Computer Communications 32 (2009) 78–85

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Path selection in multi-layer networks

Fernando Kuipers [a,*], Freek Dijkstra [b]

[a] Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands
[b] University of Amsterdam, Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 20 October 2007
Received in revised form 12 September 2008
Accepted 15 September 2008
Available online 7 October 2008

Keywords:
Path selection
Multi-layer network
Network description

ABSTRACT

Multi-layer networks are computer networks where the configuration of the network can be changed dynamically at multiple layers. However, in practice, technologies at different layers may be incompatible to each other, which necessitates a careful choice of a multi-layer network model. Not much work has been done on path selection in multi-layer networks. In this paper, we describe how to represent a multi-layer network and we provide algorithms for selecting paths in them. Throughout the paper we will use examples drawn from practical experience with routing in hybrid optical networks.

1. Introduction

Multi-layer networks are computer networks where the configuration of the network can be changed dynamically at multiple layers. An example of a multi-layer network is an optical network where both the WDM, TDM (SONET/SDH) and Ethernet layers can be dynamically re-configured. In an ideal case, we would be able to model a multi-layer network by a simple graph, consisting of nodes and links. However, nodes at different layers may be incompatible to each other, which necessitates a careful choice of our multi-layer network model.

We will first explain the terminology and notation that is used. A computer network consists of a set $\mathscr{D}$ of D network devices. A multi-layer network consists of a set $\mathscr{L}$ of L network layers and the set $\mathscr{T}$ of all technologies. In this paper, we define technologies as a specific data encoding. A layer is a group of one or more technologies $\mathscr{T}(l)$. The distinction between layer and technology follows the actual distinction made by network engineers. Different technologies represent incompatible data formats. We consider two different incompatibilities: incompatible *labels*, after the term used in GMPLS [8], and incompatible *adaptations*. For example two different wavelengths on a fiber are regarded as two incompatible labels, and thus two technologies, which are grouped in the same layer, the wavelength division multiplexing (WDM) layer. 1 Gigabit/second Ethernet can be embedded or *adapted* in SONET STS channels in at least three different ways (requiring 48, 24 or 21 individual STS channels per Gigabit Ethernet channel). We regard incompatible adaptations $a \in \mathscr{A}(l)$ as different technologies of the server layer l of the adaptations.

We will define multiple approaches to map a network on a graph $G = (\mathscr{N}, \mathscr{E})$ consisting of a set $\mathscr{N}$ of N nodes and a set $\mathscr{E}$ of E edges. Nodes represent either devices or interfaces in a network, while the edges represent either communication links or adaptation functions between network layers. A specific edge in the set $\mathscr{E}$ between nodes u and v is denoted by (u, v). Each edge $(u, v) \in \mathscr{E}$ from node u to node v is characterized by a single weight, or weight vector if there are $m > 1$ measures characterizing an edge. In case of additive measures, the weight of the measure along a path is the sum of the weights on the edges defining that path. For min–max measures, the path weight is the minimum (or maximum) of the weights of the edges that constitute that path.

The rest of this paper is organized as follows. In Section 2, we present three representations of a multi-layer network. Section 3 elaborates on the complexity of path selection in multi-layer graphs. In Sections 4 and 5, we discuss two path selection algorithms for multi-layer graphs consisting of two layers. Section 6 extends this work to an arbitrary number of layers. We end with the conclusions in Section 7.

2. Multi-layer network model

In this section, we provide three network descriptions. The first is a commonly used model in which each network device represents one node in a graph G, and physical network links are represented as edges. The second model represents each network device as multiple nodes in a graph G_l; one for each "layer". In this model, links still represent edges, but adaptations also represent edges.

* Corresponding author. Tel.: +31152781347.
E-mail addresses: F.A.Kuipers@tudelft.nl (F. Kuipers), F.Dijkstra@uva.nl (F. Dijkstra).

0140-3664/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2008.09.026

Finally, a model where we transform the multi-layer network into a graph G_s consisting of nodes and links on different "technologies".

2.1. Device-based network description G

Fig. 1 gives an example of a multi-layer network. This network consists of six devices, $\mathscr{D} = \{A, B, C, D, E, F\}$, and we only consider two layers $\mathscr{L} = \{\text{Ethernet}, \text{STS}\}$, ignoring the optical carrier (OC) layer for simplicity. There are two incompatible adaptations: Gigabit Ethernet (GE) can either be adapted in 24 STS channels or in 21 STS channels (seven virtually concatenated groups of three concatenated channels). We represent this as $\mathscr{A}(STS) = \{24c, 3c7v\}$. The network has six physical links, as shown in Fig. 1, and not all devices support all adaptations. Devices *A* and *G* are only aware of the Ethernet layer, and not of the STS layer, while device *E* only has knowledge about the STS layer, and has no knowledge about Ethernet. In the given network the shortest path from *A* to *C* is not *A–B–E–F–C*, because then GE is adapted in STS-24c at node *B*, but cannot be de-adapted back into GE at node *F*. Also, *A–B–D–E–F–C* is not a valid path, because only 22 STS channels are available between *B* and *D*, where 24 are required. The shortest correct path in this example is *A–B–E–D–B–E–F–C*. Note that this path uses the edge *B–E* twice. Consequently, our path-finding algorithm will have to take the (de)adaptation functions into account.

