Verifying functional requirements in multi-layer networks:

a case for formal description of computer networks

Maarten Aertsen
maarten@rtsn.nl 4096R/14789500

March 5, 2014

Master of Science thesis in Telematics

Committee:	Pieter-Tjerk de Boer, Boudewijn Haverkort,	(University of Twente)
	Jeroen van der Ham,	(University of Amsterdam)
	Henri Hambartsumyan	(Deloitte)
Chair:	Design and Analysis of Communication Systems	
Faculty:	Electrical Engineering, Mathematics and Computer Science	
Institution:	University of Twente, The Netherlands	

Abstract

Major outages and hacks in corporate networks show that the mounting complexity in computer networks has a direct impact on business. Although users have an intuitive understanding of how they would like the network to behave, network operators lack tools to match these implicit requirements against actual infrastructure. Verification is in the stone-age, with visual tracing of diagrams and drawings being the most common method to check network topology for conformance with user expectations. It is both tiresome and error-prone and does not scale beyond SME networks.

The main aim of this research is to examine the feasibility of automated property checking of real world networks. We build on work on network topology descriptions and multi-layer path selection and study the possibility of expressing expected behaviour on top of existing technology-independent algorithms, answering the question:

> *"Is it possible to verify operational requirements in multi-layer networks via algorithms for path selection in their topology descriptions?"*

We have found that there are topology descriptions languages available that allow formal description of networks for the purpose of reasoning. We have shown that it is possible to formally define operational requirements using a graph definition obtained from such descriptions. Using this formal definition we have synthesised verification algorithms to test conformance to such a requirement. These algorithms have been deomonstrated to decompose in multi-layer path selection and verification. And finally, we have shown a working proof of concept which puts these ideas in practice in a real world network.

Our conclusion: it is possible to verify operational requirements in multi-layer networks via algorithms for path selection in their topology descriptions.

Contents

Chapter 1.

Introduction

The unconnected computer is no more. We live in a world where communication and sharing are killer applications, software is delivered as a service and data is stored far away from its creators. In the midst of all associated advantages we tend to take the enabler of these advantages for granted: fast, robust and omnipresent telecommunication networks.

In the consumer sphere, networks have become a commodity. Everyone has several at home: for telephony, for television, for alarm systems or energy control and obviously for internet access. There are also plenty of networks in the mobile sphere: for phones, for tablets and to connect multimedia appliances. Given that most of these are self-installed by non-experts, networking is clearly no longer rocket science. Networks *just work*.

Meanwhile in the professional sphere, companies are having great trouble defining what *"just work"* means for them. For the brave few who have defined the expected properties of their networks, there appears to be no easy way to check their infrastructure for consistency with their definitions – and this shows.

In the Netherlands alone, we have seen spectacular single points of failure with the Vodafone fire [1] and far-reaching violation of security separation with the DigiNotar breach [7]. Clearly the ease of design and operation of networks with the proper characteristics has not kept step with plug and play style home networking.

1.1. Problem description

Technology stacks are getting more and more complex. Even though most development focuses on abstracting away from the underlying technology, the underlying stack does not magically disappear and strongly influences operation. Examples of this phenomenon can be seen at Telecommunications companies implementing 4G on top of their existing 2G and 3G infrastructure; in cluster and cloud environments that use network virtualisation to overlay topologies; and in hybrid packet/circuit switched networks as used in the research community.

Traditionally, this problem has been addressed with careful design, appropriate safety margins, continuous monitoring and regular audits. But these approaches scale badly with the addition of each technology layer unless they have been designed with multi-layering in mind from the ground up.

The main aim of this research is to examine the feasibility of automated property checking of real world networks. We will build on work on network topology descriptions and multi-layer path selection and study the possibility of expressing expected behaviour on top of existing technology-independent algorithms.

1.2. Research questions

In the next two subsections we detail our main research question and divide it into sub questions to be answered directly. The main question will be answered by combining their results. Before stating our research questions we will take the liberty of defining some terms and clarify some prerequisites.

The term "operational requirement" is used as formalized counterpart to "expected property", i.e. a property that is not only expected but also intended. We choose "operational requirement" and not merely "requirement" in order to emphasise our focus on real world day-to-day issues in networking. Operational requirements will be defined with more rigour in a subsequent chapter.

To reason about the requirements for real-world computer networks, this work explores the possibility of building upon two existing areas of research: network topology description and multi-layer path selection. Building upon network descriptions seems a natural fit, but the choice to involve path selection requires some justification, which we will provide in the next paragraph.

In a nutshell, multi-layer path selection is chosen for three reasons. First, it is a deployed technology which simplifies adoption of this research. Second, path selection is a core primitive in networking and reusing it allows tight coupling between network operation and property checking. Finally, utilizing path selection allows us to scope and focus this research on property checking by abstracting away from some of the complexity inherent in a multi-layered approach. More detailed justification will be provided in a subsequent chapter.

We presuppose the availability of an accurate, machine readable description of our target network. This description will be the foundation for our subsequent reasoning. This is not a trivial requirement and one that few networks meet today. We argue that without compelling reasons to engage in formal network description, this number will not be increasing any time soon.

Our goal is then to increase insight into the advantages of formal network description as a method to manage the mounting complexity in network stacks. We advance towards this goal by exploring one compelling use for these descriptions: continuous, real-time and automated verification of operational requirements in computer networks.

1.2.1. Main research question

With the introduction, problem description and prerequisites in mind, we now turn to the question this research aims to answer.

> *"Is it possible to verify operational requirements in multi-layer networks via algorithms for path selection in their topology descriptions?"*

1.2.2. Sub questions

1. *How do we describe multi-layer network topologies?*

   This question summarizes the current state of the art in academic literature with respect to the description of networks. The aim is to identify a description format for subsequent use throughout the research.

2. *What operational requirements can we relate to using topology descriptions?*

   This question aims to get an overview of the possibilities and non-possibilities of relating operational requirements to topology descriptions. An overview of possible requirements is given and real world use-cases are identified.

3. *Is it possible to formulate checks for operational requirements in terms of path selection?*

   To equate a requirement with the actual topology, we need to establish a common ground. This question tries to establish this common ground by reducing the verification of operational requirements to path selection problems.

4. *What multi-layer path selection algorithms are available?*

   Having established a method to formulate verification of operational requirements as path selection problems, this question surveys the existing algorithms for multi-layer path selection. In addition, we identify the changes required to these algorithms in order to utilise them for verification.

5. *How can the results be validated?*

   The aim of this question is to identify possible ways to validate any approach found answering the previous questions. This validation is subsequently performed in order to substantiate the real world value of a chosen approach.

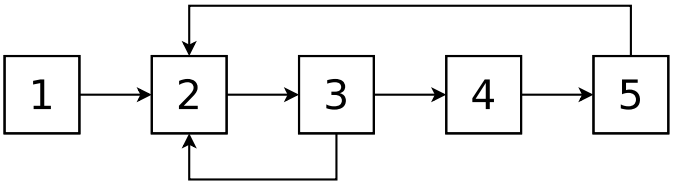


Figure 1.1.: research flow

1.3. Approach

The flow of research is as indicated in Figure 1.1. The numbers indicate the different sub questions while the arrows represent the approach taken to answering them.

The identification of a suitable topology format (sub question 1) and the overview of available path selection algorithms (sub question 4) are answered directly. The other question form part of a feedback loop, where work on identifying operational requirements (sub question 2) resumes if we fail to reduce it to a path selection problem (sub question 3) or if validation proves our method unsound (sub question 5).

This non-sequential approach suits the experimental and incremental nature of this research, as opposed to waterfall-style up front synthesis of theory followed by validation. Using the feedback loop we can evaluate the suitability of operational requirements for continuous, real-time and automated verification in an iterative way. Each requirement is examined step-by-step, with the hope of yielding interesting results even in the failure case.

On the contrary, the reader is not expected to trace the execution of the loop. Instead this thesis follows a more traditional flow for the sake of readability. This is further detailed in the next section.

1.4. Outline of the thesis

The remainder of this thesis is structured as follows. Chapter 2 covers existing work on topology description and path selection, answering sub questions 1 and 4. Background on and identification of operational requirements is the topic of Chapter 3, answering sub question 2. In Chapter 4 we answer sub question 3 and propose a reduction to path selection for each of the operational requirements selected. The proposal is validated (sub question 5) in Chapter 5. Finally, Chapter 6 features conclusions and an outlook on future work.

Chapter 2.

Literature

This chapter covers the two literature surveys which have been performed in preparation of our own work. The first part of this chapter covers topology description formats, intended as input for requirement verification. The second part covers multi-layer path selection algorithms, which will serve as the foundation of the actual verification. Ensuing chapters will build upon these topics to introduce requirements (Chapter 3) and formulate our proposal (Chapter 4).

2.1. Topology description

This section is based on "The Architectural Description of Networks, a Survey" by M. Aertsen, which is included in full in Appendix A.

This section summarizes the current state of the art in academic literature with respect to the description of networks topologies. The actual survey is prefaced with a subsection describing what we mean when we talk about topologies and define the term *topology information*. In the second subsection we list the results, answering research question 1: *How do we describe multi-layer network topologies?* This section is concluded with a look at the description format chosen for this research, the Network Markup Language version 1 (hereafter: NML).

2.1.1. Topology information

The Oxford dictionary defines the word topology as "the way in which constituent parts are interrelated or arranged". A network topology thus concerns the way elements in a network are interrelated or arranged. In practice *interrelation* includes information on physical items such as devices (routers, repeaters, switches, etc.) and links (aerial, fiber, copper etc.), while *arrangement* might refer to their capability and configuration. Topology is interesting in networking because it de facto defines the primary purpose of networks, their connectivity, in terms of the components delivering this service.

Network layering

Throughout the introduction in Chapter 1, and continuing through the remainder of this research, the term *multi-layer* will be used to indicate the presence of a specific network model involving multiple network layers. But up until this point, the term layer has not been explained. We will do so here.

There are many well-known works in networking which involve the notion of a network layer, including the OSI reference model [6] (or ITU-T X.200) and the TCP/IP protocol suite [2]. More informally, one can refer to a layer as the subset of a protocol using services provided by a less specialised subset of a protocol and providing services to a more specialised subset of a protocol.

The above informal definition gives rise to an almost infinite number of layers to be modelled. However, layers of interest to be modelled in the context of a network topology are usually only those layers which provide an independent service and are composable into layer stacks. This bring us to the term multi-layer networks.

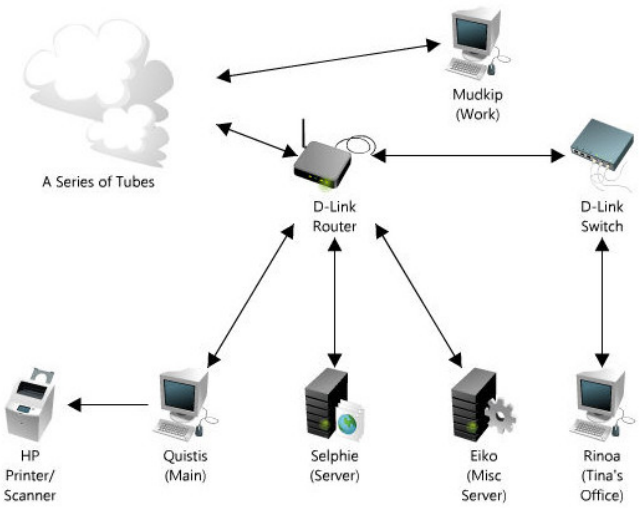
Although almost all networks in the real world comprise of multiple layers (and therefore qualify for the name multi-layer network), a distinction can be made based on the level of abstraction. A single-layer network is a network model where the focus lies with the top layer and lower layers are abstracted away. A good example is the term "IP-convergence", the process of transitioning telecommunication networks to a network layer based on the Internet Protocol.

A multi-layer network refers to a network model where multiple layers of the network are in scope, for example because the network is configurable at multiple levels each capable of delivering different services. In this research, when talking about property verification in multi-layer networks, the intention is to convey that this verification is possible not just for more than one, but for all layers available.

Example topologies

Before getting into formal topology description, we first take a look at the way knowledge about topologies is commonly transferred in practice. Three network topologies are briefly described here. Each example describes a topology using a different style, although they have in common that they are graphical (the predominant format for network topologies, as we will discover shortly). In the next paragraphs we will briefly comment on these sample topologies and make some observations about them.

home network The simplest example of a topology could be the network at home, connecting desktops, laptops and appliances to the internet. An example of such a topology can be seen in Figure 2.1. This figure does not contain much information; it



(Excerpt of "The Lameazoid Home Network" by Josh Miller, CC-BY-NC https://secure.flickr.com/photos/lameazoid/3718433170/)

Figure 2.1.: Sample home network

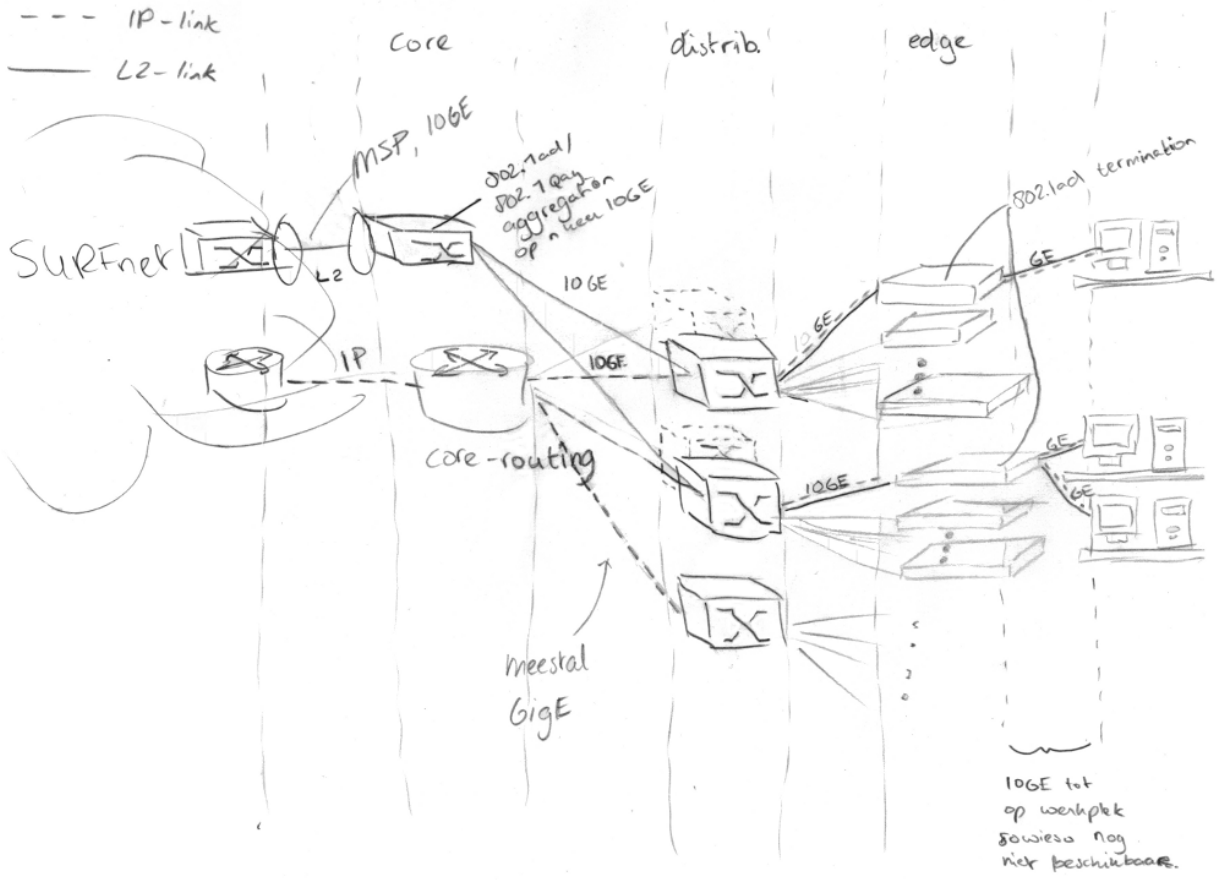
is limited to the different elements in the topology and the way the are connected. All other details are abstracted away.

campus network A second example of a topology is that of a campus network supplying connectivity to a university. Figure 2.2 features an informal drawing of such a topology. This drawing mixes information from different layers of the same network (as can be seen by the terms "IP-link" and "L2-link"). Additionally, details are given about the technology used by the links. Still, much of the detail is abstracted away (e.g. the underlying physical layer).

backbone network The third example concerns the network of a national research and education network. Figure 2.3 shows a single layer of this network, namely the fiber plant. This example is different in the fact that it purely focusses on structure, there are no devices, just the optical links and their cross connects.

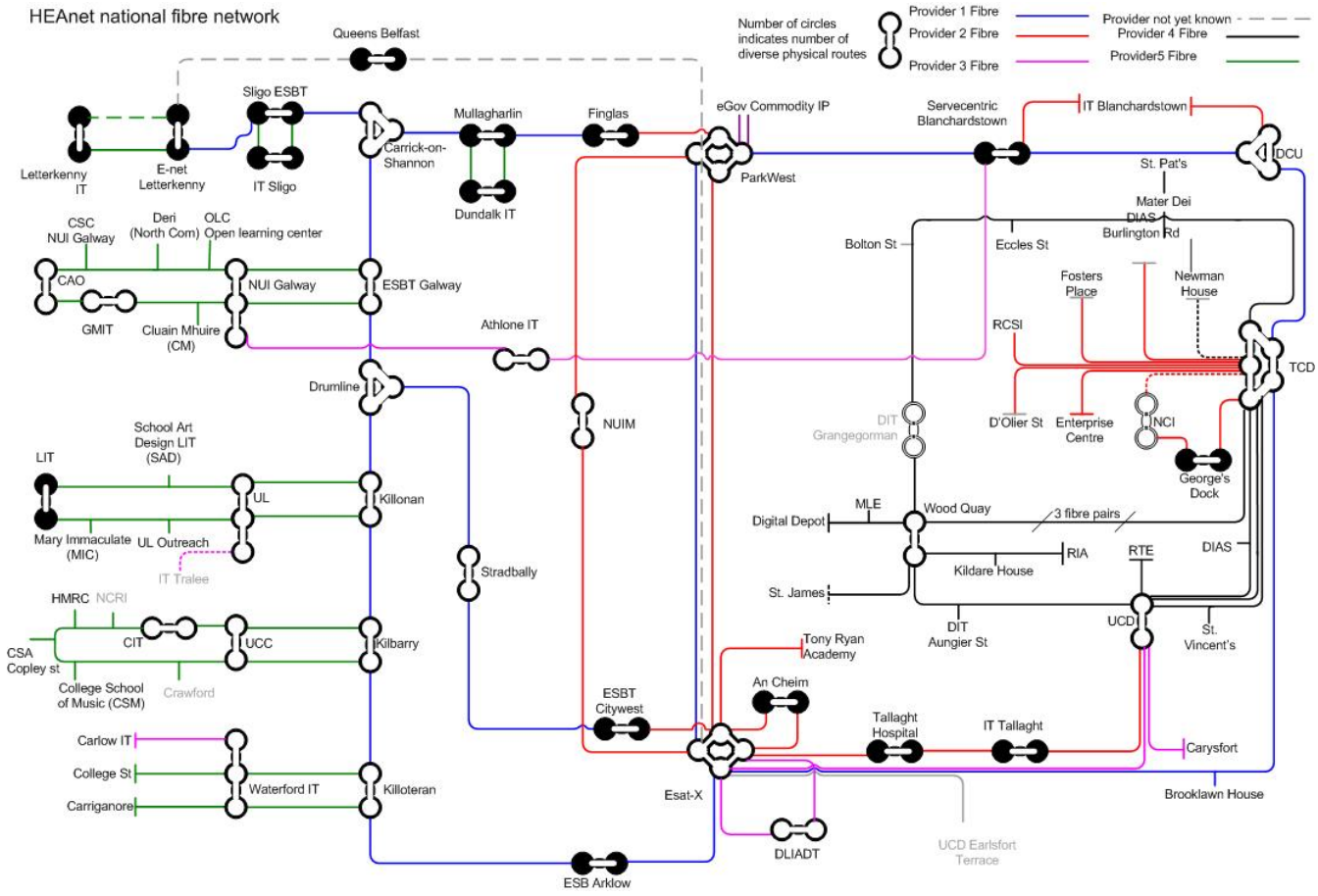
With a feel for different expressions of topology in the real world, we now generalize to establish a definition of topology information, which will be used throughout the remainder of this research. We will refer back to the example topologies in greater detail in Chapter 5.

Combining the textbook definition of topology, the insight in layering and having in mind the type of information included in real world topology we define topology information to mean: *All information that is a function of the interrelation or arrangement of the parts of a telecommunication network.* Consequently, a topology description is a format



(Concept drawing of lightpath implementation on UTnet, used with permission.)

Figure 2.2.: Sample campus network



(Dark fibre plant of Ireland's NREN HEAnet, used with permission.)

Figure 2.3.: Sample backbone network

language	single / multi layer	internal / exchangeable	scope	technology-neutral
NDL	single	exchangeable	network	yes
NDLv2	multi	exchangeable	network	yes
NML	multi	exchangeable	network	yes
INDL	multi	exchangeable	(virtual) network & infrastructure	yes
VXDL	single	exchangeable	network	no, IP-only
DEN-ng	multi	internal	not defined	yes
NNDL	single	exchangeable	node, service & network	no, IP-only
perfSONAR TS	multi	exchangeable	interface statistics	yes

Table 2.1.: summary of findings

to convey topology information. Finding real world topology descriptions is the topic of the next subsection.

2.1.2. Topology description

The goal of the topology description survey was "to summarize the state of art in academic literature with respect to the architectural description of networks". It answers the question "How is a network architecture formally described?". The terms *network*, *architecture*, *formally* and *described* are defined as follows.

By *network* the survey denotes a computer or telecommunication network, in *architecture* our focus is on physical topology, in particular OSI layer 1-3. With *formally* we mean suitable for calculation or reasoning. Finally, when talking about *described* we are looking for something which is exchangeable[1] and machine-readable.

The survey concludes that formal network architecture description is a small area of research, primarily driven by grid and cloud computing needs. There are a handful of languages available, each serving different specific needs. There is no wide adoption of any of the languages found, which leads to a selection based on our specific needs. For the purpose of future research in reasoning about the characteristics of topology the survey concludes that NML is most suitable as formal topology description language.

The results are summarized in Table 2.1. For a description of each of the description formats found, we refer to the full paper in Appendix A. We now turn to NML, the most promising of existing network description formats.

[1] Exchangeable in the sense that the description format allows for inter-operability, contrary to e.g. a binary database format, which can technically be exchanged, but is certainly not designed for that purpose.

2.1.3. NML

Quoting[2] from the NMLv1 base schema [17]:

> *The Network Markup Language is designed to create a functional description of multi-layer networks and multi-domain networks. An example of a multi-layered network can be a virtualised network, but also using different technologies. The multi-domain network descriptions can include aggregated or abstracted network topologies. NML can not only describe a primarily static network topology, but also its potential capabilities (services) and its configuration.*
>
> *NML is aimed at logical connection-oriented network topologies, more precisely topologies where switching is performed on a label associated with a flow, such as a VLAN, wavelength or time slot. NML can also be used to describe physical networks or packet-oriented networks, although the current base schema does not contain classes or properties to explicitly deal with signal degradation, or complex routing tables.*

Having covered the rationale behind NML, we will now turn to its practical application. Figure 2.4 provides a concise overview of its different elements. The most important element in NML is the *Network Object*. A *Node* is a Network Object and so is a *Port* or a *Link* (all found in the first column of the figure). A configurable part of a Network Object is modelled by as a *Service*. A sample of an NML description of one Node, two Ports and two Links can be found in Appendix C.1.

