-2010 [IEE10]. The MACsec data frame, defined by IEEE Std 802.1AE [IEE06], adds a security tag and an integrity check value to the Ethernet frame. Both are checked by the receiver to ensure that the data has not been compromised during transmission. Since MACsec is a simple protocol, it is able to achieve a high data throughput with low latency/overhead on Ethernet links. However, if data traverses multiple Ethernet links with enabled MACsec, then each L2 device must encrypt/decrypt the frame. This may cause performance degradation, compatibility problems and security issues, if an intermediate device is untrusted. Also, addressing/reachability limitations of L2 apply.
IPsec [SK05] is an end-to-end security protocol on the network layer (L3). Also, unlike others, the communication in IPsec typically uses multiple secure channels and has separate keys to communicate with different destinations. For these reasons, IPsec forms the basis of many VPN solutions. Services encrypted with IPsec are more independent from the infrastructure and flexibly deployable at many points of the network. In IPsec, the entire IP packet is encrypted and authenticated by the initiator and a new IP header is added to route the packet to its destination. Encrypted IP packets pass through the network unchanged until they reach their destination. Intermediate routers have no means of decrypting the packets. A major drawback of IPsec is its complexity and the introduced latency [RWW04]. It also significantly enlarges the size of the IP header, which causes network inefficiencies and adds penalties in the form of latency and reduced throughput to the overall solution cost.

5.3.3. System Architecture

The intent-based multilayer orchestrator was developed in the ACINO project and is an open-source effort built on top of ONOS [Ber+14]. Many extensions, with the goal of making it more application-centric, have been introduced to the original controller. The ACINO orchestrator's simplified high-level architecture is presented in Fig. 5.5. Starting from the top, Application Centric Intents (ACIs), issued by a client application,

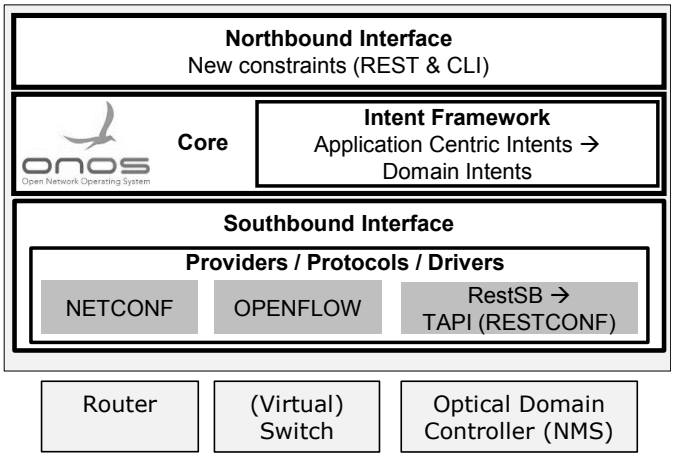


Figure 5.5.: Basic system architecture of the ACINO orchestrator.

are submitted through a REST NBI or a command-Line Interface (CLI). The ACI is an incarnation of ONOS' intents that supports additional constraints and needs to be processed by a specialized compiler. The orchestrator routes all requests through the intent framework, compiles the submitted ACI and selects the actions that need to be taken in order to satisfy the intent. Before returning the required actions, an intermediate representation is created: a domain intent. On a high level, they represent intents that a domain controller that is responsible for a part of the network can execute, in contrast to device-level intents. A more detailed explanation of this extension is given in App. C. Domain intents represent installable actions that need to be mapped to the appropriate transport representation. Then they are sent through southbound protocols to the devices that need to be configured. The devices themselves are either accessed directly or through a mediation layer like an intermediate controller, e.g., a NMS. Protocols for southbound interactions include OF, NETCONF and RESTCONF. The latter two define only the transport protocol and require YANG models for the description of the content, e.g., the ONF TAPI [Qia+16]. In any case, specialized drivers are implemented to handle the device-specific behavior that is not covered by the common protocol definition. For this reason, a TAPI driver was implemented in ONOS (App. C) using the REST southbound protocol, which is based on the REST proxy (App. C) and provides a basic set of methods for accessing web resources. The communication with the optical devices is done through ADVA's NMS, which exposes an experimental TAPI interface. The hardware side comprises optical equipment with physical-layer encryption capabilities — on a subset of the ports — as well as switches which are able to install encrypted tunnels, i.e., MACsec. Even though routers could be easily included, they were omitted because of missing support for IPsec in the available hardware. The extended explanation of changes that have been applied to ONOS in order to support encryption is available in App. C.

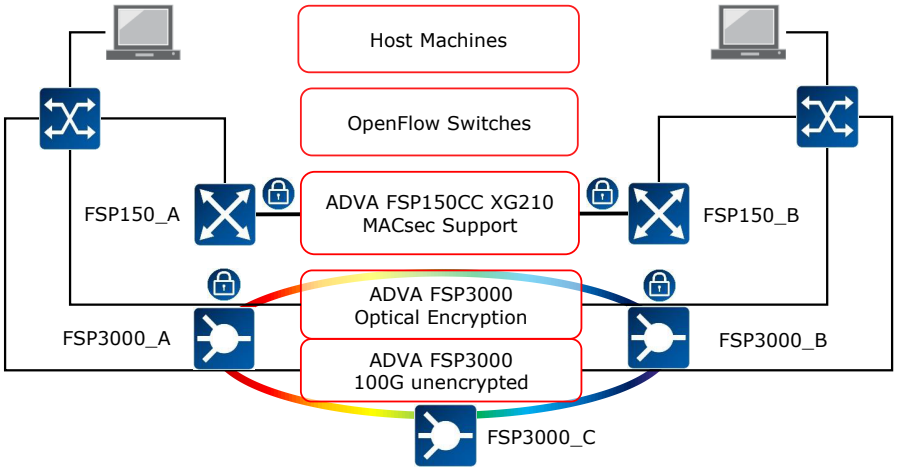


Figure 5.6.: Testbed setup in the lab based on commercial networking hardware.

5.3.4. Experimental Validation

As shown in Fig. 5.6, the presented system architecture was implemented and evaluated with commercial hardware in a lab. The testbed comprised two off-the-shelf computers, representing the hosts, and two servers. One server was running the ADVA NMS and the other the ACINO orchestrator. In addition, the two computers hosted an OVS instance each. These virtual OF switches were under ONOS' control and steered the traffic into the right direction depending on the encryption scheme. Both servers (not shown in Fig. 5.6) were connected to the optical equipment as well as the OVS instances through a management network. The testbed included an ADVA FSP3000 ROADM ring consisting of three nodes at the optical layer. Two of them were equipped with 10G AES cards, which encrypt all traffic on the physical layer. Also, two ADVA FSP150CC-XG210 were part of the setup. These Ethernet demarcation devices are capable of encrypting traffic using MACsec and were connected directly to each other. The hardware equipment is controlled and configured by the NMS. The secret keys for the encrypted connections were preconfigured by the administrator. For the unencrypted connection two 100G multiplexer cards were used. We evaluated three scenarios, of which two requested a secure service and one needed an unencrypted connection. For the first two, the best suited layer for the encryption was chosen automatically by the orchestrator based on the required bandwidth.

Now a workflow for installing an ACI is given to explain the process in more detail. The ACI presented in Fig. 5.7 is submitted through the NBI. It defines the requirements of the application. This intent comprises three constraints: 1. The `DomainConstraint` activates the creation of domain intents. 2. The `EncryptionConstraint` indicates that the service needs to be encrypted. 3. The `BandwidthConstraint` specifies the required bandwidth in bit/s. The endpoints are defined by `one` and `two` and the L2 addresses belong to the two computers. This intent is handed over to a specialized compiler that is able to handle ACIs. Since in the case of the testbed devices are controlled through

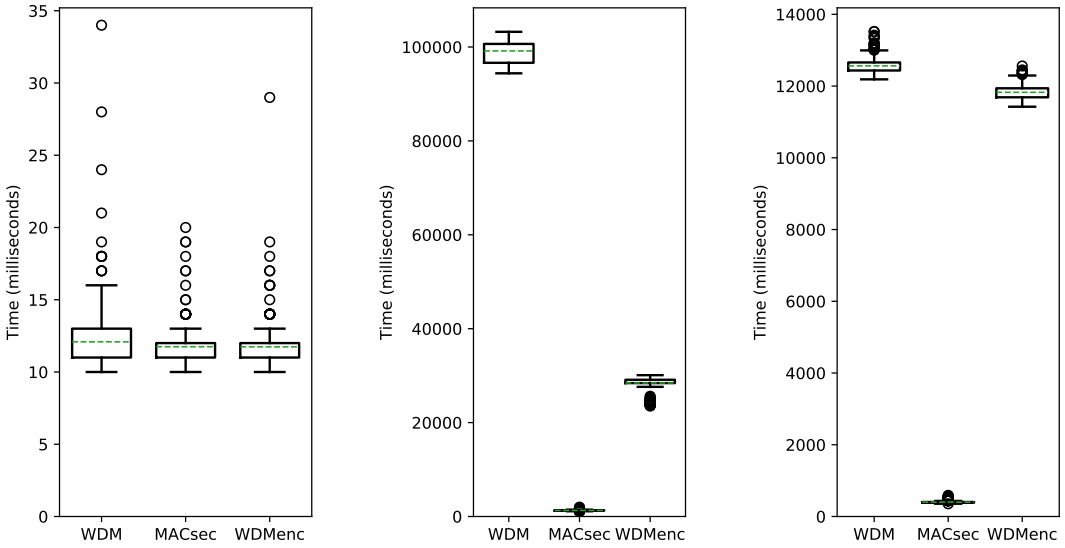
```
{
  "type": "AciIntent",
  "appId": "org.onosproject.cli",
  "priority": 100,
  "constraints": [
    {"type": "DomainConstraint"},
    {"type": "EncryptionConstraint"},
    {
      "type": "BandwidthConstraint",
      "bandwidth": 10000000
    }
  ],
  "one": "7E:1D:D7:77:7E:06/-1",
  "two": "CA:B8:53:D4:2A:84/-1"
}
```

Figure 5.7.: Request for an encrypted service in a JSON representation that can be submitted through the NBI.

a single entry point, i.e., the domain controller, the compiler needs to be able to create domain intents. These intents are then installed through a protocol, e.g., REST, and the corresponding driver, e.g., TAPI. Devices outside this domain, like the OVSs, are still configured individually via OF.

