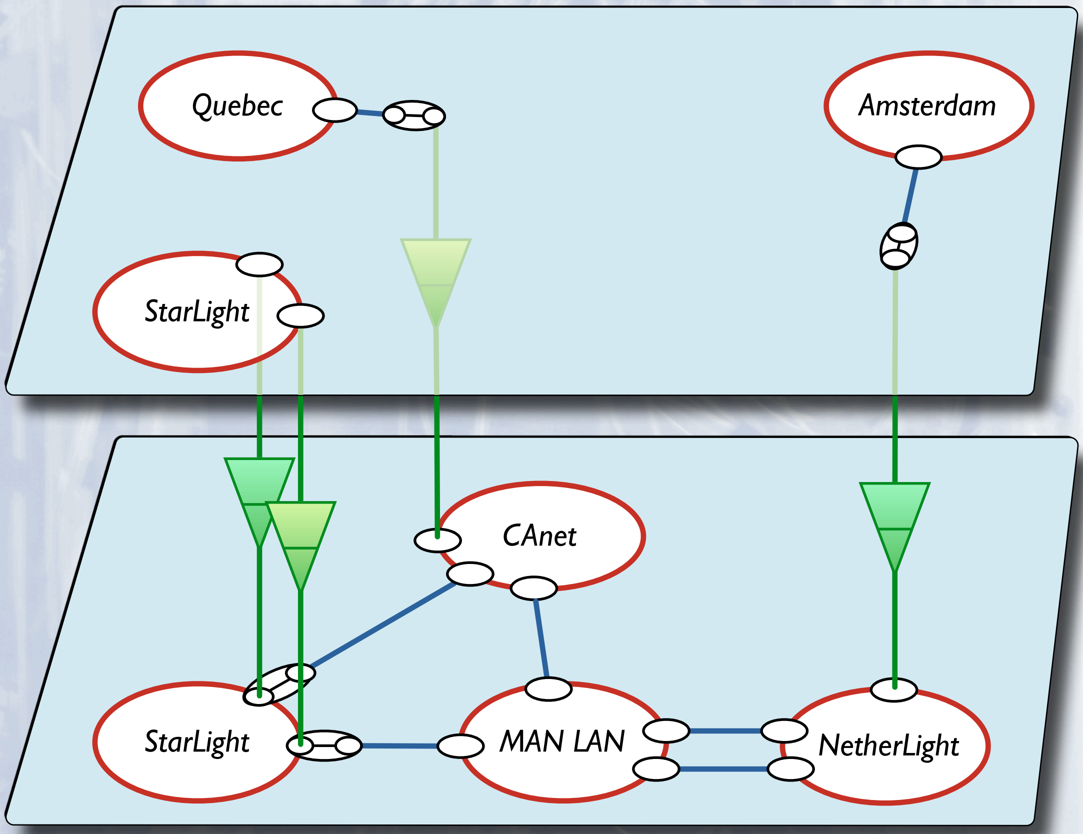
Framework for Path Finding in Multi-Layer Transport Networks



Freek Dijkstra

Framework for Path Finding in Multi-Layer Transport Networks

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
Mw. prof. dr. D.C. van den Boom
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Agnietenkapel
op donderdag 18 juni 2009, te 12.00 uur

door Freek Dijkstra

geboren te Hilversum.

Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Universiteit van Amsterdam
Science Park 904
1098 XH Amsterdam

The investigations were supported by the GigaPort project, which is led by SURFnet and funded by the Dutch Ministry of Economic Affairs under grant number BSIK03020.

Typeset by XƎLaTeX. Printed and bound by your printer.
This thesis is also available for download at `http://www.macfreek.nl/work/`.

ISBN: 978–90–814160–1–6
NUR code: 986   (data communication and networks)

Motto

Hackers, slackers, generation-X backpackers
Tribesmen and rock-stars on silicone screens
Virtual emotions, digital dreams
Rushing for tomorrow where the future used to be

I see the picture; I feel strong
I see the river flowing on

Do you see the picture? Do you still feel strong?
Do you see the river flowing on and on and on?

Blue electric oceans
Information streams
Webs and nets in motion
Connecting you and me

Into the system
Into the light
Into tomorrow
Into the light
Into the future
Into the night

From the lyrics of *Into the Picture* by Johnny Clegg on his album *New World Survivor* (South Africa, 2002)

Contents

Contents

Chapter 1

Introduction

1.1 Computer Networks

Communication networks are ubiquitous in our society: we use them to make phone calls, send e-mails, and browse the web. In all cases an underlying physical infrastructure of wires, fibres, switches and routers provides network services for applications on the network.

Network services can be classified as circuit switched and packet switched network services. The public switched telephone service (PSTN), also known as plain old telephone service (POTS), is a circuit based switching technology. The Internet on the other hand is largely based on packet switched technology. Both technologies have their merits. Packet switched networks are very robust against failures, but can not guarantee a certain quality. Packets switched networks are like highways where all cars use the same roads, and may end up in traffic jams. The analogy for a circuit switched networks would be a dedicated highway between certain origins and destinations. Cars on these dedicated highways may not drive faster, but it is guaranteed that they do not encounter traffic jams.

The work described in this thesis focuses on networks for applications that require better quality of services (QoS) than the regular Internet can offer. Such applications may require so much bandwidth that if they use the regular Internet, it causes congestion and they fail to run smoothly or they disrupt other Internet traffic. These applications require dedicated network connections, such as the circuits in the above analogy. This work masters and manages the complexity of the networks that are offered to these demanding applications.

1.2 e-Science Applications

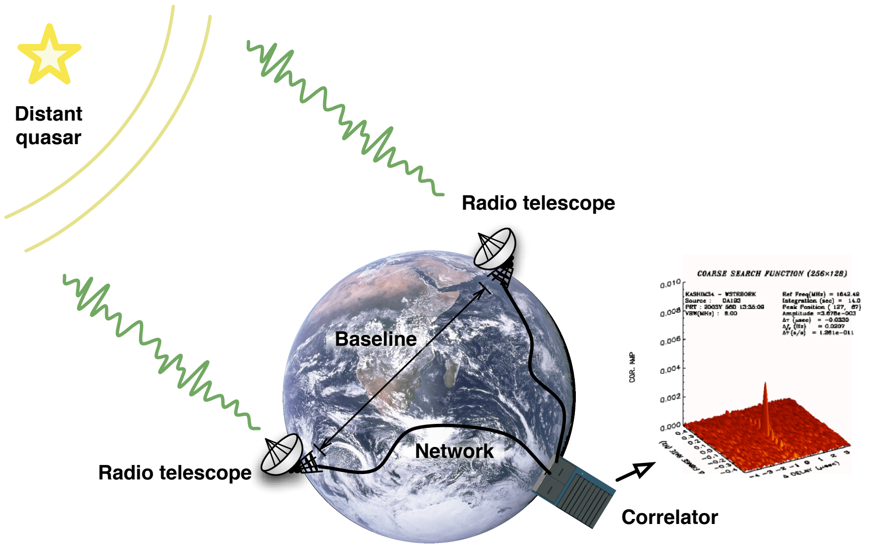


Figure 1.1: *Schematic diagram of very large baseline interferometry (VLBI).*

One of the applications that requires a better service than the regular Internet can offer is very large baseline interferometry (VLBI). Two or more far apart radio telescopes pick up signals from the sky, as shown in figure 1.1. The received data is sent to a correlator for processing. The resolution of the correlated signal improves with the distance between the telescopes. Ideally, the telescopes are located on different continents.

Historically, the data is shipped on tape from the telescope to the correlator. Experiments in 2004 have shown that the data can be transmitted over networks [p38, p42, p34]. Transmitting the data in real time requires a bandwidth of 1 to 10 Gb/s. Since the raw measured signal is nearly white noise, it can not be compressed.

VLBI with data transport over a network (e-VLBI) is an example of an *e-science application*, a scientific application that heavily relies on computer networks [s54]. Sending the data over the regular Internet is not always possible for e-science applications [p23, a11]. This means that the data must be sent over a dedicated network connection.

Typical observation times for telescopes are in the order of hours, and different experiments may link different telescopes together. It is undesirable to change the hardware to reconfigure the network topology for each experiment. Ideally, radio astronomers create a dedicated network connection in software for each experiment with the same ease as it takes to establish a telephone connection.

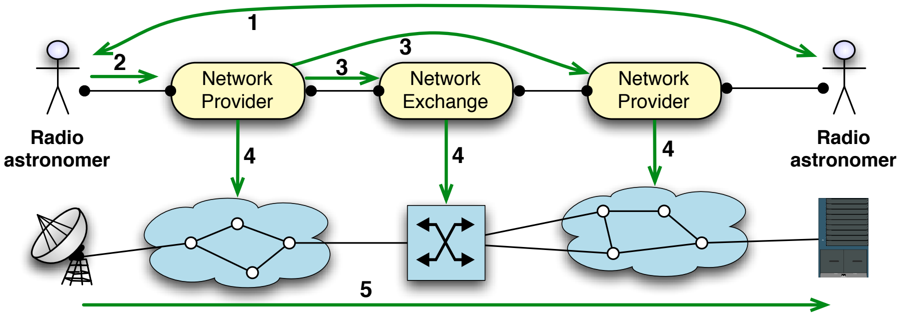


Figure 1.2: *Steps required to set up a network connection between two radio astronomers.*

Figure 1.2 shows the steps that needs to taken to establish a network connection between a radio telescope and a correlator.

The first step is to decide on the end-points, and the characteristics of the required network connection (e.g. the amount of bandwidth required). Secondly, the astronomer makes a request to his or her network provider. The network provider then must find a path that falls within the specified parameters. If the source and destination of the required connections fall in different administrative network domains, the involved domains must collaborate in finding a path. After a valid path is found, the fourth step is to provision this path in the network. Finally, the astronomer can use the network connection to transport his or her data from the telescope to the correlator.

The third step in this process, finding a valid path from source to destination, is covered in this thesis.

1.3 Hybrid Networking

The idea of providing e-science applications with deterministic point-to-point connections was fostered by a community of research networks, later organised in the Global Lambda Integrated Facility (GLIF) [u8]. The ideas in this community led to the concept of *hybrid networking*, the offering of both packet switched (IP) services as well as circuit switched connections over the same physical network infrastructure, the transport networks.

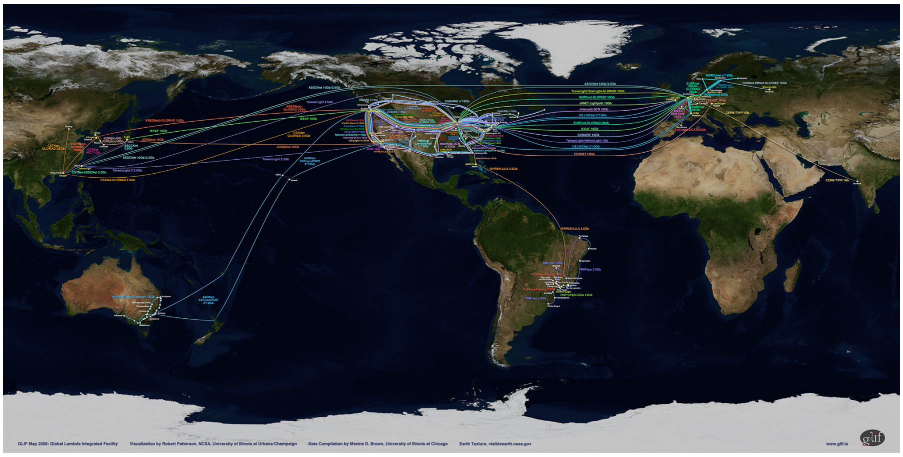

GLIF Map 2008: Global Lambda Integrated Facility
Visualization by Robert Patterson, NCSA, University of Illinois at Urbana-Champaign
Data Compilation by Maxine D. Brown, University of Illinois at Chicago
Earth Texture, visibleearth.nasa.gov
www.glif.is
glif

Figure 1.3: *GLIF world map of May 2008, with all network connection offered by NRENs participating in the GLIF. Source: Patterson, Brown [u4].*

Since most e-science applications operate in a large-scale environment, with collaborators at different universities, the networks required for these applications are nearly always multi-domain networks. De Laat estimated in 2000 that a typical network connection for a physics experiment crosses seven domains [p22]. To achieve interdomain operation, the different networks have to collaborate. For dedicated network connections, this collaboration is done in the GLIF community. In a few years time a few dozen international network connections have been established to provide the interdomain connectivity. Figure 1.3 shows a collection of the interconnections provided by partners in the GLIF community as of May 2008.

1.4 Research Overview

The concept of circuit switched networking potentially puts users and applications in the driver seat. Grid applications already treat computing and storage as dynamic resources, and they may want to treat the network as a dynamic and manageable resource.

The paradigm shift to offer dedicated network services to specific applications has lead to many new questions, including questions on usage models, implementation models, manageability and interoperability of multi-layer networks. Common models, shared by all parties are especially important, given that applications must now interface with the network, and network connections cross multiple domains. All parties involved must somehow share information and act accordingly.

Rather than reinventing the wheel, network engineers turn to existing solutions to model and manage hybrid networks and their circuits.

This brings us to the main research question in this field: **Is there a fundamental difference between hybrid networks and the Internet or the telephony network? Can existing models and algorithms be reused or should new models and algorithms be developed?** Section 1.5.2 shows which part of this research is covered in this thesis.

The final goal of this research is to fully automate the use of dedicated network connections, as sketched in the radio astronomy example of section 1.2.

The GLIF community provides useful input to break down the research question in smaller questions. In particular the issues analysis provided by Bos *et al.* [t6] gives an excellent overview of the problems encountered in the field.

In addition, we can draw from our own experience in setting up and using dedicated network connects across the globe [p13]. The Dutch national supercomputing centre SARA [u13] also provided us with insights [p29]. From these experiences, we can categorise these problems by simply looking at the steps that are typically taken when the user or an e-science application wants to get a dedicated network connection:

1. The user must formulate the requirements, including the end points and the network characteristics like bandwidth, latency, jitter, minimum packet size (if applicable), reliability, etc.

2. These requirements must be communicated to their upstream network provider, usually the national research and education network (NREN).

3. The network provider must gather information about available resources, including the resources in other networks, as the two end-points are typically in different networks (the multi-domain aspect of the question).

4. The network provider must, in collaboration with the other network owners, determine a valid path that uses available resources, and is within the specs of the user.

5. The resources needed for the path must be reserved for all involved networks.

6. The reserved resources must be configured in the networks.

7. The end-to-end path must be tested, and in case of faults the faults must be examined and resolved.

8. The network provider informs the user, and the user must configure the end nodes (e.g. configure the IP addresses and set the routing table).

9. The user runs the applications.

This short overview is not complete and makes a few assumptions. First of all, it assumes that the network provider of the user takes a central role in orchestrating the available network resources across domains. This is called the *master contractor* role, and is similar to the role of an agent who acts on behalf of the user [t11, s14]. Secondly, this process only deals with the provisioning (set up), not with the reconfiguration and deprovisioning (tear down) or the service monitoring after it is in service. For instance, it does not deal with protection, recovery and restoration. Finally, this overview is focused at the administrative control of the network, not at technological developments of the data transport itself.

We can organise the above steps in three categories:

Architectural work, including the design of the network, modelling of the data plane and control plane, and network descriptions to share the information between each entity. This is required before a user can even formulate a question for a network connection.

Path configuration, the actual path finding and provisioning of the network connections. This includes the process of debugging, and monitoring the state of the network and sharing that state with neighbours as input for a scheduling mechanism or for debugging purposes.

Application usage, the use of a network connection by an application. This includes the transport protocols, node addressing and co-allocation of different kind of resources (network, CPU, storage, etc.)

Figure 1.4 gives a schematic overview of the different topics in our field of research. Solid arrows represent dependencies between the topics. The boxes represent topics and subdivisions of a larger topic in subtopics.

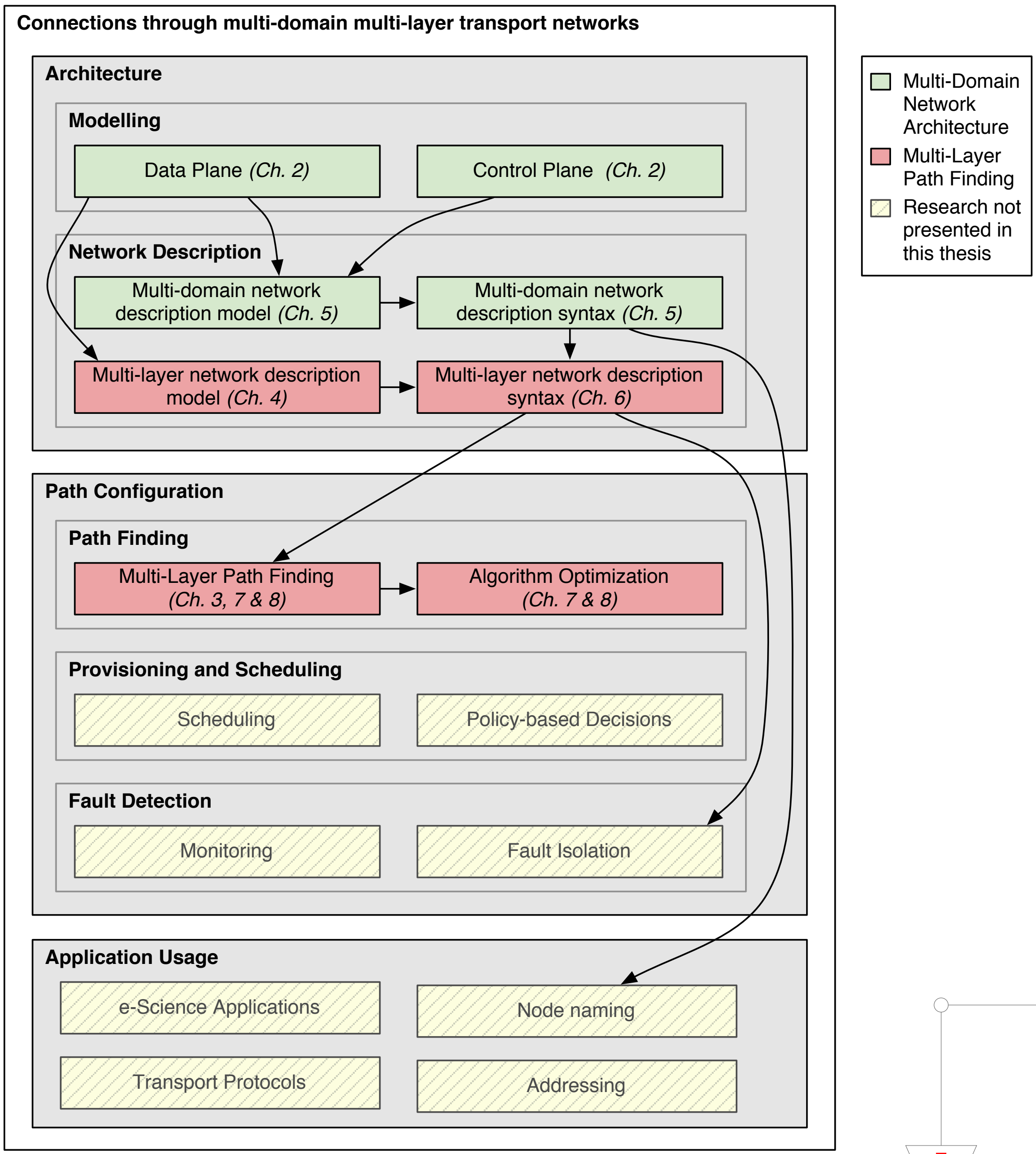

Connections through multi-domain multi-layer transport networks
Architecture
Modelling
Data Plane (Ch. 2)
Control Plane (Ch. 2)
Network Description
Multi-domain network description model (Ch. 5)
Multi-domain network description syntax (Ch. 5)
Multi-layer network description model (Ch. 4)
Multi-layer network description syntax (Ch. 6)
Path Configuration
Path Finding
Multi-Layer Path Finding (Ch. 3, 7 & 8)
Algorithm Optimization (Ch. 7 & 8)
Provisioning and Scheduling
Scheduling
Policy-based Decisions
Fault Detection
Monitoring
Fault Isolation
Application Usage
e-Science Applications
Node naming
Transport Protocols
Addressing
Multi-Domain Network Architecture
Multi-Layer Path Finding
Research not presented in this thesis

Figure 1.4: *Topics covered in this manuscript compared to the research field, and dependency relations between the various topics.* Ch. *denotes the chapters.*

1.5 Thesis Overview

1.5.1 Papers and Topics Covered

Figure 1.4 not only gives an overview of the research field as a whole, but also shows the contribution of this thesis to the field.

The colours in figure 1.4 denote the sections in this manuscript:

Slanted Light Yellow boxes represent work not covered in this thesis.

Light Green boxes represent topics on *Multi-domain Network Architecture*. This part contains mostly the architectural work.

Dark Red boxes represent topics on *Multi-layer Path Finding*. This part includes path finding work, as well as the architectural work that is related to creating model for path finding in multi-layer networks.

The reason I focus myself to architectural problems and multi-layer path finding is twofold. First of all, there has been no common architectural model used by the different researchers and network providers, and this has hampered interoperability across domains, even though this interoperability is required for most applications. Current tools for provisioning circuit switched networks found out the hard way [t13], and only recently is there a move towards standardisation and interoperability in communities like the GLIF, OGF and GÉANT [u2, t17, t9]. During the course of our research, we were surprised to find that there was only very little research on modelling multi-layer computer networks. The only model in broad use is graph theory, but it turns out that this can only be used for single-layer computer networks.

The second motivation is that there currently are not many multi-layer path finding algorithms, so there is no baseline to begin with. Rather than trying a heuristic approach (which may be the best solution in the long run), we attempted to establish a baseline by first giving an exact algorithm.

Table 1.1 gives references to all my co-authored papers, including topics not covered in this thesis. In addition, this table lists which articles have been at the basis of each chapter.

The work in chapter 5 has been done in collaboration with Van der Ham [a3, a9, p16] and the work in chapter 7 in collaboration with Kuipers [a12].

1.5.2 Research Question

The goal of this thesis is to answer the following research question:

Is it possible to use the same path finding algorithms in multi-layer transport networks as those in use for the Internet and telephony networks? If not, what kind of algorithm is required?

This question will be covered in detail in chapter 3.

Architecture	**Papers**	**Thesis Chapter**
Modelling		
Data Plane	[a1]	Chapter 2
Control Plane	[a4]	Chapter 2
Network Description		
Multi-domain network description model	[a3]	Chapter 5
Multi-domain network description syntax	[a9]	Chapter 5
Multi-layer network description model	[a10]	Chapter 4
Multi-layer network description syntax	-	Chapter 6
Path Setup		
Path Finding		
Multi-Layer Path Finding	[a12, a13, o8]	Chapters 3, 7 and 8
Algorithm Optimization	[a12, o8]	Chapter 7 and 8
Provisioning		
Scheduling	-	-
Policy-based Decisions	[o3, o6]	-
Signalling	-	-
Fault Detection		
Monitoring	-	-
Fault Isolation	-	-
Application Usage		
e-Science applications	[a11]	-
Transport protocols	[o5, o4]	-
Node naming	-	Chapter 5
Addressing	[o7]	-

Table 1.1: *Relation between research topics, published papers and chapters in this thesis.*

1.5.3 Methodology

The goal of our work is to find fundamental differences between the architecture of circuit switched networks and other network models, such as graphs or models in use for the Internet, and telephony network. This means that we first must make a model to describe circuit switched networks.

The first step we take in this modelling is to define a concise terminology. If possible, we re-use terminology. If that fails, we turn to network standards. We validate our work by comparing our model with practical applications and actual networks. In particular, we use the networks in the GLIF community to validate our models and principles. If our model is not consistent with the use in the GLIF community, we modify our model until it is.

The creation of a model is partly a logical deduction from relations between terminology, and partly an engineering craft. The scientific value lays in the understanding of the subject at hand – in this case a certain type of computer networks. To make the scientific output more clear we make series of claims closely related to the research question defined in section 1.5.2.

We will prove our claims using any of the standard logical means available to use. For example, a negative claim can be proven using a counter example. Positive claims are harder to prove. In general, we prove those using an implementation that shows it can be done. The risk of such proof is that we inadvertently only prove a special case, rather than the general case. We solve that by using strict boundary conditions.

1.5.4 Chapter Outline

The first chapter of this thesis describes the larger context of the transport networks, and gives the basic model for these networks.

Chapter 2, Optical Exchanges looks at the design of optical exchanges and the difference between Internet exchanges. This chapter makes two claims: **Exchanges can only be ignored during pathfinding if the connections through an exchange are modelled as direct connections, and the exchange does not define a usage policy on its own**. This chapter also defines the terminology used later in this thesis.

The main body of this thesis (chapters 3 to 8) builds upon the observation in chapter 2 that transport networks are inherently multi-layer networks.

Chapter 3, Going in Loops: Path Finding in Multi-Layer Networks introduces the problem of path finding in multi-layer networks, and makes the claim: **Link-constrained algorithms are not sufficient for path finding in multi-layer networks, if links are 1:1 mapped to edges**. This is interesting, since most solutions use such a heuristic link-constrained algorithm. This claim is proven by a counter example. In addition, this chapter claims that for all practical purposes, **graphs can not be used for path finding in multi-layer networks**, even though it is very common to use graphs to describe single layer networks.

Chapter 3 also makes the claim that **Path-constrained algorithms *are* sufficient for path finding in multi-layer networks**. This is proven by an implementation, which is covered in chapters 4, 6, 7 and 8.

Chapter 4, Multi-Layer Network Model introduces a model to describe multi-layer networks, based on ITU-T recommendation G.805 and the label concept in Generalized Multi-Protocol Label Switching (GMPLS). It thus provides an alternative to graphs, which can not be used for multi-layer networks.

Chapter 5, Network Description Language builds upon the terminology in chapter 2 and forms the basis for the research presented in chapter 6. This chapter claims that **it is possible to create a distributed network description, without a central repository**. It shows this by building on the semantic web technology. En passant, this work solves the issue of naming of end-nodes by using uniform resource identifiers (URIs). This work is a collaboration between Jeroen van der Ham and myself.

Chapter 6, Multi-Layer Network Description Syntax describes a syntax and implementation of the model described in chapter 4, and examines if the model is sufficient to describe all technologies that are in use in the networks within the GLIF community.

Chapter 7, Path Finding Algorithms introduces two path-constrained algorithms that can be used to find path in multi-layer networks.

Chapter 8, Path Finding Implementation shows how one of the algorithms of chapter 7, applied to any network with the technologies of chapter 6 indeed is capable of finding paths through a multi-layer network, proving the claim that **Path-constrained algorithms *are* sufficient for path finding in multi-layer networks**.

Finally, this thesis discusses and summarises the results in a concluding chapter.

Chapter 9, Discussion and Conclusion summarises the claims and research question in the individual chapters and concludes this thesis.

Chapter 2

Optical Exchanges

This chapter is based on *Optical Exchanges* by F. Dijkstra and C.T.A.M. de Laat [a1] and on *A Terminology for Control Models at Optical and Internet Exchanges* by F. Dijkstra, B. van Oudenaarde, B. Andree, L. Gommans, P. Grosso, J. van der Ham, K. Koymans and C. de Laat [a4]

In the introduction chapter, we have seen that national research and education networks (NRENs) responded to the demand for dedicated network connections by building hybrid networks, which provide both packet switched IP connections as well as circuit switched point-to-point connections.

This chapter dives into the design of these networks. In doing so, it establishes a model and terminology that is later used in chapter 5. Section 2.1 of this chapter introduces the concept of optical and hybrid networks.

The rest of this chapter is focussed on the interconnection points between existing hybrid networks, the network exchanges. By comparing the functions and services between Internet exchanges on one hand and so-called optical exchanges that are deployed in emerging transport networks, the differences and commonalities will become clear. During this comparison, this chapter answers the following side question: **Can optical exchanges, just like Internet exchanges, be completely transparent to a path finding algorithm in circuit switched networks?**.

The layout of this chapter is as follows. Section 2.2 examines the functional differences between exchanges for optical networks, and existing Internet exchanges. Section 2.3 discusses how technological incompatibilities affects functions and services of an optical exchange. Section 2.4 introduces a terminology for the owner and operator of exchanges, which facilitate descriptions of exchanges. Section 2.5 answers the research question and section 2.6 ends with conclusions.

2.1 Network Terminology

2.1.1 Photonic Networks

Originally, the term *optical networks* or *photonic networks* was used to denote fibre based networks, with laser light as the transmission technology. Examples of an optical transport technology are optical carriers (OC) in synchronous optical network (SONET) [s1] and wavelengths in wavelength division multiplexing (WDM). Articles dealing with optical transmission as well as some network design articles such as Saleh *et al.* [p35] use the term *optical* in its original sense, photonics. Others use *optical* to not only describe optical transmission such as optical carriers in synchronous optical network (SONET) [s1], but also use optical to describe the electrical components in otherwise optical networks, such as the SONET switches, resulting in some confusion.

2.1.2 Optical Networks and Transport Networks

Because of the wavelength division multiplexing (WDM) capabilities of optical networks, the term *optical network* was also used to describe networks that could provision dedicated network circuits. The term *lambda* (after the symbol λ for wavelength), which originally meant a single wavelength on a fibre [p5], started to be used for any link segment, and the term *lightpath* was used to refer to end-to-end circuits over these networks. This idea exists at least since 1992 [p7]. Optical BGP as defined by St. Arnaud *et al.* [s34] clearly uses *optical* in a non-photonic sense. Nowadays, the scope of the term has so broadened that it also includes non-optical technologies such as Ethernet. Authors such as DeFanti, De Laat and Smarr use the term *optical networks* to refer to networks providing lightpaths, circuit-switched based services [p37, p23, t16, p10, t15]. This is consistent with the use of word *optical*, *lambda* and *light* in projects like OptIPuter, Global Lambda Integrated Facility (GLIF), NetherLight, StarLight, etc. [p36, u8, u11, u15].

To avoid confusion with photonic networks and actual optical transmission technologies, we will use the term *transport networks* to refer to networks providing circuit-switched based services, regardless of the use of optical or other transmission technologies.

Transport networks can be created with any circuit-switched technology. Example technologies included wavelength division multiplexing (WDM), time division multiplexing (TDM) technologies such as synchronous digital hierarchy (SDH) [s40] and synchronous optical network (SONET) [s1], Optical Transport Network (OTN) [s44, s39], Asynchronous transfer mode (ATM) [s48]. In addition, the connection oriented properties of some packet switched technologies can also be used to create virtual circuits. Examples of the later technologies include MPLS [s15], Ethernet with the use of IEEE 802.1Q tags (VLAN tagged links) [s3] and Ethernet with Provider Backbone Bridge Traffic Engineering (PBB-TE) (the addition of quality of service to VLAN tagged links) [s5].

This thesis will not use the terms 'lightpath' and 'lambda' but use the terms 'dedicated network connection', or simply 'network connection' for end-to-end dedicated network connection and 'link connection' for a segment of a dedicated network connection. This terminology is in line with the one used in the ITU-T G.805 recommendation [s42] (see section 4.3.3 for the precise definitions).

Whereas a lightpath refers to a single network connection, a collection of lightpaths for the same user or organisation is referred to as an *optical private network* (OPN) for that organisation. Just like a virtual private network (VPN) is an overlay network over the regular Internet, an optical private network is an overlay network over a transport network.

2.1.3 Hybrid Networks

National research and education networks (NRENs) such as SURFnet (Netherlands) and Canarie (Canada) have responded to the need for dedicated network connections by building hybrid networks [p25, p13]. *Hybrid networks* in this context are networks that provide both regular Internet as well as point-to-point connections using the same physical infrastructure. Figure 2.1 shows a basic layer stack for such hybrid networks.

Example implementations of hybrid networks include SURFnet6 (Netherlands), CAnet4 (Canada) and National Lambda Rail (USA) [t4, u10, u6].

This use of the term *hybrid network* as a network providing both packet and circuit based services, should not be confused with the same term meaning partial wired and wireless networks, as can be commonly found in the literature.

Hybrid networks utilise the property that most computer networks are *layered*, where services on a lower layer are offered to a client on a higher layer. For example, in figure 2.1 the fibre infrastructure provides services to

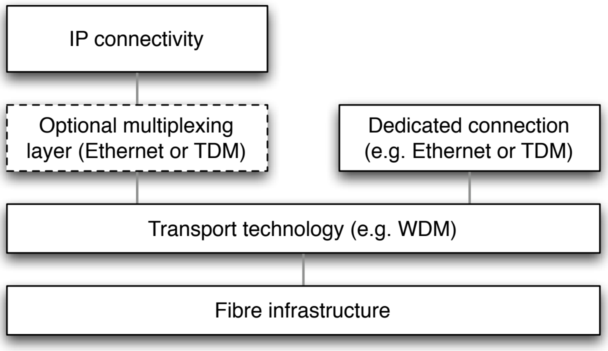

IP connectivity
Optional multiplexing layer (Ethernet or TDM)
Dedicated connection (e.g. Ethernet or TDM)
Transport technology (e.g. WDM)
Fibre infrastructure

Figure 2.1: *Dual stack for a hybrid network example: the underlying infrastructure provides services at multiple layers.*

the Wavelenght Division Multiplexing (WDM) layer. The OSI model [s53] is a multi-layer model that is commonly referred to in network modelling, although the actual layer stack that it defines has hardly gained any use. The TCP-IP network stack has been more common since the rise of the Internet. Unless otherwise noted, the term *multi-layer network* in this thesis refers to a network that is *configurable at multiple layers*. Hybrid networks are an example of a network which can be reconfigured both at a circuit switched layer and the IP layer.

2.2 Exchanges

The Internet is literally a collection of interconnected networks. Due to economic advantages the interconnections between networks often cluster at specific locations [p6, p18, p28]. This argument holds for all interconnections between networks, not just the regular Internet.

This section examines the commonalities and differences between interconnection points in the Internet and in transport networks.

2.2.1 Peering, Exchanges and Members

Peering, in most literature, is limited to providing connectivity to each other's networks and to the customers of the other network, but not to other destinations. *Transit* on the other hand implies that the traffic is handled for all destinations, usually for a fee [p28]. In this thesis, we do not differentiate between these relations, and simply speak of *peering* to refer to the exchange of data between two networks.

The most trivial interconnection point is a co-location that provides no other functionality than rack space and power. This already gives the networks at the co-location the ability to initiate bilateral peerings with other networks at the same facility.

Our research focus is not these simple co-location facilities, but on *exchanges*, networks dedicated to facilitate the interconnections between different networks. We use the term *member* for the networks connected to an exchange. In our terminology *member* does not imply an organisational structure of the exchange. The term is common in the Internet exchange community. The GLIF community has no particular term, although *autonomous optical domains* is used in some descriptions.

2.2.2 Classification

Based on the functions, rather than the technical implementation, we observe four types of interconnection points:

- Internet exchange points (IXP),
- Mobile roaming exchanges (MRX),
- GLIF Open Lambda Exchange (GOLE), and
- Points of presence (POP).

An **Internet exchange point** (IXP), or simply *Internet exchange* serves as an interconnection point to exchange packet data between individual members. The members have one or a few physical connections to a central core infrastructure. The core network can be Ethernet LAN, SONET, ATM, or MPLS-based. The Ethernet variant is by far the most common variant [t2] and is stateless, while the other three are stateful and require that the individual members set up a path between them. Such a path is a channel in the physical connection. Internet exchanges are known in the USA as *Network Access Points* (NAP).

Mobile roaming exchanges (MRX), such as GPRS roaming exchanges [p4] and UMTS exchanges, exchange packet data for respectively 2.5th and 3rd (3G) generation mobile telephony. In telecommunications, however, the term *exchange* is different from the usage in this thesis and refers to a transit provider rather than an interconnection point. An exchange point between mobile roaming exchanges is technically not different from a packet-based[1] Internet exchange.

GLIF Open Lambda Exchanges are interconnection points where members exchange traffic at OSI layer 1. The switched circuits through the exchanges are part of larger circuits – the lightpaths. Most GLIF Open Lambda Exchanges (GOLEs) use SONET/SDH technology to provide the circuits, although some create pseudo-circuits using Ethernet VLANs. The term *GOLE* is used within the GLIF community since 2005. Examples of GOLEs are: NetherLight in Amsterdam, StarLight in Chicago, MAN LAN in New York, T-LEX in Tokyo, HKOEP in Hong Kong, UKLight in London, NorthernLight and Pacific Wave [u8]. The later two exchanges are geographically distributed (along Scandinavia and the East coast of the USA respectively).

In this thesis, we use the term *optical exchanges* [a1] instead of GOLE since that term will also be recognised outside the GLIF community. Optical refers to optical networks, and does not require that the switching technology is purely photonic. Optical exchanges are also known as grid exchange points (GXP), since they focus on switching circuits on transport networks for use in Grid-based applications. GMPLS Internet exchanges (GMPLS-IX) as defined by Tomic and Jukan [p39] share the concept of circuit-switched interconnection points, but have not been implemented yet.

Points of presence (POP) are interconnection points where access networks connect with a transit network provider. While the transit network provider may connect to multiple networks, we do not regard it as an exchange, since the function of the transit network is not primarily to facilitate data exchange between the members, but between a member and itself.

We will examine Internet exchanges and optical exchanges in more detail, and see where they differ in function and technology.

2.2.3 Internet Exchanges

There are currently three types of Internet exchanges [p18, p27] defined:

LAN-based Internet exchanges: The most common exchange [t2], creating a Local Area Network (LAN) that connects the routers of the member networks with each other. The network is usually a star topology with a layer 2 Ethernet switch at the core, though earlier variants had a ring topology (based on Fiber Distributed Data Interface). This is the only stateless exchange. Congestion is possible if multiple members want to send traffic to the same member at the same time.

[1]GPRS and UMTS are packet based. The older CSD system is circuit switched.

ATM-based Internet exchanges: A stateful exchange, with an asynchronous transfer mode (ATM) network. The member networks are connected with each other with permanent virtual circuits (PVCs) through the ATM network. PVCs in an ATM network can either be variable bit rate or constant bit rate [s49]. If variable bit rate (VBR) circuits are used, there is no guaranteed congestion-free transmission. Usage of constant bit rate (CBR) on the other hand, results in very inefficient usage of resources and poor scalability.

MPLS-based Internet exchanges: Stateful packet based exchange, based on Multiprotocol Label Switching (MPLS) [s15]. The destination of incoming packets is determined by a lookup at the IP routing table at the ingress edge label switching router (LSR). This is a relative expensive operation for very-high bandwidth data streams.

Given their properties, these types of exchanges are not designed to support few high bandwidth flows that do not require routing. Either it is technically impossible (for LAN-based Internet exchanges) or it yields unnecessary and costly overhead (for ATM- and MPLS-based Internet exchanges).

2.2.4 Internet versus Optical Exchanges

Table 2.1 highlights the differences between Internet exchanges and optical exchanges.

Members at an Internet exchange interconnect to exchange IP traffic. This is shown at the first two rows in the Internet exchange column in table 2.1. An Internet exchange contains exactly one circuit per peering relation between two members. In contrast, a transport network supports circuits between end-users, so at an optical exchange there is a circuit between members for each end-to-end connection that goes through the exchange. The table further emphasises that for an optical exchange, these circuits can carry any layer 1 or layer 2 traffic. Differences in function and purpose lead to different choices in technology between Internet exchanges and optical exchanges. Finally, the table highlights that an optical exchange may offer more advanced services than an Internet exchange.

There is no clear boundary between the different interconnection points since each interconnection point may take multiple roles. We expect that the differences listed in table 2.1 will change over time, as new technologies become available and are implemented. For example, customers at a POP may also directly peer with each other, a function typically seen at exchanges. Circuit

	Internet Exchange	**Optical Exchange**
OSI Layer	Transports traffic at layer 2, members connect with layer 3 devices	Transports traffic at layer 1 or layer 2, members connect at that same layer.
Traffic type	IP traffic only	Any packet data or any data at a specific framing or bit rate
End-points	Connection between two member networks	Connections are part of a larger circuit between two end-hosts
Dynamics	Stateless, or state changes only when (peering) relation between members changes	State changes for each data transport
Technology	Often packet switched, sometimes label-switched (with virtual circuits like MPLS and ATM)	Circuit or virtual-circuit switched (e.g. using SONET or VLANs)
Services	Only data transport	Data transport and other services, like the conversion of data between different formats and layers

Table 2.1: *Functional differences between Internet exchanges and current optical exchanges. These characteristics will change over time, as new technologies become available and are implemented.*

switching is typically associated with optical exchanges, but not a technical necessity: ATM- and MPLS-based Internet exchanges are also circuit switched and it might be possible to create a non-circuit switched optical exchange using optical burst switching (OBS) [p32].

In fact, we have reason to believe that the distinction between Internet exchanges and optical exchanges will disappear as time progresses. First of all, it is economically beneficial to build optical exchanges at a location where a large number of member networks are already present. The location where most networks are present is at Internet exchanges, since networks that connect to optical exchanges are often hybrid networks that are also connected to Internet exchanges. Secondly, Internet exchanges also tend to offer more services that are now regarded as optical exchange functions, like private circuits between two members. For example, the Amsterdam Internet Exchange (AMS-IX) already provides private interconnects and closed user groups [t1].

Third, optical exchanges must often provide all sorts of services, as we will see in section 2.3.3. Multiparty peering like in a LAN-based Internet exchange, is just another service.

2.3 Incompatibilities

2.3.1 Progressing Technology

The interfaces in a network may be of different type. For example, one interface may be used to carry undefined traffic over 32 wavelengths using dense wavelength division multiplexing (DWDM) [s36], while an other interface may only carry one signal at 1310 nanometre, which carries SDH-framed traffic [s40]. A third interface may be LAN-PHY based Ethernet [s6], where traffic must reside in the 192.0.2.0/24 subnet.

Most of these types map to different network layers, and some devices may deal with layer 1 functions, while another device may deal with layer 2 functions. A device that deals with higher layer functions will nevertheless make use of the lower layer functions to transport data on its interfaces. In the above example the SDH framed traffic was carried over a 1310 nanometre wavelength.

This variety of encodings at different network layers would not be a problem if everyone would use the same encoding for each layer. Unfortunately, there are many, possible incompatible ways to transport data. There are multiple ways to carry signals in one fibre using DWDM, due to the variety of wavelength bands and channel spacings. Similarly not all devices encapsulate Ethernet in the same way over a SONET/SDH connection.

Networks can only communicate with each other if they agree on a specific encoding of their data. A common interface may be agreed upon if one technology has been proven to be robust and sufficient for all network operators. If a new network is built, the operator will choose that particular technology. For example, the de-facto standard on the Internet is IP, and alternatives such as the OSI protocols [s53] are no longer used.

If technologies are still in development and new possibilities are introduced on the market as time progresses, network operators may choose to use the new technology instead, which may be incompatible with the technologies from networks that have been built while that technology was not available. This has a severe impact on the design of networks, as networks using two different technologies can not communicate by default, and the data must be translated to an agreed upon encoding.

Technology advances are rather common. For example, during the time this thesis was written, multiple technologies have been used to create dynamic circuits: MPLS [s15], GMPLS [s20], Ethernet VLANs[s3], SONET/SDH [s1, s40], and Ethernet PBB-TE [s5]. Also technologies that were considered stable for decades still change. For IP there is a distinction between IPv4 and IPv6 [s12], and recently the size of AS-identifiers has changed from 2 bytes to 4 bytes [s29], causing potential incompatibilities.

Summarising, we can conclude that **the use of multiple technologies causes incompatibilities**. In addition **technologies evolve over time**, thus **incompatibilities will continue to exist**.

2.3.2 Impact on Optical Exchanges

While technologies evolve, this gives rise to possible incompatibilities, and the design of an exchange should take that into account. Either the exchange should decide on a common technology, or only members with the same technology can communicate, or the exchange must contain a service to enable the interworking between different technologies.

Network engineers avoid technology incompatibilities as much as possible. Indeed, it is very uncommon to see incompatibilities within a single domain, but it is very common to see them across domains. Networks are not built at the same time, and network owners may have different opinions about the best technology. This means that **exchanges must take potential incompatibilities into account to avoid non-working network connections**.

If an exchange must perform one of the services mentioned above, it must also be aware about the possible incompatibilities between the different interfaces, as explained in section 2.3. At a minimum, it must know about the layer and framing of the interface. For example, it must know the difference between 1 Gbit/s and 10 Gbit/s Ethernet links, and must know how to extract the Ethernet frames from the carrier. A more advanced incompatibility is the different ways in which Ethernet can be encapsulated in SDH and SONET channels.

Exchanges try to avoid incompatibilities by standardising the interfaces to the members. For GLIF Open Lambda Exchanges (GOLEs), the optical exchanges in the GLIF community, exchanges either switch individual VC-4 timeslots (for SDH), or use Ethernet VLANs to switch. The embedding of Ethernet channels in VC-4 timeslots is standardised using Framed Generic Framing Procedure (GFP-F) [s45], virtual concatenation (VCAT) [s47] and Link Capacity Adjustment Schema (LCAS) [s46]. The effect of all these standards is that egress timeslots can be chosen independently of the ingress timeslots, and thus are less possible incompatibilities. Nevertheless, some exchanges standardise on Ethernet, others on SDH, and others on GMPLS. Thus technology conversions are still required for end-to-end connections.

2.3.3 Services

Since it is undesirable that only members of an exchange with the same technology can communicate, and because technology conversions are still required for end-to-end connections, exchanges may offer interworking services. The following list gives a partial overview of the different services that may exist for optical exchanges.

Cross connect: The basic functionality of any exchange is to transport traffic from one member to another member. For interfaces of equal type, this can be accomplished using a simple cross connect between the two interfaces.

Adaptation: If a signal goes in at one layer and goes out at a different layer, the optical exchange effectively acts as an 'elevator', lowering or raising traffic to different layers. For example if a signal comes in as a wavelength using DWDM, and is injected in a VLAN, the signal is elevated from layer 1 to layer 2.

Multiplexing: Combining different signals into a single carrier is called multiplexing. The extraction is called de-multiplexing. For example, combining multiple wavelengths into a single fibre.

Interworking: Conversion between different incompatible technologies, for example, conversion between WAN PHY Ethernet and LAN PHY Ethernet or between a wavelength with 50 GHz and 100 GHz spacing.

Wavelength conversion: A special case of Interworking. Changing the colour (wavelength) of a lambda

Signal regeneration: Either simple amplification or attenuation of the power levels to match a certain output power level, or full regeneration consisting of reamplification, reshaping and retiming (3R) of the signal.

Optical multicast: The ability to duplicate an optical signal as-is. This can only be done one-way, not for bidirectional connections (at least not without merging the return signals, or ignoring all but one).

Traffic aggregation: Traffic may be aggregated as packets into a single data stream. This is a special case of multiplexing. Aggregation with (IP or Ethernet) packets, (ATM) cells or optical burst switching may lead to congestion and packet loss.

Store-and-forward: One way to reduce blocking chances is to transport large chunks of data on a hop-by-hop basis, rather than reserving the entire end-to-end path. The data may first be transported along a first segment of the end-to-end path, stored at an intermediate location, and transported along the rest of the path as soon as that section of the path becomes available for the data transport.

2.3.4 Control Plane Services

So far we only discussed data plane services.

The *data plane* of a network is where all data transport functions take place. The control plane of a network is where the operational control takes plane, and externally influenced state changes of the network are processed. The control plane includes the routing and signalling protocols that react to state changes in the network (e.g. links coming up or down, external connectivity announcements), and to connectivity requests by users or applications (e.g. path requests and tear down messages).

The control plane may also contain services for users or applications. These control plane services may include:

Provisioning service: set up or tear down of a network connection based on a user's request

Scheduling service: checking the availability of certain resources and making future reservations;

Authentication service: verifying the usage policy on a resource and authorising their usage;

Index server: listing the available resources;

Broker service: combining different resources together necessary to fulfil an entire request.

The resources in the list are typically network resources, but could be other types of resources, like storage or computational resources.

The services offered by the control plane are sometimes also referred to as the *service plane.*

2.4 Ownership

In this section we will introduce a concise terminology to describe exchanges, in particular the facilitating role of exchanges where the members rather than the exchange decide on the business policy. We need this terminology later in chapter 5.

One feature often quoted by exchanges is that they take a *neutral* role, and do not prefer one member over another. While neutrality is often seen as a commercial aspect, there are also technical aspects at stake. We need to define different roles to distinguish between the different aspects.

We relied as much as possible on existing terminology. In particular, the ownership terminology in section 2.4.1 builds upon the management layers in Telecommunication Management Network (TMN) [s50] and current practice in economic and legal communities [t10], while the term open exchange draws upon discussion in the GLIF community [t8].

2.4.1 Owner, Operator and Users

Network element is a generic term to include network devices, links, interfaces and hosts.

We distinguish between *legal owner*, *economic owner*, *operator* and *user(s)* for each network element. The *legal owner* of a network element is the entity that purchased the device and the *economic owner* is the entity that acquired the usage rights from the legal owner.

The *economic owner* determines its *policy* for the network, a set of rules to control access to the network resources [s16]. This entity carries the responsibility for the behaviour of a device and has the final responsibility in case of hazards and abuse. In addition, each network element can also have a separate *operator*, the person, organisation, or software component that configures and operates the device on behalf of the economic owner. The economic owner determines the policy for a network element; the operator enforces this policy. Finally, the *users* may use or invoke an element, if their request is in compliance with the active policy.

In this thesis, we use the terms *network administrator* and *network operator* interchangeably. This is in line with the Telecommunication Management Network (TMN) standard, which notes that “the expression ‘Administration’ is used for conciseness to indicate both a telecommunication administration and a recognized operating agency”. Network operators in the GLIF community use the term *network administrator* to mean *network economic owner*. To avoid confusion, we prefer the term *network operator*.

We assume that each network element has exactly one legal owner, one economic owner, and one operator, but may have multiple users over time (though typically only one at a specific time).

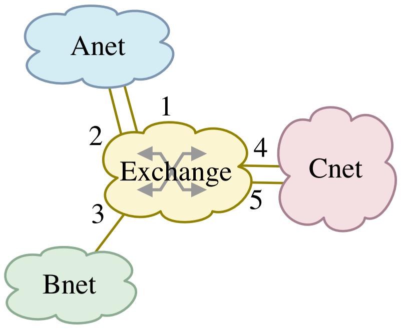
2.4.2 Open Control

Often the legal owner, economic owner, and operator of a network element are the same entity. For example, in the Internet, a transit provider is typically owner and operator of its network. But this is not always the case.

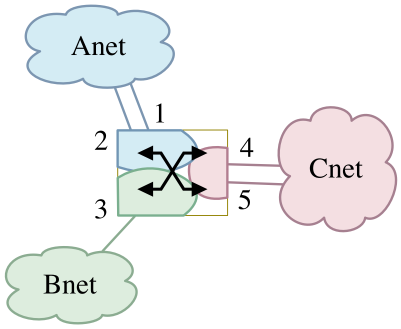
If an organisation leases a trans-oceanic fibre from a carrier to lease a fibre for a year, the carrier is the legal owner, while the other organisation is the economic owner.

If an organisation outsources the maintenance of its network, the economic owner and operator of this network are different entities.

An exchange may be neutral. Rather than applying its own policy, it outsources the policy decision to allow or deny a connection request between two interfaces to the domains that are connected to those interfaces. Therefore, members of an open exchange have the ability to configure 'their' interface in the exchange and thus can decide who connects to their network. We call this concept *open control*, and exchanges that apply such a policy are called *open exchanges*.



(a) *Closed control model*



(b) *Open control model*

Figure 2.2: *Comparison between the closed control model and the open control in the same network.*

Figure 2.2 shows an example of a closed and an open control model. While control model on the left follows the physical topology of the operational domains, the control model on the right follows the *owner domains*. In the open control model, there is no owner domain for the exchange, because not the exchange, but its members decide on policies.

2.4.3 Domains

We define a *domain* as a set of network elements. A set of network elements may even be disjoint. An *owner domain* is a set of network elements with the same economic owner. An *operational domain* is a set of network elements with the same operator. Observe that this definition does not require that an operational domain contains *all* network elements with the same operator.

An *exchange network* is an operational domain within an interconnection point whose primary goal is to transport data between all its members. Exchanges tend to be neutral, and are policy-free. Exchange networks are of special interest throughout this chapter and we use the term *exchange* to refer to a core network and its operator.

2.5 Transparency

So far, we investigated the properties of Internet and optical exchanges in detail. It is time to answer our research question: **Can optical exchanges, just like Internet exchanges, be completely transparent to a path finding algorithm in circuit switched networks?**

The routing protocol on the Internet is the Border Gateway Protocol (BGP) [s24]. It works by distributing reachability paths, consisting of lists of individual networks, the autonomous systems (AS). These lists of AS numbers do only contain autonomous systems with routers. It does not included Internet exchanges, since Internet exchanges typically don't contain a router. Thus, Internet exchanges are transparent for the path finding algorithm in the Internet.