A common example of such a service is switching, to link ports within a device. A second example of a service is adaptation, a term we will use repeatedly in the remainder of this chapter. An adaptation marks the crossing from one technology to another, by means of an adaptation function. An example of an adaptation function is an encapsulation in a lower layer for the purpose of transportation, such as the encapsulation of IP in Ethernet. Before reaching the destination, this encapsulation will have to be undone at some point in the remainder of the path (deadaptation).

Groupings of Network Objects can be found in the third column and are also Network Objects in themselves, allowing for abstraction by multi-level grouping.

Our coverage of NML is deliberately limited, because we will mainly deal with the Network Object abstraction. As we will see in Chapter 4, this abstraction conveniently hides much of the complexity, while providing sufficient structure to obtain the formal underpinning for operational requirements. In keeping with this idea, the fourth column covers miscellaneous information which will not be explicitly covered here. However, for the interested reader, Figure 2.5 is included to get a feel for the relations between the different elements and the general expressiveness of NML. In general, the NML

[2] Quote used in accordance with full copyright statement as detailed on p. 99.

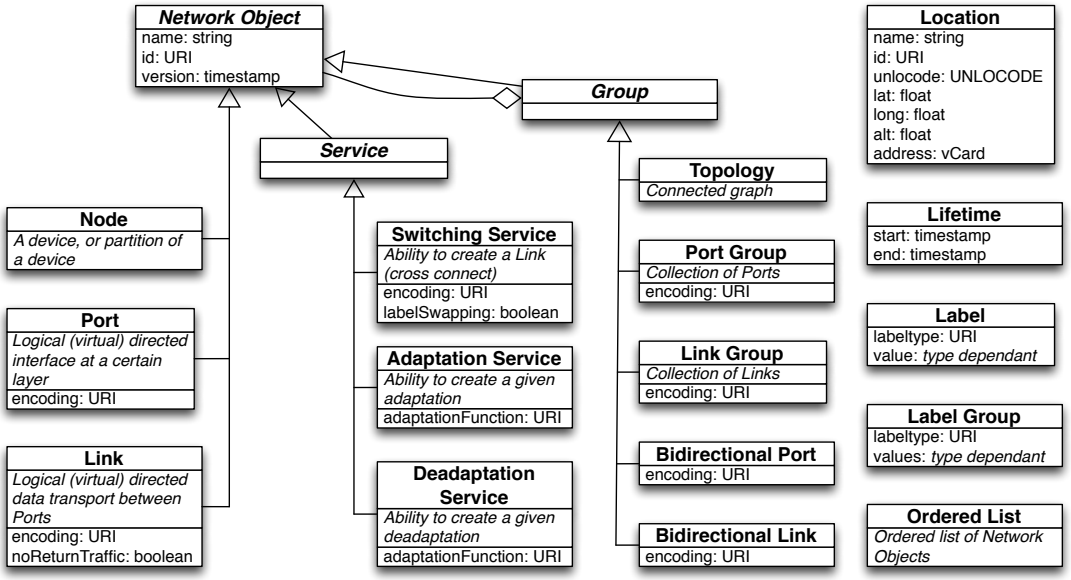


Figure 2.4.: A UML class diagram of the classes in the NML-schema and their hierarchy

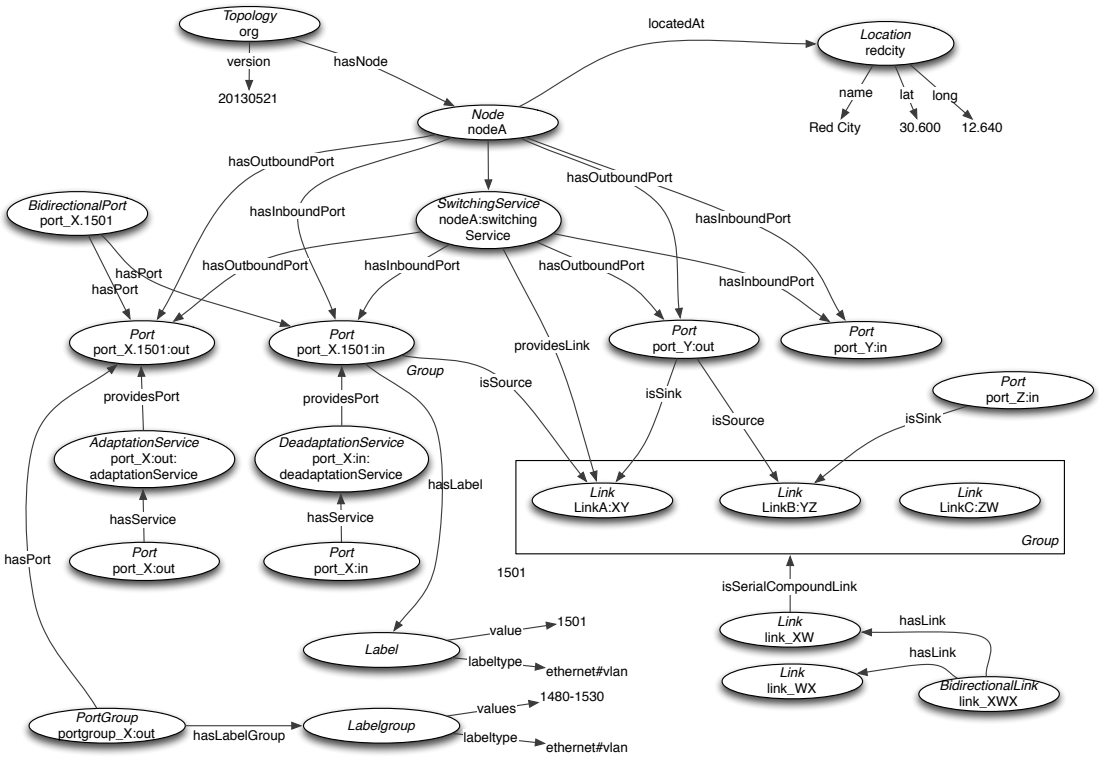
specification [17] is an excellent resource for those seeking to understand the mechanics in detail[3].

Having introduced the term topology information, answered research question 1 and briefly described NML, we now turn to the second part of this chapter, covering path selection.

2.2. Multi-layer path selection

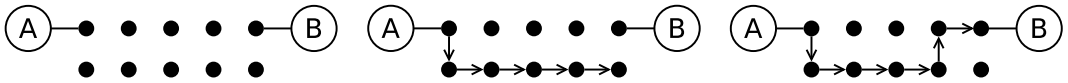
In this section we survey existing algorithms for multi-layer path selection. In order to present the survey, we first describe the difference between single-layer and multi-layer path selection. This builds on the previous section on topology description. We then turn to properties required for verification and subsequently present the actual algorithms. Finally, we describe ways to adapt existing algorithms to fit these requirements.

[3]We feel obliged to mention that within the scope of this thesis, intimate knowledge is only necessary for the implementation of path selection and for the formulation of a network description, both of which are instrumental to but not subject of this research.



(Figure 3 from GDF-R-P.206, used in accordance with full copyright statement as detailed on p. 99.)

Figure 2.5.: Graphical overview of example topology information in NML



(a) 2 layer network (b) infeasible path, no deadaptation (c) feasible path, using (de)adaptations

Figure 2.6.: path constraints in multi-layer path selection

2.2.1. Link-constrained versus path-constrained

Path selection means dealing with constraints. Common examples of such constraints are available capacity, latency or perhaps even monetary cost, all of which are so called link constraints. The topic of constrained path selection is a widely researched area. A good overview and performance comparisons are available in [12, 11].

A major difference between single layer path selection and multi layer path selection is that link constraints are not sufficient to always yield feasible paths.

A path constraint is a constraint which comes into play by choosing a particular sequence of links (a path). A good example of a path constraint within the context of multi-layer path selection is the use of adaptation functions (as explained earlier in the context of NML). In the context of path selection between two endpoints on the same layer (Figure 2.6a), a path constraint comes into play when using an adaptation somewhere along that path (Figure 2.6b), because deadaptation is required before arriving at the destination (Figure 2.6c). Therefore, by choosing a path featuring adaptation, the remainder is constrained to those paths allowing for deadaptation.

Dijkstra[4] showed that a shortest path in a multi layer network can contain a loop and that a segment of a shortest path in a multi layer network does not have to be the shortest path by itself. These are the reasons why multi layer path selection is a path-constrained problem, while single layer path selection is a link-constrained problem [5].

In conclusion, we need path-constrained path selection in order to express technology adaptations, which are a common occurrence in multi-layer networks. Path-constrained multi-layer path selection is an area less widely researched. The next subsection will provide an overview of the available algorithms.

2.2.2. Required properties

This subsection covers the required properties for a path-constrained path selection algorithm in order to be suitable in the context of this research. As indicated in Chapter 1, this answers research question 4, which although answered after research questions 2 and

[4]Incidentally, this is not the same Dijkstra who invented the shortest path algorithm named after him [3], a point the younger Dijkstra stresses in his dissertation [4], noting that the commonality gave him doubts about pursuing his research in shortest path algorithms.

3 (subjects of respectively Chapters 3 and 4) is covered here in order to group together the literature review of this thesis.

A path selection algorithm is required to return all paths that an actual path selection algorithm in the network could return. With the obligation to check all paths, the "shortest" metric is not of particular interest[5]. The ideal path selection algorithm is one which is closest to the actual network behaviour. A secondary concern is that of performance, which could prove to be vital when moving from proof of concept to real-time verification of a production network.

Finally (and only applicable to this research), taking into account the finite time available for a proof of concept, a path selection algorithm is most suitable when its input is closely matched by the used topology description. With our choice of NML in the previous part of this chapter, this means a close fit to NML.

2.2.3. Available algorithms

In this subsection we present an overview of current multi-layer path selection algorithms as found in academic literature. The coverage is divided into multiple sections, each covering a distinct approach. In order, we cover: breadth first search, layer by layer, k-shortest path, integer programming and automata theory.

breadth first search

In [10], Kuipers and Dijkstra describe two graph-like representations for multi-layer networks. Their first representation, denoted G_l evolves around devices on the same layer, while the second, named G_s is concerned with the available technology stacks. These choices represent two of the possible trade-offs involved in representing multi-layer networks in a graph-like form. The first representation (G_l) is a good match for a breadth first search (BFS) style algorithm, which the authors consequently present. We will cover their second algorithm for the G_s representation in the "k-shortest path" section.

In addition to the representation and algorithms, the authors show that multi-layer path selection in general is NP-complete by demonstrating a polynomial time reduction to the 3SAT problem. As a consequence, their exact algorithm and each of the other approaches which will follow are exponential in running time (assuming $P \neq NP$).

An evaluation of a Python implementation of this algorithm for G_l can be found in [5], which does this calculation on top of network descriptions in NDLv2. This approach is

[5]The exception to this statement is for non-exact verification by means of examining a subset of all paths. We believe that when verifying a subset of all paths it makes sense to cover paths in their most likely order of selection. This somewhat comparable to a heuristic, where one trades off exactness for performance. The use of heuristics for verification is not covered in this research, although it is an interesting direction for future research.

very suitable for the purpose of requirement verification due to the representation's close fit with NML-style description languages, although it uses the older NDLv2 standard which is unlikely to see further adoption with the standardisation of NML.

layer by layer

In [20], Xu et al. present an alternative to BFS by reasoning about path selection on a layer by layer basis. The authors present a bottom-up approach using a forward chaining reasoning engine[6]. This is very much suitable to network description formats re-using widespread formats for knowledge representation such as RDF. The authors use NDLv2, but a similar approach is possible using the RDF format for NML. The authors describe implementations both in the query language SPARQL[7] and the general purpose logic language Prolog.

k-shortest path

The second algorithm in [10] on the G_s representation is a k-shortest path approach. K-shortest path (e.g. Yen [21]) extends a normal shortest path approach by returning not just the shortest path, but k such paths, in increasing cost order. This in contrast to Dijkstra's algorithm [3].

In a number of algorithms, each node traversed from source to destination will store a number of shortest path, in increasing cost order. A variant on k-shortest path is to limit this storage to some upper bound (say k_{max}). This bound trades off exactness for performance because the storage limitation has the potential to prevent consideration of all possible paths. Note that such a variant need not return more than one path, the multiple path stored per node are simply a way to deal with the path-constraints.

An approach like the one just described is taken for the G_s representation in [10]. The more advanced TARA-family of algorithms in [8] use the same kind of optimisation. The authors of the latter even named the different algorithms based on the choice of k_{max}: ETARA is the exact algorithm without bound, while kTARA denotes the variant bounded by k (for $k \in \mathbb{N}^+$).

integer programming

In [16], Shirazipour and Pierre describe a hybrid method of path selection, combining Yen's original k-shortest path approach [21] with a binary integer program to optimize the constraints. A different network representation is used, based on GMPLS, which

[6] The use of bottom-up suggests the existence of a top-down approach, which we have been unable to find.

[7] SPARQL is a SQL-like query language to explore information in RDF.

makes this approach less suitable for NML. While Yen's algorithm has a polynomial runtime, Karp proved integer programming to be NP-hard [9]. No direct comparison is made with other algorithms.

automata theory

A radically different approach is taken in [13] by Lamali et al. who show that in a topology description using push down automata there exist polynomial time algorithms to find the shortest path if one disregards additional QoS parameters, either optimizing for smallest number of hops or for smallest number of adaptations. With the earlier proof by Kuipers and Dijkstra [10], though including bandwidth as a QoS constraint on top of the path-constrained technology incompatibilities, this result is surprising.

Unfortunately, the representation of the topology information in the algorithm is quite different from the available topology description formats (among them NML, used in this research). Short of a transformation to the required format, this makes this algorithm not very suitable for the purpose of this research. But the direction is promising, and using Lamali's algorithm for path selection does make for a suitable future extension of this work.

2.2.4. Choice of algorithm and required changes

Before explaining which algorithm was used to create a proof of concept for property verification, we want to stress that this research is essentially independent of this choice. That is, our goal is to build verification on top of path selection, without regard for the actual implementation. This holds as long as the algorithm conforms to the requirements stated in Section 2.2.2 (either directly, or through adaptation). Finally, being able to swap path selection algorithms with those actually used in the network under study is a marked advantage for the purpose of verification.

Having said that, for the purpose of a proof of concept, it is only natural to focus on one algorithm. In this research we choose to adapt the algorithm created for the G_l representation described in [10]. Because the original algorithm only returns the shortest path, whereas we need all paths, the algorithm was adapted to not terminate directly on finding the destination, but only after the queue is empty. The full algorithm adapted for the purpose of a proof of concept can be found in Appendix C.

With respect to future research, it should be possible to use the path selection algorithm of each respective network layer instead of one multi-layer path selection algorithm as done here. Such a choice would add network *behaviour* to our current approach based on network topology at a relatively low cost (because the path selection algorithm is swappable). Due to time constraints, this addition is out of scope for this research, but it is most definitely a promising direction for future research.

2.3. Summary

This chapter served to answer research questions 1 and 4, which will be repeated below. In the first half of the chapter a previous survey with this explicit purpose was summarized, answering the question:

> *How do we describe multi-layer network topologies?*

Moreover, the Network Markup Language was chosen as the topology description language of choice in this research.

By listing the different approaches to multi-layer path selection, the second part of the chapter answered the question:

> *What multi-layer path selection algorithms are available?*

In addition, an algorithm was picked and adapted to be used in a proof of concept, which is the topic of subsequent chapters.

This concludes the literature part of this thesis. In the next chapter, operational requirements are explained and identified. Chapters 4 and 5 then respectively propose a method for verification and validate this proposal.

Chapter 3.

Operational requirements

In this chapter we concern ourselves with the question "what operational requirement can we relate to using topology descriptions?". Answering this question is split up in multiple parts. We start by defining operational requirements, expanding on the introduction and including examples to illustrate the practical meaning of the term. We continue, with the results of Chapter 1 in mind, to identify the suitability of certain classes of requirements. Finally we select a number of requirements and motivate their appropriateness for subsequent evaluation in later chapters. We conclude the chapter with a summary of our findings.

3.1. From expectation to requirement

In the introduction we motivated the need to define expected properties when networks get more complex. In the research questions we dubbed these expected properties "operational requirements". Before delving into examples, we will first attempt to define this term here.

operational requirement We define an operational requirement to constitute *the intended and expected behaviour of a telecommunications network during its operation.* This expectation is ideally formally documented and used throughout the design process, but the more realistic implicit expectation is equally reasonable to use as part of our definition.

Different entities have different perspectives on operational requirements. Contrast the office-worker accessing the network to get his job done, the network operator configuring a redundant link for a new customer, a systems integrator deploying systems in a trusted zone or even a telecom regulator mandating accessibility of emergency services. Although these present distinct use-cases, the general observation is consistent: violating the expectation of any of these users means the network fails to deliver[1].

By formalizing each of these expectations as part of a set of operational requirements and keeping these tightly coupled to the network, the network engineer can make informed

[1] Clearly, this does not hold for any *unreasonable* expectations on the part of a user with respect to services a network is unable to facilitate (e.g. teleportation). In this research, when referring to "expectation", we intend to convey expectation within the realm of reason.

choices during the design and operations. In this research, we attempt to automate checking of expectations as part of a set of operational requirements. This does not immediately solve the problem of implicit requirements, but does allow for an iterative approach by adding additional requirements as soon as they surface.

3.1.1. Motivating examples

To gain intuition for operational requirements we will proceed to cover two examples motivating the need to move from expectation to requirement. For examples, we return to the Dutch incidents referred to in the introduction. In each case we briefly summarize and then extract a possible operational requirement.

Vodafone fire

> *"Vodafone Group Plc (VOD), the worlds largest mobile-phone operator, said clients in the western part of the Netherlands arent receiving service after a fire in a network center in Rotterdam.*
>
> *Customers in cities including Rotterdam and The Hague, and in Kennemerland, a region around Amsterdam, cannot call or be called, Anniek de Ruijter, a spokeswoman for Vodafone in the Netherlands, said by phone. About 700 transmission towers are out of service after the fire broke out this morning at the Rotterdam site, affecting both mobile and fixed lines, she said. "*
>
> – Bloomberg, April 4th, 2012 [1]

The stricken network center controlled a 2G and 3G network servicing 1.3 million customers and additionally hosted voicemail systems and connections to other operators, both local and abroad, impacting customers throughout multiple countries. Restoration of the functionality was reported a week later[2].

A disruption such as faced by Vodafone is a complex event. In our analysis, we choose to start coverage from the user perspective, subsequently moving down to the service delivered and associated requirements.

Every Vodafone customer expects to be able to call and (possibly) use network services at his or her convenience. From Vodafone's perspective, this translates into the availability of the 'call' and 'data' services. The call service includes connectivity to and from countries abroad through connections with partners. With the disruption of roaming

[2]Impact and time line were obtained from Vodafone's concluding press release (`http://over.vodafone.nl/nieuwscentrum/nieuws/statement-storing-vodafone-door-grote-brand-bedrijfspand-rotterdam-0`, retrieved on 22-10-2013), affected customers were obtained by combining Vodafone's annual fact sheet (`http://over.vodafone.nl/vodafone-nederland/feiten-en-cijfers/factsheet-vodafone-nederland` using data points 30 June 2011 and 31 March 2013, retrieved on 22-10-2013).

and international calling, the call service and therefore the customer expectation was violated.

Assuming fire and other (natural) disasters are unavoidable, it seems natural to reason on the basis that outages of a network centre will happen. If Vodafone wants its services to tolerate such an outage, it should provision its services so that they can continue operating with the loss of a network centre. Returning to the international call service, it seemed that the diversity of the international connectivity was lacking. An example operational requirement for the international call service might be:

Connections between Vodafone NL and foreign telecommunications providers for the purpose of roaming and international calling shall be diverse enough to suffer a path outage without loss of service.

Similarly, with respect to the transmission tower outage, it appears that control was centralised to such an extent that the outage of one switch location took a substantial area out of service, with manual restoration of control on a site-by-site basis as the only option.

Control of transmission towers and/or base stations shall be possible from more than one remote location.

Vodafone makes a relevant statement in a later press release[3], yielding another example of a operational requirement:

"Thus we are working on separating the 2G and 3G systems, so that if one system fails, the other system continues to work and vice versa, minimising the impact on our customers."

DigiNotar intrusion

> *"DigiNotar provided digital certificate services as a Trusted Third Party and hosted a number of Certificate Authorities (CAs). [..] In June and July of 2011 DigiNotar suffered a breach, which resulted in rogue certificates being issued that were subsequently abused in a large scale attack in August of 2011."*
>
> – Fox-IT, August 13th, 2012 [7]

The investigators commissioned to investigate the breach note in their initial *"state of affairs"* that *"The construction of the network security zones corresponded with best practices [..]"* [7, p. 18]. Still, the attacker, by using a web server in the `DMZ-ext-net`,

[3] `http://over.vodafone.nl/nieuwscentrum/nieuws/reactie-vodafone-op-onderzoek-veiligheidsregio-rotterdam-rijnmond` reacting to a safety inquery, retrieved on 23-10-2013. Original text (Dutch): *"Zo werken we aan het scheiden van de 2G- en 3G-systemen, zodat als het ene systeem uitvalt, het andere systeem blijft werken en vice versa, waardoor de impact op onze klanten zo klein mogelijk is."*

a database server in the `Office-Net` and escalated administrative access to the actual certificate authorities in the `Secure-Net`[4], managed to sign and exfiltrate rogue certificates. The investigators conclude in the investigative summary that *"The zones were not strictly described or enforced and the firewall contained many rules that specified exceptions for network traffic between the various segments."* [7, p. 4].

The "lessons learned" section provides some additional insight on what was amiss. Here the investigators list a number of recommendations to Certificate Service Providers (i.e. DigiNotar and competitors), possibly filling the gap between DigiNotar's security zones and actual segmentation. Two of their recommendations make for nice examples of operational requirements for a CSP network:

- *"Air gap vital systems as much as possible, to make sure that they are physically separated on a network level from untrusted networks such as the Internet."* [7, p. 68]
- *"Separate vital logging services from the systems that perform other vital functions. In an infrastructure where secure logging is vital, a logging server can be placed behind a unidirectional security gateway."* [7, p. 69]

3.2. Relating operational requirements to topology descriptions

In Chapter 1 we selected a suitable topology description language, answering research question 1. Here, we apply our knowledge of its expressiveness to identify classes of requirements which are a natural match for requirement verification based on topology information. By discussing the characteristics of operational requirements we deem unsuitable (Section 3.2.1), we arrive at the definition of a class of requirements suitable for verification (Section 3.2.2). We subsequently derive generalised requirements in the next section, using the same examples for inspiration, and show the applicability of the entire class to automated verification in later chapters.

3.2.1. Unsuitable requirements

A number of the traditional properties of networking are concerned with real-time performance. In particular, this includes latency, jitter, bandwidth, throughput and packet loss[5]. Recall our previous definition of topology information as *"all information that is a function of the link-level implementation and configuration of a telecommunication*

[4]This is postulated by the attacker [7, p. 51], but unverified by the investigator. The report notes on the suspected source of access to `Secure-Net` that *"Due to limitations on the investigation this workstation was not examined"*.

[5]We distinguish bandwidth and throughput by noting that bandwidth concerns data rates, while throughput measures packet or connection rates.

network" (Section 2.1.1). All the properties listed above are a function of both topology information and input (i.e. traffic).

The higher the number of information sources required for verification, the higher the bar for actual verification. Consequently, by singling out topology information as information sources, we put an upper bound on the required information for verification. We believe this conservatism with respect to the number of information sources has real merit for real world application of this work by reducing the initial complexity of an implementation.