The graph *G* in Fig. 1 is a fairly common way to describe the physical properties of a network, with devices represented as nodes, and (physical) links as edges. All information on (de)adaptation capabilities is present in the nodes, but is not explicit in the graph defined by $G = (\mathscr{N}, \mathscr{E})$, or in another format readable for regular path-finding algorithms. In the next two sections, we present the graphs G_l and G_s that do contain this information.

2.2. Layer-based network description G_l

Given the set $\mathscr{L}$ of the *L* network layers, the set $\mathscr{D}$ of the *D* network devices, and the set $\mathscr{A}$ of *A* different adaptation functions, we create the graph $G_l = (\mathscr{N}_l, \mathscr{E}_l)$ as follows. The set $\mathscr{N}_l$ consist of all nodes $n(d,l)$ for all devices $d \in \mathscr{D}$ and all layers $l \in \mathscr{L}$, provided that the device *d* "has knowledge of" layer *l*. The set $\mathscr{E}_l$ is the union $\mathscr{E}_{lA} \bigcup \mathscr{E}_{lL}$. An edge $(u(p,l), v(q,l))$ exists in $\mathscr{E}_{lL}$ if the devices *p* and *q* can directly communicate with each other without any (de)adaptations on either side. Thus, $\mathscr{E}_{lL}$ consists of all edges representing physical links.

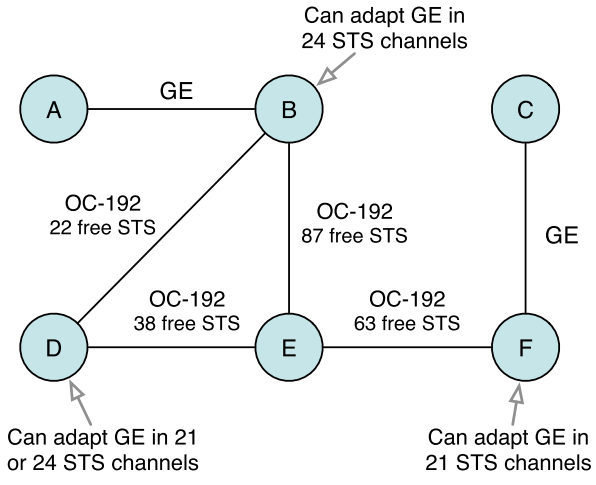

Fig. 1. An example of a multi-layer and multi-domain network, as presented in [5]. GE refers to Gigabit/second Ethernet, OC-192 is a SONET-based optical carrier carrying 192 STS channels.

Only links on the "lowest" layer are represented as edges. $\mathscr{E}_{lA}$ is the set of edges (u,v), consisting of one edge $(u(p,l), v(p,m))$ for each adaptation from client layer *l* to server layer *m*, as supported by device *p*. The order *l*,*m* is important.

Fig. 2 shows the graph G_l for the network described in Fig. 1.

We have a maximum of $D \times L = 6 \times 2 = 12$ nodes in $\mathscr{N}_l$, although we only draw nine of them: devices *A* and *C* are not aware of the STS layer, and device *E* has no knowledge of Ethernet. $\mathscr{E}_{lL}$ consists of six edges, representing the six links present in Fig. 1. However, the Ethernet links between devices *p* and *q* are represented as edges $(u(p,Ethernet), v(q,Ethernet))$, while STS/SONET links between devices *p* and *q* are represented as edges $(u(p,STS), v(q,STS))$. $\mathscr{E}_{lA}$ consists of four edges, representing the support for a certain adaptation function $a \in \mathscr{A}$. The difference between client and server layer in an adaptation is signified by the triangles in the edges (the standard graphical representation of adaptation in ITU-T G.805 [7]).

2.3. Stack-based network description G_s

In our last model, not the layers are represented, but the specific technology stacks. Our goal is to come to a, in the algorithmic sense, simple network description, which only consists of nodes and edges. (De)adaptations from one technology to another are not represented as a function, but simply by vertical edges. For instance, consider the example network in Fig. 1. We can identify three different adaptation stacks: $\mathscr{S}$ = {Ethernet, Ethernet over 24 STS channels, Ethernet over 21 STS channels}.

We construct the graph $G_s = (\mathscr{N}_s, \mathscr{E}_s)$ of a multi-layer network as follows. If the network consists of *D* devices and *S* different technology stacks, our graph will consist of maximally $N_s = S \times D$ nodes. The nodes are aligned as a matrix consisting of *S* rows and *D* columns (a similar approach has also been deployed in the context of wavelength routing in WDM networks [2]). Nodes on the same row have an edge between them if they can directly communicate with each other without any (de)adaptations. An edge from a node in one row to a node in another row represents a (de)adaptation. By assigning weights to the horizontal edges we can represent the cost of using an edge, and by assigning weights to the vertical edges we can represent the cost of (de)adaptation.

In Fig. 3, we have given the new representation (with edge weights for the link capacity) of the example network displayed in Fig. 1. Node *B* can only adapt Ethernet in 24 channels (and de-adapt back), while node *D* can adapt Ethernet either into 24 channels or into 21 channels. Nodes on the same row that are linked together by a fixed line can communicate with each other without any (de)adaptations. The dotted line, states that in theory these nodes should be able to communicate with each other, but in this case not enough (<24) channels are available. We disregard this edge in our path computation (topology filtering).

3. Complexity of multi-layer path selection

In the previous section, we introduced different ways of representing a multi-layer network. Given such representations, we would like to develop algorithms for routing in multi-layer networks. Hence, we need to solve the multi-layer path selection (MLPS) problem, which is defined as the problem of finding the shortest feasible path from a source to a destination in a multi-layer network. Unfortunately, this problem is NP-complete, as shown below.

