Framework for path finding in multi-layer transport networks

Dijkstra, F.

Publication date
2009
Document Version
Final published version

Chapter 3

Going in Loops: Path Finding in Multi-Layer Networks

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

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

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

Path finding: the calculation of the shortest viable path;

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

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

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

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

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

3.1 Algorithms

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

3.1.1 Breadth-first and Depth-first

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

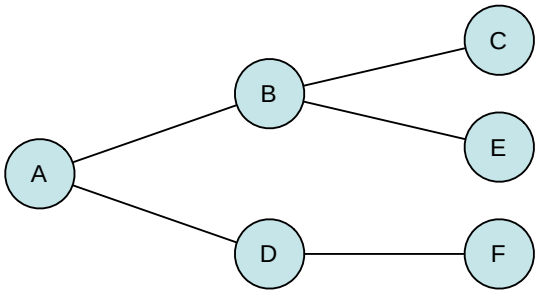


Figure 3.1: *A simple tree graph*

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

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

3.1.2 Bellman-Ford and Dijkstra Algorithms

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

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

3.1.3 Constrained Shortest Path First

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

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

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

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

3.1.4 Path-Constraint Algorithms

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

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

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

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

3.1.5 k-Shortest Path

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

3.2 Routing Protocols

3.2.1 Distributed Path Finding

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

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

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

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

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

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

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

3.2.2 The Internet

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

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

3.2.3 Public Switched Telephone Service

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

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

3.2.4 Generalized Multiprotocol Label Switching

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

The GMPLS framework works as follows:

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

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

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

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

GMPLS deploys three mechanisms to confine scalability within reasonable limits:

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

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

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

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

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

3.3 Path Finding in Multi-Layer Networks

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

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

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

network may provide dynamic switching capabilities using Ethernet PBB-TE, and a third network using DWDM lightpaths. So as long as technologies evolve, multi-domain networks are also multi-layer networks.

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

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

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

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

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

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

The above claims are proven in the following subsections.

The root of both claims is that path finding in single layer networks is a problem in complexity class P, that is, a decision problem that is verifiable in polynomial time by a non-deterministic Turing machine. Algorithms like

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

3.3.1 Practical Example

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

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

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

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

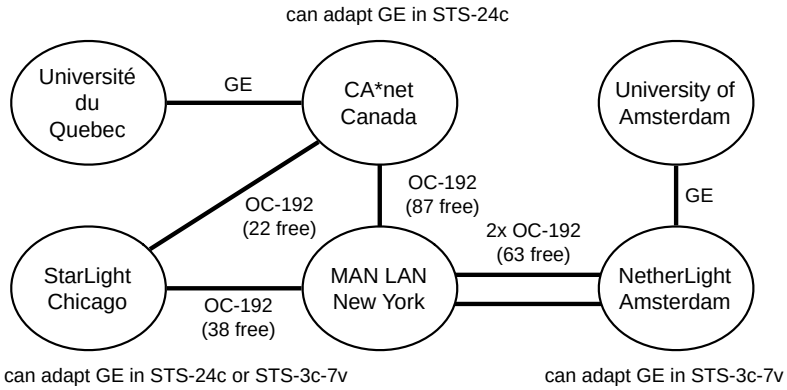


Figure 3.2: *Example of a multi-layer and multi-domain network.*

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

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

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

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

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

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

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

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

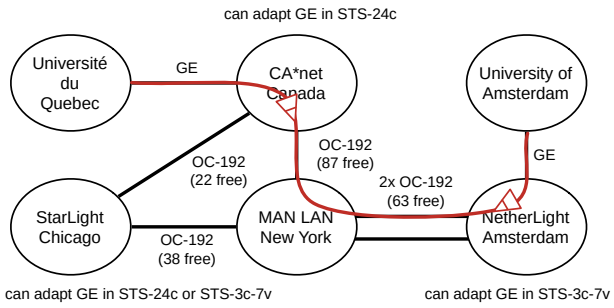


Figure 3.3: *An invalid network connection: the adaptation of gigabit Ethernet (GE) in 24 STS channels (STS-24c) is incompatible with the adaptation of gigabit Ethernet in 21 STS channels (STS-3c-7v).*

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

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

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

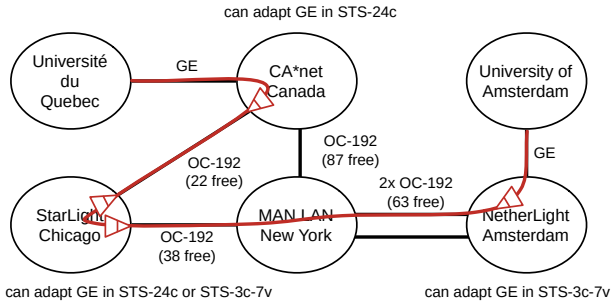


Figure 3.4: *An invalid network connection: there are only 22 STS channels available between CAnet and StarLight, while 24 are required.*

available between CAnet and StarLight, while 24 are required.