5.3.5. Measurements of Secure Service Creation

Two layers of encryption, i.e., physical and MACsec, and an unencrypted WDM connection are evaluated. Due to the software based implementation relying on OVS and therefore limited comparability, the previously presented IPsec is omitted [Szy+16]. More than 595 test runs are done for each of the three experiments. The resulting graphs are presented in Fig. 5.8 and the findings are discussed next.
First the time between the submission and the end of the compilation process was captured (see Fig. 5.8a). The goal is to evaluate if an additional delay is introduced by adding the encryption processing to the pipeline. The measured times for the forwarding and processing of all three intents are about the same. For the encrypted intents both mean values are 11.7 ms and for the unencrypted connection the mean value is 12.1 ms. This might be related to the slightly different implementation of the unencrypted branch. The margin of error for all of them is below 0.14 ms for a confidence level of 95 %. The compilation time itself was below 1 ms on average and is only responsible for less than 10 % of the processing time. As a result, we can state that the processing time remains about the same independent of the encryption handling.
The second graph (Fig. 5.8b) shows the time it took the SBI to complete the installation of the connection. There is a big difference when it comes to the installation time of



(a) Time from the submission of the intent until the compilation is finished.

(b) SBI installation time.

(c) SBI deletion time.

Figure 5.8.: Measurements for the compilation, installation and deletion.

a connection. For the unencrypted case, the mean value is about 99.1 s. On the other hand, the encrypted WDM connection only needs 28.4 s. The significant difference in setup time is a result of the used technologies. While the unencrypted multiplexed 100G connection uses coherent detection, the encrypted 10G applies direct detection. If both used the same technology, the results would be similar because the encryption has no influence on the setup time. As expected, the MACsec setup, being based on an Ethernet connection, needs the least amount of time (1.3 s) for setting up the connection for a single hop. The purpose of this graph is to visualize the difference in establishing a connection with growing capacity. Switching from Ethernet to optical increases the delay by a factor greater than 20. Delays in setup might lead to a preference of another technology option and should be considered when fulfilling applications' intents.

The last graph (Fig. 5.8c) captures the time it takes to delete an existing connection through the SBI. These values decide how long it takes before the resources are available again. The teardown of a lightpath in the testbed takes 12.6 s for an unencrypted and 11.8 s for the encrypted connection. Here, the difference between a 100G and a 10G connection is not very prominent. With 401 ms, MACsec is also the fastest when it comes to the cleanup afterwards. The conclusion is similar to the installation: the time for a deletion should be taken into account. This time, a factor of about 30 is introduced by switching to optical, although the absolute difference is lower than before.

In conclusion, even though the optical setup — encrypted and unencrypted — takes longer in the beginning, it is required for an efficient serving of higher bandwidths or to

satisfy latency constraints. Even from an economical point of view, it can be better to assign a dedicated lightpath instead of reusing higher layer transmission. MACsec has some drawbacks on its own, like the encryption/decryption of every frame at each hop along the path. While the optical transmission can be used for long distances without intermediate processing, L2 technologies need multiple hops to reach endpoints that are further apart and the reachability is limited to a L2 domain by default. Another result is that the selection of an appropriate encryption should also consider the setup and teardown times. For short-lived connections or demands that need to be fulfilled as soon as possible, optical connections might be unfit or need to be preprovisioned.

Summary

The presented scenarios showcase the integration of SDONs into multilayer networks and demonstrate their feasibility through proof-of-concept implementations. The data-center automation shows the benefits of a flexible control of optical network resources and multilayer operations by a data-center orchestrator, even though it also illustrates the drawbacks of packet-switching protocols and controllers with regard to OCS. The second scenario, representing an evolutionary step, uses an SDN controller that is able to recognize the different layers. In addition, a protocol for transport networks allows a more natural way of controlling the optical network. Both instances show how optical networks can be part of a multilayer network and thereby enhance the efficiency of control and augment the available options for applications. Still, there is more time needed until the techniques from these proofs of concept find their way into production networks.

Conclusion

This thesis investigated the introduction of SDN and NV to optical networks with flexible DWDM grids. The presented results and use cases pave the way toward a new era of dynamic optical networking. In particular, the thesis has shown that currently available models for the control of optical models lack a way of easily creating virtual topologies. The proposed intent interface provides a solution for these requests, while existing models can be leveraged for representing the physical and virtual topology inside of controllers. For the creation of virtual topologies, a procedure is defined focusing on the mapping of virtual links. A two-step process — first a (shortest) path calculation, then a spectrum assignment — has been chosen for the mapping of virtual links. For the shortest path computation, an algorithm utilizing precomputed data has been selected as a starting point. The results conclude that the provided extensions, which enable the addition and deletion of edges, work correctly and that the overhead is significantly smaller compared to performing a new precomputation. Measurements for optical networks and a large-scale computer network verify the algorithm's feasibility and indicate its scalability. The developed Optical Virtualization Controller (OVC) shows how SDN and NV can be applied to optical networks and provides a migration strategy for legacy equipment at the same time. The delay introduced by this mediation layer is identified to be very small, especially compared to set-up times in optical networks. Simultaneously, the architecture provides great flexibility in distribution and extensibility. Different deployment options for the SBI have been explored and are found to provide benefits with regard to the experienced delay and the transferred data for geographically distributed deployments. This thesis has also documented and confirmed the feasibility of an embedding into a multilayer environment by proofs of concept. A data-center integration showcased the advantages of a combined control of all data-center resources. Secure service creation demonstrated the control of multiple layers, including optical, through intents in the larger context of application-centric networking.

Future Work

The definition of models for the control and management of optical networks is an ongoing process. It will be important to keep track of updates and changes in the models in order to reevaluate the classifications and ratings. This includes the presented model for requesting virtual topologies, which should be thoroughly evaluated, possibly revised and brought into standardization. An exploration in the direction of further abstracting this interface to make it application-centric is desirable.
The implemented shortest path algorithm presents a starting point and an extension toward k-shortest paths is the obvious next step. By doing so, the blocking probability

for the resource assignment can be reduced. Also, an optimization and a potential parallelization of the code are required in order to achieve better comparability with existing algorithms and their implementations.
A large-scale evaluation of the OVC is another interesting field, which had to be omitted because of missing access to bigger networks and the absence of a suited emulator. Such an evaluation will also include other deployment options and migrations on a larger scale.

A. Code Generation from YANG Models

YANG facilitates the creation of models for configuration and state data (see Sec. 2.2). For an efficient workflow based on these models, a tool chain for the code generation is essential. Commonly used tool chains for YANG do not generate code with sufficient details. For example, the tool chain provided with the TAPI uses two steps to generate code. First, the YANG files are parsed by `pyang`, which is a YANG validator and converter [Pya18], and converted to Swagger's JSON format [Sma18]. Swagger is an API framework for designing, testing and deploying interfaces. Second, the available Swagger code generators for various programming languages are subsequently used to create stubs based on the previously created JSON representation. These stubs can then be filled with code and included in a server or client application. Unfortunately, the representation used by Swagger is not as expressive as the YANG modeling language. In this thesis, another approach was chosen. A plug-in for `pyang` was implemented that is able to generate all the data structures and stubs required for client and server applications. Directly extending a YANG converter has the benefit of having access to all details of the modeling language. Therefore, no information is lost in translation and all information can be used for the code generation process. Java was the first output language because it was needed for the implementation of the OVC. Java beans are the primary output for the data structures. In addition, code stubs for akka web interfaces are created and only need to be attached to existing code. For this reason the plug-in was named "alpakka". It has been used and extended throughout this thesis and ADVA will release it as an OSS soon.

B. Virtual Topology API YANG model

This is the YANG definition of the developed intent interface for requesting virtual topologies:

```
module virtual-topology-api {
    yang-version 1;
    namespace "http://www.advaoptical.com/virtop";
    prefix "virtop";

    organization "ADVA Optical Networking";
    contact "Thomas Szyrkowiec (tszyrkowiec@advaoptical.com)";

    description "Definition of an intent interface for requesting
    ↪  virtual topologies.";

    revision 2017-02-16 {
        description "Initial revision.";
    }

    /*******************
    * Type Definitions *
    *******************/

    typedef endpoint-identifier {
        type string;
        description "Unique identifier for an endpoint, e.g. node,
        ↪  module, port, interface.";
    }

    typedef intent-identifier {
        type string;
        description "Unique identifier for an intent group.";
    }

    typedef topology-identifier {
        type string;
        description "Unique identifier for a (virtual) topology.";
    }
```

```
typedef bit-rate {
    type uint64;
    units "Mbit/s";
    description "Data rate in Mbit/s.";
}

typedef disjoint {
    type enumeration {
        enum no;
        enum link;
        enum node;
        enum partially;
    }
    description "Types of disjoint paths.";
}

/*************
* Groupings *
*************/

grouping intent-group {
    leaf intent-id {
        type intent-identifier;
        description "Unique intent ID.";
    }
    leaf-list endpoints {
        type endpoint-identifier;
        min-elements 2;
        description "The intents are applied to this set of
        ↪  endpoints.";
    }
    leaf dedicated-bandwidth {
        type bit-rate;
        description "Reserved (always available) bandwidth between
        ↪  endpoints.";
    }
    leaf flexible-bandwidth {
        type bit-rate;
        description "Shared (best-effort) bandwidth between
        ↪  endpoints.";
    }
    leaf minimum-paths {
        type uint8;
```

```
        default 1;
        description "The minimum number of parallel paths.";
    }
    leaf disjoint-paths {
        type disjoint;
        default no;
        description "The type of disjointness for parallel paths.";
    }
    leaf protection {
        type boolean;
        default "false";
        description "True if optical protection is required.";
    }
    leaf maximum-active-connections {
        type uint8;
        description "The maximum number of installed lightpaths at
        ↪  the same time.";
    }
    leaf satisfied {
        config false;
        type boolean;
        description "Indicates if the intent is satisfied.";
    }
    description "An intent group describes the requirements for the
    ↪  connectivity between a set of endpoints.";
}

/*************
* Containers *
*************/

container topologies {
    config true;
    list installed-topologies {
        key topology-id;
        leaf topology-id {
            type topology-identifier;
            description "Unique topology ID.";
        }
        list intents {
            key intent-id;
            uses intent-group;
            description "A list of intent groups that need to be
            ↪  satisfied by the topology.";
```

```
            }
            description "A topology consists of an identifier and the
             ↪  intents that need to be satisfied by it.";
        }
        description "Starting point for the virtual topologies.";
    }

    container endpoints {
        config false;
        list assigned-endpoints {
            key endpoint-id;
            leaf endpoint-id {
                type endpoint-identifier;
            }
            description "List of all assigned endpoints that can be used
             ↪  for requesting virtual topologies.";
        }
        description "Starting point for the state data about assigned
         ↪  endpoints.";
    }
}
```