Theorem 1. *The MLPS problem is NP-complete.*

Let us first define the one-in-three 3SAT problem [6], which is used in the proof of Theorem 1.

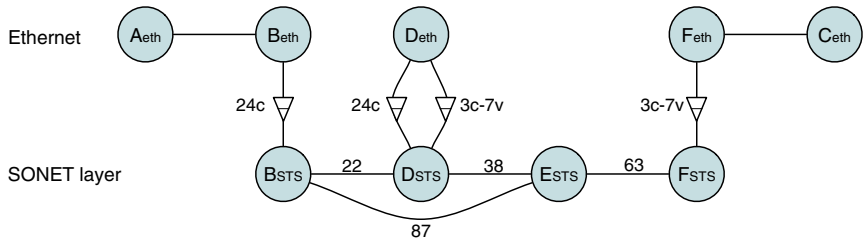

Fig. 2. The layer-based representation G_l of graph G in Fig. 1.

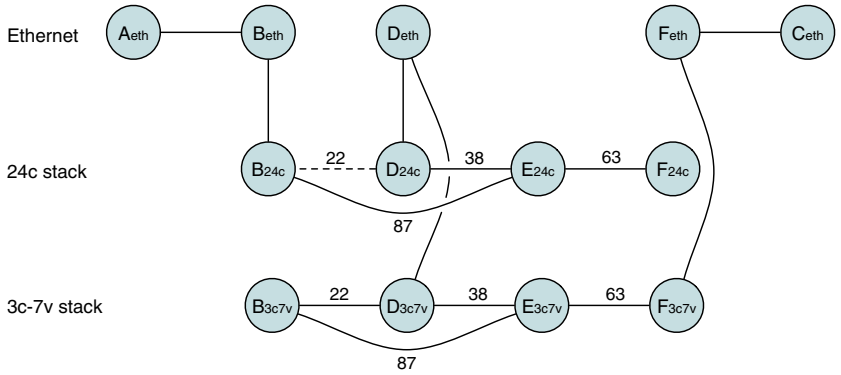

Fig. 3. Representation of the network in Fig. 1 as a multi-layered graph.

Definition 1. One-in-three 3SAT: Given a set $U = \{u_1, \ldots, u_k\}$ of Boolean variables, where u_i and $\bar{u}_i$ are literals of u_i, and a collection C of clauses over U, such that each clause $c \in C$ contains three literals ($|c| = 3$). A clause is satisfied by a truth assignment if and only if one of its members is true. A collection C of clauses over U is satisfiable if and only if there exists some truth assignment for U that simultaneously satisfies all the clauses of C. The one-in-three 3SAT problem asks whether there is a truth assignment for U such that each clause has exactly one true literal.

Proof. The proof of Theorem 1 is provided for the graph G_s, which is obtainable from G in polynomial time, and which therefore also holds for G. If the use of one link cannot prevent the use of another link in G_s (e.g., in case of "enough" available bandwidth), then the problem is polynomially solvable. In this proof, we therefore consider the other extreme, where the use of a link from A_j to B_j on stack layer j prevents the use of the link (if it exists) from A_i to B_i on any other stack layer $i \neq j$. Thus, the use of a link in G_s may prevent the use of one or more other links in G_s. It is easy to verify whether a given path is a feasible path and hence MLPS is in NP. We continue by showing that the NP-complete one-in-three 3SAT problem can be transformed to the MLPS problem in polynomial time.

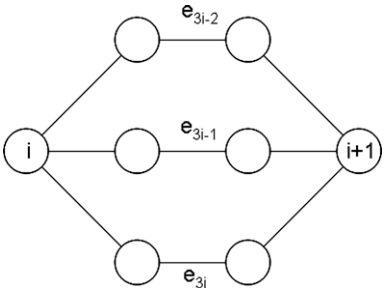

Fig. 4. Graph representation of a clause in one-in-three 3SAT.

For each clause $c_i \in C$ we construct a subgraph as displayed in Fig. 4.

Each of the links $\{e_{3i-2}, e_{3i-1}, e_{3i}\}$ represents a literal of a Boolean variable in U. If we concatenate the graphs of all clauses, we obtain a graph representation of the collection C of clauses. The use of a link in one clause only prevents the use of a link in another clause if the corresponding literals are negations of each other. A (simple) path will only use one link per clause. By finding a path from source $i = 1$ to destination $|C| + 1$ (if it exists), we can retrieve a solution to one-in-three 3SAT by setting all literals corresponding to the links of the path to true. Consequently, the MLPS problem is NP-complete.

In the following sections, we present exact algorithms for path selection in G_l and G_s. Given the NP-complete nature of the problem, the algorithms have an exponential running time.

4. Path selection in G_l

Given a source node s and a destination node t in $G_l = (\mathcal{N}_l, \mathcal{E}_l)$, our objective is to find the path P^* for which $w(P^*) \leqslant w(P)$ for all feasible paths P between s and t. As discussed in Section 2, we may have to pass through nodes and even edges multiple times, and therefore cannot simply use Dijkstra's shortest paths algorithm [4]. Below we present MULTI-LAYER-BREADTH-FIRST(G_l, s, t, B) to compute the shortest path P^* from s to t in G_l. This algorithm has two main features: (1) we keep track of the (de)adaptations along the path (similar to label stacking in MPLS) and (2) multiple paths may be stored at a single node (similar to a k-shortest paths algorithm). Instead of working with a queue of nodes, as in the breadth-first-search algorithm and Dijkstra's algorithm, we maintain a queue of paths. Basically, this algorithm simply tries all possible paths P in order of length, even those with loops or already visited nodes.