It is desirable that exchanges are neutral. The members of an exchange expect that an exchange has a facilitating role, and does not have its own policy.

An exchange is *transparent* to a path finding algorithm if the path finding algorithm needs no knowledge of the exchange. Thus, if the path finding algorithm regards two members as directly connected. This is true if two conditions are met. First of all, the members must be able to connect at will. Thus, the exchange must have a facilitating role, and not have a policy on its own. Secondly, the connections through the exchange must behave as a direct connection. So, the interfaces must be of the same type and the exchange does not offer any services. **Exchanges can only be ignored during path finding if the connections through an exchange are modelled as direct connections, and the exchange does not define a usage policy on its own**.

Optical exchange can, like Internet exchanges, be transparent. The condition is that the optical exchange does not provide a policy of its own. If the path finding algorithm finds a path between these two domains, then the domains are responsible to signal the exchange to provision a connection between the two members, just as the members are responsible for setting up the connection through their own network. If the optical exchange would have its own policy, then the members can not be sure that the exchange adheres to their request, and the exchange must be visible to the path finding algorithm. Given that exchanges have a facilitating role, this condition is typically true.

The second condition is that a path finding algorithm can treat the connection through the exchange as a direct connection between the members. This condition may not be true: if the two interfaces are of different type, or if the exchange offers services besides simple cross connects, then the exchange can not be regarded as a direct connection, and it is necessary that the path finding algorithm is aware of the exchange.

2.6 Conclusion

In this chapter, we investigated the fundamental differences between the Internet and transport networks. We created a definition framework that makes comparisons between exchanges possible, not only between Internet and optical exchanges, but also between different optical exchanges. Our terminology on circuits and lightpaths, the service overview in section 2.3.3 and the classification in section 2.1 helped the discussion in the community. The distinction between hybrid network and multi-layer network was not obvious before these definitions. Furthermore, We were the first to define the concept of ‘open control’ and relate that the difference between an owner and an operator of a network.

The most obvious distinction between the Internet and transport networks is that the Internet is packet based, while transport networks are circuit based. This difference also has implications for the exchanges in these networks. While Internet exchanges and optical exchanges may use the same technologies (e.g. Ethernet and SONET), an important distinction is that optical exchanges change state with each connection request, while the state of Internet exchanges only change when peering relations change.

Since technologies change over time, and so do networks, incompatibilities will continue to occur. Exchanges must take potential incompatibilities into account to avoid non-working network connections. Services like adaptation and interworking must be offered by exchanges or elsewhere in the network to solve incompatibilities. This last implication means that optical exchanges may offer services, such as interworking between different interfaces, and multiplexing of channels. If such services are offered, they are relevant to a path finding algorithm, and optical exchanges can not be transparent to a path finding algorithm, as Internet exchanges are.

It is possible for an optical exchange to retain its facilitating role, even when it is considered to be a regular domain by a path finding algorithm. In this case, the policy of the domain would *not* be to always grant access, but instead, to delegate the decision making to the affected neighbours.

Chapter 3

Going in Loops: Path Finding in Multi-Layer Networks

This chapter is based on *A Path Finding Implementation for Multi-Layer Networks* by F. Dijkstra, J.J. van der Ham, P. Grosso, and C.T.A.M. de Laat [a13].

The provisioning of circuit switched network connections is a three-step process:

Routing: the distribution of topology and network state across different domains;

Path finding: the calculation of the shortest viable path;

Signalling: the provisioning of the actual network elements across the chosen path.

A routing protocol is a network protocol that is responsible for distributing information about the state of a local network or node to neighbouring networks or nodes. A path finding algorithm will use that information to calculate a shortest path.

This chapters also articulates the main research question of this thesis:

Is it possible to use the same path finding algorithms in multi-layer transport networks as those in use for the Internet and telephony networks? If not, what kind of algorithm is required?

In order to answer this question, we describe a few common protocols in section 3.1, and summarise which algorithms are used in the Internet and telephony networks work in section 3.2. Section 3.3 describes how this differs from path finding in multi-layer networks. Sections 3.4 and 3.5 discuss the different kind of constraints in path finding. Section 3.6 ends this chapter by summarising the conclusions.

3.1 Algorithms

This section describes the most common path finding algorithms in use [p9], the shortest path first (SPF) algorithms. All described algorithms find a path in a graph.

3.1.1 Breadth-first and Depth-first

Breadth first and depth first search algorithms are two distinct approaches to walking down a tree graph, given the root of the tree.

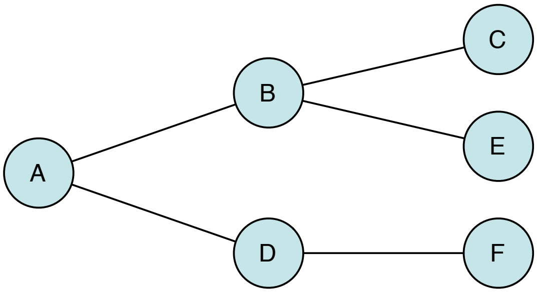


Figure 3.1: *A simple tree graph*

Breadth first search algorithm starts with the root, and first traverses the direct child nodes, before traversing the child nodes of the child nodes. In the tree of figure 3.1, it would traverse the vertices in the order A, B, D, C, E, F.

Depth first search algorithm starts with the root, and traverses child nodes recursively. It first traverses an entire branch of one child before proceeding to another child. In the tree of figure 3.1, it would traverse the vertices in the order A, B, C, E, D, F.

3.1.2 Bellman-Ford and Dijkstra Algorithms

The Bellman-Ford [p3] and Dijkstra [p11] algorithms are both single source shortest path first algorithm published in the late 1950s, and are still in wide use today [p9].

Both algorithms find all shortest paths in a graph, given a certain source vertex. The Bellman-Ford algorithm can deal with vertices with negative weight, while the Dijkstra algorithm can not. The Bellman-Ford algorithm scales with $\mathcal{O}(|E| \times |V|)$, which is worse than the scalability of Dijkstra's algorithm $\mathcal{O}(|E| \times log|V|)$, with $|V|$ the number of vertices in a graph and $|E|$ the number of edges in a graph.

3.1.3 Constrained Shortest Path First

Constrained Shortest Path First (CSPF) algorithms find shortest path fulfilling a certain constraint. For example, only edges with a minimal available bandwidth can be part of the shortest path. The solution is to prune all edges that do not meet the constraint from the graph before calculating the shortest path.

Kuipers et al. published a performance comparison of different constrained shortest path first algorithms [p20].

We have not defined what “shortest” path means. Usually, either the hop count is used (the number of edges or number of vertices in a path), or each edge has a given metric, the “weight” of an edge.

All algorithms above assume that a subsection $B \rightsquigarrow C$ of a shortest path $A \rightsquigarrow D$ is also the shortest path from B to C. This is true if the total metric of a path is the sum of the metric of each individual edge. Kuipers and Van Mieghem developed the SAMCRA algorithm, which can deal with multiple constraints, where this assumption does not have to be valid [p19, p26].

3.1.4 Path-Constraint Algorithms

Constrained shortest path first algorithms only deal with a specific kind of constraint: link-constraints that apply to a particular edge. These constraints can be filtered out in the initialisation phase of the algorithm.

Path-constraints are constraints that are based on a particular combination of multiple edges. This type of constraints can not be solved in the initialisation phase of an algorithm because it does not apply to single edges but to combinations of edges. For example, it may be allowed to use the edge $A - B$, but not in combination with the edge $B - C$.

We distinguish between these two constraints in this thesis by referring to link-constrained path finding algorithms and path-constrained path finding algorithms.

Examples of path-constrained algorithms can be in a routing problem with different forms of transportation. For example, it may always be allowed to travel from A to B on a map by foot, but it is only possible to continue the travel from B by car if there is a car parked at B. Another practical example of such algorithms is in analysing formal languages [p2]. For example, the code of a programming language may only contain an 'end' statement if it was first preceded by a corresponding 'begin' statement

3.1.5 k-Shortest Path

Usually, we are only interested in the shortest path between two nodes. However, in some circumstances, it is desirable to find a list of shortest paths. An algorithm that finds the first k shortest paths is called a k-shortest path algorithm.

3.2 Routing Protocols

3.2.1 Distributed Path Finding

The algorithms in the previous section take a graph as input. Thus, before applying an algorithm, it first needs to have full topology knowledge. While this may be feasible for path finding within a domain, it does not scale to a worldwide solution.

Algorithms can only be applied on a world-wide scale if they are able to do path finding without having full topology knowledge. A common solution is to let each node make a *local* path finding decision based on a limited amount of information, but in such a way that the final result is still a shortest path.

There are three common routing protocols in use on the Internet: link-state routing, path distance routing and path vector routing.

All these routing protocols build a routing table. This table, which is different for each node, contains a list with all destinations and the next hop for the shortest path to that destination.

A link state routing protocol will distribute all topology knowledge. With that information, each node can apply an exact algorithm such as Dijkstra's algorithm for a path finding request, and find the complete end-to-end path.

A host using the distance vector routing protocol does not distribute the full topology knowledge between neighbours, but only announces the distance in which it can reach a certain destination. A host that receives distance information to a particular destination from multiple neighbours only needs to pick the neighbour with the shortest distance. Distance vector routing protocols are a variant of the Bellman-Ford algorithm, and are sometimes referred to as the Ford-Fulkerson algorithm [p12]. Dijkstra's algorithm can not be used with a distance vector protocol.

The path vector routing protocol is very similar to the distance vector routing protocol, but distributes the information about the shortest path next to the distance. The advantage is that this allows easy detection of cycles, and solves the problem that the slow convergence with the distance vector problem if a link goes down.

3.2.2 The Internet

The routing protocol between domains on the Internet, the Border Gateway Protocol (BGP) [s24, s25], uses a path vector algorithm. It keeps a routing table, a list of destinations with the associated next hop. Before a route is added to the routing table, and before it is announced to neighbours, it is filtered using a policy.

Multiple protocol standards exist to distribute routing information within a domain, each with a different routing protocol. OSPF [s10] and IS-IS [s7] are link-state routing protocols, while RIP [s11] is a distance vector routing protocol.

3.2.3 Public Switched Telephone Service

The telephony network reduces the information required by the routing algorithm by standardising on a single technology, so there are no incompatibilities. Scheduling constraints are solved using a route congestion statue, and by massively overprovisiong the system.

The routing algorithm in Signalling System 7 (the control plane of the telephone network) uses a routing table with prefixes (of the telephone number) [s51, s52]. The routing protocol is similar to the is approach is similar to the distance vector routing protocol. Policies (e.g. transfer-prohibited) are distributed across a limited set of nodes.

3.2.4 Generalized Multiprotocol Label Switching

Generalized Multiprotocol Label Switching (GMPLS) is a set of protocols under development by the Internet Engineering Task Force (IETF) for a control plane that supports data forwarding based on labels [s20]. GMPLS supports switching at different technology layers, and specifically mentions time division multiplexing (TDM), wavelength and spatial (fibre) switching.

The GMPLS framework works as follows:

- A routing protocol distributes topology knowledge across the network.
- This information is stored in Traffic Engineering Databases (TED).
- Users and applications can send path setup requests to the network.
- A path computation element (PCE) in the network takes responsibility for the computation of the actual network connection [s27].
- The signalling of the decided path to the network elements that do the actual provisioning is done with a signalling protocol.

While this framework does not enforce a specific routing or signalling protocol, the de-facto standards are Traffic Engineering (TE) extensions to Open Shortest Path First (OSPF) and Resource Reservation Protocol (RSVP), OSPF-TE and RSVP-TE respectively [s23, s18]. The path computation elements (PCE) framework in GMPLS also does not enforce a specific path finding algorithm [s27].

GMPLS describes the network topology using OSPF-TE Link State Announcements (LSAs). The connections through this network are built using Label Switched Paths (LSPs), which reside on a single layer. All hops of an LSP can be described using a Record Route Object (RRO) in RSVP-TE.

Since GMPLS uses OSPF-TE, the full topology information is distributed across all nodes. Without modifications this does not scale for larger networks, let alone multi-domain scenarios. RFC 4655 says (section 4.9.1) "PCE is not considered to be a solution that is applicable to the entire Internet. That is, the applicability of PCE is limited to a set of domains with known relationships." [s27, s28, s30].

GMPLS deploys three mechanisms to confine scalability within reasonable limits:

Choice of labels during signalling: Instead of distributing information about available labels in the routing protocol, a list of k-shortest paths is chosen without this information. During the signalling phase, the ReSerVation

Protocol (RSVP) announces the available *labels*, such as wavelengths or VLAN tags, to its neighbours. The given paths are then tried in order until a path is found where an unused label is available.

Loose hops: The path computation can be done per-layer. Instead of providing a list of *strict hops*, it is possible to use *loose hops* in the Explicit Route Object (ERO), and only specify hops on the same layer, and let another path computational element (PCE) deal with the hops on another layer.

Abstraction of domains: The size of the graph can be reduces by abstracting the topology information per domain or per OSPF area. For example, the DRAGON project [p24] contains a Network Aware Resource Broker (NARB) that abstracts the topology in either a set of edge nodes (nodes connected to a neighbouring domain) or even one node per domain [t3].

Unlike OSPF, IS-IS, RIP en BGP, the network model of GMPLS is not based on a graph but mostly on ideas in the ITU-T, such as G.805 and G.800 [s42, s41].

3.3 Path Finding in Multi-Layer Networks

The Internet and the telephony network as examined in the previous section are essential single layer networks with very few incompatibilities (the distinction between IPv4 and IPv6 is a notable exception). GMPLS deals with complexity by assuming that incompatibilities are rare: it only applies to single or few domains, with one or only few different technologies, and it uses k-shortest path heuristics to deal with potential incompatible labels.

The use of multi-layer networks leads to additional technology constraints. Technology constraints are potential incompatibilities such as incompatible packet size leading to packet loss, a laser transmitting light at a wavelength undetectable by a receiver, or two devices supporting a different *adaptation*, the encapsulation of data of one layer into another layer.

These technology constraints are important for two reasons. First of all, we argued in section 2.3 that technology constraints will continue to exist. Ideally, transport protocols will evolve to a single de-facto standard, eliminating incompatibilities. However, as technologies evolve over time, incompatibilities will continue to exist. This surfaces in multi-domain networks, as engineers in different domains choose different technology standards. For example, one network may provide dynamic switching capabilities using MPLS, while another network may provide dynamic switching capabilities using Ethernet PBB-TE, and a third network using DWDM lightpaths. So as long as technologies evolve, multi-domain networks are also multi-layer networks.

Secondly, technical incompatibilities for multi-layer networks can exist between two geographically separated domains. They are not limited to domains directly connected with each other, and thus can not be solved locally, but needs to be announced across multiple domains. We will see an example of such a geographically separated incompatibility in the example later in this section.

At the beginning of this chapter, the question was raised if it is possible to use the same path finding algorithms in multi-layer transport networks as those in use for the Internet and telephony networks. The Internet and telephony networks rely on graph theory for modelling and algorithms that find a shortest path through a graph in polynomial time.

We have not yet given an exact problem statement for path finding in multi-layer networks, as currently no all-encompassing model for multi-layer networks exists (chapter 4 will define such a model). For now, all we know is if graph theory and shortest-path algorithms in graphs are to be sufficient to be used in multi-layer networks, they must deal with dynamic capabilities as described in section 2.3.3 and technology constraints such as incompatible packet size, lasers with different wavelength, etc. We will now claim that graph theory and algorithms in graphs are not sufficient to deal with this.

First of all, **graphs can not be used for path finding in multi-layer networks**. To be precise, we claim that graphs can not be used for path finding in multi-layer networks, if it is required that (1) the outcome of the path finding algorithm is sufficient to reconstruct the original path in the network; (2) the graph can be created from the actual network in polynomial time; and (3) a path finding algorithm as discussed in section 3.1 can be used.

A network is not a graph. A graph is merely a representation of a network, and it is possible to map the same network onto multiple graphs. This distinction is causes the three conditions in the above claim.

Secondly, **it is not possible to use existing path finding algorithms as used on the Internet and telephony network for multi-layer networks**. To be precise, we claim that **link-constrained algorithms are not sufficient for path finding in multi-layer networks, if links are 1:1 mapped to edges**, but that **path-constrained algorithms *are* sufficient for path finding in multi-layer networks**.

The above claims are proven in the following subsections.

The root of both claims is that path finding in single layer networks is a problem in complexity class P, that is, a decision problem that is verifiable in polynomial time by a non-deterministic Turing machine. Algorithms like breadth first search and Dijkstra are non-deterministic Turing machines, and not only verify but find the solution in polynomial time (their running time is $\mathcal{O}(n)$ and $\mathcal{O}(n \times log(n))$ respectively, with n the number of vertices in the graph). On the other hand, path finding in multi-layer networks is a *NP-hard* problem, that is, a decision problem whose time-complexity is at least as large of that of *NP-complete* problems. In a joint paper, Kuipers recently proved that the 3SAT problem can be mapped onto the multi-layer path finding problem [a12]. Since 3SAT is a known *NP-complete* problem, this proves that that the multi-layer path finding problem is *NP-Hard*. Thus, **path finding in a single layer network belong to a different complexity class (P) than path finding in a multi-layer network (NP-hard)**. The exact complexity class to which this problem belong is yet unknown, and will depend on the exact formulation of the multi-layer path finding problem. The problem to find a path in a multi-layer network shorter than a given length is easy to verify, making that problem *NP-complete*. The problem to find the shortest path in a multi-layer network may be harder to verify. That problem has either *NP-complete*, *EXPTIME* or *NEXPTIME* complexity.

3.3.1 Practical Example

We first turn to our original question, is it possible to use the same path finding algorithms in multi-layer transport networks as those in use for the Internet and telephony networks? We claim that this is not true. path finding algorithms in use for the Internet and telephony networks are based on graphs, as described in section 3.1. Each edge in these graphs represents a single links in the network. The essence of path finding algorithms in graphs, is that a shortest path can not contain cycles in the graph. If the existing algorithms can be used in a multi-layer network, this means that no shortest path exists which the same link twice.

We will now give a counter-example where the shortest path does traverse the same link twice.

Figure 3.2 introduces an example network, which is used throughout the remainder of this thesis. Each circle in the picture represents an operational network domain. The domains are interconnected by links: the edges in the figure. Each domain is a participant in the Global Lambda Integrated Facility (GLIF). University of Amsterdam and Université du Quebec are universities, CAnet is a National Research and Education Network (NREN), and StarLight, MAN LAN and NetherLight are optical exchanges. All these domains collaborate to provide researchers with circuit switched connections, the *lightpaths*.

While this example is based on a historic scenario, the topology is modified

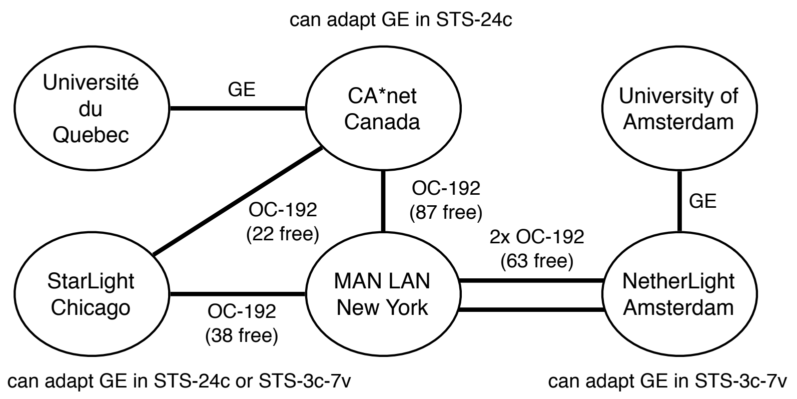


Figure 3.2: *Example of a multi-layer and multi-domain network.*

to emphasise our point, and the mentioned incompatibilities are not present at these specific domains in real-life.

The network in the example is not only a multi-domain network, but also a multi-layer network. The connection between the Université du Quebec and CAnet, as well as the connection between the Universiteit van Amsterdam and NetherLight is a Gigabit/second Ethernet (GE) connection.

While Ethernet is a common technology in domains and between universities and their access network, the international connections are currently mostly based on SONET and SDH. These technologies provide the monitoring capabilities that Ethernet lacks, but which are required for such long distances. The Ethernet packets are transported over the SONET/SDH network. This encapsulation of data of one layer (Ethernet) in another layer (SONET) is called *adaptation*. The extraction of the data is called *de-adaptation*.

The links in SONET and SDH are optical carriers (OC), and the OC-192 variant is divided in 192 timeslots, referred to as STS channels. The total capacity of an OC-192 connection is 9.8 Gigabit/second, and each timeslot is 51 Megabit/second. 51 Megabit/second is less than the 1 Gigabit/second for Gigabit Ethernet, so each Gigabit Ethernet connection needs to be packed in multiple STS channels.

There are at least five different standards to encapsulate Gigabit Ethernet in STS channels[1]. CAnet supports a Cisco-developed variant that can embed

[1]GE over STS3c-7v/VC4-7v using GFP-F [s45]; GE over STS-24c using LEX [s9]; GE over STS-24c using Cisco HDLC [s57]; GE over STS-24c using PPP/BCP [s19]; GE over STS-24c using Ethernet in HDLC framing over PPP over SONET [s8, s13]; and GE over

Gigabit Ethernet in 24 concatenated STS channels (an STS-24c) [s57]. NetherLight supports a ITU-developed variant called Generic Framing Procedure (GFP) [s45] which embeds Gigabit Ethernet in 7 VC-4 containers, each in 3 concatenated STS channels: 21 STS channels in total (an STS-3c-7v). StarLight supports both methods to adapt Ethernet in STS channels.

In this example, an application wants to have a Gigabit/second Ethernet (GE) connection between the Université du Quebec in Montreal (Canada) and the University of Amsterdam (the Netherlands). This can be achieved by creating a switched circuit through a set of the interconnected networks. Not all 192 STS channels in this example are available. The numbers next to the links show how many channels are free.

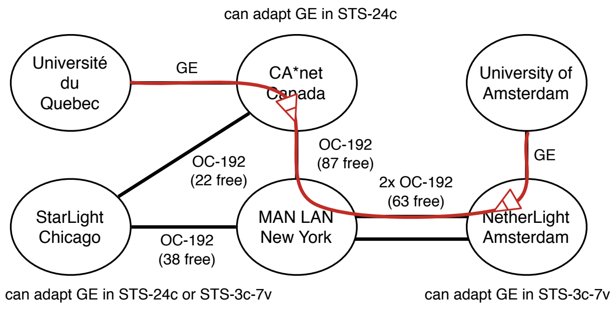


Figure 3.3: *An invalid network connection: the adaptation of gigabit Ethernet (GE) in 24 STS channels (STS-24c) is incompatible with the adaptation of gigabit Ethernet in 21 STS channels (STS-3c-7v).*

If we would treat figure 3.2 as a graph, the shortest path from the Université du Quebec to the University of Amsterdam would traverse CAnet, MAN LAN and NetherLight respectively, as shown in figure 3.3. However, in practice this would be a non-functioning network connection since the adaptation performed at CAnet, which adapts the GE in 24 STS channels, is incompatible with the adaptation of GE in 21 STS channels, as performed in NetherLight.

In this example network, StarLight is able to convert gigabit Ethernet (GE) in 24 STS channels (STS-24c) to gigabit Ethernet in 21 STS channels (STS-3c-7v). Nevertheless, the network connection from Université du Quebec via CAnet, StarLight, MAN LAN, and NetherLight to the University of Amsterdam, as shown in 3.4, is also non-functioning: there are only 22 STS channels

STS-24c using PPP [s13]. This ignores another four variants that use STS3c-8v/VC4-8c instead of STS-24c using VCAT [s47]

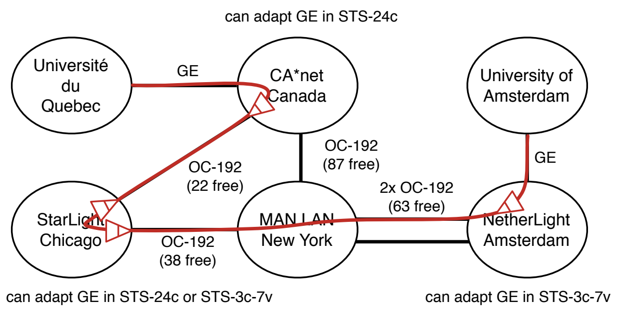


Figure 3.4: *An invalid network connection: there are only 22 STS channels available between CAnet and StarLight, while 24 are required.*

available between CAnet and StarLight, while 24 are required.

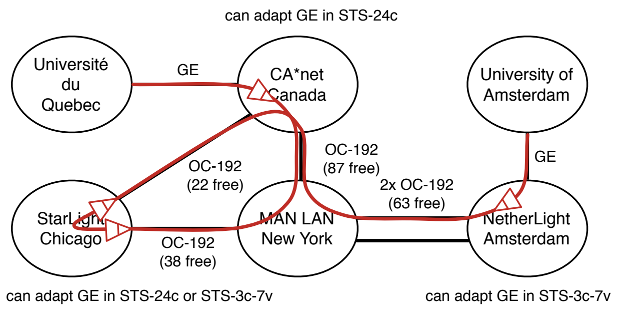


Figure 3.5: *The shortest valid network connection from Université du Quebec to University of Amsterdam through the example network of figure 3.2.*

In fact, the shortest functional path between the Université du Quebec and the University of Amsterdam is shown in figure 3.5, and traverses the link between CAnet and MAN LAN twice. Once as Gigabit Ethernet in 24 STS channels, and the second time as Gigabit Ethernet in 21 STS channels.

If we define a ‘loop’ in a network connection as a network connection that traverses the same physical link twice, then we can say that the shortest network connection between the Université du Quebec and the University of Amsterdam has a loop.

So we now have a multi-layer network where the shortest path does traverse the same link twice. This proves our claim that **the algorithms used in the**

Internet and telephony network can not be used for path finding in multi-layer transport networks, if links in the network are 1:1 mapped to edges in the graph.

3.3.2 Path-Constrained Problem

We earlier mentioned that path finding in a single layer network is a problem in complexity class *P*, while path finding in a multi layer network is a *NP-hard* problem.

Fundamentally, path finding in multi-layer networks contains *conditional* constraints based on the chosen path. In the given example, it is only possible to extract Ethernet data from the SONET layer if the Ethernet data was earlier embedded in the SONET layer *with the same adaptation function.* A corresponding problem is travel problem with multiple transport vehicles where you can always depart by bus or train from Amsterdam central station, but only by bike if you somehow first managed to get your bike at the station. This means that path finding in multi-layer networks requires a *path-constrained* algorithm. The algorithms in use for single layer networks such as the Internet and telephony network are *link-constrained* algorithms. This is the fundamental reason why path finding algorithms as used in the Internet or telephony network can not be used for multi-layer path finding.

Our earlier claim that *the algorithms used in the Internet and telephony network can not be used for path finding in multi-layer transport networks, if links in the network are 1:1 mapped to edges in the graph*, can be rewritten as **link-constrained algorithms are not sufficient for path finding in multi-layer networks, if links are 1:1 mapped to edges**.

Similarly, the third condition in the claim that *graphs can not be used for path finding in multi-layer networks*, that *a path finding algorithm as discussed in section 3.1 can be used*, can now also be rewritten by the condition that *a link-constrained path finding algorithm is used.*

The full claim is now: **graphs can not be used for path finding in multi-layer networks, if it is required that (1) the outcome of the path finding algorithm is sufficient to reconstruct the original path in the network; (2) the graph can be created from the actual network in polynomial time; and (3) a link-constrained path finding algorithm is used**.

We can proof this claim by contradiction. Lets assume that graphs could be used for path finding in multi-layer networks, while all conditions hold. By condition (3), path finding in the graph would be solvable (and thus also verifiable) in polynomial time, as link-constrained path finding algorithms (without further conditions) have P complexity. By condition (1), this would give a solution for the original problem, and by condition (2) the solution for path finding in multi-layer networks would be solvable in polynomial time as well. However, Kuipers has shown that the multi-layer problem is NP-hard, so this would lead to a contradiction. So we must conclude that our claim is true.

Our example network has two conditional constraints. First, a de-adaptation may only be used if the corresponding adaptation occurred earlier in the path. Secondly, a link may only be (re)used if the capacity is sufficient.

The incompatibility caused by the two adaptation functions occurred between StarLight and NetherLight. These domains are not direct neighbours. This implies that **multi-layer incompatibilities can not be resolved locally**. Information about these incompatibilities needs to be distributed across domains.

Multi-layer path finding algorithm must not only have information of the layers and adaptations of its direct neighbours, but also of the layers and adaptations of domains further down the path. Another way to look at this is that a path finding algorithm must not only take the *topological neighbours* into account, thus the neighbours at the physical layer, but also the *technological neighbours*: the neighbours on higher layers (for instance StarLight and NetherLight as seen in figure 3.3).

Individual domains deal with multi-layer complexity by choosing to switch at a single technology layer only, but since different domains choose different layers, the complexity remains in the system as a whole.

We have proven the that link-constrained algorithms are not sufficient for path finding in multi-layer networks, if links are 1:1 mapped to edges, and now postulate the theorem that **path-constrained algorithms *are* sufficient for path finding in multi-layer networks**. So far we only made this theorem plausible by the describing of the problem and have proven it for one example only. In order to prove it for networks, we must show that a path constrained algorithm can be applied to all network examples, using any technology. We will construct such a path-constrained path finding algorithm that can be used for all technologies in the remainder chapters, in particular in chapter 7.

3.3.3 Graphs

The counter example in section 3.3.1 contains a shortest path where a certain link is used twice. The proof relied on the fact that we mapped each link in the network to a single edge in the graph. Using this condition, we could map loops in a network to cycles in a graph.

This mapping between a network and a graph is important: A graph is merely a representation of a network, and it is possible to map the same network onto multiple graphs.

Earlier, we proved the following claim: **graphs can not be used for path finding in multi-layer networks, if it is required that (1) the outcome of the path finding algorithm is sufficient to reconstruct the original path in the network; (2) the graph can be created from the actual network in polynomial time; and (3) a link-constrained path finding algorithm is used**.

All three conditions in this claim are essential. If any of these conditions is left out, it is possible to use graphs for path finding in multi-layer networks. We will now prove this for the first two conditions, and get back to the third condition in the next subsections.

The first condition, *the outcome of the path finding algorithm is sufficient to reconstruct the original path in the network*, is required to only allow useful mappings from networks to graphs.

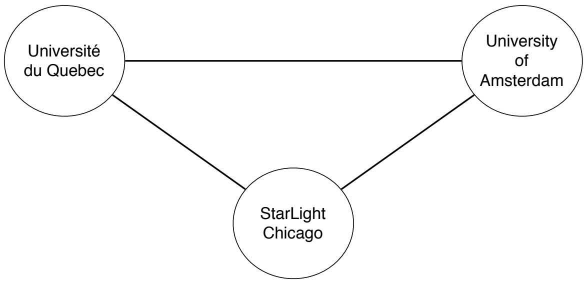


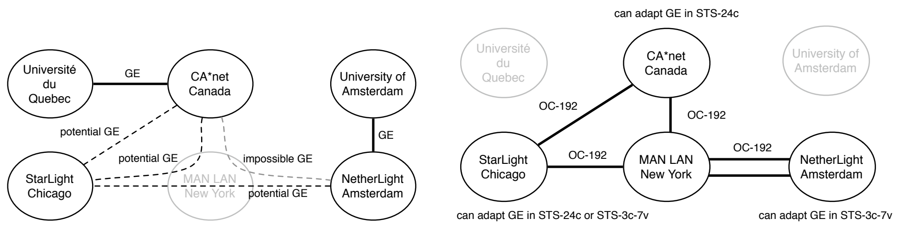
Figure 3.6: *Graph where each edge represents the shortest path in the network of figure 3.2.*

Figure 3.6 shows a mapping from network to graph where this condition is not met. In this figure, all Ethernet domains are mapped onto a vertex, and the shortest path between each Ethernet domain is mapped onto an edge. In this graph each edge represents a series of links in the network. Path finding in this graph can be done with conventional algorithms, and is trivial: For example the single edge between Quebec and Amsterdam represents is the shortest path between these two locations and represent the full network connection described in figure 3.5. Of course, this graph is not very useful: the graph is so condensed that it is not possible to determine the actual path in the network from a path in the graph.

The second condition, *the graph can be created from the actual network in polynomial time* is also required. It is possible to create a graph that retains all information about the network, but only requires a conventional path finding algorithm. Given a source device, construct a tree graph, starting with a vertex representing this device. Extend the tree by an edge for each link connected to the device. Each new vertex represents not only the connected device, but also the state of the network so far. Disallow vertices that yield conflicting states (for example because the link is used twice, while not enough capacity is left in the network). Recursively construct the branches of the tree until the destination is reached in each branch. Path finding in this graph is trivial, but it would take an exponential time to construct this graph. This graph does not so much represent the network as well as the result of a path finding algorithm, and we want to disallow it for that reason.

3.3.4 Multi-Layer Representations

A common representation of a network is to map nodes or domains to vertices and links to edges. If we would do this with the network of figure 3.2, we get a graph with 6 vertices and 7 edges. Such a graph lacks essential information about adaptation in the network and can no be used for multi-layer path finding.



(a) *Ethernet layer* **(b)** *SONET layer*

Figure 3.7: *Two layers of the multi-layer network of figure 3.2. This still does not visualise the adaptations between the two layers.*

Figure 3.7 describe the example multi-layer network as multiple graphs: one graph per layer. Edges in such a graph would represent channels rather than physical links. Observe that the topology of the network is different at each layer, and such a multi-graph description makes this explicit. This is also not complete because it does not explicitly describe the adaptation functions, and thus also not the compatibilities and incompatibilities caused by adaptation functions.

It is possible to abstract the possible and impossible connections on the SONET layer in figure 3.7b and place these as edges in the Ethernet layer in subfigure 3.7a. Besides that the generation of this graph is non-trivial (all possible links at a lower layer need to be determined, before the higher layer graph can be created), it also needs information about possible links that can not be created together if the run over the same limited capacity link.

The recurring problem is that information about the relation between different network layers is lost in the mapping from network to graph. The fundamental limitation of graphs is that graph theory only provides two basic building blocks, edges and vertices, while multi-layer computer networks have at least three building blocks: links, devices and adaptations, and perhaps four if interfaces are counted as well.

With only two building blocks, vertices and edges, the following choices must be considered when mapping multi-layer networks onto a graph:

- A vertex in a graph may either represent a device, a physical interface, a logical interface, an adaptation stack at an interface, or even a link (in bipartite graphs).

- An edge in a graph may either represent a link, a channel in a link (for instance wavelength 1310 nm in a fibre), or an adaptation function.

Given this dissimilarity between multi-layer networks and graphs, we ask ourselves the following question:

> **Are there other network models, beside graphs, that can describe multi-layer networks? Can such a model be technology-independent?**

Either a new model needs to be chosen, or the graph model needs to be extended to contain all information, and the path finding algorithm needs to be adjusted to take this additional information into account.

It is possible to map both links and adaptations to edges. Figure 3.8 gives such a representation for our example network. In order to describe adaptations, the network elements that provide an adaptation must be described as multiple logical vertices: at least one for each layer, with an edge in between them. Multi-layer network description must distinguish between physical interfaces and logical interfaces, and between physical links and logical channels in these links. This leads us to the following claim: **multi-layer networks**

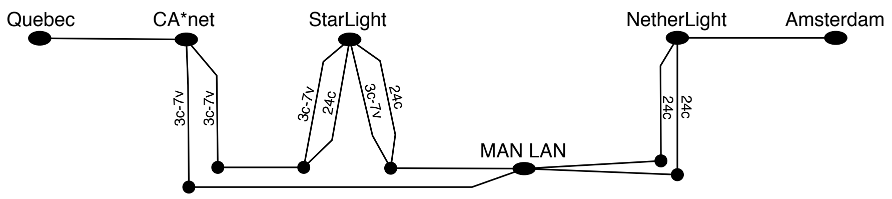


Figure 3.8: *The network of figure 3.2, modelled as graph with edge properties.*

can only be mapped to a graph if devices are mapped to multiple vertices, or if information about the adaptation is lost.

This graph in figure 3.8 shows that the third condition in the claim that *the graph can be created from the actual network in polynomial time* is also required. This graph does provide all information, if all edges and vertices are labelled (which is necessary to distinguish between adaptation and de-adaptation and incompatible adaptation functions). However, we still need a path-constrained algorithm to find the shortest path in that graph.

Links and adaptations have different properties, even though they are both mapped onto edges in figure 3.8. It is non-trivial to represent multiplexing adaptation functions (adaptations with multiple channels over a single link). In the next few chapters we will attempt to overcome these shortcomings and find a more suitable approach to model multi-layer networks.

3.4 Path Finding in Transport Networks

So far, we have looked at the technology constraints caused by the multi-layer nature of transport networks. We will continue to do so in the next chapter, but for the moment, we take a step back and look at the other properties of transport networks.

We claim that path finding in multi-layer transport networks is different from path finding in the regular Internet or the telephony network, because of the circuit-switched nature of transport networks.

Transport networks consist of scarce resources, which can be reserved for existing services. This means that links may not be assumed to be available at all time. As a consequence, the pair of end-nodes is not enough to compute a path, but the state of the network must be taken into account for each path setup request.

In fact, at least four different types of information may be required to find a valid path:

Topology information – the interconnections between devices and between domains;

Technology information – the potential technical incompatibilities;

Scheduling information – the simultaneous availability of required resources;

Policy information – the user authorisation level on the available resources.

Path Finding Software

Whilst it is possible to distribute all this information in a routing protocol, that would give a routing table that scales with the product of the number of each constraint type (topology, technology, scheduling and policy).

3.5 Multi-Stage Path Finding

Because transport networks as described in this thesis are still relatively new, it is yet unknown which constraints are important and which can be ignored.

In case all constraints are relevant, a possible routing table would not scale. Since the different constraints are orthogonal to each other, the size would scale with the product of each number of constraints, $\mathcal{O}(\prod_i |C_i|)$ with $|C_i|$ the number of different values for constraint C_i. Thus the number of destinations, the number of incompatibilities, the number of potential time slots and the number of per-user policies. Clearly, if different constraints are relevant and orthogonal to each other, using a routing table will not scale.

If not all path request can be pre-computed in a routing table, then for each path set up request a new path needs to be calculated at the time of the request. It can not be done by distributing all routing information beforehand.

One of the questions to ask is if it is possible to create a scalable multi-domain path finding algorithm without relying on a routing table (path vector algorithms) or on knowledge about each domain (link state algorithms which require full topology knowledge)? If so, does this alternative approach have advantages over current path vector or link state algorithms?

Torab *et al.* did show that other algorithms than shortest path first (SPF) are possible [p40]. In particular, they distinguished between (1) no collaboration, (2) collaboration but no cooperation, (3) model-based cooperation and (4) ad-hoc cooperation.

no collaboration: only intradomain calculation. Each domain independently determines the egress (exit) port, without consulting the domain where it leads to in this decision. No path information exchange between PCE in each domain.

collaboration but no cooperation: receive time frame or event based topology and resource availability information. No per-request information exchange.

model-based cooperation: limited information (”model”) of dynamic info of few neighbouring domains, and additionally availability information exchange.

ad-hoc cooperation: information exchange as soon as a request comes in. No a-priori information.

Independent from Torab *et al.*, we turned to a very basic algorithm for our path finding problem: a broadcast algorithm. The basic idea is that domains simply forward a path setup request by checking if the request can be accommodated, and for each possible egress interface, forwarding the path request along with the possible path so far to the next domain. While we anticipate that this is not a very efficient algorithm, it is a first approach and it may be possible that all sorts of constraints will limit the flooding of such an algorithm.

The broadcast algorithm would effectively be a *breadth first search algorithm* that checks all possible paths in parallel. Alternatively, it is possible to intelligently try one egress domain at a time, making it a *depth first search algorithm*, with some sort of back-track algorithm in case of dead ends.

Our ideal approach is very similar to the model-based co-operation, and was partially inspired by a talk by Lehman [t12].

In our view, domains should push ‘static’ and ‘non-sensitive’ information to their neighbours (e.g. “this is our topology”), and they should provide a service to either get more information (e.g. “there is a link here, what wavelengths are still available?”) or provide a (web)service where to ask a specific configuration question (e.g. “can I use wavelength 1552 nm for this link for the next hour, for a user in my virtual organisation?”)

In particular, domains should announce (push) the following information to their neighbours, for each type of information (topology, technology, scheduling and policy information):

- either the information itself; or
- an access method to retrieve this information; or

- an access method to check for usage availability.

Our vision is that a path computation element acts as a broker and uses these information services from different domains to stitch a working end-to-end path together.

3.6 Conclusion

Multi-layer networks give rise to technology incompatibilities. Individual domains deal with multi-layer incompatibilities by choosing to switch at a single technology layer only, but since different domains choose different layers, the complexity remains in the system as a whole.

We have shown that algorithm as used in the Internet or telephony networks can not be used for multi-layer path finding. To be precise, we have proven that **Link-constrained algorithms are not sufficient for path finding in multi-layer networks, if links are 1:1 mapped to edges**. We have shown this using a counter example.

In addition, we have shown that graphs can not be used for path finding in multi-layer networks, if it is required that (1) the outcome of the path finding algorithm is sufficient to reconstruct the original path in the network; (2) the graph can be created from the actual network in polynomial time; and (3) a link-constrained path finding algorithm is used.

The standard mapping of one link to one edge leads to loss of information in the graph, and other mappings from a network to a graph have similar problems. The bottom line is that path finding in a single layer network and path finding in multi-layer network belong to different complexity categories.

The rest of this thesis is devoted to prove that **Path-constrained algorithms *are* sufficient for path finding in multi-layer networks**. In order to prove this claim, we present a path-constrained algorithm in the next chapters, and show that it can find path in multi-layer networks for all technologies known to us.

Chapter 4

Multi-Layer Network Model

This chapter is based on *A Multi-Layer Network Model Based on ITU-T G.805* by F. Dijkstra, H.M. Andree, K. Koymans, J.J. van der Ham, P. Grosso, and C.T.A.M. de Laat [a10].

4.1 Introduction

In this chapter we examine abstract representations of multi-layer networks. Network models can help users and applications to understand the complexity of multi-layer networks. In particular models can support path finding, scheduling, fault isolation, and visualisation applications.

Path finding in multi-layer networks requires knowledge about the relation between different network layers (the adaptation).

The most common representation of a network is a graph. As we saw in the previous chapter, a simple graph with devices represented as nodes and links as edges is not able to describe multi-layer networks, so we turn to more elaborate models.

For our network model, we set two goals: first, such a model should be able to explicitly describe multi-layer networks, including the relation between the layers, and second, it should be able to find valid potential connections through a given network. For example, it should be possible to distinguish between the invalid paths and the valid paths in the example networks presented in the previous chapter.

Since we did not find suitable models to describe multi-layer computer networks, we developed a new model, which is technology independent, but layer aware. This network model is based on functional elements as defined in ITU-T Recommendation G.805 [s42], combined with the label concept in GMPLS [s20] and the addition of capability information. Furthermore, we show that it is possible to use a simple algebra to verify the validity of an end-to-end network connection, traversing multiple layers.

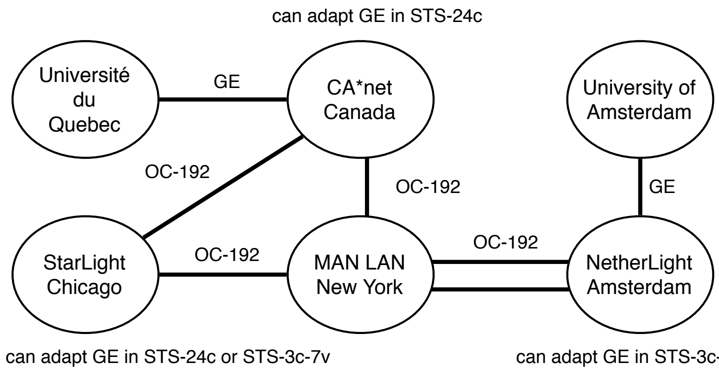


Figure 4.1: *Example of a multi-layer and multi-domain network.*

We will explain the model using the example network in figure 4.1. This is the same example as we have seen in the previous chapter, but we will ignore the availability constraints for now.

Each circle in the picture represents an operational domain. The domains are interconnected by links: the edges in the figure.

The organisation of this chapter is as follows. Section 4.2 starts with the related work. In section 4.3 we will introduce ITU-T Recommendation G.805. The network model is introduced in section 4.4, along with a simple algebra to verify validity of network connections through the network. Section 4.5 demonstrates the usability of the network model by an example network and section 4.6 describes a few possible extensions of this model. Finally, we present related work and conclusions in section 4.7.

4.2 Related work

The great advantage of technology-independent network description is that a path finding algorithm only needs to know about the generic concepts such as 'layer' and 'adaptation', but not about the specific technologies. It does not need to be tuned or adjusted as new network technologies come along.

Since new technologies and thus incompatibilities will come along as time progresses (see our claim in section 2.3.1), our interest lays in technology independent, multi-layer network description. Table 4.1 shows some of the related work.

	Technology Specific	**Technology Independent**
Single Layer	*most network models*	Graphs
Multi Layer	GMPLS model, CIM, network simulators	ITU-T G.805, G.800

Table 4.1: *Categorization of related work. Single layer technology specific models are not listed, since they are of no interest to us.*

ITU-T Recommendations G.805 and G.800 and graph theory are all technology independent: they can be applied to any technology.

G.805 is the first standard to clearly define the terms *adaptation* and *termination* to describe the relation between different layers. G.805 is based on transport networks and can only be applied to circuit-switched data networks. ITU-T G.809 [s43] extends these definitions to include packet switched networks.

The few models that take multiple layers into account are often geared towards very specific cases (for instance simulation of a few specific layers, like in network simulators).

As early as 1995, Laarhuis developed a model where the network was divided in three layers [p21]. The physical media layer containing all network components and fibres, the optical layer consisting of wavelength channels, and the electrical layer which uses the virtual topology of the optical layer to obtain connections. Like us, he based his work on ITU-T G.805 functional elements.

G.800 [s41] is a recent extension of ITU-T G.805 that adds the concepts of domains, symbols and labels. Both G.805 and G.800 provide functional elements to describe the state of a network. Neither provides a description how the state can be changed, and by whom. For path-finding, the information how to change a network is just as crucial as knowing the current state. The capability of a network describes how the state can be changes, while the policy defines who may change the state.

Whereas Graphs, and the functional models in G.805 and G.800 are technology independent, there are many more models that are specific for a limited set of technologies. We describe two of these models that have generated a considerable momentum at the moment of this writing, the model in Generalized Multi-Protocol Label Switching (GMPLS) and the Common Information Model (CIM).

4.2.1 Generalized Multi-Protocol Label Switching

Generalized Multi-Protocol Label Switching (GMPLS) is a set of protocols for routing and signalling in circuit switched networks [s20, s22, s18]. Its path finding properties were already discussed in section 3.2.4, and this section concentrates on its network model only.

GMPLS does not specify a formal network model, but merely specifies parameters for different technologies, as required by a network control plane.

GMPLS can describe the layers and switching capability of devices at a layer. However, it currently only has a limited concept of adaptation, by using a Generalized Protocol IDentifier (G-PID) to specify the payload of the channels. But this information is only used during the signalling phase, when the path is already established. In agreement with our findings, it was independently determined that *the advertisement of the internal adaptation capability of hybrid nodes* is required in the routing protocol [s32]. A proposal for these routing extensions is in draft as of this writing [s31, s33].

4.2.2 Common Information Model

The Common Information Model (CIM) [s2] is a schema defined by the Distributed Management Task Force (DMTF). CIM is an object oriented schema, which can describe hardware elements in high detail. It can describe networks and has a collection of schemes to describe a configuration of IP, BGP, OSPF, Ethernet (including VLAN), NAT, pipes and filters.

This makes CIM a very useful tool for describing a (network) configuration in detail. In particular, it makes a great database for access networks, especially if tools like SNMP can automatically generate the data.

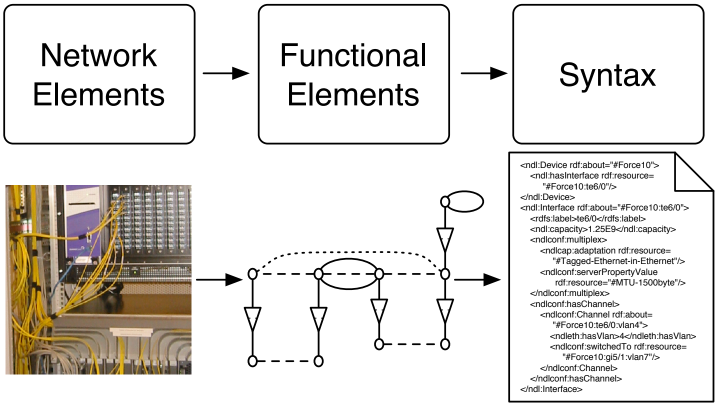
CIM is less suitable for core networks since it can not describe DWDM or TDM networks. CIM is a technology specific model, which makes it less suitable for our purpose: a technology independent model.

4.3 ITU-T G.805 Concepts

This section serves as a short explanation to ITU-T G.805 terminology [s42]. A few concepts are simplified for readability. For example, we do not distinguish between a link, and the transport entity across a link. The section on functional elements and connection partitioning are our own additions.

4.3.1 Functional Elements

The process of creating an abstract description of a network involves two steps, as shown in figure 4.2.



```
<ndl:Device rdf:about="#Force10">
  <ndl:hasInterface rdf:resource=
     "#Force10:te6/0"/>
</ndl:Device>
<ndl:Interface rdf:about="#Force10:te6/0">
  <rdfs:label>te6/0</rdfs:label>
  <ndl:capacity>1.25E9</ndl:capacity>
  <ndlconf:multiplex>
    <ndlcap:adaptation rdf:resource=
       "#Tagged-Ethernet-in-Ethernet"/>
    <ndlconf:serverPropertyValue
       rdf:resource="#MTU-1500byte"/>
  </ndlconf:multiplex>
  <ndlconf:hasChannel>
    <ndlconf:Channel rdf:about=
       "#Force10:te6/0:vlan4">
      <ndleth:hasVlan>4</ndleth:hasVlan>
      <ndlconf:switchedTo rdf:resource=
         "#Force10:gi5/1:vlan7"/>
    </ndlconf:Channel>
  </ndlconf:hasChannel>
</ndl:Interface>
```

Figure 4.2: *The two steps required to create an abstract description of a network.*

The first step consists of the creation of an abstract representation of the physical network elements. Individual components in the abstract representation are so-called functional elements, like *device* and *interface*. The mapping of network elements to functional elements is called information modelling [s17]. A second step is the mapping of the functional elements to a certain syntax. In this chapter, we only examine the first step, the modelling.

The distinction between model and syntax is important for interoperability. If two control planes use a different syntax, but the same model it is straightforward to translate between the to syntaxes. It is hard, and sometimes impossible without making assumptions, to translate between two different models.

ITU-T G.805 provides a set of generic functional elements, without actually specifying this mapping between network elements and functional elements. Neither does it provide a syntax to describe the functional elements.

The functional elements defined by ITU-T G.805 allow one to describe a circuit switched network connection through multiple layers, or a network at a single layer. In addition, ITU-T G.809 [s43] allows the same for packet switched network connections. However, in our view, G.809 tries to map the terminology of circuit switched networks to packet switched networks, ignoring important characteristics of packet switched networks like packet sizes or buffer sizes. We mostly ignore packet switched networks in this chapter.

4.3.2 Connection Point and Layer

ITU-T G.805 defines a *connection point* as a source and sink for data transport. A good way to think about it is as a hop or (virtual) interface on a network connection. One physical interface can consist of multiple logical interfaces. For instance one for each distinguishable data flow.

A *layer* is defined as the set of all possible connection points of the same type. Two connection points are of the same type if a data-transport function can be created between them. So each connection point resides at one specific layer.

4.3.3 Connections

Informally, a *connection point* can be thought of as a vertex in a graph, and a *link connection* as an edge in the graph. A *tandem connection* is a series of contiguous link connections (a *path* in graph theory), and a *network connection* is a tandem connection between two connection points where the connection is terminated for that layer: an end-to-end connection on a certain layer.

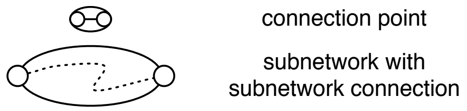
More formally G.805 defines a link connection as “a *transport entity* that transfers information between parts across a link”. A network connection is defined as “a series of contiguous link connections and/or subnetwork connections between *termination connection points*”.

The termination connection point in the previous definition means that the network connection is an end-to-end connection on that layer.