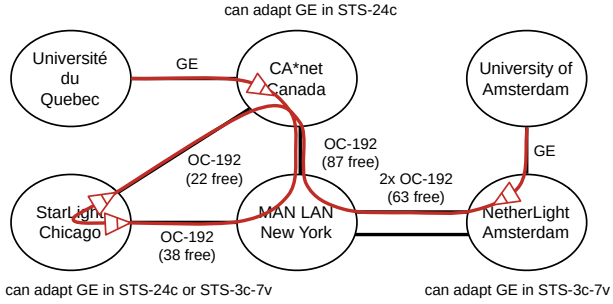


Figure 3.5: *The shortest valid network connection from Université du Quebec to University of Amsterdam through the example network of figure 3.2.*

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

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

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

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

3.3.2 Path-Constrained Problem

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

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

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

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

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

We can proof this claim by contradiction. Lets assume that graphs could be used for path finding in multi-layer networks, while all conditions hold. By condition (3), path finding in the graph would be solvable (and thus also verifiable) in polynomial time, as link-constrained path finding algorithms (without

further conditions) have P complexity. By condition (1), this would give a solution for the original problem, and by condition (2) the solution for path finding in multi-layer networks would be solvable in polynomial time as well. However, Kuipers has shown that the multi-layer problem is NP-hard, so this would lead to a contradiction. So we must conclude that our claim is true.

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

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

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

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

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

3.3.3 Graphs

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

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

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

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

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

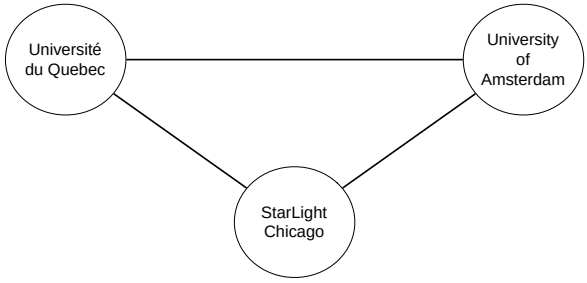


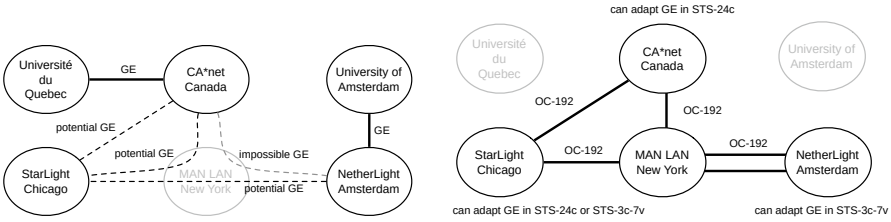
Figure 3.6: *Graph where each edge represents the shortest path in the network of figure 3.2.*

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

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

3.3.4 Multi-Layer Representations

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



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

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

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

thus also not the compatibilities and incompatibilities caused by adaptation functions.

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

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

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

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

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

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

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

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

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

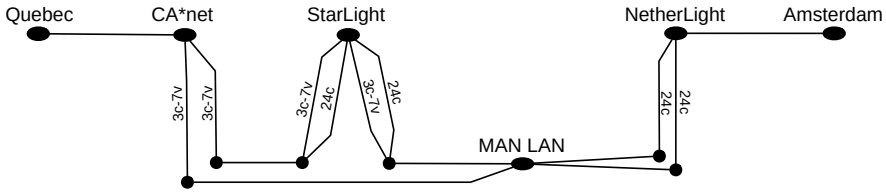


Figure 3.8: *The network of figure 3.2, modelled as graph with edge properties.*

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

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

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

3.4 Path Finding in Transport Networks

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

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

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

In fact, at least four different types of information may be required to find

a valid path:

Topology information – the interconnections between devices and between domains;

Technology information – the potential technical incompatibilities;

Scheduling information – the simultaneous availability of required resources;

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

Path Finding Software

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

3.5 Multi-Stage Path Finding

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- an access method to check for usage availability.

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

3.6 Conclusion

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

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

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

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

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