C. Contribution to ONOS

Multiple extensions to ONOS have been developed throughout this thesis, especially in the project ACINO. Some of them have been already contributed back to the community. Others are just available in the project repository [ACI18] or are about to be submitted. The most noteworthy contributions are the REST proxy, the domain intents, the TAPI drivers and the extensions for encryption.

REST Proxy

One of the initial issues in ONOS was the missing concept of a mediator or domain controller that controls parts of the network. The default approach in ONOS is to configure devices individually. This method might be feasible in a network with switches and routers but for optical networks and other domains managed by controllers, it leads to increased complexity. One reason is for example the setup of lightpaths through a network. Besides configuring analog parameters, like launch power, the process itself needs a particular structure, e.g., equalization and hop-by-hop setup of lightpaths. Additionally, a local controller might have a better knowledge about the domain because of additional information or a view without abstractions. Therefore, a mediator or a so-called proxy was introduced. This proxy is responsible for a set of devices, e.g., an optical domain, and exposes all devices and links to the controller through one connection. Due to available HTTP interfaces, a REST implementation was chosen. It followed the COP description initially. This REST proxy was developed in ACINO and later contributed to ONOS by a project partner.

TAPI Driver

Even though ONOS provides numerous protocols out of the box, they only cover the common behavior in most cases. To apply a protocol based on a YANG description, it is currently necessary to extend one of those protocols with the details of a particular implementation. The TAPI version 1.1 [Ope18a] was implemented based on the output of the code generator presented in App. A. The REST southbound protocol was used as a starting point because it supports intermediate controllers (see previous section). The TAPI is used for topology discovery, which includes nodes and links, as well as service setups. Topologies are exposed by the underlying controller. The driver extracts and exposes the individual elements and their adjacencies to ONOS. Most information is directly mapped to available entities, some additional data is needed in order to be able

to set up connections. This information is stored in the form of annotations, compliant with ONOS' architectural requirements.

Domain Intents

Domain intents are an important prerequisite for installing services in an optical domain. They have been developed in the ACINO project in collaboration with other institutes, contributed to ONOS [ONO18a] and merged in version 1.10.0 (Kingfisher). The main idea behind these intents is to configure parts of the network that are under control of a single controller and therefore, part of a domain. A domain intent specifies the ingress and egress points and optionally contains information on a preferred path inside the domain. It is assumed that the local controller is able to cover all steps for fulfilling the incoming requests. Without an included path the domain controller is free to choose any valid path for provisioning. In order to activate the domain processing, an NBI intent has to use a `DomainConstraint` flag.

Extensions for Encryption

The task of introducing an intent-based secure service setup to ONOS, which automatically assigns the best layer of encryption, includes many steps. The enhancements of the existing ONOS-based ACINO orchestrator [San+16] affect in particular the NBI, the intent processing and the SBI. Since these extensions have been presented in several publications [Cha+17; Szy+16; Szy+18], only a short summary is given.
First, the Application Centric Intents (ACIs) have been extended so that encryption can be requested via ONOS' NBI and CLI. It is noteworthy that a `DomainConstraint` and an `EncryptionConstraint` are part of a request for a secure service. The ACI compiler was extended to support the processing of such intents. A restriction or selection of the layer is considered a technological detail and therefore, not included in the intent. The best layer for encryption is currently chosen based on the bandwidth. If the encryption flag is missing, the (existing) unencrypted handling is applied. Devices and ports are annotated with information about encryption capabilities. This way, the compiler can check if an option satisfying all constraints is available. Otherwise, the intent fails and the user has to decide how to proceed. After the compilation, the compiler communicates the need for an encryption to the underlying network. The SBI implements new functionality to discover encryption capabilities. It also propagates encryption requests to the underlying hardware or mediation layer. TAPI's label fields are used for the retrieval of information about the encryption capabilities and the encryption layer of devices and ports.

Acronyms

ABNO Application-Based Network Operations.

ACI Application Centric Intent.

AES Advanced Encryption Standard.

API Application Programming Interface.

APSP All-Pairs Shortest Paths.

BFS Breadth-First Search.

BGP Border Gateway Protocol.

BGP-LS BGP with Link-State.

BVT Bandwidth Variable Transponder.

CAPEX Capital Expenditure.

CD Colorless, Directionless.

CDC Colorless, Directionless, Contentionless.

CDPI Control Data Plane Interface.

CLI command-Line Interface.

COP Control Orchestration Protocol.

CP Connection Point.

CRUD Create, Read, Update and Delete.

CVNI Control Virtual Network Interface.

DH Diffie-Hellman.

DHCP Dynamic Host Configuration Protocol.

DWDM Dense Wavelength Division Multiplexing.

EDFA Erbium Doped Fiber Amplifier.

EON Elastic Optical Network.

FEC Forward Error Correction.

GMPLS Generalized Multi-Protocol Label Switching.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

ILP Integer Linear Programming.

IP Internet Protocol.

IPsec Internet Protocol Security.

ISP Internet Service Provider.

ITU International Telecommunication Union.

ITU-T ITU Telecommunication Standardization Sector.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

L1 Layer 1.

L2 Layer 2.

L3 Layer 3.

LiTP Link Termination Point.

LLDP Link Layer Discovery Protocol.

LMP Link Management Protocol.

LoTP Logical Termination Point.

LSDB Link-State Database.

LSP Label Switched Path.

MACsec Media Access Control Security.

MIB Management Information Base.

MPLS Multiprotocol Label Switching.

NBI Northbound Interface.

NE Network Element.

NIC Network Interface Card.

NMS Network Management System.

NV Network Virtualization.

OADM Optical Add-Drop Multiplexer.

OCS Optical Circuit Switching.

ODU Optical Channel Data Unit.

OF OpenFlow.

OMS Optical Multiplex Section.

ONC Optical Network Controller.

ONF Open Networking Foundation.

ONOS Open Network Operating System.

OPEX Operating Expenditure.

OS Operating System.

OSC Optical Supervisory Channel.

OSI Open Systems Interconnection.

OSPF-TE Open Shortest Path First with Traffic Engineering.

OSS Open-Source Software.

OTN Optical Transport Network.

OTU Optical Channel Transport Unit.

OVC Optical Virtualization Controller.

OVS Open vSwitch.

OXC Optical Cross-Connect.

PCE Path Computation Element.

PCEP Path Computation Element Protocol.

PIP Physical Infrastructure Provider.

PLL Pruned Landmark Labeling.

QAM Quadrature Amplitude Modulation.

QPSK Quadrature Phase-Shift Keying.

REST Representational State Transfer.

ROADM Reconfigurable Optical Add-Drop Multiplexer.

RPC Remote Procedure Call.

RSA Routing and Spectrum Assignment.

RSVP-TE Resource Reservation Protocol with Traffic Engineering.

RWA Routing and Wavelength Assignment.

S-BVT Sliceable BVT.

SBI Southbound Interface.

SDH Synchronous Digital Hierarchy.

SDN Software-Defined Networking.

SDON Software-Defined Optical Network.

SNMP Simple Network Management Protocol.

SONET Synchronous Optical Networking.

SP Service Provider.

SPT Shortest-Path Tree.

SRG Shared Risk Group.

SSSP Single-Source Shortest Paths.

T-SDN Transport SDN.

TAPI Transport API.

TCP Transmission Control Protocol.

TE Traffic Engineering.

TED Traffic Engineering Database.

TET TE Topology.

TP Termination Point.

TTP Tunnel Termination Point.

UDP User Datagram Protocol.

UML Unified Modeling Language.

URL Uniform Resource Locator.

VLAN Virtual Local Area Network.

VM Virtual Machine.

VNE Virtual Network Embedding.

VNF Virtual Network Function.

VNP Virtual Network Provider.

VON Virtual Optical Network.

VPN Virtual Private Network.

WAN Wide Area Network.

WDM Wavelength Division Multiplexing.

WSON Wavelength Switched Optical Network.

WSS Wavelength Selective Switch.

XML Extensible Markup Language.

Bibliography

[AIY13] T. Akiba, Y. Iwata, and Y. Yoshida. "Fast Exact Shortest-path Distance Queries on Large Networks by Pruned Landmark Labeling". In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD '13. New York, New York, USA: ACM, 2013, pp. 349–360. ISBN: 978-1-4503-2037-5. DOI: 10.1145/2463676.2465315.