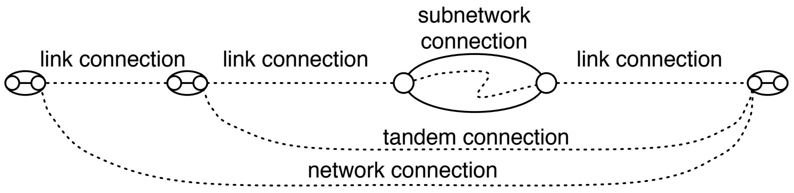
Subnetworks can represent parts of the network *at a single layer*. In general, subnetworks may be partitioned into smaller subnetworks interconnected by link connections. The minimal subnetwork in G.805 is called a *matrix*. A connection through a subnetwork is called a *subnetwork connection*. An indivisible subnetwork connection is called a *matrix connection*. As subnetworks and matrices are defined at a single layer, subnetwork connections and matrix connections can only be described at a single layer as well.

We will later use subnetworks to represent network devices.

The diagrammatic conventions are depicted in figures 4.3a and 4.3b.



(a) *connection point and subnetwork connection*



(b) *link connection, tandem connection and network connection*

Figure 4.3: *Graphical representations of functional elements.*

4.3.4 Adaptation and Termination

If we want to send data belonging to layer X over a different layer Y, the data needs to be transformed. This transformation is defined by an *adaptation function.*

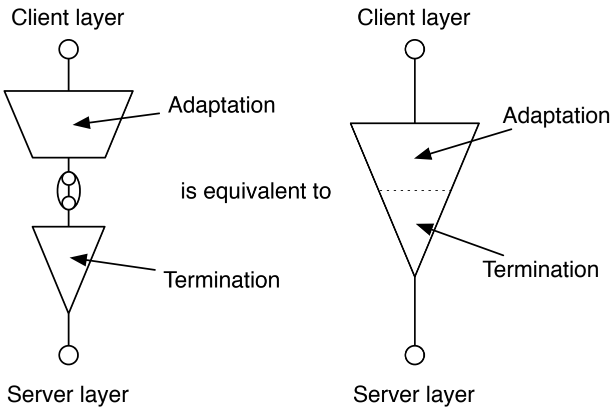


Figure 4.4: *Graphical representation of the adaptation and termination functions. In this chapter we will use the simplified representation shown at the right.*

ITU-T G.805 defines the *adaptation function* and the *termination function.*

The adaptation function defines how data belonging to a client layer (the 'higher' layer) network is embedded into data of a server layer (the 'lower' layer) network. The termination function adds monitoring information to the server layer network connection, taking care of a reliable data transmission.

The trail termination function is defined by G.805 as *a transport processing function that consists of the trail termination source where monitoring information is added and the trail termination sink which removes the monitoring information*. So in short, a termination function just adds *monitoring information*. For example, a termination function adds a checksum field to each data packet.

A graphical representation of these functional elements is depicted in figure 4.4. The adaptation function is visualised by the upper part of the triangle and the termination function is visualised by the lower part of the triangle.

Adaptation and termination have analogies in the real world. For example, we want to send five pairs of wooden shoes from Amsterdam to Quebec. Rather than sending them as-is, we wrap them in a box for shipping. This is the adaptation. However, a box can only contain 3 pairs of shoes, so we use two boxes and mark them as 'box 1/2' and 'box 2/2'. This allows the recipient to verify that the shipment arrived complete and unmodified. This is the termination.

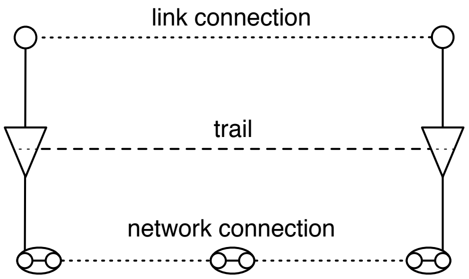


Figure 4.5: *A network connections on the server layer with two adaptations functions yields a link connection on the client layer.*

Figure 4.5 shows how adaptations can be used to build network connections on a higher layer, the client layer, using network connections on an underlying layer, the server layer. According to the G.805 definition, a link connection "represents a pair of adaptation functions and a trail in the server layer network", where the trail is a termination network connection. So if there is a network connection on a (lower) server layer network, and both ends have the same termination and adaptation functions, then there is a link connection at the client layer above.

The embedding of data of one layer into another is a recursive process. For example, the OSI model [s53] defines 7 network layers where the data of each layer is embedded in the layer directly underneath. We call such a sequence of adaptation in adaptation an *adaptation stack*.

4.3.5 Multiplexing

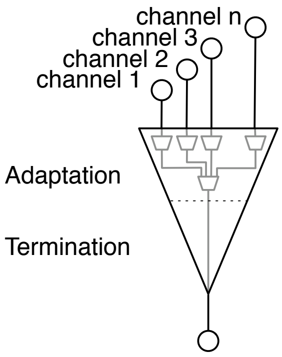
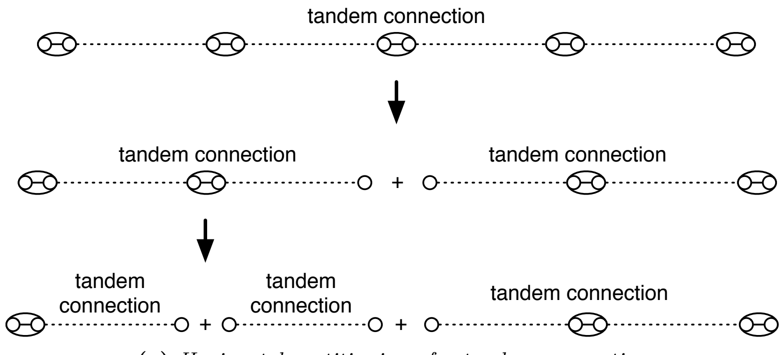


Figure 4.6: *Implementing multiple connections over a network link is equivalent with multiplexing at the adaptation function.*

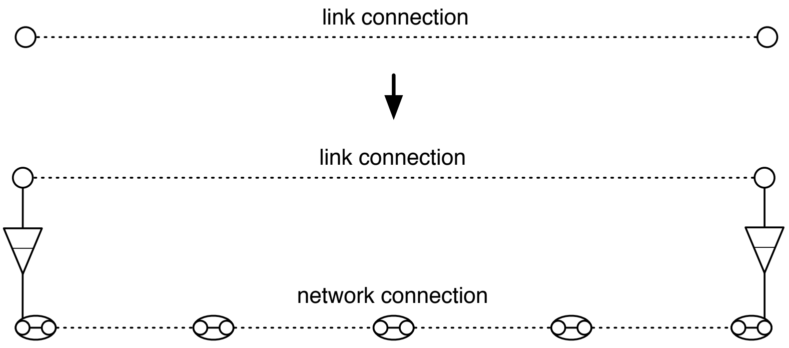
Channeling, implementing multiple connections over a network link, is equivalent to multiplexing at the adaptation function in G.805. As figure 4.6 shows, the adaptation function may consist of specific processes for each channel at the client layer and one common process that converts these adapted client layer channels to the server layer. Each logical channel interface is represented as a connection point on the client layer, while there is only one (termination) connection point at the server layer.

4.3.6 Connection Partitioning

Figure 4.7 repeats the two different ways how a connection can be partitioned. A tandem connection can be split in multiple tandem connections, up to the smallest unit, a link connection. This is a partitioning on a single layer, and we refer to it as horizontal partitioning. Horizontal partitioning is ambiguous. For example, the connection $A-B-C-D$ can either be split in $A-B-C$



(a) *Horizontal partitioning of a tandem connection*



(b) *Vertical partitioning of a link connection*

Figure 4.7: *Two partitionings of a connection in smaller parts.*

and $C - D$ or in $A - B$ and $B - C - D$. The chosen partitioning depends on organisational reasons like the boundary of the operational domains.

The other partitioning is of a link connection on a client layer into a network connection on another layer. This is partitioning between different layers, and we refer to it as vertical partitioning. It is determined by the actual technology and therefore unambiguous, not driven by organisational decisions.

4.4 Network Model

The ITU-T G.805 recommendation can be used for describing connections in multi-layer networks. The model we present here is based on the ideas in G.805, and the label concept in GMPLS. In addition, we model the capability, thus how the state of a network can be changed.

4.4.1 Mapping to Functional Elements

Table 4.2 shows an overview of our mapping from real-life network elements (for instance links, and devices with interfaces) to G.805 functional elements. We model the switching core of a network device as a subnetwork. A network device contains interfaces, which are modelled as multiple connection points (one or more for each layer) and optional adaptation capabilities. Finally, we map links between interfaces to link connections in G.805.

Network Element	**Functional Elements**
Domain	Subnetwork(s)
Device	Matrix (Subnetwork)
Interface	Connection point(s) and adaptation function(s)
Link	Link connection

Table 4.2: *Mapping of network elements to G.805 functional elements*

An interface is modelled as multiple connection points, one for each channel on each layer. For example, an OC-192 interface in a SONET device is modelled as 194 connection points: one connection point representing the interface at the fibre layer, one connection point representing the wavelength, and 192 connection points representing the 192 available STS channels.

The switching capability of a device is modelled as a switch matrix on a specific layer. For example, an SDH device which is capable of switching data with the granularity of STS channels has a switch matrix at the STS layer, while an SDH device which is capable of switching data with the granularity of virtual tributaries groups (VTG) has a switch matrix at the VTG layer.

Domains are treated as 'virtual' devices, and modelled as subnetworks, just like devices are. A difference is that physical devices in general can only switch on one granularity, represented by a subnetwork at a specific layer, while a domain may switch at different granularities, represented by multiple subnetworks.

Physical links are modelled as link connections on one of the physical layers. So a fibre is modelled as a link connection at the fibre layer and an unshielded twisted pair (UTP) cable is modelled as a link connection at the UTP layer.

An adaptation function defines the relation between the connection points that represent the different layers of an interface.

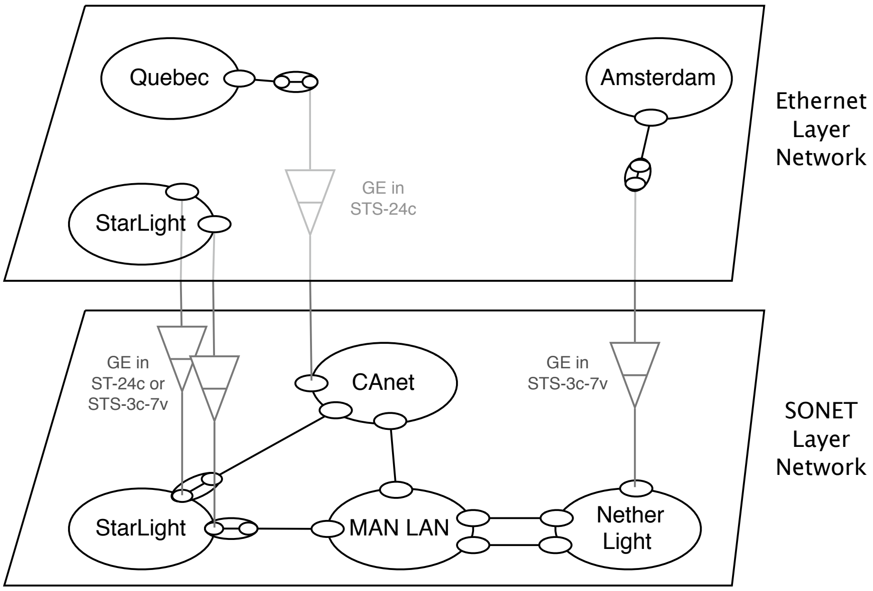


Figure 4.8: *The network of figure 4.1, modelled as functional elements.*

Figure 4.8 shows an example network description using functional elements. The network is a slightly simplified version[1] of the network described in figure 4.1. Unlike figure 3.7 in the previous chapter, we explicitly modelled the adaptation functions. The two layers are separated vertically, while the different domains are separated horizontally. For example CAnet is represented by one subnetwork, five connection points and one adaptation function: the device is represented as a subnetwork, each SONET interface as one connection point and the Ethernet interface as two connection points (one on the Ethernet layer, one on the SONET layer), with an adaptation function in between.

Since StarLight can both switch at the Ethernet layer as well as the SONET layer, it is represented as two subnetworks: one at the Ethernet layer, one at the SONET layer. In this drawing, each interface only has one adaptation function (either STS-3c-7v or STS-24c), while in practice it may be possible

[1]For simplicity, the Ethernet-in-STS-channels adaptation is modelled as a one-to-one relation, instead of the actual one-to-many relation.

to dynamically switch between these two adaptation functions at the same interface. It is possible to model this as two adaptation functions with a multi-point connection point (MPCP) to dynamically switch between them. These kind of choices needs to be made in order to turn the information model of this chapter into a data model. We will come back to these decisions in the next chapter.

4.4.2 Notation

We define the function that combines the adaptation of data flow T from client layer to data flow U at the server layer, and the termination of the data flow U as

$$A : T^n \to U^m$$

with n and m equal to 1 for regular adaptation functions, $n > 1$ for multiplexing adaptation functions, and $m > 1$ for inverse multiplexing adaptation functions. For simplicity, we will simply write $A : T \to U$, and refer to both the data as well as the layers as T and U.

Except for section 4.4.6, we will simply refer to the combined adaptation and termination function as the adaptation. This implies that A is noncommutative.

Given an adaptation function $A : T \to U$, then by definition a de-adaptation function[2] $D : U \to T$ exists such that $D \circ A = id : T \to T$.

$$\forall(A : T \to U)\ \exists(D : U \to T) : D \circ A = id : T \to T \tag{4.1}$$

Two adaptation functions A_1 and A_2 are considered a pair if $A_2^{-1} \circ A_1 = id$. Typically, because $A_1 = A_2$

We will denote the adaptation performed between connection points $cp1t$ at the client layer T and $cp1u$ at the server layer U as $A_{cp1u}^{cp1t} : T \to U$. The corresponding de-adaptation function will be named $D_{cp1t}^{cp1u} : U \to T$, or equivalently, $(A_{cp1u}^{cp1t})^{-1} : U \to T$.

Unless noted otherwise, a function A will refer to an adaptation function, and a function D to a de-adaptation function.

Figure 4.9 shows an example of a description of a network connection between two computers. As we can see, both interfaces are modelled (as connection points) on all applicable layers. For instance for interface $if1$, as $cp1f$ at the fibre layer, $cp1e$ on the Ethernet layer and $cp1i$ at the IP layer.

[2] In mathematical terms D is a retraction or a split epimorphism.

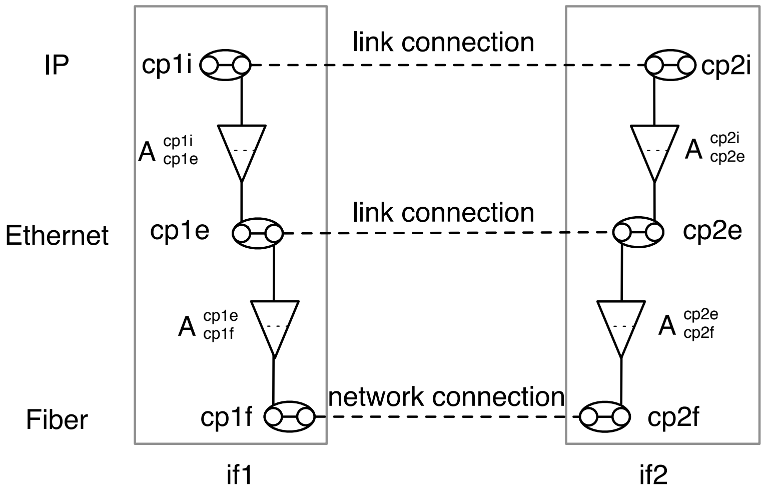


Figure 4.9: *Example of a multi-layer network connection. Interfaces $if1$ and $if2$ are modelled as connection points at all three layers. The relation between the connection points is defined by the adaptation and termination functions.*

4.4.3 Channel Labels

In 4.4.1 we wrote that each channel is represented as a connection point. So an OC-192 interface has 192 STS connection points, a tagged Ethernet interface has 4096 VLAN connection points and an ATM VPI can contain 65536 VCI channels.

Seemingly, this does not scale very well. However, that would be a misunderstanding, since it is often not needed to describe all individual connection points in a syntax. Only the channels that are configured or actively in use need to be described in detail. The other channels can simply be described as a set or range of available channels. This is an important distinction between the model and the syntax describing a model: **a model can be verbose, while the syntax is compact**.

The use of channels requires an addition to our model. Consider the adaptations pair $A_{cp1u}^{cp1t_1;cp1t_2;cp1t_3;\ldots;cp1t_n} : T^n \to U$ and $A_{cp2u}^{cp2t_1;cp2t_2;cp2t_3;\ldots;cp2t_n} : T^n \to U$ in figure 4.10. This is an example of a multiplexing adaptation function with client layer connection points $cpit_1; cpit_2; cpit_3; \ldots; cpit_n$ with associated notation.

From the logic of section 4.3.4 it follows that since there is a network con-

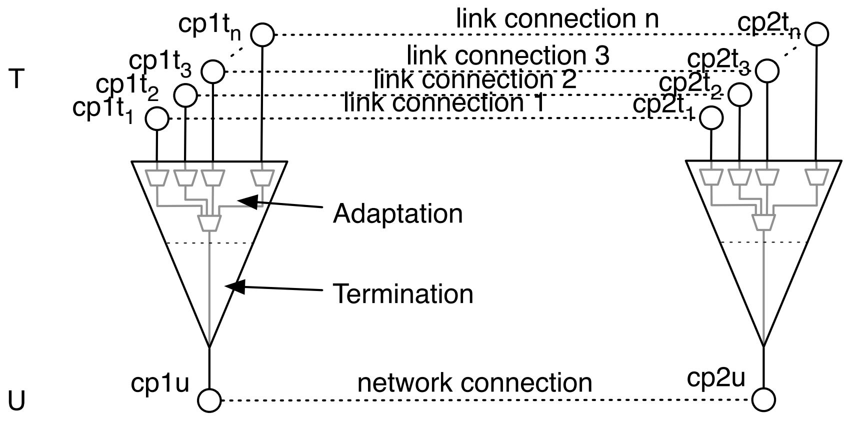


Figure 4.10: *Channels correspond with multiple link connections at the client layer over one link connection at the server layer.*

nection on layer U and the two adaptations are equal, there is a link connection on layer T. However, it is not obvious between which pair of connection points there is a link connection. Without further specification, it could for example be between *cp1t₁* and *cp2t₃*. As a remedy, we introduce the concept of *labels*, inspired by GMPLS [s20].

Each connection point has two associated labels for each link connection connected to it: the *ingress label* and *egress label.* These labels uniquely identify the channel of an adaptation. Examples of labels would be STS timeslots, IEEE 802.1Q (VLAN) tags or wavelengths.

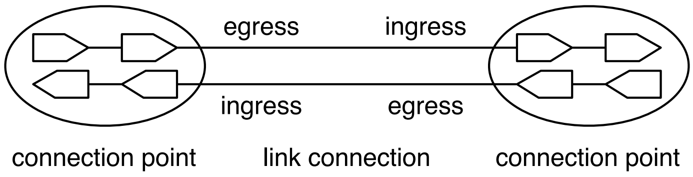


Figure 4.11: *The ingress and egress part of an connection point with respect to a link connection.*

Figure 4.11 shows two connection points and a link connection. For labels, we distinguished between the two uni-directional link connections that constitute a bi-directional link connection.

For uni-directional connections a link connection from *cp1* to *cp2* can only exist if the egress label of connection point *cp1* is equal to the ingress label of connection point *cp2*. For a bi-directional link connection, we also require that the egress label of connection point *cp2* is equal to the ingress label of connection point *cp1*.

For bi-directional circuit switched connections, the ingress and egress label are typically the same, and we simply talk about the label of a connection point, meaning both the ingress and egress label.

We define the functions:

- $Lb_{out}(cp)$ to be the egress label of connection point *cp*.
- $Lb_{in}(cp)$ to be the ingress label of connection point *cp*.

If the egress and ingress labels are equal, as for bi-directional circuit switched network connections, we can define the equality:

$$Lb(cp) := Lb_{out}(cp) = Lb_{in}(cp) \tag{4.2}$$

If an interface does not have a particular label, we consider it to have an “empty” label, ϵ. For all other purposes, we consider ϵ just to be a regular label.

4.4.4 Capability Model

The functional elements of G.805 and G.800 only allow a description of the state of a network at a certain point in time. The capability describes how the state of a network can be changed.

In our model, there are only two parameters that can change:

- the labels of a connection point and
- the subnetwork connections within a subnetwork.

A label can only be changed to a predefined value. Each connection point has an egress and an ingress labelset, $Ls_{out}(cp)$ and $Ls_{in}(cp)$ respectively. The label of a connection point *cp* can be changed to any value $Lb(cp) \in Ls(cp)$.

A subnetwork is a set of connection points. A subnetwork connection exists between two connection points in this set. Rather than allowing subnetwork connections between all connection points in a subnetwork, we place a restriction on it. Subnetworks can have either or both of the *switching* and *swapping* capability. The switching capability refers to a switch matrix that can forward data as long as the label is the same. For example a WDM device without wavelength conversion, or an Ethernet switch that can not convert between VLAN tags. The swapping capability refers to a switch matrix that can forward data while changing the label. For example a WDM device with wavelength conversion capabilities or an SDH device with virtual concatenation (VCAT) [s47] and Link Capacity Adjustment Schema (LCAS) [s46] support that can forward data from one timeslot to another timeslot without constraints.

4.4.5 Validation of Network Connections

In this section, we introduce a mathematical concept to check the validity of a network connection. We use a recursive definition to verify that a network connection is *valid*.

Given connection points *cp1* and *cp2*, we will define the following binary relations:

- $L(\mathit{cp1}, \mathit{cp2}) \iff$ a directional Link from *cp1* to *cp2* exists;
- $\mathit{SNC}(\mathit{cp1}, \mathit{cp2}) \iff$ a directional Subnetwork Connection from *cp1* to *cp2* exists;
- $\mathit{LC}(\mathit{cp1}, \mathit{cp2}) \iff$ a directional Link Connection from *cp1* to *cp2* exists;
- $\mathit{TC}(\mathit{cp1}, \mathit{cp2}) \iff$ a directional Tandem Connection from *cp1* to *cp2* exists.

We postulate a network $N = (\mathit{CP}, L, \mathit{SN}, A)$ as a set of connection points CP, physical links L, subnetworks SN, and adaptations A. The network configuration $C = (\mathit{LB}, \mathit{SC})$ is a set of labels LB, and subnetwork connections SC. Given these basic premises N and C, we deduce the link connections and tandem connections: the valid connections through the network.

G.805 defines a tandem connection as a transport entity formed by a series of contiguous link connections and/or subnetwork connections. We define a tandem connection recursively to be either a link connection, a subnetwork connection or a tandem connection followed by another tandem connection.

A link connection is defined either to be a link or a combination of an adaptation source, a terminated tandem connection at the server layer, and an adaptation sink.

Mathematically the definitions of tandem connection and link connection can be written as:

$$TC(cp1, cp2) = \begin{cases} LC(cp1, cp2) \lor & \text{(4.3a)} \\ SNC(cp1, cp2) \lor & \text{(4.3b)} \\ \exists cp3 : TC(cp1, cp3) \land TC(cp3, cp2) & \text{(4.3c)} \end{cases}$$

and

$$LC(cp1, cp2) = \begin{cases} L(cp1, cp2) \lor & \text{(4.4a)} \\ \exists cp3, cp4, T, U, A^{cp1}_{cp3}, D^{cp4}_{cp2} : & \\ \quad TC(cp3, cp4) \land & \\ \quad A^{cp1}_{cp3} : T \to U \land & \\ \quad D^{cp4}_{cp2} : U \to T \land & \text{(4.4b)} \\ \quad D^{cp4}_{cp2} \circ A^{cp1}_{cp3} = Id : T \to T \land & \\ \quad Lb_{out}(cp1) = Lb_{in}(cp2) \land & \\ \quad Lb_{in}(cp1) = Lb_{out}(cp2) & \end{cases}$$

For simplicity, we will now restrict ourselves to bidirectional connections. Thus:

$$L(cp1, cp2) \to L(cp2, cp1) \tag{4.5a}$$
$$LC(cp1, cp2) \to LC(cp2, cp1) \tag{4.5b}$$
$$TC(cp1, cp2) \to TC(cp2, cp1) \tag{4.5c}$$
$$SNC(cp1, cp2) \to SNC(cp2, cp1) \tag{4.5d}$$

and even:

$$(A^{cp1t}_{cp1u} : T \to U) \to (D^{cp1u}_{cp1t} : U \to T) \tag{4.6}$$

with

$$D^{cp1u}_{cp1t} \circ A^{cp1t}_{cp1u} = Id : T \to T \tag{4.7}$$

These definitions can easily be transformed to those for uni-directional connections, or explicitly allowing multiplexing and inverse multiplexing adaptation functions.

These recursive definitions, in particular the one for link connections, need a short explanation. We will refer to figure 4.12 to illustrate the concepts. This figure shows a network N_{ex} with links, five link connections, nine tandem connections and one subnetwork connection in total.

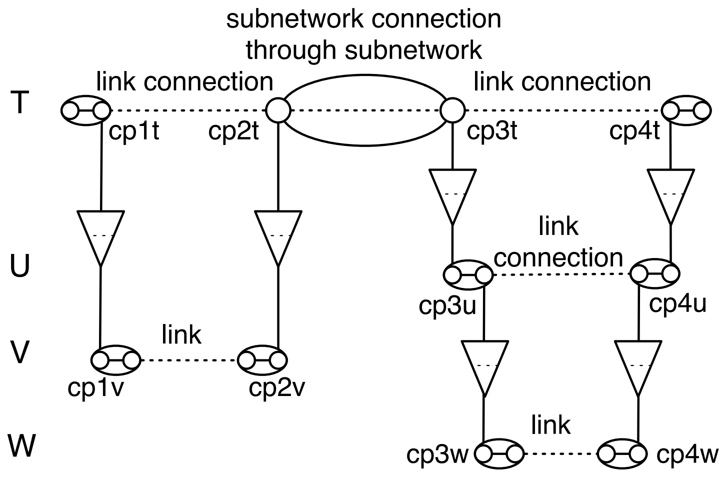


Figure 4.12: *example of a valid network connection. A valid tandem connection consisting of two link connections and a matrix connection.*

Formally, we postulate the network $N_{ex} = (CP_{ex}, L_{ex}, SN_{ex}, A_{ex})$ as the sets:
$CP_{ex} = \{cp1t, cp2t, cp3t, cp4t, cp1v, cp2v, cp3u, cp4u, cp3w, cp4w\}$,
$L_{ex} = \{L(cp1v, cp2v), L(cp3w, cp4w)\}$,
$SN_{ex} = \{\{cp2t, cp3t\}\}$, and
$A_{ex} = \{A^{cp1t}_{cp1v}, A^{cp2t}_{cp2v}, A^{cp3t}_{cp3u}, A^{cp4t}_{cp4u}, A^{cp3u}_{cp3w}, A^{cp4u}_{cp4w}\}$.

In this network, $A^{cp1t}_{cp1v} = A^{cp2t}_{cp2v}$, $A^{cp3t}_{cp3u} = A^{cp4t}_{cp4u}$, and $A^{cp3u}_{cp3w} = A^{cp4u}_{cp4w}$.

The configuration $C_{ex} = (LB_{ex}, SC_{ex})$ of network N_{ex} are the sets of labels and subnetwork connections: $LB_{ex} = \{\forall_{cp \in CP_{ex}} : Lb(cp)$ with $Lb(cp) = \epsilon$ (no explicit labels defined)$\}$, and
$SC_{ex} = \{SNC(cp2t, cp3t)\}$.

The most simple link connection is simply a link. So $L(cp1v, cp2v)$ implies $LC(cp1v, cp2v)$ and $L(cp3w, cp4w)$ implies $LC(cp3w, cp4w)$. By definition of a tandem connection, a link connection is also a tandem connection, so $LC(cp1v, cp2v)$ and $LC(cp3w, cp4w)$ imply $TC(cp1v, cp2v)$ and $TC(cp3w, cp4w)$ respectively.

We just saw that $TC(cp1v, cp2v)$ holds. Furthermore, $A^{cp1t}_{cp1v} = A^{cp2t}_{cp2v}$, thus:

$$D^{cp2v}_{cp2t} \circ A^{cp1t}_{cp1v} = Id : T \to T \tag{4.8}$$

with $D^{cp2v}_{cp2t} = (A^{cp2t}_{cp2v})^{-1}$. Therefore, from equation 4.4 we must conclude that

$LC(cp1t, cp2t)$. In G.805 terminology, the *adaptation source* $cp1t$ and the *adaptation sink* $cp1v$ are *paired*.

Similarly, $LC(cp3u, cp4u)$, and therefore $TC(cp3u, cp4u)$ hold because $TC(cp3w, cp4w)$ and $D(cp4w, cp4u) \circ A(cp3u, cp3w) = Id : U \rightarrow U$, and $LC(cp3t, cp4t)$ holds because $TC(cp3u, cp4u)$ and $D(cp4u, cp4t) \circ A(cp3t, cp3u) = Id : T \rightarrow T$.

Furthermore, $LC(cp1t, cp2t)$, $SNC(cp2t, cp3t)$, and $LC(cp3t, cp4t)$ respectively imply $TC(cp1t, cp2t)$, $TC(cp2t, cp3t)$, and $TC(cp3t, cp4t)$. Two consecutive tandem connections are also a tandem connection, so from this follows that $TC(cp1t, cp3t)$ and $TC(cp2t, cp4t)$. Finally, $TC(cp1t, cp4t)$ holds because $LC(cp1t, cp2t)$ and $TC(cp2t, cp4t)$.

4.4.6 Well Typed Adaptations

So far, we combined the adaptation and termination function.

We did so to make our definition of $LC(cp1, cp2)$ in equation 4.4 compatible with the definition of link connection in G.805, where a link connection "represents a pair of adaptation functions and a trail in the server layer network." Since a *trail* is a terminated network connection in G.805, the adaptation and termination functions are always combined.

For validation, in the definition of link connections we required that the server layer network connection was terminated. In this section we will loosen this restriction. We call a link connection that is formed by a combination of an adaptation source, a server layer tandem connection, and an adaptation sink *well-typed*, even if the server layer network connection is not terminated as required for *validity*.

Refer to figure 4.13 for a *well-typed*, but invalid link connection between $cp1t$ and $cp2t$. An example of such an invalid link connection could be if A_{cp1v}^{cp1t} adds a header to a packet, and A_{cp1w}^{cp1v} adds a tail to the result. Then, D_{cp2u}^{cp2w} first removes the header and finally D_{cp2t}^{cp2u} removes the tail. While the result is the very same packet, the intermediate result for adaptation and de-adaptation was different: a packet with header (layer V) during adaptation and a packet with tail (layer U) during de-adaptation. Since $cp1v$ and $cp2u$ are on different layers, no termination is possible at A_{cp1v}^{cp1t} and D_{cp2t}^{cp2u}.

Loosening the restriction that each adaptation function is followed by a termination function has consequences for a possible definition of atomic or combined adaptation functions. We will not pursue this idea further, but assume that each adaptation function is followed by a termination function.

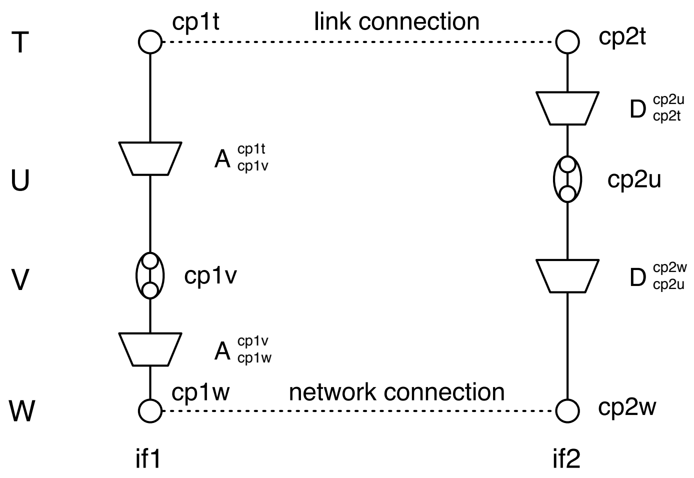


Figure 4.13: *Example of a well typed, but invalid connection. U and V are different layers.*

4.5 Validation

In the introduction, we sketched an example network which had some restrictions in the validity of connections through the network. We will now show how the model in section 4.4 can be used to make this explicit.

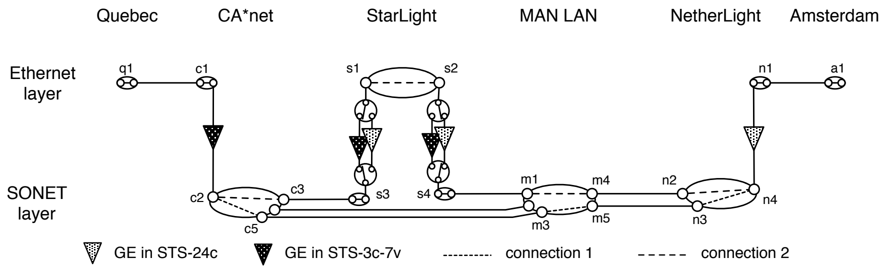


Figure 4.14: *A network representation of the network of figure 4.1, using functional elements. Dark-gray adaptation functions represent adaptation of Gigabit/second Ethernet (GE) over STS-24c, while light-gray adaptation functions represent GE over STS-3c-7v. StarLight is capable of either adaptation function.*

Figure 4.14 gives a representation the network N_g of figure 4.1 as functional elements, using the mapping of table 4.2.

This network is identified by $N_g = N_g(CP_g, E_g, SN_g, A_g))$, with:
$CP_g = \{q1, c1, c2, c3, c4, c5, s1, s2, s3, s4, m1, m2, m3, m4, m5, n1, n2, n3, n4, a1\}$,
$E_g = \{L(q1, c1), L(c3, s3), L(c4, m2), L(c5, m3), L(s4, m1), L(m4, n2), L(m5, n3), L(n1, a1)\}$,
$SN_g = \{\{c2, c3, c5\}.\{s1, s2\}, \{m1, m2, m3, m4, m5\}, \{n2, n3, n4\}\}$, and
$A_g = A^{c1}_{c2}, A^{s1}_{s3}, A^{s2}_{s4}, A^{n1}_{n4}\}$ where $A^{c1}_{c2} = STS24c$ and $A^{n1}_{n4} = STS3c7v$.

The shortest path (traversing fewest link connections) between connection point $q1$ at the Université du Quebec and connection point $a1$ at the University of Amsterdam traverses StarLight, MAN LAN and NetherLight. This would result in connection 1 in the figure 4.14. Formally, connection 1 is dataflow through the network elements $[L(q1, c1), A^{c1}_{c2}, SNC(c2, c5), L(c5, m3), SNC(m3, m5), L(m5, n3), SNC(n3, n4), D^{n4}_{n1}, L(n1, a1)]$ and is identified by the subset $C1 = \{SNC(c2, c5), SNC(m3, m5), SNC(n3, n4)\}$ of the network configuration.

Given the above sets, we can use the definitions of section 4.4.5 to derive the valid link connections and tandem connections in this network.

$$SNC(c2, c5) \xrightarrow{equation\ 4.3b} TC(c2, c5) \tag{4.9}$$

$$L(c5, m3) \xrightarrow{equation\ 4.4a} LC(c5, m3) \xrightarrow{equation\ 4.3a} TC(c5, m3) \tag{4.10}$$

$$SNC(m3, m5) \xrightarrow{equation\ 4.3b} TC(m3, m5) \tag{4.11}$$

$$L(m5, n3) \xrightarrow{equation\ 4.4a} LC(m5, n3) \xrightarrow{equation\ 4.3a} TC(m5, n3) \tag{4.12}$$

$$SNC(n3, n4) \xrightarrow{equation\ 4.3b} TC(n3, n4) \tag{4.13}$$

$$TC(c2, c5) \wedge TC(c5, m3) \wedge TC(m3, m5) \wedge TC(m5, n3) \wedge TC(n3, n4) \xrightarrow{equation\ 4.3c\ recursive} TC(c2, n4) \tag{4.14}$$

However, from A^{c1}_{c2}, $TC(c2, n4)$, D^{n4}_{n1} does *not* follow $LC(c1, n1)$ since $D^{n4}_{n1} \circ A^{c1}_{c2} = STS3c7v^{-1} \circ STS24c \neq Id : Ethernet \rightarrow Ethernet$. Therefore, connection 1 does not lead to a valid tandem connection from Quebec to Amsterdam, given these links and subnetwork connections:

$$N, C1 \nvdash TC(q1, a1) \tag{4.15}$$

StarLight is capable of supporting either adaptation function. This is modelled in figure 4.14 using two multi-point connection points (MPCP). A^{s1}_{s3} is either equal to $STS24c$, or to $STS3c7v$.

Let's now consider connection 2, identified by the subset $C2 = \{SNC(c2, c3)$, $SNC(s1, s2)$, $SNC(m1, m4)$, $SNC(n2, n4)\}$ of the network configuration, $A^{s1}_{s3} = STS24c$ and $A^{s2}_{s4} = STS3c7v$

$$SNC(c2, c3) \xrightarrow{equation\ 4.3b} TC(c2, c3) \tag{4.16}$$
$$L(c3, s3) \xrightarrow{equation\ 4.4a} LC(c3, s3) \xrightarrow{equation\ 4.3a} TC(c3, s3) \tag{4.17}$$

$$TC(c2, c3) \wedge TC(c3, s3) \xrightarrow{equation\ 4.3c} TC(c2, s3) \tag{4.18}$$

$$A^{s1}_{s3} = STS24c \xrightarrow{equation\ 4.6} D^{s3}_{s1} = STS24c^{-1} \tag{4.19}$$

$$TC(c2, s3) \wedge \tag{4.20}$$
$$D^{s3}_{s1} \circ A^{c1}_{c2} = STS24c^{-1} \circ STS24c = Id : \text{Ethernet} \to \text{Ethernet} \wedge \tag{4.21}$$
$$Lb_{out}(c1) = \epsilon = Lb_{in}(s3) \wedge \tag{4.22}$$
$$Lb_{in}(c1) = \epsilon = Lb_{out}(s3) \tag{4.23}$$
$$\xrightarrow{equation\ 4.4b} TC(c1, s1) \tag{4.24}$$

Similarly for the connection between $s2$ and $n1$:

$$L(s4, m1) \xrightarrow{equation\ 4.4a} LC(s4, m1) \xrightarrow{equation\ 4.3a} TC(s4, m1) \tag{4.25}$$
$$SNC(m1, m4) \xrightarrow{equation\ 4.3b} TC(m1, m4) \tag{4.26}$$
$$L(m4, n2) \xrightarrow{equation\ 4.4a} LC(m4, n2) \xrightarrow{equation\ 4.3a} TC(m4, n2) \tag{4.27}$$
$$SNC(n2, n4) \xrightarrow{equation\ 4.3b} TC(n2, n4) \tag{4.28}$$

$$TC(s4, m1) \wedge TC(m1, m4) \wedge TC(m4, n2) \wedge TC(n2, n4) \xrightarrow{equation\ 4.3c} TC(s4, n4) \tag{4.29}$$

$$A^{n1}_{n4} = STS3c7v \xrightarrow{equation\ 4.6} D^{n4}_{n1} = STS3c7v^{-1} \tag{4.30}$$

$$TC(s4, n4) \wedge \tag{4.31}$$

$$D^{n4}_{n1} \circ A^{s2}_{s4} = STS3c7v^{-1} \circ STS3c7v = Id : \text{Ethernet} \rightarrow \text{Ethernet} \wedge \tag{4.32}$$

$$Lb_{out}(s2) = \epsilon = Lb_{in}(n1) \wedge \tag{4.33}$$

$$Lb_{in}(s2) = \epsilon = Lb_{out}(n1) \tag{4.34}$$

$$\xrightarrow{equation\ 4.4b} LC(s2, n1) \tag{4.35}$$

We can now combine the derived truth statements $LC(c1, s1)$ and $LC(s2, n1)$ with other statements in the network description N_g and it's configuration $C2$:

$$L(q1, c1) \xrightarrow{equation\ 4.4a} LC(q1, c1) \xrightarrow{equation\ 4.3a} TC(q1, c1) \tag{4.36}$$

$$LC(c1, s1) \xrightarrow{equation\ 4.3a} TC(c1, s1) \tag{4.37}$$

$$SNC(s1, s2) \xrightarrow{equation\ 4.3b} TC(s1, s2) \tag{4.38}$$

$$LC(s2, n1) \xrightarrow{equation\ 4.3a} TC(s2, n1) \tag{4.39}$$

$$L(n1, a1) \xrightarrow{equation\ 4.4a} LC(n1, a1) \xrightarrow{equation\ 4.3a} TC(n1, a1) \tag{4.40}$$

$$\tag{4.41}$$

$$TC(q1, c1) \wedge TC(c1, s1) \wedge TC(s1, s2) \wedge TC(s2, n1) \wedge TC(n1, a1)$$
$$\xrightarrow{equation\ 4.3c} TC(q1, a1) \tag{4.42}$$

Thus $TC(q1, a1)$ is true. This proves that there is now a valid tandem connection from $q1$ at the Université du Quebec to $a1$ at the University of Amsterdam:

$$N, C2 \vdash TC(q1, a1) \tag{4.43}$$

4.6 Extensions of the Model

This section highlights a few of the possible extensions to our current model.

One of our goals is to describe actual networks in a technology-independent way. In order to implement some of the extensions mentioned in this section, it is likely that some of the (mathematical) simplicity of the current model will be lost while gaining a model able to describe additional technologies, or additional logic used in some technologies only. Care should be taken to retain the basic logic.

4.6.1 Layer Properties

One motivation to describe networks is to make incompatibilities between interfaces specific. We did so for incompatible adaptations (for instance Ethernet over STS-24c or over STS-3c-7v), and for incompatible labels (for instance a wavelength with label '1310 nm' or a wavelength with label '850 nm'.)

This does not cover all possible incompatibilities. For example, a network connection may not be possible due to a difference in the allowed packet size (for instance Ethernet packets with an MTU of 1500 bytes or 9000 bytes, or anything in between).

It is technically possible to model this as a few thousand different adaptation functions, but this is not efficient. An alternative is to define the layer properties, and extend the model by defining when two values cause an incompatibility. This later approach is used by the stitching framework in GÉANT2 [t9]. This stitching framework defines a "method of logic" for each property (same, different, comparable, overlapping, open, different, min, or sum).

4.6.2 Inverse Multiplexing

Both G.805 as well as our model support inverse multiplexing: the adaptation of one data stream in multiple channels. Ethernet in STS channels, as described in examples in this chapter, is an instance of inverse multiplexing. The model as presented in this chapter is limited to a single underlying network connection. For inverse multiplexing, $cp3$, $cp4$ in equation 4.4 needs to be changed to $cp3_1, \ldots, cp3_n$, $cp4_1, \ldots, cp4_n$, and $TC(cp3, cp4)$ must be changed to $\forall i \in [1, \ldots, n] : TC(cp3_i, cp4_i)$.

Furthermore, the use of inverse multiplexing can lead to a sequence of de-adaptation and adaptation at the same interface. For example, a wavelength is demultiplexed from a signal on a fibre, and Ethernet packets are demultiplexed from the wavelength. This is the de-adaptation. Then, the Ethernet packets are inverse multiplexed (adapted) in multiple STS channels at the same interface.

Such sequences of demultiplexing and inverse multiplexing gives two adaptation stacks at the same interface. We coined these the external and the internal adaptation stack.

It is simple to prove that there are at most two (de-)adaptation stacks for valid descriptions of an interface: one de-adaptation stack and one adaptation stack. Two adaptation stacks can be collapsed to a single adaptation stack, and two de-adaptation stacks can be combined into a single de-adaptation stack. Furthermore, a single adaptations followed by the inverse de-adaptation function cancel each other out, and for a (de-)adaptation stack this process can be repeated. Thus an adaptation stack followed by a de-adaptations stack can be collapsed till one of the stacks is fully cancelled out, and only a single adaptation or a single de-adaptation stack remains. Unless of course, an adaptation was followed by a different de-adaptation, which would leave us with an invalid (non-working) interface description.

4.6.3 Broadcast and Multicast

ITU-T G.805 does not explicitly support broadcast and multicast. Our model can describe broadcast networks using multiple subnetwork connections. This scales with $O(n^2)$ with n the number of nodes. Since this only works fine for small broadcast networks, we added a specific description for broadcast networks to our syntax to support Ethernet VLANs. For IP and MAC layers, it is probably inevitable to define a more elaborate model for switch matrices, including lookup tables, and hop-by-hop routing.

4.6.4 Physical Layer Properties

According to G.805, a concatenation of link connections and subnetwork connections placed in series form a valid tandem connection, which is able to transport data. We followed this concept in section 4.4.5.

This assumption is not generally true on the physical layer. For example, the power loss of two individual link connections may fall within acceptable limits, but the power loss of the serial-compound link may fall outside the specified range.

G.805 implicitly considers human-engineered networks only, by assuming that if all link connections, adaptations and terminations are applied correctly, indeed everything functions properly. This is generally true on higher layers (TDM and above), but not on the physical layer, where signal degradation is an important factor to take into account.

In order to apply G.805 on the physical layer, including wireless networks, layer parameters as mentioned in subsection 4.6.1 must be defined for the network elements. For the lower layers, this includes power levels, signal degradation, cable length, and optical dispersion. For higher layers, parameters like delay and jitter may also be defined.

4.6.5 Uniqueness of Layers

ITU-T G.805 defines a *layer* in section 4.3.2 as the set X of all possible connection points of the same type. Two connection points are of the same type, if a data-transport function can be created between them.

This definition, which is taken from G.805, is ambiguous. Imagine three connection points a, b and c, where data-transport between a and b and between b and c is possible, but not between a and c. In this case, it is unclear if we are dealing with one, two or even three layers.

An example of such ambiguity is if a, b and c are Ethernet interfaces with a supporting untagged Ethernet, b supporting both tagged and untagged Ethernet at the same time and c supporting only tagged Ethernet.

Another example is if a, b, and c are all Ethernet interfaces, with interface a operating at a capacity of 10 Mbit/s, c at 100 Mbit/s and b auto-sensing supporting both 10 Mbit/s and 100 Mbit/s.

Our solution to this problem is to define interfaces with potential incompatibilities as two or more different layers. In the later example, a 10 Mbit/s Ethernet layer and a 100 Mbit/s Ethernet layer. Interface b would then support two adaptations functions. We have in fact shown this earlier in figure 4.14, where the interfaces at StarLight supported two adaptation functions.

4.6.6 Tunnels

Because there is no ordering in the layers in the G.805 model, it is entirely possible to model layer A over layer B over layer A, effectively describing network tunnels.

4.6.7 Uniqueness of Adaptations

One of our goals is to be able to describe potential incompatibilities we like to expose to path finding algorithms. We already mentioned in chapters 2 and 3 that technologies and incompatibilities change over time.

The progress in technology makes that potential incompatibilities come and go. If everyone would use 850 nm lasers, there is no need to describe the wavelength, since there are no incompatibilities. As soon as lasers with other colours are deployed, this might lead to incompatibilities, so it has to be described. However, as soon as every device is able do colour conversion on the fly, the incompatibility would again disappear.

At first Ethernet over an optical fibre may be described as an adaptation of Ethernet over fibre. However, later on, the same adaptation may be described as Ethernet over a wavelength over a fibre. A mechanism is needed to described that these two representations are in fact the same.

4.7 Conclusion

At the beginning of this chapter we set two goals: a model for multi-layer networks and an algebra to validate potential connections through a given network.

We fulfilled the first goal with a mapping from network elements to function elements. We satisfied the second goal with a simple algebra, without relying on complex path constraints.

Our mapping from network elements to functional elements is based on previous work in the ITU-T G.805, G.800 and GMPLS standards. Our contribution is the use of the subnetwork matrix to represent the switching capability of a device, the integration of the label concept, the distinction between the switching and swapping capability. Furthermore, we supplemented the model with an algebra, and confirmed the mapping with two example networks.

To validate a network connection, we postulate a network as a set of connection points, label values, and links, and the network configuration as a set of subnetwork connections and labels. Using this information and a recursive definition for link connections and tandem connections, we can deduce information about the validity of network connections.

In section 4.5, we have explained how our approach is successful in detecting possible and impossible network connections in case of multiple incompatible adaptation functions in the network.

A technology-independent network model, as we defined in this chapter, means that a path finding algorithms only needs to know about the generic concepts such as 'layer', 'label' and 'adaptation', but not about the specific technologies. The advantage is that path finding algorithms does need to be tuned or adjusted as new network technologies come along.

Chapter 5

Network Description Language

This chapter is based on *Using RDF to Describe Networks* by J.J. van der Ham, F. Dijkstra, F. Travostino, H.M.A. Andree and C.T.A.M. de Laat [a3] and on *A Distributed Topology Information System for Optical Networks Based on the Semantic Web* by J.J. van der Ham, F. Dijkstra, P. Grosso and R. van der Pol, A. Toonk and C.T.A.M. de Laat [a9]. This chapter would not have been possible without the kind permission of Jeroen van der Ham to include part of his work here.

5.1 Introduction

The routing step, which is required for provisioning of circuit switched network connections, is responsible for distributing topology information and network state across different domains. This chapter examines the distribution of topology information. It presents the Network Description Language (NDL), which builds upon the Resource Description Format (RDF) [u12] and its linking capabilities to produce a distributed Topology Knowledge Base (TKB) [p41]. It is worth to emphasise that proposed network description language is only a method to describe topology information. It does not enforce a specific way of distributing this information, nor does it eliminate the need for a control plane for signalling and provisioning.

The modelling process, as outlined in figure 4.2 in the previous chapter 4, consists of two steps, the mapping from network to model and the mapping from model to syntax. In chapters 2 and 4 took the first step. Chapter 2 defined the terminology and model for transport networks and chapter 4 defined a model for multi-layer networks. In this chapter and the following chapter we take the second step. This chapter defines a syntax for describing (a single layer of a) transport network, while the next chapter defines a syntax for describing multi-layer networks.

5.2 Introduction to the Semantic Web

The World Wide Web has allowed us to publish and share documents and information with other people in the world. However, because the web has become so popular and so widespread, it has almost become the victim of its own success. Because of the large-scale and the abundant availability of data, it becomes very hard to find what we want. Search-machines, such as Google or Yahoo, have come to the rescue and have indexed the data. However, computers still have no common sense, so the search capabilities of the search machines are rather limited. Consider for example the following two sentences:

- A is connected to B.
- There is a connection between A and B.

Even humans can differ in opinion whether these two sentences have the same meaning. So there is no way that a computer without common sense will understand that these two lines mean the same thing. This is where the Semantic Web comes to the aid of computers (and people). The following is an excerpt of the activity statement of the Semantic Web initiative [u14]:

> *The goal of the Semantic Web initiative is to create a universal medium for the exchange of data where data can be shared and processed by automated tools as well as by people. For the Web to scale, tomorrow's programs must be able to share and process data even when these programs have been designed totally independently.*

In 2000 the Semantic Web initiative was started by the World Wide Web Consortium (W3C). Since then they have been working on several specifications to publish and share (meta)data, including the Resource Description Framework (RDF) [u12]. In the following section we provide a brief introduction to RDF. Examples of RDF are given later on in section 5.3.

5.2.1 Resource Description Framework

The Resource Description Framework (RDF) is a method for representing information about resources on the Web. It provides a common framework for expressing metadata so that it can be exchanged between applications without loss of meaning.

Information in RDF is expressed as statements. Each statement is a triplet, with the following elements:

Subject The resource being described

Predicate The property of the subject that is described

Object The value of the property for the subject

A set of triplets is called a graph. Using the property that an object can also be the subject of another triplet, complex graphs can be created. An example of such a graph is shown in figure 5.1. This graph shows that `Thesis` is written by `Freek Dijkstra` and it also provides some additional information about him, his name, affiliation and email address.