Each path object p in queue Q contains the following properties:

(1) sequence of edges $M(p)$;

(2) the last visited vertex $V(p)$
(3) technology stack $S(p)$;
(4) the list of used bandwidths $B(p,e)$ for every edge $e \in M(p)$;
(5) distance $w(p)$.

MULTI-LAYER-BREADTH-FIRST(G_l, s, t, B)

1. $p \leftarrow$ path with $M(p) = \{s\}$; $S(p) = \{\}$; $w(p) = 0$; $B(p,e) = 0$ for all $e \in \mathscr{E}_l$
2. INSERT(Q, p)/*Queue path p*/
3. **while** $Q \neq \emptyset$
4. EXTRACT-MIN(Q) $\rightarrow p$
5. **if** $V(p) = t$ and $S(p) = \emptyset$
6. **return** p
7. **else**
8. **for** each $v \in adj[V(p)]$/*for each adjacent node v of p/
9. EXTEND-PATH(p, v) $\rightarrow p_{new}$
10. **if** $p_{new} \neq$ **unfeasible**
11. INSERT(Q, p_{new})

EXTEND-PATH(p, v):

/*p is a path, v an adjacent node, connected with the edge $(V(p), v)$*/
/*B represents the requested capacity that corresponds to this stack. $b(u,v)$ is the available capacity between vertices u and v.*/
12. $p_{new} \leftarrow p$
13. INSERT($M(p_{new}), (V(p), v)$)/*Add edge $(V(p), v)$ to the list of edges*/
14. $V(p_{new}) \leftarrow v$
15. $B(p_{new}, (V(p), v)) \leftarrow B(p, (V(p), v)) + B$
16. **if** $B(p_{new}, (V(p), v)) > b(V(p), v)$/*no bandwidth available*/
17. **return unfeasible**
18. $w(p_{new}) = w(p) + w(V(p), v)$
19. **if** edge $(V(p), v)$ represents an adaptation a_e
20. /*$V(p)$ is client layer and v server layer of a_e*/
21. PUSH($S(p_{new}), a_e$)/*add adaptation to the stack*/
22. **else if** edge $(V(p), v)$ represents a de-adaptation a_e
22. /*$V(p)$ is the server layer and v the client layer of a_e*/
23. **if** POP($S(p)$) $\neq a_e$/*wrong de-adaptation*/
24. **return unfeasible**
25. **if** $g(V(p), p_{new}) \cap g(v, p_{new}) = \emptyset$/*incompatible labels*/
26. **return unfeasible**
27. **return** p_{new}

The base of the algorithm is a queue Q of path objects. Each path object p has the following properties (1) sequence of visited edges $M(p)$, (2) current technology stack $S(p)$, (3) the set of used bandwidths $B(p,e)$, (4) distance $w(p)$. We further define $V(p)$ to be the last node in p, $a(S(p))$ the last adaptation in the stack $S(p)$, $g(u,p)$ the available labels at node u in path p.

B is the required capacity of the path.

Lines 1–2 initialize the algorithm with a path starting at node s, and without an adaptation in the technology stack. Lines 3–11 form the main loop. It takes the path with the shortest length from the queue, and extends it by one hop in all directions. Line 5 checks if the shortest path ends at the destination, and if so, returns that path as the result of the algorithm. Not all extensions of a path with one hop will result in a feasible path. Line 10 checks for this condition, and only considers feasible paths. The actual extension of the path and the feasibility check is done in the Extend-Path routine. This takes the existing path p and extends it via edge $(V(p), v)$ to node v. It sets the four properties of the path accordingly. In case of an adaptation (lines 20–21), the new adaptation is added to the stack. In case of de-adaptation, the last adaptation is removed from the stack (lines 22–24), but only if the de-adaptation is equal to the last adaptation on the stack. Lines 25–26 are an extension to the algorithm to check if two adjacent nodes have a common channel available for the transmission of data.

5. Path selection in G_s

If we would apply the Dijkstra algorithm to the graph G in Fig. 1 with source node A and destination node C, we would find the path A–B–E–F–C, which is not a feasible path. By applying the Dijkstra algorithm to the graph G_s in Fig. 3, we would find the path A_{Eth}–B_{Eth}–B_{24c}–E_{24c}–D_{24c}–D_{Eth}–D_{3c7v}–E_{3c7v}–F_{3c7v}–F_{Eth}–C_{Eth}, which relates to path A–B–E–D–E–F–C. If the link between D and E would have enough capacity, this would result in a feasible path, which is not the case here. The correct path is A–B–E–D–B–E–F–C. Consequently (contrary to the approach in [2]), we have to modify our algorithm to account for capacity on physical links traversed multiple times. This can be solved by examining whether the path, which is to be extended by a certain edge, did not use that edge before. If so, we need to check for enough bandwidth. The case of insufficient bandwidth may lead to the property that *subsections of shortest paths are not necessarily shortest themselves*. The NP-complete multi-constrained path (MCP) problem also holds this property. We have developed the exact algorithm SAMCRA [9] for the MCP problem, which uses a non-linear length function, a k-shortest paths approach, and search space reducing techniques. Our algorithm for path selection in G_s is named MULTI-LAYER-DIJKSTRA (G_s, s, t, B) and will also adopt a k-shortest paths approach. The meta-code of MULTI-LAYER-DIJKSTRA(G_s, s, t, B) is given below.