[AIY14] T. Akiba, Y. Iwata, and Y. Yoshida. "Dynamic and Historical Shortest-path Distance Queries on Large Evolving Networks by Pruned Landmark Labeling". In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14. Seoul, Korea: ACM, 2014, pp. 237–248. ISBN: 978-1-4503-2744-2. DOI: 10.1145/2566486.2568007.

[Aki+14] T. Akiba, Y. Iwata, K.-i. Kawarabayashi, and Y. Kawata. "Fast Shortest-path Distance Queries on Road Networks by Pruned Highway Labeling". In: *Proceedings of the Meeting on Algorithm Engineering & Expermiments*. Portland, Oregon: Society for Industrial and Applied Mathematics, 2014, pp. 147–154. DOI: 10.1137/1.9781611973198.14.

[Aki+15] T. Akiba, T. Hayashi, N. Nori, Y. Iwata, and Y. Yoshida. "Efficient Top-k Shortest-path Distance Queries on Large Networks by Pruned Landmark Labeling". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 2–8. ISBN: 0-262-51129-0.

[AlS+14] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow. "OpenVirteX: Make Your Virtual SDNs Programmable". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: ACM, 2014, pp. 25–30. ISBN: 978-1-4503-2989-7. DOI: 10.1145/2620728.2620741.

[Alv+17] R. Alvizu, G. Maier, N. Kukreja, A. Pattavina, R. Morro, A. Capello, and C. Cavazzoni. "Comprehensive survey on T-SDN: Software-defined Networking for Transport Networks". In: *IEEE Communications Surveys Tutorials* PP.99 (2017), pp. 1–1. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2715220.

[Aut16] A. Autenrieth. "Multilayer network planning — A practical perspective". In: *2016 Optical Fiber Communications Conference and Exhibition (OFC)*. Mar. 2016, pp. 1–3.

[AY16] T. Akiba and Y. Yano. "Compact and Scalable Graph Neighborhood Sketching". In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 685–694. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939762.

[AYM16] T. Akiba, Y. Yano, and N. Mizuno. "Hierarchical and Dynamic k-Path Covers". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. CIKM '16. Indianapolis, Indiana, USA: ACM, 2016, pp. 1543–1552. ISBN: 978-1-4503-4073-1. DOI: 10.1145/2983323.2983712.

[Azo+12] S. Azodolmolky, R. Nejabati, S. Peng, A. Hammad, M. P. Channegowda, N. Efstathiou, A. Autenrieth, P. Kaczmarek, and D. Simeonidou. "Optical FlowVisor: An OpenFlow-based Optical Network Virtualization Approach". In: *Optical Fiber Communication Conference*. Optical Society of America, 2012, JTh2A.41.

[Ber+14] P. Berde et al. "ONOS: Towards an Open, Distributed SDN OS". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: ACM, 2014, pp. 1–6. ISBN: 978-1-4503-2989-7. DOI: 10.1145/2620728.2620744.

[Bha97] R. Bhandari. "Optimal physical diversity algorithms and survivable networks". In: *Proceedings Second IEEE Symposium on Computer and Communications*. July 1997, pp. 433–441. DOI: 10.1109/ISCC.1997.616037.

[Ble+16] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer. "Survey on Network Virtualization Hypervisors for Software Defined Networking". In: *IEEE Communications Surveys Tutorials* 18.1 (Firstquarter 2016), pp. 655–685. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2489183.

[Cas+18] R. Casellas, R. Martínez, R. Vilalta, and R. Muñoz. "Control, Management, and Orchestration of Optical Networks: Evolution, Trends, and Challenges". In: *Journal of Lightwave Technology* 36.7 (Apr. 2018), pp. 1390–1402. ISSN: 0733-8724. DOI: 10.1109/JLT.2018.2793464.

[CB09] N. M. M. K. Chowdhury and R. Boutaba. "Network virtualization: state of the art and research challenges". In: *IEEE Communications Magazine* 47.7 (July 2009), pp. 20–26. ISSN: 0163-6804. DOI: 10.1109/MCOM.2009.5183468.

[CB10] N. M. K. Chowdhury and R. Boutaba. "A Survey of Network Virtualization". In: *Comput. Netw.* 54.5 (Apr. 2010), pp. 862–876. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2009.10.017.

[Coh+02] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. "Reachability and Distance Queries via 2-hop Labels". In: *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '02. San Francisco, California: Society for Industrial and Applied Mathematics, 2002, pp. 937–946. ISBN: 0-89871-513-X.

[Coh+03] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. "Reachability and Distance Queries via 2-Hop Labels". In: *SIAM Journal on Computing* 32.5 (2003), pp. 1338–1355. DOI: 10.1137/S0097539702403098.

[Coh+13] R. Cohen et al. "An intent-based approach for network virtualization". In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. May 2013, pp. 42–50.

[Cru+14] B. de la Cruz, O. G. de Dios, V. Lopez, and J. P. Fernández-Palacios. "Operational expenditures savings in IP/MPLS over DWDM networks by Multi-layer restoration". In: *OFC 2014*. Mar. 2014, pp. 1–3. DOI: 10.1364/OFC.2014.M3B.5.

[CY09] E. P. F. Chan and Y. Yang. "Shortest Path Tree Computation in Dynamic Graphs". In: *IEEE Transactions on Computers* 58.4 (Apr. 2009), pp. 541–557. ISSN: 0018-9340. DOI: 10.1109/TC.2008.198.

[Dij59] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1.1 (Dec. 1959), pp. 269–271. ISSN: 0945-3245. DOI: 10.1007/BF01386390.

[Dio17] O. G. de Dios. "Control Plane architectures for Flexi-Grid Networks". In: *Optical Fiber Communication Conference*. Optical Society of America, 2017, W1H.2. DOI: 10.1364/OFC.2017.W1H.2.

[DK15] R. Durner and W. Kellerer. "The cost of Security in the SDN control Plane". In: *ACM CoNEXT 2015 - Student Workshop*. Dec. 2015.

[DPM12] S. Das, G. Parulkar, and N. McKeown. "Why OpenFlow/SDN Can Succeed Where GMPLS Failed". In: *European Conference and Exhibition on Optical Communication*. Optical Society of America, 2012, Tu.1.D.1. DOI: 10.1364/ECEOC.2012.Tu.1.D.1.

[EA12] J. P. Elbers and A. Autenrieth. "From static to software-defined optical networks". In: *2012 16th International Conference on Optical Network Design and Modelling (ONDM)*. Apr. 2012, pp. 1–4. DOI: 10.1109/ONDM.2012.6210207.

[EA13] J. P. Elbers and A. Autenrieth. "Extending network virtualization into the optical domain". In: *2013 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*. Mar. 2013, pp. 1–3. DOI: 10.1364/OFC.2013.OM3E.3.

[Egi+10] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, L. Mathy, and P. Papadimitriou. "A Platform for High Performance and Flexible Virtual Routers on Commodity Hardware". In: *SIGCOMM Comput. Commun. Rev.* 40.1 (Jan. 2010), pp. 127–128. ISSN: 0146-4833. DOI: 10.1145/1672308.1672332.

[Epp98] D. Eppstein. "Finding the k Shortest Paths". In: *SIAM Journal on Computing* 28.2 (1998), pp. 652–673. DOI: 10.1137/S0097539795290477.

[Faw+04] W. Fawaz, B. Daheb, O. Audouin, M. Du-Pond, and G. Pujolle. "Service level agreement and provisioning in optical networks". In: *IEEE Communications Magazine* 42.1 (Jan. 2004), pp. 36–43. ISSN: 0163-6804. DOI: 10.1109/MCOM.2004.1262160.

[Fis+13] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach. "Virtual Network Embedding: A Survey". In: *IEEE Communications Surveys Tutorials* 15.4 (Fourth 2013), pp. 1888–1906. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.013013.00155.

[FMN00] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. "Fully Dynamic Algorithms for Maintaining Shortest Paths Trees". In: *J. Algorithms* 34.2 (Feb. 2000), pp. 251–281. ISSN: 0196-6774. DOI: 10.1006/jagm.1999.1048.

[Fur+14] M. Furdek, N. Skorin-Kapov, S. Zsigmond, and L. Wosinska. "Vulnerabilities and security issues in optical networks". In: *2014 16th International Conference on Transparent Optical Networks (ICTON)*. July 2014, pp. 1–4. DOI: 10.1109/ICTON.2014.6876451.

[GBX13] S. Gringeri, N. Bitar, and T. J. Xia. "Extending software defined network principles to include optical transport". In: *IEEE Communications Magazine* 51.3 (Mar. 2013), pp. 32–40. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6476863.

[GE13] K. Grobe and M. Eiselt. *Wavelength Division Multiplexing: A Practical Engineering Guide*. 1st. Wiley Publishing, 2013. ISBN: 9780470623022.

[Ger+12] O. Gerstel, M. Jinno, A. Lord, and S. J. B. Yoo. "Elastic optical networking: a new dawn for the optical layer?" In: *IEEE Communications Magazine* 50.2 (Feb. 2012), s12–s20. ISSN: 0163-6804. DOI: 10.1109/MCOM.2012.6146481.

[Gio+12] A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi. "OpenFlow and PCE architectures in Wavelength Switched Optical Networks". In: *2012 16th International Conference on Optical Network Design and Modelling (ONDM)*. Apr. 2012, pp. 1–6. DOI: 10.1109/ONDM.2012.6210213.

[GLS15] O. Gerstel, V. Lopez, and D. Siracusa. "Multi-layer orchestration for application-centric networking". In: *2015 International Conference on Photonics in Switching (PS)*. Sept. 2015, pp. 318–320. DOI: 10.1109/PS.2015.7329039.

[Gol74] R. P. Goldberg. "Survey of virtual machine research". In: *Computer* 7.6 (June 1974), pp. 34–45. ISSN: 0018-9162. DOI: 10.1109/MC.1974.6323581.