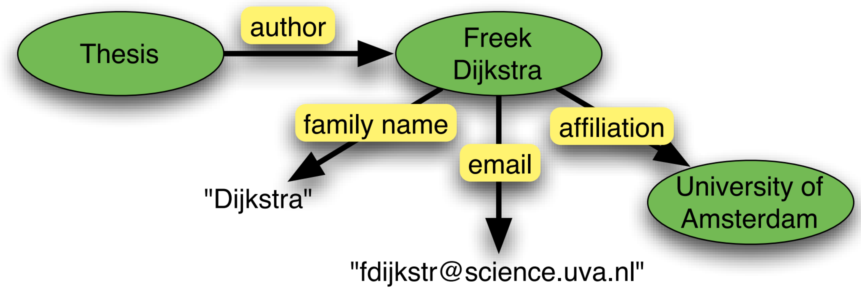


Figure 5.1: *A simple RDF graph (source: Jeroen van der Ham)*

The graph shown in figure 5.1 still has the same problem as the two lines as shown before; we have provided an abstract way of defining relations, but we still use plain English as labels for identifying these relations. Consider for example the `author` relationship, we could also have expressed this as `creator` without much loss of meaning to human readers. RDF solves this terminology problem by using Uniform Resource Identifiers (URIs). Related terms are usually defined using the same URI-prefix, taking the form of XML namespaces. See for example the Dublin Core Metadata Initiative [s58].

5.2.2 RDF Schemata

```
<http://www.science.uva.nl/~fdijkstr/thesis/> <http://purl.org/dc/elements/1.1/creator>
      <http://www.macfreek.nl/freek/#freek>.
<http://www.macfreek.nl/freek/#freek> <http://xmlns.com/foaf/0.1/family_name> "Dijkstra".
<http://www.macfreek.nl/freek/#freek> <http://xmlns.com/foaf/0.1/mbox>
      "fdijkstr@science.uva.nl".
<http://www.macfreek.nl/freek/#freek>
      <http://www.w3.org/2002/01/p3prdfv1#thirdparty.employer> <http://www.uva.nl/>.
```

Listing 5.1: *The N3 representation of the semantic graph of figure 5.1.*

Listing 5.1 describes the semantic graph of figure 5.1. Each triplet contains three URIs followed by a dot to indicate the end of a triplet. The notation used in this listing is N-triplet, which is a subset of the Notation3 syntax [t5].

The predicates defined in this list are URIs with a well-defined meaning. For example, the `author` relationship is defined by the URI `http://purl.org/dc/elements/1.1/creator`, which is defined by the Dublin Core Metadata Initiative [s58] in their Metadata Element Set. Related terms are defined in the same XML namespaces with the URI-prefix `http://purl.org/dc/elements/1.1/`.

```
<http://www.macfreek.nl/freek/#freek> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://xmlns.com/foaf/0.1/Person>.
```

Listing 5.2: *Use of the rdf:type predicate.*

An XML namespace with definitions of related terms is called an RDF schema. RDF schemata define the URIs and properties of RDF classes and RDF predicates. RDF classes define the types of subjects and objects. Listing 5.2 defines `Freek Dijkstra` to be (an instance of) a person. RDF predicates define the properties of attributes of instances. For example the Dublin Core definition for `creator` gives a name, description, type and version information. A predicate definition can also contain the compulsory classes for the subject (the domain of a predicate) and object (the range of a predicate).

It is possible to state triplets, but only if these statements use predicated and classes defined in RDF schemata, a computer can reason about its meaning. For example, a computer can find other documents created by the same author, or reason that the range of an author is a human being, and the author of this thesis is Freek Dijkstra, then Freek Dijkstra must be a human being.

5.2.3 RDF versus XML

There are several ways of expressing RDF graphs, one is the graphical form as in figure 5.1, and another is the statements of triplets in Notation3 in listing 5.1. The most common textual form is *RDF/XML* [s55], where the graph is encoded in an XML format. Throughout this thesis we use the RDF/XML notation, which allows us to leverage tools for XML as well as RDF. Examples and explanation of the RDF/XML syntax are given in the next section.

Besides that RDF allows *reasoning* about statements, it also has a few other technical advantages over other descriptions languages, such as plain XML.

Unique Identification Objects in RDF are identified by a URI. This is an advantage in multi-domain environments, since it makes it easy to clearly and uniquely define network elements in requests.

Flexible Graph Structure The relations between network elements can lead to cycles in the relation-graph. RDF extends the tree structure of XML with reference pointers so that it is able to deal with cycles.

Distributed Descriptions In order to describe inter-domain connections, the interrelation of different (operational) network domains must be described. Each domain must be able to independently publish its own network information and point to other network domains. The RDF `seeAlso` predicate provides an elegant solution for this problem. We will get back to this in section 5.3.3.

Extendable RDF schemata are easily extensible. That is, it allows users to publish all information they care about, and mix it with network schemata. The extensibility applies to both current schemata (e.g. geographic information or organisational information in `geo` and `vcard`), as well as future schemata.

5.3 Network Description Language

Given that we want to describe extensible, distributed network descriptions, we set out to create a simple ontology in RDF to describe networks. The result of this work is the Network Description Language (NDL).

5.3.1 Topology Schema

The Network Description Language consists of multiple schemata, each describing a separate part of the ontology.

The topology schema of the network description language is the ontology we created to describe the topology of computer networks. An overview of the classes and properties of the topology schema is given in figure 5.2.

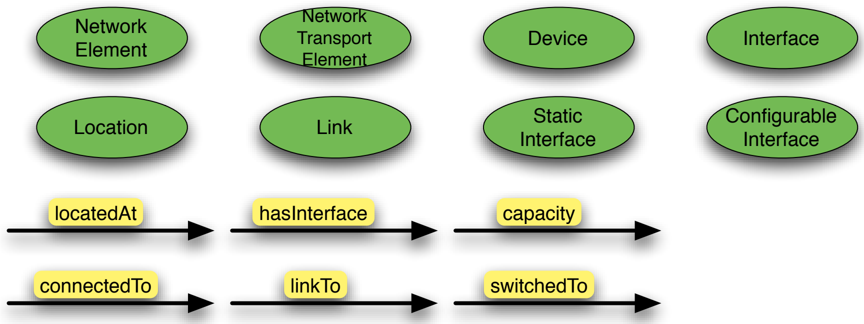


Figure 5.2: *Overview of the classes and predicates in the NDL topology schema*

NDL has eight classes, shown at the top, that define what kind of resources can be described. The three main classes are:

Location Physical places where devices are located.

Device Devices that are part of a network.

Interface The interfaces with which devices are connected to a network.

NDL has six properties, shown at the bottom in the figure, to define the relations between instances of the classes and other information.

locatedAt A relation between resources and their location.

hasInterface A relation between devices and interfaces.

linkTo A relation between two interfaces, describing that they are *externally* connected with a direct link.

connectedTo This property is similar to *linkTo*, but the connection does not have to be direct; the interfaces may be connected by a series of links.

switchedTo This property is used to describe cross connects, *internal* connections within a device.

In addition, the predicates *label* from the RDF schema, and *description* from the Dublin Core schema can be used to describe the name and description of network elements.

A more extensive definition of the different classes and predicates can be found in the NDL schemata itself [u3].

5.3.2 Domain Schema

The topology schema allows the description of physical network topologies. The NDL domain schema also allows group description of devices, links and interfaces in networks. The classes and predicates of the domain schema are shown in figure 5.3.

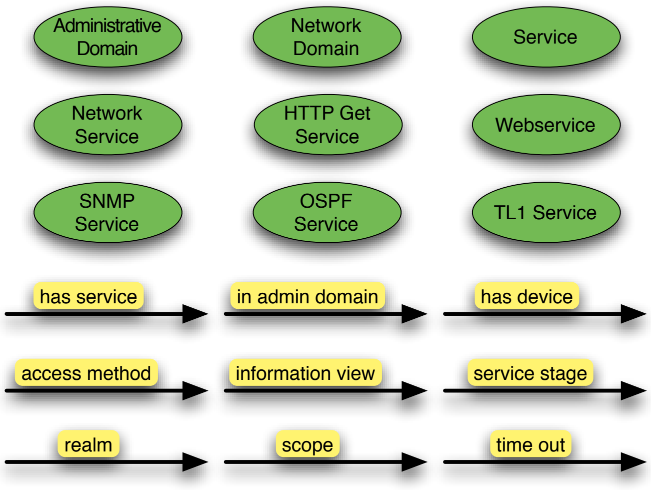


Figure 5.3: *Overview of the classes and predicates in the Network Description Language domain schema*

The two main classes in the domain schema are:

NetworkDomain is a collection of network elements. It behaves very similar to a Devices in the topology schema, but describes a domain rather than a physical device.

AdministrativeDomain is a organisational entity that is responsible for the operational control of resources (including network resources).

While a network domain is focused on the physical network, the operational domain is focused on the organisation of the control plane. An operator domain as defined in section 2.4.3 translates to a NetworkDomain (a collection of network elements with the same operator). The operator as defined in section 2.4.3 translates to the AdministrativeDomain[1].

Using the combination of the topology and domain schema, it is possible to create descriptions of network domains. An example of such a description is shown in listing 5.3. The picture in figure 5.4 shows what is being described.

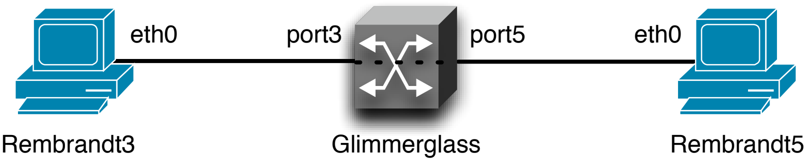


Figure 5.4: *A simple network.*

Lines 14 to 18 of listing 5.3 define the device `Rembrandt3`. The #-prefix on line 14 states that the device is defined in the local namespace. Line 15 provides a human readable name and line 16 states that this device is located in the location `Lighthouse` (defined on lines 11 to 13). Finally, line 17 defines that `Rembrandt3` has an interface, `Rembrandt3:eth0`. This interface is defined on lines 19 to 22. The connection to another interface is defined using the `linkTo` property on line 21, in this case it is defined to be connected to `Glimmerglass:port3`. The `Glimmerglass` device is defined similarly on lines 23–38, and the `Rembrandt5` device on lines 39–47.

The connection between the `Rembrandt3` and the `Glimmerglass` is defined in both directions. This is used to denote a duplex connection and further ensures the consistency of the description.

Our network description does not only contain a topology description, but also describes the current configuration of the Glimmerglass device. The `switchedTo` statement in line 32 states that the `Glimmerglass:port3` has an *internal* connection to `Glimmerglass:port5`.

Just like the `linkTo` property, the `switchedTo` property must be defined in both directions. So the inverse `switchedTo` property from `Glimmerglass:port5` to `Glimmerglass:port3` is also be given on line 37. With the `linkTo` and `switchedTo` statements as given above, we have defined a path from the device `Rembrandt3` to `Rembrandt5`.

[1]These class names resemblances an older terminology we used.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:ndl="http://www.science.uva.nl/research/sne/ndl#"
        xmlns:domain="http://www.science.uva.nl/research/sne/ndl/domain#">
   <domain:NetworkDomain rdf:about="#UvALight">
      <rdf:label>UvA Light</rdf:label>
      <domain:hasDevice rdf:resource="#Rembrandt3"/>
      <domain:hasDevice rdf:resource="#Rembrandt5"/>
      <domain:hasDevice rdf:resource="#Glimmerglass"/>
   </domain:NetworkDomain>
   <ndl:Location rdf:about="#Lighthouse">
      <rdf:label>Lighthouse</rdf:label>
   </ndl:Location>
   <ndl:Device rdf:about="#Rembrandt3">
      <rdf:label>Rembrandt3</rdf:label>
      <ndl:locatedAt rdf:resource="#Lighthouse"/>
      <ndl:hasInterface rdf:resource="#Rembrandt3:eth0"/>
   </ndl:Device>
   <ndl:Interface rdf:about="#Rembrandt3:eth0">
      <rdf:label>eth0</rdf:label>
      <ndl:linkTo rdf:resource="#Glimmerglass:port3"/>
   </ndl:Interface>
   <ndl:Device rdf:about="#Glimmerglass">
      <rdf:label>Glimmerglass</rdf:label>
      <ndl:locatedAt rdf:resource="#Lighthouse"/>
      <ndl:hasInterface rdf:resource="#Glimmerglass:port3"/>
      <ndl:hasInterface rdf:resource="#Glimmerglass:port5"/>
   </ndl:Device>
   <ndl:Interface rdf:about="#Glimmerglass:port3">
      <rdf:label>port3</rdf:label>
      <ndl:linkTo rdf:resource="#Rembrandt3:eth0"/>
      <ndl:switchedTo rdf:resource="#Glimmerglass:port5"/>
   </ndl:Interface>
   <ndl:Interface rdf:about="#Glimmerglass:port5">
      <rdf:label>port5</rdf:label>
      <ndl:linkTo rdf:resource="#Rembrandt5:eth0"/>
      <ndl:switchedTo rdf:resource="#Glimmerglass:port3"/>
   </ndl:Interface>
   <ndl:Device rdf:about="#Rembrandt5">
      <rdf:label>Rembrandt5</rdf:label>
      <ndl:locatedAt rdf:resource="#Lighthouse"/>
      <ndl:hasInterface rdf:resource="#Rembrandt5:eth0"/>
   </ndl:Device>
   <ndl:Interface rdf:about="#Rembrandt5:eth0">
      <rdf:label>eth0</rdf:label>
      <ndl:linkTo rdf:resource="#Glimmerglass:port5"/>
   </ndl:Interface>
</rdf:RDF>
```

Listing 5.3: *An example description of the network of figure 5.4.*

5.3.3 Distributed Repositories

So far we have described how to create descriptions of (local) networks and how to gather information from these descriptions.

In multi-domain environments there is a big potential for inconsistencies if information for each domain is not centrally maintained, and each domain replicates network descriptions of other domains. NDL addresses this issue by creating a distributed topology description, where descriptions link to each other. These links are provided using RDF's `seeAlso` statement, which points to other documents. An example of this is shown in listing 5.4.

```
1 <ndl:Interface rdf:about="#Glimmerglass:port27">
2    <ndl:name>Glimmerglass:port27</ndl:name>
3    <ndl:linkTo rdf:resource="http://www.netherlight.nl/ndl.rdf#C6509:port7"/>
4 </ndl:Interface>
5 <!-- test -->
6 <ndl:Interface rdf:about="http://www.netherlight.nl/ndl.rdf#C6509:port7">
7    <rdfs:seeAlso rdf:resource="http://www.netherlight.nl/ndl.rdf"/>
8 </ndl:Interface>
```

Listing 5.4: *Example of distributed repositories.*

As shown before, lines 1 to 4 show the description of an interface of the `Glimmerglass`. However, note that in line 3, this port is defined to be connected to `http://www.netherlight.nl/ndl.rdf#C6509:port7`. On lines 5 to 7 is the definition of the interface `http://www.netherlight.nl/ndl.rdf#C6509:port7`. The `rdfs:seeAlso` statement is used to link to the network description of NetherLight. An application or crawler can then follow this pointer to the description of NetherLight and get more information about the interface there.

Concluding, it is possible to create a distributed network description, without a central repository.

5.3.4 Addressing

RDF uses Uniform Resource Identifiers (URIs) [s21] to denote classes, properties, and instances of classes. For example, each Device or Interfaces is identified using a URI. This provides an elegant solution to make sure each object is unique, since only the owner of a certain domain may publish authoritative information of objects with an URI in his namespace.

In addition, the use of URIs along with explicit properties for each object follows the separation of naming and addressing [t14, t7]. This is an important concept, since the name of an object does not change, even if its properties do change.

Concluding, the use of semantic web provides the URI as a solution for globally unique addressing of network resources. . The essential advantage is that information is kept only at the owner of the resource.

5.3.5 Extensibility

The use of RDF allows easy extension of the topology and domain schema with other schemata. In chapter 6 we will see extensions to layer and capability descriptions.

Our vision is that a application will be able to consume the descriptions of all the architectural components that form an end-to-end infrastructure. This information include computing resources, storage resources, visualisation resources, network resources, content descriptions, etc. etc. All resources can be linked with loose couplings to allow a metascheduler application to orchestrate all resources together in a combined effort [a6].

5.4 Applications

NDL provides a powerful language to solve many of the operational issues that operators and users face in hybrid networks. It allows the automatic creation of network maps; it facilitates path finding algorithms used by reservation and network management systems; it enhances the interoperability and the exchange of information between different operational domains.

This section highlights a few of these applications [p15, a9].

5.4.1 Visualisation using RDF tools

Network maps are a visual aid used by network engineers to design, examine and troubleshoot circuits.

One of the advantages of using NDL as the language for description of hybrid networks is the availability of semantic web tools for RDF, which can parse and consume the information in each NDL file. This means that extracting the information needed for network management is straightforward and simple.

Van der Ham implemented a visualisation tool that takes a network description in NDL format, and uses a SPARQL query [s56] to get the connections between the devices and their names. Using a small script, this data is then converted to serve as input to GraphViz, an open source graph visualisation tool [u9].

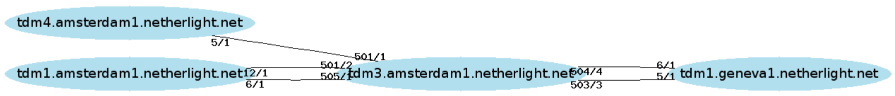

tdm4.amsterdam1.netherlight.net
5/1
501/1
tdm1.amsterdam1.netherlight.net
12/1
6/1
501/2
505/1
tdm3.amsterdam1.netherlight.net
504/4
503/3
6/1
5/1
tdm1.geneva1.netherlight.net

Figure 5.5: *A graph of NetherLight resources, as extracted from a NDL file (Source: Van der Ham [p15])*

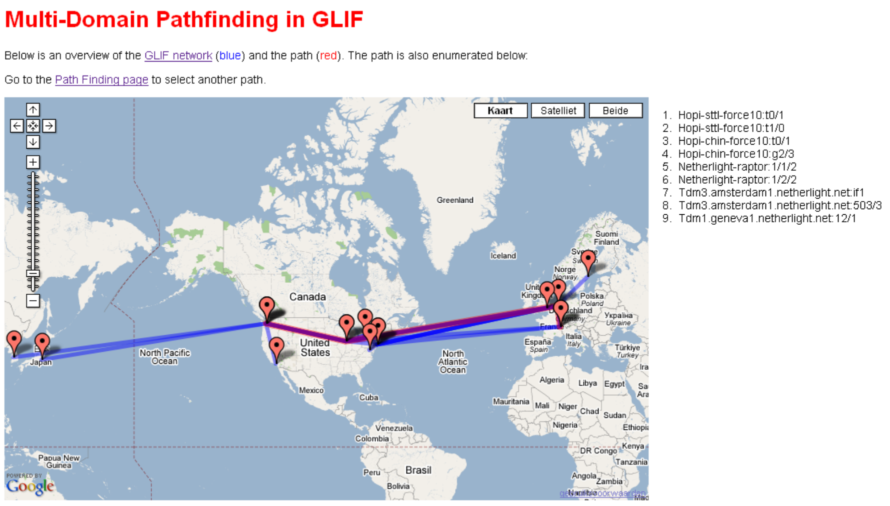
An example of such a graph is shown in figure 5.5. This is the map of NetherLight [u11], one of the network domains participating in the GLIF. The script used to generate the graph can be found at the NDL homepage [u3].

5.4.2 Path Finding and Google Mash-up

At SuperComputing 2006, Van der Ham *et al.* demonstrated path finding in the GLIF infrastructure [p14]. This infrastructure consists of interconnected optical exchanges. For this demonstration most of the GOLEs provided a description of their network in NDL format. A key feature is that each description was published independently of the others, allowing GOLEs to stay in charge of the data they publish. The descriptions defined the physical resources in the GOLEs and the links to other GOLEs.

An application gathered all the NDL files from the different GOLEs, and generates a list of endpoints. Using a web interface, a user can select two endpoints from this list, and the application applies the Dijkstra algorithm [p11] to find the shortest path between the two endpoints. The resulting path is displayed in the web browser as a highlighted path through the network graphically presented using Google Maps. A list of hops is also provided next to the map. Figure 5.6 shows the example output for a path between Seattle and Geneva.

This application is mostly a proof of concept, because there are still more challenges with regard to inter-domain path finding. Issues such as policy and multi-layer network need to be addressed, as well as information regarding utilisation. The NDL domain schema can help here by providing pointers to relevant information services on policy, utilisation, or reservations. This is part of future implementations.


Multi-Domain Pathfinding in GLIF

Below is an overview of the GLIF network (blue) and the path (red). The path is also enumerated below:

Go to the Path Finding page to select another path.

1. Hopi-sttl-force10:t0/1
2. Hopi-sttl-force10:t1/0
3. Hopi-chin-force10:t0/1
4. Hopi-chin-force10:g2/3
5. Netherlight-raptor:1/1/2
6. Netherlight-raptor:1/2/2
7. Tdm3.amsterdam1.netherlight.net:if1
8. Tdm3.amsterdam1.netherlight.net:503/3
9. Tdm1.geneva1.netherlight.net:12/1

Figure 5.6: *Path finding in GLIF, presented in Google Maps.*

5.4.3 Lightpath Planning in SURFnet6

SURFnet6 is the Dutch national research and education network. SURFnet6 is a hybrid network, offering both IP services and lightpath services. Toonk and Van der Pol of the Dutch national supercomputing centre SARA have written a tool for planning new lightpaths based on NDL [p30, p31]. In this application they automatically generate an NDL topology description of the SURFnet6 network based on information gathered from network devices using their TL1-Toolkit [u7]. Additionally, a network state database holds the cross-connect information for each network element in the network. That is, information about currently provisioned lightpaths. This enables the application to determine the amount of time-slots still available on each interface.

A network operator at the SURFnet 6 network operations centre (NOC) can use a web interface to query for a lightpath between two endpoints of the SURFnet6 network. The user first selects two endpoints from a list of the available endpoints and specifies some properties for this new lightpath, such as a name, the capacity and whether this should be a protected or unprotected path. Subsequently, a software tool uses the information from the NDL file to construct a graph of the network. Using the network state database, this graph is pruned taking out sections that do not have enough bandwidth available. Then the Dijkstra algorithm is applied using the current load of the network as link metric.

5.4.4 Lightpath Monitoring in NetherLight

NDL can play an important role in lightpath monitoring as well. SARA developed Spotlight, a tool for lightpath monitoring in SURFnet 6 and in NetherLight.

To monitor the lightpaths, SARA uses NDL to specify their topology details, and actively query the network elements involved. The output is stored in a network state database with alarm and configuration information. This enables us to correlate the configuration data with the alarm information and determine whether a specific lightpath is up or down. If a failure is detected somewhere in the lightpath route, this will be clearly indicated using a visualisation of the lightpath. The Spotlight application is available online, see [u5].

5.5 Conclusion

In this chapter we have introduced a way of applying the Resource Description Framework to describing networks by way of the Network Description Language. By using the linking capabilities of RDF, a distributed Topology Knowledge Base as described by Travostino [p41] can easily be created.

NDL unlocks the potential of machine-readable metadata about the network for control-planes, service planes and other applications that require data about the network. The goal of NDL is to provide a common topology information base and thus facilitate interworking between networks.

By leveraging the `seeAlso` property in RDF, it is possible to create a distributed (decentral) information system to describe networks.

The use of semantic web solves the addressing problem by providing URIs as globally unique addresses.

The investment in codifying models and RDF vocabularies pays dividend when new tools emerge that harvest upon these models, and unite network descriptions across administrative boundaries. By building upon existing Semantic Web techniques, applications are now appearing.

Chapter 6

Multi-Layer Network Description Language

6.1 Goal

In section 3.3.3 of chapter 3 we asked ourselves if it is possible to create a technology-independent model and syntax to describe current multi-layer transport networks?

In the previous chapter, we introduced the Network Description Language (NDL) to describe transport networks. However, the schemata presented there did not provide tools to describe the multi-layer properties of these networks. In this chapter we will extend the syntax by providing the multi-layer and technology schemata. So far, we only applied the model to a simplified model of STS, which did not contain inverse multiplexing. In this chapter, we also examine the extend to which we can apply the same model to other technologies. Our explicit goal is to do so without sacrificing the technology independence of the model.

We do so by mapping the functional elements defined in chapter 4 into a syntax.

6.1.1 Scope

In this chapter, we will test the applicability of our model and apply it to as many technologies as possible, including at least all technologies that are in use in the GLIF community:

- Ethernet VLANs
- SONET and SDH
- WDM with wavelength selective switches

- Photonic cross connects

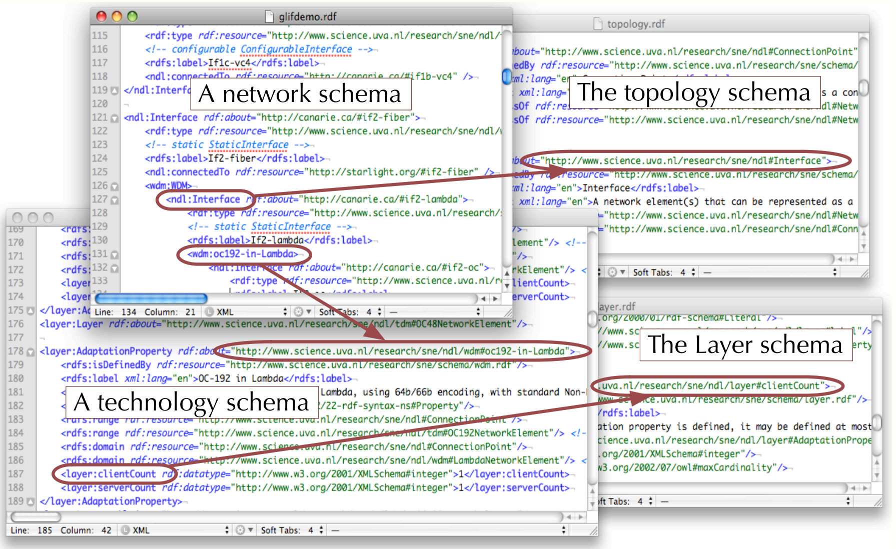
Other technologies we examine are: MPLS, ATM, PPP, Ethernet Q-in-Q [s4], IP, VPN tunnel, wireless, and fibre bundles in a trunk.

6.1.2 Technology Independence

In section 2.3 we have seen that technologies (and thus incompatibilities) change over time. Thus **what needs to be described changes over time**.

Any good path finding algorithm is network independent; it should not need to be modified to take the specifics of a particular network into account. Similarly, **a multi-layer path finding algorithms should be layer independent**; it should not need to be modified to take the specifics of a particular technology into account.

If a path finding algorithm is technology-specific, that means it needs to be updated as new network technologies come along. In chapter 3 we claimed that technologies and incompatibilities change over time. We have also seen that multi-layer incompatibilities can not be resolved locally, but that it crosses



```
    <rdf:type rdf:resource="http://www.science.uva.nl/research/sne/ndl/
    <!-- configurable ConfigurableInterface -->
    <rdfs:label>If1c-vc4</rdfs:label>
    <ndl:connectedTo rdf:resource="http://canarie.ca/#if1b-vc4" />
</ndl:Interf

<ndl:Interface rdf:about="http://canarie.ca/#if2-fiber">
    <rdf:type rdf:resource="http://www.science.uva.nl/research/sne/ndl/
    <!-- static StaticInterface -->
    <rdfs:label>If2-fiber</rdfs:label>
    <ndl:connectedTo rdf:resource="http://starlight.org/#if2-fiber" />
    <wdm:WDM>
        <ndl:Interface rdf:about="http://canarie.ca/#if2-lambda">
            <rdf:type rdf:resource="http://www.science.uva.nl/research/
            <!-- static StaticInterface -->
            <rdfs:label>If2-lambda</rdfs:label>
            <wdm:oc192-in-Lambda>
                <ndl:Interface rdf:about="http://canarie.ca/#if2-oc">
                    <rdf:type rdf:resource="http://www.science.uva.nl/r
```

Line: 134 Column: 21 XML Soft Tabs: 4

```
bout="http://www.science.uva.nl/research/sne/ndl#ConnectionPoint"
edBy rdf:resource="http://www.science.uva.nl/research/sne/schema/
ml:lang="en
 xml:lang="                                                     s a con
sOf rdf:re                                                    dl#Netw
sOf rdf:resource="http://www.science.uva.nl/research/sne/ndl#Netw


bout="http://www.science.uva.nl/research/sne/ndl#Interface">
edBy rdf:resource="http://www.science.uva.nl/research/sne/schema/
ml:lang="en">Interface</rdfs:label>
 xml:lang="en">A network element(s) that can be represented as a
="http://www.science.uva.nl/research/sne/ndl#Netw
="http://www.science.uva.nl/research/sne/ndl#Conn
```

Soft Tabs: 4

```
    <rdfs:
    <rdfs:
    <rdfs:
    <rdfs:
    <layer
    <layer
</layer:Ad
<layer:Layer rdf:about="http://www.science.uva.nl/research/sne/ndl/tdm#OC48NetworkElement"/>

<layer:AdaptationProperty rdf:about="http://www.science.uva.nl/research/sne/ndl/wdm#oc192-in-Lambda">
    <rdfs:isDefinedBy rdf:resource="http://www.science.uva.nl/research/sne/schema/wdm.rdf"/>
    <rdfs:label xml:lang="en">OC-192 in Lambda</rdfs:label>
    <                                    Lambda, using 64b/66b encoding, with standard Non-
    <                                    /22-rdf-syntax-ns#Property"/>
    <rdfs:range rdf:resource="http://www.science.uva.nl/research/sne/ndl#ConnectionPoint"/>
    <rdfs:range rdf:resource="http://www.science.uva.nl/research/sne/ndl/tdm#OC192NetworkElement"/> <!-
    <rdfs:domain rdf:resource="http://www.science.uva.nl/research/sne/ndl#ConnectionPoint"/>
    <rdfs:domain rdf:resource="http://www.science.uva.nl/research/sne/ndl/wdm#LambdaNetworkElement"/> <
    <layer:clientCount rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1</layer:clientCount>
    <layer:serverCount rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1</layer:serverCount>
</layer:AdaptationProperty>
```

Line: 185 Column: 42 XML Soft Tabs: 4

```
.org/2000/01/rdf-schema#Literal"/>
//www.sc                                              l"/>
//www.sc                                              erty