In addition to the primary motivation listed above, our focus on topology information has the advantage of clearly restricting our scope to topology information. We consequently classify as unsuitable those requirements that can not be checked using only topology information. Note that this choice does not preclude the "unsuitable" requirements from being checked in general, it just requires using a secondary information source on top of topology information. Reintroducing some of these "unsuitable" requirements could consequently make for an interesting future extension of this work.

A second characteristic of requirements we deem unsuitable to include in this initial research concerns connectionless network layers[6]. Although today's Internet predominantly uses the connectionless IP protocol to transport traffic on higher layers, the underlying (physical) infrastructure is often connection-oriented (i.e. composed of nodes *connected by* links[7], or circuit-oriented transports such as 802.1q vlans, MPLS LSP's or SONET/SDH channels).

As a consequence of both our definition of topology information and our chosen description language NML, we work with a connection-oriented knowledge base. This has an adverse effect on the requirements we can verify using topology information. For example, we will not be able to check properties on IP-level connectivity unless a suitably scalable extension is created for NML to capture complexity associated with (connectionless) packet-switching, such as routing and forwarding tables. In order to keep the scope of this research within bounds we choose not to tread outside the topology model as present in the first version of the NML standard. Nonetheless, we stress that a current lack of support in NML does not prove or disprove the general ability to do requirement verification for connectionless layers in a topology.

In summary, our choice to exclusively focus on topology information using NML precludes requirements depending on secondary data sources or pertaining to connectionless layers from being verified within the scope of this research. Having excluded two classes, we will now characterize requirements suitable for verification.

[6] Note: we use the term connection in the same way the authors of NML do, as explained in Section 2.1.3.

[7] An interesting exception to this statement is wireless technology, where no physical link is required to transfer information. Therefore monikers connection-oriented/connectionless are less clear here.

3.2.2. Suitable requirements

Suitable for verification are those requirements that relate to the structure of a topology or its resulting behaviour. In particular, verifying connectedness or other graph theoretic properties of a telecommunications network seem a natural match for automated verification based on topology information. Using the topology information to abstract over or generalizes node or link specific properties without losing exactness is another alley worth exploring.

Questions remain whether any of these bottom-up derived possibilities match with real world requirements. In the next section we will find that quite a few real world expectations about networks fit in suitable operational requirements.

Note that we make no attempt to prove or disprove that these classes are unique. It so happens to be that they are a good match with real world needs. As such they are suitable to show the advantages of formal network description, thus aligning with the stated goals of this research.

3.3. Selected requirements

We previously gained intuition for the concept of operational requirements and have established criteria for our ability to relate to operational requirements using topology information. This section covers a selection of generalized operational requirements which are abstract enough to cover the requirements distilled from the two examples first used in the introduction.

For each of the requirements we define their meaning and explain their use. Subsequently, every requirement is tied to one or more of the examples described in the first part of this chapter.

In later chapters we will establish whether the verification of these specific operational requirements is possible using path selection and determine if this yields a valid solution for their verification.

3.3.1. Segmentation

In the DigiNotar case, we gave two specific examples of operational requirements related to a more general operational requirement we will call a segmentation requirement. We will define segmentation as *the absence of connectivity between two endpoints (at some layer in the network)* or more generally as *the absence of connectivity between two subnetworks.*

Our aim will be to use this segmentation requirement to prove administrative (no configured path), logical (no available path) or full separation (no existing path or "airgapped")

at the network level. With the limitations of NML in mind, particularly with respect to connectionless network layers (e.g. IP), we realize that verification of administrative separation without false negatives is not attainable[8]. Our goal is to prove false positive-free verification of administrative, logical and full separation.

Returning to our examples, the "air gap" is a trivial case of a segmentation requirement, whereas the "unidirectional security gateway" can be viewed to unidirectional or one-way segmentation, which nicely fits NML's unidirectional model of networks.

3.3.2. Control

Knowledge of a network topology allows for the abstraction of a requirement on a single network element to an entire network. We define control as *the existence of a guarantee about the connectivity between two endpoints (at some layer in the network).* We thus obtain a topology level verification of an arbitrary control criterion for individual nodes and links, resulting in something we will call a control requirement.

To illustrate this rather high level definition we will provide a concrete example of using a control requirement. Suppose a financial service provider wishes to strictly conform to privacy regulations which require it to obtain explicit permission from a regulator before allowing access to personal identifiable information to countries outside the European Union.

In order to realize conformance to this regulation, the company formulates a control requirement stating that *all communication between a payment terminal and the processing backend should be under EU jurisdiction.* Here the control requirement would be "under EU jurisdiction", which might be simplified to "located on EU territory"[9]. This in turn is part of topology information as a property of all physical equipment in that topology.

Note that the abstraction from a specific control criterion means that a control requirement could easily involve a secondary information source as a future extension of this work.

3.3.3. Time to recovery

Time to recovery abstracts over the individual recovery properties of nodes and links in the network. Time to recovery in this context denotes the complete replacement of an element after its (physical) failure. Although such a replacement could involve suppliers

[8] Not attainable in the sense that a network which is not administratively separated on any connection-oriented layer might still be separated on a connectionless layer (e.g. on an IP layer using firewall rules or router ACL's).

[9] We could also simplify to "owned by a EU corporation", but we have no intention to dive deeply into legal matter just to illustrate a concept.

whose reaction time can be viewed as an additional data source, we view the recovery time as a static property of an element and as such as a part of the topology information.

We define time to recovery as *the maximum tolerable duration for the recovery of connectity between two endpoints (at some layer in the network) assuming failure of the least favorable element.*

Abstracting time to recovery from individual network element to entire topologies allows one to quantify the actual guarantees the network delivers, taking into account the characteristics the topology provides. This enables bottom-up quantification, complementing by nature in-exact risk based estimates of network robustness and resilience.

A time to recovery requirement can help to provide insight in the resilience of a network, check for conformity to existing Service Level Agreements or even underpin the formulation of new SLA's.

3.3.4. Path diversity

The first two examples in the Vodafone case are operational requirements concerning what we will call path diversity, or *a minimum fault-tolerance for connectivity between two endpoints (at some layer in the network) based on the number of distinct paths available.* It is possible to distinguish between distinct nodes and distinct links, and choose to define path diversity accordingly. We choose to require both: distinct paths imply both distinct nodes and distinct links.

The minimum mentioned in our definition allows for a quantified measure of diversity by supplying the number of required distinct paths. For example, one could specify that two endpoints need to be connected by at least three distinct paths.

In the first Vodafone example it appears that the available paths from Vodafone NL to competitors, domestic and abroad, used some of the infrastructure in the Rotterdam data center. For a path diversity requirement to hold, an additional, distinct path should be provisioned to each competitor. The second example, the transmission tower control, a path diversity requirement would be met if there existed fall-back connectivity to each tower in the case of a primary link failure.

3.4. Summary

In this chapter, the term operational requirements was defined. The usefulness of operational requirements was illustrated by two real world examples, each showcasing several instances of an operational requirement. The second part of the chapter considered the suitability of certain classes of operational requirements for verification based on topology information, tying in to the sub question this chapter meant to answer:

What operational requirements can we relate to using topology descriptions?

Requirements utilising multiple data sources and requirements relating to connectionless network layers were classified as unsuitable for inclusion in this research, respectively in order to keep focus on topology information and because the chosen (best available) topology description format has no support for connectionless network layers. It was then conjectured that requirements relating to graph theoretic properties or generalized node and link specific properties would be a natural match for verification based solely on network topology information.
The third part of this chapter saw the identification of four distinct generalized operational requirements, each covering one or more of the example requirements from the first part.

In Chapter 4 we will propose a method of verification for the four selected generalised operational requirements using path selection. Then in Chapter 5, we determine whether this yields a valid solution.

Chapter 4.

Proposal: verification using path selection

To equate a requirement with the actual topology, we need to establish a common ground. This question tries to establish this common ground by reducing the verification of operational requirements to path selection problems.

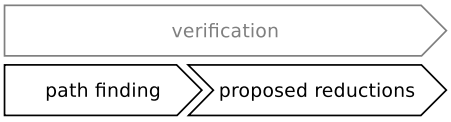


Figure 4.1.: verification as a two piece composition

Using this approach we replace verification by the composition of path selection and some additional computation. This is illustrated in Figure 4.1. Before turning to the individual requirements, we discuss prerequisites for this composition.

4.1. Prerequisites

There are three prerequisites to cover: the motivation to build upon path selection; an explanation of the different parts involved in our proposed reductions and finally the introduction of the notation used throughout this chapter.

4.1.1. Motivating the choice for path selection

Before proposing a way to verify operational requirement using path selection, we will motivate why path selection is involved in the first place. This subsection expands on the introduction, where three reasons were given to consider path selection. Each reason will be covered separately.

deployed technology The first reason to involve path selection is to leverage existing research and implementation, both to build upon and to ease adoption of our solution. The primary use-case for the standardisation of NML and its predecessors was to to enable path selection through multi-domain optical networks. By following this approach and use path selection on top of NML, it is possible to learn from previous experience.

core network primitive The second reason follows from the fact that path selection is a core network primitive. The target network for the verification of operational requirements is directly influenced by its path selection algorithms at various layers. By re-using path selection, we allow tight coupling of verification of properties in a network to the path selection algorithm used in that same network.

When re-using the path selection capability of the network, it is possible to reduce the computation required for verification. If the results of path selection can be shared, this part of the computation essentially comes in for free.

The alternative to building upon path selection would be to do direct computation of requirements from NML. This would most likely involve algorithms on graph or logic representations of the full topology, which in the end would largely correspond to the composition of path selection and our proposal. Network engineers are presumed to be more familiar with path selection than with the underlying graph theory or advanced logic. With the need to trust automatic verification performed, this increased familiarity can only yield positive effects.

scope and focus The final reason to build upon path selection instead of working on raw topology data is a matter of focus. It allows us to scope and focus this research on property checking by abstracting away from some of the complexity inherent in the multi-layered approach we take to networks.

We conjecture that, due to their inherent applicability to networks, the requirements we want to verify are decomposable with at least one part depending on the outcome of path selection. Consequently, we predict that the path selection part of the full computation required can be abstracted and re-used. We will verify this conjecture in the remainder of this chapter. Our choice gives rise to the question whether this composition of path selection and some other computation is actually feasible. Although a prototype can yield useful information in this respect, a full complexity analysis is out of scope for this thesis.

4.1.2. Breaking down verification

We break down our reduction in two distinct ways. First, we split the reduction in computation and validation and motivate this choice. Then we cover a requirement by NML which allows us to focus on communication in a single direction. Both transformations simplify the algorithms to be proposed.

Separating computation and validation

A effective method to break down the verification of operational requirements is to separate the reasoning about the underlying property (computation) and comparing it to the required result (validation). This idea is illustrated in Figure 4.2.

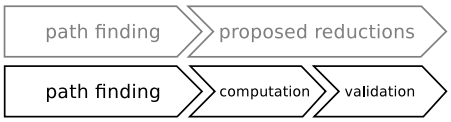


Figure 4.2.: verification as a three piece composition

We list two reasons for this choice:

annotation In some occasions, the results of the full computation can be used to annotate the result of the validation. This way, the user is presented with more than just a Boolean outcome of the verification process.

simplification Breaking down the verification in subcomponents makes it easier to reason about the parts. In particular, validation usually comprises of a simple comparison performed sequentially on the result of the computation. Keeping validation out of scope simplifies our coverage of the computation and also eases verification and implementation of this work.

It is important to note that for some problems, full computation might actually be of different complexity than mere validation. We will deal with this later.

Requirements imposed by NML

By using NML as format for topology information, we are bound to the constraints it imposes as a format. Specifically, NML models communication as uni-directional. Bidirectional communication between entities is modelled by paths in both ways.

In the real world, communication is rarely unidirectional. In line with the representation chosen for NML we do not assume symmetric paths. Therefore, to perform verification of bidirectional communication we compose the verification of the two directions, as illustrated in Figure 4.3.

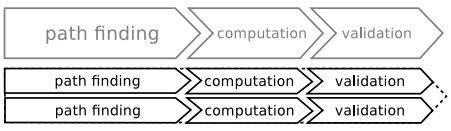


Figure 4.3.: verification as a parallel 3 piece composition

Although this choice in NML requires us to perform computation in both ways[1], it comes with the advantage of simplifying the reasoning involved. To this end, we will treat all verification as unidirectional, unless indicated otherwise.

4.1.3. Topology graph

The final prerequisites before covering our actual proposal deal with the topology graph used throughout the remainder of this chapter. In this section we define a topology graph on top of the implicit graph in NML. The next section introduces notation on top of this topology graph.

Figure 4.4 shows relations between the classes in NML. We define a topology graph over a subset of these classes and relations, by using classes as vertices and relations as edges. The classes and relations in black are those that are traversed by the multi-layer path selection algorithm and have been previously explained[2] in Chapter 2. The others, in grey, can be used for other purposes within the algorithm, such as discerning between different traversels of the same NML Link (`Label`) or to identify a port in the reverse direction (`Bidirectional Port`), but are not traversed directly. As such, they are no part of the topology graph defined here.

We proceed to formally define Topology graph $\mathbb{T}$ in multiple steps. Throughout the definitions, it is helpful to recall that $\mathbb{T}$ is essentially a subset of NML, represented as a graph.

Using the NML schema as defined in version 1 of the standard [17]:

$\mathbb{C} :=$ the set of *classes* in the NML schema

$\mathbb{R} :=$ the set of *relations* between classes in the NML schema

A relation of type R_t is of form $(C_1, R_t, C_2) \in \mathbb{R} : C_1, C_2 \in \mathbb{C}$

a topology graph is defined as follows:

$\mathbb{T} := (\mathbb{N}, \mathbb{E})$

$\mathbb{T}$ is a directed graph consisting of the set of vertices $\mathbb{N}$ and the set of edges $\mathbb{E}$

$\mathbb{N} := \{N \mid type(N) \in \mathbb{C} \setminus \{SwitchingService, Group\}\}$

The set of types in $\mathbb{N}$ consists of {Node, Port, Link and (De)AdaptationService}, and consequently is a subset of $\mathbb{C}$

$\mathbb{E} := \{(N_1, N_2) \mid (N_1, N_2) \in \mathbb{R}_\mathbb{T}\}$

$\mathbb{E}$ consists of seven possible (directional) relations between NetworkObjects in $\mathbb{N}$, defined as the set $\mathbb{R}_\mathbb{T}$ in the first two columns of Table 4.1

[1] In so far this was not already required by the behaviour of the network layer itself.

[2] We do not explicitly explain the relations between the NML classes covered. Their names should give a good indication of their meaning.

$\mathbb{R}_T := \{$	$(\mathbb{R}_\mathbb{T} := \{(N_1,$	$N_2)\}$	$(C_1,$	$R_t,$	$C_2))\}$
	Node	Port	Node	hasOutBoundPort	Port
	Port	AdaptationService	AdaptationService	providesPort	Port
	Port	Link	Port	isSource	Link
	Port	DeadaptationService	Port	hasService	DeadaptationService
	Link	Port	Port	isSink	Link
	AdaptationService	Port	Port	hasService	AdaptationService
	DeadaptationService	Port	DeadaptationService	providesPort	Port

Table 4.1.: Definitions of sets $\mathbb{R}_\mathbb{T}$ (first two columns) and $\mathbb{R}_T$ (all five columns), with the elements of $\mathbb{R}_T$ defined over the elements of $\mathbb{R}_\mathbb{T}$

Moreover, on a network described using NML:

$D := (D_{Objects}, D_{Relations}) : \forall O \in D_{Objects} \,.\, type(O) \in \mathbb{C},\ \forall R \in D_{Relations} \,.\, type(R) \in \mathbb{R}$

A network description in NML, consisting of Objects and Relations[3].

a specific topology graph T over D is defined as follows:

$T := (\mathbb{N}_D, \mathbb{E}_D)$

T is a directed graph consisting of the set of vertices $\mathbb{N}_D$ and the set of edges $\mathbb{E}_D$

$\mathbb{N}_D := \mathbb{N} \cap D_{Objects}$

$\mathbb{N}_D$ is the subset of $D_{Objects}$ that is also part of $\mathbb{N}$

$$\mathbb{E}_D := \{(N_1, N_2) \mid (N_1, N_2) \in \mathbb{N}_D \,.\, \exists (C_1, R_t, C_2) \in D_{Relations} : ((type(N_1), type(N_2)), (C_1, R_t, C_2)) \in \mathbb{R}_T\}$$

$\mathbb{E}_D$ consists of seven possible (directional) relations between NetworkObjects in $\mathbb{N}_D$, with the five-tuple connecting $\mathbb{R}$ and $\mathbb{R}_\mathbb{T}$ defined as the set $\mathbb{R}_T$ in Table 4.1

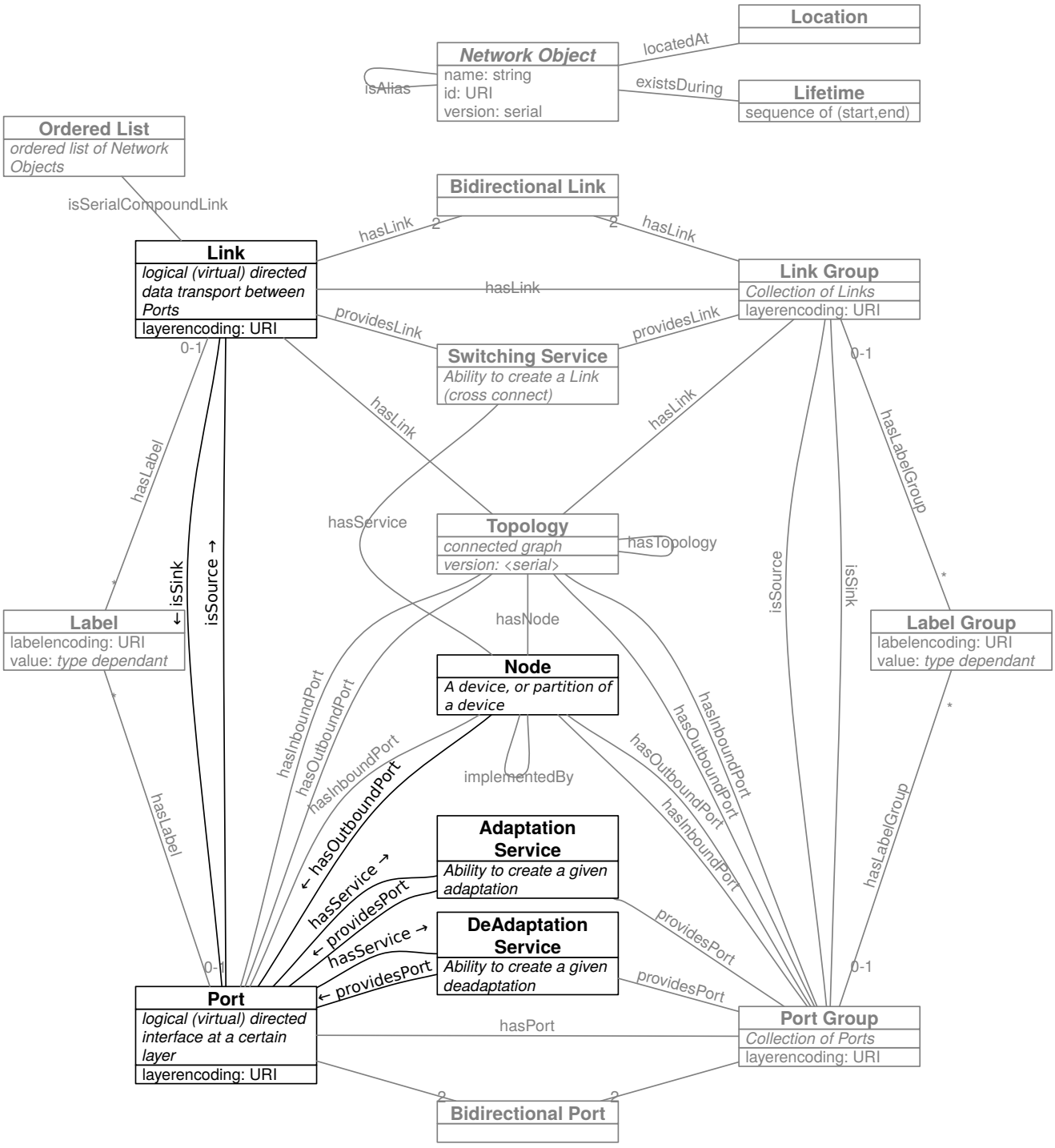
4.1.4. Notation

Having previously defined the topology graph, we now introduce a notation used throughout the remainder of this chapter. We first describe the notation for formal reasoning about the requirements and then the notation used to describe the algorithms.

formal description

We previously covered NML in Section 2.1.3. Here we define a notation based on the entities in NML used to describe the reasoning involved in and later prove the correctness

[3]Moreover, NML defines attributes per Class, which are not relevant nor covered here.



(Annotated version of NML-relations.pdf from the source repository of GDF-R-P.206, used in accordance with full copyright statement of GDF-R-P.206 as detailed on p. 99.)

Figure 4.4.: The full NML relation graph (partly greyed out) and the subset of relations used to define the topology graph $\mathbb{T}$ (in black)

of our proposed algorithms. We heavily reuse common notation for set theory, which we will not cover here.

$$
\begin{aligned}
(A \to B) &:= \text{a path through } \mathbb{T} \text{ from A to B (zero or one)} \\
(A \rightrightarrows B) &:= \text{all paths through } \mathbb{T} \text{ from A to B (zero or more)} \\
\mathit{elements}(\cdot),\ \mathit{paths}(\cdot) &:= \text{NetworkObjects, paths contained in } \cdot
\end{aligned}
$$

A major choice made is the to abstract from the graph-oriented distinction between edges and vertices (or more commonly links and devices). We choose to follow the NetworkObject abstraction used in NML, where both links and nodes are an instance of a NetworkObject.

This choice is natural when you consider the fact that from the perspective of our chosen operational requirements, the distinction between nodes and links is meaningless. After all, segmentation is equally influenced by the existence of relevant links and devices and this same reasoning applies to the other requirements covered. Furthermore, when describing multiple network layers, the question what exactly denotes a link (and subsequently an edge) becomes subjective. We prefer not to deal with this question and will deal with NetworkObjects from now on.

Note that it is not impossible to either define an operational requirement dealing with actual links nor is it impossible to verify such an operational requirement. However, we prefer to deal with the abstraction when possible, both to simplify and to generalize.

Algorithm description

To describing the algorithms, the goal is to keep as close as possible to the formal descriptions used to synthesise them. At the same time we wanted to use some standardized format and not invent another home-brew pseudo language.

Throughout the remainder of this research we will use the Haskell language for both description and evaluation of our proposal. The Haskell syntax is very clean and, by virtue of its referential transparency, very close to mathematics. Haskell satisfies our two goals for a description format: it is very close to our formal description and it is formally standardized [14].

By using Haskell for the description of our algorithms, we get two extras for free in addition to our listed goals. First, we obtain executable specifications[4], which will come in handy when attempting empirical validation. Second, strong typing in Haskell works in our favour when using a compiler to check our specifications.

[4]Complementing the algorithms is a tool written in Haskell to check operational requirements on a NML topology. Excerpts of the most relevant pieces of source code of the this tool can be found in Appendix C. All source is released under BSD 3 clause and available from `https://rtsn.nl/thesis/`.

Although its popularity has been strongly increasing over the past few years, we feel that the relative obscurity of Haskell necessitates additional explanation, especially where constructs uncommon outside the functional paradigm are used. This is a slight disadvantage, but one we are prepared to bear in favour of the listed advantages. After all, additional documentation never hurts.

4.2. Segmentation

Recall that our goal is to transform the problem of verifying operational requirements into the composition of finding a path and a secondary algorithm. We start with the most trivial of transformations: the identity transformation, which we will find to suite the verification of a segmentation requirement.