[Gri+10] S. Gringeri, B. Basch, V. Shukla, R. Egorov, and T. J. Xia. "Flexible architectures for optical transport nodes and networks". In: *IEEE Communications Magazine* 48.7 (July 2010), pp. 40–50. ISSN: 0163-6804. DOI: 10.1109/MCOM.2010.5496877.

[Gud+10] V. Gudla, S. Das, A. Shastri, G. Parulkar, N. McKeown, L. Kazovsky, and S. Yamashita. "Experimental demonstration of OpenFlow control of packet and circuit switches". In: *2010 Conference on Optical Fiber Communication (OFC/NFOEC), collocated National Fiber Optic Engineers Conference*. Mar. 2010, pp. 1–3. DOI: 10.1364/OFC.2010.OTuG2.

[Guo14] A. Guo. "Network Virtualization". In: *Optical Fiber Communication Conference*. Optical Society of America, 2014, M2B.5. DOI: 10.1364/OFC.2014.M2B.5.

[HAK16] T. Hayashi, T. Akiba, and K.-i. Kawarabayashi. "Fully Dynamic Shortest-Path Distance Query Acceleration on Massive Networks". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. CIKM '16. Indianapolis, Indiana, USA: ACM, 2016, pp. 1533–1542. ISBN: 978-1-4503-4073-1. DOI: 10.1145/2983323.2983731.

[Han+16] Y. Han, J. Li, D. Hoang, J. H. Yoo, and J. W. K. Hong. "An intent-based network virtualization platform for SDN". In: *2016 12th International Conference on Network and Service Management (CNSM)*. Oct. 2016, pp. 353–358. DOI: 10.1109/CNSM.2016.7818446.

[Hua+12] N. Hua, Y. Liu, X. Wan, X. Zheng, and Z. Liu. "Dynamic routing and spectrum assignment algorithms in flexible optical networks: An overview". In: *7th International Conference on Communications and Networking in China*. Aug. 2012, pp. 251–255. DOI: 10.1109/ChinaCom.2012.6417485.

[IGL13a] P. Iovanna, A. Germoni, and V. López. "Integration of planning and control plane in packet optical multilayer network". In: *2013 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*. Mar. 2013, pp. 1–3. DOI: 10.1364/NFOEC.2013.NW1F.2.

[IGL13b] P. Iovanna, A. Germoni, and V. López. "Integration of planning and control plane in packet optical multilayer network". In: *2013 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*. Mar. 2013, pp. 1–3. DOI: 10.1364/NFOEC.2013.NW1F.2.

[Jan+16] C. Janz, L. Ong, K. Sethuraman, and V. Shukla. "Emerging transport SDN architecture and use cases". In: *IEEE Communications Magazine* 54.10 (Oct. 2016), pp. 116–121. ISSN: 0163-6804. DOI: 10.1109/MCOM.2016.7588279.

[Ji+14] P. N. Ji, T. J. Xia, J. Hu, M.-F. Huang, Y. Aono, T. Tajima, G. A. Wellbrock, and T. Wang. "Demonstration of OpenFlow-enabled traffic and network adaptive transport SDN". In: *OFC 2014*. Mar. 2014, pp. 1–3. DOI: 10.1364/OFC.2014.W2A.20.

[JP13] R. Jain and S. Paul. "Network virtualization and software defined networking for cloud computing: a survey". In: *IEEE Communications Magazine* 51.11 (Nov. 2013), pp. 24–31. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6658648.

[KG08] E. Keller and E. Green. "Virtualizing the Data Plane Through Source Code Merging". In: *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*. PRESTO '08. Seattle, WA, USA: ACM, 2008, pp. 9–14. ISBN: 978-1-60558-181-1. DOI: 10.1145/1397718.1397721.

[Kin+16] D. King, C. Rotsos, A. Aguado, N. Georgalas, and V. Lopez. "The Software Defined Transport Network: Fundamentals, findings and futures". In: *2016 18th International Conference on Transparent Optical Networks (ICTON)*. July 2016, pp. 1–4. DOI: 10.1109/ICTON.2016.7550669.

[Kle12] A. Klekamp. "Multi-layer network optimization: Benefits of elastic optical networks". In: *2012 14th International Conference on Transparent Optical Networks (ICTON)*. July 2012, pp. 1–5. DOI: 10.1109/ICTON.2012.6254388.

[KRV13] D. Kreutz, F. M. Ramos, and P. Verissimo. "Towards Secure and Dependable Software-defined Networks". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China: ACM, 2013, pp. 55–60. ISBN: 978-1-4503-2178-5. DOI: 10.1145/2491185.2491199.

[Li+14] Y. Li, X. Chen, N. Hua, and X. Zheng. "A Novel Virtual Optical Network Embedding Strategy for Optical Network Virtualization". In: *Advanced Photonics for Communications*. Optical Society of America, 2014, NT1C.3. DOI: 10.1364/NETWORKS.2014.NT1C.3.

[Liu+11] L. Liu, T. Tsuritani, I. Morita, H. Guo, and J. Wu. "OpenFlow-based Wavelength Path Control in Transparent Optical Networks: a Proof-of-Concept Demonstration". In: *37th European Conference and Exposition on Optical Communications*. Optical Society of America, 2011, Tu.5.K.2. DOI: 10.1364/ECOC.2011.Tu.5.K.2.

[Lop+17a] V. Lopez, D. Konidis, D. Siracusa, C. Rozic, I. Tomkos, and J. P. Fernandez-Palacios. "On the Benefits of Multilayer Optimization and Application Awareness". In: *Journal of Lightwave Technology* 35.6 (Mar. 2017), pp. 1274–1279. ISSN: 0733-8724. DOI: 10.1109/JLT.2017.2674180.

[LV16] V. López and L. Velasco. *Elastic Optical Networks*. Springer, Cham, 2016. ISBN: 9783319301730. DOI: 10.1007/978-3-319-30174-7.

[Mcd13] D. Mcdysan. "Software defined networking opportunities for transport". In: *IEEE Communications Magazine* 51.3 (Mar. 2013), pp. 28–31. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6476862.

[McK+08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: Enabling Innovation in Campus Networks". In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746.

[Muk06] B. Mukherjee. *Optical WDM Networks (Optical Networks)*. Springer, Boston, MA, 2006. ISBN: 9780387290553. DOI: 10.1007/0-387-29188-1.

[Pag+12a] A. Pagès, J. Perelló, S. Spadaro, J. A. García-Espín, J. F. Riera, and S. Figuerola. "Optimal allocation of virtual optical networks for the future internet". In: *2012 16th International Conference on Optical Network Design and Modelling (ONDM)*. Apr. 2012, pp. 1–6. DOI: 10.1109/ONDM.2012.6210209.

[Pag+12b] A. Pages, J. Perello, S. Spadaro, and G. Junyent. "Strategies for Virtual Optical Network Allocation". In: *IEEE Communications Letters* 16.2 (Feb. 2012), pp. 268–271. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2011.120211.111900.

[Pal+14] E. Palkopoulou, O. Gerstel, I. Stiakogiannakis, T. Telkamp, V. López, and I. Tomkos. "Impact of IP layer routing policy on multi-layer design". In: *OFC 2014*. Mar. 2014, pp. 1–3. DOI: 10.1364/OFC.2014.W1K.4.

[Pen+11] S. Peng, R. Nejabati, S. Azodolmolky, E. Escalona, and D. Simeonidou. "An impairment-aware virtual optical network composition mechanism for future Internet". In: *Opt. Express* 19.26 (Dec. 2011), B251–B259. DOI: 10.1364/OE.19.00B251.

[Pen+14] S. Peng et al. "A novel SDN enabled hybrid optical packet/circuit switched data centre network: The LIGHTNESS approach". In: *2014 European Conference on Networks and Communications (EuCNC)*. June 2014, pp. 1–5. DOI: 10.1109/EuCNC.2014.6882622.

[Pic+06] M. Pickavet, P. Demeester, D. Colle, D. Staessens, B. Puype, L. Depre, and I. Lievens. "Recovery in multilayer optical networks". In: *Journal of Lightwave Technology* 24.1 (Jan. 2006), pp. 122–134. ISSN: 0733-8724. DOI: 10.1109/JLT.2005.861118.

[RLL12] R. Raghavendra, J. Lobo, and K.-W. Lee. "Dynamic Graph Query Primitives for SDN-based Cloudnetwork Management". In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland: ACM, 2012, pp. 97–102. ISBN: 978-1-4503-1477-0. DOI: 10.1145/2342441.2342461.

[RM02] R. Ramamurthy and B. Mukherjee. "Fixed-alternate routing and wavelength conversion in wavelength-routed optical networks". In: *IEEE/ACM Transactions on Networking* 10.3 (June 2002), pp. 351–367. ISSN: 1063-6692. DOI: 10.1109/TNET.2002.1012367.

[RWW04] R. Ramaswamy, N. Weng, and T. Wolf. "Characterizing network processing delay". In: *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*. Vol. 3. Nov. 2004, 1629–1634 Vol.3. DOI: 10.1109/GLOCOM.2004.1378257.

[Sam+15] N. Sambo et al. "Next generation sliceable bandwidth variable transponders". In: *IEEE Communications Magazine* 53.2 (Feb. 2015), pp. 163–171. ISSN: 0163-6804. DOI: 10.1109/MCOM.2015.7045405.

[Sch+09] G. Schaffrath et al. "Network Virtualization Architecture: Proposal and Initial Prototype". In: *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*. VISA '09. Barcelona, Spain: ACM, 2009, pp. 63–72. ISBN: 978-1-60558-595-6. DOI: 10.1145/1592648.1592659.

[Sch12] D. A. Schupke. "Multilayer and Multidomain Resilience in Optical Networks". In: *Proceedings of the IEEE* 100.5 (May 2012), pp. 1140–1148. ISSN: 0018-9219. DOI: 10.1109/JPROC.2012.2183330.