.uva.nl/research/sne/ndl/layer#clientCount">
www.science.uva.nl/research/sne/schema/layer.rdf"/>
/rdfs:label>
ation property is defined, it may be defined at most
cience.uva.nl/research/sne/ndl/layer#AdaptationPrope
.org/2001/XMLSchema#integer"/>
w3.org/2002/07/owl#maxCardinality"/>
```

Soft Tabs: 4

Figure 6.1: *A network description relies on the topology and specific technology schemata. The technology schemata rely on the layer schema.*

multiple domains. This infers that if a new technology comes along, all domains need to update their path finding algorithms. This is not realistic.

Graphs and the Network Description Language are network-independent. They provide simple building blocks to describe networks. Similarly, we want our multi-layer network description to be able to be layer independent by providing building blocks to describe technologies.

This is a clear de-coupling of topology and technology information.

Figure 6.1 shows our implementation of the decoupling between topology and technologies. A network description creates instances of classes defined in the topology schema and in one or more technologies schemata (such as Ethernet, WDM or TDM). The technology schemata are defined as subclasses or instances of classes defined in the layer schema. A path finding algorithm should only have knowledge of the topology schema and layer schema, and learn about a specific network or about specific technologies by reading specific descriptions based on these schemata. With that information, it can find a path.

6.2 NDL Schemata

The implementation of the model is done in RDF. This is a natural extension of the Network Description Language (NDL) presented in chapter 5, which also uses RDF.

The new NDL classes and properties are organised in five modular schemata:

Topology schema that describes the concept of devices, interfaces and connections between them on a single layer;

Layer schema that describes the concept of network layers, and the relation between network layers;

Capability schema that describes device capabilities, rather than just the current state of devices;

Domain schema that describes grouping of network elements in operational domains, describes services, and allows an abstracted view of the network in a domain;

Physical schema that describe locations and physical properties of network elements.

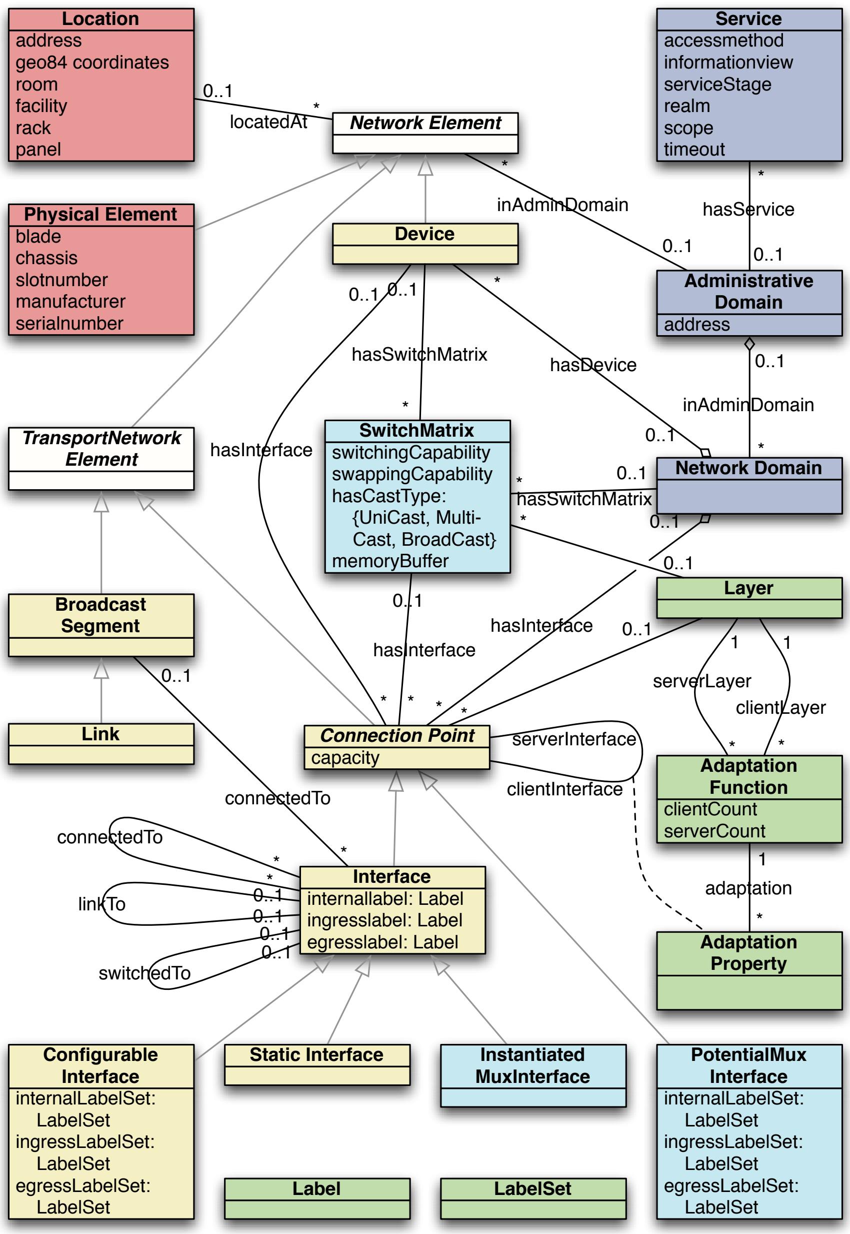


Figure 6.2: *UML representation of the NDL schemas*

The precise schemata are defined in an RDF schema, published on their designated URI [u3]. Figure 6.2 shows a representation of the most important classes and properties of the schemata in the Unified Modelling Language (UML).

In this figure, yellow classes appear in the topology schema, green classes appear in the layer schema, purple classes appear in the domain schema, cyan classes appear in the capability schema and red classes appear in the physical schema[1].

The physical schema is not relevant for path finding, and we skip its details.

6.2.1 NDL Topology and Domain Schema

We already discussed the topology and domain schemata in section 5.3 of the previous chapter. It suffices to mention that we can formally describe the `linkTo` and `connectedTo` properties in ITU-T G.805 terms as unidirectional link connection and unidirectional tandem connection.

Each `Interface` as defined in the topology schema is a connection point; a *logical interface*, not a physical interface. Each channel on each layer is –by default– represented as a single logical interface.

6.2.2 NDL Layer Schema

The layer schema defines the concepts of `Layer`, `Adaptation`, `Label` and `LabelSet`. Our current implementation is to model Layers as a generic class. For example, the Ethernet Layer is modelled as an RDF class *EthernetNetworkElement*. which is of type *Class*, as well as a subclass of *Layer*. The alternative would be to make layers a class instance, and define an RDF predicate *onLayer*. The advantage of our approach is that we can use the domain and range of a predicate to define the client and server layers of adaptations (see below).

Figure 6.3 shows the classes and properties in this schema. The layer schema does not define actual adaptation functions, but instead provides a common vocabulary to describe technologies, layers and the relation between layers.

A *Layer* is a specific encoding, or set of compatible encodings in network connection.

Most *Layers* have an associated *Label Set* are used to distinguish between different channels of a multiplexing adaptation function. For example, the label on the Ethernet layer is a VLAN, and the labelset is the set on integers $0 \ldots 4095$.

[1] For historic reasons, the Location class appears in the topology schema, but semantically, it belongs to the physical schema, and we classify it as such in this thesis.

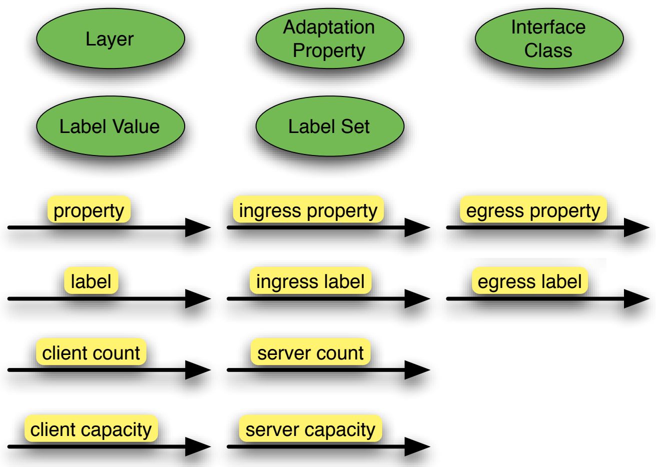


Figure 6.3: *Classes and predicates in the NDL layer schema.*

Each *Adaptation* describes a specific adaptation function, and has four properties, besides its identifier: the client layer, the server layer, the client count and the server count. The client (layer) and server (layer) refer to the *Layers* before and after the *Adaptation*. The client count represents the maximum number of client layer interfaces. The server count represents the number of required server layer interfaces. For 1:1 adaptations, the client count and server count are 1. For multiplexing adaptations, the client count is greater than 1, and the server count is 1. For inverse multiplexing adaptations, the client count is 1, and the server count is greater than 1.

The client count of a multiplexing adaptation is usually equal to the size of the label set for the client layer of the adaptation.

6.2.3 NDL Capability Schema

The NDL capability schema defines the concepts of Switch Matrix and Potential Interface.

Figure 6.4 shows the classes and properties in this schema. The capability schema allows a descriptions of the capabilities, rather than the current state of a network device or network domain.

A Switch Matrix represents a subnetwork in G.805 terminology. It represents the switching capability of a device or domain *at a single layer*. If a

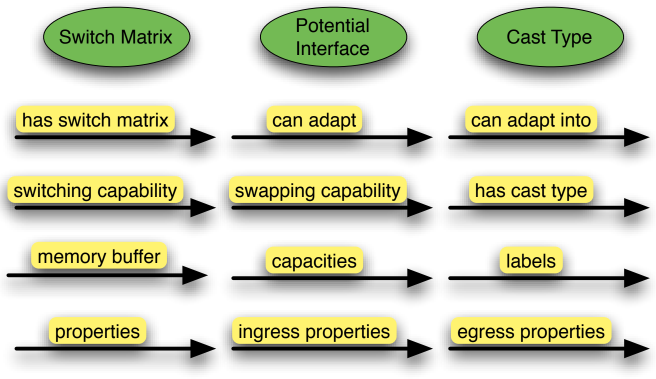


Figure 6.4: *Classes and predicates in the NDL capability schema.*

domain or switch can switch with multiple granularity, it may have multiple switching matrices: one for each layer. In general devices have exactly one switch matrix.

By default, each switch matrix can be configured to forward data from one logical interface to another logical interface. These configurations are represented by a switchTo property in NDL.

The switching capability of a switch matrix is defined by two orthogonal properties:

Switching / swapping capability Some devices are not able to convert between labels. For example, most WDM devices can not convert the wavelength, and most Ethernet switches can not change the VLAN tag. The *switching capability* represents the ability of a device to *forward data from one interface to another interface with the same label.* Two interfaces without a label are considered to have equal labels – both the “empty” label, ϵ. The *swapping capability* represents the ability of a device to *forward data from one interface to another interface with a different label.*

Cast type Most switch matrices can only unicast, meaning that it is possible to make a cross connect from one to another unused interface. Multicast switch matrixes can also make a cross connect from A to B, even if there is already another cross connect with source A. Broadcast switch matrices are entirely different: if two interfaces have the same label, then they must exchange data. Most switch matrices can not only multicast, but also merge data: there can be multiple source interfaces with the same destination.

The Potential Interface, also defined in the NDL capability schema is an optimisation. NDL defines four different interfaces in total:

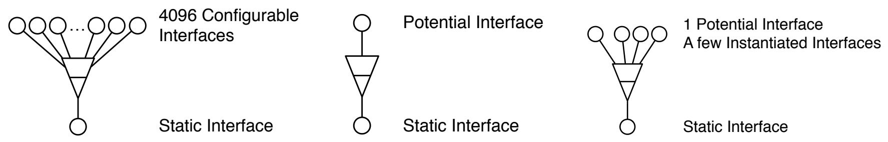
Static Interface is a non-configurable interface. E.g. a laser at a 1310 nm;

Configurable Interface is has not only an actual value, but also a range of possible values. E.g. a tuneable laser;

Potential Mux Interface is an abstract interface, and is semantically equivalent to a set of multiple optional configurable interfaces. For example the set op potential tagged VLAN channels in Ethernet; non-configurable interface. E.g. a laser at a 1310 nm.

Instantiated Mux Interface is an instantiation of a potential interface.

The static and configurable interface are defined in the topology schema. The potential mux and instantiated mux interfaces in the capability schema.



(a) *Ethernet VLANs as 4096 configurable interfaces.* **(b)** *Ethernet VLANs as 1 potential interface.* **(c)** *3 configured VLANs and 4093 potential others.*

Figure 6.5: *Use of a potential interface to describe Ethernet VLANs.*

The potential interface is used to describe that a device can create one or more logical interfaces on the fly. Typically, it is used as a shortcut for multiplexing adaptation functions. Take for example Ethernet VLANs. That would be described as 4096 logical channels in a single fibre. This could be described as 4096 *configurable interfaces* over one *static interface*, as seen in figure 6.5a. While this is fine for a model, it is not efficient for a syntax (a model can be verbose, a syntax should be compact, see section 4.4.3). Instead of saying 4096 times that a channel can be created, it is more efficient to say once that a channel can be created 4096 times. This is described using a *potential interface*, as seen in figure 6.5b. The count of 4096 can be deduced by the number of available labels for the potential interface, as well as the client count in the adaptation function. Only if certain VLANs are actually in use, then it is required to distinguish between the different configured VLANs. We use an *instantiated interface* instead of a configurable interface to signify that the interface is dynamically created and not permanent.

6.3 Technology Schemata

We proceeded to create a technology schema for each of these technologies: IP, Ethernet, ATM, PPP, MPLS, VPNs, copper, WDM, TDM (SDH/SONET), fibre, fibre bundle, wireless.

All these schemata are *technology specific*. The success of our model is determined by the ability to describe each of these schemata using *only* our technology independent model. If we are successful in doing so, that means that all technologies can be described using a technology independent model, and we can say that our *model* and path finding algorithm are still technology independent.

6.3.1 Encodings

As indicated in section 4.6.5 in the previous chapter, the choice of layers and sublayers for a technology is to some extent arbitrary. Many technologies define more than one possible encoding. An encoding defines the format of data on a link (e.g. the header and the payload, or the framing).

For all practical purposes, we define two different encodings as incompatible, and two equal encodings to be compatible.

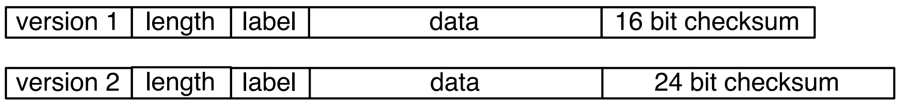


Figure 6.6: *Sample of two encodings for the same layer.*

Figure 6.6 shows a (fictitious) example of two encodings in a single technology: one with a 16-bit checksum, and one with a 24-bit checksum.

Table 6.2 lists a few more examples of layers and encoding types.

We have four options to describe different encodings, and thus to model the possible incompatibilities between network elements:

- Each encoding is modelled as a different Layer;

Layer	**Incompatible Encodings**
Ethernet	untagged, tagged or Q-in-Q tags
Ethernet	different maximum packet size (MTU)
DWDM spacing	100, 50 or 25 GHz spacing between wavelengths
Ethernet in UTP	10, 100, 1000 or 10000 Mbit/s

Table 6.1: *Examples of encoding types for different layers.*

- Each encoding is modelled as a different adaptation;
- Each encoding is modelled as different labels;
- Each encoding is modelled as a layer property.

Each method has its advantages and disadvantages.

The advantage of different layers, different adaptations or different labels is that the path finding algorithm remains technology-independent. Layer properties are technology specific, since clear rules about compatibilities and incompatibilities need to be defined

The advantage of the different adaptations, labels or the layer property is that it allows the definition of an interface that is not aware of the difference in encoding. For example, while most Ethernet interfaces can extract the packets and the VLAN information, some can not. In chapter 3, we saw an Ethernet in a SONET device, which has no knowledge of the difference between tagged an untagged Ethernet; it simple adapt all data in STS channels, without knowing the exact contents. If we would model tagged and untagged Ethernet as two layers, it would not be possible to properly describe such an interface.

Finally, the advantage of the different layers, different labels and the layer property is that they allow to describe incompatibilities at a single layer. For example, if we want to describe the maximum packet size (MTU) of Ethernet using different adaptations, it would not be possible to describe that (in)compatibility without also describing the layer above Ethernet.

When deciding how to model a certain encoding, we used the following approach:

1. If the two encodings can never appear in conjunction on the same link, model it as two different layers. For example, it is unlikely that there will be a SONET interface which can automatically switch from OC-48 to OC-192, so we modelled OC-48 and OC-192 as two distinct layers.

2. Else, if the compatibility appears in different labels, model it as a label. For example the difference between tagged and untagged Ethernet is the presence or absence of a label.

3. Else, if the incompatibility has many distinct options (such as a MTU from 1518 to 16114 bytes), then we model it as a layer property to avoid an explosion of possible adaptations or possible layers.

4. If two encodings only occur in combination with one or a few other layers, we model it as a different adaptation. For example, we model the different spacings of WDM systems (CWDM, DWDM with 25, 50 or 100 GHz spacing) as distinct adaptations.

5. If all alternatives are exhausted, model it as a (technology-specific) layer property.

In addition to the above, we define layer properties that are not directly associated with different encodings. For example, we define the power level for fibres. While this information is ignored by a technology-independent path finding, it can be used by a technology-specific path finding or fault isolation software. The advantage of RDF is that it can easily be extended with these kind of properties.

6.3.2 Layers and Labels

Table 6.2 lists the layers and label types of the technologies we modelled. The schemata are available from the NDL website [u3]. Note that there is no hierarchy in this list: in our model, all layers are treated equally.

Most of the choices are straightforward, given the method described in the previous section. In particular, MPLS, ATM, and SONET are all straightforward to model.

For SONET and SDH, we relied on the sublayering chosen by GMPLS, as defined in RFC 4606 [s26]. ATM (Asynchronous Transfer Mode) is modelled as four layers, with two VPI (Virtual Path Identifier) layers, since there are two possible VPI labels (8-bit and 12-bits long), and they are never mixed on the same link as far as we are aware.

The next few subsections discusses the technologies where the mapping to our model is not straightforward.

Technology	**(Sub)Layer**	**Label type**
IP	IP	IPv4 address
IP	IP	IPv6 address
MPLS	MPLS	MPLS label
Ethernet	MAC	MAC address
Ethernet	Ethernet	802.1q (VLAN) tags
ATM	AAL0 layer	VCI
ATM	VPI (UNI)	12-bit VPI
ATM	VPI (NNI)	8-bit VPI
ATM	ATM cell layer	*none*
SONET/SDH	VT1.5 / VC-11 layer	M label
SONET/SDH	VT2 / VC-12 layer	M label
SONET	VT3 layer	M label
SONET/SDH	VT6 / VC-2 layer	*none*
SONET/SDH	VTG / TUG-2 layer	L label
SONET/SDH	STS-1 SPE / VC-3 layer	U label
SDH	TUG-3 layer	K label
SONET/SDH	STS-3c SPE / VC-4 layer	*none*
SONET/SDH	STS-3 / AUG-1 layer	stm
SONET/SDH	OC-1 layer	*none*
SONET/SDH	OC-3 layer	*none*
SONET/SDH	OC-12 layer	*none*
SONET/SDH	OC-48 layer	*none*
SONET/SDH	OC-192 layer	*none*
SONET/SDH	OC-768 layer	*none*
SONET/SDH	OC-3092 layer	*none*
WDM	lambda layer	wavelength
WDM	fibre layer	*strand identifier*
UTP/STP	copper layer	*strand identifier*
PPP	PPP layer	*none*
L2TP	L2TP layer	*none*
802.11	802.11 layer	SSID
Fibre bundle	bundle layer	*none*

Table 6.2: *Examples of label types for different layers.*

6.3.3 Wavelength Division Multiplexing

Wavelength Division Multiplexing (WDM) technology is relatively straightforward to model. Using the switching and swapping capability, it is possible to distinguish between a wavelength selective switch (WSS) that can switch individual wavelengths. With the swapping capability it is also possible to describe a device that is able to convert between wavelengths, something which is currently not possible in commercially available devices.

The only difficulties in WDM technologies comes from the continuous (non-discrete) nature of wavelengths, the different spacings between wavelengths and differences in switching granularity. In general, the following WDM systems exist:

- A single wavelength per fibre;
- One wavelength in each optical window: 850 nm, 1310 nm, and 1550 nm;
- One wavelength in each of the O, E, S, C, L and U bands;
- Coarse Wavelength Division Multiplexing (CWDM) as defined in ITU-T G.694.2, with a spacing of 20 nm [s38];
- Dense Wavelength Division Multiplexing (DWDM) as defined in ITU-T G.694.1, with a central frequency of 193.1 THz and a spacing of 100 GHz between different frequencies [s37];
- DWDM as defined in ITU-T G.694.1, with a spacing of 50 GHz between different frequencies;
- DWDM as defined in ITU-T G.694.1, with a spacing of 25 GHz between different frequencies;
- DWDM as defined in ITU-T G.694.1, with a spacing of 12.5 GHz between different frequencies.

Since wavelengths are intrinsic properties of the wavelength layer, they are modelled as different labels. The different spacings on the other hand can either be implied by the different wavelengths (which means that $1552.52 \pm 0.41\ nm$ and $1552.52 \pm 0.21\ nm$ are different labels), by modelling it as layer properties, or as different adaptation functions of a wavelength over a fibre.

Our first attempt modelled this as a layer property, but we later realised we could model it as different adaptations, which would allow us to model different spacings in the technology independent model.

In this model, the adaptation using one spacing and de-adapt in an interface with a different spacing would be considered impossible, while in practice it may work. Also, with only two layers (wavelength and fibre layer), it is not possible to describe a switch that switches with the granularity of a group of wavelengths. Both are reasonable restrictions, and it is possible to alleviate these restrictions by another choice of layers and adaptations, without modifying the underlying model.

A more fundamental problem is the description of available wavelengths. For most multiplexing adaptation functions, the client count is equal to the size of the label set for the client layer of the adaptation. That is not true for WDM, since the label set is continuous (a float representing the wavelength) rather than discrete (such as an integer for VCI, VPI, VLANs or MPLS labels).

One way to solve this is to require the label to be an integer, defined in the context of the adaptation function, rather than the wavelength itself. For DWDM, this would be the integer n in the equation for the wavelength $\lambda = \frac{c}{f_0 + n \cdot f_s}$ with c the speed of light (299792458 m/s), f_0 the central frequency (193.1 THz), and f_s the spacing (12.5, 25, 50 or 100 GHz).

We choose to use the wavelength in nanometre as the label, in correspondence with the proposed choice in GMPLS [s35]. This allowed us to compare wavelengths regardless of their adaptation. The consequence is that our implementation always says there are available wavelengths -floats are continuous- as long as the client count of the adaptation is not set.

Finally, the switching granularity of WDM may not be static. While most devices can switch individual wavelengths, some may switch waveband: a continuous group of wavelength in a frequency range. This can be solved by modelling an additional layer. However, since this is an uncommon technology, we decided to ignore wavebands.

6.3.4 Signal Degeneration

Layer properties such as power levels have been defined in our schema, and are used for fault isolation, the localization of network configuration error [p33]. However, these properties are technology-specific. A technology-independent path finding algorithm can not use them. Our model does not define a logic for signal degeneration, and this is thus not taken into account in our path finding algorithm.

6.3.5 Shared Risk Link Groups

The generic approach of our model allow the description of shared risk links. A shared risk link is a set of fibres that use the same duct. Backup connections should not use two fibres in the same risk group because a digging machine may break both fibres in the same accident. We modelled this as a multiplexing function of multiple fibres adapted in one duct. We use the optional 'strand' label to distinguish between channels (fibres) in the same duct.

6.3.6 Packet Layers

All technologies we discussed so far create bidirectional connections, with the same label in both directions. The SONET timeslot is the same for both directions. This is generally not true for packet switched layers that use lookup tables for routing. For example, the MAC and IP layers use the destination MAC address and destination IP address to find the egress interface in a switch. In our model, we distinguish between the ingress and egress labels (see figure 4.11 in the previous chapter).

The modelling of MAC and IP requires routing tables, and many logical interfaces to describe all connections in terms of circuits. While we are technically able to distinguish between the ingress and egress labels, we found that the description of MAC and IP layers as circuits yields so many logical interfaces that we feel our model is not suitable. While it is certainly possible to extend the RDF schema to describe these kind of switches, it would also require a new logic, which we have not defined yet.

The reason for this limitation lays in the routing table nature of MAC and IP layers, not in their packet switched nature. For example, Ethernet VLANs does not suffer from this scaling issue.

6.3.7 Ethernet

It turns out that it is non-trivial to model Ethernet according to our model for three reasons: (1) Ethernet can be tagged or untagged, (2) Ethernet has different labels (VLAN and I-SID) and (3) Ethernet is a broadcast technology as opposed to a unicast or multicast technology.

As we mentioned in section 6.3.1 above, we had to model Ethernet as a single layer; otherwise we would not be able to describe an Ethernet interface in a STS device. Another advantage of this approach is that we could model tagged Ethernet as 4096 Ethernet channels in Ethernet, and similarly Q-in-Q is modelled as Ethernet in Ethernet in Ethernet.

While the client layers in the Ethernet in Ethernet adaptation has labels, the server layer Ethernet generally does not. There is a distinction that Ethernet makes between labels which is uncommon for other technologies:

Internal labels are used to decide if a cross connect can be made. In case of a switch matrix with switching, but no swapping capability, both ends of the cross connect (the subnetwork connection) must have equal labels. This is the VLAN for both tagged and untagged Ethernet.

External labels are used to distinguish between channels in a multiplexing adaptation function. This is the 802.1Q tag in tagged Ethernet, but is not defined for untagged Ethernet.

So in order to support Ethernet, a distinction must be made between internal and external labels, and untagged Ethernet interfaces must be marked as not using an external label. Alternatively, we can add the logic that a link connection over a 1:1 (non multiplexing) adaptation function does not carry a label, and thus can covert label on the wire. We prefer the explicit notation.

A second complexity is that the Ethernet switch matrix is a broadcast switch matrix. While a regular unicast switch matrix (without swapping capability) **may** make a cross connect between interfaces with same label, a Ethernet switch matrix **must** have a cross connect between interfaces with the same label.

With the addition of this logic, we were able to implement Ethernet VLANs.

6.4 Conclusion

We successfully applied the model of the previous chapter to MPLS, ATM, SONET, SDH, UTP, PPP, WDM and fibre trunk technologies. In addition, with some additional logical on broadcast matrices it was possible to model Ethernet VLANs technologies.

Since the technologies in use within the GLIF community are Ethernet VLANs, SONET, SDH and WDM, we conclude that the model we presented in chapter 4 provides a technology-independent way to describe multi-layer networks.

Chapter 7

Path Finding Algorithms

This chapter is based on *Path Selection in Multi-Layer Networks* by F. Kuipers and F. Dijkstra [a12]. The comparison between the two algorithms in this would not have been possible without the kind permission of Fernando Kuipers to include part of his work here.

7.1 Introduction

In the previous two chapters, we created a network model and syntax. The goal in this chapter is to define a path finding algorithm based on this model and syntax. We consider two algorithms: path finding in G_l in section 7.4, developed by us, and an alternative algorithm proposed by Kuipers, path selection in G_s in section 7.5.

Both algorithms operate on a graph, although the graphs (G_l and G_s) are different for each algorithm. In addition, we define a third mapping from the model we presented in chapter 4 to a graph, resulting in the graph G_p.

This chapter is organised as follows. Section 7.2 explains the terminology and notation that is used and section 7.3 presents three graph representations of a multi-layer network. In sections 7.4 and 7.5 we discuss several path selection algorithms for multi-layer graphs consisting of two layers. Section 7.6 extends this work to incompatible labels and an arbitrary number of layers. We end with the conclusions in section 7.8.

7.2 Terminology

7.2.1 Definition of a Network

Section 4.4.5 of chapter 4 defines a network $N = (CP, L, SN, A)$ as a set of connection points CP, physical links L, subnetworks SN, and adaptations A, and its configuration $C = (LB, SC)$ as a set of labels LB, and subnetwork connections SC.

For simplicity, we ignore the details of subnetwork connections in this chapter, and replace connection points and subnetwork connections by the more generic concept of nodes. This simplification allows us to focus on the principles of the algorithms without going in too much detail.

The network technology description in this chapter uses the following sets:

The set $\mathcal{Y}$ of $|\mathcal{Y}|$ layers A layer $y \in \mathcal{Y}$ is set of related encodings, whose only difference may be in the payload, the label (as defined in section 4.4.3), or the encapsulation of the data from a higher layer;

The sets $\mathcal{A}(y)$ of $|\mathcal{A}(y)|$ adaptation functions for each layer y with y the server layer of the adaptation function. Each item $\alpha \in \mathcal{A}(y_s)$ is a tuple (y_c, y_s, b_s) with $y_c, y_s \in \mathcal{Y}$ representing an adaptation from client layer y_c to server layer y_s and b_s the required bandwidth usage at the server layer. $\mathcal{A}(y)$ may contain multiple adaptation functions between the same layer pair;

The sets $\mathcal{LB}(y)$ of $|\mathcal{LB}(y)|$ possible labels for each layer y. The length of each of these sets must be at least 1. If no labels are available, this is the set $\{\epsilon\}$ (ϵ representing the empty label).

The network description in this chapter uses the following sets:

The set N of $|N|$ nodes The definition of a node depends on the granularity, as described below. Roughly, a node can be a domain, device or physical interface;

The set C of $|C|$ channels for each node n and the layers y for that node. Each element $c \in C$ is a tuple (n, y) with $n \in N$ and $y \in \mathcal{Y}$. If a node is a physical interface, this is the same as a connection point;

The set L physical links between two channels c_1, c_2 and as defined in $N = (CP, L, SN, A)$. Each element $l \in L$ is a tuple (c_1, c_2, b) with $c_1, c_2 \in C$ and b the available bandwidth on the link. A network can be either unidirectional or bidirectional;

The set A of adaptations between channel c_1 and c_2 inside a node. Each element $a \in A$ is a tuple (c_1, c_2, α), representing the adaptation function $\alpha \in \mathcal{A}$ from client layer c_1 to server layer c_2. We assume bidirectional adaptations only: an adaptation from c_1 to c_2 implies a de-adaptation from c_2 to c_1. Defined in the network description (CP, L, SN, A);

The set $LB(c)$ of all available labels for each channel $c \in C$. Defined in the network configuration (LB, SC). $LB(c)$ is a subset of $\mathcal{LB}(y)$ with y the layer of channel c.

Note that the set of adaptations A (in $N = (CP, L, SN, A)$) and the set of adaptation functions $\mathcal{A}$ are different sets. $\mathcal{A}$ is the set of adaptation functions, and describes the technology. A is a set of adaptations between two channels in a node, and describes the implementations of this technology in a network. Similarly, the set LB describes the available labels for a specific channel, while $\mathcal{LB}$ describes the labels for a specific layer.

We define $B_a\big((y_c, y_s, b_s)\big) = b_s$, the required bandwidth b_s of the server layer for each adaptation function $(y_c, y_s, b_s) \in \mathcal{A}$.

For convenience, we also define $C_n(n) \subset C$ for all $n \in N$ to be the subset of channels in node n:

$$C_n(n) = \{(n, y) \mid y \in \mathcal{Y} \wedge (n, y) \in C\} \tag{7.1}$$

Furthermore, we define $Y_c(c) \in \mathcal{Y}$ for all $c \in C$ to be the layer of channel c, and $Y_n(n)$ to be the **set** of layers for node n:

$$Y_n(n) = \{Y_c(c) \mid c \in C_n(n)\} \tag{7.2}$$

We restrict the number of channels to one channel per node per layer:

$$C \subset N \times \mathcal{Y} \tag{7.3}$$

This restriction means that each layer can only be present once in $Y_n(n)$:

$$\forall n \in N : Y_n(n) \subset \mathcal{Y} \tag{7.4}$$

Also, this restriction allows us to give an upper limit to the size of C:

$$|C| \leq |N| \times |\mathcal{Y}| \tag{7.5}$$

The downside of this restriction is that the algorithms described in this chapter do not support explicit descriptions of multiplexing and inverse multiplexing. However, the concept of bandwidth yields a surrogate way to define inverse multiplexing (surrogate, since multiplexing is not properly supported, it is always assumed there is only one channel at the client layer). We also allow that an edge is used twice for a path, so implicit multiplexing is still supported.

7.2.2 Granularity

Each of the mappings from model to graph can be done using a different granularity.

Domain granularity corresponds to mapping of domains to vertices, ignoring the intra-domain connections. With this granularity, each node $n \in \mathcal{N}$ is a domain.

Device granularity maps devices to vertices, ignoring the cross connects within a device. With this granularity, each node $n \in \mathcal{N}$ is a physical device.

Interface granularity maps interfaces to vertices, without any abstraction. With this granularity, each node $n \in \mathcal{N}$ is a physical interface.

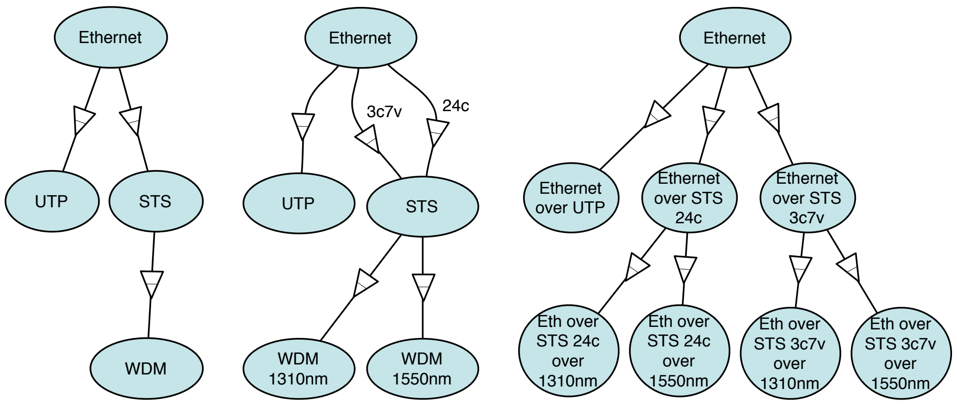
For granularity of devices, $Y_n(n)$ are the layers of the switch matrices SC, as well as the layer of the interfaces in the device. For granularity of interfaces, $C_n(n)$ are the different connection points CP that constitute the interface. For example, the layers $Y_n(n)$ of a node n which represent a Ethernet interface in a SONET switch is $\{\text{Ethernet}, \text{STS}\}$ or perhaps even $\{\text{Ethernet}, \text{STS}, \textit{lambda}, \textit{fibre}\}$

Chapter 8 describes an implementation of the first algorithm with the granularity of individual interfaces (so it can take the intrinsic details of switch matrices into account). In this chapter, we will map domains to vertices, since that corresponds to the example network we have seen in chapter 3. This also reduces the complexity of the examples.

7.2.3 Technology Stacks

An adaptation function describes the technology how data from a client layer can be embedded in the data of a server layer. An adaptation stack describes a sequence of adaptations. A technology stack, or protocol stack, describes a list of layers, where each layer acts as a server layer of the previous layer in the list.

Figure 7.1 shows three representations of a fairly common technology description. It includes four layers: $\mathcal{Y} = \{\text{Ethernet, STS, UTP, WDM}\}$ (figure 7.1a). Furthermore, there are two ways to encapsulate Ethernet over STS,



(a) *Protocol stack* **(b)** *Technology diagram, with adaptations and labels* **(c)** *Tree of all possible technology stacks*

Figure 7.1: *Three representations of a technology diagram.*

and there are two different labels at the WDM layer: $\mathcal{A}(STS) = \{3c7v, 24c\}$ and $\mathcal{LB}(WDM) = \{1310nm, 1550nm\}$ (figure 7.1b). While this diagram uses G.805 graphical convention to describe adaptations, a technology diagram can be represented as a directed graph with layers for vertices and edges from a client layer to a server layer (in the Layer schema of NDL, adaptations are predicates from server layer to client layer, only because that results in smaller RDF/XML files). This technology diagram leads to in total 8 possible adaptation stacks, provided that the top layer is Ethernet (figure 7.1c):

- Ethernet
- Ethernet over UTP
- Ethernet over STS with 24c adaptation
- Ethernet over STS with 3c7v adaptation
- Ethernet over STS with 24c adaptation over 1310nm
- Ethernet over STS with 24c adaptation over 1550nm
- Ethernet over STS with 3c7v adaptation over 1310nm
- Ethernet over STS with 3c7v adaptation over 1550nm

Given a technology description (the layers $\mathcal{Y}$, and for all layers y the adaptations $\mathcal{A}(y)$ and labels $\mathcal{LB}(y)$), and a choice for a 'root' layer, we can create a technology diagram, as displayed in figure 7.1b. The choice of root layer is significant. For example, had we chosen STS to be the root, the three possible adaptation stacks would have been:

- STS
- STS over 1310nm
- STS over 1550nm

We define a technology $t \in \mathcal{T}$ as a tuple *(layer, adaptation function, label)*.

$$\mathcal{T} = \{(y, \alpha, lb) \mid y \in \mathcal{Y} \wedge \alpha \in \mathcal{A}(y) \wedge lb \in \mathcal{LB}(y)\} \tag{7.6}$$

For convenience, we define $T(y) \subset \mathcal{T}$ as all technologies with layer y:

$$T(y) = \{(y, \alpha, lb) \mid \alpha \in \mathcal{A}(y) \wedge lb \in \mathcal{LB}(y)\} \tag{7.7}$$

The number of technologies per layer is the product of the number of adaptations and the number of different labels for that layer:

$$|T(y)| = |\mathcal{A}(y)| \times |\mathcal{LB}(y)| \tag{7.8}$$

For example, consider the tributary group layer in time division multiplexing. Seven of these groups can be embedded in the underlying layer, each identified by a different label. Thus $|\mathcal{LB}(y)| = 21$. In addition, there are two way to embed tributary groups in the underlying layer: SONET packs the 7 Virtual tributary groups (VTGs) in one STS-1 SPE, while SDH packs the 7 Tributary Unit Group (TUG-2) in one TUG-3. If we model this as incompatible adaptations, then $|\mathcal{A}(y)| = 2$, and thus $|T(y)| = 14$.

A technology stack s is an ordered list of technologies $[t_0, t_1, \ldots, t_n]$, from highest to lowest layer, with each consecutive layer the server layer of the previous layer (the client layer adaptation of t_{i+1} must have the same layer as technology t_i and its server layer must be the same as the layer of t_{1+1})

For example, Ethernet over STS with 24c adaptation over 1310nm is:

$$s = \left[(\text{Ethernet}, \textsc{nil}, \epsilon), (\text{STS}, \mathit{24c}, \epsilon), (\mathit{WDM}, \mathit{STS\ over\ WDM}, 1310.0)\right]$$

In this example, NIL means there is no higher layer, and thus no adaptation from a client layer. ϵ means the empty label.

The set of all adaptation stacks is $\mathcal{S}$.

For convenience, we define $Y_s(s)$ for every stack $s \in \mathcal{S}$ as the layer of the last technology in the stack s. For example, $Y_s([(\text{Ethernet}, \textsc{nil}, \epsilon), (\text{STS}, \mathit{3c7vc}, \epsilon)])$

= STS. Similarly, we define $\mathcal{A}_s(s)$ to be the adaptation function of the last item of the stack s, and $\mathcal{LB}_s(s)$ to be the label of the last item of the stack s.

A root technology t_1 consists of an given (root) layer y_1, no adaptation function, and a given label lb_1. Given the root technology t_1, we can formally define the set of all technology stacks, $\mathcal{S}$ as a recursion:

$$\mathcal{S}(y_1, lb_1) = \begin{cases} [(y_1, \text{NIL}, lb_1)] \vee & \text{(7.9a)} \\ \{s + [(y, \alpha, lb)] \mid t = (y, \alpha, lb) \in \mathcal{T} \wedge \alpha = (y_c, y, b_s) \wedge \\ \qquad s \in \mathcal{S}(y_1, lb_1) \wedge y_c = Y_s(s)\} & \text{(7.9b)} \end{cases}$$

Equation 7.9b denotes that any valid extension of a technology stack $s \in \mathcal{S}$ that leads to a new valid technology stack.

While the choice of the root layer is usually obvious (in the above example, Ethernet is the only layer that is not a server layer to another layer), this is not always true. In fact, the technology diagram can contain cycles in case of tunnels: the embedding of one technology in itself. For example, IP over Ethernet over IP for IP tunnels, or –in our model in section 6.3.7– Ethernet over Ethernet over Ethernet using stacked (Q-in-Q) VLAN tags.

Cycles in a technology diagram lead to an infinite size of the adaptation stack tree. In order to avoid this problem, if we work with adaptation stacks, we will require that there are no cycles in the (directed) technology diagram.

7.2.4 Definition of a Graph

Graphs $G(\mathcal{V}, \mathcal{E})$ consist of a set $\mathcal{V}$ of $|\mathcal{V}|$ vertices and a set $\mathcal{E}$ of $|\mathcal{E}|$ edges. A specific edge in the set $\mathcal{E}$ between nodes u and v is denoted by (u, v).

A path P is a sequence of edges, rather than a sequence of nodes.

Each edge $e = (u, v) \in \mathcal{E}$ from node u to node v is characterised by one or more weight parameters $W_e(e)$. The distance of a path $d(p)$ is a function of the weight parameters of each edge, typically the sum of the individual weights $W_e(e)$ for all $e \in P$.

Finally, we define $B_e(e)$ as the bandwidth b of edge e.

Graphs can be directed or undirected. For a undirected graph, the edges $(u, v) \in \mathcal{E}$ and (v, u) are the same edge. Some definitions of undirected graphs require that there can only be one edge between every pair of vertices. We explicitly require that there can be more edges between the same pair of vertices.

7.3 Multi-layer Network Model

In this section we provide three network descriptions. The first is a commonly used model in which each network device or network domain represents one vertex in a graph G_p, and physical network links are represented as edges. The second model represents each network device as multiple vertices in a graph G_l; one for each 'layer'. In this model, links still represent edges, but adaptations also represent edges. Finally, a model where we transform the multi-layer network into a graph G_s consisting of vertices and links on different 'encodings'.

7.3.1 Example Network

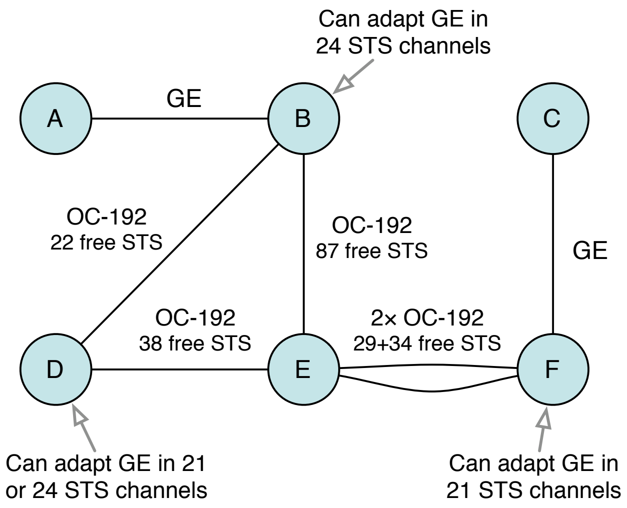


Figure 7.2: *An example of a multi-layer network, equal to the example in chapter 3. GE refers to Gigabit/second Ethernet, OC-192 are SONET-based optical carriers carrying 192 STS channels.*

Figure 7.2 is a representation of the example network we presented in chapter 3. This network consists of 6 domains, the nodes $\mathcal{N} = \{A, B, C, D, E, F\}$, and we only consider two layers $\mathcal{Y} = \{\text{Ethernet}, \text{STS}\}$, ignoring the optical carrier (OC) layer for simplicity. There are two incompatible adaptations: Gigabit Ethernet (GE) can either be adapted in 24 STS channels or in 21 STS channels (7 virtually concatenated groups of 3 concatenated channels). Since the network definition of section 7.2.1 can not explicitly represent (inverse) multiplexing, we use the concept of bandwidth instead. $\mathcal{A}(\text{STS}) = \{\mathit{24c}, \mathit{3c7v}\} = \{(\text{Ethernet}, \text{STS}, 24), (\text{Ethernet}, \text{STS}, 21)\}$. $\mathcal{A}(\text{Ethernet}) = \varnothing$.

Devices A and C are only aware of the Ethernet layer, and not of the STS layer, while device E only has knowledge about the STS layer, and has no knowledge about Ethernet: $Y_n(A) = Y_n(C) = \{\text{Ethernet}\}$; $Y_n(E) = \{\text{STS}\}$; $Y_n(B) = Y_n(D) = Y_n(F) = \{\text{Ethernet}, \text{STS}\}$. We will denote the channels as A_{Eth}, B_{Eth}, B_{STS}, etc. Not all devices support all adaptations: $A_A = A_C A_E = \varnothing$, $C_B = \{(B_{Eth}, B_{\text{STS}}, \mathit{24c})$, $(D_{Eth}, D_{\text{STS}}, \mathit{24c})$, $(D_{Eth}, D_{\text{STS}}, \mathit{3c7v})$, $(F_{Eth}, F_{\text{STS}}, \mathit{3c7v})\}$.

The network has 6 physical links, as shown in figure 7.2 ($L = \{(A_{Eth}, B_{Eth}, 1)$, $(C_{Eth}, F_{Eth}, 1)$, $(B_{\text{STS}}, D_{\text{STS}}, 22)$, $(B_{\text{STS}}, E_{\text{STS}}, 87)$, $(D_{\text{STS}}, E_{\text{STS}}, 38)$, $(E_{\text{STS}}, F_{\text{STS}}, 29)$, $(E_{\text{STS}}, F_{\text{STS}}, 34)\}$).

As we have shown in section 3.3.1, the shortest correct path in this example is $A - B - E - D - B - E - F - C$. This shortest path uses the edge $B_{STS} - E_{STS}$ twice. Consequently, our path finding algorithm will have to take the (de)adaptation functions into account.

As we have stated in chapter 3, path finding in multi-layer networks is a path-constrained problem. One of the consequences is that **a segment of a shortest path does not have to be a shortest path in itself**. For example, a segment of the shortest path between A and C is $D - B - E - F$. However, the shortest path between D and F is $D - E - F$, also if adaptation and channel availability is taken into account.

7.3.2 Device-Based Network Description G_p

We define the graph $G_p = (\mathcal{V}_p, \mathcal{E}_p)$ (p for physical) as follows:

$$\begin{aligned} \mathcal{V}_p &= N \\ \mathcal{E}_p &= \{(n_1, n_2) \mid \big((n_1, y_1), (n_2, y_2), b\big) \in L\} \end{aligned} \tag{7.10}$$

This is a fairly common way to describe the physical properties of a network, with nodes represented as vertices, and (physical) links as edges ($\mathcal{E}_p = L$ for all practical purposes; the formal difference is that L has 3-tuples and is between channels ($c_1 = (n_1, y_1)$ and $c_2 = (n_2, y_2)$), while $\mathcal{E}_p$ has 2-tuples and is between nodes (n_1 and n_2).

If the network is bidirectional, then the graph can be bidirectional. If the network is not fully bidirectional, the graph must be undirected.

Observe that G_p may have multiple edges between the same pair of nodes, like the edge $E - F$ in the example. Some definitions of graphs do not allow this.

The bandwidth $B_e(e)$ of edge e is the bandwidth b of link $(n_1, n_2, b) \in L$

The information on (de)adaptation capabilities is not explicit in the graph defined by G_p, or in another format readable for regular path finding algorithms. Therefore, as we already saw in section 3.3.3, this graph is not very suitable for path finding.

In the next two sections, we present the graphs G_l and G_s which do contain (de)adaptation information.

The graph G_p encodes information on the nodes N and links L. For path finding, we would additionally need information about the channels C, adaptations A and labels LB.

7.3.3 Layer-Based Network Description G_l

Given the set $\mathcal{N}$ of network nodes, and the sets $Y(n)$ of layers for each node n, we construct the graph $G_l = (\mathcal{V}_l, \mathcal{E}_l)$ (l for layer) as follows:

$$\begin{aligned} \mathcal{V}_l &= C \\ \mathcal{E}_l &= \mathcal{E}_{lA} \cup \mathcal{E}_{lD} \cup \mathcal{E}_{lL} \\ \text{with } \mathcal{E}_{lA} &= \{(c_1, c_2) \mid (c_1, c_2, \alpha) \in A\} \text{ (adaptations)} \\ \mathcal{E}_{lD} &= \{(c_2, c_1) \mid (c_1, c_2, \alpha) \in A\} \text{ (de-adaptations)} \\ \text{and } \mathcal{E}_{lL} &= \{(c_1, c_2) \mid (c_1, c_2, b) \in L\} \text{ (links)} \end{aligned} \tag{7.11}$$

The set $\mathcal{V}_l$ consist of all (logical) channels $c = (n, y)$ for all devices $n \in \mathcal{N}$ and for all layers $y \in \mathcal{Y}$ that the node ‘has knowledge of’. The notation we use for vertices $v \in \mathcal{V}$ is n_y. For example, node B in our example network maps to two vertices B_{eth} and B_{STS}

The set $\mathcal{E}_l$ is the union of adaptations $\mathcal{E}_{lA}$, de-adaptations $\mathcal{E}_{lD}$ and (unidirectional) physical links $\mathcal{E}_{lL}$. Only physical links are represented as edges, not derived (logical) links at a higher layer.

For adaptations $(v_{client}, v_{server}) \in A$, the order is important. If $(v_{client}, v_{server}) \in \mathcal{E}_{lA}$ is an adaptation, then (v_{server}, v_{client}) is a de-adaptation rather than an adaptation. Since we want to be able to distinguish between adaptations and de-adaptations, the resulting graph must be directed, even if the network is fully bidirectional.

The adaptation $\mathcal{A}_e(e)$ of edge $e \in mathcalE_{lA}$ is the adaptation function α of $(c_1, c_2, \alpha) \in A$, which is by definition associated with $e \in mathcalE_{lA}$.

The bandwidth $B_e(e)$ of edge $e \in \mathcal{E}_{lL}$ is the bandwidth b of link $(v_1, v_2, b) \in \mathcal{E}_{lL}$. We assume that the adaptations have no bandwidth restrictions, and set $B_e(e)$ of edge $e \in \mathcal{E}_{lA} \cup \mathcal{E}_{lD}$ to ∞.

Note that the functions $\mathcal{A}_e(e)$ and $B_e(e)$ require that each edge has at least two parameters per edge: the bandwidth and the adaptation.

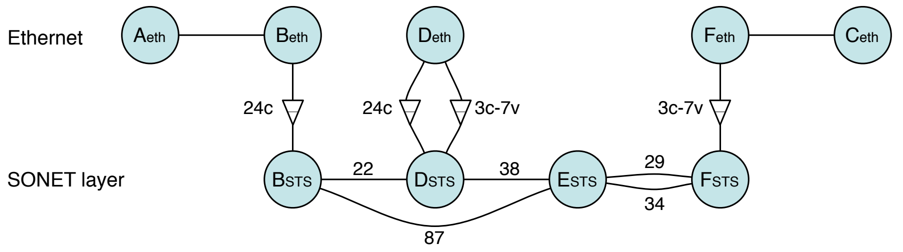


Figure 7.3: *The layer-based representation G_l of graph G in figure 7.2, with adaptation symbols signifying the direction of adaptation edges, rather than using directed edges.*

Figure 7.3 shows the graph G_l for the network described in figure 7.2. The vertices are vertically grouped by layer, and horizontally by node.

For good comparison with the other algorithms, we decided to draw the graph with single edges for each bi-directional network connection, instead of using two directed edges. We signify the direction of the adaptation by the triangles in the edges. This is the standard graphical representation of adaptation in ITU-T G.805 [s42], as we have seen in section 4.3.4.

The number of vertices is (using equation 7.5) $|\mathcal{V}_l| = |C| \leq |N| \times |\mathcal{Y}|$. In our example, $|N| \times |\mathcal{Y}| = 6 \times 2 = 12$, but C is only 9 vertices: nodes A and C are not aware of the STS layer, and node E has no knowledge of Ethernet.

The number of edges $|\mathcal{E}_l|$ is $|A \cup D \cup L'|$. However, since there is no overlap between A, D and L' (an adaptation is never a link or de-adaptation): $|\mathcal{E}_l| = |A| + |D| + |L'|$.

The network in our example is bidirectional, while the graph is directed. So $|L'| = 2|L|$, and $|A| = |D|$, and therefore $|\mathcal{E}_l| = |A| + |D| + |L'| = 2|A| + 2|L|$. For a good comparison with the other graphs G_p and G_s, this compares to $|A| + |L|$ undirected edges, as we can see from figure 7.3.

$$\begin{aligned} |\mathcal{V}_l| &\leq |N| \times |\mathcal{Y}| \\ |\mathcal{E}_l| &= |A| + |L| \qquad \text{for bidirectional edges} \end{aligned} \tag{7.12}$$

In our example network, L consists of 6 physical links, so L' contains 12 directed edges. Furthermore, this network contains four adaptations: nodes B and D support the *24c* adaptation, while nodes D and F support the *3c7v* adaptation. Since each adaptation results in 2 directed edges, this results in another 8 directed edges in $\mathcal{E}_l$.

The graph G_l encodes information on the nodes N, channels C, links L and adaptations A. For path finding, we would additionally need information about the labels LB.

7.3.4 Stack-based network description G_s

The last model, proposed by Kuipers and developed in collaboration with us, explicitly models the possible technology stacks. Our goal is to come to a, in the algorithmic sense, simple network description, which only consists of vertices and edges.

In our example network we can identify three different technology stacks: Ethernet, Ethernet over 24 STS channels, and Ethernet over 21 STS channels:

$$\begin{aligned}\mathcal{S} = \{&[(\text{Ethernet}, \textsc{nil}, \epsilon)],\\ &[(\text{Ethernet}, \textsc{nil}, \epsilon), (\text{STS}, \mathit{24c}, \epsilon)],\\ &[(\text{Ethernet}, \textsc{nil}, \epsilon), (\text{STS}, \mathit{3c7vc}, \epsilon)]\}\end{aligned}$$

Again, NIL means there is no higher layer, and ϵ means the empty label.

We first define the graph $G_s = (\mathcal{V}_s, \mathcal{E}_s)$ (s for stack). The set of vertices $\mathcal{V}_s$ of graph G_s is defined as follows:

$$\begin{aligned}\mathcal{V}_s = &\{(n,s) \mid n \in N \wedge s \in \mathcal{S} \wedge \\ &\big(\exists c \in C_n(n) : Y_s(s) = Y_c(c) \wedge \mathcal{LB}_s(s) \in LB(c)\big)\}\end{aligned} \tag{7.13}$$

Note that:

$$\mathcal{V}_s \subset N \times \mathcal{S} \tag{7.14}$$

Due to the definition of $\mathcal{V}_s$, we can associate one of more channels $C_v(v) \subset C$ with each vertex $v = (n,s) \in \mathcal{V}_s$:

$$C_v(v) = C_v\big((n,s)\big) = \{c \mid c \in C_n(n) \wedge Y_s(s) = Y_c(c) \wedge \mathcal{LB}_s(s) \in LB(c)\} \tag{7.15}$$

The definition in equation 7.13 states that each vertex $v \in V_s$ is defined as a tuple (n,s) with $n \in N$ and $s \in S$. And thus, each vertex $v \in V_s$ has exactly one tuple (n,y) with $n \in N$ and $y = Y_s(s) \in \mathcal{Y}$. If we apply the restriction of equation 7.3, there may be only one channel per (node,layer) tuple, which means that the size of $C_s(v)$ in equation 7.15 is at most one.

Since the definition of vertices $v \in V_s$ requires that there is at least one such channel, we must conclude that, with this restriction, the length $|C_v(v)| = 1$ for every $v \in V_s$.

Using the definition of $C_v(v)$ we can now define the edges $\mathcal{E}_s$ of graph G_s:

$$\begin{aligned}
\mathcal{E}_s =& \mathcal{E}_{sL} \cup \mathcal{E}_{sA} \\
\text{with } \mathcal{E}_{sL} =& \{\big((n_1, s_1), (n_2, s_2)\big) \mid v_1, v_2 = (n_1, s_1), (n_2, s_2) \in \mathcal{V}_s \wedge \\
& \big(\exists (c_1, c_2, b) \in L : c_1 \in C_v(v_1) \wedge c_2 \in C_v(v_2) \wedge \\
& B_a(\mathcal{A}_s(s_1)) \geq b\big)\} \qquad (7.16) \\
\text{and } \mathcal{E}_{sA} =& \{\big((n_1, s_1), (n_2, s_2)\big) \mid v_1, v_2 = (n_1, s_1), (n_2, s_2) \in \mathcal{V}_s \wedge \\
& \big(\exists (c_1, c_2, \alpha) \in A : c_1 \in C_v(v_1) \wedge c_2 \in C_v(v_2) \wedge \\
& \mathcal{A}_s(s_2) = \alpha \wedge \mathcal{LB}_s(s_2) \in LB(c_2)\big)\}
\end{aligned}$$

$\mathcal{E}_{sL}$ is the set of edges representing physical links and $\mathcal{E}_{sA}$ is the set of edges representing adaptations. Both definitions state that an edge exists if a corresponding link or adaptation function exists. In addition, only physical links with enough bandwidth are present, and only adaptations with the correct adaptations function and matching label are present in the graph.

The bandwidth $B_e(e)$ of edge $e \in \mathcal{E}_{sL}$ is the sum of the bandwidths b of the associated links $(c_1, c_2, b) \in L$ (plural, since multiple links may be mapped onto the same edge e). The bandwidth $B_e(e)$ of each edge $e \in \mathcal{E}_{sA}$ is ∞.

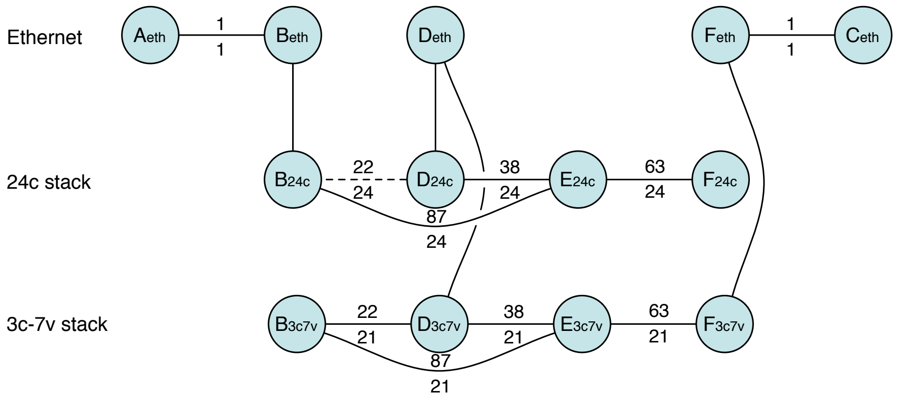


Figure 7.4: *Representation of the network in figure 7.2 as a multi-layered graph.*

Figure 7.4 shows the graph G_s for the network in figure 7.2. The vertices are grouped according in a matrix with each node n in a column, and a row for each of the three technology stacks (Ethernet, Ethernet over 24 STS channels, and Ethernet over 21 STS channels). Such grouping has also been deployed in the context of wavelength routing in WDM networks [p8].

Node B can only adapt Ethernet in 24 channels (and de-adapt back), while node F can only adapt Ethernet into 21 channels. Thus, there is an edge between B_{eth} and B_{24c}, but not between B_{eth} and B_{3c7v}. This is a direct result of the condition $\mathcal{A}_s(s_2) = \alpha$ in the definition of $\mathcal{E}_{sA}$.

When comparing this graph with figure 7.3, it is clear that the row for each stack s in figure 7.4 corresponds to the row for the associated layer $Y_s(s)$ in figure 7.3. The only difference is that in G_s, there is only one edge per pair of vertices. This is because the definition for $\mathcal{E}_{sL}$, which states that an edge exists if a corresponding link exists, but not that there must be an edge for *every* corresponding link. This is an optimisation to reduce the size of G_s for path finding. While a shortest path may contain two links between two nodes can be used twice, it will never be with the same encoding ('loops' in a shortest path are only present if a remote host performs an unavoidable conversion between two encodings).

The dotted line between B_{24c} and D_{24c} states that in theory these nodes should be able to communicate with each other, but in this case not enough $(22 < 24)$ channels are available, and due to the link constraint $B_a(\mathcal{A}_s(s_1)) \geq b$ in the definition of $\mathcal{E}_{sL}$, the actual edge is disregarded in the graph G_s.

The number above edges in represent the available bandwidth for the associated links. The number beneath each edge represent the required bandwidth as determined by the adaptation function. In this case of multiple links between two nodes, the edge weight is the sum of all link capacities.

The set of technology stack $\mathcal{S}$ is *not a local property* of the nodes, but depends on the choice of the (root technology of the) end-nodes. For example, node E has only knowledge of the STS layer, and for path finding between D and F, the only encoding would be $[(\text{STS}, \textsc{nil}, \epsilon)]$. However, for path finding between A and C, the possible encodings are $[(\text{Ethernet}, \textsc{nil}, \epsilon), (\text{STS}, \mathit{24c}, \epsilon)]$ and $[(\text{Ethernet}, \textsc{nil}, \epsilon), (\text{STS}, \mathit{3c7vc}, \epsilon)]$. Because $\mathcal{S}$ depends on the choice of the root technology, the graph G_s is different if the layer of the end-nodes is different.

In order to give an estimate of the number of vertices and edges in G_s, we first prove that there is only one channel associated with each vertex $v \in V_s$, and then proceed to give an estimate of the size $|\mathcal{S}|$ of $\mathcal{S}$. We need these numbers to estimate the running time of the algorithms.

The number of possible stacks $|\mathcal{S}|$ highly depends on how the technology diagram looks like. We define $\mathcal{S}_y \subset \mathcal{S}$ to be the set of all stacks with lowest layer y:

$$\mathcal{S}_y = \{s \mid s \in \mathcal{S} \wedge Y_s(s) = y\} \tag{7.17}$$

Furthermore, we define the set of client layers for a given (server) layer y_s as:

$$Y_{client}(y_s) = \{y_c \mid y_c \in \mathcal{Y} \wedge \exists (y_c, y_s) \in \mathcal{A}(y_s)\} \tag{7.18}$$

Given the recursive definition of stacks in equation 7.9b, we can see that a stack s_s with lowest layer $Y_s(s)$ can only be formed by appending an adaptation and label to an existing stack s_c, with $Y_s(s_c) \in Y_{client}(Y_s(s_c))$ (for all non-root stacks):

$$\begin{aligned}
\mathcal{S}_y &= \{s \mid s \in \mathcal{S} \wedge Y_s(s) = y\} \\
&= \{s_c + [(y, (y_c, y), lb)] \mid (y, (y_c, y), lb) \in \mathcal{T} \wedge s_c \in \mathcal{S} \wedge y_c = Y_s(s)\} \\
&= \{s_c + [(y, (y_c, y), lb)] \mid (y, (y_c, y), lb) \in \mathcal{T} \wedge s_c \in \mathcal{S}_{y_c}\} \\
&= \{s_c + [(y, (y_c, y), lb)] \mid (y, (y_c, y), lb) \in \mathcal{T}(y) \wedge s_c \in \mathcal{S}_{y_c}\} \\
&= \{s_c + [(y, (y_c, y), lb)] \mid lb \in \mathcal{LB}(y) \wedge (y_c, y) \in \mathcal{A}(y) \wedge s_c \in \mathcal{S}_{y_c}\} \\
&= \{s_c + [(y, \alpha, lb)] \mid lb \in \mathcal{LB}(y) \wedge \alpha \in \mathcal{A}(y) \wedge y_c \in Y_{client}(y) \wedge s_c \in \mathcal{S}_{y_c}\}
\end{aligned} \tag{7.19}$$

Using equations 7.19 and 7.8, we can express the size of $\mathcal{S}_y$, in the size of its client stacks:

$$|\mathcal{S}_y| = \sum_{y_c \in Y_{client}(y)} |\mathcal{S}_{y_c}| \times |\mathcal{LB}(y)| \times |\mathcal{A}(y)| = \sum_{y_c \in Y_{client}(y)} |\mathcal{S}_{y_c}| \times |\mathcal{T}(y)| \tag{7.20}$$

In order to make further estimates of $|\mathcal{S}|$, we will now assume that the protocol stack is an ordered set of layers, ordered from 'highest' to 'lowest' layer, without any branches. There must be at least one adaptation between consecutive layers in the list, and there may be only one root technology (thus the root layer has only one label). We denote this list of layers $\mathcal{Y} = [y_1, y_2, \ldots, y_Y]$ (with $Y = |\mathcal{Y}|$), and denote $\mathcal{Y}_i$ the list of only the first $1 \geq i \geq |\mathcal{Y}|$ layers: $[y_1, y_2, \ldots, y_j]$. With this constraint, each (non-root) layer has only one client layer: $Y_{client}(y_i) = \{y_{i-1}\}$, and thus equation 7.20 reduces to

$$|\mathcal{S}_{y_i}| = |\mathcal{S}_{y_{i-1}}| \times |\mathcal{T}(y_i)| \tag{7.21}$$

Now we can easily calculate the number of different technology stack for layer y_j:

$$\begin{aligned} |\mathcal{S}_{y_i}| &= |\mathcal{S}_{y_{i-1}}| \times |T(y_i)| \\ &= |\mathcal{S}_{y_{i-2}}| \times |T(y_{i-1})| \times |T(y_i)| = \dots \\ &= |\mathcal{S}_{y_1}| \times \prod_{1 \le j \le i} |T(y_j)| \\ &= \prod_{1 \le j \le i} |T(y_j)| \end{aligned} \tag{7.22}$$

The last step uses the fact that $|\mathcal{S}_{y_1}| = 1$, since there is only one technology at the highest layer.

Since both $|\mathcal{A}(y_i)| \geq 1$ and $|\mathcal{LB}(y_i)| \geq 1$, also $|T(y_i)| \geq 1$:

$$|\mathcal{S}_{y_i}| = |\mathcal{S}_{y_{i-1}}| \times |T(y_i)| \geq |\mathcal{S}_{y_{i-1}}| \tag{7.23}$$

The number of all technology stacks for an ordered list of layers is:

$$|\mathcal{S}| = \sum_{y_i \in \mathcal{Y}} |\mathcal{S}_{y_i}| = \sum_{y_i \in \mathcal{Y}} \prod_{1 \le j \le i} |T(y_j)| \tag{7.24}$$

Equation 7.23 gives an upper limit for each $|\mathcal{S}_{y_i}|$, and thus an upper limit on $|\mathcal{S}|$

$$|\mathcal{S}| = \sum_{y_i \in \mathcal{Y}} \prod_{1 \le j \le i} |T(y_j)| \leq |\mathcal{Y}| \times \prod_{y \in \mathcal{Y}} |T(y)| \tag{7.25}$$

The estimate for $|\mathcal{S}|$ is considerable lower, as $|\mathcal{S}| \approx |\mathcal{S}_{y_Y}|$, provided that $|T(y_Y)| >> 1$, with $y_Y = y_{|\mathcal{Y}|}$ the 'lowest' network layer.

$$|\mathcal{S}_{y_i}| \leq |\mathcal{S}_{y_Y}| \text{ for all } y_i \in \mathcal{Y} \tag{7.26}$$

$$|\mathcal{S}| \gtrapprox |\mathcal{S}_{y_Y}| = \prod_{1 \le j \le |\mathcal{Y}|} |T(y_j)| = \prod_{y \in \mathcal{Y}} |T(y)| \tag{7.27}$$

The upper limit of the number of vertices $|\mathcal{V}_s|$ (using equation 7.14) is:

$$|\mathcal{V}_s| \leq |N| \times |\mathcal{S}| \tag{7.28}$$

If most of the nodes can carry all technology stacks (i.e. $\forall n \in N : |C_n(n)| \approx |\mathcal{Y}|$), this upper limited becomes an estimate: $|\mathcal{V}_s| \lessapprox |N| \times |\mathcal{S}|$. The lower estimate of equation 7.27 and upper estimate of equation 7.28 roughly cancel each other out:

$$|\mathcal{V}_s| \approx |N| \times |\mathcal{S}| \approx |N| \times \prod_{y \in \mathcal{Y}} |T(y)| \tag{7.29}$$

Equation 7.29 results in an estimate of $|\mathcal{V}_s| \approx |N| \times \prod_{y\in\mathcal{Y}} |T(y)| = |N| \times |T(\text{Ethernet})| \times |T(\text{STS})| = 6 = 12$ vertices, while G_s has in fact 13 vertices.

The number of edges in G_s is $|\mathcal{E}_s| = |\mathcal{E}_{sL}| + |\mathcal{E}_{sA}|$. Physical links $l = (c_1, c_2, b) \in L$ map to zero or more edges $(v_1, v_2) \in \mathcal{E}_{sL}$. We define the layer of link $(c_1, c_2, b) \in L$ as $Y_l\big((c_1, c_2, b)\big) = Y_c(c_1) = Y_c(c_2)$ (links must have terminating channels at the same layer).

Given a channel $(n, y) \in C$, there are exactly $|\mathcal{S}_y|$ stacks s with $Y_s(s) = y$, by definition of equation 7.17. By definition of $C_n(n)$ (equation 7.1), the channel $(n, y) \in C_n(n)$. Also, for every tuple (n, s) with $s \in \mathcal{S}_y$, $Y_s(s) = y$. Therefore, by definition of $\mathcal{V}_s$ (equation 7.13), every tuple (n, s) with $s \in \mathcal{S}_y$ is a valid vertex with associated channel $(n, y) \in C_n(n)$ if the label of stack s is a valid label of channel c. Thus, for every channel $c = (n, y) \in C$, there are at most $|\mathcal{S}_y|$ vertices in $\mathcal{V}_s$:

$$\forall c \in C: \ |\{v \mid v \in \mathcal{V}_s \wedge c \in C_v(v)\}| \leq |\mathcal{S}_{Y_c(c)}| \tag{7.30}$$

Consequently, for every link $(c_1, c_2, b) \in L$ there are also at most $|\mathcal{S}_y|$ edges in $\mathcal{E}_{sL}$, with $y = Y_l\big((c_1, c_2)\big)$. Given that two physical links between the same pair of nodes result in the same set of edges, we must only count unique links:

$$|\mathcal{E}_{sL}| \leq \sum_{\text{unique link } l\in L} |\mathcal{S}_{Y_l(l)}| \leq \sum_{l\in L} |\mathcal{S}_{Y_l(l)}| \tag{7.31}$$

Given that we require that an adaptation exists between all subsequent layers, the set $\mathcal{S}_y$ contains at least one stack for every layer. This gives us the lower limit $|\mathcal{E}_{sL}| \geq |\text{unique links in } L|$.

If all physical links are at the lowest layer y_Y, and there are no links between the same channels (since that would result in an equal set of edges), then the number of edges $|\mathcal{E}_{sL}|$ is equal to the upper limit. This is a reasonable estimate:

$$|\mathcal{E}_{sL}| \lessapprox |L| \times |\mathcal{S}_{y_Y}| = |L| \times \prod_{y\in\mathcal{Y}} |T(y)| \tag{7.32}$$

The estimate for $|\mathcal{E}_{sA}|$ is similar to that of $|\mathcal{E}_{sL}|$, although the exact quantification is more complex due to the added label restriction (not every vertex pair as defined in equation 7.30 should have an adaptation). Nevertheless, each adaptation $a \in A$ results in 1 to $|\mathcal{S}_{y_Y}|$ edges in $\mathcal{E}_{sA}$:

$$|A| \leq |\mathcal{E}_{sA}| \leq |A| \times |\mathcal{S}_{y_Y|}| = |A| \times \prod_{y\in\mathcal{Y}} |T(y)| \tag{7.33}$$

This gives us an estimate of $|\mathcal{E}_s|$:

$$|A| + |\text{unique links in } L| \leq |\mathcal{E}_s| \leq (|A| + |L|) \times \prod_{y \in \mathcal{Y}} |T(y)| \tag{7.34}$$

Equation 7.34 results in an lower limit of $|\mathcal{E}_s| \geq |A| + |\text{unique links in } L| = 4 + 5 = 9$ edges, and a high estimate of $|\mathcal{E}_s| \leq (|A| + |L|) \times \prod_{y \in \mathcal{Y}} |T(y)| = (|A| + |L|) \times |T(\text{Ethernet})| \times |T(\text{STS})| = (4 + 6) \times 1 \times 2 = 20$ edges. In reality, G_s has 13 edges after removal of one edge due to bandwidth constraints (4 edges representing adaptations, and 9 representing links).

The graph G_s encodes information on the nodes N, channels C, links L, adaptations A and labels LB. In addition, edge labels can be used for available and required bandwidths. For multi-layer path finding, this is all the information we need.

7.4 Path Selection in G_l

Given a layer-based graph $G_l = (\mathcal{V}_l, \mathcal{E}_l)$ and a source vertex v_{src} and a destination vertex v_{dst} in $\mathcal{V}_l$, our objective is to find the path P^* for which $W(P^*) \leq W(P)$ for all feasible paths P between v_{src} and v_{dst}. As discussed in the previous section 7.3.3, we may pass through vertices and even edges multiple times, and since G_l maps links 1:1 to edges, we can not simply use a link-constrained algorithm (see our claim in section 3.3.2).

Algorithm 7.1 presents MULTI-LAYER-BREADTH-FIRST(G_l, v_{src}, v_{dst}) to compute the shortest path P^* from v_{src} to v_{dst} in G_l. We first explain the general properties of the algorithm, than explain it line by line.

This algorithm has two main features: (1) we keep track of the stack of (de)adaptations along the path and (2) multiple paths may be stored at a single vertex (similarly to a k-shortest paths algorithm). So instead of working with a queue of vertices, as in the breadth-first-search algorithm and Dijkstra's algorithm [p9, p11], the algorithm keeps a queue of paths. Basically, this algorithm simply tries all possible paths P ordered by length, even those with loops or already visited vertices.

The base of the algorithm is a queue Q of path objects. The algorithm stores the following properties for each path object $p \in Q$:

1. Sequence of edges $\mathcal{E}[p]$;
2. The last visited vertex $V_p[p]$
3. Current technology stack $S_p[p]$;

Algorithm 7.1 MULTI-LAYER-BREADTH-FIRST(G_l, v_{src}, v_{dst})

```
1: $p \leftarrow$ new path with $\mathcal{E}[p] = \varnothing; V_p[p] = v_{src}; S_p[p] = \varnothing; B[p] = 1; R[p] =$
   $\varnothing; B[p,e] = \varnothing; d[p] = 0$
2: ENQUEUE$(Q, p)$ {Queue $Q$ consists of paths}
3: while $Q \neq \varnothing$ do
4:   $p \leftarrow$ DEQUEUE$(Q)$ {Replace with EXTRACT-MIN for the shortest path}
5:   $v \leftarrow V_p[p]$ {The last vertex in the path}
6:   if $v = v_{dst}$ and $\mathcal{A}_s(S_p[p]) = \varnothing$ then
7:     return $p$ {Reached destination}
8:   else
9:     for all $v \in$ adj$[u]$ do
10:      $p_{new} \leftarrow$ EXTEND-PATH$(p, (u,v), v)$
11:      if $p_{new} \neq$ unfeasible then
12:        ENQUEUE$(Q, p_{new})$
```

4. The available labels $LB(S_p[p])$ (this extension is discussed in section 7.6);
5. The list of used bandwidths $B[p,e]$ for every edge $e \in \mathcal{E}[p]$;
6. The set of visited (vertex, stack) tuples $R[p]$;
7. Distance $d[p]$.

Only the $\mathcal{E}[p]$ property is required, and all other properties can be deduced from $\mathcal{E}[p]$ (except $V_p[p]$ if $\mathcal{E}[p] = \varnothing$), but they are present for speed.

Lines 1-2 initialise the algorithm with a starting path starting at vertex v_{src}, and without an adaptation in the technology stack. Lines 3-12 form the main loop. It takes the path with the shortest length from the queue, and extends it by one hop in all directions. Lines 6-7 checks if the shortest path ends at the destination, and if so, returns that path as the result of the algorithm. Not all extensions of a path with one hop will result in a feasible path. Line 11 checks for this condition, and only considers feasible paths.

The actual extension of the path and the feasibility check is done in the Extend-Path routine in algorithm 7.2. This algorithm takes the existing path p and extends it via edge e to vertex v. It sets the six properties of the path accordingly. In case of an adaptation (lines 5-6), the new adaptation is added to the stack. In case of de-adaptation, the last adaptation is removed from the stack (lines 7-10), but only if the de-adaptation is equal to the last adaptation on the stack. If it is not, it is an unfeasible path. Lines 14-16 are an extension to the base algorithm to check if two adjacent vertices have a common channel label available for transmission of data.Lines 21-24 check if the vertex has been

Algorithm 7.2 EXTEND-PATH(p, e, v)