Segmentation was defined in Section 3.3.1 as *the absence of connectivity between two endpoints (at some layer in the network)*. It directly follows from this definition that segmentation is the dual of connectedness.

4.2.1. formal definition

Written formally, the dualism quickly becomes apparent:

properties

$$\begin{aligned} \exists(A \to B) &\iff (A \rightrightarrows B) \neq \varnothing && \text{connectedness} \\ \neg\exists(A \to B) &\iff (A \rightrightarrows B) = \varnothing && \text{segmentation} \end{aligned}$$

Splitting verification in computation and validation, we define the outcome of computation as the process of obtaining any paths violating the segmentation requirement. Checking then becomes the check whether any such paths exist.

We mentioned the trivial transformation because the only thing we have to do in the computation step is to produce the result of path selection. Validation is almost as simple, featuring a simple check for the empty set. Both 'algorithms' can be directly extracted from the formalisation above.

4.2.2. proposed algorithm

In every algorithm we present in this chapter, we define a computation function to transform the output of path selection, a list of `Path` (written as `[Path]` in Haskell) to the outcome of the computation, again as a list of Path[5]. Additionally we define

[5] More specifically the type signature notation preceding most functions, e.g. `foo :: [Path] -> Bool` (which can be read as `foo` is a function from type list of `Path` to type `Bool`), is helpful to see this transformation.

a validation function to transform the output of the computation to a boolean value representing the outcome of the validation. Together, these two functions comprise the proposed transformation on top of path selection to produce verification of the desired property[6].

Our earlier observation that segmentation can be seen as the trivial transformation is reflected in both the computation and the validation part of the algorithm, both of which are extremely simple:

```
module Reduction.Segmentation where
import NML (Path)

connectingPaths :: [Path] -> [Path]
connectingPaths = id -- Haskell's identity function

-- | validateSegmentation returns True when ther are no connecting paths,
-- and False otherwise.
validateSegmentation :: [Path] -> Bool
validateSegmentation ps = null ps
```

Listing 4.1: Segmentation verification in Haskell

The computation features the identity function. After all, every path connecting A to B is a violation of this property and thus a useful result of the computation. The validation function features the check for the empty set (`null` in Haskell).

We will now turn to examples requiring more and more "additional computations", though the basic principle and structure stays the same.

4.3. Control

Control was defined in Section 3.3.2 as *the existence of a guarantee about the connectivity between two endpoints (at some layer in the network)*. In this section we formalize this definition to obtain a method of verification.

4.3.1. formal definition

Formalizing our requirement, we obtain the following. Given A and B and control criterion C, determine whether paths between A and B satisfy control criterion C. We

[6]The code for the actual composition can be found in Appendix C.3. Compositions are not listed in the chapter because they add little to the algorithms themselves. Moreover, the reader is not assumed to be familiar with the monadic plumbing required to make the composition work in a larger program. For similar reasons, the code in this chapter is simplified not to use currying, lambdas or point-free expressions.

start with a function ϕ defined in terms of C on a single node and abstract to the paths from A to B in two steps.

additional notation:

$\phi(N)$ $\quad N \in \mathbb{N}$

Property ϕ holds if N is under control (wrt. to criterion C)

properties:

$\phi(A \to B) \iff \forall N \in elements(A \to B) : \phi(N)$

A path from A to B is under control if all elements on the path are under control

$\phi(A \rightrightarrows B) \iff \forall (S \to F) \in paths(A \rightrightarrows B) : \phi(S \to F)$

Communication from A to B is under control if all paths from A to B are under control

Using this reasoning in the reverse order, we obtain a formulation for control in terms of path selection. That is, for all paths and for all elements of those paths, we check the property ϕ, retaining those paths that fail to satisfy the control constraint in some way. This process is the computation phase of verification. Validation is as simple as checking whether computation yielded any unsafe paths. If such paths exist the verification fails, and succeeds otherwise.

4.3.2. proposed algorithm

Algorithm 4.2 is a direct translation of the previous section to Haskell code. In the algorithm, ϕ is represented by `Criterion`, which returns a result depending on (internal) knowledge of C.

```
module Reduction.Control (Criterium, unsafePaths, validateControl) where
import NML (NetworkObject, Path)

type Criterium = NetworkObject -> Bool

-- | inControl abstracts a 'Criterium' over a 'Path'
-- It yield True only if the entire 'Path' is under control
inControl :: Criterium -> Path -> Bool
inControl f p = all f p

-- | unsafePaths returns a list of 'Path' not satisfying the 'Criterium'
unsafePaths :: Criterium -> [Path] -> [Path]
unsafePaths f ps = filter (not . inControl f) ps

-- | validateControl return True if all Path's are under control, False
-- otherwise.
validateControl :: [Path] -> Bool
validateControl ps = null ps
```

Listing 4.2: Control requirement verification in Haskell

The computation part of the verification is represented by the function `unsafePaths`, while validation forms the standard name `validateControl`.

4.4. Time to recovery

Time to recovery was defined in Section 3.3.3 as *the maximum tolerable duration for the recovery of connectity between two endpoints (at some layer in the network) assuming failure of the least favorable element.* This definition again suggests a consecutive compute and validate approach: first we compute the maximal duration and then we compare it to the maximal tolerable duration to validate the requirement.

In a way, time to recovery resembles control as covered in the previous section. They both are an abstraction over the property of a single element in a path to the entire path or even all paths between two endpoints. But with time to recovery the abstraction function is no accumulation of a binary yes/no result, but the accumulation of recovery times. Specifically, we are interested in the maximal duration for recovery for each path or set of paths.

With the resemblance in mind, we proceed to present a formal definition of time to recovery, much in the same way as was done for control.

4.4.1. formal definition

additional notation:

$T_r(N)$ $\quad N \in \mathbb{N}$

Function $T_r(N)$ denotes the time to recovery of N.

$max_{T_r}(O)$, $min_{T_r}(O)$ $\quad O \subseteq \mathbb{N}, O \neq \varnothing$

Functions max_{T_r} and min_{T_r} compute the NetworkObject with the maximum, respectively minimum time to recovery among a set O.

properties:

$T_r(A \to B) :=$ $\quad max_{T_r}(elements(A \to B))$

the worst case recovery time of a path

$T_r(A \rightrightarrows B) :=$ $\quad min_{T_r}(\{T_r(K \to L) \mid (K \to L) \in (A \rightrightarrows B)\})$

the worst case recovery time of set of paths

The generalisation from a single path to a set of paths (e.g. all paths from A to B) using min_{T_r} merits some explanation. With the availability of multiple paths, we assume that recovery only takes place once all paths are down. That is, bringing back only a subset

of the paths is not a recovery, because nothing failed in the first place. In the case that all paths are down, it makes sense to recover the paths with the lowest recovery cost. Therefore generalising the worst case time to recovery for a set of paths involves taking the best (i.e. minimum cost) among the paths to be restored.

An implicit assumption made above is that recovery of elements on a given path can happen in parallel. For a network, this seems to be reasonable. However, even in the case that recovery would proceed in a sequential manner, the resulting change to our formal definition of time to recovery is highly contained. The effect on the overall algorithm is therefore very small: with an appropriate alternative definition of max_{T_r} and min_{T_r} the same algorithm applies.

Once again reasoning in the reverse order yields an algorithm to compute time to recovery between endpoints A and B. That is, for all paths, we take the NetworkObject with the worst recovery time and then select the best among those to yield the result.

Validation is then a simple comparison of the result of the computation against the required value. A cost higher than specified violates the requirement, a cost lower or equal to the requirement passes.

4.4.2. proposed algorithm

```
module Reduction.TimeToRecovery (
worstCaseRecoveryCost, validateTimeToRecovery
) where
import Data.List (maximumBy, minimumBy)

import NML

type RecoveryCost = NetworkObject -> Cost


-- | Compute the 'NetworkObject' with the worst 'Cost' among a list of
-- 'Path's.
worstCaseRecoveryCost :: RecoveryCost -> [Path] -> (NetworkObject, Cost)
worstCaseRecoveryCost f [] = error "no_path_supplied"
worstCaseRecoveryCost f paths = (pair . min . max) paths
    where
        -- find the bottleneck for each path
        max :: [[NetworkObject]] -> [NetworkObject]
        max nss = map (maximumBy (recoveryCompare f)) nss

        -- find the optimum amongst the bottlenecks
        min :: [NetworkObject] -> NetworkObject
        min ns = minimumBy (recoveryCompare f) ns

        pair :: NetworkObject -> (NetworkObject, Cost)
        pair n = (n, f n)
```

```
-- | Specialised comparison function for 'NetworkObject's in terms of
-- 'RecoveryCost'
recoveryCompare :: RecoveryCost -> NetworkObject -> NetworkObject -> Ordering
recoveryCompare f a b = f a `compare` f b

-- | validateTimeToRecovery returns True when there is a 'Path' within the
-- worst case recovery time given, False otherwise.
validateTimeToRecovery :: Int -> (NetworkObject, Cost) -> Bool
validateTimeToRecovery n (_,c) = n >= c
```

Listing 4.3: Time to recovery verification in Haskell

In Listing 4.3, `worstCaseRecoveryCost` represents the computation, while `validateTimeToRecovery` allows validation. Helper function `recoveryCompare` can be seen as the subscript T_r in the max_{T_r} and min_{T_r} functions: it provides a way to do comparison based on the recovery cost of NetworkObjects.

4.5. Path diversity

Path diversity was defined in Section 3.3.4 as *a minimum fault-tolerance for connectivity between two endpoints (at some layer in the network) based on the number of distinct paths available.* Checking such a requirement requires the ability to count the number of distinct paths between two endpoints, to subsequently compare this number to a specified minimum. Once again a natural split between computation and validation.

4.5.1. formal definition

Given $(A \rightrightarrows B)$ determine whether there are pairwise distinct paths.

Elaborating on the description given earlier, when talking about set of distinct paths we mean that there is no overlap between any of the paths. Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a graph where $\mathbb{V}$ represents the set of all paths and $\mathbb{E}$ represent a binary no-overlap relation (i.e. $((P_i, P_j)$ between paths. A set of distinct paths is a fully connected (sub)graph of $\mathbb{G}$, also called a clique. Figure 4.5 shows examples of cliques of increasing size, illustrating the rapid increasing number of edges required ($\frac{n^2-n}{2}$ for $n = |\mathbb{V}|$).

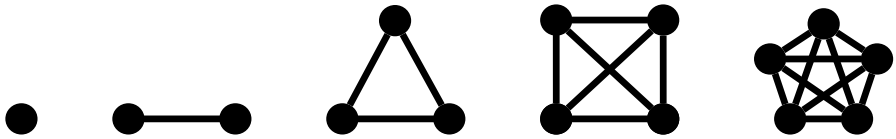


Figure 4.5.: Cliques of increasing size

Finding clique's (more commonly, "the clique problem") is a heavily studied subject in Mathematics and Computer Science. Deciding whether a graph contains a clique larger

than a given size (the clique decision problem) was one of the original 21 problems Carp showed to be NP-complete [9]. Finding the maximum clique is therefore NP-hard. Finding cliques of fixed size k is significantly easier, with recent algorithms reaching running time complexity of $O(n^k/(\epsilon \log n)^{k-1})$, polynomial if k is not part of the input [19].

With this formulation of $\mathbb{G}$ and the idea to search for cliques, we are only left with the definition of the no-overlap relation resulting in $\mathbb{E}$. This relation can formally specified as follows:

$$
\begin{aligned}
(A \to_i B) \text{ does not overlap } (A \to_j B) \iff & \forall N_i \in elements(A \to_i B), \\
& \forall N_j \in elements(A \to_j B) : \\
& N_i \neq N_j \qquad (4.1) \\
\iff & elements(A \to_i B) \cap elements(A \to_j B) = \varnothing \qquad (4.2)
\end{aligned}
$$

Considering computation, we note that we are not interested in all cliques, but only in the clique of the largest size. Incidentally the largest clique contains cliques of all lesser sizes. But for mere validation, we are not interested in the largest (or more commonly 'maximum') but in the size equal to the required number of distinct paths.

4.5.2. proposed algorithm

With the previous statements on complexity in mind, this is an example where validation represents significantly less work than full computation of the underlying property. This is reflected in the algorithm, which allows switching implementation for the clique finding depending on the ultimate use: computation uses maximum clique finding, while computation composed with validation uses k-clique finding.

In sequence, the algorithm pairs all path combinations together, filters them for distinctness and finally uses clique finding to produce the result.

```
module Reduction.Diversity (distinctPaths, validateDiversity) where
import Data.List (delete, intersect, maximumBy)
import Data.Maybe (mapMaybe, fromMaybe)
import qualified Data.Map.Lazy as Map

import NML (Path)
import Util.Clique (getCliques)

-- | 'distinctPaths' returns zero or more groups of the highest # of distinct
-- paths. Every group consists of pairwise distinct paths. An optional maximum
-- diversity can be supplied, which is used to switch to (polynomial) k-clique
-- algorithms.
distinctPaths :: Maybe Int -> [Path] -> [[Path]]
distinctPaths _ [] = error "no path supplied"
distinctPaths max ps = (getCliques max . filter distinct . pairs) ps
```

```
    where
        distinct :: (Path, Path) -> Bool
        distinct (a,b) = null (intersect a b)

                pairs :: [Path] -> [(Path, Path)]
        pairs [] = []
        pairs (p:ps) = [(p,b)|b<-ps] ++ pairs ps

validateDiversity :: Int -> [[Path]] -> Bool
validateDiversity n pss = ((>=n) . length . head) pss
```

Listing 4.4: Path diversity verification in Haskell

We choose not to implement clique finding ourselves at this stage, instead using available specialized libraries. This presents a possible area for optimisation which is not considered at this point.

4.6. Conclusion

In this chapter, an approach was presented to verify operational requirements using multi layer path selection. After covering prerequisites; the motivation for the choice of path selection; the choice for a compositional approach and an explanation of the notation used throughout the chapter, algorithms were explained for all four operational requirements. This chapter served to answer research question 3, which is repeated below.

> *Is it possible to formulate checks for operational requirements in terms of path selection?*

Keeping in mind the limitations on operational requirements as formulated in Chapter 3, the proposed composition with computation and validation in this chapter show that it is possible to formulate checks for operational requirements in terms of path selection. We illustrated this claim with the presented algorithms for four generic operational requirements.

Due to the nature of our proposal, we get an implicit theoretical proof of correctness for free. This is illustrated in Figure 4.6, which shows that the verification of operational requirements in multi-layer networks follows naturally from a set of transformations on operational requirements and network descriptions. First an operational requirement is formally defined using a graph definition obtained from our chosen network description format. From this definition a verification algorithm is synthesised, which in turn is decomposable into path selection and some additional computation. Although we refrain from formally writing out this implicit proof, the construction used provides an implicit theoretical justification for the correctness of the proposed verification method.

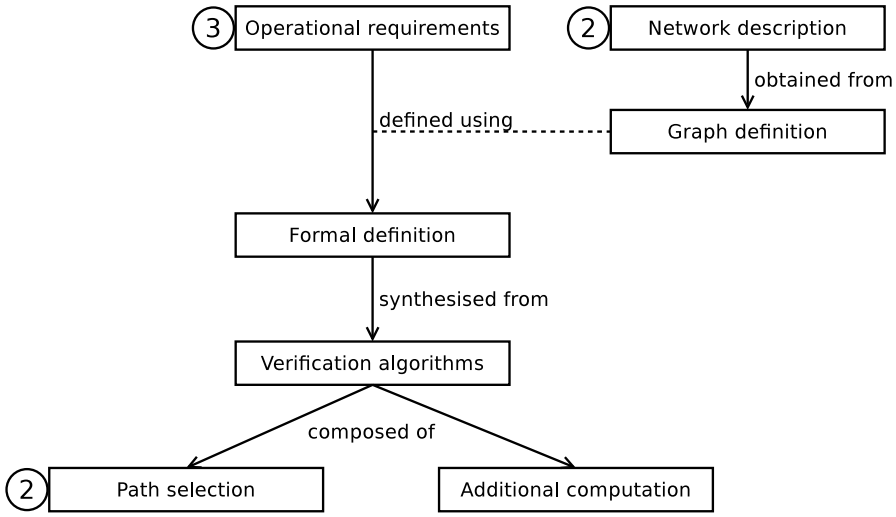


Figure 4.6.: Proposal construction in terms of previous chapters (numbers encirceled)

The next chapter builds on this theoretical result supporting the main thesis that it is possible to verify operational requirements using topology descriptions by showing that verification as proposed in this chapter is possible in a real world network. These combined results lead to our conclusions in Chapter 6.

Chapter 5.

Validation

This chapter examines ways to validate the proposal formulated in Chapter 4 in order to show its feasibility in the real world. In the conclusion of the previous chapter we established that our proposal, by construction, supplies us with an implicit formal proof of correctness. In this chapter, we demonstrate a working proof of concept for requirement verification in real word computer networks. It brings together the theory from Chapter 2, the selected operational requirements from Chapter 3 and the algorithms from Chapter 4. We start by presenting the methodology.

5.1. Methodology

In this section, the methodology to test a proof of concept is outlined. The methodology gives structure to this chapter. It describes how the different tests are prepared, how they are to be conducted and how results are interpreted.

First we give an overview of the different steps required before tests can be conducted. Then we cover the two methods to be compared: manual verification versus automated verification. Finally, we describe how the results will be compared and how conclusions will be derived from results.

5.1.1. Validation methodology

We start with very a brief overview of the methodology, with references to the different parts of this chapter.

1. Determine the characteristics of a suitable network (Section 5.2)
2. Select a network (Section 5.3)
3. Select operational requirements (Section 5.4)
4. Conduct tests and detail results (Section 5.5)
5. Summarize and reach conclusions (Section 5.6)

5.1.2. Benchmark: manual verification of requirements

We start with the description of the benchmark the proof of concept's results are evaluated against. As benchmark, we use manual verification of each requirement.

In particular, for each of the cases, available topology information (e.g. in available network information systems) is used to manually trace and check the requirements. For some of the requirements, this might involved tremendous amounts of work (e.g. path diversity). This limits the possible complexity of any network involved in validation by the sheer impossibility to conduct a manual verification to benchmark against. As we will see later, a balance has been struck between required characteristics and feasibility for manual verification.

5.1.3. Proof of concept: automated verification of requirements using netPropCheck

As counterpart to the benchmark, a software implementation of the proposal presented in Chapter 4 was created. The resulting binary, called netPropCheck is able to perform computation and validation of properties using a network description in NML's OWL (or XML/RDF) format[1].

netPropCheck is written in Haskell and includes the four algorithms presented in Chapter 4. In addition, it features a Haskell implementation of the multi-layer breadth first search algorithm chosen in Chapter 2, adapted to work on the topology graph defined in Section 4.1.3. For verification, netPropCheck produces a boolean result, with the option to inspect the results of computation or path selection. netPropCheck is open source software, available under a BSD 3-clause license[2]. Key parts of netPropCheck are included in Appendix C.

5.1.4. Comparing results and reaching conclusions

The final part to the description of our validation methodology deals with results. Our working hypothesis is that automated verification of operational requirements is possible in a real world network with netPropCheck. If manual verification yields the same results for tests, we conclude that the hypothesis is true. If there are differences, we reject the hypothesis. For results that are not identical, we attempt to analyse and describe the reason for the difference. This should help constructing new working hypotheses and attempts at validation in the future.

We continue with the first preparatory step to perform validation by identifying the characteristics of a network suitable to serve as input.

[1] With respect to the cost function and control criterion it must be noted that only the cases described in this chapter are currently available in the code base.

[2] netPropCheck is available from `https://rtsn.nl/thesis/`

5.2. Choice of network

In order for the demonstration of netPropCheck to provide convincing evidence that our proposal works in the real network, a network is needed that is representative for the class of networks this research is aimed at. In this section, we list the different properties such a network should possess.

To get a grip on the class of networks that benefit from automated property verification, we provide a quick summary of network characterisation so far. Returning to Chapter 1, the problem description stated the need for verification scalable up to multiple technology layers. In the literature survey covered in Chapter 2, the search was scoped to cover *"networks complex enough to warrant automated reasoning. A method needs to be powerful enough to describe topologies at least the size of university campus networks"*. The characterisation of networks benefiting from this work was further reinforced by the two examples used throughout Chapter 3, which covered two types of networks that warrant automated as opposed to manual verification: networks complex enough (e.g., a telecom operator, Vodafone) or critical enough (e.g. a certificate authority, DigiNotar).

We keep this latest distinction as our first characterisation of a network suitable as input to our proof of concept: a network that is either complex enough or critical enough to warrant automated as opposed to manual verification.

The second required characteristic of the class of networks benefiting from this work is that they have clear real world requirements. Without defined requirements it is impossible to do computation, let alone validation of the outcome. In Chapter 3 four distinct requirements were identified. In particular, for the proof of concept it would be most beneficial to have a network which had real world requirements corresponding to all four identified types of requirements as covered in this work

Finally, in order to do verification of a network, the network needs to be described in enough detail to do automated reasoning. Ideally, a network would have such a description available, but in any case such a description should be obtainable. In both cases it is highly preferable that this description is automatically generated from existing network management systems, to guarantee a continuous and correct relation between description and reality.

In summary, this section motivated the need for a representative choice of networks as input to a convincing proof of concept. In order, the need for a network sufficiently complex or critical; the definition of clear requirements and the availability of a network description were listed as characteristics of a representative network. In the next section the choice for a sample network is presented, selected to conform to these criteria.

population	
connected working population	20 buildings, 12.000 people
connected residential population	12 locations, 2.800 people
technology	
core network	L3 routed core, 6 routers
	L2 transports using MPLS
distribution network	32 distribution switches
wired access network	320 access switches
wireless access network	1250 access points
fiber plant	4 private fiber rings on campus
	multiple leased circuits off campus

Table 5.1.: Campus network statistics (numbers are approximations)

5.3. Sample network

Selecting a sample network did not prove to be easy. As noted in the literature study of Chapter 2, there are few networks in the world working on (let alone using) formal network description. A search among networks and research institutions involved in the standardisation of NML unfortunately did not lead to better results. In other words, we were unable to find a network that had a formal description available.

Consequently, the only option available was to find a network operator that was willing and able to create such a description from scratch and possibly assist in this endeavour. The process of obtaining this description is described in its own subsection. We will first proceed to describe the selected network and motivate why it fits the selection characteristics formulated in the previous section.

5.3.1. Campus network

The network selected as input to our proof of concept is the network of a Dutch university (hereafter "campus network"). It connects all the universities research and education facilities and additionally supplies private internet access to residential buildings for students and staff. Table 5.1 lists some approximate numbers to get a feel for the full network scale.

The campus network uses technology from multiple vendors at different layers in the stack. In turn, this means that none of the vendors involved is able to deliver their own solution to manage the entire network. Instead, custom built information systems have evolved over the years to fulfil management and reporting needs.

☐ L3 datagram protocols (IPv4 and IPv6)
secondary stack
☑ Ethernet
☐ MPLS pseudowires (one per vlan)
primary stack
☑ 802.1Q (vlans)
☐ spanning tree protocols or ☑ multi-chasis LACP
☑ Ethernet
☑ fiber or ☑ UTP or ☑ misc. physical layer transports
☑ fiber ring (ducts, hand holes)

Table 5.2.: Technology stack in use at campus network. Checkboxes indicate whether the technology was described in the NML generated.

Table 5.2 lists the primary technology stack in use throughout most of the network. To combine a routed core network with the need to provide end-to-end circuits, MPLS pseudowires [15] are used to establish a layer 2 transport through the core. The devices involved in these circuits consequently use a secondary stack, on top of the primary stack. In the table, for each of the technologies in the stack a check denotes its inclusion in the network description, whereas an empty box denotes its exclusion. More detail is provided in the subsection on generation.