MULTI-LAYER-DIJKSTRA(G_s, s, t, B)

1. DIJKSTRA(G_s, t) $\rightarrow r[v]$/*Lower bounds for all nodes*/
2. **for** all nodes $v \in \mathscr{N}_s$
3. $d[v] \leftarrow \infty$/*The distance to s*/
4. $\pi[v] \leftarrow$ NIL/*The predecessor node to s*/
5. counter$[v] \leftarrow 0$
6. $maxlength \leftarrow \infty$
7. $d[s] \leftarrow 0$
8. counter$[s] \leftarrow$ counter$[s] + 1$
9. INSERT(Q, s, counter$[s]$, NIL, $d[s] + r[s]$)
10. **while** $Q \neq \emptyset$
11. EXTRACT-MIN(Q) $\rightarrow u[i]$
12. **if** $u[i] = t$
13. **return** path
14. **else**
15. **for** each $v \in adj[u[i]]$/*for each neighbor v of $u[i]$*/
16. **if** $d[u[i]] + w(u[i], v) + r[v] <$ $maxlength$
17. **if** FEASIBLE ($u[i], v, B$)/*Backtracking*/
18. $d[v] \leftarrow d[u[i]] + w(u[i], v)$
19. $\pi[v] \leftarrow u[i]$
20. counter$[v] \leftarrow$ counter$[v] + 1$
21. INSERT (Q, v, counter [v], $u[i], d[v] + r[v]$)
22. **if** $v = t$
23. $maxlength \leftarrow d[v] + r[v]$

FEASIBLE($u[i], v, B$)

/*Backtracking to check whether the path is loop-free and has enough capacity available. $u[i]$ is our current node, v is the node under consideration for extending our path with. We assume that it is known to which stack $u[i]$ and v belong. B represents the requested capacity that corresponds to this stack.*/
24. $x \leftarrow u$
25. $y \leftarrow i$
26. $B' \leftarrow b(u, v)$

27. **while**($\pi[x[y]] \neq$ NIL)
28. **if** $L(\pi[x[y]],x) = L(u[i],v)$
29. $B' \leftarrow B' - B$
30. $x[y] \leftarrow \pi[x[y]]$
31. **if** $x = v$
32. **return** FALSE
33. **if** $B' < B$
34. **return** FALSE
35. **else**
36. **return** TRUE

Our objective is to find a shortest path from s to t in G_s, which has at least bandwidth of B. $d[v]$ gives the shortest found distance from s to v and can only decrease during the course of the algorithm. $\pi[v]$ gives the predecessor of node v that was used to reach node v with distance $d[v]$. counter[v] refers to the number of paths stored at node v and *maxlength* refers to the maximum length that a (sub)path may have. $w(u,v)$ and $b(u,v)$ give the weight and the available bandwidth on the edge (u,v), respectively. Formally, we define $c(e)$ to be the available capacity of the physical link e. $L(u,v)$ defines the relation of a given edge (u,v) to the corresponding physical link. As we constructed nodes n as $n(d,s)$ for device d and technology stack s, $L(u,v) = L(u(d_u,s_u),v(d_v,s_v))$ is the physical link between devices d_u and d_v. For simplicity, we define $b(u,v) = c(L(u,v))$. C is the set of capacities for all physical links in the network. B is the required capacity of the path, which may also differ per stack (again, for ease of notation, we have removed the subscript per stack).

Lines 1–9 of the meta-code initialize all nodes. Line 1 computes the shortest paths from t to all other nodes in the graph by one execution of the Dijkstra shortest paths algorithm that disregards any bandwidth constraints. The weights of these paths serve as lower bound estimates, referred to as $r[v]$ for all nodes $v \in \mathcal{N}_s$. Note that if the shortest path from s to t is feasible, we can stop the algorithm and return this solution. Else the algorithm should proceed. Line 9 inserts the source node with predicted length $d[s] + r[s] = r[s]$ and predecessor NIL in the queue Q, which was initially empty. The main algorithm starts at line 10. Line 11 extracts the node $u[i]$ from the queue that has the shortest weight. Since multiple paths can be stored at a node u, $u[i]$ is used to denote the ith path at node u. Not entire paths are stored, but by backtracking the predecessor list $\vec{\pi}$, the entire path can be reconstructed. Node $u[i]$ can be regarded as the new scanning node towards destination t. If $u[i] = t$ we have found the shortest feasible path, else we continue. Lines 15–21 perform the relaxation procedure [3] for each adjacent node v of u. Line 16 checks whether the length of the path extended from $u[i]$ to v does not exceed *maxlength*, otherwise it is discarded because we already have a better candidate. If this first test is passed, we continue by backtracing the predecessor list to check, with the module FEASIBLE(), for loops and for enough available bandwidth on the physical link (u,v). If these tests are also passed, we may insert v into the queue.