[Sco15] S. Scott-Hayward. "Design and deployment of secure, robust, and resilient SDN controllers". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. Apr. 2015, pp. 1–5. DOI: 10.1109/NETSOFT.2015.7258233.

[She+10] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. "Can the Production Network Be the Testbed?" In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. OSDI'10. Vancouver, BC, Canada: USENIX Association, 2010, pp. 365–378.

[Sim14] J. M. Simmons. *Optical Network Design and Planning*. 2nd ed. Springer, Cham, 2014. ISBN: 9783319052267. DOI: 10.1007/978-3-319-05227-4.

[SOS13] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. "Sdn Security: A Survey". In: *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. Nov. 2013, pp. 1–7. DOI: 10.1109/SDN4FNS.2013.6702553.

[SZ16] J. Spooner and S. Y. Zhu. "A Review of Solutions for SDN-Exclusive Security Issues". In: *International Journal of Advanced Computer Science and Applications (ijacsa)* 7.8 (2016). DOI: 10.14569/IJACSA.2016.070817.

[Thy+16] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer. "Software Defined Optical Networks (SDONs): A Comprehensive Survey". In: *IEEE Communications Surveys Tutorials* 18.4 (July 2016), pp. 2738–2786. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2586999.

[Tom+14] I. Tomkos, S. Azodolmolky, J. Solé-Pareta, D. Careglio, and E. Palkopoulou. "A tutorial on the flexible optical networking paradigm: State of the art, trends, and research challenges". In: *Proceedings of the IEEE* 102.9 (Sept. 2014), pp. 1317–1337. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2324652.

[Tre+11] K. Tretyakov, A. Armas-Cervantes, L. García-Bañuelos, J. Vilo, and M. Dumas. "Fast Fully Dynamic Landmark-based Estimation of Shortest Path Distances in Very Large Graphs". In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. CIKM '11. Glasgow, Scotland, UK: ACM, 2011, pp. 1785–1794. ISBN: 978-1-4503-0717-8. DOI: 10.1145/2063576.2063834.

[Vil+14] R. Vilalta, R. Muñoz, R. Casellas, and R. Martínez. "Performance evaluation of novel resource allocation algorithms for virtual elastic optical networks". In: *2014 16th International Conference on Transparent Optical Networks (ICTON)*. July 2014, pp. 1–4. DOI: 10.1109/ICTON.2014.6876320.

[Wan+11] X. Wan, L. Wang, N. Hua, H. Zhang, and X. Zheng. "Dynamic Routing and Spectrum Assignment in Flexible Optical Path Networks". In: *Optical Fiber Communication Conference/National Fiber Optic Engineers Conference 2011*. Optical Society of America, 2011, JWA055. DOI: 10.1364/NFOEC.2011.JWA055.

[Wan+15] X. Wang, Q. Zhang, I. Kim, P. Palacharla, and M. Sekiya. "Virtual network provisioning over distance-adaptive flexible-grid optical networks [Invited]". In: *IEEE/OSA Journal of Optical Communications and Networking* 7.2 (Feb. 2015), A318–A325. ISSN: 1943-0620. DOI: 10.1364/JOCN.7.00A318.

[WC14] B. Wu and K. Chao. *Spanning Trees and Optimization Problems*. Chapman and Hall/CRC, 2014. ISBN: 9781584884361.

[Wei10] F. Wei. "TEDI: Efficient Shortest Path Query Answering on Graphs". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. SIGMOD '10. Indianapolis, Indiana, USA: ACM, 2010, pp. 99–110. ISBN: 978-1-4503-0032-2. DOI: 10.1145/1807167.1807181.

[WLV13] P. Wright, A. Lord, and L. Velasco. "The network capacity benefits of Flexgrid". In: *2013 17th International Conference on Optical Networking Design and Modeling (ONDM)*. Apr. 2013, pp. 7–12.

[Xu+16] Q. Xu, X. Zhang, J. Zhao, X. Wang, and T. Wolf. "Fast shortest-path queries on large-scale graphs". In: *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. Nov. 2016, pp. 1–10. DOI: 10.1109/ICNP.2016.7784419.

[Zha+13a] J. Zhang et al. "First demonstration of enhanced software defined networking (eSDN) over elastic grid (eGrid) optical networks for data center service migration". In: *2013 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*. Mar. 2013, pp. 1–3.

[Zha+13b] Q. Zhang, W. Xie, Q. She, X. Wang, P. Palacharla, and M. Sekiya. "RWA for Network Virtualization in Optical WDM Networks". In: *Optical Fiber Communication Conference/National Fiber Optic Engineers Conference 2013*. Optical Society of America, 2013, JTh2A.65. DOI: 10.1364/NFOEC.2013.JTh2A.65.

[Zha+13c] S. Zhang, L. Shi, C. S. Vadrevu, and B. Mukherjee. "Network virtualization over WDM and flexible-grid optical networks". In: *Optical Switching and Networking* 10.4 (2013), pp. 291–300. ISSN: 1573-4277. DOI: 10.1016/j.osn.2013.03.005.

[Zha+13d] X. Zhao, V. Vusirikala, B. Koley, V. Kamalov, and T. Hofmeister. "The prospect of inter-data-center optical networks". In: *IEEE Communications Magazine* 51.9 (Sept. 2013), pp. 32–38. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6588647.

[Zha+13e] Y. Zhao, J. Zhang, H. Yang, and Y. Yu. "Which is more suitable for the control over large scale optical networks, GMPLS or OpenFlow?" In: *2013 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*. Mar. 2013, pp. 1–3.

[Zhu+15] Y. Zhu, K. K. Goo, Y. Liang, Q. Zhang, X. Wang, P. Palacharla, and M. Sekiya. "Scalable virtual network provisioning over distance-adaptive flexible grid optical networks". In: *2015 Optical Fiber Communications Conference and Exhibition (OFC)*. Mar. 2015, pp. 1–3.

[ZJM00] H. Zang, J. P. Jue, and B. Mukherjee. "A Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical WDM Networks". In: *Optical Networks Magazine* 1.1 (2000), pp. 47–60. ISSN: 1572-8161. DOI: 10.1023/A:1017212720008.

[ZSB13] J. Zhao, S. Subramaniam, and M. Brandt-Pearce. "Virtual topology mapping in elastic optical networks". In: *2013 IEEE International Conference on Communications (ICC)*. June 2013, pp. 3904–3908. DOI: 10.1109/ICC.2013.6655167.

Online Documents

[4WA09] 4WARD consortium. *D-3.1.1 Virtualisation Approach: Concept*. Public Deliverable. The 4WARD Project, Sept. 2009. URL: http://www.4ward-project.eu/index9d96.html?s=Deliverables.

[Abr+12] I. Abraham, D. Delling, A. Goldberg, and R. Werneck. *Hierarchical Hub Labelings for Shortest Paths*. Tech. rep. MSR-TR-2012-46. Microsoft Research, Apr. 2012. URL: https://www.microsoft.com/en-us/research/publication/hierarchical-hub-labelings-for-shortest-paths/.

[ACI18] ACINO. *GitHub - Application-Centric IP/optical Network Orchestration*. 2018. URL: https://github.com/ACINO-H2020 (visited on 04/15/2018).

[ADH12] M. Auster, N. Damouny, and J. Harcourt. *OpenFlow-Enabled Hybrid Cloud Services Connect Enterprise and Service Provider Data Centers*. Solution Brief. Open Networking Foundation, Nov. 2012. 9 pp. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-hybrid-cloud-services.pdf.

[Aus+14] M. Auster, S. Balakrishnan, M. McBride, G. Nehib, L. Ong, M. Shirazipour, V. Shukla, A. Tong, and M. Vissers. *OpenFlow-enabled Transport SDN*. Solution Brief. Open Networking Foundation, May 2014. 16 pp. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-of-enabled-transport-sdn.pdf.

[BBW17] A. Bierman, M. Bjorklund, and K. Watsen. *RESTCONF Protocol*. RFC 8040. Jan. 2017. DOI: 10.17487/rfc8040. URL: https://rfc-editor.org/rfc/rfc8040.txt.

[BCM17] D. Bogdanovic, B. Claise, and C. Moberg. *YANG Module Classification*. RFC 8199. July 2017. DOI: 10.17487/RFC8199. URL: https://rfc-editor.org/rfc/rfc8199.txt.

[Bee+16] V. P. Beeram, T. Saad, H. C. Shah, O. G. de Dios, X. Liu, and I. Bryskin. *YANG Data Model for TE Topologies*. Internet-Draft draft-ietf-teas-yang-te-topo-06. Work in Progress. Internet Engineering Task Force, Oct. 2016. 100 pp. URL: https://tools.ietf.org/html/draft-ietf-teas-yang-te-topo-06.

[Bjo15] M. Bjorklund. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. RFC 6020. Oct. 2015. DOI: 10.17487/rfc6020. URL: https://rfc-editor.org/rfc/rfc6020.txt.

[Bjo16] M. Bjorklund. *The YANG 1.1 Data Modeling Language*. RFC 7950. Aug. 2016. DOI: 10.17487/RFC7950. URL: https://rfc-editor.org/rfc/rfc7950.txt.

[Cra+17a] E. Crabbe, I. Minei, J. Medved, and R. Varga. *PCEP Extensions for Stateful PCE*. Internet-Draft draft-ietf-pce-stateful-pce-21. Work in Progress. Internet Engineering Task Force, June 2017. 54 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-pce-stateful-pce-21.

[Cra+17b] E. Crabbe, I. Minei, S. Sivabalan, and R. Varga. *PCEP Extensions for PCE-initiated LSP Setup in a Stateful PCE Model*. Internet-Draft draft-ietf-pce-pce-initiated-lsp-10. Work in Progress. Internet Engineering Task Force, June 2017. 18 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-pce-pce-initiated-lsp-10.