```
Require: $p$ is a path, $n$ an adjacent vertex, $e$ the connecting edge
Require: The sets of edges $\mathcal{E}_{lL}$, $\mathcal{E}_{lA}$, and $\mathcal{E}_{lD}$
Require: Adaptation $\mathcal{A}_e$ for each edge $e \in \mathcal{E}_{lA} \cup \mathcal{E}_{lD}$
Require: The weight of all edges $W_e(e)$
Require: Set of possible labels per vertex $LB(v)$
 1: $p_{new} \leftarrow p$
 2: $\mathcal{E}[p_{new}] \leftarrow \mathcal{E}[p] + e$ {Extend $p_{new}$ with edge $e$}
 3: $V_p[p_{new}] \leftarrow v$
 4: $d[p_{new}] \leftarrow d[p] + W_e(e)$ {Length of path increases. If $W_e$ is always 1, the
    queue $Q$ remains sorted}
 5: if $e \in \mathcal{E}_{lA}$ then {Adaptation}
 6:     $S_p(p_{new}) \leftarrow S_p[p] + t$ with $t = (y, \alpha, Lb)$; $y = Y_c(v)$; $\alpha = \mathcal{A}_e(e)$; $Lb \in$
        $LB(v)$ {The choice of $Lb$ is ambiguous if $|LB(v)| > 1$}
 7: else if $e \in \mathcal{E}_{lD}$ then {De-adaptation}
 8:     if $\mathcal{A}_s(S_p(p_{new})) \neq \mathcal{A}_e(e)$ then {Wrong de-adaptation}
 9:         return unfeasible
10:     else
11:         POP-ELEMENT$(S_p(p_{new}))$ {Remove last adaptation from the stack}
12: else {$e \in \mathcal{E}_{lL}$, Link}
13:     $S_p(p_{new}) \leftarrow S_p[p]$
14:     $LB(S_p(p_{new})) \leftarrow LB(S_p(p_{new})) \cap LB(v)$
15:     if $LB(S_p(p_{new})) = \varnothing$ then {No compatible labels left}
16:         return unfeasible
17: $b \leftarrow$ BANDWIDTH-REQUIRED$(S_p(p_{new}))$ {Determined by adaptation}
18: $B[p_{new}, e] \leftarrow B[p, e] - b$
19: if $B[p_{new}, e] < 0$ then {No bandwidth available}
20:     return unfeasible
21: if $(\mathcal{V}_p(p_{new}), S_p(_{new})) \in R[p]$ then {Node $\mathcal{V}_p(p_{new})$ has been visited before
    with the same stack}
22:     return unfeasible
23: else
24:     $R[p_{new}] \leftarrow R[p] \bigcup (\mathcal{V}_p(p_{new}), S_p(p_{new}))$
25: return $p_{new}$
```

Algorithm 7.3 Bandwidth-Required(s)

```
Require: $s$ is an adaptation stack
1: if $\mathcal{A}_s(s) \neq$ NIL then
2:     $(y_c, y_s, b_s) \leftarrow \mathcal{A}_s(s)$ {$b_s$ determined by the adaptation function}
3:     $b \leftarrow b_s$
4: else
5:     $b \leftarrow 1$
6: return $b$
```

covered earlier in the current path with the same stack, and if so, skip this extension. This reduces the flooding nature of this algorithm.

The running time of a breadth first search algorithm is $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ [p9]. However, this assumes that each vertex is processed only once. The running time for this algorithm is considerably longer since each vertex can be covered multiple times. In fact, in the worst-case scenario, the length of the queue Q can grow exponentially with the average out-degree of the vertices.

Kuipers showed that the multi-layer path finding problem is NP-complete [a12]. In contrast, path finding in a single layer network can be solved in polynomial time (e.g. using Dijkstra's algorithm or a breadth first search algorithm). This is in accordance with an exponential running time.

The running time of algorithm 7.1 is (see appendix A.4):

$$\begin{aligned}\mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|Q|) \times \mathcal{O}(\text{loop})) = \\ &= \mathcal{O}(|Q|) \times (\mathcal{O}(\text{DEQUEUE}) + \mathcal{O}(|adj|) \times \mathcal{O}(\text{Extend-Path}))\end{aligned} \tag{7.35}$$

With $\mathcal{O}(\text{DEQUEUE})$ caused by line 4, $\mathcal{O}(|adj|)$ by line 9, and $\mathcal{O}(\text{Extend-Path})$ by line 10.

As we have seen in section 7.3.1, a segment of a shortest path may not be a shortest path in itself. In order to limit the flooding nature of this algorithm even further, it is possible to make this assumption. This makes the algorithm faster at the cost that it sometimes will return a false negative (no path is found, even though it exists). A proper replacement of this check would verify if the vertex with that stack is already present in any path, instead of in path p only. This check be accomplished by making $R[p]$ a global variable, rather than tied to a particular path p.

If we would restrict the search space of the algorithm using the above modification, the running time would reduce to (see equation A.15)

$$\begin{aligned}\mathcal{O}(\text{Algorithm 7.1}) = {} & \mathcal{O}(|N|) \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\textsc{dequeue}) + \\ & \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\textsc{Extend-Path}) \\ \approx {} & \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \\ & \mathcal{O}\big(\big(\log(|N|) + \log(|\mathcal{Y}|) + |\mathcal{Y}| \cdot \log(T)\big) + \log(\langle lb \rangle)\big)\end{aligned} \tag{7.36}$$

The estimated average running time is (see equation A.16):

$$\mathcal{O}(\text{Algorithm 7.1}) \approx \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|} \times \big(\log(|N| \times T^{|\mathcal{Y}|}) + \log(\langle lb \rangle)\big)) \tag{7.37}$$

7.5 Path Selection in G_s

If we would apply the Dijkstra algorithm [p11] to the graph G_s in figure 7.4, we would find the path $A_{Eth} - B_{Eth} - B_{24c} - E_{24c} - D_{24c} - D_{Eth} - D_{3c7v} - E_{3c7v} - F_{3c7v} - F_{Eth} - C_{Eth}$, which relates to path $A - B - E - D - E - F - C$. If the link between D and E would have enough capacity, this would result in a feasible path, which is not the case here. The correct path is $A - B - E - D - B - E - F - C$. Consequently (contrary to the approach in [p8]), we have to modify our algorithm to account for capacity on links traversed multiple times.

Our goal is to come up with a simple path finding algorithm. The graph G_s seemed a prime candidate for this purpose, given that a shortest path in the graph G_s will never contain the same vertex twice. This can easily be proven. Assume a path $P_{src-dest}$ from *src* to *dest* contains the vertex v twice. Then we can split path $P_{src-dest}$ in three segments, P_{src-v} from *src* to v, P_{v-v} from v to v, and P_{v-dest} from v to *dest*. Given that the path length d is additive, then $d(P_{src-dest}) = d(P_{src-v}) + d(P_{v-v}) + d(P_{v-dest})$. The path $P'_{src-dest} = P_{src-v} + P_{v-dest}$ has then length $d(P'_{src-dest}) = d(P_{src-v}) + d(P_{v-dest}) \leq d(P_{src-dest})$ (provided that $d(P_{v-v}) \geq 0$). $P'_{src-dest}$ is a valid path, according to the definition in section 4.4.5 if the technology vertex v is the same each encounter. This is true for a vertex $v = (n, s) \in G_s$, as the technology is uniquely defined by the adaptations stack s. This may not be true for vertices in G_p or in G_l as the adaptation stack is not uniquely defined for each vertex.

Given that a shortest path in the graph G_s will never contain the same vertex twice, our first approach was to create a variant of the Dijkstra algorithm (see appendix A). However, it turned out that such variant implicitly assumes that a segment of a shortest path is also a shortest path, while we saw in section 7.3.1 that this assumption is not true for multilayer networks.

Algorithm 7.4 MULTI-LAYER-K-SHORTEST-PATH(G_s, v_{src}, v_{dst})

```
 1: $r[v] \leftarrow$ DIJKSTRA($G_s, v_{dst}$) {Lower bounds for all nodes}
 2: for all vertices $v \in \mathcal{V}_s$ do
 3:     $counter[v] \leftarrow 0$
 4: $maxlength \leftarrow \infty$
 5: $counter[v_{src}] \leftarrow counter[v_{src}] + 1$
 6: $\pi[v, counter[v_{src}]] \leftarrow$ NIL, NIL {Source $v_{src}$ has no predecessor vertex and
    index}
 7: ENQUEUE($Q, v_{src}, counter[v_{src}], 0, r[v_{src}]$) {Queue $Q$ consists of vertex,
    path index, path length so far, and lower bound of the total path length}
 8: while $Q \neq \varnothing$ do
 9:     $u, i, d \leftarrow$ EXTRACT-MIN($Q$){Extract path with lowest lower bound of
        the total path length}
10:     if $u = v_{dst}$ then
11:         return path {Created by backtracing $\pi[v, i]$, starting with $v_{dst}, i$}
12:     else
13:         for all $v \in \text{adj}[u]$ do {for each neighbour of $u$}
14:             if $d + w(u, v) + r[v] < maxlength$ then {Skip paths that exceed
                known maximum length}
15:                 if FEASIBILE($G_s, u, i, v$) then {Backtracking}
16:                     $d' \leftarrow d + w(u, v)$ {Distance is sum of weights}
17:                     $counter[v] \leftarrow counter[v] + 1$
18:                     $\pi[v, counter[v]] \leftarrow u, i$
19:                     ENQUEUE($Q, v, counter[v], d', d' + r[v]$)
20:                     if $v = v_{dst}$ and $d' + r[v] < maxlength$ then
21:                         $maxlength \leftarrow d' + r[v]$
```

The algorithm MULTI-LAYER-K-SHORTEST-PATH(G_s, v_{src}, v_{dst}), which is presented in listing 7.4 is a k-shortest path algorithm, and does not make the assumption that a segment of a shortest path is also a shortest path. It employs brute force to always return an exact answer, just as the algorithm for path finding in G_l which is discussed in the previous section.

Our objective is to find a shortest path from v_{src} to v_{dst} in G_s. The MULTI-

Algorithm 7.5 FEASIBLE(G_s, u, i, v)

```
Require: Available bandwidth $B_e(e)$ for the edge $e = (u, v)$
Require: Required bandwidth for the vertex $v = (n, s)$, based on its stack $s$
Ensure: {Backtracking to check whether the path is loop-free and has enough
    capacity available. $u, i$ is our current node, $v$ is the node under considera-
    tion for extending our path with.}
 1: $t \leftarrow u$
 2: $j \leftarrow i$
 3: $B' \leftarrow B_e\big((u, v)\big)$
 4: while $\pi[t, j] \neq$ NIL,NIL do
 5:    $t', j' \leftarrow \pi[t, j]$ {Previous hop}
 6:    if $C_v(t') = C_v(u) \wedge C_v(t) = C_v(v)$ then {$(u, v)$ and $(t', t)$ are the same
       link}
 7:       $(n, s) \leftarrow t$ {Stack $s$ determined by the vertex $t$}
 8:       $B' \leftarrow B' -$ BANDWIDTH-REQUIRED($s$) {See algorithm 7.3}
 9:    $t, j \leftarrow \pi[t, j]$ {Trace back}
10:    if $t = v$ then
11:       return FALSE {loop}
12: $(n, s) \leftarrow v$ {Stack $s$ determined by the vertex $v$}
13: if $B' <$ BANDWIDTH-REQUIRED($s$) then
14:    return FALSE {no bandwidth available}
15: else
16:    return TRUE
```

LAYER-K-SHORTEST-PATH algorithm is based on SAMCRA [p19, p26], and the Multi-Layer-Breadth-First algorithm in listing 7.1. The two main variables are a queue Q of paths, and a list of previous hops, stored in matrix π. Multiple paths can be stored at a vertex u, u, i is used to denote the i-th path at node u. For example $\pi[E_{3c7v}, 1]$ may refer to $D_{3c7v}, 1$ while $\pi[E_{3c7v}, 2]$ may refer to $B_{3c7v}, 1$. By backtracking the predecessor list $\vec{\pi}$, an entire path can be reconstructed, starting at the last hop. *counter*$[v]$ refers to the number of paths stored at node v.

The queue Q keeps track of a path using the tuple (u, i), and in addition keeps track of the length of the path. A feature taken from SAMCRA is to use two search-space reduction techniques. First, the *maxlength* variable keeps track of the maximum length of the end-to-end path. Second, the Dijkstra algorithm is used to give a lower bound of the path length. While both techniques increase the running time in the worst-case scenario, they significantly optimise the order in which all possibilities are searched, eliminating possibilities that will never lead to a shortest path.

Lines 1-7 of the meta-code initialises all nodes. Line 1 computes the shortest paths from v_{dst} to all other nodes in the graph by one execution of the Dijkstra shortest paths algorithm that disregards any bandwidth constraints. The weights of these paths serve as lower bound estimates, referred to as $r[v]$ for all nodes $v \in \mathcal{V}_s$. Note that if the shortest path from v_{src} to v_{dst} is feasible, we can stop the algorithm and return this solution. Else the algorithm should proceed. Line 7 inserts the source node with predicted length $d[v_{src}] + r[v_{src}] = r[v_{src}]$ and predecessor NIL in the queue Q, which was initially empty. The main algorithm starts at line 8. Line 9 extracts the node u and index i from the queue that has the predicted length. Hop u, i can be regarded as the new scanning node towards destination v_{dst}. If $u = t$ we have found the shortest feasible path, else we continue. Lines 13 to 19 perform the relaxation procedure [p9] for each adjacent node v of u. Line 14 checks whether the length of the path extended from u to v does not exceed *maxlength*, otherwise it is discarded because we already have a better candidate. If this first test is passed, the module FEASIBLE is called to check for loops (each vertex in G_s should be only used once in a shortest path) and for enough available bandwidth on the link (u, v). If the result is a feasible path, we insert v into the queue.

The FEASIBLE algorithm is described in listing 7.5. $B_e(u, v)$ contains the available bandwidth on the edge (u, v). $b(u, v)$ gives the remaining available bandwidth on the edge (u, v), which is initialised as $B_e(e)$ in line 3 and decreases in the course of the algorithm (lines 6-8). Lines 4-11 form loop through all edges in the path, by backtracking $\vec{\pi}$ (line 9). If the path contains the same vertex twice (line 10-11) or not enough bandwidth is left on edge (u, v) (line 13-14), the path is deemed unfeasible.

7.6 Extension to Multiple Labels

So far, we mostly looked at two layers, with the adaptations between the two layers as the only technological incompatibility. All algorithms can easily be extended to support multiple layers, and for the most part we already put those extensions in the definitions. In this section we will illustrate this with the example network in figure 7.5.

The example network of figure 7.5 consists of 7 nodes, $N = \{A, B, C, D, E, F, G\}$, 3 layers $\mathcal{Y} = \{\text{Ethernet, SONET, WDM}\}$, with two incompatibilities at the SONET layer, the previously mentioned STS-24c and STS-3c-7v adaptations, $\mathcal{A}(STS) = \{\text{3c7v, 24c}\}$ and two incompatible wavelengths at the Wavelength

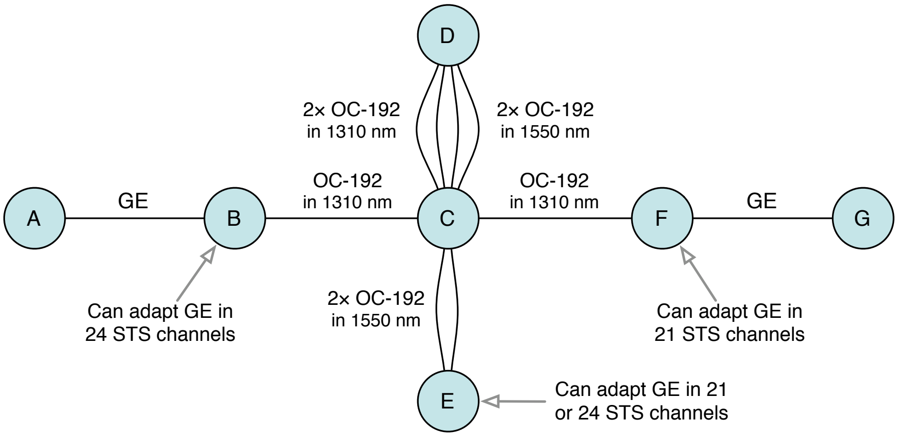


Figure 7.5: *Example of a three-layer network..*

Division Multiplexing (WDM) layer, $\mathcal{LB}(WDM) = \{1310\text{nm}, 1550\text{nm}\}$. Nodes A and G are pure Ethernet devices, nodes B, D, E and F are SONET devices, with B, D and F capable of (de)adapting Ethernet in SONET, and C is an optical cross connect. Nodes B and F are equipped with 1310 nm lasers, node E is equipped with 1550 nm lasers, and node D has tuneable lasers.

The shortest working path from A to G is $A - B - C - D - C - E - C - D - C - F - G$. B and F can not directly communicate due to the different adaptation of Ethernet in SONET, and B and E and also E and F can not communicate because of the different wavelengths.

7.6.1 Extension to Graph G_l

Graph G_l of this network is given in figure 7.6. The shortest path through this graph is $A_{Eth} - B_{Eth} - B_{STS}(24c) - B_{WDM}(1310) - C_{WDM}(1310) - D_{WDM} - DSTS - D_{WDM}(1550) - C_{WDM}(1550) - E_{WDM} - E_{STS}(24c^{-1}) - E_{Eth}(3c - 7v) - E_{STS} - E_{WDM}(1550) - C_{WDM}(1550) - D_{WDM} - D_{STS} - D_{WDM}(1310) - C_{WDM}(1310) - F_{WDM} - F_{STS}(3c - 7v^{-1}) - D_{Eth} - G_{Eth}$, with the label as used on the following edge denoted between brackets. A few new characteristics of G_l emerge. There is only a single adaptation between D_{STS} and D_{WDM}, even though that edge is used four times in the shortest path. In G_l edges can be traversed multiple times, as long as the available bandwidth is not exceeded. On the other hand, there are 4 edges between D_{WDM} and C_{WDM}, since the actual network has four different physical links.

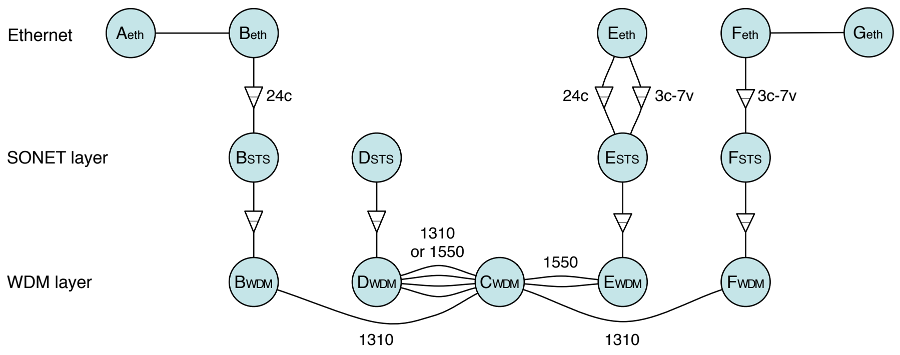


Figure 7.6: *Graph G_l of the network of figure 7.5.*

G_l treats the various incompatibilities differently. In section 6.3.1 we distinguish between four distinct incompatibilities: (1) in layer, (2) in adaptation, (3) in label (channel identifier), and (4) in other encoding characteristics (e.g., different MTU size for Ethernet). The graph G_l represents different adaptations as different edges, while different labels are represented as different labels or label sets for the edges. See the algorithm 7.2 in section 7.4: the incompatibility check for adaptation (line 15) is different from the incompatibility check for labels (line 22). This is a conscious choice: usually there are only a few (if any) incompatible adaptation functions, but many incompatible labels. In addition, G_l allows for *sometimes* incompatible labels. For example, one node may not be able to convert wavelengths, while another node may convert between wavelengths without de-adapting and adapting the wavelength. This is also referred to as *label swapping*. To support this, line 18 in the Extend-Path algorithm needs to be changed to an if statement:

```
if vertex $v$ supports label swapping then
    $LB(S_p(p_{new})) \leftarrow LB(v)$
else
    $LB(S_p(p_{new})) \leftarrow LB(S_p(p_{new})) \cap LB(v)$
```

Furthermore, the list of visited channels, $R[p]$, needs to be updated to include the labels, not just the node and layer.

The estimate of the running time algorithm as we presented in section 7.4 is not correct if we distinguish between labels and adaptations. In particular line 13 of algorithm takes *one* label $LB \in LB(v)$. This is ambiguous if $|LB(v)| > 1$. One solution to solve this is to simply return a set of path extensions, one for each label, rather than one label. In that case, the number of different stacks is approximately T^Y, and the estimate for the running time is correct. However, it is more efficiently to instead keep a set of allowed labels rather then a single label in the technology stacks. Thus each entry LB in $(y, \alpha, LB) \in \mathcal{S}$ would no longer be a single label, but a set of labels. In fact, the extension on lines 21-23 already does this: it reduces the set of allowed labels (or increases for devices with swapping capability), instead of terminating paths with incompatible label.

We define $|\langle Lb\rangle|$ as the number of different labels per layer, and $|\langle A\rangle|$ as the number of different adaptations per layer. T, the average number of different technologies per layer is:

$$T \approx |\langle Lb\rangle| \times |\langle A\rangle| \tag{7.38}$$

The maximum number of adaptation stacks no longer depends on the labels, and reduces from $T^{|\mathcal{Y}|}$ to $|\langle A\rangle|^{|\mathcal{Y}|}$. The running time for algorithm 7.2 increases from $\mathcal{O}(1)$ to $\mathcal{O}(|\langle Lb\rangle|)$. Thus, the estimate of the average running time of algorithm 7.1 in equation 7.37 changes to:

$$\mathcal{O}(\text{Algorithm 7.1}) \approx \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times |\langle A\rangle|^{|\mathcal{Y}|} \times \big(\log(|N| \times T^{|\mathcal{Y}|}) + \log(|\langle Lb\rangle|)\big)) \tag{7.39}$$

7.6.2 Extension to Graph G_s

Graph G_s of the network of figure 7.5 is given in figure 7.7. The first notable characteristic is the many different adaptation stacks. The five different technologies (one for Ethernet, two for SONET and two for WDM) lead to seven possible adaptations stacks: Ethernet, Ethernet in STS-24c, Ethernet in STS-3c-7v, Ethernet in STS-24c in 1310 nm, Ethernet in STS-24c in 1550 nm, Ethernet in STS-3c-7v in 1310 nm, and Ethernet in STS-3c-7v in 1550 nm. ($|\mathcal{S}| = 7$, and $\prod_{y\in\mathcal{Y}} |T(y)| = 4$)

The shortest path through the graph G_s is $A_{Eth} - B_{Eth} - B_{24c} - B_{24c;1310} - C_{24c;1310} - D_{24c;1310} - D24c - D_{24c;1550} - C_{24c;1550} - E_{24c;1550} - E_{24c} - E_{Eth} - E_{3c7v} - E_{3c7v;1550} - C_{3c7v;1550} - D_{3c7v;1550} - D_{3c7v} - D_{3c7v;1310} - C_{3c7v;1310} - F_{3c7v;1310} - F_{3c7v} - D_{Eth} - G_{Eth}$.

Many of the vertices in this graph only have two neighbours. One can filter the topology by removing one-degree vertices and by removing two-degree vertices and connecting their neighbours via a direct edge. However, in the process of filtering, we would have to keep track of the adaptation. The graph G_s can be considered as the memory needed to find a path in a layered network, as it explicitly keeps track of adaptation stacks along the path.

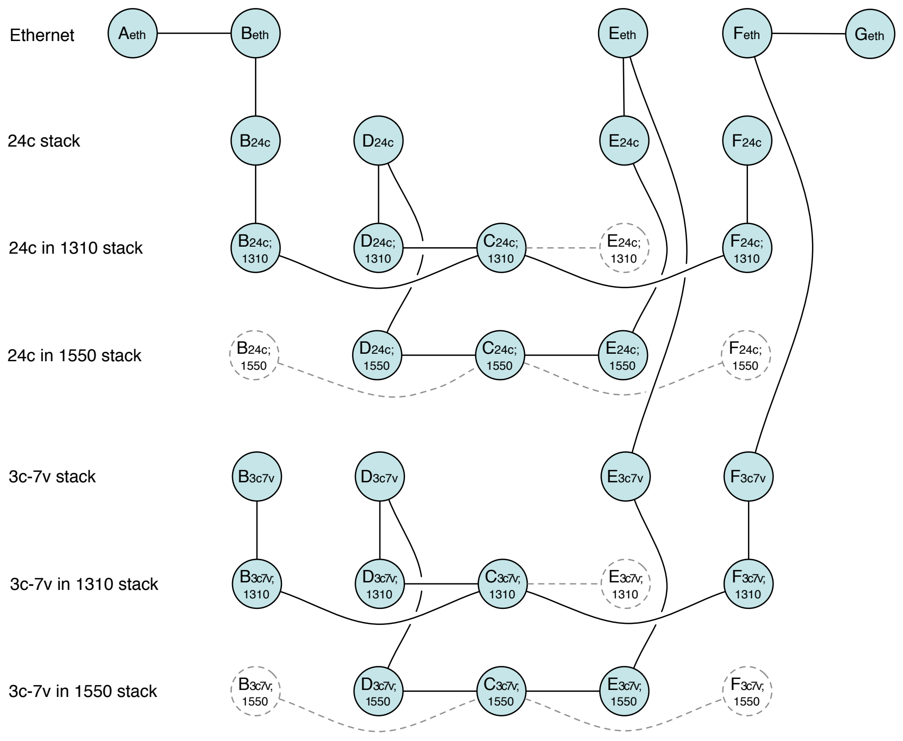


Figure 7.7: *Graph G_s of the network of figure 7.5.*

A few of the vertices in figure 7.7 are greyed, and are not part of $\mathcal{V}_s$. This is due to condition $\mathcal{LB}_s(s) \in LB(c)$ in the definition of $\mathcal{V}_s$ (see equation 7.13). These greyed vertices represent labels, which are not supported by that node. Vertex E for example does not support 1310 nm lasers.

There are only few extensions required to support labels in G_s. In fact, both labels and adaptations are simply instantiations of different technologies, which are reflected in the different technology stacks. The only difference between labels and adaptations is that in $\mathcal{V}_s$, the unsupported labels correspond to missing vertices in the graph G_s, while unsupported adaptations correspond to missing edges.

The changes to include label support do not change the time complexity as mentioned in equation A.18. It only changes the value of T, which is already present in this estimate.

Contrary to G_l, the shortest path through G_s never traverses the same vertex twice. In fact, this is a very useful property, since it means we can limit the number of edges between two vertices to at most one edge, even if there are multiple physical links. The resulting graph G_s is rather efficient. In fact, removing all 'dead ends' in the graph of figure 7.7, ends up with only the links of the shortest path, proving that there is only one shortest path in this example. Such a proof would be very hard for G_l.

In multi-layer networks, the number of incompatibilities may grow quite large. In the given example, the WDM layer only has two wavelengths. However, for Dense Wavelength Division Multiplexing (DWDM), about 100 different wavelengths are not uncommon. For Ethernet, there are 4096 incompatible VLAN tags (incompatible, since it is uncommon for devices to be able to alter the IEEE 801.1Q tag in packets). Tagged Ethernet over DWDM would thus yield about $4096 + 100 \times 4096 = 413696$ rows in G_s. This is contrary to G_l, which would have 2 rows. Of course, the complexity is inherent to the network itself, and emerges as increased memory use for path finding in G_l. For improved scalability, it may be possible to aggregate available channels into groups. For instance, consider an interface with the following VLAN channels available $[1, 50]$, $[53]$, $[89, 93]$, $[106, 123]$, $[400, 530]$ and another interface with the channels $[20, 30]$, $[50, 55]$, $[100, 110]$, $[3000, 4095]$. An intersection of these groups yields: $[1, 19]$, $[20, 30]$, $[31, 50]$, $[51, 52]$, $[53]$, $[54, 55]$, $[89, 93]$, $[100, 105]$, $[106, 110]$, $[111, 123]$, $[400, 530]$, $[3000, 4095]$ which can be represented by 12 (instead of 4096) rows. Both path finding in G_s and G_l benefit from such condensation of the topological constraints.

7.7 Discussion and Comparison

7.7.1 Commonalities

If we consider path finding in G_l and path finding in G_s, there are more similarities than differences between the two algorithms.

- Both algorithm keep track of stacks, and find a path starting at the source, and extending the path hop by hop. G_l does so by explicitly keep track of paths, G_s does so by keeping track of the shortest route to the source for each possible adaptation stack.

- Both algorithms are capable of finding paths with loops (that is, that use the same physical link twice).

- Neither algorithm supports real multiplexing, but only has a concept of bandwidth.
- Both network descriptions assume there is only one channel per layer per node.

7.7.2 Differences

We summarise the differences between the graphs G_l and G_s and the associated algorithms:

- G_s has a simple graph in the algorithmic sense. It only extends vertices, edges, with a bandwidth shared among those edges representing the same physical link. The shortest path through G_s never visits the same vertex twice, so there only has to be one edge between different vertices.
- G_s has multiple edges representing the same physical link. Algorithms needs to be adapted to take this into account.
- The edges in G_l representing an adaptation are directed, even in fully bidirectional networks. Moreover, the shortest path through G_l may traverse edges and vertices multiple times.
- Path finding in G_s assumes that the technology diagram has no cycles. If there are cycles, the number of possible adaptation stacks becomes infinite, and G_s can not be created.
- A local network node can be mapped to one or more vertices in G_l without having further knowledge of the network. To map a local node to one or more vertices in G_s, knowledge about the technologies of all layers above is needed.

The last point requires some explanation. In our second example, to describe node C in G_l, knowledge is required about which layer C is aware of (the WDM layer), as well as all physical links connected to C. To describe C in G_s, in addition knowledge is required about the incompatibilities on the higher layer (e.g. STS-24c versus STS-3c-7v), to know that we need to describe C as four vertices in G_s. This property makes it difficult to update G_s is a small aspect of the network topology changes.

The Multi-Layer-Breadth-First algorithm for G_l does not require full knowledge of the whole network, only of the set $\mathcal{V}$ of vertices v at distance $w(s, v) <$

$w(P^*)$, including all adjacencies $ajd[v]$ of v, with s the start point of the algorithm and P^* the shortest path from s to t. While in practice this set $\mathcal{V}$ can easily span most of the network, it may be interesting for extremely large multi-layer networks, and if s and t are relatively close to each other.

7.7.3 Time Complexity

One of the assumptions that can be made in link-constrained algorithms is that a segment of a shortest path is also a shortest path. As we have seen in section 7.3.1, this assumption is not true. Both algorithms introduced in this chapter can deal with this, but at the cost of severely diminished scalability.

The time estimates given in this chapter made the following assumptions, even though the algorithms are not restricted to these cases:

- It is assumed that it is only necessary to keep track of shortest path per technology stack per node.
- Path finding in G_s assumes that the technology diagram has neither branches nor cycles.

The estimate number of vertices and edges, as well as the actual figures for both example networks (figure 7.2 and figure 7.5) is listed in table 7.1. In these numbers, multiple links between the same pair of nodes are counted once, so we can more easily compare these figures. The numbers between brackets count all individual links.

The number of iterations required for each algorithm is listed in table 7.2. The shortest path in example 1 contains 12 vertices (11 edges). The shortest

		Example 1 (figure 7.2)		Example 2 (figure 7.5)	
		estimate	real	estimate	real
G_p	$\lvert\mathcal{V}_p\rvert = \lvert N\rvert$	6	6	7	7
	$\lvert\mathcal{E}_p\rvert = \lvert L\rvert$	6 (7)	6 (7)	10 (6)	10 (6)
G_l	$\lvert\mathcal{V}_l\rvert \lessapprox \lvert N\rvert \times \lvert\mathcal{Y}\rvert$	12	9	21	14
	$\lvert\mathcal{E}_l\rvert = \lvert A\rvert + \lvert L\rvert$	10 (11)	10 (11)	14 (18)	14 (18)
G_s	$\lvert\mathcal{V}_s\rvert \approx \lvert N\rvert \times T^{\lvert\mathcal{Y}\rvert}$	12	13	28	27 (33)
	$\lvert A\rvert + \lvert L'\rvert \leq \lvert\mathcal{E}_s\rvert \lessapprox (\lvert A\rvert + \lvert L\rvert) \times T^{\lvert\mathcal{Y}\rvert}$	9 - 20	13 (14)	14 - 72	26 (32)

Table 7.1: *Estimates and real sizes of the graphs.*

path in example 2 contains 23 vertices (22 edges).

		Example 1 (figure 7.2)			Example 2 (figure 7.5)		
		queue size	adj. checks	hop iter.	queue size	adj. checks	hop iter.
G_l	Breadth First	17	43	284	836	3113	46321
	Breadth First w/ collapsed links	13	31	189	36	81	945
G_s	k-Shortest Path w/o node removal	13	27	95	28	57	381
	k-Shortest Path	13	25	90	25	48	306
	k-Shortest Path w/ previous adjacency skipping	13	14	79	25	25	283
	k-Shortest Path w/ full distance estimate	13	24	89	24	45	298

Table 7.2: *Number of vertices and edges processed for path finding in two example networks.*

Each algorithm loops through a number of paths or vertices. For each of these iterations, it checks all adjacent neighbours. The following variants are tested:

Collapse Link By default, we model multiple links between the same pair of nodes as one link, with the combined capacity. Alternatively, the links can be individually described. In G_s all links are collapsed by default;

Node Removal The graph G_s contains information about every possible adaptation stack by default. This can be used to remove additional nodes, such as E with 24 STS channels over 1310 nm (see figure 7.4);

Skip previous adjacency By default, all adjacencies are checked. In G_s, a shortest path will never return to the node it was coming from, so that adjacency can be skipped. In G_l this is not true. A valid path may return over the previous edge in G_l if the node is capable of label conversion;

Estimate full distance By default, the Extract-Min procedure in both algorithms extracts the path with the shortest length so far. However, the Multi-Layer k-Shortest Path algorithm (listing 7.4) will instead choose the path with the shortest estimate of the total length. The total length is calculated by appending a lower bound (found with Dijkstra's algorithm) to the length of a path so far. Although it is possible to implement such estimate in path finding in G_l, we have not implemented that.

Only exact algorithms are listed. Variants such as the Multi-Layer-Dijkstra (see appendix A.2) are not included in this table. In order to compare the algorithms for G_l and G_s the variants *Multi-Layer Breadth First with collapsed links* for G_l and *Multi-Layer k-Shortest Path without node removal and without Dijkstra's algorithm* for G_s should be examined, as these algorithms are basically the same, albeit using a different graph.

		Barabási-Albert ($n = 30$, single link)			Barabási-Albert ($n = 30$, double link)		
		queue size	adj. checks	hop iter.	queue size	adj. checks	hop iter.
G_l	Breadth First w/ collapsed links	196	245	762	191	237	737
G_s	k-Shortest Path w/o node removal	194	240	656	207	257	711
	k-Shortest Path w/o node removal	21	29	37	21	29	36

Table 7.3: *Average number of vertices and edges processed for path finding in twenty random networks. The twenty networks were generated with* 30 *nodes and up to* 3 *edges per new node.* $n = 30, m = 3$. *Both networks are single layer networks.*

Before we draw conclusions, we first must realise that that two example graphs in this chapter are specifically crafted to contain many incompatibilities. Table 7.3 gives the average results for 20 random networks, generated according to the Barabási-Albert preferential attachment model. Each random network was processed four times: once with a capacity of 1 per link, and once with a capacity of 2 per link. All random graphs are single layer network, and do not contain any incompatibility whatsoever. It remains to be seen how this applies to real life networks. From these 'incompatibility-free' networks we can conclude that indeed, the *Multi-Layer Breadth First with collapsed links* algorithm and the *Multi-Layer k-Shortest Path without node removal and* *without Dijkstra's* algorithm have a similar running time. Also, it is immediate clear that a breadth first search algorithms is very inefficient if it has no hint as the direction of the destination. The trick used to use a lower bound using the Dijkstra algorithm (first presented by Kuipers) certainly helps in this respect.

We expected algorithms in G_s to be marginally faster than algorithms in G_l. While this is the case for single layer networks, it is not the case for the manually crafted networks. In particular, the extremely high results for the Multi-Layer Breadth First search in example network 2 were unexpected. Apparently, the degrees of freedom caused by the abundance of links severely degrades the running time in this particular example.

7.8 Conclusion

In this chapter we have discussed the problem of finding paths in multi-layer networks. We have considered modelling a multi-layer network as a graph based on nodes and layers (G_l), and a graph based on nodes and technology stacks (G_s). For each model we have discussed the problem of finding feasible paths, given an algorithm and discussed the pros and cons of both approaches, including an estimate of the running times of the algorithm.

We found that a shortest path can contain loops, and both algorithms can deal with that situation.

In addition, a segment of a shortest path does not have to be a shortest path in itself. Again, both algorithms can deal with that situation, but the running time becomes exponential.

G_s is a simple graph in the algorithmic sense, and a shortest path can not contain loops. Nevertheless, it is not possible to apply a simple algorithm such as Dijkstra's algorithm or a Breadth search first algorithm to G_s.

Concluding, a graph based on layers and nodes is probably most suitable to describe actual networks in a multi-domain environment, where domains are reluctant to provide details about their own networks and there is no full topology knowledge. However, the simplicity of the algorithm for the graph based on nodes and stacks makes it more suitable for path finding calculations.

Whichever algorithm is used, we have conclusively proven that path finding in multi-layer networks is far more complex than path finding in single layer networks, and that assumptions valid for path finding in single layer networks no longer hold.

Chapter 8

Path Finding Implementation

This chapter brings the results of the previous chapters together: the model in chapter 4, the syntax and technology applicability in chapter 6, and the path finding algorithm in chapter 7. This chapter describes a software implementation of the algorithm, as well as description of the technology and a network, and shows this input and algorithm indeed are capable of producing the shortest paths in a given multi-layer network.

8.1 Modelling the Network

We turn to the example network in figure 3.2 of chapter 3, and model that network in practice. Figure 4.8 already showed how to model this network using functional elements. However, that model did not explicitly take inverse multiplexing into account. A practical implementation should model that, and may also describe the layers below the SDH/SONET layer.

Beside the topology, the technologies must also be described, and we did so in chapter 6. In section 6.2 we saw the implementation of a conceptual layer schema, and in section 6.3 we saw the implementation of technology schemata. Our claim is that we described most technologies in a generic way, using only a few classes (Layer, Label, and Adaptation). This allows the creation of a path finding algorithm that does not have to be changed if new technologies come along, by only relying on these three concepts instead.

Looking at table 6.2, we model the technologies in our example network as the following layers: Ethernet, VC-4, STS-3, OC-192, Lambda, Fibre and UTP, with the following adaptations: Ethernet in 8 VC-4, Ethernet in 7 VC-4, VC-4 in STS-3, 64 STS-3 in OC-192, OC-192 in Lambda, Lambda in Fibre, and Ethernet in UTP.

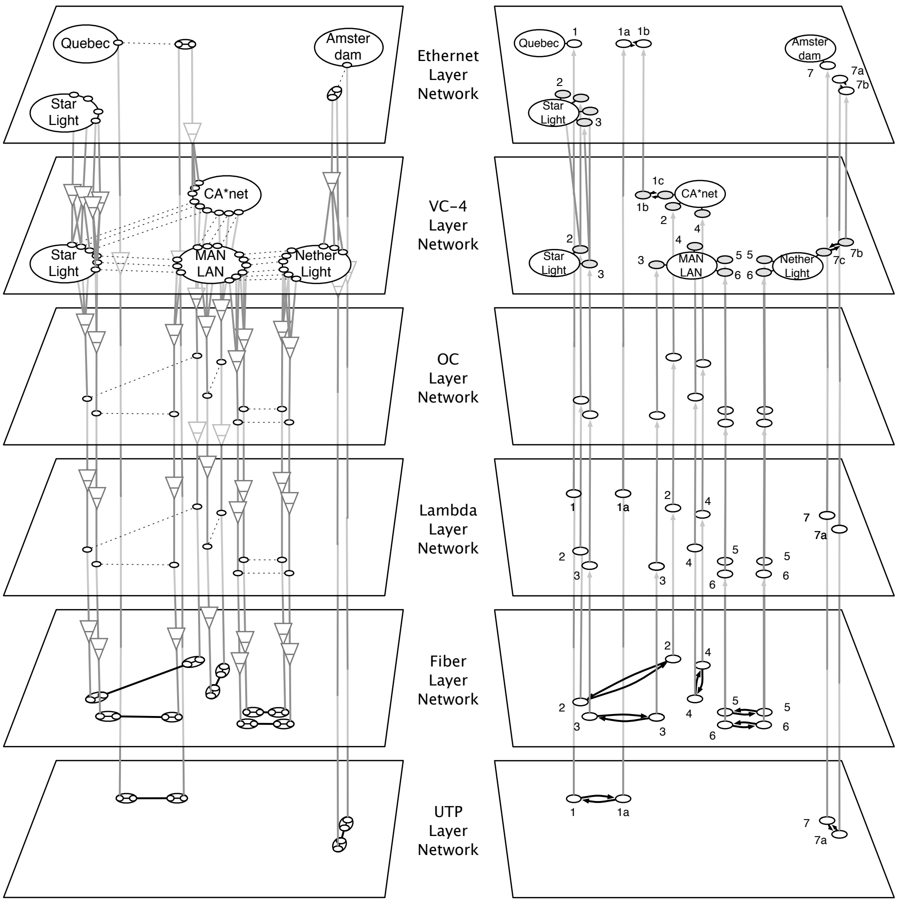


Figure 8.1: *Functional model (left) and graphical representation of the syntax (right) of the network of figure 3.2. The use of* potential interfaces *(the grey interfaces) reduces the number of logical connection points.*

If we also incorporate inverse multiplexing, we model the figure as seen in figure 8.1 on the left.

In order to turn this functional model into the syntax, we use the RDF syntax as defined in chapters 5 and 6. The result is a set of instances with triplets to describe the relation between instances. We can visualise this by using vertices for instances and edges for statements. If we group all instances with the same layer property together, we get the image as seen on the right of figure 8.1. Each circle is an instance of an RDF class; either a *SwitchMatrix* (the named circles), *ConfigurableInterface*, *StaticInterface* (the white circles) or *PotentialInterface* (the grey circles). All lines are RDF properties, either an adaptation properties we defined in the technology schemata, a *switchTo*, *linkTo*, or *hasInterface* property.

While the general structure of the syntax is the same as the functional model on the left, there are a few changes, beside the obvious change from functional elements to RDF classes and properties.

The syntax on the right is more compact than the functional model on the left due to the use of **potential interfaces**. Rather than describing each channel, the VC-4 and STS-3 channels are described as a set of potential interfaces for each physical interface.

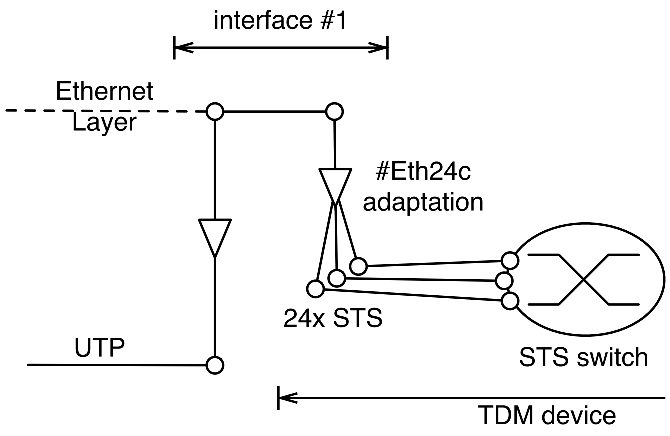


Figure 8.2: *Functional model an inverse multiplexing network interface.*

In section 4.6.2 we discussed that there can be interfaces that have two adaptation stacks. Interface 1 at CAnet (the one connected to Quebec) is such an interface. Figure 8.2 shows an abstract representation of this interface.

Incoming traffic is first de-adapted from the physical layer (in this example, the UTP layer) to Ethernet. The Ethernet packets are subsequently adapted in 24 STS channels, which are connected to the switch matrix. While the connection point after the de-adaptation is logically the same as the connection point just before the adaptation, our network description modelled it as two Interfaces, connected by a cross connect (*switchTo* property). In addition, the connection points after the adaptation are modelled as two Interfaces, connected with a link (*linkTo* statement). While this is not required by the syntax, and arguably is not completely in line with the true meaning of the linkTo and switchTo semantics, it makes our model easier to parse: with this restriction, all de-adaptations lead to a switch matrix (or cross connect), while all adaptations lead to a link. Also, no de-adaptation is immediately followed by an adaptation (there is always a switchTo in between them).

8.2 Software Framework

In addition to the schemata for both the technologies and the network, we created an experimental software framework in Python, called Python NDL toolkit (Pynt). The framework implements classes for layers, adaptations, labels, label sets, as well as devices, logical interfaces, switch matrices, links and domains.

The Pynt software is freely available under a BSD-style license [u1].

This framework is intended for use in various tools:

- Description of the current configuration of our network, and trace network connections;
- Generation of sample networks;
- Path finding of multi-layer connections through the network;
- Fault Isolation of errors in multi-layer network connections.

Using this framework, we were able to make multi-layer network descriptions of the example network given in chapter 3.

In addition, we also modelled an Ethernet network with tagged and untagged links, and our own network containing optical cross connects, Ethernet switches and end hosts. The configuration of our network is automatically generated from the state of the devices using a cron job. The other sample networks were created by hand.

The framework could be used to create a random network. However, it is not obvious how a realistic random network generator would look like. While algorithms such as Barabási-Albert preferential attachment model [p1] or Holme-Kim algorithm [p17] create realistic graphs with power-law degree distribution, this only helps in a realistic network of devices or of domains, thus a single-layer network. It does not help to generate a realistic layer distribution– most multi-layer networks are far from random, since network engineers consciously try to avoid incompatibilities. It is unclear how the distribution of incompatibilities is in practice.

8.3 Path Finding Software

The path finding algorithm is one of the modules of the whole software framework.

8.3.1 Path Finding in G_l

Using the software framework described in the previous section, we build a path finding algorithm. We used the algorithm for path finding in G_l, as described in section 7.4. The reason to choose for path finding in G_l as opposed to G_s is that it is is easy to map the network topology to the model, as explained in section 8.3.5.

We did modify the algorithm as described in chapter 7 slightly: instead using domains for nodes, we used interfaces for nodes. The advantage was twofold: first this is closer to the model we defined (which does define physical devices and logical interfaces, but not logical devices), and secondly it allows us to take the switching and swapping capabilities of switch matrices into account.

8.3.2 Software Logic

Input:

- URI of source and destination interface; and
- URL of network description that includes the source interface.

Output:

- Sequence of connection points, connection type between the connection points (link, adaptation, de-adaptation or cross connect), and available labels for each connection point; or

- Error, in case no shortest path can be found.

In the initialisation phase, the software reads the network description and determines the technologies that are used for that network description. It then downloads and parses those network descriptions before proceeding. Figure 8.3 shows the output of this initialisation phase for an example network.