Although, there was no existing mechanism in place to document, monitor or reason about the infrastructure[3], the IT staff of the campus network were able to provide machine-readable information on most of the layers. The missing information could be obtained by inference. We will now turn to an explanation of the suitability of the campus network in terms of the previously defined characteristics.

5.3.2. Fit with characteristics

As noted in Section 5.2, for a network to provide convincing evidence in a proof of concept, it needs to adhere to three criteria. In this subsection we will explain how the campus network meets these criteria.

The first required characteristic of a network was that it is either complex or critical enough to warrant automatic verification. With over ten thousand people dependent on the campus network to do their work or alternatively almost three thousand households to keep connected to the Internet, it could be argued that the network has a critical

[3] I.e. apart from monitoring the upper layers and consequently assuming that the lower layers must be working.

function[4]. Considering complexity, the multi-vendor technology making up the campus network and the resulting custom management systems account for an above-average complexity. In addition, the scale of the network (both in terms of infrastructure and locations) make it difficult to reason about it in its entirety. Both are good arguments to classify the campus network as sufficiently complex to serve as a representative case for our proof of concept.

The second required characteristic of a network was the existence of clear real world requirements about network behaviour. These requirements exist within the campus network due to, amongst other things: separation of roles between student and IT staff (segmentation), a healthy view on oversubscription (control) and a pursuit of low impact downtime (time to recovery, path diversity). We will cover the specifics of each requirement in Section 5.4. In general, the availability of clear requirements is more than sufficient to make the campus network meet this requirement.

The final required characteristic is the availability of a sufficiently detailed description to enable requirement verification. Although there was no pre-existing formal description available for the campus network, existing management systems providing partial information were in place to provide partial topology information. It was the availability of and ease of access to machine-readable information on most individual network layers that guided the choice for the campus network over an available alternative: a national research and education network. The generation of a topology description for the campus network is the subject of the next subsection. With all requirements covered, we conclude that the campus network fits the required characteristics which make it suitable for our proof of concept.

5.3.3. Manual generation of a network topology description

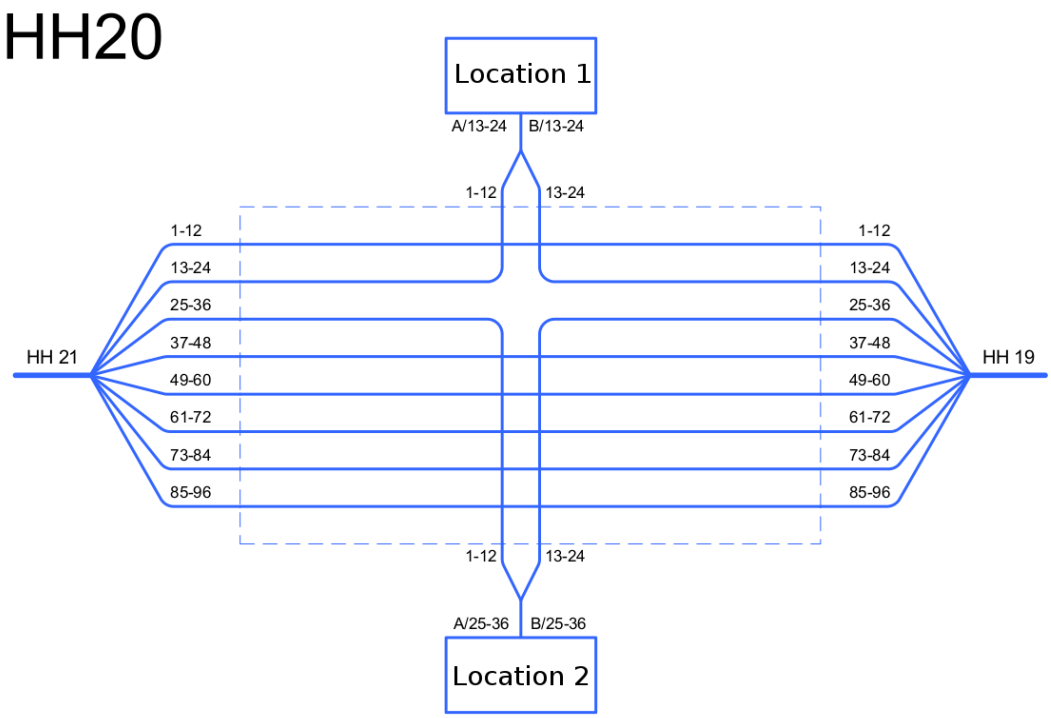
In this section we describe the process of formally describing the campus network. As such, this section would not exist if the description would have been available.

To structure this subsection, we return to Table 5.2. Each of the technology layers described (checkbox checked) was obtained from a separate management system and/or information source, with the exception of Ethernet and 802.1Q (vlans) which came from the same source. Subsequently, we will cover the description of each layer in a separate subsection, moving up the stack from the lower to the higher layers.

Fiber ring

When talking about a fiber ring, we mean two things. First, there is a tube or duct, tracing a ring across campus, either from and to one data centre location, or from one

[4]Naturally, the notion of criticality is relative, with a nuclear power plant likely having a more critical network. In the end, critical is most likely defined in terms of loss, a subject we won't get into here.



(Hand hole diagram from the campus network's fiber ring documentation, used with permission.)

Figure 5.1.: Schematic for hand hole 22, on one of the campus network's fiber rings

data centre to the other[5]. At set locations, the duct is accessible in a so-called hand hole. Secondly, there is the actual fiber. On the campus network, all ducts contain a bundle of 96 fibers. At hand hole locations, a (possible empty) subset of these 96 fibers is branched of and subsequently returned to the ring. Consequently, every hand hole has 0 to 192 "fiber patches" on the ring. Figure 5.1 illustrates the concept of a hand hole using a real schematic from the campus network.

In practice, the data centres terminate 96 fibers, while each of the hand holes has zero or more multiples of 12 fibers and double that amount of fiber patches. The configuration per hand hole is more or less static, with a change requiring fiber splicing. However, by connecting two fiber patches, the ring is once again continued. With multiple hand holes branching of the same subset of fibers, this is a way to "patch through". This fiber patching is a more dynamic form of configuration, making the network topology partly reconfigurable at its lowest layer. At this point of time, this reconfiguration happens manually.

To keep track of the "static" and "dynamic" configuration of the four available fiber rings on campus, the IT staff involved keeps a digital administration of the fiber plant. A design document was available on the static configuration, while the dynamic "patch" configuration was essentially kept as the tuples `((location, patch port), (location, patch port))` in a spreadsheet. Some of these pairs were annotated with a connected device on each of the locations, enabling coupling with higher layers.

Although all four fiber rings used the same documentation style, only the ring connecting the residential buildings was consistent enough in its description to make automated parsing feasible within a two week development time frame[6]. For that reason, only one ring is modelled, connecting 12 (mostly residential) locations on campus to the two data centres.

The generated NML for the fiber ring, including hand holes, patch panels and patching consists of 12 NML Nodes, 5,000 NML Ports, 2,700 NML Links for a total of 31,500 triples in an N-Triple representation of NML's XML/RDF format. Each triple is a three element tuple (subject, predicate, object) representing a labelled relation between subject and object. Consequently, the number of triples provides an indication of the amount of information contained in an NML description.

Physical layer transports

The next layer up from the fiber rings cover physical devices and links, other than provided in the fiber rings. This translates to fiber patches, UTP cabling, switches,

[5] Although this technically would not correspond to a circle, the two data centres are close enough that the resulting U-shape resembles a circle. With the two data centres connected by a direct fiber path, the result is called a ring.

[6] With the complexity of the interconnections, manual modelling was essentially impossible. If not for the amount of work, then due the lack of correctness.

routers and miscellaneous devices and links.

The IT staff manages this layer in an information system that collects its data directly from the network using SNMP, configuration dumps and other management protocols. Export was possible through the Graphviz dot format, a graph format used within the information system to draw topology maps. Every device in the information system (routers, switches, access points) is represented in this topology, as are the links between them, including the associated port numbers. This export allows the generation of NML for devices and physical links. Unfortunately it only includes a single link between devices, even if there are multiple links in reality. This has implications for path diversity verification, a fact we will get back to later.

Coupling with the fiber ring was possible by querying the NML generated previously to identify devices connected through the fiber ring. For those connections, the fiber ring was used instead of generating a new NML Link. Consequently, this means that new information on additional fiber rings can be plugged into this generation step, and its infrastructure automatically used by the physical layer. Distinction between fiber, copper and miscellaneous technology used was not made at this point, but was retroactively supplied in the Ethernet and 802.1Q layer by use of interface type (e.g. 1000BASE-LX vs. 100/1000BASE-T).

The generated NML for the physical layer, including devices and patches on the residential part of the campus network consists of 1,300 NML Nodes, 2,000 NML Ports, 1,000 NML Links for a total of 9,600 triples in an N-Triple representation of NML's XML/RDF format.

Ethernet and 802.1Q (vlans)

With all physical links and devices in place, the next step would be to model the configuration of the switching fabric for each of these devices, including the use of 802.1Q vlans.

For this purpose, a database export was made to provide tuples `(device, port, vlans)` amongst other information. This allowed the creation of Ports for vlans and their interconnection, as well as the correct interconnection of untagged (bare Ethernet) ports. In addition, where applicable, logical ports (possibly including vlans) were created, to be connected as LACP trunks in a later generation phase. The database export additionally provided information on spanning tree state, a technology layer we decided not to include to keep the generation work within bounds. Additionally, blocking ports, courtesy of STP state, would only decrease the complexity of the network and thus yield a weaker result for the proof of concept.

The generated NML for the Ethernet/802.1Q layer, including switching fabrics and vlans consists of 3,000 NML Ports and 400 NML Links for a total of 27,000 triples in an N-Triple representation of NML's XML/RDF format.

LACP and multi-chasis LACP

At this point, large parts of the network were sufficiently modelled to allow path selection, with exception of the relatively new residential part of the campus network. There, a technology most commonly known as multi-chasis LACP is used to provide interconnections between each residential distribution and the two core distribution switches, providing active-active loop-free redundant interconnection on Ethernet level. Without modelling this technology, path selection would not find a path between the residential access switches and the core.

Using exports of the physical-logical port mapping for each device, the logical port created during the previous generation step were connected to the physical ports providing the actual connection. Although not strictly necessary, normal LACP trunks were included in this process, because they essentially came for free with the current implementation.

The generated NML for the LACP layer, connecting logical to physical ports, consists of 90 NML Links for a total of 270 triples in an N-Triple representation of NML's XML/RDF format.

Summary and general observations

In summary, the generated NML for the campus network, including all previously described and their interrelations consists of 1,300 NML Nodes, 10,000 NML Ports and 4,000 NML Links for a total of 68,000 triples in an N-Triple representation of NML's XML/RDF format. This network topology description is subsequently used throughout this chapter. But before moving on to describe the campus network's sample requirements, we state some observations made during the generation process.

Modelling a real world network was not originally intended within the scope of this research. However, lacking any real world network descriptions for use with netPropCheck, no other option was available than to dive in and create one from scratch. Modelling layer interactions, matching information and writing parsers took almost a month, making it no trivial investment to enable research in this area.

Because we suspect that the lack of availability of real world datasets hampers research in this area, we are proud to make the generated network topology available as an open dataset. In addition, we intend to share a small wrapper-library around NML that is not specific to the campus network but applicable to NML generation in general to help simplify future research[7].

The IT staff of the campus network have expressed interest in examining NML for applicability as description format for a next generation network management system. To this end, it is our intention to share back all software written explicitly for NML

[7]The available software and the open dataset can be found on `https://rtsn.nl/thesis/`

generation of the campus network. This software will not be shared publicly at this point. Naturally, the inclusion of the requirement verification described in this research would become very practical once the network is natively described in a formal description language.

5.4. Sample requirements

Having motivated and described the network chosen to serve as input to our proof of concept, we now proceed to describe real world requirements that match each of the four requirements described throughout this work.

The campus network has several implicit assumptions which lend themselves for formalisation as an operational requirement. In each of the next subsections we will translate a real world assumption or expectation into an operational requirement. In order, we cover segmentation, control, time to recovery, and path diversity.

5.4.1. Segmentation

A real world use-case for the segmentation property can be found in the separation of production and management segments of the campus network. We take the example of a residential building, equipped with an edge-switch and multiple access points. Although residents have network access through both types of devices, the devices themselves are also accessible from the network for management purposes. Our chosen segmentation requirement states that a user connected to the switch cannot reach the management network of the access point on the same floor.

5.4.2. Control

To show the versatility of the ability to lift element specific requirements to the abstraction level of a full topology, we choose a non-standard use-case for the control property. Specifically, we define the requirement that traffic from a residential end user to the core network only uses elements with a bandwidth capacity greater than or equal to the bandwidth capacity of the edge port. In other words, we validate that a single edge-user is unable to saturate the uplink, thereby blocking access for his fellow users.

5.4.3. Time to recovery

Table 5.3 shows recovery times for different device roles in the residential subset of the campus network. Using these recovery times, we want to compute the worst case downtime for a single device failure in the residential path to the internet. In particular, we compute the time to recovery for the path between a host connected to a residential

devices			*links*	
building level	access switch	4 hours	patch/cable	1 hour
quarter level	distribution switch	3-4 days	fiber ring	3 days
campus level	residential core switch	3-4 days		
	core router	5 days		

Table 5.3.: Recovery times per device role for residential subset of the campus network, assuming full hardware failure, as estimated by the campus network's IT staff (numbers are approximate)

access switch and one of the core routers routing the residential subset of the campus network.

We define a recovery time of one day acceptable for the type of outage which requires full replacement of hardware, which will serve as our requirement to test against.

5.4.4. Path diversity

For the same residential to core connection, we check the number of redundant paths available. Combined with the requirement for time to recovery this allows quantification of failure modes.

The residential subset of the campus network has been designed with redundancy in mind. Past the access switches, the entire infrastructure should be redundant. We therefore check whether the infrastructure between a residential access switch and the core network does indeed provide a minimum of two independent paths.

5.4.5. Conversion of the operational requirements

Having described the operational requirements selected for the campus network in general terms, we will now specifically detail the conversion of each requirement for verification with netPropCheck. To this end, we briefly return to each of the four requirements.

segmentation In the case of segmentation, the conversion is relatively straightforward. We have modelled a system and connected it to a residential edge switch manually to the campus network topology. Moreover, the wireless access point located closest to the system is identified. To check the segmentation property, we verify segmentation between the outbound interface of the system and inbound interface on the management vlan of the wireless access point.

control To check the segmentation property, we again use the residential system used before, this time with a core router as counterpart. Connection speeds are determined by using the encoding property on NML Ports, information which is contained in the topology description. The different encodings on the physical level are used as an estimator for link speeds. The connection speed of the system's interface is obtained and subsequently compared to all NML ports on the physical layer on the path to the core, forming the control criterion. Because we are only interested in the ability of the user to directly hinder other residential users through direct link saturation, this property is only checked in one direction.

time to recovery The time to recovery property is checked by using the information contained in Table 5.3. We define a recovery cost function by assigning to every NML Port and Link a recovery cost value. The mapping between a switch and its respective device class is given as input to the computation. The devices involved are once again the system connected to a residential edge switch and the core router. The required worst case recovery time is specified as 24 hours.

path diversity Although we stated the intention to check the number of independent paths between the residential system and a core router, this is not possible with the generated topology description. As indicated in Section 5.3.3, the data provided on the physical layer only includes a single connection per pair of devices. And although the residential access switches are connected to the quarter level distribution switches using two distinct links, this gap in the data does not allow us to verify their impact on the overall path diversity. Consequently, we choose to verify the number of distinct paths between the quarter level distribution switch and one of the core routers.

Moreover, to work around an implementation choice for the path selection algorithm combined with the lack of an NML construct for trunking/bonding, the path The required path diversity is specified as 2 independent paths.

5.5. Results

In this section we list the results for each of the four requirements. For every requirement, we present the results of manual verification and verification using the proof of concept.

Because the full output of netPropCheck would increase this thesis by 200 pages, only the non-verbose output is shown. Full (verbose) output is available for download[8]. In order to keep the naming scheme of the campus network's topology confidential, both the summary and the full version of the output have anonymized URIs and lack names of network elements. Because NML defines URIs to be opaque and name attributes

[8] `http://rtsn.nl/thesis/`

to be optional this does not influence the correctness of the topology description nor the results in any way. Note that the naming of the non-anonymized version provides a human with meaningful clues as to the network structure. An example of a naming scheme can be seen in the sample NML included as Appendix C.1.

Execution time for netPropCheck varied between 7 to 10 seconds for each property. Most of this time was spent reading and parsing the 5 megabyte topology description. These numbers were constant over all performed test runs.

5.5.1. Segmentation

manual verification For both the residential system and the access point possible paths were traced towards devices in the network where an interconnection between the two was possible. These included the core routers and every switch on the way there. No such interconnection was found, leading us to the conclusion that the segmentation property holds.

netPropCheck Verification using netPropCheck yielded segmentation in both directions[9].

SEGMENTATION
segmentation
netPropCheck /tmp/campusnetwork.nml segmentation −s urn:ogf:network:rtsn.nl:thesis:2014:7cf018cd671f55e27f65 −d urn:ogf:network:rtsn.nl:thesis:2014:208b8d3485b906c25dfa

True
segmentation (**reverse** direction)
netPropCheck /tmp/campusnetwork.nml segmentation −s urn:ogf:network:rtsn.nl:thesis:2014:89dc602d1e5e62228654 −d urn:ogf:network:rtsn.nl:thesis:2014:29ca57c71031eefedc94

True

5.5.2. Control

manual verification Manual verification was performed by tracing the path between the residential system and the core router, checking the link speeds for each link along the way. The manual verification was made easier by the knowledge that all equipment from quarter level distribution up to core routers were connected by one or multiple 10 Gigabit links. The conclusion of manual verification was that the defined control property holds.

[9] As noted in Section 4.1.2, both directions are verified independently and then combined to yield bidirectional results.

netPropCheck Verification using netPropCheck yielded control in the single direction verified.

```
CONTROL
control
netPropCheck /tmp/campusnetwork.nml control -s urn:ogf:network:rtsn.nl:thesis:2014:8555
    d3fc19499a0d152f -d urn:ogf:network:rtsn.nl:thesis:2014:6cac7451a240ce3b787b
Please enter the required minimum speed
1000

True
```

5.5.3. Time to recovery

manual verification Manually verifying time to recovery would be extremely tedious if it were not for the fact that the mostly hierarchical structure of the residential part of the campus network and the rising recovery times closer to the core make it easy for a human to draw conclusions. With a router as destination and no device or link class with a higher recovery time, it is trivial to see that this requirement does not hold.

netPropCheck The verification of time to recovery by netPropCheck yielded two failures. The output shows that the most costly element, one of the routers, takes five times longer than the required 24 hours. This means that if redundancy were to fail, the failure of the router in question would break the time to recovery requirement of 24 hours.

```
TIME TO RECOVERY
time to recovery
netPropCheck /tmp/campusnetwork.nml timetorecovery -s urn:ogf:network:rtsn.nl:thesis
    :2014:8555d3fc19499a0d152f -d urn:ogf:network:rtsn.nl:thesis:2014:6cac7451a240ce3b787b
Please enter the required time to recovery as an integer
24

("sample NetworkObject with worst recovery cost","(Single Port \"urn:ogf:network:rtsn.nl:
    thesis:2014:0ed81df5fcf55b561de5\",120)")

False

time to recovery (reverse)
netPropCheck /tmp/campusnetwork.nml timetorecovery -s urn:ogf:network:rtsn.nl:thesis
    :2014:c54fb2fc4fceac3cbcdd -d urn:ogf:network:rtsn.nl:thesis:2014:99946f53fa645a894297
Please enter the required time to recovery as an integer
24

("sample NetworkObject with worst recovery cost","(Single Port \"urn:ogf:network:rtsn.nl:
    thesis:2014:16b2053b7b01c42f997b\",120)")
```

False

5.5.4. Path diversity

manual verification Computing an upper bound for path diversity by hand is almost impossible on the campus network, even access to topology maps and management information. However, finding two independent paths proved to be relatively simple. The conclusion of manual verification is that the path diversity requirement holds.

netPropCheck netPropCheck yielded two distinct paths for both directions.

PATH DIVERSITY
path diversity
netPropCheck /tmp/campusnetwork.nml diversity −s urn:ogf:network:rtsn.nl:thesis:2014:
e1bb9c1d50168cb828d0 −d urn:ogf:network:rtsn.nl:thesis:2014:6cac7451a240ce3b787b
Please enter the required path diversity as an integer
2

("number_of_distinct_paths","2")

True

path diversity (**reverse**)
netPropCheck /tmp/campusnetwork.nml diversity −s urn:ogf:network:rtsn.nl:thesis:2014:
c54fb2fc4fceac3cbcdd −d urn:ogf:network:rtsn.nl:thesis:2014:1b6396ef10effb675789
Please enter the required path diversity as an integer
2

("number_of_distinct_paths","2")

True

5.6. Validation summary

With four equal results between the manual verification and automated verification using netPropCheck, following our methodology, we accept our working hypothesis. We conclude that automated verification of operational requirements is possible in a real world network with netPropCheck.

In summary, this chapter answered subquestion 5:

> *How can the results be validated?*

Two methods were demonstrated to validate our proposal for verification of requirements in multi-layer networks: a formal method and an empirical method. The formal method was explained in Chapter 4 and follows implicitly from the construction of the proposal, as was illustrated in Figure 4.6.

The empirical method was covered in this chapter. A proof of concept was built and used to verify properties in a real world network. With the results matching those of manual verification, we conclude that it is possible to verify operational requirements in a multi-layer network via an algorithm for path selection in its topology description.

Chapter 6.

Conclusions

In this final chapter, we summarize the results of previous chapters, comment on future work and conclude this work by answering the main research question.

6.1. Summary of results

This section summarizes the answers to the subquestions covered in Chapters 2 to 5. The subquestions are covered in the ordering of Chapter 1, which does not correspond fully to the chapter ordering. Therefore we will indicate the corresponding chapter for each subquestion summarized.

1. *How do we describe multi-layer network topologies?*

   In the first part of Chapter 2, a previous survey by the authors covering the state of the art in academic literature with respect to the architectural description of networks was recapitulated. The Network Markup Language was selected as the topology description language of choice in this research.

2. *What operational requirements can we relate to using topology descriptions?*

   Chapter 3 identified four distinct generalized operational requirements. Each of these relate to either graph theoretic properties or generalized node or link specific properties. In general, the chapter conjectured that these two classes of properties are especially suitable for verification solely based on network topology information.

3. *Is it possible to formulate checks for operational requirements in terms of path selection?*

   Barring some limitations, a proposed composition between computation and validation presented in Chapter 4 shows that it is possible to formulate checks for operational requirements in terms of path selection. This claim is illustrated with the presentation of algorithms for four generic operational requirements.

4. *What multi-layer path selection algorithms are available?*

   The second part of Chapter 2 listed the different approaches to multi-layer path selection. A multi-layer variant to the Breadth first search algorithm was picked and adapted for use in a proof of concept.

5. *How can the results be validated?*

   Two methods were demonstrated to validate the proposal in multi-layer networks: a formal method and an empirical method. The formal method was explained in Chapter 4, while the empirical method was subject of Chapter 5. We conclude that it is possible to verify operational requirements in a multi-layer network via an algorithm for path selection in its topology description.

With all subquestions answered, we are now ready to answer the full research question. But before we conclude, we first look at future work.

6.2. Future work

We have made references to limits in scope, possible improvements or future extensions throughout this research. In this section, these references are collected and summarized, serving as a basis for future work in this area.

NML and formal network description

From the perspective of requirement verification in networks, there are several possible improvements to NML and the surrounding tooling to simplify network description in general.