The complexity of MULTI-LAYER-DIJKSTRA(G_s,s,t,B) can be computed as follows. Lines 1–9 have the same complexity as of Dijkstra's algorithm, namely $O(N_s \log N_s + E_s)$. Let us denote by $k_{\max}$ the maximum number of paths stored at any node. When using a Fibonacci or Relaxed heap to structure the queue, selecting the minimum path length among $k_{\max}N_s$ different path lengths takes at most a calculation time of the order of $O(\log(k_{\max}N))$ [3]. As each node can be selected at most $k_{\max}$ times from the queue, the EXTRACT-MIN function in Line 11 takes $O(k_{\max}N_s \log(k_{\max}N_s))$ at most. The for-loop starting on Line 15 is invoked at most $k_{\max}$ times from each side of each link in the graph, leading to $O(k_{\max}E_s)$. Line 17 takes $O(N_s)$. Combining all these contributions yields a total worst-case complexity for MULTI-LAYER-DIJKSTRA $O(k_{\max}N_s \log(k_{\max}N_s) + k_{\max}N_sE_s)$, where $k_{\max}$ may be exponential in N_s. By restricting $k_{\max}$ at the expense of possibly losing exactness, a heuristic version of MULTI-LAYER-DIJKSTRA is obtained. It is also possible to stop in Line 22, when a feasible (not necessarily shortest) path is found.

6. Extension to incompatible labels

So far, we have only considered two layers, with the adaptations between the two layers as the only technological incompatibility. All algorithms can easily be extended to support multiple layers. In this section we will illustrate this with the example network in Fig. 5.

The example network of Fig. 5 consists of seven devices, three layers $\mathscr{L} = \{\text{Ethernet}, \text{SONET}, \text{WDM}\}$, with two incompatibilities at the SONET layer, the previously mentioned STS-24c and STS-3c-7v adaptations, and two incompatibilities at the WDM layer, wavelengths of 1310 nm and 1550 nm. Devices A and G are pure Ethernet devices, devices B, D, E and F are SONET devices, with B, D and F capable of (de)adapting Ethernet in SONET, and C is an optical cross connect. Devices B and F are equipped with 1310 nm lasers, device E is equipped with 1550 nm lasers, and device D has tunable lasers.

The shortest working path from A to G is A–B–C–D–C–E–C–D–C–F–G. B and F cannot directly communicate due to the different adaptation of Ethernet in SONET, and B and E and also E and F cannot communicate because of the different wavelengths.

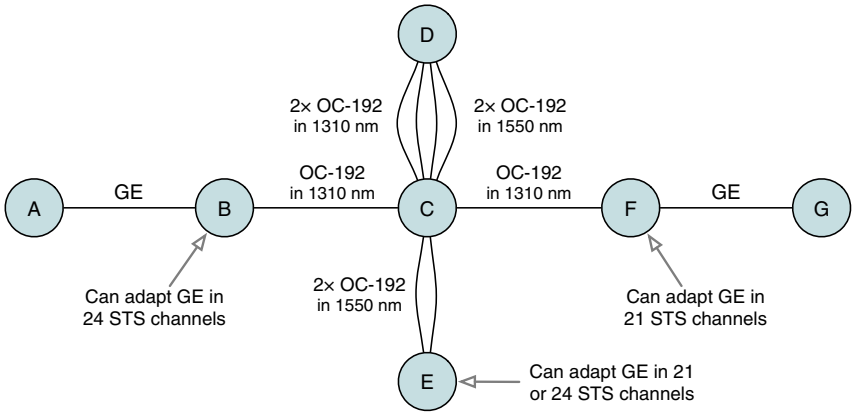

Fig. 5. Example of a three-layer network.

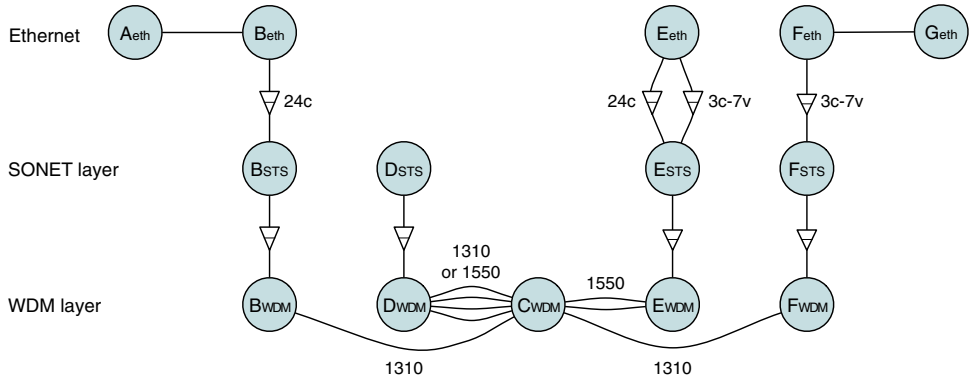

Fig. 6. Graph G_l of the network of Fig. 5.

6.1. Extension to graph G_l

Graph G_l of this network is given in Fig 6. The shortest path through this graph is A_{Eth}–B_{Eth}–$B_{STS}(24c)$–$B_{WDM}(1310)$–$C_{WDM}(1310)$–D_{WDM}–D_{STS}–$D_{WDM}(1550)$–$C_{WDM}(1550)$–E_{WDM}–$E_{STS}(24c^{-1})$–E_{Eth} $(3c$–$7v)$–E_{STS}–$E_{WDM}(1550)$–$C_{WDM}(1550)$–D_{WDM}–D_{STS}–$D_{WDM}(1310)$–$C_{WDM}(1310)$–F_{WDM}–$F_{STS}(3c–7v^{-1})$–D_{Eth}–G_{Eth}, with the label as used on the following edge denoted between brackets. A few new characteristics of G_l emerge. There is only a single adaptation between D_{STS} and D_{WDM}, even though that edge is used four times in the shortest path. In G_l edges can be traversed multiple times, as long as the available bandwidth is not exceeded. On the other hand, there are four edges between D_{WDM} and C_{WDM}, since the actual network has four different physical links.