[Del+14] D. Delling, A. Goldberg, T. Pajor, and R. Werneck. *Robust Exact Distance Queries on Massive Networks*. Tech. rep. MSR-TR-2014-12. Microsoft Technical Report, July 2014. URL: https://www.microsoft.com/en-us/research/publication/robust-exact-distance-queries-on-massive-networks/.

[Enn+11] R. Enns, M. Bjorklund, A. Bierman, and J. Schönwälder. *Network Configuration Protocol (NETCONF)*. RFC 6241. June 2011. DOI: 10.17487/rfc6241. URL: https://rfc-editor.org/rfc/rfc6241.txt.

[Gre+16] H. Gredler, J. Medved, S. Previdi, A. Farrel, and S. Ray. *North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP*. RFC 7752. Mar. 2016. DOI: 10.17487/RFC7752. URL: https://rfc-editor.org/rfc/rfc7752.txt.

[IBM17] IBM. *Cost of Data Breach Study*. 2017. URL: http://www-03.ibm.com/security/data-breach/ (visited on 04/15/2018).

[IEE06] IEEE. "IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security". In: *IEEE Std 802.1AE-2006* (Aug. 2006), pp. 1–150. DOI: 10.1109/IEEESTD.2006.245590.

[IEE10] IEEE. "IEEE Standard for Local and metropolitan area networks–Port-Based Network Access Control". In: *IEEE Std 802.1X-2010 (Revision of IEEE Std 802.1X-2004)* (Feb. 2010), pp. 1–205. DOI: 10.1109/IEEESTD.2010.5409813.

[ITU00] ITU Telecommunication Standardization Sector. *Overview of TMN Recommendations*. Recommendation M.3000. International Telecommunication Union, Feb. 2000. URL: https://www.itu.int/rec/T-REC-M.3000-200002-I.

[ITU12] ITU Telecommunication Standardization Sector. *Spectral grids for WDM applications: DWDM frequency grid*. Recommendation G.694.1. International Telecommunication Union, Feb. 2012. URL: https://www.itu.int/rec/T-REC-G.694.1-201202-I.

[ITU16] ITU Telecommunication Standardization Sector. *Optical transport network: Protocol-neutral management information model for the network element view*. Recommendation G.874.1. International Telecommunication Union, Nov. 2016. URL: https://www.itu.int/rec/T-REC-G.874.1-201611-I.

[KF15] D. King and A. Farrel. *A PCE-Based Architecture for Application-Based Network Operations*. RFC 7491. Mar. 2015. DOI: 10.17487/RFC7491. URL: https://rfc-editor.org/rfc/rfc7491.txt.

[LD15] K. Lam and N. Davis. *Core Information Model (CoreModel)*. Technical Recommendations ONF TR-512 (Version 1.1). Work in Progress. Open Networking Foundation, Nov. 2015. 162 pp. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/ONF-CIM_Core_Model_base_document_1.1.pdf.

[Lee+17] Y. Lee, D. Dhody, X. Zhang, V. Lopezalvarez, D. King, B.-Y. Yoon, and A. Guo. *A Yang Data Model for WSON Optical Networks*. Internet-Draft draft-ietf-ccamp-wson-yang-04. Work in Progress [Accessed on 31.03.2017]. Internet Engineering Task Force, Jan. 2017. 12 pp. URL: https://tools.ietf.org/html/draft-ietf-ccamp-wson-yang-04.

[Lig18] Lightbend Inc. *Akka*. 2018. URL: https://akka.io/ (visited on 04/15/2018).

[LK14] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. June 2014. URL: http://snap.stanford.edu/data (visited on 04/15/2018).

[Mad+16] U. A. de Madrid, V. Lopezalvarez, O. G. de Dios, Z. Ali, D. King, and Y. Lee. *YANG data model for Flexi-Grid Optical Networks*. Internet-Draft draft-vergara-ccamp-flexigrid-yang-03. Work in Progress. Internet Engineering Task Force, July 2016. 34 pp. URL: https://tools.ietf.org/html/draft-vergara-ccamp-flexigrid-yang-03.

[Med+17] J. Medved, N. Bahadur, H. Ananthakrishnan, X. Liu, R. Varga, and A. Clemm. *A Data Model for Network Topologies*. Internet-Draft draft-ietf-i2rs-yang-network-topo-11. Work in Progress. Internet Engineering Task Force, Feb. 2017. 34 pp. URL: https://tools.ietf.org/html/draft-ietf-i2rs-yang-network-topo-11.

[Miz+12] S. Mizuno, H. Sakai, D. Yokozeki, K. Iida, and T. Koyama. "IaaS Platform Using OpenStack and OpenFlow Overlay Technology". In: *NTT Technical Review* 10.12 (Dec. 2012). URL: https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201212fa1.html.

[NC17] B. Naveh and Contributors. *JGraphT*. 2017. URL: http://jgrapht.org/ (visited on 04/15/2018).

[ONO18a] ONOS. *GitHub - Open Network Operating System*. 2018. URL: https://github.com/opennetworkinglab/onos (visited on 04/15/2018).

[ONO18b] ONOS. *ONOS - Open Network Operating System*. 2018. URL: https://onosproject.org/ (visited on 04/15/2018).

[Ope13] Open Networking Foundation. *OpenFlow Switch Specification*. Technical Specification ONF TS-012. Version 1.4.0 (Wire Protocol 0x05). Open Networking Foundation, Oct. 2013. 206 pp. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[Ope15] Open Networking Foundation. *Optical Transport Protocol Extensions*. Technical Specification ONF TS-022. Version 1.0. Open Networking Foundation, Mar. 2015. 33 pp. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/Optical_Transport_Protocol_Extensions_V1.0.pdf.

[Ope16] OpenROADM. *Metro Open ROADM Network Model*. Interoperability Specification. Work in Progress. July 2016. URL: http://0201.nccdn.net/4_2/000/000/002/4ae/Open-ROADM-Network-Model-Whitepaper-v1-1.pdf.

[Ope18a] Open Networking Foundation. *GitHub - ONF Transport API Repository (Snowmass)*. 2018. URL: https://github.com/OpenNetworkingFoundation/TAPI (visited on 04/15/2018).

[Ope18b] Open Networking Foundation. *SDN Technical Specifications*. 2018. URL: https://www.opennetworking.org/software-defined-standards/specifications/ (visited on 04/15/2018).

[Ope18c] Open Networking Foundation. *Software-Defined Networking (SDN) Definition*. 2018. URL: https://www.opennetworking.org/sdn-definition/ (visited on 04/15/2018).

[Ope18d] OpenConfig. *GitHub - OpenConfig*. 2018. URL: https://github.com/openconfig/public (visited on 04/15/2018).

[Ope18e] OpenDaylight Project. *OpenDaylight*. 2018. URL: https://www.opendaylight.org/ (visited on 04/15/2018).

[Ope18f] OpenROADM. *GitHub - Open ROADM MSA*. 2018. URL: https://github.com/OpenROADM/OpenROADM_MSA_Public (visited on 04/15/2018).

[Ope18g] OpenStack. *OpenStack*. 2018. URL: http://www.openstack.org/ (visited on 04/15/2018).

[Pro18] Project Floodlight. *Project Floodlight*. 2018. URL: http://www.projectfloodlight.org/ (visited on 04/15/2018).

[Pya18] Pyang. *GitHub - An extensible YANG validator and converter in python*. 2018. URL: https://github.com/mbj4668/pyang (visited on 04/15/2018).

[Qia+16] C. Qiaogang et al. *Functional Requirements for Transport API*. Technical Recommendations ONF TR-527 (Version No.01). Work in Progress. Open Networking Foundation, June 2016. 71 pp. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-527_TAPI_Functional_Requirements.pdf.

[She+09] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. *FlowVisor: A Network Virtualization Layer*. Technical Report. Oct. 2009. URL: http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf.

[SK05] K. Seo and S. Kent. *Security Architecture for the Internet Protocol*. RFC 4301. Dec. 2005. DOI: 10.17487/RFC4301. URL: https://rfc-editor.org/rfc/rfc4301.txt.

[Sma18] SmartBear Software. *Swagger*. 2018. URL: https://swagger.io/ (visited on 04/15/2018).

[STR14] STRAUSS consortium. *Deliverable 3.1 - Reference architecture, use cases and high level requirements for the transport network virtualization, OpenFlow control and SDN orchestrator within multi-layer (OPS and flexi-grid OCS) networks*. Public Deliverable. The STRAUSS Project, May 2014. URL: http://www.ict-strauss.eu/deliverables/D3.1.v.3.0_final.pdf.

[STR16] STRAUSS Project. *STRAUSS Project*. 2016. URL: http://www.ict-strauss.eu/en/ (visited on 04/15/2018).

[VFA06] J. Vasseur, A. Farrel, and G. Ash. *A Path Computation Element (PCE)-Based Architecture*. RFC 4655. Aug. 2006. DOI: 10.17487/RFC4655. URL: https://rfc-editor.org/rfc/rfc4655.txt.

[Vis+16] M. Vissers, I. Busi, M. Betts, L. Ong, and G. Zhang. *SDN Architecture for Transport Networks*. Technical Recommendations ONF TR-522. Work in Progress. Open Networking Foundation, Mar. 2016. 17 pp. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN_Architecture_for_Transport_Networks_TR522.pdf.

[VR09] J. Vasseur and J.-L. L. Roux. *Path Computation Element (PCE) Communication Protocol (PCEP)*. RFC 5440. Mar. 2009. DOI: 10.17487/RFC5440. URL: https://rfc-editor.org/rfc/rfc5440.txt.

[Zha+16] X. Zhang, A. Sharma, S. Belotti, and T. Cummings. *YANG Models for the Northbound Interface of a Transport Network Controller: Requirements and Gap Analysis*. Internet-Draft draft-zhang-ccamp-transport-yang-gap-analysis-01. Work in Progress. Internet Engineering Task Force, Oct. 2016. 18 pp. URL: https://tools.ietf.org/html/draft-zhang-ccamp-transport-yang-gap-analysis-01.