With respect to standardisation, the author deems it helpful to standardise technology identifiers for use with the attributes `encoding` and `adaptationFunction`. Even better would be the availability of entire Technology Schema's as were available for NDLv2 [18] to simplify description of common technology layers. Other helpful standardisation work would include the ability to explicitly describe "trunking", as already alluded to in the NML standard as the experimental `ParallelCompound` extension. All three would help towards simplifying the creation and exchange of network descriptions in NML, which ultimately determines the success of verification checking based on such descriptions.

More generally, if formal network description is to succeed as a concept beyond specific application areas such as optical light exchanges, there is a need for the development of more use-cases for their application. This work provides one such use case, but more examples (preferably including an exploration of the economic benefits) are needed before there any chance of broader adoption.

In the area of tooling, interesting topics for further research are visualisation of formal network descriptions and their use in interoperability between management systems. The lack of research in both areas came up during development of our proof of concept, respectively for use during debugging and demonstration; and to ease the export and generation of NML from the campus network's systems.

Requirement verification using topology descriptions

We see several interesting areas to continue research on the topic of requirement verification using topology descriptions. In order, we list improvements to the algorithms described in this work; the extension of this work to more or different requirements; and the promising idea of switching to per-layer path selection algorithms for verification.

The algorithms for computation and the customized path selection algorithm based on [10] are all open for improvement. The authors have limited expertise in complexity analysis of algorithms (in particular, as implemented in a functional language such as Haskell) and have therefore not included such analysis. Formal complexity analysis of the algorithms in this work and time and space profiling of their behaviour under varying inputs is consequently an open topic. Moreover, the authors believe that the implementation of the path selection algorithm is suitable to be parallelised, which would make sense when doing verification on increasingly larger network topologies.

In Chapter 3, it was decided to only cover requirements that did not need secondary information sources (i.e. information outside the topology description). However, this work does not preclude the specification of requirements using such sources, which consequently constitutes a possible future extension of this work. Another way to extend our current set of requirements is to define ways of stacking their respective computations before performing one or more validations, effectively enabling the verification of composite requirements, much like you would do with the combined information on e.g. time to recovery and path diversity in practice.

Finally, we would have liked to swap out the generic multi-layer path selection algorithm, used throughout this work, for a per-layer instance. These instances would be specific to the technology layer, mimicking the way actual path selection is done in the protocols on that layer. This also opens the way to technologies which are currently not very suitable for description in NML due to their reliance on e.g. routing tables. The specific path selection algorithm on the respective layers would use routing information (external to NML) to determine a path through the graph, while the overall topology can still be described as in NML itself.

This concludes our overview of future research. This chapter continues with conclusions.

6.3. Conclusion

This work was started with the goal of increasing insight into the advantages of formal network description as a method to manage the mounting complexity in network stacks. We advanced towards this goal by exploring one compelling use for these descriptions: continuous, real-time and automated verification of operational requirements in computer networks, answering the main research question:

"Is it possible to verify operational requirements in multi-layer networks via algorithms for path selection in their topology descriptions?"

We have found that there are topology descriptions languages available that allow formal description of networks for the purpose of reasoning. We have shown that it is possible to formally define operational requirements using a graph definition obtained from such descriptions. Using this formal definition we have synthesised verification algorithms to test conformance to such a requirement. These algorithms have been demonstrated to decompose in multi-layer path selection and verification. And finally, we have shown a working proof of concept which puts these ideas in practice in a real world network.

Our conclusion: it is possible to verify operational requirements in multi-layer networks via algorithms for path selection in their topology descriptions.

Appendix A.

Literature review (research topics)

The Architectural Description of Networks, a Survey

research topics in Telematics

Maarten Aertsen <maarten@rtsn.nl>

August 8, 2013

This research summarizes the current state of the art in academic literature with respect to the architectural description of networks. We find that formal network architecture description is a small area of research, primarily driven by grid and cloud computing needs. There are a handful of languages available, each serving different specific needs. There is no wide adoption of any of the languages found, which leads to a selection based on our specific needs. For the purpose of future research in reasoning about the characteristics of topology we find that NML is most suitable as formal topology description language.

1 Introduction

Reasoning about the characteristics of a topology in relation to its functional requirements calls for a concise description of both. This research is the first step towards this goal by reviewing existing approaches to the description of operational requirements and network architecture.

We present answers to research questions 1 and 2 posed in the author's research proposal[1]:

1. *How do we represent operational requirements for networks?*
   (a) *What are operational requirements?*
   (b) *How are operational requirements formally described?*
2. *How do we represent the architecture of networks?*
   (a) *What characteristics contribute towards their operational requirements?*
   (b) *How is a network architecture formally described?*

Two methods are used to answer these questions: to satisfy questions 1 and 2a we identify real world needs by clients of Deloitte. To answer question 2b a literature survey is performed, which is the main focus of this work[2].

First, we narrow down the scope of question 2b by defining its terms. By *network* we denote a computer or telecommunication network, in *architecture* our focus is on physical topology, in particular OSI layer 1-3. With *formally* we mean suitable for calculation or reasoning. Finally, when talking about *described* we are looking for something which is exchangeable and machine-readable.

This paper is organized as follows. In Section 2 we describe the methodology used for the survey. Section 3 presents its results, followed by an analysis of the findings relevant for topology analysis in Section 4. We conclude the survey with a discussion in Section 5 and conclusions in Section 6.

2 Methodology

During initial informal exploration of literature it became apparent that network topology description is not a discipline on itself, but rather a foundation to build on, signified by small (loosely connected) islands of research. To perform a meaningful survey, the choice was made to select a big sample without regard for connectedness and subsequently filter down to a "multi-island" set from which to proceed to discover related research.

The actual methodology used is loosely based on the guidelines by Wolfswinkel et al. [13] and Webster and Watson [12]. In particular, we follow the process of Wolfswinkel et al. for conducting the search and paper selection, reproduced in Table 1 for the convenience of the reader. In the following paragraphs we will discuss the choices made during the search and select phase, corresponding to tasks

[1] *"Parity analysis in telecommunication networks: architecture characteristics versus operational requirements"*, master thesis in Telematics, research proposal. Maarten Aertsen, April 2013.

[2] The survey falls within the domain of research topics. Questions 1 and 2a are out of scope for this survey but are nevertheless important to address and covered in the Appendix.

#	**Task** (sequential and partially iterative)
1	Define scope (including criteria for inclusion)
2	Identify fields of research
3	Find corresponding databases/outlets
4	Define search terms
5	Search
6	Filter out doubles
7	Cut down sample based on title + abstract
8	Cut down sample based on full text
9	Forward and backward citations
10	Verify final dataset

Table 1: SEARCH & SELECT phase by Wolfswinkel et al. [13]

one to four in Table 1.

scope The definition of scope is directly related to the definition of terms in the question. The survey intends to summarize current methods to describe network architecture in a formal way. In addition to the limits established by our definition of terms, we limit our search to methods that allow for description of network complex enough to warrant automated reasoning. A method needs to be powerful enough to describe topologies at least the size of university campus networks.

fields of research In the beginning of this section we identified topology description as a foundation to build on. It could therefore be the case that research is embedded in multiple fields, widening the scope for the survey. We argue that publications on complex networks require specialised IT staff. IT staff is in turn most likely to publish results in either computer science or telecommunication. We consequently assume that publications are made in the fields of computer science and telecommunications.

databases/outlets To obtain a large sample of papers, three separate databases were queried. Without knowledge about the existence of specific journals or conference proceedings (outlets), the selection of search engines was essentially random. Two general purpose engines: Web of Science (Reuters) and Scopus (Elsevier) were used, supplemented by a search engine focussing specifically on computer science: DBLP. Google scholar was used to obtain information on forward and backward citation.

search terms The chosen search terms correspond to the definitions given in Section 1, with the addition of synonyms to match different wordings. During preliminary inspection of the Web of Science results, it was noticed that the search terms matched numerous papers relating to neural networks. It was therefore decided to add additional filtering for subsequent database queries. Listings 1, 2 and 3 give the search terms used for each search engine.

Search terms 1 Web of Science

```
    SU=(telecommunication OR
        computer science)
AND TI=network
AND TS=(topolog*          OR
        architect*        OR
        infrastruct*)
AND TI=(format            OR
        language          OR
        specification     OR
        description)

DocType=All document types;
Language=All languages;
```

Search terms 2 Scopus

```
SUBJAREA(comp) AND
TITLE("network") AND
(
  TITLE-ABS-KEY(topolog*) OR
  TITLE-ABS-KEY(architect*) OR
  TITLE-ABS-KEY(infrastruct*)
) AND
(
  TITLE("format") OR
  TITLE("language") OR
  TITLE("specification") OR
  TITLE("description")
) AND
NOT TITLE("neural")
```

Search terms 3 DBLP

```
network
topolog~|architect~|infrastruct~
format|language|specification|description
-neural
-natural.language
```

query broken into multiple lines, newlines denote binary AND

filtering of doubles Both Web of Science and Scopus support export of meta-data rich results in (extended) bibtex format, including everything but the paper itself. DBLP unfortunately offered no option to export meta-data rich results, its XML and JSON(P) export formats only cover basic fields as author, title and venue. Automated bibtex parsing

was performed on the results, followed by fuzzy title comparison to rudimentarily filter residual doubles from both samples[3].

manual filtering steps To further cut down the sample, selections on title, abstract and full text were made. The first two filtering steps used summary output from the scripts; only the last step involved actually fetching the papers itself. This approach allowed a last-minute call on the usefulness of a paper; waiting with the rejection of moderately interesting papers until it was beyond doubt that they were no fit for the survey.
The reduced export capabilities of DBLP proved troublesome: filtering on abstract required locating and downloading the paper itself, infeasible for hundreds of results. To reduce the burden, a further pre-selection was done on the DBLP dataset by keyword scoring on their titles. Popular keywords from the sample known not to be of interest would be applied as filter, with this trick repeated until there were no obvious unrelated keyword left. We will return to this method and its consequences in the discussion (Section 5).

final verification Wolfswinkel [13] assumes the ability of a researcher to verify a final dataset by discussing it with peers before proceeding into the processing of results. This proved to become challenging without existing local experience in architectural description and a similar lack in contacts in this area.
The reality was different. During the search, we came in contact with van der Ham [2, 4, 8, 7, 9] who kindly offered to comment on our findings. Later, he shared his ongoing work on a more general survey paper [9], previously unknown to the author. Although his survey greatly diminishes the value of this work, their similarity in findings give strong evidence for the validity of our resulting sample.

3 Results

We present the survey results in two parts. The first part describes all the findings, while the second discusses meta results such as the number of results, their interconnectedness and general observations. We start with descriptions of the research found. In Section 4 we discuss the relevance of the main findings in this section.

[3]To this end, some simple one-off scripts were written in the Python language. These were later extended to perform keyword scoring for DBLP (see next paragraph).

Findings

In this section we present all findings. The main findings are covered in one paragraph per description format. The remainder, related but not fully fitting our definitions, are summarized in a miscellaneous section. An attempt is made to link related research, this is also reflected in the ordering.

NDLv1 NDL [10] started as an ontology to describe the interconnection of network elements in the RDF language. Its goal was to establish a distributed "topology knowledge database", where network operators shared their topology information towards the common goal of provisioning grid networks. The advantage of using RDF (i.e. the semantic web) as foundation lies in its natural ability to refer to distributed data as if part of the same information structure. The original version of NDL features a limited set of relations ("properties" in RDF terms) to express topology: `locatedAt`, `hasInterface`, `connectedTo`, `description`, `name` and `switchedTo`.

NDLv2 In [8] the authors of NDLv1 describe efforts to extend NDL to allow the expression of multi transport layers, e.g. as used in hybrid optical/IP networks. Using concepts from the ITU-T G.805 architecture standard for transport networks [2], NDLv2 is a language capable of describing multilayer networks while avoiding explicit change to the base language for each new technology. This works by defining Technology Schemata capturing technology details on top of a more abstract Layer Schema and finally defining the topology in terms of the Technology Schemata. This enables the creation of algorithms to use the Layer definitions underlying each technology to trace paths, compute connectedness etc. without having to actually understand the technology in itself, or worse embed the technology in the NDLv2 design.

NML Following adoption of NDL in the context of optical and grid networking, a working group was formed to standardized a common language backed by the research on NDL, perfSONAR and other projects. This standardized language, dubbed "Network Markup Language", incorporates work focussed on resource request and monitoring. The resulting standard [7] is the preferred NDL/NML-style language for real world use. It is important to note that at the time of writing, the NML specification is hot of the press. Therefore some earlier work supplementing NDLv2, including Technology Schemata, have not yet been ported to

NML. It is therefore not entirely trivial to immediately start topology mapping in NML.

INDL While NML was being standardized, a parallel effort was ongoing to extend NML with additional capabilities intended to describe computing resources and virtualisation technology. This language, dubbed "Infrastructure and Network Description Language" is tailored for the management of grid and cloud technology [4]. The extensions proposed in INDL are of little use given the scope of this research.

VXDL The "Virtual Resources and Interconnection Networks Description Language" [5] is a specification language for virtual infrastructures. It allows for the description of a complete computing environment, including the topology that connects the different nodes. As a resource request language it abstracts from real topology and describes aggregated characteristics.

DEN-ng DEN-ng [**?**] is an evolution of "directory enabled networking", aiming to unify network management using business rules. This goal makes DEN-ng more a management model/paradigm than a network description, although it is definitely capable of being use that way. Its name stems from the use of (L)DAP-enabled backing stores. The use of freely definable ontologies make it less suitable as an exchange format and similarly the chosen abstraction is somewhat arbitrary, depending on the amount of information stored in the paradigm.

NNDL In [**?**], Dobrilovic et al. describe a novel XML based language to describe network nodes. This "Network Node Description Language" is primarily intended to describe virtual network test beds and allows description of services and detailed node configuration. The authors claim that NNDL is also usable as a more general description language for computer networks and its expressiveness for IP layer networking (including routing) is remarkable. However, there appears not to be any support for non-IP network layers, which makes NNDL unfit for our purposes.

perfSONAR "PERFormance Service Oriented Network monitoring ARchitecture" or perfSONAR is a service oriented architecture to standardize collaborative monitoring of networks [**?**]. Of interest is its topology component called "Topology Service". This component makes the topology information available for use throughout the service. The topology itself is described using an XML format standardized by the OpenGridForum's Network Management working group called `nmwgtopo`, describing interfaces, groups of interfaces (called links) and groups of links[4]. It is readily apparent that the topology service is considered to be a building block, with its own expressiveness severely limited to the grouping of interfaces for which statistics are collected. perfSONAR has been deployed in GEANT and connected National Research and Educational Networks.

miscellaneous We conclude the findings section with an overview of related work which did not fully meet our search requirements, but is still useful as related work. We describe an approach based on a geographical information system (GIS); a graphical format inspired by commercial drawing tools by HP/IBM; the DataCenter Markup Language; a description model for computational grids and conclude with FleRD, a description language with vagueness support.
Dumitrescu et al. [3] find that most ISPs and telecommunication providers describe their networks using general purpose mapping tools to combine topology information with geographical information. They note that these general purpose tools lack the ability to directly describe infrastructure, instead relying on the operator to "draw" the correct symbols. To this end they propose to use an ontology for network infrastructure to standardize, validate and simplify information input and storage. Unfortunately, no actual (base) ontology is standardized for possible use as exchange format.
In [11] Wang et al. describe the need to standardize a common format to enable interoperability between various proprietary network management applications. They present a prototype for such format based on XML and describe how to visualize their format using Scalable Vector Graphics. Unfortunately, the format has not risen beyond proof of concept and no subsequent work has been published.
DCML was an effort to standardize a description format for data center environments and their operations. Although backed by significant industry players, it has not evolved beyond an initial idea [1].
Lacour et al. [6] specifically target network description for computational grids, in order to facilitate automatic provisioning of the resources required to run user applications. The authors make special effort to enable the description of non-hierarchical networks and middleboxes (e.g. fire-

[4]Although this format appears not to have been standardized, the most recent (i.e. 2007) schemas can be find at `http://anonsvn.internet2.edu/svn/nmwg/trunk/nmwg/schema/rnc/topo/`.

walls, NAT translators), which makes their model useful for non-heterogeneous grids. Unfortunately, no real format is defined beyond some simple examples.
Schaffrath et al. take on the more general problem of communicating resource requirements between cloud providers, in order to provision such services on top of federated networks [**?**]. For this purpose they introduce FleRD, a flexible resource description language with support for "vagueness". FleRD has very generic network description support by using so-called "NetworkElement" and "NetworkInterfaces" objects. This is very much suited to requesting resources, but not so much for physical description of actual network infrastructure. The authors make the case for vagueness support in description languages as a way to scope information sharing and interpretation, resulting in overall decreased complexity.

Meta results

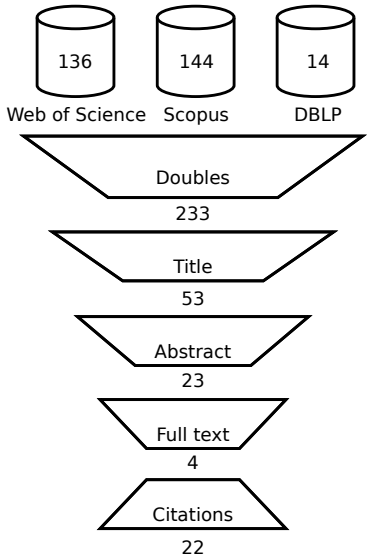


Figure 1: Decreasing and increasing the sample

Figure 1 presents numerical results on the search and select phase. Our first observation is the small number of papers resulting after full text reading. The subsequent rise in papers by tracing forward and reverse citations shows that keyword search has its limitations. However, it must be noted that most of the new papers found pertain to the same technology (NDL/NML) and are written by a select group of authors. The final sample and its inter-relations are shown in Figure 2, with papers on NDL/NML highlighted[5]

[5]This diagram was originally created to track progress during step 9 of Table 1 and later to make sense of the interconnections in the resulting sample. It later proved useful to integrate in the report due to the high connectedness of the NDL/NML part of the sample. It was created using the Graphviz suite of automatic graph drawing software, using data manually obtained from Google Scholar's citation indexes.

Of particular note is the relatively small outcome after the filter steps. This can best be explained by our choice to last-minute filter the set like we described in the paragraph on "manual filtering steps" in Section 2.
The small number of papers produced by the search query in DBLP can best be explained by its lack of support for complex queries. We suspect that its search facilities are better at interactive browsing (e.g. finding papers for known authors or outlets) than for a set of previously unconnected keywords. Later inspection showed that none of the 14 results output by DBLP survived the filter steps. We refer to the discussion for further coverage of DBLP's shortcomings.

4 Relevant findings

Having discussed the full set of findings and the existing relations therein, we now turn to the question of relevance. Using the definitions of *network*, *architecture*, *formally* and *described* as given in Section 1, we contrast all main findings to determine whether any of them is suitable for use in subsequent research.
Following these definitions, we contrast our findings on their layeredness (i.e. single vs. multi), the exchangeability (internal system format or designed for exchange), scope (what is being described) and technology neutrality (fixed capabilities or technology agnostic). Table 2 gives a compact summary of the main findings.
There are three languages which are multi layer, exchangeable, scoped to networks and technology neutral, highlighted in grey in the table. Among this family of languages, NML has been standardized and is considered most suitable for real world use. We conclude that NML is the most suitable description format for use in future research, modulo the caveats mentioned in Section 3.

5 Discussion

To put our work in perspective we will point out several caveats in our methodology and results before heading of to final conclusions. We cover our methodology, search & select and results.

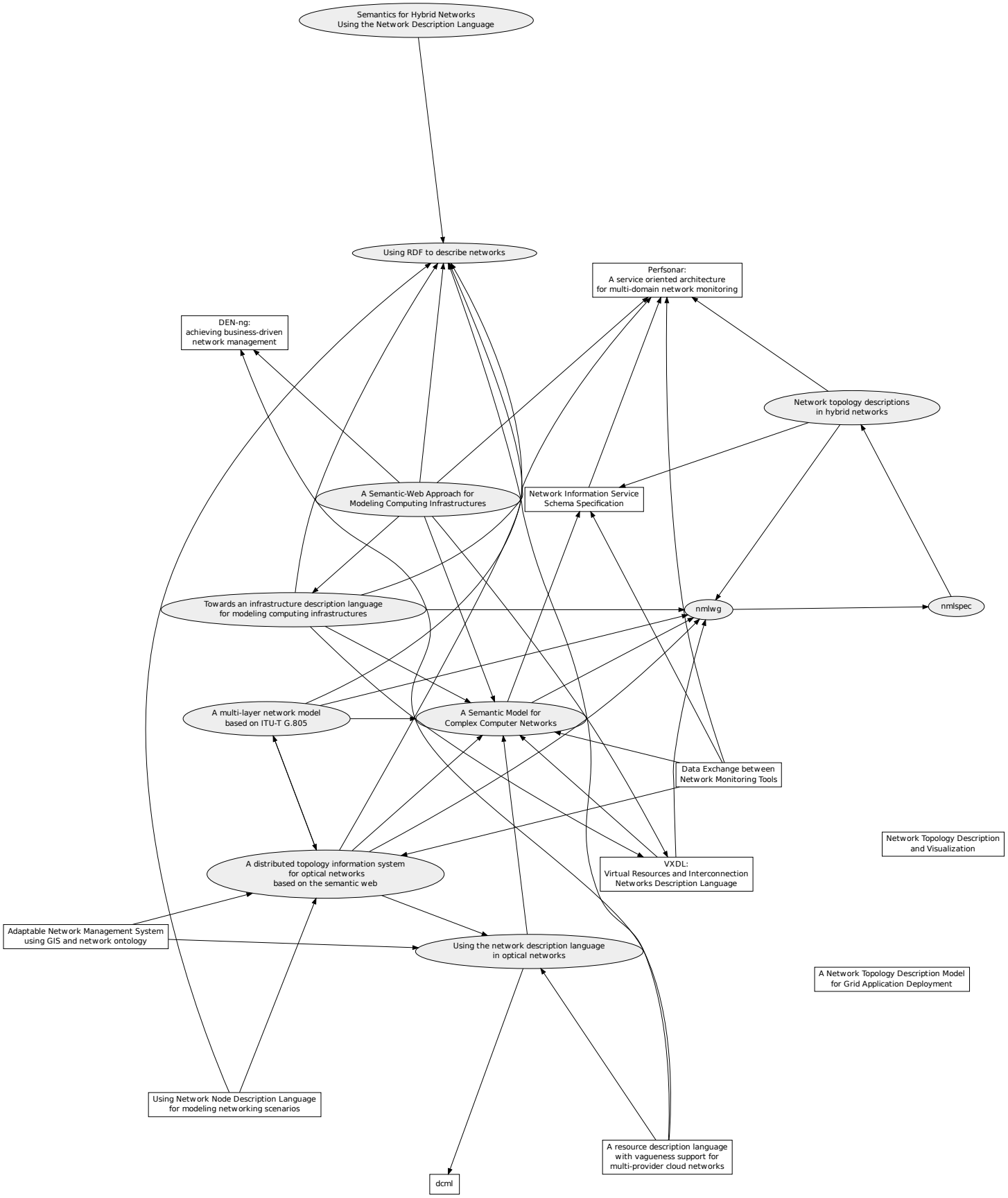


Figure 2: Citation network of results

language	single / multi layer	internal / exchangeable	scope	technology-neutral
NDL	single	exchangeable	network	yes
NDLv2	multi	exchangeable	network	yes
NML	multi	exchangeable	network	yes
INDL	multi	exchangeable	(virtual) network & infrastructure	yes
VXDL	single	exchangeable	network	no, IP-only
DEN-ng	multi	internal	not defined	yes
NNDL	single	exchangeable	node, service & network	no, IP-only
perfSONAR TS	multi	exchangeable	interface statistics	yes

Table 2: summary of findings

methodology Ideally one knows relevant keywords in combination with conference proceedings and/or periodicals before starting the research. This allows the selection of search engines and keywords for best coverage of these outlets. In this survey we did not have such up front knowledge and a repetition of the survey with such information might yield results without requiring the elaborate methodology used here.

search & select During search and select there were multiple factors negatively influencing the output. First, the DBLP search engine proved to be highly unsuitable for initial selection of a previously unknown topic due to its lack for search in abstract, keywords and subject area. Its support for complex search queries also proved mediocre, with simple queries giving huge results and simple incremental filtering diminishing all but some remaining unrelated artefacts.