```
Fetching technology descriptions
Parsing RDF input http://www.science.uva.nl/research/sne/schema/ethernet.rdf
Parsing RDF input http://www.science.uva.nl/research/sne/schema/tdm.rdf
Parsing RDF input http://www.science.uva.nl/research/sne/schema/wdm.rdf
Parsing RDF input http://www.science.uva.nl/research/sne/schema/copper.rdf
Building description of given network in memory
```

Figure 8.3: *Output of the initialisation phase of the path finding software.*

The main routine of the software keeps a list of possible paths, which is initialised with one path consisting only of the source interface.

After initialisation, the software enters a loop where it continuously picks the shortest path from the list and checks if it is a valid path from source to destination. If so, it returns it as the shortest valid path. If not, it extends the path with a hop to a neighbouring interface, and checks if that could lead to a valid path, eliminating hops with incompatible labels or incompatible de-adaptation functions. It adds these extended options to the list of possible paths and continuous the loop. This is the Multi-Layer-Breadth-First algorithm described in chapter 7.

8.3.3 Path Walk

The routine in the path finding algorithm that checks if an extension of a path can lead to a valid path or is unfeasible contains the logic of the algorithm. This is the Extend-Path subroutine that is part of the Path selection in G_l algorithm, as described in chapter 7.

If we modify this subroutine, we can create other algorithms. For example, we also created a path walk algorithm. This path walk algorithm does not extend the path with *possible connections*, but extends it with *currently configured connections*. The result is that it simply follows existing connections (including point-to-multi-point connection in multicast or broadcast technologies such as Ethernet), rather than viable connections.

We use this path walk algorithm as the basis for a fault isolation framework, which is able to detect anomalies in the published network configuration, and thus detect and isolate faults across domains.

8.3.4 Switch Matrix Properties

One of the findings whilst implementing the logic to determine if a path is valid or unfeasible is more complex than we anticipated. For example, we first defined that a bi-directional cross connect between $cp1$ and $cp2$ exists if there is an unidirectional cross connection from $cp1$ to $cp2$, and there is an unidirectional cross connection from $cp2$ to $cp1$. Now, for a uni-directional loopback connection holds that $cp1 = cp2$. Thus according to our definition, this is also a bi-directional cross connection. However, this is not intuitive, and we had to redefine our definition of “bi-directional” to explicitly exclude this particular case.

The following section lists the properties of a switch matrix:

- switching capability: two interfaces with the same label can be crossed.
- swapping capability: two interfaces with a different label can be crossed.
- unicast: an interface can be crossed to another interface, provided that the labels match, and no other cross connect exist for the source or destination.
- multicast: an interface can be crossed to another interface, even if other cross connects with the same source exist.
- broadcast: if and only if the labels of two interfaces match, they are crossed. Broadcast is mutually incompatible with unicast.

The labels as defined above are the internal labels of interfaces. These can be different from the external (egress and ingress) labels, even though in practice the same channel identifier is used. An empty or undefined labelset ($\varnothing$) is considered to be a labelset with one element, the empty label ($\{\epsilon\}$).

Given the switching capabilities of a switch matrix, and the connected interfaces, software can enquire about the following cross connects:

Actual cross connects returns currently configured cross connects.

Potential cross connects returns cross connect that can be made by allowing any other cross connect to be broken. Takes label sets into account, but ignores current labels and current cross connects.

Available cross connects return cross connect that can be made without affecting any existing traffic (cross connects). Honours label sets, cross connects, but only labels if they are part of existing cross connects.

By default all checks only verify if a unidirectional cross connect from source to destination can be made. If the bidirectional modifier is given, it checks for the reverse connection as well. This check significantly reduced the number of possible cross connects for switch matrices with multicast capability.

The request for cross connects set can be further modified by the modifiers:

bidirectional Makes sure the reverse cross connect is also in place or is (potentially) available. (Applies to actual, potential and available cross connects)

break own cross connect Do not honour cross connects originating from source and (if bidirectional) destined to the source interface. All other cross connects remain in place (including multicast and broadcast cross connects that are part of the same group). (Applies to available cross connects only)

allow data merger Allows cross connects that create new data mergers (currently only supported for broadcast switch matrices with more than two interfaces with the same label.) A merge may affect the available bandwidth of existing connections. (Applies to available cross connects only)

honour label Honour the current label of all interfaces, not just the labels of interface that are part of cross connects. (Applies to potential and available cross connects)

So a request for available bi-directional cross connect, for a unicast switching matrix will not return cross connects with a connection point as destination if that connection point is already in use by an existing cross connect. If *allow data merger* is set, then it will return such cross connect.

While the exact details of the above logic are interesting, the main point is that it shows that the logic to check for available network connections is far from trivial, despite our relative simple network model. The logic of a technology dependent model would be even more complex, and it would indeed be unfeasible to deploy such logic reliably in a multi-domain environment.

8.3.5 Multi-Domain scalability

We used the algorithm for path fining in G_l as opposed to the algorithm for path finding in G_s since path finding in G_s would require the construction of a graph incorporating every possible adaptation stack, which requires a-priori knowledge of the whole network.

The algorithm for path finding in G_l only has to fetch information about a domain if there is a path that leads to that domain, but not earlier than that. For that reason, G_l scales better in a multi-domain environment, and that is the reason we implemented that algorithm.

Basically, the path find and path walk algorithm find end-to-end paths by following the links, adaptations and cross connects through domains. When reading a network description, the algorithm stores information about the network elements in memory, including information `seeAlso` statements pointing from a given network element to external resource with further information about that network element.

As soon as the path finding algorithm extends the path to a network element that contains a `seeAlso` statements, it then loads that information, but not earlier than that. In this way, the algorithm can follow the path through multiple operational domains.

8.3.6 Result

If we feed the implemented algorithm in our example network, it is indeed able to find the shortest valid path.

Figure 8.4 shows a graphical representation of the classes and properties that make up the network description. Edges in figure 8.4 represent RDF properties between the different instances in the model, which in turn represent either link connections or adaptations in the network.

This vertices and edges in this figure should be **exactly** the same as the right hand side of figure 8.1. The only difference is that the vertices (RDF instances) in figure 8.1 are grouped by layer, while in figure 8.4 they are grouped by domain. The different colours on the vertices represent the different layers.

The figure also shows the end result of path finding algorithm. The lighter green edges are part of the resulting path. The part of the network between CAnet and MAN LAN is used twice, as is represented by a thicker line.

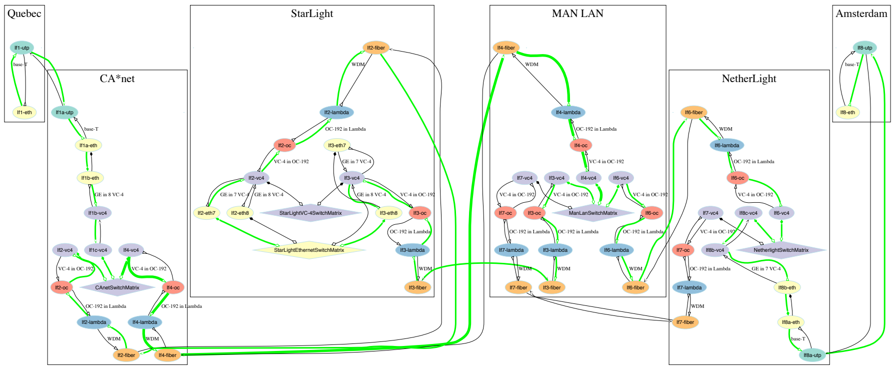


Figure 8.4: *The visual output of an automated path finding algorithm.*

Indeed, this path is the shortest path as we have seen in figure 3.5 in chapter 3.

Given that our algorithm is a path-constrained path finding algorithm, the algorithm and implementation have no dependencies on the actual network or technology, but only on the technology-independent multi-layer network model, and given that we could describe all technologies that are in use in the GLIF community in this model, then we must conclude that **path-constrained algorithms are sufficient for path finding in multi-layer networks**, at least for all technologies as used in the GLIF community.

This claim has implications for the information exchange between domains. In case a path finding is done domain-by-domain (as opposed to having one central orchestrator), each request must contain information about the final source and final destination. If that information is not present, the request may lead to false positives or false negatives.

8.3.7 Ambiguity of Labels

In section 6.3.7, we discussed that the label concept is used for two separate meanings: (1) to distinguish between channels in multiplexing; and (2) to determine allowed cross connects in switch matrices using the switching and swapping capability.

Recall that Ethernet labels are VLANs, and are either an integer in the range $0 - 4095$ or the empty label ϵ. In all circumstances, connection points have a label which is used in the second meaning, to determine valid cross connects (the labels must be the same: it is not possible to exchange traffic between two different VLANs). For untagged Ethernet, this VLAN label is not present on the wire, and the external egress label is ϵ. For tagged Ethernet, modelled as Ethernet over Ethernet, the VLAN label is present on the wire as the IEEE 802.1q label, and the external egress label must be set after the cross connect.

We came up with the following logic to determine if the external egress label is set:

- If a label is set for an Interface, then it is assumed to be the internal label, and used in the switch matrix to determine if a cross connect (switchTo) is allowed.
- If a switch matrix has swapping capability, the label may change. The next hop uses the label constraints of the interface. If the switch matrix has no swapping capability, the next hop uses the intersection of the label constraints of the interface and the path so far.
- In addition to label swapping in a matrix, labels may change along the path if the label is not carried on the wire. For example, an untagged Ethernet interface in VLAN 42 may be linkTo an untagged Ethernet interface in VLAN 28. The logic to determine if a label is carried on the wire is:
    - there is a `hasexternallabel` property explicitly set to true
    - if the interface is client of a multiplexing adaptation, the label is carried on the wire.
    - if the interface is client of a non-multiplexing adaptation, the label is **not** carried on the wire (and may change)
    - if there is no adaptation, but only a link, the label is carried on the wire.

Since this logic is mostly deterministic, and not 100% accurate, it is now recommended to explicitly set the `hasexternallabel` property, which signifies if a specific interface is tagged or untagged Ethernet.

8.4 Optimization

Section 8.3.3 discussed the routine in the path finding algorithm that checks if an extension of a path can lead to a valid path. This routine should never return a path that is unfeasible. That is, a section of a path which can never be part of a valid connection.

The software documentation describes this routine as follows:

> Verifies if a path is valid. Only the last hop in the path has to be checked; it can be assumed that the rest of the path has been checked earlier.
>
> Typical checks to perform are:
>
> - is the adaptation stack valid, especially if the connection is De-Adaptation
> - does the current connection point further restrict the number of allowed labels
> - are there more available labels left then the interfacecount
> - does the current connection point restrict the number of layer Properties to None
> - does it not give a loop (i.e. has the interface been used before in the same path, with the same stack). This a non-obvious check in multi-layer networks: data may get transported through two channels, which use the same physical interface.
> - is the configuration not forbidden because an earlier configuration (e.g., we can't use the same VLAN if it was already used earlier in the same path)
> - Are there policy constraints.
> - etc., etc.

However, there are no restrictions if it should only return a subsection of a path which can never be part of the shortest **valid** connection. In fact, since it does not know the shortest valid connection beforehand, it will regularly return paths turn out to be "dead ends", paths that are not part of the shortest valid connection. It is possible to optimise this part.

We developed a few variants to our base algorithm.

Unrestricted flooding No optimisation. Practically impossible loopbacks, such as $A-B-C-B-C-D$ are not filtered out from the intermediate results, although they never show up as final result since they are clearly not shorter than $A - B - C - D$.

No loopbacks Do not allow a path to return in the direction it just came from. E.g. $A - B - C - B$. Such loopbacks can be part of a shortest path if C is a potential interface in a switch matrix with label switching capability.

Explicit direction Define two directions: inbound and outbound, and keep track if a path is currently going inbound or outbound. A switch matrix or cross connect changes the direction from inbound to outbound, and a link connection changes it from outbound to inbound. Adaptation are always outbound, de-adaptations always inbound. This simple check eliminates the need to check for impossible loopbacks inside a device, such as a cross connect followed by another cross connect.

No repeated stacks in path Broadcast suppression. If we reach an interface with a certain stack, but this path already contains a hop with this interface and the same stack, we can stop the algorithm.

All the above algorithms will come up with the correct shortest path. Crucial to all above algorithms is the loop detection. Each interface may be used multiple times if it contains enough available channels. These channels may not be on the layer of the last connection point in the path, but may be available higher up the adaptation stack. This can be avoided by only checking the available channel count check right after a cross connect of a potential interface, which is the client interface of a multiplexing adaptation function, and not to check at all other interfaces.

Besides these algorithms, we also developed a few heuristic algorithms that contain a less elaborate loop detection mechanism, at the cost of given false positives or false negatives.

Shortest path to stack only Broadcast suppression. If there are two ways to get from A to B, then the second path that traverses along switch matrix B stops there. This yields false negatives if the shortest path uses a link that is required later. This optimisation greatly reduces the flooding mechanism of breadth first search algorithm, but can not be used if the goal is to find multiple (backup) paths.

Interfaces used once Interfaces may only be used once. This yields false negatives if the shortest path contains a loop (that path will not be found).

Adaptation Restriction only Do not check for the number of available channels nor check for compatible labels. This will yield false positives if the number of available channels affects the shortest path, such as in our example network on the link between CAnet and StarLight.

Topology Restriction only Do not check the number of available channels, compatible labels or compatible adaptations. This will yield false positives if the shortest path is restricted by the number of available channels, or is restricted by an incompatible adaptation such as in our example network between CAnet and NetherLight.

Table 8.1 shows the number of iterations that are required to find the shortest path in the example network introduced in chapter 3.

Algorithm variant	**Number of iterations**
Unrestricted flooding	$1.6 \times 10^{15} \pm 36\%$ iterations
No loopbacks	$18 \times 10^{12} \pm 27\%$ iterations
Explicit direction	587 iterations
No repeated stacks in path	486 iterations
Shortest path to stack only	245 iterations
Interfaces used once	No result found
Adaptation restriction only	False positive after 219 iterations
Topology restriction only	False positive after 134 iterations

Table 8.1: *Number of iterations required by algorithm variants to find the shortest path in our example network.*

The *unrestricted flooding* and *no loopback* variants are extremely inefficient, and it was not possible to finish the algorithm run. Thankfully, the tree built by the algorithm grows nearly exponentially (the correlation parameter R^2 is between 0.9993 and 0.9998), and it is possible to estimate the finishing time after running the algorithm for roughly 50,000 iterations, since we know that the shortest path is 57 hops long (a metric length of 56.0 if adaptations, links and switched all have a metric length of 1.0). The error in the table comes from the sheer number of possible paths of 57 hops long, and it is unknown if the correct path is found in the first or the last iteration, or somewhere in between.

The *explicit direction* variant finds alternative paths. In our example, it finds the path Quebec – CAnet – MAN LAN – NetherLight – MAN LAN – StarLight – CAnet – MAN LAN – NetherLight – Amsterdam (with a pointless extension MAN LAN – NetherLight – MAN LAN) after 1855 iterations. The *no repeated stacks in path* variant does not finds alternative paths, and stops after 317 iterations.

The *explicit direction* variant, which is the fastest exact variant, does return two solutions, after 486 and 488 iterations. The second solution uses the second link between MAN LAN and NetherLight. The *shortest path to stack only* variant does not find this alternative path.

The *adaptation restriction only* variant finds the false positive Quebec – CAnet – StarLight – MAN LAN – NetherLight – Amsterdam, as we predicted in figure 3.4 in chapter 3. The *topology restriction only* variant finds the false positive Quebec – CAnet – MAN LAN – NetherLight – Amsterdam, as we predicted in figure 3.3.

The *interface used once* variant does not find a path, because the shortest path in our example contains a loop: the same interface *is* used twice. In order to compare this algorithm variant with others, we created an alternative network example with two link between CAnet and MAN LAN, and additional restrictions in available time slot labels. In this case, the *interface used once* variant is able to find the solution after 6710 iterations. This a high number. The *no repeated stacks in path* variant applied to the same networks finds the shortest path after 534 iterations.

8.5 Conclusion

In conclusion, both network example gives us a clear indication that the no repeated stacks in path variant of the algorithm is able to significantly restrict the broadcast nature of our algorithm, and this seems like a very viable multi-domain multi-layer path finding algorithm. Multi-layer, because it can cope with constraints in layers, adaptation and labels. Multi-domain, because it does not need full topology knowledge beforehand.

Further optimisation, including the use of completely different algorithms, may be subject of further study. We have presented a model, syntax, algorithm and software framework that are a good start for such research.

Chapter 9

Discussion and Conclusion

9.1 Context and Goals

A number of e-Science applications need deterministic point-to-point connections with very high bandwidth and predictable behaviour.

The need for dedicated network connections is in practice fulfilled by interconnecting multi-layer networks where the dedicated resources are allocated for each network connection.

In this field of multi-layer multi domain transport networks there is (until now) neither an agreed upon network model, nor a suitable end-to-end path provisioning algorithm.

The aim of this work is to make a system level analysis of multi-domain, multi-layer transport networks. In doing so, it contributes to the design of a comprehensive control plane for these transport networks.

Actual contributions include the proof that graphs that simply represent devices or domains as vertices and links as edges are not sufficient to describe multi-layer networks, because that ignore the adaptation functions. We have shown that that link-restriction algorithms are not sufficient for path finding algorithms but path-restriction algorithms are sufficient.

Logical reasoning and the implementation and demonstration of path-restriction algorithm is evidence of the correctness of our claims.

9.2 Contributions to the Field

This thesis contributes to the new field of hybrid networking, including the modelling of optical exchanges, ontology for network descriptions with separation of topology and technology, and proof that link-constrained path finding algorithms do not work in multi-layer networks. The work is validated by implementations and proof of concepts.

Path finding through multi-layer networks It was shown that link-constrained algorithms in use for single layer networks such as the Internet and the telephony network are not applicable for multi-layer networks. A path-constrained algorithm was introduced that is suitable for path finding in multi-layer networks. A shortest path in a multi-layer network can contain a loop.

Modelling of optical exchanges Both practical models and a concise terminology for exchanges was created. While the data plane for Internet exchanges and optical exchanges are similar, there is a clear distinction on the control plane. Optical exchanges can not be transparent to a path finding if they offer services such as data conversion.

Ontology for network descriptions In order to describe networks and network technologies, multiple ontologies were created. The ontologies are built upon the resource description framework (RDF), and allow integration with other ontologies. The network description ontology is used in network tools for visualisation, path finding and fault isolation, and is used in production at the SURFnet6 NOC.

Separation of topology and technology information Separate ontologies were created to describe networks topologies and network technologies. It is possible to reason about topologies independent from the technologies. The combination of the two allows network engineers to describe compatibilities and incompatibilities in their network.

9.3 Strengths and Weaknesses

9.3.1 Architecture

When we started this work, the concept of hybrid networks was just established [p23], but there was no concise model or even description of optical exchanges or hybrid networks. In two peer-reviewed articles, we described the architecture of optical exchanges, both at the data plane as well as the control plane. This work was essential to establish a more formal model, as we later did.

We asked ourselves two questions, a generic question and a specific question. The generic question asks if there is a fundamental difference between hybrid networks and existing networks such as the Internet or the telephony network? The specific question narrows this to: Can optical exchanges, just like Internet exchanges, be completely transparent to a path finding algorithm in circuit switched networks?

We could answer our specific question with a reasoning about the terminology for transparency. We conclude that exchanges that offer adaptation or interworking services can not be transparent to a path finding algorithm.

This conclusion is relevant, as we also reason that most optical exchanges will have to offer adaptation or interworking services. Technologies change over time, which implies that incompatibilities between interfaces, adaptations and layers will continue to occur. Exchanges must take potential incompatibilities into account to avoid non-working network connections. Services like adaptation and interworking must be offered by exchanges or elsewhere in the network to solve incompatibilities.

Our work did not stop by answering this question alone. We found that the distinction between Internet exchanges and optical exchanges lays at the control plane, rather than at the data plane. While Internet exchanges and optical exchanges may use the same technologies, they are different at the control plane: optical exchanges change state with each connection request, while the state of Internet exchanges only change when peering relations change.

9.3.2 Modelling

A large part of the work in this thesis is modelling of multi-domain and multi-layer networks.

This work was initiated after we found that, to our surprise, there was no comprehensive multi-layer network model available yet. Our work made three distinct choices:

- The clear distinction between a network, a functional representation of the network, and a syntax describing the functional description.
- The use semantic web technologies such as RDF [p41, a3].
- The use of G.805 functional elements and GMPLS labels to model multi-layer networks.

Using the resource description framework (RDF) immediately solved two engineering questions: naming of interfaces, domains and hosts, and the multi-domain problem using a distributed knowledge base. In addition, it allowed easy integration of the terminology we developed earlier into a formal ontology.

The very strict distinction between the physical network and the functional representation of that network (the model) allowed us to see that graphs are not sufficient to describe multi-layer networks, despite their use to describe single-layer networks. As far as we are aware, we are the first to recognise this.

Not only did we recognise that graphs are not sufficient to describe multi-layer networks, we also provided an alternative, using G.805 functional elements in combination with the label concept of GMPLS.

The distinction between model (using functional elements) and syntax was also beneficial. While this distinction may seem trivial, it is extremely important when it comes to creating control plane software for those multi-domain networks. As we write, each domain deploys their own control plane software (for example, UCLP, DRAC, G-Lambda, AutoBAHN, DRAGON, etc.). If each software uses a different syntax, but the same model, it is easy to translate between the different network topology descriptions. However, if the model is different, this translation is very hard, and sometimes even impossible. In reality, it is often ever harder, since most software packages do not define a formal network model in the first place. For example, GMPLS did not define a formal model, but only a syntax to describe networks. Our work makes software developers aware of this problem, and offers a solution by providing a syntax-independent model.

The strict of modelling not only helps software designers, but also protocol designers. Indeed, proof that our model is considered useful is the interest for our model by members of the network markup language (NML) working group in the Open Grid Forum (OGF), whose goal it is to come up with a standard for network descriptions. In addition, the interest of external parties for the work of the NML working group also shows the importance of such a model [u2]. More and more, the value of a solid network model is recognised.

The network markup language (NML) working group has another benefit: it allows a comparison between our model and the efforts of others. While our model still lacks practical extensions such as change information (our model assumes a static network, while in reality networks change over time), it appears to have a few unique properties that make it stand out in comparison. First is the technology independence: our model is the only that allows a path finding algorithm that does not need specific knowledge about each technology. The clear advantage is that our model is *forward compatible*: it is not only compatible with existing technologies, but also with future technologies, provided that they can fit in our model. Secondly is our use of the label concept, we took from GMPLS. This technology independent feature is considered a asset of our model by others. Third, and finally, our model is one of the few that distinguishes in terminology between the network at that data plane and the domain at the control plane.

We have tested the technology independence of layers, adaptation and labels for all network technologies we are familiar with (IP, Ethernet, PPP, ATM, SONET/SDH, VPN tunnel, Wireless, DWDM and CDWM, OXC, Fibre bundles). We concluded that the model works correct for circuit-switched technologies, including all technologies that are used at existing optical exchanges (Ethernet VLANs, SONET/SDH, DWDM and fibre layers), while extensions would be required for physical layers and packet switch technologies that use a routing table. We believe this to be very strong result.

9.3.3 Path finding

By using logical reasoning, we have shown that link-constrained path finding algorithms are not sufficient for path finding in multi-layer networks, if links are 1:1 mapped to edges. We defined a shortest network connection with a loop is defined as a shortest network connection that uses the same physical link twice. We assert that link-constrained path finding algorithms never find a solution with the same edge used twice. Therefore, our claim is valid.

Routing protocols such as BGP and SS7 use a path-vector algorithm and the routing protocol in OSPF-TE/PCE (as used in GMPLS) uses a link-state algorithm for path finding. Both are link-constrained algorithms, and can not be used as-is in multi-layer networks.

We have given two path finding algorithms, both path-constrained, that are designed to find paths in multi-layer networks. One of these is a broadcast-like algorithm, which has the advantage that it uses a graph that is very similar to the actual network topology. It does not a-priori need information about other domains until it has to check for paths through those domains.

This variant is currently the only variant that is capable of dealing with loops, as well as the assertion that a segment of a shortest path does not need to be a shortest path in itself.

Our strength is that we not only defined a model and algorithm, but also made an implementation of our algorithm. By applying this algorithm to a few complex example networks that contain loops, we have shown its applicability. We conclude that path-constrained algorithms are sufficient for path finding in multi-layer networks.

This last proof is not a formal mathematical proof, but merely a proof of concept. By strictly retaining the technology independence of the algorithm, we have shown that the path-constrained path finding algorithm works for all networks and technologies that can be described by our model. That includes all relevant technologies, since those can all be described with our model.

However, we did not formally prove that there is no future technology can be invented that can not be applied to our model, and which would require an even more complex algorithm.

We have seen examples of incompatibilities that are not described in our model. Incompatible packet sizes in Ethernet, and incorporation of scheduling and policy constraints have been explicitly mentioned. Our path-constrained path finding algorithm can still handle those scenarios, by simply adding more constraints to the "feasible path" constraint code. This is a trade-off between the number of constraints that can be handled and the technology independence of the model and algorithm.

Our algorithm is basically a breadth first search algorithm. We have shown a few optimisations that reduce the flooding mechanisms by dropping paths that will not result in the shortest path for whatever reasons (e.g. we drop paths with unnecessary loops). Further research may reveal radical new algorithms that are even more efficient than these optimisations. Despite our collaboration with renowned algorithmic experts from Delft University, such radical new approaches have not surfaced yet.

9.4 Claims and Statements

We focused on one specific question in this thesis.

- Is it possible to use the same path finding algorithms in multi-layer transport networks as those in use for the Internet and telephony networks?

The research we carried out in order to answer this and subsequent questions have accumulated in miscellaneous statements as we put forward in this thesis.

- The use of multiple technologies causes incompatibilities (Section 2.3.1, repeated in Chapter 3).
- Technologies evolve over time. Thus, incompatibilities will continue to exist (Section 2.3.1).
- Exchanges must take potential incompatibilities into account to avoid non-working network connections. These incompatibilities must be described to deal with them (Section 2.3.2).
- Exchanges can only be ignored during path finding if the connections through an exchange are modelled as direct connections, and the exchange does not define a usage policy on its own (Section 2.5).

- Path finding in a single layer network belong to a different complexity class (P) than path finding in a multi-layer network (NP-hard) (Section 3.3, source: Kuipers [a12]).
- the algorithms used in the Internet and telephony network can not be used for path finding in multi-layer transport networks, if links in the network are 1:1 mapped to edges in the graph (Section 3.3.1).
- Link-constrained algorithms are not sufficient for path finding in multi-layer networks, if links are 1:1 mapped to edges (Section 3.3.2).
- Graphs can not be used for path finding in multi-layer networks, if it is required that (1) the outcome of the path finding algorithm is sufficient to reconstruct the original path in the network; (2) the graph can be created from the actual network in polynomial time; and (3) a link-constrained path finding algorithm is used (Section 3.3.2).
- Each of the conditions in the above claim is essential for the claim (Section 3.3.3 and 3.3.4).
- Multi-layer incompatibilities can not be resolved locally, but need to be distributed across domains (Section 3.3.2).
- Path-constrained algorithms are sufficient for path finding in multi-layer networks (Postulated in section 3.3.2, proven in section 8.3.6).
- Multi-layer networks can only be mapped to a graph if devices are mapped to multiple vertices, or if information about the adaptation is lost (Section 3.3.4).
- G.805 and G.800 allow descriptions of the state of a network. No model exists to describe how to change that state, and who may do so (Section 4.2).
- It is easy to translate between two different syntaxes with the same model. It is hard to convert between two models (Section 4.3.1).
- A model can be verbose, while the syntax is compact (Section 4.4.3).
- It is possible to create a distributed network description, without a central repository (Section 5.3.3, source: Van der Ham [a3]).
- The use of semantic web provides the URI as a solution for globally unique addressing of network resources (Section 5.3.4, source: Van der Ham [a3]).
- Incompatibilities change over time, and what needs to be described changes over time (Section 6.1.2).

- A multi-layer path finding algorithms should be layer independent (Section 6.1.2).
- A segment of a shortest path in a multi-layer network does not have to be a shortest path in itself (Section 7.3.1).

9.5 Conclusion

In chapter 1, we put forward a general research question of the research in this thesis is only a small part of, *Is there a fundamental difference between hybrid networks and existing networks such as the Internet or the telephony network? Which of the existing models and approaches can be re-used and which can not?*

Given the statements we made in this thesis and repeated in the previous section, we can only conclude that indeed there are fundamental differences between hybrid networks and existing networks such as the Internet or the telephony network. The concept of networks, and operational domains remains the same. Hybrid networks encompass both circuit switched as well as packet switched technologies. The largest difference between hybrid on one hand, and the Internet or telephony network on the other hand is not packet versus circuit switching, but the fact that the circuit switched technology of hybrid networks is a **multi-layer network** service. This multi-layer nature gives rise to interworking services to stitch technological incompatibilities together. This causes that optical exchanges that offer these interworking services are visible to a path finding algorithm, and that link-constrained path finding algorithms as used for single layer networks can no longer be used. Instead, we have shown that path-constrained algorithms are required, and that simple graph as common to describe single layer networks can not simply describe the multi-layer networks.

All these differences mean that if no single specific technology for hybrid networks is chosen, new multi-layer models and path finding algorithms needs to be developed.

Given that technologies evolve, and thus incompatibilities continue to exist, no single layer or single technology is standardised, and multi-layer network models and path finding algorithms continue to be needed for hybrid networks.

Appendix A

Algorithm Time Complexity

A.1 Running Time of Multi-Layer Path Finding

In both the Multi-Layer-Breadth-First and Multi-Layer-k-Shortest-Path algorithms (listings 7.1 and 7.4 respectively) the running time will mostly depend on the size of the queue Q. The best estimate we can give is to first estimate the length of the shortest path, and than estimate the number of paths of that length. Since the algorithms are basically a flooding mechanism, we assume that the path branches at each hop, and the number of branches after i hops is $\mathcal{O}(r^i)$, with r the number of branches per hop. If we ignore all suppression mechanisms, as we should for the worst-case scenario, the queue length is proportional to the number of branches, $\mathcal{O}(r^i)$. The number of branches per hop r is roughly proportional to the average out-degree and to the number of possible labels $|\langle Lb\rangle|$ per layer. The average out-degree is the average number of adjacencies:

$$|\langle Adj\rangle| = \frac{|\mathcal{E}_l|}{|\mathcal{V}_l|} \tag{A.1}$$

Now we can estimate the queue length.

$$\begin{aligned}\mathcal{O}(|Q|) &= \mathcal{O}(r^i)\\ &= \mathcal{O}(\left(\frac{|\mathcal{E}_l| \times |\langle Lb\rangle|}{|\mathcal{V}_l|}\right)^{|V|})\\ &= \mathcal{O}(\left(\frac{(|N| \times |\mathcal{Y}| + |L|) \times |\langle Lb\rangle|}{|N| \times |\mathcal{Y}|}\right)^{(|N|\times|\mathcal{Y}|)})\end{aligned} \tag{A.2}$$

Assuming that our network is a small world network, the average path length is $\mathcal{O}(\log(|\mathcal{V}|)$ [p1]. While the estimate of the worst-case remain the same, the estimate for the *average* running time would reduce i from $|\mathcal{V}|$ to $\log(|\mathcal{V}|)$:

$$\begin{aligned} \mathcal{O}(|Q|) &= \mathcal{O}(r^i) \\ &= \mathcal{O}(\left(\frac{|\mathcal{E}_l| \times |\langle Lb\rangle|}{|\mathcal{V}_l|}\right)^{\log(|V|)}) \\ &= \mathcal{O}(\left(\frac{(|N| \times |\mathcal{Y}| + |L|) \times |\langle Lb\rangle|}{|N| \times |\mathcal{Y}|}\right)^{\log(|N|\times|\mathcal{Y}|)}) \end{aligned} \tag{A.3}$$

Such exponential behaviour is typical for a NP-complete problem, such as a path-constrained path finding algorithm.

A.2 Multi-Layer Dijkstra's Algorithm

Listing A.1 shows Multi-Layer-Dijkstra, a variant of the Dijkstra algorithm applied to the graph G_s as defined in section 7.5.

This algorithm is an improvement over Dijkstra's algorithm [p11]. Dijkstra's algorithm applied to the graph G_s in figure 7.4 would find the path $A_{Eth} - B_{Eth} - B_{24c} - E_{24c} - D_{24c} - D_{Eth} - D_{3c7v} - E_{3c7v} - F_{3c7v} - F_{Eth} - C_{Eth}$ as shortest path from A to C. This is not correct, due to the limited capacity between D and E. The Multi-Layer-Dijkstra algorithm in listing A.1 would find the correct shortest path, $A_{Eth} - B_{Eth} - B_{24c} - E_{24c} - D_{24c} - D_{Eth} - D_{3c7v} - B_{3c7v} - E_{3c7v} - F_{3c7v} - F_{Eth} - C_{Eth}$.

Nevertheless, the Multi-Layer-Dijkstra algorithm is still imperfect. For example, while it finds the shortest path from A to C, it will not find the shortest path from C to A. The reason is that this algorithm does check for the used bandwidth, but it only keeps track of the bandwidth usage as a global variable, rather than per path. This means that it adds edges to the list of used bandwidth, even if that edge later turns out not be used anymore. This condition is too strict, resulting in false negatives, such as the above.

Lines 1-6 of the meta-code initialises all vertices and edges. Line 7 inserts all vertices in the queue Q. The main algorithm starts at line 8. Line 9 extracts the vertex u from the queue that has the shortest weight (i.e., $d[u] \leq d[v]$ $\forall v \neq u \in Q$). Vertex u can be regarded as the new scanning vertex towards destination v_{dst}. Consequently, we have to reduce the bandwidth of the last edge in the path to vertex u with the amount of consumed bandwidth (which in this case we take from the edge parameter $B_e(e)$). Lines 13 and 14 make sure we only retrieve the shortest path between a source and a single destination (as

Algorithm A.1 Multi-Layer-Dijkstra(G_s, v_{src}, v_{dst})