[Zus06] Zuse-Institute Berlin (ZIB). *SNDlib*. 2006. URL: `http://sndlib.zib.de/home.action` (visited on 04/15/2018).

Papers with own contribution

[Agu+15] A. Aguado et al. “Dynamic virtual network reconfiguration over SDN orchestrated multi-technology optical transport domains”. In: *Optical Communication (ECOC), 2015 European Conference on*. Sept. 2015, pp. 1–3. DOI: 10.1109/ECOC.2015.7341834.

[Agu+16] A. Aguado et al. “Dynamic Virtual Network Reconfiguration Over SDN Orchestrated Multitechnology Optical Transport Domains”. In: *Journal of Lightwave Technology* 34.8 (Apr. 2016), pp. 1933–1938. ISSN: 0733-8724. DOI: 10.1109/JLT.2016.2522823.

[Agu+17] A. Aguado, V. Lopez, J. Martinez-Mateo, T. Szyrkowiec, A. Autenrieth, M. Peev, D. Lopez, and V. Martin. “Hybrid Conventional and Quantum Security for Software Defined and Virtualized Networks”. In: *J. Opt. Commun. Netw.* 9.10 (Oct. 2017), pp. 819–825. DOI: 10.1364/JOCN.9.000819.

[Aut+14] A. Autenrieth, T. Szyrkowiec, K. Grobe, J. P. Elbers, P. Kaczmarek, P. Kostecki, and W. Kellerer. “Evaluation of virtualization models for optical connectivity service providers”. In: *Optical Network Design and Modeling, 2014 International Conference on*. May 2014, pp. 264–268.

[Aut+15] A. Autenrieth, J. P. Elbers, T. Szyrkowiec, P. Kaczmarek, and W. Kellerer. “Optical network programmability - Requirements and applications”. In: *2015 International Conference on Photonics in Switching (PS)*. Sept. 2015, pp. 321–323. DOI: 10.1109/PS.2015.7329040.

[Cha+17] M. Chamania, T. Szyrkowiec, M. Santuari, D. Siracusa, A. Autenrieth, V. Lopez, P. Sköldström, and S. Junique. “Intent-Based In-flight Service Encryption in Multi-Layer Transport Networks”. In: *Optical Fiber Communication Conference*. Optical Society of America, 2017, Tu3L.10. DOI: 10.1364/OFC.2017.Tu3L.10.

[Fàb+17] J. M. Fàbrega et al. “Demonstration of Adaptive SDN Orchestration: A Real-Time Congestion-Aware Services Provisioning Over OFDM-Based 400G OPS and Flexi-WDM OCS”. In: *Journal of Lightwave Technology* 35.3 (Feb. 2017), pp. 506–512. ISSN: 0733-8724. DOI: 10.1109/JLT.2017.2655418.

[Lop+15] V. Lopez et al. “Demonstration of SDN orchestration in optical multi-vendor scenarios”. In: *Optical Fiber Communications Conference and Exhibition (OFC), 2015*. Mar. 2015, pp. 1–3. DOI: 10.1364/OFC.2015.Th2A.41.

[Lop+16a] V. Lopez et al. "The Role of SDN in Application Centric IP and Optical Networks". In: *Journal of Green Engineering* 6.3 (July 2016), pp. 317–336. DOI: 10.13052/jge1904-4720.634.

[Lop+16b] V. Lopez et al. "The role of SDN in application centric IP and optical networks". In: *2016 European Conference on Networks and Communications (EuCNC)*. June 2016, pp. 138–142. DOI: 10.1109/EuCNC.2016.7561020.

[Lop+17b] V. Lopez, J. P. F. Palacios, T. Szyrkowiec, and M. Chamania. "Multi-layer resilience schemes and their control plane support". In: *DRCN 2017 - Design of Reliable Communication Networks; 13th International Conference*. Mar. 2017, pp. 1–7.

[Mao+16] I. Maor et al. "First demonstration of SDN-controlled Multi-Layer Restoration and its advantage over Optical Restoration". In: *ECOC 2016; 42nd European Conference on Optical Communication*. Sept. 2016, pp. 1–3.

[Mar+17] R. Martinez, A. Mayoral, R. Vilalta, R. Casellas, R. Munoz, S. Pachnicke, T. Szyrkowiec, and A. Autenrieth. "Integrated SDN/NFV Orchestration for the Dynamic Deployment of Mobile Virtual Backhaul Networks Over a Multilayer (Packet/Optical) Aggregation Infrastructure". In: *J. Opt. Commun. Netw.* 9.2 (Feb. 2017), A135–A142. DOI: 10.1364/JOCN.9.00A135.

[May+16] A. Mayoral et al. "First experimental demonstration of a distributed cloud and heterogeneous network orchestration with a common Transport API for E2E services with QoS". In: *2016 Optical Fiber Communications Conference and Exhibition (OFC)*. Mar. 2016, pp. 1–3.

[May+17] A. Mayoral et al. "Control Orchestration Protocol: Unified Transport API for Distributed Cloud and Network Orchestration". In: *J. Opt. Commun. Netw.* 9.2 (Feb. 2017), A216–A222. DOI: 10.1364/JOCN.9.00A216.

[Mun+15a] R. Munoz, R. Vilalta, R. Casellas, R. Martinez, T. Szyrkowiec, A. Autenrieth, V. Lopez, and D. Lopez. "Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks [invited]". In: *IEEE/OSA Journal of Optical Communications and Networking* 7.11 (Nov. 2015), B62–B70. ISSN: 1943-0620. DOI: 10.1364/JOCN.7.000B62.

[Mun+15b] R. Munoz, R. Vilalta, R. Casellas, R. Martinez, T. Szyrkowiec, A. Autenrieth, V. Lopez, and D. Lopez. "SDN/NFV orchestration for dynamic deployment of virtual SDN controllers as VNF for multi-tenant optical networks". In: *Optical Fiber Communications Conference and Exhibition (OFC), 2015*. Mar. 2015, pp. 1–3.

[Ped+16] F. Pederzolli et al. "SDN application-centric orchestration for multi-layer transport networks". In: *2016 18th International Conference on Transparent Optical Networks (ICTON)*. July 2016, pp. 1–4. DOI: 10.1109/ICTON.2016.7550670.

[SAE15] T. Szyrkowiec, A. Autenrieth, and J.-P. Elbers. "Toward flexible optical networks with Transport-SDN". In: *it - Information Technology* 57.5 (Oct. 2015), pp. 295–304. DOI: 10.1515/itit-2015-0020.

[SAK17] T. Szyrkowiec, A. Autenrieth, and W. Kellerer. "Optical Network Models and their Application to Software-Defined Network Management". In: *International Journal of Optics* 2017.5150219 (2017), p. 9. DOI: 10.1155/2017/5150219.

[San+16] M. Santuari, T. Szyrkowiec, M. Chamania, R. Doriguzzi-Corin, V. Lopez, and D. Siracusa. "Policy-based restoration in IP/optical transport networks". In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. June 2016, pp. 357–358. DOI: 10.1109/NETSOFT.2016.7502409.

[Szy+13] T. Szyrkowiec, A. Autenrieth, P. Gunning, P. Wright, A. Lord, J. P. Elbers, and A. Lumb. "First field demonstration of cloud datacenter workflow automation employing dynamic optical transport network resources under OpenStack amp; OpenFlow orchestration". In: *Optical Communication (ECOC 2013), 39th European Conference and Exhibition on*. Sept. 2013, pp. 1–3. DOI: 10.1049/cp.2013.1693.

[Szy+14a] T. Szyrkowiec et al. "Demonstration of SDN Based Optical Network Virtualization and Multidomain Service Orchestration". In: *2014 Third European Workshop on Software Defined Networks*. Sept. 2014, pp. 137–138. DOI: 10.1109/EWSDN.2014.31.

[Szy+14b] T. Szyrkowiec, A. Autenrieth, P. Gunning, P. Wright, A. Lord, J.-P. Elbers, and A. Lumb. "First field demonstration of cloud datacenter workflow automation employing dynamic optical transport network resources under OpenStack and OpenFlow orchestration". In: *Opt. Express* 22.3 (Feb. 2014), pp. 2595–2602. DOI: 10.1364/OE.22.002595.

[Szy+16] T. Szyrkowiec, M. Santuari, M. Chamania, D. Siracusa, A. Autenrieth, and V. Lopez. "First Demonstration of an Automatic Multilayer Intent-Based Secure Service Creation by an Open Source SDN Orchestrator". In: *ECOC 2016 - Post Deadline Paper; 42nd European Conference on Optical Communication*. Sept. 2016, pp. 1–3.

[Szy+18] T. Szyrkowiec, M. Santuari, M. Chamania, D. Siracusa, A. Autenrieth, V. Lopez, J. Cho, and W. Kellerer. "Automatic Intent-Based Secure Service Creation Through a Multilayer SDN Network Orchestration". In: *J. Opt. Commun. Netw.* 10.4 (Apr. 2018), pp. 289–297. DOI: 10.1364/JOCN.10.000289.

[Vil+15a] R. Vilalta et al. "Network virtualization controller for abstraction and control of OpenFlow-enabled multi-tenant multi-technology transport networks". In: *Optical Fiber Communications Conference and Exhibition (OFC), 2015*. Mar. 2015, pp. 1–3. DOI: 10.1364/OFC.2015.Th3J.6.

[Vil+15b] R. Vilalta et al. "The need for a Control Orchestration Protocol in research projects on optical networking". In: *Networks and Communications (EuCNC), 2015 European Conference on*. June 2015, pp. 340–344. DOI: 10.1109/EuCNC.2015.7194095.

[Vil+16] R. Vilalta et al. "Peer SDN Orchestration: End-to-End Connectivity Service Provisioning Through Multiple Administrative Domains". In: *ECOC 2016; 42nd European Conference on Optical Communication*. Sept. 2016, pp. 1–3.