Our manual filtering work fell short of compensating these shortcomings due to its additional lack of meta-data export: after scripting selection on titles, we found no feasible way to access data to filter down the resulting 400+ hits short of manually downloading every single paper from its respective academic publisher's website. The resulting limited output from DBLP can be seen in Figure 1, effectively reducing our input to two search engines.

A second input related caveat is the seemingly small overlap between Web of Science and Scopus. This suggests the addition of a fourth search engine might yield a significant number of new papers. We think such an addition would reduce the increase of papers found by citation tracing but would not lead to significant new content. This assumption is backed by our verification step as described in Section 2. Nevertheless, a future repetition of this survey should include additional search engines, both to verify our claim and to compensate for DBLP's shortcomings.

Finally, two papers were inaccessible due to their common Chinese download site being pay-walled, apparently too small to be included in the universities institutional access bundles. These papers did not turn up again while tracing citations or during verification.

results The limited output of the search phase of our survey questions the coverage of academic literature for operational networking. We feel obliged to note that there might be (commercial) unpublished offerings beyond the scope of this survey.

6 Conclusions

This survey is an attempt to answer the question *"How is a network architecture formally described?"*. To this end, we surveyed the existing academic literature on network topology description formats. After covering methodology in Section 2 we presented survey results in Section 3. We then analysed the relevance of our findings in Section 4. Finally, Section 5 covered caveats related to our work and its results.

We find that formal network architecture description is a small area of research, primarily driven by grid and cloud computing needs. There are a handful of languages available, each serving different specific needs. There is no wide adoption of any of the languages found, which leads to a selection based on our specific needs. For the purpose of future research in reasoning about the characteristics of topology (using the definitions of Section 1) we find that NML is most suitable as formal topology description language.

References

[1] DCML. Data Center Markup Language Framework Specification. Technical report, 2004. v0.11, retrieved from `http://www.dcml.org/technical_info/` on 2013-07-19.

[2] Freek Dijkstra, Bert Andree, Karst Koymans, Jeroen Van Der Ham, Paola Grosso, and Cees de Laat. A multi-layer network model based on itu-t g. 805. *Computer Networks*, 52(10):1927–1937, 2008.

[3] Stefan Daniel Dumitrescu, Alexandru Smeureanu, Andreea Diosteanu, and Liviu Adrian Cotfas. Adaptable network management system using gis and network ontology. In *Roedunet International Conference (RoEduNet), 2010 9th*, pages 310–315. IEEE, 2010.

[4] Mattijs Ghijsen, Jeroen van der Ham, Paola Grosso, Cosmin Dumitru, Hao Zhu, Zhiming Zhao, and Cees de Laat. A semantic-web approach for modeling computing infrastructures. Technical report.

[5] Guilherme Piegas Koslovski, Pascale Vicat-Blanc Primet, and Andrea Schwertner Charao. VXDL: Virtual Resources and Interconnection Networks Descriptio Language. volume 2 of *Lecture Notes of the Institute for Computer Sciences Social Informatics and Telecommunications Engineering*, pages 138–154, 2009.

[6] S Lacour, C Perez, and T Priol. A network topology description model for grid application deployment. In *Fifth IEEE/ACM International Workshop on Grid Computing, proceedings*, pages 61–68, 2004.

[7] J. van der Ham, F. Dijkstra, R. Łapacz, and Zurawski J. Network markup language base schema version 1. Technical report, Open Grid Forum.

[8] Jeroen Van Der Ham, Freek Dijkstra, Paola Grosso, Ronald Van Der Pol, Andree Toonk, and Cees De Laat. A distributed topology information system for optical networks based on the semantic web. *Optical Switching and Networking*, 5(2):85–93, 2008.

[9] Jeroen van der Ham, Mattijs Ghijssen, Paola Grosso, and Cees de Laat. Trends in Computer Network Modelling towards Future Internet.

[10] Jeroen J Van der Ham, Freek Dijkstra, Franco Travostino, Hubertus Andree, and Cees TAM de Laat. Using rdf to describe networks. *Future Generation Computer Systems*, 22(8):862–867, 2006.

[11] H. Wang and Y. Chen. Network topology description and visualization. In *ICACTE 2010 - 2010 3rd International Conference on Advanced Computer Theory and Engineering, Proceedings*, volume 6, pages V652–V656, 2010.

[12] Jane Webster and Richard T Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, 26(2), 2002.

[13] Joost F Wolfswinkel, Elfi Furtmueller, and Celeste PM Wilderom. Using grounded theory as a method for rigorously reviewing literature. *European Journal of Information Systems*, 22(1):45–55, 2011.

Appendix

The remainder of this paper covers research questions 1 and 2a. To keep the workload of the literature survey within bounds, it was decided to answer these questions using real world experience based on interviews and inquiries. Consequently, the following sections are no representation of the current state in academia, instead providing a glimpse into problems faced by clients of Deloitte.
A consequence of the choice to gather input from the Security & Privacy team at Deloitte is that the functional requirements are all oriented towards security. The author intends to let the answers steer further research towards practical application within the overlapping fields of networking and security.
In the following three sections we will cover in order research questions 1a, 1b and 2a.

1. *How do we represent operational requirements for networks?*
   (a) *What are operational requirements?*
   (b) *How are operational requirements formally described?*
2. *How do we represent the architecture of networks?*
   (a) *What characteristics contribute towards their operational requirements?*

Operational requirements

We cover three common functional requirements, in order: *availability*, *segregation* and *ownership*. We then move on to review existing methods to formalize these requirements.

availability "making it work", "keeping it working" and "make it usable"[6] are by far the most heard phrases to describe operational requirements for networks. Only after further inquiry other requirements are heard, supporting the popular view that other requirements (e.g. security) are an afterthought. This is to be expected from a technology branch which primarily exists to connect; without connection, there is no purpose. However, we will see that our further requirements are sometimes directly opposite to this goal.
An example of an availability requirement is the uptime requirement for a critical service, perhaps in the view of required periodic maintenance. The network connecting such services to their clients needs to deliver said uptime, even though the occasional planned and unplanned maintenance is unavoidable. Checking the characteristics of the networks topology might yield information on the feasibility of such requirements.

segregation Although we determined that availability is to most the top valued requirement, this does not hold for all services. There are use-cases abound which require the absence of availability, at least in the sense that certain endpoints should not be able to exchange information. This non-connectedness property is often realised by physical or logical segregation. The (continued) existence of segregated network zones is worth checking, both during design and operation.

Examples of segregation requirements in commercial networks are abundant, situations where such a requirement would be advisable even more plentiful, though actual deployed segregation beyond "internal, external, DMZ" rare. Use cases encountered were amongst others: medical device data (e.g. MRI scanner to workstation), control plane segregation (e.g. admin interfaces, out of band access), high performance backend connections (e.g. storage area networks, high availability synchronisation links).

ownership The recent developments with respect to blanket surveillance and traffic analysis[7] have made it more and more important to understand who has access to what information. This carries over to networking. To assure the confidentiality or integrity of critical information it becomes prudent to determine ownership[8] of the communication links, if only in an attempt to provide defence in depth.

Examples of ownership requirements can be found in the medical sector where the strict data protection laws concerning medical and biometric information apply. So-called "critical infrastructure", e.g. water supply, energy networks, 911-like emergency numbers also come to mind as candidates that require the increased assurance provided by ownership of infrastructure.

[6] Strictly speaking this would be "accessibility", but we classify it as availability because a network failing to be accessible will often be perceived as not available. That makes them equal for our purposes.

[7] I.e. documents uncovered by US whistle-blower Edward Snowden indicate the existence of programs for mass surveillance in most developed countries.

[8] Note that we use "ownership" in a somewhat loose sense: e.g. a leased line could count as owned if there is reasonable cause to believe that it will not be interfered with.

Formal description of operational requirements

Throughout all conversations and interviews, the author has been unable to identify any real world application of formal (i.e. machine readable) description of operation requirements. The only, partially related, descriptions are so-called Service Level Agreements (SLA's) containing numbers on uptime requirements as part of a contract. In addition, it is questionable whether such numbers represent functional requirements or merely are the result of a service/cost trade-off, which would murk calculations based on requirements with the inclusion of premature cost optimized inputs.
This brings us to a quick conclusion. We have been unable to find examples of formal descriptions for operational requirements in the real world. We are therefore unable to re-use existing description formats to formalize functional requirements.

The contribution of characteristics to operational requirements

Having presented an overview of common operational requirements, we now proceed to present preliminary approaches to the identification of characteristics that contribute toward operational requirements. We follow the same structure as before, listing our findings paragraph by paragraph in the order availability, segregation and ownership.

availability We found two characteristics which influence availability from a perspective of the topology. First, the existence of multiple distinct paths from source to destination. With multiple paths available, failure of some might be tolerable, indicating a degree of robustness in the topology.
Second, the reverse is also true: where (multiple) paths intersect we find points whose failure have a larger impact. These "choke points" (or taken to the extreme "single points of failure") represent a negative degree of robustness for a topology.

segregation The intention of segregation is non-connectedness between certain parts of the network. In terms of topology (especially in the lower layers) the opposite of segregation can be quantified in the number of ways to connect those parts. The risk of violating segregation can therefore be expressed in the number of ways to do so. A topology allowing less such violations is better suited to provide segregation[9].

[9]Note that this includes possible, but not currently configured paths.

ownership Connecting topology characteristics to the functional requirement of ownership seems to involve the same methods as previously described for availability and segregation. Where availability involved the existence of multiple paths and the absence of choke points, ownership just excludes non-owned paths in this metric. The failure mode is therefore twofold: when there are no more paths available which are not owned, we break the ownership requirement, but availability might still hold.
The connection to segregation is similar, because we can express the risk of violating ownership as a violation of segregation between the part of a network under our own control and the rest. A topology with a low risk of segregation failure for this ownership constraint would consequently be a topology with low risk of violating ownership.

Discussion & Conclusion

It is important to note that all of the previous requirements can only hold if there is full control over the network. This in turn requires visibility and insight over what is and what isn't connected. Although seemingly obvious, it appears that this insight is a big problem in practice. Commercial entities and their IT staff often have no clear continuous view of how their networks operate and evolve.
There is no clear business requirement to have this insight and it is not a functional requirement on its own, but it is a necessary precondition to support the other three. Research in the automatic checking of functional properties can help in this respect, but not without accurate topology information up front. This is a clear barrier to adoption for any approach to reason using topology information.
To conclude the appendix, we have shown three dominant operational requirements for networks: availability, segregation and ownership. These operational requirements are not formally described in any known format. Finally, for each of the three operational requirements we identified characteristics which may indicate the degree to which they are fulfilled.

Appendix B.

Additional validation of the formal description of requirements

In this addendum to the main validation work, we use formal logic and proofs to increase confidence in the construction proposed in Chapter 4. Before diving in, we present a short overview of what has been proved and what remains to be shown.

Our proposal introduced a way to verify operational requirements using a composition of path selection and an additional computation-validation step. For each of the four requirements covered, this composition is synthesised directly from a formal description, absolving us from the need to verify the composition itself. However, to show that the formal description of the requirements itself is not flawed, it remains to be shown that our proposal does not yield any results which are contrary to the expectations which lead to the formal specification of the requirement in the first place.

We will show that our proposed composition does not allow for false positives and negatives. This way, we confirm that the behaviour of the composition complies with the expected meaning of each requirement. The terms false positive and false negative correspond to their usual meaning, applied to verification:

- A false positive in the context of verification means that the composition incorrectly produces a positive verification of a requirement although the requirement *does not* hold in the network under review.
- Conversely, a false negative in the context of verification means that the composition incorrectly produces a negative verification of a requirement although the requirement *does* hold in the network under review.

The approach is as follows. For each of the four requirements possible failure cases are listed corresponding to either false positive or false negative results. We then prove that the algorithm, by definition, contradicts the occurrence of these results. Using this approach we prove conformance of the formal descriptions to the expectations they are intended to capture.

In this section, we make a number of assumptions about a number of external parts with respect to our work. Specifically, we assume the correctness of the path selection

algorithm; the network description[1]; and (for the path diversity requirement) the correctness of the clique checking algorithm(s). We believe these assumptions are reasonable considering the scope of this work.

The requirements are covered in the now familiar order segmentation, control, time to recovery and path diversity. As was the case previously, the complexity of the proofs increases with each section, with the final proof requiring a completely different approach to the other three.

B.1. Segmentation

Like the composition of Segmentation in Chapter 4, the proofs for the non-occurrence of false positives for segmentation verification is very short and simple.

B.1.1. False positives

A false positive for segmentation would mean that the algorithm incorrectly reports that a network is segmented, while it is not segmented in the description. To prove: the proposed segmentation verification does not produce false positives.

Proof. (by contradiction)

We assume that algorithm S(A,B) to check the segmentation requirement of communication from A to B produces false positives. I.e. S(A,B) yields True, but communications from A to B is possible, violating the segmentation requirement.

$$\text{“A can communicate to B”} \iff \exists(A \to B) \tag{B.1}$$

$$\text{“}S(A,B)\text{ yields True”} \iff (A \rightrightarrows B) = \varnothing \tag{B.2}$$

$$\text{definition of } \rightrightarrows: \quad \forall(P \to_i Q)\ .\ (P \to_i Q) \in (P \rightrightarrows Q) \tag{B.3}$$

$$\text{B.1} \wedge \text{B.2} \wedge \text{B.3} \iff (A \to B) \in \varnothing \tag{B.4}$$

$$\lightning \qquad \square$$

B.1.2. False negatives

A false negative for segmentation would mean that the algorithm fails to report segmentation, while the network described is segmented. To prove: the proposed segmentation verification does not produce false negatives.

[1] If the topology description is not correct, the results of requirement verification does not necessarily correspond to the real network. It does not however detract from the correctness of the verification of the requirements with respect to the description.

Proof. (by contradiction)

We assume that algorithm S(A,B) to check the segmentation requirement of communication from A to B produces false negatives. I.e. S(A,B) yields False, but communications between A and B is impossible, satisfying the segmentation requirement.

$$\begin{aligned}
\text{“A cannot communicate to B”} &\iff \neg\exists(A \to B) && \text{(B.5)}\\
\text{“}S(A,B)\text{ yields False”} &\iff (A \rightrightarrows B) \neq \varnothing \\
&\iff \exists(A \to B) \in (A \rightrightarrows B) && \text{(B.6)}\\
&\lightning \text{ (by B.3} \wedge \text{B.5} \wedge \text{B.6)} && \square
\end{aligned}$$

We conclude that the formal description of segmentation and the proposal resulting from a composition with path selection produces expected results.

B.2. Control

B.2.1. False positives

A false positive for control would mean that the algorithm reports control over paths, while there exist paths in the network described which are not under control. To prove: the proposed control verification does not produce false positives.

Proof. (by contradiction)

We assume that algorithm C(A,B) to check the control criterion of communication between A and B produces false positives. I.e. C(A,B) yields True, but communications from A to B traverse a node Q which violates the control criterion.

$$\begin{aligned}
\text{“}Q\text{ is traversed between A and B”} &\iff \exists(A \to B) : Q \in \mathit{elements}(A \to B) \\
&\iff \exists Q \in \mathit{elements}(A \to B)(\subseteq \mathit{elements}(A \rightrightarrows B)) \\
&\iff \exists Q \in \mathit{elements}(A \rightrightarrows B) && \text{(B.7)}\\
\text{“}Q\text{ violates the control criterion”} &\iff \neg\phi(Q) && \text{(B.8)}\\
\text{“}C(A,B)\text{ yields True”} &\iff \phi(A \rightrightarrows B) \\
&\iff \forall(A \to_i B) \in \mathit{paths}(A \rightrightarrows B)\ .\ \phi(A \to_i B) \\
&\iff \forall(A \to_i B) \in \mathit{paths}(A \rightrightarrows B), \\
&\quad \forall R \in \mathit{elements}(A \to_i B)\ .\ \phi(R) && \text{(B.9)}\\
&\quad \lightning \text{ (by B.7} \wedge \text{B.8} \wedge \text{B.9)} && \square
\end{aligned}$$

B.2.2. False negatives

A false negative for control would mean that the algorithm fails to report control over paths, while all paths in the network described are under control. To prove: the proposed control verification does not produce false negatives.

Proof. (by contradiction)

We assume that algorithm C(A,B) to check the control criterion of communication between A and B produces false negatives. I.e. C(A,B) yields False, but communications from A to B can only use nodes which adhere to the control criterion.

$$\text{“Comms. from A to B use controlled nodes”} \iff \forall Q \in elements(A \rightrightarrows B) \;.\; \phi(Q) \tag{B.10}$$

let $\mathbb{U} := \{(A \rightarrow_i B) | (A \rightarrow_i B) \in paths(A \rightrightarrows B), \neg\phi(A \rightarrow_i B)\}$ (set of unsafe paths)

$$\begin{aligned} \text{“}C(A,B) \text{ yields False”} &\iff \mathbb{U} \neq \varnothing \\ &\iff \exists (A \rightarrow B) \in \mathbb{U} \;.\; \neg\phi(A \rightarrow B) \\ &\iff \exists R \in elements((A \rightarrow B) \in \mathbb{U}) \;.\; \neg\phi(R) \\ &\iff \exists R \in elements(A \rightrightarrows B) \;.\; \neg\phi(R) \end{aligned} \tag{B.11}$$

$$\lightning \text{ (by B.10} \wedge \text{B.11)}$$ □

We conclude that the formal description of control and the proposal resulting from a composition with path selection produces expected results.

B.3. Time to recovery

B.3.1. False positives

A false positive for time to recovery would mean that the algorithm incorrectly reports that a network has a certain time to recovery, while this recovery time would not be truthful for the network described. To prove: the proposed segmentation verification does not produce false positives.

Proof. (by contradiction)

Let $T_r(x)$ denote the time to recovery of an element x.

We assume that algorithm T(A,B,k) to compute the time to recovery for communication between A and B produces false positives. I.e. T(A,B,k) yields k, but communications

from A to B always traverse some element Q_i with larger time to recovery $T_r(Q_i) > k$ (e.g. k in seconds).

$$\text{“}Q_i \text{ is always traversed”} \iff \forall (A \to_i B) \in (A \rightrightarrows B) \;.\; \exists Q_i : Q_i \in elements(A \to_i B) \tag{B.12}$$

$$\text{“All } Q_i \text{ have recovery time } > k\text{”} \iff \forall Q_i : T_r(Q_i) > k \tag{B.13}$$

let $slowestPaths := \{max_{T_r}(elem(K \to L)) \mid (K \to L) \in (A \rightrightarrows B)\}$

$$\text{B.12} \wedge \text{B.13} \iff \forall E \in slowestPaths : T_r(E) > k$$

let $fastestPath := min_{T_r}(slowestPaths)$

$$\iff T_r(fastestPath) > k \tag{B.14}$$

$$\text{“}T(A, B, k) \text{ yields True”} \iff T_r(fastestPath) \leq k \tag{B.15}$$

$$\lightning \text{ by B.14} \wedge \text{B.15}$$ □

B.3.2. False negatives

A false negative for time to recovery would mean that the algorithm fails to verify a certain time to recovery, while this recovery time would be truthful for the network described. To prove: the proposed time to recovery verification does not produce false negatives.

Proof. (by contradiction)

We assume that algorithm T(A,B,k) to compute the time to recovery for communication between A and B produces false positives. I.e. T(A,B,k) yields False, but communications from A to B can traverse some path with a better worst case recovery ($\leq k$).

$$\text{“A path with better worst case recovery”} \iff \exists (A \to B) : (\forall E \in elems(A \to B) : T_r(E) < k)$$

let $slowestPaths := \{max_{T_r}(elem(K \to L)) \mid (K \to L) \in (A \rightrightarrows B)\}$

$$\iff \exists E \in slowestPaths : T_r(E) < k$$

let $fastestPath := min_{T_r}(slowestPaths)$

$$\iff T_r(fastestPath) < k \tag{B.16}$$

$$\text{“}T(A, B, k) \text{ yields False”} \iff T_r(fastestPath) \geq k \tag{B.17}$$

$$\lightning \text{ by B.17} \wedge \text{B.16}$$ □

We conclude that the formal description of time to recovery and the proposal resulting from a composition with path selection produces expected results.

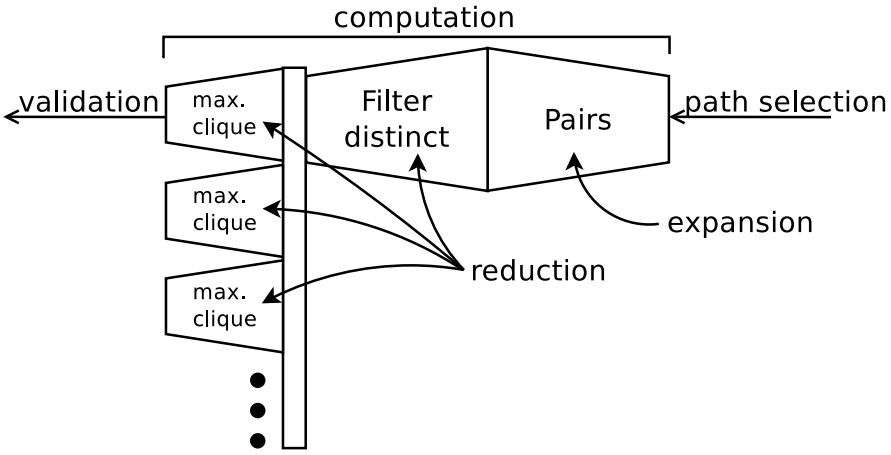


Figure B.1.: Schematic display of the computation component of the path diversity algorithm

B.4. Path diversity

To find possible ways to obtain false {positives, negatives} we examine Listing 4.4. Here we find a composition three separate steps: pair creation, distinctness filtering and clique finding. Figure B.1 gives a schematic view of the different steps, corresponding to the code in Listing 4.4. In order, pairing expands, while distinctness filtering and clique finding steps reduce the number of outputs.

The notion of expansion and reduction are important, because the simplest way to look at false {positives, negatives} is a off-by-one error[2], which has to be introduced somewhere if it occurs. A false positive is essentially an additional path(-pair) which does not correspond to a real path(-pair) in the description. Likewise, a false negative is essentially a missing path(-pair) which exists in the description, but is somehow lost during computation.

The sketch of the full proof is the following. For both false positives and negatives, there are five components which can be the cause of error. By proving the absence of errors in each of these components, we prove their absence from the whole. The five components are:

1. path selection
2. pair generation
3. distinctness filtering
4. clique finding

[2]This assumption can be made without loss of generality. We opt for the k, $k-1$ and $k+1$ purely for ease of understanding.