```
Require: Available bandwidth $B_e(e)$ for each edge $e$
Require: Required bandwidth for each vertex $v = (n, s)$
 1: for all vertices $v \in \mathcal{V}_s$ do
 2:    $d[v] \leftarrow \infty$ {The distance to $v_{src}$}
 3:    $\pi[v] \leftarrow$ NIL {the predecessor vertex to $v_{src}$}
 4: for all edges $(u, v) \in \mathcal{E}_s$ do
 5:    $b\big((u, v)\big) = B_e\big((u, v)\big)$ {Still available bandwidth}
 6: $d[v_{src}] \leftarrow 0$
 7: Queue $Q \leftarrow \mathcal{V}_s$
 8: while $Q \neq \varnothing$ do
 9:    $u \leftarrow$ Extract-Min($Q$)
10:    if $\pi[u] \neq$ NIL then
11:       $(n, s) \leftarrow u$
12:       $b(u, \pi[u]) \leftarrow b(u, \pi[u]) -$ Bandwidth-Required($s$)
13:    if $u = v_{dst}$ then
14:       return path {Created by backtracing $\pi[v]$, starting with $v_{dst}$}
15:    else
16:       for all $v \in \mathrm{adj}[u]$ do {for each neighbour of $u$}
17:          if $d[v] > d[u] + w(u, v)$ then {Distance is sum of weights}
18:             if $b(u, v) \geq 0$ then
19:                $d[v] \leftarrow d[u] + w(u, v)$ {Since $Q = \mathcal{V}_s$, this changes the queue}
20:                $\pi[v] \leftarrow u$
21:                Enqueue($Q, v$)
```

is our purpose). Lines 16 to 20 perform the relaxation procedure [p9] for each adjacent vertex v of u. Line 18 is not present in the original Dijkstra algorithm, but is necessary to check for enough available bandwidth on the scanned link. Line 21 is an extension to the original Dijkstra algorithm, which is necessary to cope with negative edge weights. It inserts v back into the queue, if it was previously extracted.

A.3 Running Time of Multi-Layer-Dijkstra

The running time of algorithm A.1 is slightly longer then Dijkstra's algorithm due to the extension in line 20, which re-inserts edges into the queue. If we ignore this extension, the running time is:

$$\mathcal{O}(\text{Algorithm A.1}) = \mathcal{O}(|\mathcal{V}_s| \times \mathcal{O}(\textsc{Extract-Min}) + |\mathcal{E}_s| \times \mathcal{O}(\textsc{Insert}) \tag{A.4}$$

In here, $\mathcal{O}(\textsc{Extract-Min})$ is caused by line 9 and $\mathcal{O}(\textsc{Insert})$ is caused by line 21. If he graph is sufficiently sparse ($|\mathcal{E}| < |\mathcal{V}|^2$) [p9], this equation reduces to $\mathcal{O}(|\mathcal{V}_s| \times \log(|\mathcal{V}_s|) + |\mathcal{E}_s| \times \mathcal{O}(1)) = \mathcal{O}(|\mathcal{V}_s| \cdot \log(|\mathcal{V}_s|) + |\mathcal{E}_s|)$.

Equation 7.34 estimates $|\mathcal{E}_s| \approx (|A| + |L|) \times T^{|\mathcal{Y}|}$, and we assume that $\mathcal{O}(|A|) = \mathcal{O}(|N| \times |\mathcal{Y}|)$. Equation 7.29 gives the upper limit $|\mathcal{V}_s| \approx |N| \times |\mathcal{S}| \approx |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}$ with $T = \langle|T(y)|\rangle$ the average number of technologies for each stack.

The running time of algorithm A.1 with constant edge weights $W_e(e)$ is:

$$\begin{aligned}\mathcal{O}(\text{Algorithm A.1}) &= \mathcal{O}(|\mathcal{E}_s| + |\mathcal{V}_s|)\\ &= \mathcal{O}((|A| + |L|) \times T^{|\mathcal{Y}|} + |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}\\ &= \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|} + |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}\\ &\approx \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|})\end{aligned} \tag{A.5}$$

The running time of algorithm A.1 with variable edge weights $W_e(e)$ is:

$$\begin{aligned}\mathcal{O}(\text{Algorithm A.1}) &= \mathcal{O}(|\mathcal{E}_s| + |\mathcal{V}_s| \cdot \log(|\mathcal{V}_s|))\\ &= \mathcal{O}((|A| + |L|) \times T^{|\mathcal{Y}|} + |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}\\ &\quad \times \log(|N| \times |\mathcal{Y}| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}))\\ &= \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|} + |N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}\\ &\quad \times \big(\log(|N|) + \log(|\mathcal{Y}|) + |\mathcal{Y}| \cdot \log(T)\big))\end{aligned} \tag{A.6}$$

A.4 Running Time of Multi-Layer-Breadth-First

The rough estimate of the running time in the previous section does not provide many insights. In this section, we will assume that the segment of a shortest path is also a shortest path. This means we can abort any path if it contains a (node, adaptation stack) tuple that we encountered before. Such algorithm would be comparable to Multi-Layer-Dijkstra, as described above. This allows us to do a more thorough comparison of running times between path finding in G_l and path finding in G_s.

If we can abort a path if the current vertex has been processed with the same adaptations stack s, then each vertex is processed at most $|\mathcal{S}_y|$ times, with $|\mathcal{S}_y|$ the number of possible technology stacks for layer $Y_c(v)$ of vertex v. The worst-case of $|\mathcal{S}_y|$ is $|\mathcal{S}_{|\mathcal{Y}|}|$, or $\prod_{y\in\mathcal{Y}} |T(y)|$ according to equation 7.26.

Recall that average adjacency can be found by dividing the number of edges by the number of vertices in a graph (equation A.1).

According to equation 7.35, the running time of algorithm 7.1 is:

$$\begin{aligned}\mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|Q|) \times \mathcal{O}(\text{loop}) = \\ &= \mathcal{O}(|Q|) \times \big(\mathcal{O}(\text{DEQUEUE}) + \mathcal{O}(|adj|) \times \mathcal{O}(\textsc{Extend-Path})\big) \\ &= \mathcal{O}(|Q|) \times \Big(\mathcal{O}(\text{DEQUEUE}) + \frac{\mathcal{O}(|\mathcal{E}_l|)}{\mathcal{O}(|\mathcal{V}_l|)} \times \mathcal{O}(\textsc{Extend-Path})\Big)\end{aligned} \tag{A.7}$$

With the restriction in place, the queue size is limited to one adaptation stack per node. Since there are at most $|\mathcal{S}|$ adaptations stacks, the upper limit of $|Q|$ is equal to

$$\begin{aligned}\mathcal{O}(|Q|) &= \mathcal{O}(|N| \times |\mathcal{S}|) \\ &\lessapprox \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|})\end{aligned} \tag{A.8}$$

$|Q|$ is equivalent to $|\mathcal{V}_s|$, since $\mathcal{V}_s$ is also determined by the number of adaptation stacks per node. We have seen in equations 7.25 and 7.27 that the *estimate* of $|\mathcal{S}|$ is lower than its upper limit by a factor of $|\mathcal{Y}|$. The *estimate average* of $|Q|$ is equal to

$$\begin{aligned}\mathcal{O}(|Q|) &= \mathcal{O}(|N| \times |\mathcal{S}|) \\ &= \mathcal{O}(|N| \times T^{|\mathcal{Y}|})\end{aligned} \tag{A.9}$$

These results apply to both the Multi-Layer-Breadth-First and Multi-Layer-k-Shortest-Path algorithm with restricted search space.

If we assume $\mathcal{O}(|A|) = \mathcal{O}(|N| \times |\mathcal{Y}|)$, and use equation 7.12 as the estimate of $|\mathcal{V}_l|$ and $|\mathcal{E}_l|$, we can expand equation A.7:

$$\begin{aligned}\mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|Q|) \times \Big(\mathcal{O}(\text{DEQUEUE}) + \frac{\mathcal{O}(|\mathcal{E}_l|)}{\mathcal{O}(|\mathcal{V}_l|)} \times \mathcal{O}(\textsc{Extend-Path})\Big) \\ &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \\ &\quad \Big(\mathcal{O}(\text{DEQUEUE}) + \frac{\mathcal{O}(|A| + |L|)}{\mathcal{O}(|N| \times |Y|)} \times \mathcal{O}(\textsc{Extend-Path})\Big) \\ &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\text{DEQUEUE}) + \\ &\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\textsc{Extend-Path})\end{aligned} \tag{A.10}$$

The only operations in the path extension subroutine (algorithm 7.2) which have a running time larger than $\mathcal{O}(1)$, are the operations on lines 6, 9 that checks for duplicate tuples (vertex, stack) using R, and the operation on line 22 which checks the number of labels per layer $\langle Lb \rangle$. Both operations depend if R and the labels can be sorted. If no sorting is possible, the running time is $\mathcal{O}(\langle R \rangle) = \mathcal{O}(|Q|)$ and $\mathcal{O}(\langle lb \rangle)$ respectively. If sorting is possible, the running time is $\mathcal{O}(\log(|Q|))$ and $\mathcal{O}(\log(\langle lb \rangle))$ respectively.

$$\mathcal{O}(\textsc{Extend-Path}) = \mathcal{O}(\log(|Q|) + \log(\langle lb \rangle)) \tag{A.11}$$

The original breadth first search algorithm can only deal with edge lengths of 1 ($W_e(e) = 1$ for all e). The advantage is that the DEQUEUE operation only takes $\mathcal{O}(1)$, since the queue is sorted in order of path length. If we allow different $W_e(e)$, this function needs to be replaced with an EXTRACT MIN operation. The time complexity of this operation becomes $\mathcal{O}(\log(|Q|))$, provided that the queue Q is sorted using a Fibonacci heap [p9].

$$\mathcal{O}(\textsc{dequeue operation}) = \mathcal{O}(1) \tag{A.12}$$

$$\mathcal{O}(\textsc{extract-min operation}) = \mathcal{O}(\log(|Q|)) \tag{A.13}$$

The queue length $|Q|$ highly depends on how quickly the flooding principle is suppressed by the incompatibility check and duplicate stack check. The worst-case is given in equation A.8.

If we assume $W_e(e) = 1$, $\mathcal{O}(\textsc{Extend-Path}) = \mathcal{O}(\log(|Q|)$ (no labels), the worst-case running time for algorithm 7.1 becomes:

$$\begin{aligned}
\mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\textsc{dequeue}) + \\
&\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\textsc{Extend-Path}) \\
&= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(1) + \\
&\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|Q|)) \\
&\approx \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|})
\end{aligned} \tag{A.14}$$

We can now calculate the worst-case running time of algorithm 7.1 for variable edge weights $W_e(e)$:

$$\begin{aligned}
\mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\textsc{dequeue})+\\
&\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\textsc{Extend-Path})\\
&= \mathcal{O}(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|Q|))+\\
&\quad \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|Q|) + \log(\langle lb \rangle))\\
&\approx \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|Q|) + \log(\langle lb \rangle))\\
&= \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}) \times\\
&\quad \mathcal{O}\big(\big(\log(|N|) + \log(|\mathcal{Y}|) + |\mathcal{Y}| \cdot \log(T)\big) + \log(\langle lb \rangle)\big)
\end{aligned} \tag{A.15}$$

Even so, the *estimated average* running time of algorithm 7.1 is:

$$\begin{aligned}
\mathcal{O}(\text{Algorithm 7.1}) &= \mathcal{O}(|Q|) \times \mathcal{O}(\textsc{dequeue})+\\
&\quad \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\textsc{Extend-Path})\\
&= \mathcal{O}(|Q|) \times \mathcal{O}(\log(Q))+\\
&\quad \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(Q) + \log(\langle lb \rangle))\\
&= \mathcal{O}(|N| \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|N| \times T^{|\mathcal{Y}|}))+\\
&\quad \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|}) \times \mathcal{O}(\log(|N| \times T^{|\mathcal{Y}|}) + \log(\langle lb \rangle))\\
&\approx \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|} \times \big(\log(|N| \times T^{|\mathcal{Y}|}) + \log(\langle lb \rangle)\big))
\end{aligned} \tag{A.16}$$

A.5 Running Time of Multi-Layer-k-Shortest-Path

The running time of Multi-Layer-k-Shortest-Path (algorithm 7.4) is roughly comparable to the running time of Multi-Layer-Breadth-First (algorithm 7.1) as it is basically the same algorithm.

The only difference in the running time is caused by two factors. First, Multi-Layer-k-Shortest-Path reduces the search space by using the estimate path length. This reduces the average running time by aborting path that are unfeasible due to their length. At the same time, this change *increases* the worst-case running time, because in the worst-case, no paths are aborted, and the running of Dijkstra's algorithm adds to the running time.

The second difference stems between the two algorithms from the fact that Multi-Layer-k-Shortest-Path operates on a larger graph than Multi-Layer-Breadth-First. While this may seem worse in terms of number of vertices, the running time is nearly equivalent in terms of devices and technology layers. In fact, any algorithm running on G_s is slightly faster than the equivalent algorithm running on G_l. The reason is that G_s contains more intrinsic information. For example, the loop check (lines 6-9 in algorithm 7.1, line 6 in algorithm 7.5) is more expensive in G_l than in G_s, since it has to search through a list of possible adaptations in G_l while the adaptation function is immediately obvious from the vertex in G_s. So, while the creation of the graph G_s is more computational intensive than the creation of the graph G_l, this drawback is a benefit when running the algorithm. Finally, G_s may be slightly more efficient, since G_s can collapse multiple links on the same edge, while this is not done in G_l. Again, this advantage during the algorithm running time is offset by a disadvantage when generating the graph G_s.

The worst-case running time of Multi-Layer-k-Shortest-Path is:

$$\begin{aligned}\mathcal{O}(\text{Algorithm 7.4}) = \mathcal{O}(|\mathcal{V}_s|) + \mathcal{O}(|\langle Q\rangle|) \times \big(&\mathcal{O}(\textsc{Extract-Min}) + \\ &\mathcal{O}(|adj|) \times (\mathcal{O}(\textsc{Feasible}) + \mathcal{O}(\textsc{Enqueue}))\big)\end{aligned} \tag{A.17}$$

If again, we assume that $\mathcal{O}(|\langle Q\rangle|) = \mathcal{O}(|\mathcal{V}_s|) \leq |N| \times |S|$, and further assume $\mathcal{O}(\textsc{Extract-Min}) = \mathcal{O}(\log(Q))$, $\mathcal{O}(\textsc{Enqueue}) = \mathcal{O}(1)$, and $\mathcal{O}(\textsc{Feasible}) = \mathcal{O}(|\langle p\rangle|) = \mathcal{O}(\log(\mathcal{V}_s))$ then we can specify the worst-case running time for Multi-Layer-k-Shortest-Path.

$$\begin{aligned}\mathcal{O}(\text{Algorithm 7.4}) &= \mathcal{O}(|\mathcal{V}_s|) + \mathcal{O}(|\mathcal{V}_s|) \times \big(\mathcal{O}(\log(\mathcal{V}_s)) + \\ &\quad \frac{\mathcal{O}(\mathcal{E}_s)}{\mathcal{O}(\mathcal{V}_s)} \times (\mathcal{O}(\log(\mathcal{V}_s)) + \mathcal{O}(1))\big) \\ &\approx \mathcal{O}(|\mathcal{V}_s|) \times \mathcal{O}(\log(\mathcal{V}_s)) + \mathcal{O}(|\mathcal{E}_s|) \times \mathcal{O}(\log(\mathcal{V}_s)) \\ &= \mathcal{O}(\big(|\mathcal{V}_s| + |\mathcal{E}_s|\big) \times \log(\mathcal{V}_s)) \\ &= \mathcal{O}(\big(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|} + (|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|}\big) \times \\ &\quad \log(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|})) \\ &\approx \mathcal{O}((|N| \times |\mathcal{Y}| + |L|) \times T^{|\mathcal{Y}|} \times \log(|N| \times |\mathcal{Y}| \times T^{|\mathcal{Y}|}))\end{aligned} \tag{A.18}$$

The *estimated average* running time for Multi-Layer-k-Shortest-Path is

$$\begin{aligned}\mathcal{O}(\text{Algorithm 7.4}) &= \mathcal{O}\big(\big(|\mathcal{V}_s| + |\mathcal{E}_s|\big) \times \log(\mathcal{V}_s)\big)\\ &= \mathcal{O}\big(\big(|N| \times T^{|\mathcal{Y}|} + (|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|}\big) \times \log(|N| \times T^{|\mathcal{Y}|})\big)\\ &\approx \mathcal{O}((|N| + \frac{|L|}{|\mathcal{Y}|}) \times T^{|\mathcal{Y}|} \times \log(|N| \times T^{|\mathcal{Y}|}))\end{aligned} \tag{A.19}$$

Bibliography

Reference names are prefixed according to type. *a* means author (self-references), *o* means other publication (not referred to in this thesis), *p* means (scientific) publications, *s* means standards, *t* mean technical reports and *u* means urls.

B.1 List of Author's Publications

B.1.1 Covered in this Thesis

[a1] F. Dijkstra and C. de Laat. *Optical Exchanges*. In *GRIDNETS conference proceedings* Oct. 2004. URL `http://www.broadnets.org/2004/workshop-papers/Gridnets/DijkstraF.pdf`.

[a2] C. de Laat, F. Dijkstra, and J. Mambretti. *Grid Network Services Infrastructure*. In F. Travostino, J. Mambretti, and G. Karmous-Edwards (editors), *Grid Networks: Enabling Grids with Advanced Communication Technology*, chapter 14, pages 277–292. Wiley, 2006. ISBN 978-0-470-01748-7.

[a3] J. van der Ham, F. Dijkstra, F. Travostino, H. M. Andree, and C. de Laat. *Using RDF to describe networks*. In *Future Generation Computer Systems* 22(8), Oct. 2006. doi:10.1016/j.future.2006.03.022.

[a4] F. Dijkstra, B. van Oudenaarde, B. Andree, L. Gommans, P. Grosso, J. van der Ham, K. Koymans, and C. de Laat. *A Terminology for Control Models at Optical and Internet Exchanges*. In A. K. Bandara and M. Burgess (editors), *Inter-Domain Management – First International Conference on Autonomous Infrastructure, Management and*

Security (AIMS 2007) (LNCS 4543). Oslo, Norway, Jun. 2007. doi:10.1007/978-3-540-72986-0_5. URL http://www.springerlink.com/content/nq2613223k530005/fulltext.pdf.

[a5] F. Dijkstra, B. Andree, J. van der Ham, K. Koymans, and C. de Laat. *Going in Loops to reach your Goal – Multi-Layer Path Finding With NDL*, at *SuperComputing 2007*. Reno, USA, Sep. 2007. URL http://staff.science.uva.nl/~fdijkstr/presentations/Going-in-Loops.pdf.

[a6] P. Grosso, J. van der Ham, F. Dijkstra, and C. de Laat. *Semantic Models for Optical Hybrid Networks – Lightpaths Across Domain Boundaries.* In *Proceedings of eChallenges 2007* The Hague, The Netherlands, Oct. 2007. URL http://www.science.uva.nl/~vdham/research/publications/0710-SemanticModels.pdf.

[a7] F. Dijkstra, B. Andree, K. Koymans, and J. van der Ham. *Introduction to ITU-T Recommendation G.805.* Technical Report UVA-SNE-2007-001, Unversiteit van Amsterdam, Dec. 2007. URL http://www.science.uva.nl/sne/reports/.

[a8] F. Dijkstra, J. van der Ham, P. Grosso, and C. de Laat. *Path Finding Using the Multi-Layer Network Description Language.* In *TERENA Networking Conference* Bruges, Belgium, May 2008. URL http://tnc2008.terena.org/schedule/presentations/show.php?pres_id=26.

[a9] J. van der Ham, F. Dijkstra, P. Grosso, R. van der Pol, A. Toonk, and C. de Laat. *A Distributed Topology Information System for Optical Networks Based on the Semantic Web.* In *Journal of Optical Switching and Networking* 5(2-3), Jun. 2008. doi:10.1016/j.osn.2008.01.006. URL http://staff.science.uva.nl/~vdham/research/publications/0703-ApplicationsOfNDL.pdf.

[a10] F. Dijkstra, B. Andree, K. Koymans, J. van der Ham, P. Grosso, and C. de Laat. *A Multi-Layer Network Model Based on ITU-T G.805.* In *Computer Networks* 52(10), Jul. 2008. doi:10.1016/j.comnet.2008.02.013. URL http://staff.science.uva.nl/~fdijkstr/publications/G805_Multilayer_Model.pdf.

[a11] F. Dijkstra. *Motivation for the Use of Lightpaths.* Technical Report UVA-SNE-2008-03, Universiteit van Amsterdam, Dec. 2008. URL http://www.science.uva.nl/sne/reports/.

[a12] F. Kuipers and F. Dijkstra. *Path Selection in Multi-Layer Networks.* In *Computer Communications* 32(1), Jan. 2009. doi:10.1016/j.comcom.2008.09.026. URL http://staff.science.uva.nl/~fdijkstr/publications/multilayer-pathselection.pdf.

[a13] F. Dijkstra, J. van der Ham, P. Grosso, and C. de Laat. *A Path Finding Implementation for Multi-Layer Networks.* In *Future Generation Computer Systems* 25(2), Feb. 2009. doi:10.1016/j.future.2008.07.002. URL http://staff.science.uva.nl/~fdijkstr/publications/ndl-pathfinding.pdf.

B.1.2 Other Publications

[o1] W. Weiss, J. Vollbrecht, D. Spence, D. Rago, C. de Laat, F. Dijkstra, and L. Gommans. *Framework for Binding Access Control to COPS Provisioning.* Internet-Draft (expired) draft-ietf-rap-access-bind, IETF, Apr. 2002. URL http://tools.ietf.org/html/draft-ietf-rap-access-bind.

[o2] F. Dijkstra and D. Groep. *Building a Multi-Domain Grid.* In *ERCIM news* 59, Oct. 2004. URL http://www.ercim.org/publication/Ercim_News/enw59/dijkstra.html.

[o3] B. van Oudenaarde, Z. Hendrikse, F. Dijkstra, L. Gommans, C. de Laat, and R. Meijer. *Dynamic paths in multi-domain optical networks for grids.* In *Future Generation Computer Systems* 21(4), Apr. 2005. doi:10.1016/j.future.2004.10.008.

[o4] C. Meiroşu, P. Golonka, A. Hirstius, S. Stancu, B. Dobinson, E. Radius, A. Antony, F. Dijkstra, J. Blom, and C. de Laat. *Native 10 Gigabit Ethernet Experiments between Amsterdam and Geneva.* In *Future Generation Computer Systems* 21(4), Apr. 2005. doi:10.1016/j.future.2004.10.003.

[o5] R. L. Grossman, Y. Gu, X. Hong, A. Antony, J. Blom, F. Dijkstra, and C. de Laat. *Teraflows over Gigabit WANs with UDT.* In *Future Generation Computer Systems* 21(4), Apr. 2005. doi:10.1016/j.future.2004.10.007.

[o6] L. Gommans, F. Dijkstra, C. de Laat, A. Taal, A. Wan, B. van Oudenaarde, T. Lavian, I. Monga, and F. Travostino. *Applications drive secure lightpath creation across heterogeneous do-*

mains. In *IEEE Communications Magazine* 44(3), Mar. 2006. doi:10.1109/MCOM.2006.1607872.

[o7] F. Dijkstra, J. van der Ham, and C. de Laat. *Using Zero Configuration Technology for IP addressing in Optical Networks.* In *Future Generation Computer Systems* 22(8), Oct. 2006. doi:10.1016/j.future.2006.03.021.

[o8] L. Xu, F. Dijkstra, D. Marchal, A. Taal, and C. de Laat. *A Declarative Approach to Multi-Layer Path Finding Based on Semantic Network Descriptions.* In *13th Conference on Optical Network Design and Modeling* Braunschweig, Germany, Feb. 2009. URL http://staff.science.uva.nl/~fdijkstr/publications/declarative_path_finding.pdf.

[o9] F. Dijkstra, J. van der Ham, and R. van der Pol. *Network Description Tools and Standards.* In *ERCIM news* 77, Apr. 2009. URL http://ercim-news.ercim.org/content/view/565/777/.

[o10] R. van der Pol and F. Dijkstra. *Network and Capacity Planning in SURFnet6.* In *TERENA Networking Conference* Malaga, Spain, Jun. 2009. URL http://tnc2009.terena.org/schedule/presentations/show.php?pres_id=26.

B.2 References to Scientific Publications

[p1] A.-L. Barabási and R. Albert. *Emergence of scaling in random networks.* In *Science* 286(5439), Oct. 1999. doi:10.1126/science.286.5439.509. URL http://www.sciencemag.org/cgi/reprint/286/5439/509.pdf.

[p2] C. Barrett, R. Jacob, and M. Marathe. *Formal-Language-Constrained Path Problems.* In *SIAM Journal on Computing* 30(3), 2000. ISSN 0097-5397. doi:10.1137/S0097539798337716. URL http://portal.acm.org/citation.cfm?id=586846.586970.

[p3] R. Bellman. *On a Routing Problem.* In *Quarterly of Applied Mathematics* 16(1), 1958. URL https://rand.org/pubs/papers/P1000/.

[p4] K. Blyth and A. Cook. *Designing a GPRS roaming exchange service.* In *IEEE Second International Conference on 3G Mobile Communications Technologies* Mar. 2001.

[p5] X. Cao, V. Anand, and C. Qiao. *Waveband Switching in Optical Networks.* In *IEEE Communications Magazine* Apr. 2003. doi:10.1109/MCOM.2003.1193983.

[p6] B. Chinoy and T. Salo. *Internet Exchanges: Policy-Driven Evolution.* In *Harvard Workshop On Co-Ordination Of The Internet* John F. Kennedy School Of Government, Cambridge, MA, USA, Sep. 1996. URL `http://www.caida.org/publications/papers/1996/nap/nap.html`.

[p7] I. Chlamtac, A. Ganz, and G. Karmi. *Lightpath communications: an approach to high bandwidth optical WANs.* In *IEEE Transactions on Communications* 40(7), Jul. 1992. doi:10.1109/26.153361. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=153361`.

[p8] I. Chlamtac, A. Faragó, and T. Zhang. *Lightpath (Wavelength) Routing in Large WDM Networks.* In *IEEE Journal on Selected Areas in Communications* 14(5), Jun. 1996. doi:19.1109/49.510914.

[p9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, second edition edition, 2001. ISBN 978-0-262-03293-7. URL `http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=8570`.

[p10] T. DeFanti, C. de Laat, J. Mambretti, K. Neggers, and B. St.Arnaud. *TransLight: a global-scale LambdaGrid for e-science.* In *Communications of the ACM* 46(11), Nov. 2003. doi:10.1145/948383.948407.

[p11] E. W. Dijkstra. *A Note on Two Problems in Connexion with Graphs.* In *Numerische Mathematik* 1, 1959.

[p12] L. R. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, 1962. ISBN 978-0-691079622.

[p13] P. Grosso, P. de Boer, and L. Winkler. *The network infrastructure at iGrid2005: Lambda networking in action.* In *Future Generation Computer Systems* 22(8), Oct. 2006. doi:10.1016/j.future.2006.03.013. URL `http://linkinghub.elsevier.com/retrieve/pii/S0167739X06000343`.

[p14] J. van der Ham, P. Grosso, and C. de Laat. *Semantics for Hybrid Networks Using the Network Description Language,* at *SuperComputing 2006.* Tampa, USA, Mar. 2006. URL

http://staff.science.uva.nl/~vdham/research/publications/0603-NetworkDescriptionLanguage.pdf.

[p15] J. van der Ham, P. Grosso, R. van der Pol, A. Toonk, and C. de Laat. *Using the Network Description Language in Optical Networks*. In *Tenth IFIP/IEEE Symposium on Integrated Network Management* Munich, Germany, May 2007. URL http://staff.science.uva.nl/~vdham/research/publications/0606-UsingNDLInOpticalNetworks.pdf.

[p16] J. van der Ham. *A Semantic Model for Complex Computer Networks: The Network Description Language*. Ph.D. thesis, Universiteit van Amsterdam, Kruislaan 403, 2009.

[p17] P. Holme and B. J. Kim. *Growing scale-free networks with tunable clustering*. In *Physical Review E* 65(026107), Jan. 2002. doi:10.1103/PhysRevE.65.026107. URL http://arxiv.org/abs/cond-mat/0110452.

[p18] G. Huston. *Interconnection, Peering, and Settlements*. In *Proceedings of Inet'99* Jun. 1999. URL http://www.isoc.org/inet99/proceedings/1e/1e_1.htm.

[p19] F. A. Kuipers. *Quality of Service Routing in the Internet: Theory, Complexity and Algorithms*. Ph.D. thesis, Delft University, Delft, The Netherlands, Sep. 2004. URL http://www.nas.its.tudelft.nl/people/Fernando/papers/PhDthesiskuipers.pdf.

[p20] F. A. Kuipers, T. Korkmaz, M. Krunz, and P. van Mieghem. *Performance evaluation of constraint-based path selection algorithms*. In *IEEE Network* 18(5), 2004. doi:10.1109/MNET.2004.1337731. URL http://ieeexplore.ieee.org/iel5/65/29508/01337731.pdf.

[p21] J. H. Laarhuis. *Multichannel Interconnection in All-Optical Networks*. Ph.D. thesis, Centre for Telematics and Information Technology, Sep. 1995.

[p22] C. de Laat and J. Blom. *User-Level Performance Monitoring Programme*. In *TERENA Network Conference 2000* Lisbon, Portugal, May 2000. URL http://www.terena.org/events/archive/tnc2000/proceedings/8B/8b4.ppt.

[p23] C. de Laat, E. Radius, and S. Wallace. *The Rationale of the Current Optical Networking Initiatives*. In *Future Generation Computer*

Systems 19(6), Aug. 2003. doi:10.1016/S0167-739X(03)00077-3. URL http://www.sciencedirect.com/science/article/B6V06-48V83MF-5/2/d8aac1d72ec497da8c83c4a07fdfec0c.

[p24] T. Lehman, J. Sobieski, and B. Jabbari. *DRAGON: A Framework for Service Provisioning in Heterogeneous Grid Networks.* In *IEEE Communications Magazine* 44(3), Mar. 2006. URL http://ieeexplore.ieee.org/iel5/35/33764/01607870.pdf.

[p25] O. H. Martin. *The ongoing evolution from packet based networks to hybrid networks in research & education networks.* In *XXth Anniversary International Symposium on Nuclear Electronics & Computing (NEC)* Sep. 2005. URL http://www.jinr.ru/NEC/NEC-2005/proceeding2005/Martin.doc.

[p26] P. van Mieghem and F. A. Kuipers. *On the complexity of QoS routing.* In *Computer Communications* 26(4), 2003. doi:10.1016/S0140-3664(02)00156-1.

[p27] I. Nakagawa, H. Esaki, Y. Kikuchi, and K. Nagami. *Design of Next Generation IX Using MPLS Technology.* In *IPSJ Journal* Nov. 2002.

[p28] W. Norton. *Internet Service Providers and Peering.* In *Proceedings of NANOG 19* May 2001.

[p29] R. van der Pol, A. Toonk, and C. de Laat. *Hybrid Local Area Networks.* In *TERENA Network Conference 2006* May 2006. URL http://www.terena.nl/events/tnc2006/core/getfile.php?file_id=749.

[p30] R. van der Pol and A. Toonk. *Lightpath Planning and Monitoring in SURFnet6 and NetherLight.* In *TERENA Network Conference 2007* Lynby, Denmark, May 2007. URL https://noc.sara.nl/nrg/publications/LightpathPlanningAndMonitoring.pdf.

[p31] R. van der Pol and A. Toonk. *Lightpath Planning and Monitoring.* In *eChallenges Conference 2007* The Hague, The NetherlandsH, Oct. 2007. URL https://noc.sara.nl/nrg/publications/E-Challenges-v1.4.pdf.

[p32] C. Qiao and M. Yoo. *Optical burst switching (OBS) – a new paradigm for an Optical Internet.* In *Journal of High Speed Networks* 8(1), Mar. 1999. URL http://iospress.metapress.com/link.asp?id=0r3ygmf09b6vev8k.

[p33] N. Roosen. *Fault Detection and Isolation on Transport Networks*. Master's thesis, University of Amsterdam, Sep. 2008. URL http://www.science.uva.nl/research/sne/files/ntroosen-lmon.pdf.

[p34] A. Rushton, R. Spencer, M. Strong, R. Campbell, S. Casey, R. Fender, M. Garrett, J. Miller-Jones, G. Pooley, C. Reynolds, A. Szomoru, V. Tudose, and Z. Paragi. *First e-VLBI observations of GRS 1915+105*. In *Monthly Notices of the Royal Astronomical Society: Letters* 374, 2007. doi:10.1111/j.1745-3933.2006.00262.x. URL http://arxiv.org/abs/astro-ph/0611049.

[p35] A. A. M. Saleh and J. M. Simmons. *Evolution Toward the Next-Generation Core Optical Network*. In *Journal of Lightwave Technology* 24(9), Sep. 2006. doi:10.1109/JLT.2006.880608.

[p36] L. L. Smarr, A. A. Chien, T. A. DeFanti, J. Leigh, and P. M. Papadopoulos. *The OptIPuter*. In *Communications of the ACM* 46(11), Nov. 2006. doi:10.1145/948383.948410. URL http://doi.acm.org/10.1145/948383.948410.

[p37] L. L. Smarr. *Riding the Light Towards New Science*. In *Nature Photonics* 1(3), Mar. 2007. doi:10.1038/nphoton.2007.6. URL http://www.nature.com/nphoton/journal/v1/n3/full/nphoton.2007.6.html.

[p38] A. Szomoru, A. Biggs, M. Garrett, H. J. van Langevelde, F. Olnon, Z. Paragi, S. Parsley, S. Pogrebenko, and C. Reynolds. *From truck to optical fibre: the coming-of-age of eVLBI*. In R. Bachiller, F. Colomer, J. Desmurs, and P. de Vicente (editors), *Proceedings of the 7th European VLBI Network Symposium* Joint Institute for VLBI in Europe (JIVE), Toledo, Spain, Oct. 2004. URL http://www.oan.es/evn2004/WebPage/ASzomoru.pdf.

[p39] S. Tomic and A. Jukan. *GMPLS-Based Exchange Points: Architecture and Functionalilty*. In K. Sivalingam and S. Subramaniam (editors), *Emerging Optical Network Technologies Architectures, Protocols and Performance*, chapter 8. Springer, Oct. 2004. ISBN 978-0-387-22582-X.

[p40] P. Torab, B. Jabbari, Q. Xu, S. Gong, X. Yang, T. Lehman, C. Tracy, and J. Sobieski. *On Cooperative Inter-Domain Path Computation*. In *Proceedings of the 11th IEEE Symposium on Computers and Communications* IEEE Computer Society, Sardinia, Italy, Jun. 2006. doi:10.1109/ISCC.2006.109. URL http://cnl.gmu.edu/bjabbari/payam.pdf.

[p41] F. Travostino. *Using the Semantic Web to Automate the Operation of a Hybrid Internetwork*. In *2nd International Conference on Broadband Networks* 2. IEEE, Oct. 2005. doi:10.1109/ICBN.2005.1589769.

[p42] V. Tudose, R. Fender, M. Garrett, J. Miller-Jones, Z. Paragi, R. Spencer, G. Pooley, M. van der Klis, and A. Szomoru. *First e-VLBI observations of Cygnus X-3*. In *Monthly Notices of the Royal Astronomical Society: Letters* 375, Feb. 2007. doi:10.1111/j.1745-3933.2006.00264.x. URL http://arxiv.org/abs/astro-ph/0611054.

B.3 Technical References

B.3.1 Normative References (Standards)

[s1] *Synchronous Optical Network (SONET) - Basic Description including Multiplex Structure, Rates, and Formats*. Standard T1.105, American National Standards Institute (ANSI), 2001. URL http://webstore.ansi.org/RecordDetail.aspx?sku=T1.105-2001.

[s2] *CIM Network*. Standard, Distributed Management Task Force (DMTF), Aug. 2007. URL http://www.dmtf.org/standards/cim/cim_schema_v216/CIM_Network.pdf.

[s3] *Virtual Bridged Local Area Networks*. IEEE Standard 802.1Q, IEEE, Dec. 2005. URL http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf.

[s4] *Provider Bridges*. IEEE Standard 802.1ad, IEEE, May 2006. URL http://www.ieee802.org/1/pages/802.1ad.html.

[s5] *Provider Backbone Bridge Traffic Engineering*. IEEE Draft 802.1Qay, IEEE, 2007. URL http://www.ieee802.org/1/pages/802.1ay.html.

[s6] *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access method and physical layer specifications*. IEEE Standard 802.3, IEEE, Dec. 2005. URL http://standards.ieee.org/getieee802/download/802.3-2005_section4.pdf.

[s7] D. Oran. *OSI IS-IS Intra-domain Routing Protocol*. RFC 1142 (Informational), Feb. 1990. URL http://www.ietf.org/rfc/rfc1142.txt.

[s8] W. Simpson. *PPP in HDLC-like Framing*. RFC 1662 (Standard), Jul. 1994. URL http://www.ietf.org/rfc/rfc1662.txt.

[s9] J. Chapman, D. Coli, A. Harvey, B. Jensen, and K. Rowett. *PPP Network Control Protocol for LAN Extension*. RFC 1841 (Informational), Sep. 1995. URL http://www.ietf.org/rfc/rfc1841.txt.

[s10] J. Moy. *OSPF Version 2*. RFC 2328 (Standard), Apr. 1998. URL http://www.ietf.org/rfc/rfc2328.txt.

[s11] G. Malkin. *RIP Version 2*. RFC 2453 (Standard), Nov. 1998. URL http://www.ietf.org/rfc/rfc2453.txt.

[s12] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard), Dec. 1998. URL http://www.ietf.org/rfc/rfc2460.txt.

[s13] A. Malis and W. Simpson. *PPP over SONET/SDH*. RFC 2615 (Proposed Standard), Jun. 1999. URL http://www.ietf.org/rfc/rfc2615.txt.

[s14] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence. *Generic AAA Architecture*. RFC 2903 (Experimental), Aug. 2000. URL http://www.ietf.org/rfc/rfc2903.txt.

[s15] E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol Label Switching Architecture*. RFC 3031 (Proposed Standard), Jan. 2001. URL http://www.ietf.org/rfc/rfc3031.txt.

[s16] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. *Terminology for Policy-Based Management*. RFC 3198 (Informational), Nov. 2001. URL http://www.ietf.org/rfc/rfc3198.txt.

[s17] A. Pras and J. Schoenwaelder. *On the Difference between Information Models and Data Models*. RFC 3444 (Informational), Jan. 2003. URL http://www.ietf.org/rfc/rfc3444.txt.

[s18] L. Berger. *Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions*. RFC 3473 (Proposed Standard), Jan. 2003. URL http://www.ietf.org/rfc/rfc3473.txt.

[s19] M. Higashiyama, F. Baker, and T. Liao. *Point-to-Point Protocol (PPP) Bridging Control Protocol (BCP)*. RFC 3518 (Proposed Standard), Apr. 2003. URL http://www.ietf.org/rfc/rfc3518.txt.

[s20] E. Mannie. *Generalized Multi-Protocol Label Switching (GMPLS) Architecture*. RFC 3945 (Proposed Standard), Oct. 2004. URL http://www.ietf.org/rfc/rfc3945.txt.

[s21] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986 (Standard), Jan. 2005. URL http://www.ietf.org/rfc/rfc3986.txt.

[s22] K. Kompella, Y. Rekhter, and Ed. *Routing Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS)*. RFC 4202 (Proposed Standard), Oct. 2005. URL http://www.ietf.org/rfc/rfc4202.txt.

[s23] K. Kompella and Y. Rekhter. *OSPF Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS)*. RFC 4203 (Proposed Standard), Oct. 2005. URL http://www.ietf.org/rfc/rfc4203.txt.

[s24] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271 (Draft Standard), Jan. 2006. URL http://www.ietf.org/rfc/rfc4271.txt.

[s25] D. Meyer and K. Patel. *BGP-4 Protocol Analysis*. RFC 4274 (Informational), Jan. 2006. URL http://www.ietf.org/rfc/rfc4274.txt.

[s26] E. Mannie and D. Papadimitriou. *Generalized Multi-Protocol Label Switching (GMPLS) Extensions for Synchronous Optical Network (SONET) and Synchronous Digital Hierarchy (SDH) Control*. RFC 4606 (Proposed Standard), Aug. 2006. URL http://www.ietf.org/rfc/rfc4606.txt.

[s27] A. Farrel, J.-P. Vasseur, and J. Ash. *A Path Computation Element (PCE)-Based Architecture*. RFC 4655 (Informational), Aug. 2006. URL http://www.ietf.org/rfc/rfc4655.txt.

[s28] A. Farrel, J.-P. Vasseur, and A. Ayyangar. *A Framework for Inter-Domain Multiprotocol Label Switching Traffic Engineering*. RFC 4726 (Informational), Nov. 2006. URL http://www.ietf.org/rfc/rfc4726.txt.

[s29] Q. Vohra and E. Chen. *BGP Support for Four-octet AS Number Space*. RFC 4893 (Proposed Standard), May 2007. URL http://www.ietf.org/rfc/rfc4893.txt.

[s30] J. Vasseur, A. Ayyangar, and R. Zhang. *A Per-Domain Path Computation Method for Establishing Inter-Domain Traffic Engineering (TE) Label Switched Paths (LSPs)*. RFC 5152 (Proposed Standard), Feb. 2008. URL http://www.ietf.org/rfc/rfc5152.txt.

[s31] K. Shiomoto, D. Papadimitriou, J. L. Roux, M. Vigoureux, and D. Brungard. *Requirements for GMPLS-Based Multi-Region and Multi-Layer Networks (MRN/MLN)*. RFC 5212 (Informational), Jul. 2008. URL http://www.ietf.org/rfc/rfc5212.txt.

[s32] J. L. Roux and D. Papadimitriou. *Evaluation of Existing GMPLS Protocols against Multi-Layer and Multi-Region Networks (MLN/MRN)*. RFC 5339 (Informational), Sep. 2008. URL http://www.ietf.org/rfc/rfc5339.txt.

[s33] D. Papadimitriou, M. Vigoureux, K. Shiomoto, D. Brungard, and J.-L. L. Roux. *Generalized Multi-Protocol Label Switching (GMPLS) Protocol Extensions for Multi-Layer and Multi-Region Networks (MRN/MLN)*. Internet-Draft (Work In Progress) draft-ietf-ccamp-gmpls-mln-reqs, Internet Engineering Task Force, Apr. 2009. URL http://tools.ietf.org/html/draft-ietf-ccamp-gmpls-mln-extensions.

[s34] M. Blanchet, F. Parent, and B. St.Arnaud. *Optical BGP (OBGP): InterAS lightpath provisioning*. Internet-Draft (expired) draft-parent-obgp, IETF, Mar. 2001. URL http://tools.ietf.org/html/draft-parent-obgp.

[s35] T. Otani, H. Guo, K. Miyazaki, and D. Caviglia. *Generalized Labels of Lambda-Switching Capable Label Switching Routers (LSR)*. Internet-Draft (Work In Progress) draft-otani-ccamp-gmpls-lambda-labels, Internet Engineering Task Force, Feb. 2008. URL http://tools.ietf.org/html/draft-otani-ccamp-gmpls-lambda-labels.

[s36] *Optical interfaces for multichannel systems with optical amplifiers*. Recommendation ITU-T G.692, International Telecommunication Union (ITU), Oct. 1998. URL http://www.itu.int/rec/T-REC-G.692/.

[s37] *Spectral grids for WDM applications: DWDM frequency grid*. Recommendation ITU-T G.694.1, International Telecommunication Union (ITU), Jun. 2002. URL http://www.itu.int/rec/T-REC-G.694.1/.

[s38] *Spectral grids for WDM applications: CWDM wavelength grid.* Recommendation ITU-T G.694.2, International Telecommunication Union (ITU), Dec. 2003. URL http://www.itu.int/rec/T-REC-G.694.2/.

[s39] *Interfaces for the Optical Transport Network (OTN).* Recommendation ITU-T G.709 / ITU-T Y.1331, International Telecommunication Union (ITU), Mar. 2003. URL http://www.itu.int/rec/T-REC-G.709/.

[s40] *Characteristics of synchronous digital hierarchy (SDH) equipment functional blocks.* Recommendation ITU-T G.783, International Telecommunication Union (ITU), Feb. 2004. URL http://www.itu.int/rec/T-REC-G.783/.

[s41] *Unified functional architecture of transport networks.* Recommendation ITU-T G.800, International Telecommunication Union (ITU), Sep. 2007. URL http://www.itu.int/rec/T-REC-G.800/.

[s42] *Generic functional architecture of transport networks.* Recommendation ITU-T G.805, International Telecommunication Union (ITU), Mar. 2000. URL http://www.itu.int/rec/T-REC-G.805/.

[s43] *Functional architecture of connectionless layer networks.* Recommendation ITU-T G.809, International Telecommunication Union (ITU), Mar. 2003. URL http://www.itu.int/rec/T-REC-G.809/.

[s44] *Architecture of Optical Transport Networks.* Recommendation ITU-T G.872, International Telecommunication Union (ITU), Nov. 2001. URL http://www.itu.int/rec/T-REC-G.872/.

[s45] *Generic framing procedure (GFP).* Recommendation ITU-T G.7041 / Y.1303, International Telecommunication Union (ITU), Dec. 2003. URL http://www.itu.int/rec/T-REC-G.7041/.

[s46] *Link capacity adjustment scheme (LCAS) for virtual concatenated signals.* Recommendation ITU-T G.7042/Y.1305, International Telecommunication Union (ITU), Mar. 2006. URL http://www.itu.int/rec/T-REC-G.7042/.

[s47] *Virtual concatenation of plesiochronous digital hierarchy (PDH) signals.* Recommendation ITU-T G.7043 / Y.1343, International Telecommunication Union (ITU), Jul. 2004. URL http://www.itu.int/rec/T-REC-G.7043/.

[s48] *B-ISDN asynchronous transfer mode functional characteristics.* Recommendation ITU-T I.150, International Telecommunication Union (ITU), Feb. 1999. URL http://www.itu.int/rec/T-REC-I.150/.

[s49] *B-ISDN service aspects.* Recommendation ITU-T I.211, International Telecommunication Union (ITU), Mar. 1993. URL http://www.itu.int/rec/T-REC-I.211/.

[s50] *Considerations for a telecommunications management network.* Recommendation ITU-T M.3013, International Telecommunication Union (ITU), Feb. 2000. URL http://www.itu.int/rec/T-REC-M.3013/.

[s51] *Introduction to CCITT Signalling System No. 7.* Recommendation ITU-T Q.700, International Telecommunication Union (ITU), Mar. 1993. URL http://www.itu.int/rec/T-REC-Q.700/.

[s52] *Signalling network functions and messages.* Recommendation ITU-T Q.704, International Telecommunication Union (ITU), Jul. 1996. URL http://www.itu.int/rec/T-REC-Q.704/.

[s53] *Open System Interconnection Model (OSI model).* Recommendation ISO 7498 / ITU-T X.200, International Standardisation Organisation (ISO), Jul. 1994. URL http://www.itu.int/rec/T-REC-X.200/.

[s54] T. Ferrari, J. Austin, P. Clarke, M. Fletcher, M. Gaynor, R. Hughes-Jones, T. Jackson, G. Karmous-Edwards, P. Kunszt, M. J. Leese, J. Leigh, P. D. Mealor, I. Monga, V. Sander, R. Spencer, M. Strong, and P. Tomsu. *Grid Network Services Use Cases from the e-Science Community.* OGF Grid Final Documents 122, Open Grid Forum, Dec. 2007. URL http://www.gridforum.org/documents/GFD.122.pdf.

[s55] D. Beckett. *RDF/XML Syntax Specification.* Recommendation, W3C, Feb. 2004. URL http://www.w3.org/TR/rdf-syntax-grammar/.

[s56] E. Prud'hommeaux and A. Seaborne. *SPARQL Query Language for RDF.* Proposed recommendation, W3C, Nov. 2007. URL http://www.w3.org/TR/rdf-sparql-query/.

[s57] *Cisco EoS LEX.* In *Cisco ONS 15454 and Cisco ONS 15454 SDH Ethernet Card Software Feature and Configuration Guide, Release 8.0*, pages 20–4–20–9. Cisco, Oct. 2007. URL http://www.cisco.com/en/US/docs/optical/15000r8_0/ethernet/454/guide/r8pos.html#wp1077827.

[s58] *Dublin Core Metadata Initiative.* URL http://www.dublincore.org/.

B.3.2 Informative References (Technical Reports)

[t1] *Services provided by the AMS-IX.* URL http://www.ams-ix.net/services/.

[t2] *Architecture for starting an IXP.* URL https://www.euro-ix.net/ixp/startingixp/infra/architecture.php.

[t3] *Network Aware Resource Broker (NARB) and Resource Computation Element (RCE) Architecture.* Technical report, University of Southern California and Information Sciences Institute, Aug. 2007. URL http://dragon.east.isi.edu/twiki/pub/Main/NARB/narb-rce-architecture-v2.0.pdf.

[t4] *SURFnet6 lightpaths mark start of new Internet era.* Press Release, Jan. 2006. URL http://www.surfnet.nl/info/en/artikel_content.jsp?objectnumber=107197.

[t5] T. Berners-Lee. *Notation 3 – an RDF language for the Semantic Web.* Technical report, W3C, 1998. URL http://www.w3.org/DesignIssues/Notation3.

[t6] E.-J. Bos. *Issue Analysis Hybrid Networks.* Technical report, GLIF Technical Issues Working Group, Aug. 2006. URL http://www.glif.is/working-groups/tech/hybrid-network-issues.pdf.

[t7] J. N. Chiappa. *Endpoints and Endpoint Names: A Proposed Enhancement to the Internet Architecture.* Internet-draft (expired), 1999. URL http://ana.lcs.mit.edu/~jnc/tech/endpoints.txt.

[t8] F. Dijkstra. *Terminology discussion*, Sep. 2005. URL http://www.glif.is/list-archives/tech/msg00019.html. Terminology discussion in the GLIF community.

[t9] A. Escolano, A. Mackarel, D. Regvart, V. Reijs, G. Roberts, and H. Popovski. *Report on Testing of Technology Stitching.* Deliverable DJ3.5.3, GÉANT, May 2007. URL http://www.geant2.net/upload/pdf/GN2-07-066v5-DJ3-5-3-Report_on_Testing_of_Technology_Stitching.pdf.

[t10] A. Harrison. *Definition of economic assets*, Jan. 2006. URL http://unstats.un.org/UNSD/nationalaccount/AEG/papers/m4EconAssets.pdf. The terms 'economic ownership' and 'legal

ownership' are not defined in the 1993 System of National Accounts by the UN, EC, IMU, OESO and world bank. However, economic experts often clarify these terms.

[t11] R. Hatem, A. Giesbert, and E.-J. Bos. *The ordering and fault resolution process for multi-domain Lightpaths across hybrid networks.* Draft, GLIF Technical Issues Working Group, Jul. 2006. URL http://www.glif.is/working-groups/tech/fault-resolution-0.9.pdf.

[t12] T. Lehman. *Dynamic Services Control Plane Overview and Status*, at *7th Annual Global LambdaGrid Workshop.* Feb. 2007. URL http://www.glif.is/meetings/2007/winter/controlplane/lehman-dynamic-services.pdf.

[t13] J. Mambretti. *International High Performance Digital Media With Dynamic Optical Multicast.* In *7th Annual Global LambdaGrid Workshop* GLIF, Prague, Czech Republic, Sep. 2007. URL http://www.glif.is/meetings/2007/controlplane/mambretti-hpdm.pdf.

[t14] J. F. Shoch. *A note on Inter-Network Naming, Addressing, and Routing.* Internet Experiment Note 19, Xerox Palo Alto Research Center, Jan. 1978. URL http://ana-3.lcs.mit.edu/~jnc/tech/ien/ien19.txt.

[t15] D. Simeonidou, R. Nejabati, B. St.Arnaud, M. Beck, P. Clarke, D. B. Hoang, D. Hutchison, G. Karmous-Edwards, T. Lavian, J. Leigh, J. Mambretti, V. Sander, J. Strand, and F. Travostino. *Optical Network Infrastructure for Grid.* draft (unpublished) draft-ggf-ghpn-opticalnets, Open Grid Forum, May 2004. URL http://forge.gridforum.org/sf/go/doc10908.

[t16] B. St.Arnaud, J. Wu, and B. Kalali. *Customer Controlled and Managed Optical Networks.* Technical report, CANARIE, Jan. 2003. URL http://www.canarie.ca/canet4/library/c4design/customer_controlled.pdf.

[t17] M. Wolski, S. Osinski, P. Gruszczynski, M. Labedzki, A. Patil, and I. Thomson. *common Network Information Service Schema Specification.* Deliverable DS3.13.1, GÉANT, Apr. 2007. URL http://www.geant2.net/upload/pdf/GN2-07-045v4-DS3-13-1_common_Network_Information_Service_Schema_Specification.pdf.

B.4 Miscellaneous References

[u1] F. Dijkstra and J. van der Ham. *Python NDL Toolkit homepage*. URL http://ndl.uva.netherlight.nl/trac/ndl/.

[u2] P. Grosso and M. Swany. *Network Markup Language Working Group*, Mar. 2007. URL http://forge.gridforum.org/sf/projects/nml-wg.

[u3] J. van der Ham and F. Dijkstra. *Network Description Language Homepage*. URL http://www.science.uva.nl/research/sne/ndl/.

[u4] R. Patterson and M. D. Brown. *GLIF world map*, May 2008. URL http://www.glif.is/publications/#info. Visualization by Robert Patterson, the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign. Data compilation by Maxine Brown, University of Illinois at Chicago. Earth texture provided by NASA, http://visibleearth.nasa.gov.

[u5] R. van der Pol. *Spotlight – NetherLight lightpath status*. URL http://noc.netherlight.net:8080/spotlight/.

[u6] B. St.Arnaud. *CAnet4 network*. URL http://www.canarie.ca/canet4/.

[u7] A. Toonk and R. van der Pol. *TL1 toolkit*. URL https://noc.sara.nl/nrg/TL1-Toolkit/.

[u8] *Global Lambda Integrated Facility (GLIF)*. URL http://www.glif.is/.

[u9] *Graphviz – Graph Visualization Software*. URL http://www.graphviz.org/.

[u10] *National LambdaRail network*. URL http://www.nlr.net/.

[u11] *NetherLight*. URL http://www.netherlight.net/.

[u12] *Resource Description Framework (RDF)*. URL http://www.w3.org/RDF/.

[u13] *SARA Computing and Networking Services*. URL http://www.sara.nl/.

[u14] *The Semantic Web*. URL http://www.w3.org/2001/sw/.

[u15] *StarLight*. URL http://www.startap.net/starlight/.

Samenvatting

In relatief korte tijd is het gebruik van computernetwerken enorm toegenomen en inmiddels zijn ze niet meer weg te denken uit de samenleving. Computernetwerken worden gebruikt voor surfen, e-mail en het doen van betalingen. De capaciteit van de netwerkverbindingen is zozeer toegenomen dat het netwerk tegenwoordig ook gebruikt wordt voor het verzenden van gigantische hoeveelheden data die tot voor kort nog op tapes of disks per koerier verstuurd werden. Datastromen in de kernfysica, radioastronomie en sinds kort ook het versturen van bioscoopfilms op uitzonderlijk hoge kwaliteit zijn hiervan sprekende voorbeelden.

Voor het versturen van grote hoeveelheden gegevens tussen een klein aantal locaties is het Internet niet altijd even geschikt. In sommige gevallen is het beter om een aparte verbinding aan te leggen voor één specifieke toepassing. Een aparte verbinding is beter omdat het goedkoper kan zijn of omdat de verkeersstromen zo groot zijn dat ze het reguliere Internetverkeer zouden verstoren. Naast de eerder genoemde uitwisseling van meetgegevens en films is het versturen van gegevens tussen twee vestigingen van één organisatie een mogelijke toepassing. In de meeste andere gevallen is het gebruik van het reguliere Internet beter.

Sinds ongeveer 2005 zijn met name onderzoeksnetwerken zoals CAnet in Canada en SURFnet in Nederland begonnen met het aanbieden van zogenaamde lichtpaden naast het aanbieden van regulier Internet. Lichtpaden zijn aparte verbindingen tussen twee plaatsen, in tegenstelling tot het Internet waarover gegevens naar alle plaatsen ter wereld gestuurd kunnen worden. Lichtpaden worden zo genoemd omdat ze vaak gebruik maken van glasvezelnetwerken. Netwerken waarbij over dezelfde infrastructuur zowel Internet als lichtpaden aanbieden worden *hybride netwerken* genoemd.

Het aanbieden van aparte verbindingen wordt in feite al heel lang gedaan door telecomaanbieders, al wordt de naam *lichtpad* pas sinds kort gebezigd. Hoewel lichtpaden qua technologie en capaciteit veel weg hebben van huurlijnen hebben ze in dynamiek meer weg van telefoonverbindingen. Idealiter kunnen ze automatisch opgezet worden, dus zonder tussenkomst van een netwerkbeheerder.

Een verbinding kan lopen via meerdere beheerdomeinen. Een typische netwerkverbinding tussen twee universiteiten loopt bijvoorbeeld eerst via een campus netwerk, dan een nationaal netwerk en vervolgens via een internationale koppeling terug naar een ander nationaal netwerk, campus netwerk en uiteindelijk naar een netwerk binnen een gebouw. Al deze verschillende netwerken worden beheerd door andere personen en instellingen.

Elk netwerk zal op een ander tijdstip zijn aangelegd en elke netwerkbeheerder zal een andere keuze hebben gemaakt in de technologie die gebruikt is in het netwerk, ook wel aangeduid als verschillende lagen in het netwerk. Zo kan het ene netwerk schakelen op de Ethernetlaag, een ander op de SONET-laag en kan een derde netwerk verschillende kleuren licht over glasvezels schakelen.

Het blijkt zeer relevant te zijn dat lichtpaden over verschillende netwerken lopen die elk kunnen schakelen op een andere netwerklaag. Verschillen in technologie tussen netwerken kunnen namelijk leiden tot mogelijke incompatibiliteiten. Deze incompatibiliteiten maken het vinden van paden in hybride netwerken aantoonbaar complexer dan het vinden van paden binnen netwerken die alle van dezelfde technologie gebruik maken, zoals het reguliere Internet. Het blijkt namelijk dat paden in hybride netwerken beperkingen hebben die afhankelijk zijn van keuzes elders in het pad. Dit in tegenstelling tot netwerkverbindingen met slechts één technologie waarbij de beperkingen onafhankelijk zijn van het eerder gekozen pad.

In dit proefschrift wordt aangetoond dat paden die door meerdere technologieën gaan, de zogenaamde meerlaagsnetwerkverbindingen, in een rondje kunnen lopen. Kortom, het is mogelijk dat een kortste pad toch twee keer hetzelfde stukje weg aflegt. Tevens blijkt dat een onderdeel van een kortste pad op zichzelf geen kortste pad hoeft te zijn.

Het grootste deel van dit proefschrift wordt besteed aan het vinden van een model dat meerlaagsnetwerken, zoals onder andere hybride netwerken, formeel kan beschrijven. Dit model is met opzet technologieonafhankelijk gemaakt, zodat het model en –belangrijker– de algoritmes om paden te vinden niet aangepast hoeven te worden als er later nieuwe technologieën gebruikt worden.

Naast het model worden twee bijna gelijkwaardige algoritmes gepresenteerd die paden kunnen vinden in meerlaagsnetwerken. Een van deze algoritmes is tevens geïmplementeerd en dit proefschrift sluit af met het eerste gebruik hiervan.

Abstract

In only a few decades the use of computer networks has dramatically increased. Today, networks are ubiquitous in society: they are used for surfing, email and financial transactions. The capacity of the network has increased so much, that it is now possible to transfer massive data sets that recently were shipped on tape or disk by a courier. Vast data streams in nuclear physics, radio astronomy and recent transmission of movies in exceptional high quality are only a few of the prime examples of these transfers.

The Internet is not always suitable to transmit large amounts of data between a limited number of locations. In some cases it is better to create a dedicated network connection for a specific application. A dedicated connection is better if it is cheaper or if the data streams are so large that they would disrupt the regular Internet traffic. Besides the already mentioned examples of transfer of scientific data and movies, the data exchange between two sites of the same company can be one of those applications. In most other cases the regular Internet is the better choice.

Since 2005 national research networks such as CAnet in Canada and SURFnet in the Netherlands have started offering lightpaths to their customers, besides their regular Internet connectivity service. These are dedicated connections between only two locations, as opposed to the regular Internet, where every host can connect to all places in the world. The name lightpath comes from the optical fibre networks that often provide these connections. A network that provides both lightpaths as well as regular Internet connectivity over the same physical infrastructure it is called a *hybrid network*.

Telecom providers have offered dedicated network connections since a long time, although the term *lightpath* is relatively new. While lightpaths mimic leased lines in technology and capacity, their dynamics resembles a telephone connection. Ideally, lightpaths can be provisioned automatically, without intervention from a human network operator.

Lightpaths can span multiple administrative domains. A typical network connection between two universities firsts crosses a campus network, then a national research network, then goes through an international peering to another research network, then another campus network and finally to an internal network within a building. Different persons and organisations administrate all these networks.

Each of these networks has been built and deployed at different times, and each network designer will have made different choices from the list of available technologies. Each network consists of different layers, and one network may be able to switch at the Ethernet layer, another at the SONET layer and a third may be able to switch different colours of light over a fibre.

It turns out to be very relevant that lightpaths cross multiple networks that are capable of switching at a different network layer. Different choices in technology for each network lead to potential incompatibilities. This thesis shows that these incompatibilities make the problem of finding a shortest path through the networks significantly more complex than the problem of finding paths through networks where the switching occurs at the same layer, such as on the Internet. It turns out that paths in hybrid networks have restrictions that depend on choices made elsewhere in the path. The restrictions of a path through a network with a single switching layer are independent of the choices elsewhere in the path.

This thesis shows that paths that cross multiple technologies, the so-called multi-layer network connections, can go in loops. It is possible that a shortest path traverses the same fibre twice. In addition, it turns out that a segment of a shortest path does not have to be a shortest path in itself.

The largest part of this thesis covers the exploration and description of a formal model for multi-layer networks, including hybrid networks. The presented model is technology independent. This is an essential feature, since it means that neither the model nor the path finding algorithms need to be adjusted as new technologies are invented in the future.

Besides the model, the results of this work are two comparable multi-layer path finding algorithms. One of these algorithms has been implemented, and this thesis closes with a description of its first use.

Acknowledgment

Dear reader, this book is –by definition– written for you, regardless of whether you have read everything so far or immediately skipped to this acknowledgement (yes, I'm talking to you). There is nothing as depressing as working on a book for years if no-one is even going to open it.

"Years," you ask? Yes, years. And I wasn't even the only one doing the work. Jeroen van der Ham, Bert Andree and Karst Koymans did a lot of the thinking. Bert and I were working with group theory and XML to describe networks when Jeroen listened to an idea from Franco Travostino and pulled *NDL*, the *Network Description Language*, out of his sleeve. If it wasn't for this, we would still be struggling.

Karst Koymans was the first to suggest to look at the ITU-T G.805. Karst, I dedicate the path finding 'puzzle' in section 3.3.1 to you.

"ITU-T G.805?". Sorry about the technobabble. It's a standard, and if you think the name is unreadable, wait till you see the gobbledygook inside. It took us over a year before we turned abracadabra like *"A transport processing function that consists of a co-located adaptation source and sink pair"*[1] into section 4.3 of this thesis.

Cees, you owe me a crate of beer now. You are the living proof that free-form thinking in a scientist is a solid recipe for great new ideas. Of course you also prove that it is pointless to try to steer a bright mind. "Cees, I need to talk to you about chapter 3." "Oh, that's great. Let me show you how to play a 4k movie on my iPod first." It's amazing to see how far you are in front of the troops. Your idea that led to my first paper (on the multi-layer nature of optical exchanges) was published in late 2003, but only recently attracted attention from others.

[1]So, that's what the page number graphics represent. Now you see?

Paola, where Cees lacks in orderliness, you make up for it. Lighthouse or not, *you* have been the guiding beacon at the UvA during the last few years.

And there are so many more people to mention and thank. The members of my graduation committee, Pieter Adriaans, Rob Meijer, and in particular Tom DeFanti and Pascale Vicat-Blanc for traveling from abroad. My promotor, prof. Sloot, who I thank for taking the time to read my thesis. Colleagues and students, in particular Hans, Jaap, Erik, Ralph, Matthijs, Damien, Bas, Ronald, Niels, Arie, Andree, JP, Fred, Paul, and Li, I owe you all a drink. My current colleagues at SARA deserve recognition for their patience while I was making last-minute corrections to this thesis.

When I started this work, I did not want to look into path finding. Someone with my last name could not live up to the expectations[2]. Nevertheless, the topic appeared too tempting for me to ignore. Fernando Kuipers and his colleagues at TU Delft have been extremely helpful in developing path finding algorithms (I can only hope that the quality of this thesis approaches that of Fernando's).

SURFnet has not only financially supported this work. Without the Giga-Port project, I may not have met all the partners, TU Delft, Universiteit Twente, TNO, SARA, and others. Gigaport, above all, has been a club of friendly peers. If only the EU projects would take the same approach to a no-nonsense project overhead, there would be so much more great research.

Maxine Brown and Robert Patterson allowed me to include their GLIF map (figure 1.3). SARA and UvA financially supported the printing costs of this thesis, which allowed me to have an even fancier reception after my defence.

There are so many people I met in the GLIF community, OGF organisation and at conferences, that it is undoable to list everyone of them. So I will not even try. Talking to you certainly shaped my ideas, and perhaps it sometimes also shaped your ideas. I hope to talk to all of you in the foreseeable future.

Hennie, Wil, thank you for your during support. I wouldn't even have started, let alone ended this journey with your inspiration and kind steering all these years. My sister Marrit and my friends, thank you for your continued patience when I was busy once again. Floris and Jeroen deserve special recognition. Even after putting up with me, they still support me as paranymphs during my defence. My accomplice all the way was Caroline. Impatient sometimes ("It's over four years now, are you done yet?"), but always loving, caring and supportive. Supportive even when I was coding deep into the night with little attention for her. Thanks to invention of the laptop and wireless Inter-

[2]Of course I'm talking about Edsger W. Dijkstra.

net I could fulfil two tasks at once, keeping her feet warm in bed and keep on working next to her side. Even the ridicule by St. Nicholas[3] that inevitably followed did not stop me.

And with that we come to the end of this dissertation. There is always more to discover, but I leave that up to you, dear reader. If computer science is not your cup of tea, may I suggest a statistical distribution on my grammatical flaws per chapter? I'm sure there are countless plural errors left. Or perhaps you like to know what has occupied me for so long. Section 1.1 to 1.3 or even section 3.3.1 are good starting points. I'm done writing, I hope you enjoy the reading.

[3] It is Dutch custom to make small poems for relatives on the eve of St. Nicholas' birthday.

Biography

Freek Dijkstra (Hilversum, 3 July 1975) received his *doctoraal*[4] in applied physics from the Utrecht University in 2002. Between 1995 and 2003 Freek has been student teaching assistant, independent programmer, website developer at Uselab, and ghost-writer for the virtual laboratory for e-Science (VL-e) project proposal. In 2003 he said goodbye to physics and started his *Philosophiæ Doctor* research in computer science at the University of Amsterdam. Freek's drive is to make computer networks easier to maintain by finding solutions for practical problems faced by network engineers. Freek has published on transport technologies, link-local IP addressing, and path finding. The thesis in front of you is the culmination of this work.

Freek is currently employed by SARA Computing and Network as network researcher, were he continues his work to make networks easier to maintain by developing standards and software for use in computer networks. His current interests are topology descriptions and monitoring of multi-layer and multi-domain networks. Freek is co-chairing the Network Markup Language working group in the Open Grid Forum (OGF).

In his spare time, Freek spends a considerable amount of his time in front of his computer. Occasionally he can be found enjoying a train ride or the weather outside. Freek is married to Caroline Mattheij and lives in Breukelen.

This thesis is Freek's second book publication, after the humorous *"Uw geld of uw kaartje"* (2007) (ISBN 978–90–4390–968–6). The bibliography (section B.1) gives a list of Freek's scientific publications.

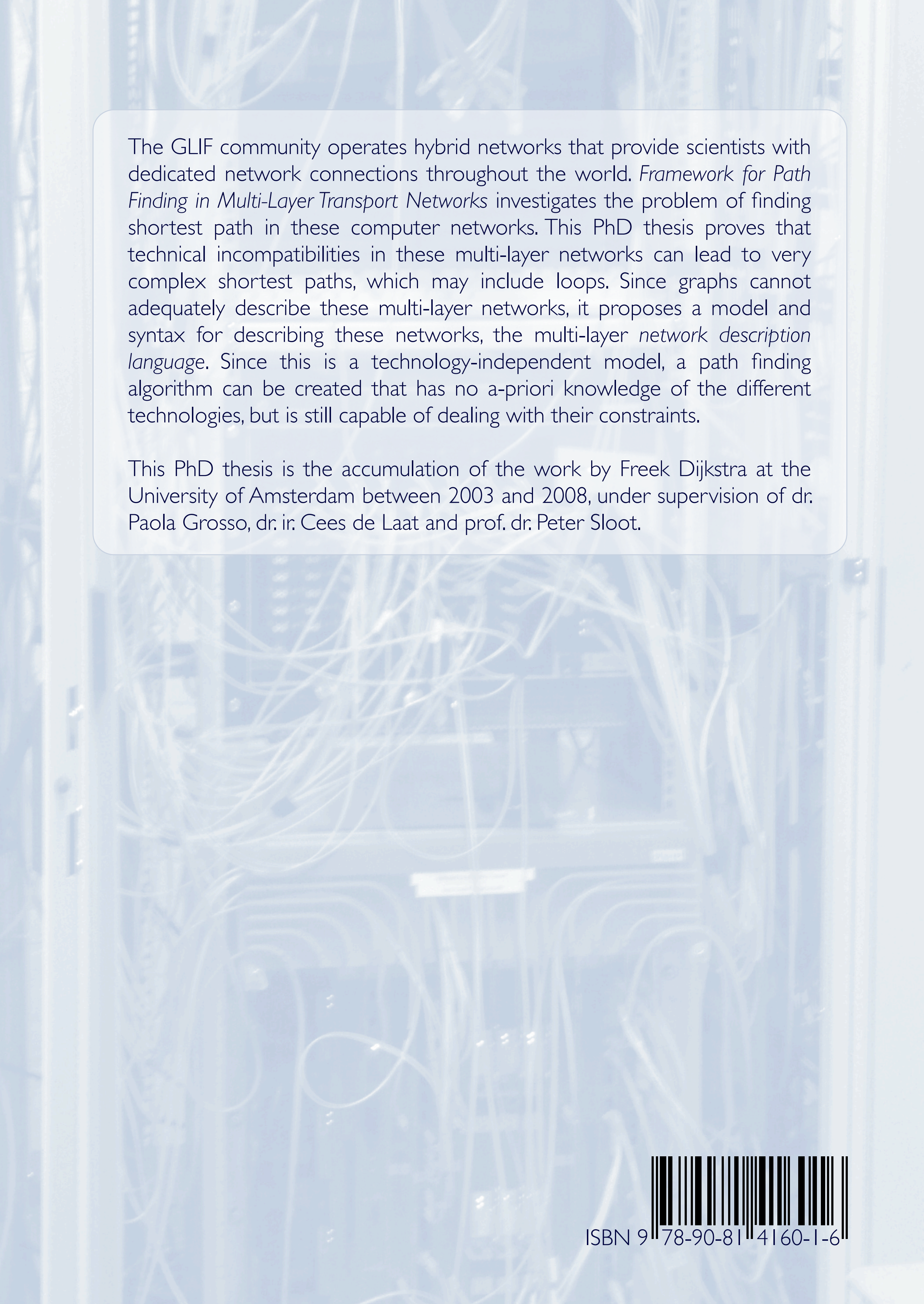
[4]Master of Science

Propositions, belonging to the dissertation
Framework for Path Finding in Multi-Layer Transport Networks
Freek Dijkstra, Amsterdam, 18 June 2009

- A network is not a graph. (§3.3.3)
- A verbose model *may* give a compact syntax, and vice versa. (§4.4.3)
- There will always be incompatibilities in networks, as long as technology evolves. (§2.3.1)
- A multi-layer path finding algorithms should be layer independent. (§6.1.2)
- Even the shortest path can contain loops. (§3.3.1)
- Computer science is 95% development and innovation and only 5% science. This is how it should be.
- In computer science, it is acceptable to study a system that other scientists created. This would be unthinkable in physics or social sciences.
- The essence of most propositions is trivial if you read them, but it is far from trivial to determine them.
- A publication that is neither searchable by Google nor downloadable does not count as a publication, since it does not exist for most people.
- Contrary to the proposition by De Laat's dissertation (1988), phonebooks for computer mail addresses turned out to be unwanted.
- Given the pace at which IPv6 is deployed, *Inertnet* is a good anagram for Internet.
- The use of disclaimers in e-mails and on websites can either be explained by lack of common sense of the recipient, or by the lack of accuracy by the sender. Both explanations are equally worrying.

Stellingen, behorende bij het proefschrift
Framework for Path Finding in Multi-Layer Transport Networks
Freek Dijkstra, Amsterdam, 18 juni 2009

- Een netwerk is geen graaf. (§3.3.3)
- Een uitgebreid model *kan* op een compacte manier beschreven worden en vice versa. (§4.4.3)
- Er zullen altijd incompatibiliteiten in netwerken aanwezig zijn, zolang de techniek voortschrijd. (§2.3.1)
- Een kortste pad algoritme in een meerlaags netwerk dient onafhankelijk van de lagen te zijn. (§6.1.2)
- Zelfs het kortste pad kan in een rondje lopen. (§3.3.1)
- Computerwetenschappen is 95% ontwikkeling en innovatie en slechts 5% wetenschap. Zo moet het ook zijn.
- Het is in de informatica geaccepteerd om een systeem te onderzoeken dat door andere wetenschapper gemaakt is. Dit is ondenkbaar in de natuurkunde of sociale wetenschappen.
- De essentie van de meeste beweringen is triviaal als je ze leest, maar om er op te komen is verre van triviaal.
- Een publicatie die niet gevonden wordt door Google en die niet te downloaden is telt niet als publicatie, omdat de meeste mensen hem niet kunnen vinden.
- In tegenstelling tot wat De Laat beweert in zijn proefschrift (1988) zijn telefoonboeken voor computer-mail adressen zeer ongewenst gebleken.
- Gegeven de snelheid waarmee IPv6 ingevoerd wordt, is *Inertnet* een goed anagram voor Internet.
- Het toevoegen van een voorbehoud (disclaimer) in e-mail en op website kan ofwel verklaard worden door het gebrek aan gezond verstand bij de ontvanger, ofwel door het gebruik aan nauwkeurigheid bij de auteur. Beide verklaringen zijn even zorgelijk.



The GLIF community operates hybrid networks that provide scientists with dedicated network connections throughout the world. *Framework for Path Finding in Multi-Layer Transport Networks* investigates the problem of finding shortest path in these computer networks. This PhD thesis proves that technical incompatibilities in these multi-layer networks can lead to very complex shortest paths, which may include loops. Since graphs cannot adequately describe these multi-layer networks, it proposes a model and syntax for describing these networks, the multi-layer *network description language*. Since this is a technology-independent model, a path finding algorithm can be created that has no a-priori knowledge of the different technologies, but is still capable of dealing with their constraints.

This PhD thesis is the accumulation of the work by Freek Dijkstra at the University of Amsterdam between 2003 and 2008, under supervision of dr. Paola Grosso, dr. ir. Cees de Laat and prof. dr. Peter Sloot.

ISBN 978-90-814160-1-6