G_l treats the various incompatibilities differently. Dijkstra et al. [5] mention three distinct incompatibilities: (1) in adaptation, (2) in label (channel identifier), and (3) in other encoding characteristics (e.g., different MTU size for Ethernet). The graph G_l represents different adaptations as different edges, while different labels are represented as different labels or label sets for the edges. See the algorithm in Section 4: the incompatibility check for adaptation (line 23) is different from the incompatibility check for labels (line 27). The other encoding differences are not represented in the graph nor the algorithm. This is a conscious choice: usually there are only a few (if any) incompatible adaptation functions, but many incompatible labels. In addition, G_l allows for *sometimes* incompatible labels. For example, one device may not be able to convert wavelengths, while another device may convert between wavelengths without de-adapting and adapting the wavelength. This is also referred to as *label swapping*. To support this, line 27 in the MULTI-LAYER-BREADTH-FIRST algorithm needs to be changed so that the condition is never True for devices capable of label swapping.

6.2. Extension to graph G_s

Graph G_s of the network of Fig. 5 is given in Fig. 7. The first notable characteristic is the many different adaptation stacks. The five different technologies (one for Ethernet, two for SONET and two for WDM) lead to seven possible adaptation stacks: Ethernet, Ethernet in STS-24c, Ethernet in STS-3c-7v, Ethernet in STS-24c in 1310 nm, Ethernet in STS-24c in 1550 nm, Ethernet in STS-3c-7v in 1310 nm, and Ethernet in STS-3c-7v in 1550 nm. The adaptation stacks Ethernet in STS-24c and Ethernet in STS-3c-7v have been displayed in the figure for the sake of clarity, but can be omitted. Note that the nodes on this stack only have two neighbors. One can filter the topology by removing one-degree nodes and by removing two-degree nodes and connecting their neighbors via a direct link. This form of topology filtering does not only apply to the two adaptation stacks Ethernet in STS-24c and Ethernet in STS-3c-7v, but can be applied iteratively to the graph. However, in the process of filtering, we would have to keep track of the adaptation. The graph G_l can for instance be considered a filtered version of the graph G_s, but one can filter even further. As such, the graph G_s can be considered as the memory with respect to adaptations needed to find a path in a multi-domain multi-layered network.

The shortest path through the graph G_s is A_{Eth}–B_{Eth}–B_{24c}–$B_{24c;1310}$–$C_{24c;1310}$–$D_{24c;1310}$–D_{24c}–$D_{24c;1550}$–$C_{24c;1550}$–$E_{24c;1550}$–E_{24c}–E_{Eth}–E_{3c7v}–$E_{3c7v;1550}$–$C_{3c7v;1550}$–$D_{3c7v;1550}$–D_{3c7v}–$D_{3c7v;1310}$–$C_{3c7v;1310}$–$F_{3c7v;1310}$–F_{3c7v}–D_{Eth}–G_{Eth}.

Contrary to G_l, the shortest path through G_s never traverses the same node twice. In fact, this is a very useful property, since it means we can limit the number of edges between two nodes to at most one edge, even if there are multiple physical links. In our example, the resulting graph G_s is rather efficient. In fact, removing all "dead ends" in the graph of Fig. 7, results in only the links of the shortest path, indicating that there is only one shortest path in this example.

6.3. Discussion and comparison

In multi-layer networks, the number of incompatibilities may grow quite large. In the given example, the WDM layer only has two wavelengths. However, for dense wavelength division multiplexing (DWDM), about 100 different wavelengths are not uncommon. For Ethernet, there are 4096 incompatible VLAN tags (incompatible, since it is uncommon for devices to be able to alter the IEEE 801.1Q tag in packets). Tagged Ethernet over DWDM would thus yield about $4096 + 100 \times 4096 = 413{,}696$ rows in G_s. This contrary to G_l, which would have two rows, but an increased memory space for the algorithm (as reflected in G_s).

For improved scalability, it may be possible to aggregate available channels into groups. For instance, consider an interface with the following VLAN channels available [1,50], [53], [89,93], [106,123], [400,530] and another interface with the channels [20,30], [50,55], [100,110], [3000,4095]. An intersection of these groups yields: [1,19], [20,30], [31,50], [51,52], [53], [54,55], [89,93], [100,105], [106,110], [111,123], [400,530], [3000,4095], which can be represented by 12 (instead of 4096) rows.

If we consider path finding in G_l and path finding in G_s, the similarities between the two algorithms are bigger than their differences:

- Both algorithms keep track of stacks, and find a path starting at the source, extending it hop by hop. G_l does so by explicitly keeping track of paths. G_s does so by keeping track of the shortest route to the source for each possible adaptation stack.