5. validation

Path selection and clique finding are both excluded from this examination, conform our assumptions listed earlier in this section. Validation (Listing B.1) is extremely trivial: it is nothing more than a comparison with k. Moreover, the absence of side-effects in pure Haskell code combined with the type signature absolves us from any need to prove that validation changes the number of path.

```
validateDiversity :: Int -> [[Path]] -> Bool
validateDiversity n pss = ((>=n) . length . head) pss
```

Listing B.1: the definition of validation (excerpt from Listing 4.4)

This leaves pair generation and distinctness filtering to be verified. Before individually addressing false {positives, negatives} we first back up a claim made earlier. In Figure B.1 and its accompanying explanation we claimed that pair generation and distinctness filtering are respectively expanding and reducing the number of paths. We employ a simple induction proof on the (already inductive) definitions of both functions.

```
1 pairs :: [Path] -> [(Path, Path)]
2 pairs [] = []
3 pairs (p:ps) = [(p,b)|b<-ps] ++ pairs ps
```

Listing B.2: the definition of pair generation (excerpt from Listing 4.4)

Proof. Listing B.2 shows the pair generation. We define the property α to mean "does not decrease" with respect to the number of pairs. For the base case, namely the pair generation of the empty list (line 1), it produces the empty list as output and α holds. Assuming α holds for a list of length n, we now prove that this is also the case for an input of length $n + 1$, completing the induction step.

On line 2, we see that there is a pattern match on the input list, splitting it up in the first path and the remaining paths (n, for this specific example). The first path is expanded to a list of n tuples, with the first path as the first member of this tuple and each of the n different paths as second member of the tuple. This list of pairs an expansion, so α holds. Finally, this list is concatenated to the result of pairs with an input size of n paths, for which we assumed α to hold. This concatenation is therefore also an expansion, α holds and we are done. □

```
1 -- | 'filter', applied to a predicate and a list, returns the list of
2 -- those elements that satisfy the predicate; i.e.,
3 --
4 -- > filter p xs = [ x | x <- xs, p x]
5 filter :: (a -> Bool) -> [a] -> [a]
6 filter _pred [] = []
7 filter pred (x:xs)
8   | pred x = x : filter pred xs
```

```
9 |   | otherwise = filter pred xs
```

(Code excerpt from the Haskell Prelude (BSD 3-clause, http://hackage.haskell.org/package/base-4.6.0.1/src/LICENSE))

Listing B.3: the definition of filter in the Haskell standard library (Prelude)

Proof. Listing B.3 shows the definition of filter (`pred` is our distinctness test, implemented as a check for an empty intersection). We define β to mean "does not increase" with respect to the number of pairs. For the base case, namely the distinctness filtering of the empty list (line 6), it produces the empty list and β holds. Assuming β holds for a list of length n, we now prove that this is also the case for an input of length $n + 1$, completing the induction step.

On line 7, there is a pattern match on the input list, splitting it up in the first path and the remaining paths (n, for this specific example). The first pair is checked for distinctness using `pred`. On lines 8 and 9, a recursive call is made to filter the remaining n pairs, but success of the `pred` call (line 8) causes the algorithm to prepend that path to the end result, while failure (line 9) discards it. Clearly, β holds for both retaining or discarding a pair. We assumed that β holds for the recursive call on n pairs. Finally, β also holds for the prepend operation and we are done. □

Having sketched the structure of the full proof and covered the common parts, we now cover the specifics in two sections.

B.4.1. False positives

A false positive for path diversity would mean that the algorithm incorrectly reports that there exist multiple independent paths (say, k paths), while these exist in the description (say, $k - 1$ paths exist). To prove: the proposed path diversity verification does not produce false positives.

Assuming the path selection is correct, this means an invalid path(-pair) is introduced in one of the consecutive steps. Therefore, we reduce the above to the following. To prove: none of the computation steps introduces invalid path(-pairs). After using our assumptions, two remaining parts of the composition remaining to be checked. Additionally, we have shown previously that distinctness filtering does not increase the number of pairs (and as such is unable to introduce invalid pairs), therefore, the only remaining source of possible invalid path(-pairs) is the pair generation step.

However pair generation, by definition (see Equation 4.1), is the process of creating all possible no-overlap relations (modulo ordering, i.e. $\frac{k(k-1)}{2}$ pairs for k paths), to be consecutively checked by the distinctness filter. With this definition, there is no such thing as an invalid pair. The only thing left to show is that pair generation does not create new paths before pairing them. This can be trivially seen to be true in Listing B.2, thereby completing the proof.

B.4.2. False negatives

A false negative for path diversity would mean that the algorithm incorrectly reports that there do not exist multiple independent paths (say, k paths), while this the case in the description (say, $k + 1$ paths exist). To prove: the proposed path diversity verification does not produce false negatives. We will use the same method used in the previous section to show that there can not be false negatives.

Assuming the path selection is correct, this means a path(-pair) is removed in one of the consecutive steps. Therefore, we reduce the above to the following. To prove: none of the computation steps removes valid path(-pairs). After using our assumptions, two remaining parts of the composition remaining to be checked. As before, we have shown previously that pair generation does not decrease the number of pairs (and as such is unable to remove valid pairs), therefore, the only remaining source of possible valid pair removal is the distinctness filtering step.

The only way that the filtering step removes a pair is when the intersection of its member paths is non-empty, i.e. they are partly overlapping (Listing 4.4). At the same time, the definition of a distinct pair is exactly the opposite, that its members have a non-empty intersection (conform Equation 4.2). In summary, the distinctness filter is unable to remove any valid pairs.

We conclude that the formal description of path diversity and the proposal resulting from a composition with path selection produces expected results.

Appendix C.

Source code

C.1. NML

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:nml="http://schemas.ogf.org/nml/2013/05/base#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <!-- system -->
  <nml:Node rdf:about="urn:ogf:network:rtsn.nl:2014:system">
    <nml:hasInboundPort>
      <nml:Port rdf:about="urn:ogf:network:rtsn.nl:2014:system:in">
        <nml:isSink rdf:resource="urn:ogf:network:rtsn.nl:2014:link:edgeswitch-system"/>
        <nml:encoding>100/1000BASE-T</nml:encoding>
      </nml:Port>
    </nml:hasInboundPort>
    <nml:hasOutboundPort>
      <nml:Port rdf:about="urn:ogf:network:rtsn.nl:2014:system:out">
        <nml:isSource rdf:resource="urn:ogf:network:rtsn.nl:2014:link:system-edgeswitch"/>
        <nml:encoding>100/1000BASE-T</nml:encoding>
      </nml:Port>
    </nml:hasOutboundPort>
  </nml:Node>
  <!-- edgeswitch:Port1 -->
  <rdf:Description rdf:about="urn:ogf:network:rtsn.nl:2014:edgeswitch">
    <nml:hasOutboundPort>
      <nml:Port rdf:about="urn:ogf:network:rtsn.nl:2014:edgeswitch:Port1:out">
        <nml:isSource rdf:resource="urn:ogf:network:rtsn.nl:2014:link:edgeswitch-system"/>
        <nml:encoding>100/1000BASE-T</nml:encoding>
      </nml:Port>
    </nml:hasOutboundPort>
    <nml:hasInboundPort>
      <nml:Port rdf:about="urn:ogf:network:rtsn.nl:2014:edgeswitch:Port1:in">
        <nml:isSink rdf:resource="urn:ogf:network:rtsn.nl:2014:link:system-edgeswitch"/>
        <nml:encoding>100/1000BASE-T</nml:encoding>
      </nml:Port>
    </nml:hasInboundPort>
  </rdf:Description>
  <!-- system <-> edgeswitch -->
  <nml:Link rdf:about="urn:ogf:network:rtsn.nl:2014:link:edgeswitch-system"/>
  <nml:Link rdf:about="urn:ogf:network:rtsn.nl:2014:link:system-edgeswitch"/>
```

```
</rdf:RDF>
```

Listing C.1: Sample NML

C.2. path selection

```
{−# LANGUAGE TupleSections #−}
module PathSelection.MultiLayerBreadthFirst (
multiLayerBreadthFirst
)where
import Control.Monad
import Control.Monad.Trans.Class (lift)
import Control.Monad.Trans.Maybe (MaybeT(..), runMaybeT)
import qualified Data.Map.Strict as Map
import qualified Data.Heap as Heap
import Data.Set (Set)
import qualified Data.Set as Set

import NML
import NML.Relations (expandForward, getParentNode)
import NML.Attributes
import NML.Query (getNetworkObjects)
import Util

−− | 'PathObject' used in MinHeap, using distance for comparison.
data PathObject = PathObject { visitedLinks :: [NetworkObject] −− ^ sequence of visited
    edges
                             , labelStack :: [Label] −− ^ current technology stack
                             , technologyStack :: [AdaptationFunction] −− ^ current
                                 technology stack
                             , distance :: !Int −− ^ distance covered
                             , loopSet :: Set (NetworkObject, Maybe AdaptationFunction,
                                 Maybe Label)
                             } deriving (Eq, Show)
instance Ord PathObject where
    PathObject{distance=d1} `compare` PathObject{distance=d2} = d1 `compare` d2

currentNode :: PathObject −> NetworkObject
currentNode PathObject{visitedLinks=ns} = head ns

previousNode :: PathObject −> Maybe NetworkObject
previousNode PathObject{visitedLinks=ns} = safeHead =<< safeTail ns

type Source = NetworkObject
type Destination = NetworkObject

−− | wrapper for 'multiLayerBreadthFirst'' returning [Path]
multiLayerBreadthFirst :: Source −> Destination −> NMLReader [Path]
```

```
multiLayerBreadthFirst = (liftM $ map (tail . reverse . visitedLinks)) .:
    multiLayerBreadthFirst'

-- | multiLayerBreadthFirst algorithm (path selection in G_l)
-- as published in
-- "Path selection in mult-layer networks" by Kuipers, Dijkstra, 2008
multiLayerBreadthFirst' :: Source -> Destination -> NMLReader [PathObject]
multiLayerBreadthFirst' src dst = go [] (Heap.singleton (PathObject [src] [] [] 0 ls)) src dst --
     enter main loop with empty result and minHeap
    where
        ls = Set.singleton (src, Nothing, Nothing)

        -- | loop until either heap is exhausted or we arrive at our
        -- destination with an empty technologystack
        go :: [PathObject] -> Heap.MinHeap PathObject -> Source -> Destination ->
            NMLReader [PathObject]
        go result heap src dst = case Heap.view heap of
                Nothing -> return result -- Heap exhausted
                Just (minPO, heap') -> (checkDestination (currentNode minPO) dst) >>=
                    \arrived ->
                                    if (arrived && (null $ technologyStack minPO))
                                    -- found a possible path, add it to the result list
                                    then go (minPO:result) heap' src dst
                                    else do let cur = currentNode minPO
                                                let prev = previousNode minPO
                                                -- use topology to determine next hops
                                                next <- expandForward prev cur
                                                -- foreach: extendpath
                                                heap'' <- foldr (f minPO) (return heap')
                                                    next
                                                -- recurse using updated heap
                                                go result heap'' src dst

        f :: PathObject -> NetworkObject -> NMLReader (Heap.MinHeap PathObject) ->
            NMLReader (Heap.MinHeap PathObject)
        f p n h = extendPath p n >>= \extension ->
                case extension of
                    Nothing -> h
                    Just p' -> liftM2 Heap.insert (return p') h

-- used to be:
-- extendPath :: PathObject -> (Link, Node) -> Maybe PathObject
extendPath :: PathObject -> NetworkObject -> NMLReader (Maybe PathObject)
extendPath p n = runMaybeT $ do
    let visitedLinks' = n:(visitedLinks p)

    let distance' = distance p + 1 -- this can be swapped for a more realistic distance metric

    -- perform adaptation / de-adaptation check
```

```
adaptationFunction <- lift $ adaptationFunction n
let tStack = technologyStack p
technologyStack' <- maybeZero $ case n of
            (Service Adaptation _) -> Just $ maybe (error $ "Adaptation without
                adaptationFunction property: "++show n)
                                        (:tStack) adaptationFunction
            -- ^ grow the technology stack with the technology in this Adaptation
            (Service Deadaptation _) -> maybe (error $ "Deadaptation without
                adaptationFunction property: " ++ show n)
                                        (\t -> if ((not $ null tStack) && t == head
                                            tStack)
            -- pop the technology stack with the technology in this Adaptation
                                        then Just $ tail tStack
            -- or short circuit Maybe monad to reject Path due to incompatibility
                                        else Nothing) adaptationFunction
            -- if this is not an adaptation, the stack does not change
            _ -> Just tStack

-- perform label check
currentLabel <- lift $ label $ currentNode p
nextLabel <- lift $ label n
let lStack = labelStack p

-- we add the label for the current node to the stack if the extension is an adaptation
labelStack' <- maybeZero $ case n of
            (Service Adaptation _) -> Just $ maybe (error $ "Adaptation from Port
                without label: "++(show $ currentNode p)) (:lStack) currentLabel
            -- grow the label stack with the label in this Adaptation
            _ -> Just lStack

-- we do an deadaptation check if the current node is an deadaptation
-- by checking whether the extension's label is already on the label stack
labelStack'' <- maybeZero $ case currentNode p of
            (Service Deadaptation _) -> maybe (error $ "Deadaptation to Port
                without label: "++show n)
                                        (\l -> if ((not $ null labelStack') && l ==
                                            head labelStack')
            -- pop the label stack with the label in this Adaptation
                                        then Just $ tail labelStack'
            -- or short circuit Maybe monad to reject Path due to incompatibility
                                        else Nothing) nextLabel
            -- if this is not an adaptation, the stack does not change
            _ -> Just labelStack'

-- check whether the triple (NetworkObject, Technology, Label) is already present in our
    loopSet,
-- if so, we just looped back to our own path and we will not continue.
let loopTriple = (n, adaptationFunction , nextLabel)
loopSet' <- maybeZero $ if loopTriple `Set.member` loopSet p
```

```
                    then Nothing
                    else Just $ Set.insert loopTriple $ loopSet p

    -- return new PathObject, extended by one hop
    maybeZero $ Just $ PathObject visitedLinks' labelStack'' technologyStack' distance'
        loopSet'

checkDestination :: NetworkObject -- ^ current location
                 -> NetworkObject -- ^ destination
                 -> NMLReader Bool -- ^ whether or not the destination has been
                     reached
checkDestination loc dest@(Single Node _) = liftM (maybe False (== dest)) $
    getParentNode loc -- if the destination is a Node, check whether the current location has
     that node as a 'parent'
checkDestination loc dest@(Group _ _) = error "checkDestination_is_undefined_for_Group's"
checkDestination loc dest@(Service _ _) = error "checkDestination_is_undefined_for_Services"
checkDestination loc dest = return $ loc == dest
```

Listing C.2: Path selection using k-shortest multi-layer BFS

C.3. miscellaneous

```
module PropertyChecker (

Compute, ComputeM,
Validate,
computeProperty, verifyProperty,

selectPaths, stripPath,

segmentation, control, diversity, timeToRecovery,
validateSegmentation, validateControl, validateDiversity, validateTimeToRecovery

) where

import Control.Monad
import Control.Monad.Loops (dropWhileM)
import Control.Monad.Trans.Class (lift)
import Control.Monad.Trans.Writer (tell)
import Data.Maybe (isNothing)

import NML
import NML.RDFRep
import NML.Relations (getParentNode)
import PathSelection.MultiLayerBreadthFirst
import Reduction.Segmentation
import Reduction.Control
import Reduction.Diversity
import Reduction.TimeToRecovery
```

```
import Util

-- | a computation on top of path selection
type Compute a = [Path] -> a
-- | a computation on top of path selection,
-- using knowledge of the topology ('NMLReader')
type ComputeM a = [Path] -> NMLReader a
-- | a validation of a computation
type Validate a = a -> Bool

-- | Computation of a property for a source and destination,
-- using knowledge of the topology ('NMLReader')
computeProperty :: (Show a) => ComputeM a -> NetworkObject -> NetworkObject ->
    NMLReader a
computeProperty p = (p <=<) . selectPaths

-- | Validation of a property for a source and destination,
-- using knowledge of the topology ('NMLReader')
validateProperty :: Validate a -> a -> NMLReader Bool
validateProperty = (return .)

-- | Verification of a property for a source and destination,
-- using knowledge of the topology ('NMLReader')
-- verification = computation + validation
verifyProperty :: (Show a) => ComputeM a -> Validate a -> NetworkObject ->
    NetworkObject -> NMLReader Bool
verifyProperty c v n1 n2 = validateProperty v =<< computeProperty c n1 n2

selectPaths :: NetworkObject -> NetworkObject -> NMLReader [Path]
selectPaths = (paths =<<) .: multiLayerBreadthFirst

stripPath :: NetworkObject -> NetworkObject -> [Path] -> NMLReader [Path]
stripPath s d = stripSourcePrefixes s >=> stripDestinationPostfixes d

stripSourcePrefixes :: NetworkObject -> [Path] -> NMLReader [Path]
stripSourcePrefixes s = mapM removePrefix
    where
        removePrefix :: Path -> NMLReader Path
        removePrefix p = getParentNode s >>= \sourceParent ->
                         dropWhileM (\n -> getParentNode n >>= \parent -> return $
                                           isNothing parent || parent == sourceParent) p

stripDestinationPostfixes :: NetworkObject -> [Path] -> NMLReader [Path]
stripDestinationPostfixes d ps = return . map reverse =<< stripSourcePrefixes d (map
    reverse ps)

segmentation :: ComputeM [Path]
segmentation = liftM connectingPaths . return
```

```
control :: ComputeM [Path]
control = (verboseM'M "unsafe_paths" $ liftM2 annotateUnsafePaths getControlCriterium .
    return) . liftM2 unsafePaths getControlCriterium . return

annotateUnsafePaths :: Criterium -> [Path] -> [[Either NetworkObject NetworkObject]]
annotateUnsafePaths f = map (map (\n -> if not $ f n then Left n else Right n))

diversity :: ComputeM [[Path]]
diversity = (verboseM' "number_of_distinct_paths" (length . head)) . liftM2 distinctPaths
    getMaximumCliqueSize . return

timeToRecovery :: ComputeM (NetworkObject, Cost)
timeToRecovery = (verboseM "sample_NetworkObject_with_worst_recovery_cost") . liftM2
    worstCaseRecoveryCost getCostFunction . return
```

Listing C.3: Combining computation and validation into verification

Bibliography

[1] Bloomberg. Vodafone dutch clients lose service after rotterdam fire. `http://www.bloomberg.com/news/2012-04-04/vodafone-reports-fire-in-rotterdam-network-equipment-site.html`, April 2012. Accessed: 2013-09-19.

[2] V. Cerf, Y. Dalal, and C. Sunshine. Specification of Internet Transmission Control Program. RFC 675, December 1974.

[3] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[4] Freek Dijkstra. *Framework for path finding in multi-layer transport networks.* PhD thesis, Faculteit der Natuurwetenschappen, Wiskunde en Informatica (FNWI), Universiteit van Amsterdam (UvA), 2009.

[5] Freek Dijkstra, Jeroen van der Ham, Paola Grosso, and Cees de Laat. A path finding implementation for multi-layer networks. *Future Generation Computer Systems*, 25(2):142–146, 2009.

[6] International Organization for Standardization ISO. ISO/IEC 7498-1Information technology — open systems interconnection — basic reference model: The basic model. Technical report, June 1994.

[7] Fox-IT. Black Tulip report of the investigation into the diginotar certificate authority breach. Technical report, August 2012. `http://www.rijksoverheid.nl/ministeries/bzk/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update.html`, accessed: 2013-09-19.

[8] Farabi Iqbal, Jeroen van der Ham, and Fernando Kuipers. Technology-aware multi-domain routing in optical networks. 2013. (under submission, for a copy please contact the first author).

[9] Richard M Karp. *Reducibility among combinatorial problems.* Springer, 1972.

[10] Fernando Kuipers and Freek Dijkstra. Path selection in multi-layer networks. *Computer Communications*, 32(1):78–85, 2009.

[11] Fernando Kuipers, Turgay Korkmaz, Marwan Krunz, and Piet Van Mieghem. Performance evaluation of constraint-based path selection algorithms. *Network, IEEE*, 18(5):16–23, 2004.

[12] Fernando A Kuipers. An overview of algorithms for network survivability. *ISRN Communications and Networking*, 2012:24, 2012.

[13] Mohamed Lamine Lamali, HéLia Pouyllau, and Dominique Barth. Path computation in multi-layer multi-domain networks: A language theoretic approach. *Computer Communications*, 36(5):589–599, 2013.

[14] Simon Marlow. Haskell 2010 language report. Technical report, April 2010. `http://www.haskell.org/onlinereport/haskell2010/`, accessed: 2013-11-07.

[15] L. Martini, E. Rosen, N. El-Aawar, T. Smith, and G. Heron. Pseudowire Setup and Maintenance Using the Label Distribution Protocol (LDP). RFC 4447 (Proposed Standard), April 2006. Updated by RFCs 6723, 6870.

[16] Meral Shirazipour and Samuel Pierre. Multi-layer/multi-region path computation with adaptation capability constraints. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5. IEEE, 2010.

[17] J. van der Ham, F. Dijkstra, R. Łapacz, and Zurawski J. Network markup language base schema version 1. Technical report, Open Grid Forum.

[18] Jeroen Van Der Ham, Freek Dijkstra, Paola Grosso, Ronald Van Der Pol, Andree Toonk, and Cees De Laat. A distributed topology information system for optical networks based on the semantic web. *Optical Switching and Networking*, 5(2):85–93, 2008.

[19] Virginia Vassilevska. Efficient algorithms for clique problems. *Information Processing Letters*, 109(4):254–257, 2009.

[20] Li Xu, Freek Dijkstra, Damien Marchal, Arie Taal, Paola Grosso, and Cees De Laat. A declarative approach to multi-layer path finding based on semantic network descriptions. In *Optical Network Design and Modeling, 2009. ONDM 2009. International Conference on*, pages 1–6. IEEE, 2009.

[21] Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

Acknowledgements

This research would not have been possible without help and support from a number of people, some of which I will explicitly mention here[1].

First, I would like to thank my supervisors for their feedback and guidance. Jeroen, I enjoyed our conversations and the ability to discuss options just after I envisioned them. Henri, thanks for keeping me on-track and your coaching during our weekly calls. Pieter-Tjerk, I am thankful for your trust in my ability to work independently abroad, on a subject of my choice. I realise that giving a student the freedom to graduate externally, on his own topic, and with the involvement of an (unconnected) fourth supervisor, was quite a leap. I am very happy that it worked out the way it did. Finally, Boudewijn, thank you for your insights and feedback, especially since your time was understandably limited.

Second, for the awesome time in Dublin and their help in arranging this stay abroad, I would like thank the people at Deloitte. In Dublin, the collegues in ERS in general and the members of the security/forensics team in particular, for including me in daily office life, making me feel at home right from the start. Sean Wills, thank you for your help and feedback on "stylistic issues" :-). In the Netherlands, I owe a great deal of thanks to Tom Schuurmans, Derek Wieringa and Marko van Zwam for giving me the opportunity to work abroad. Deloitte NL financially supported this research while fully respecting my independence, and allowed me to publish my full work under open (access) licenses. For all of this I am very grateful.

Finally, I would like to thank Jeroen van Ingen, Dave Wilson and Andrew Mackarel for the opportunity to discuss the practical impact of this work and their insights into practical network topology description. This work would not have been nearly as practical if not for their openness and interest in new research.

[1] And with respect to the others, you know who you are!

Copyright

Source code

The Haskell source code listed in this document originates from the netPropCheck package, which is licensed under the BSD 3-Clause License. The full license can be found in the LICENSE file in the source distribution. netPropCheck can be obtained from `https://rtsn.nl/thesis/`.

Quotations and figures from GFD-R-P.206

In Chapters 2 and 4 quotations and figures are used sourced from GFD-R-P.206 "Network Markup Language Base Schema version 1" [17]. The following copyright statement applies to that content.

> *Copyright ©Open Grid Forum (2008-2013). Some Rights Reserved.*
>
> *This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included as references to the derived portions on all such copies and derivative works. The published OGF document from which such works are derived, however, may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing new or updated OGF documents in conformance with the procedures defined in the OGF Document Process, or as required to translate it into languages other than English. OGF, with the approval of its board, may remove this restriction for inclusion of OGF document content for the purpose of producing standards in cooperation with other international standards bodies.*
>
> *The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.*