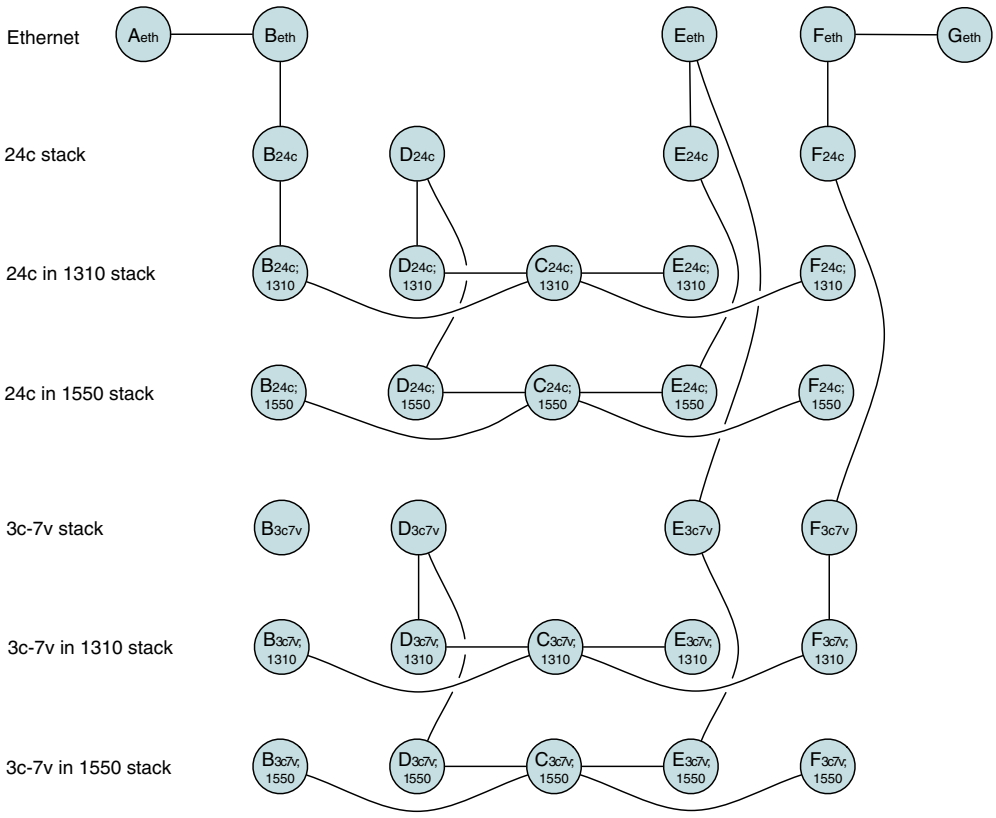

Fig. 7. Graph G_s of the network of Fig. 5.

- Both algorithms are capable of finding feasible paths (possibly using the same physical link multiple times) that do not exceed the available bandwidth.

We summarize the differences between the graphs G_l and G_s and the associated algorithms:

- G_s has a simple graph in the algorithmic sense. It only extends nodes, edges, with a bandwidth shared among those edges representing the same physical link. The shortest path through G_s never visits the same node twice, so there only has to be one edge between different nodes.
- G_s has multiple edges representing the same physical link. Algorithms need to be adapted to take this into account.
- G_l has a simple graph when considered from a practical point of view. It provides a coherent network representation, which is useful to practitioners.
- The edges in G_l representing an adaptation are directed, even in fully bi-directional networks. Moreover, the shortest path through G_l may traverse edges and nodes multiple times.
- The number of nodes in G_s scales in worst case as $\mathcal{O}(D \times t^L)$ with D the number of devices, t the number of technologies (incompatibilities) per layer, and L the number of layers. The number of nodes in G_l scales with $\mathcal{O}(D \times L)$, but the algorithmic complexity of both algorithms is equivalent.

7. Conclusions

In this paper, we have discussed the problem of finding paths in multi-layer networks. It turns out that in order to find a feasible path, we may have to cross a physical link multiple times, resulting in loops and the property that a segment of a shortest path does not have to be a shortest path itself. This MLPS problem is proven to be an NP-complete problem. We have considered modeling a multi-layer network as a graph based on devices and layers (G_l), and a graph based on devices and technology stacks (G_s). For each model we have discussed the problem of finding feasible paths, given an algorithm and discussed the pros and cons of both approaches.

A graph based on layers and nodes is more suitable from a practical point of view to describe actual networks in a multi-domain environment, where domains are reluctant to provide details about their own networks and there is no full topology knowledge. However, the simplicity of the algorithm for the graph based on devices and stacks makes it more suitable for path finding calculations.

Whichever algorithm is used, we have shown that path finding in multi-layer networks is far more complex than path finding in single-layer networks, and that assumptions valid for path finding in single-layer networks no longer hold.

Acknowledgements

This work has been supported by the GigaPort project, which is led by SURFnet and funded by the Dutch Ministry of Economic Affairs under Grant No. BSIK03020.

References

[2] I. Chlamtac, A. Faragó, T. Zhang, Lightpath (wavelength) routing in large WDM networks, IEEE Journal on Selected Areas in Communications 14 (5) (1996).
[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 2000.
[4] E.W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 269–271.
[5] F. Dijkstra, B. Andree, K. Koymans, J. van der Ham, C. de Laat, A multi-layer network model based on ITU-T G.805, Computer Networks 52 (10) (2008) 1927–1937.

[6] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[7] International Telecommunication Union, ITU-T Recommendation G.805: Generic functional architecture of transport networks, Tech. rep., March 2000, see http://www.itu.int/rec/T-REC-G.805.
[8] E. Mannie, Generalized multi-protocol label switching (GMPLS) architecture, RFC 3945 (Proposed Standard), Oct. 2004, see <http://www.ietf.org/rfc/rfc3945.txt>.
[9] P. Van Mieghem, F.A. Kuipers, Concepts of exact quality of service algorithms, IEEE/ACM Transaction on Networking 12 (5) (2004) 851–864.