Path computation in multi-layer networks: Complexity and algorithms

Mohamed Lamine Lamali, Nasreddine Fergani, Johanne Cohen, Hélia Pouyllau

Path computation in multi-layer networks: Complexity and algorithms

Mohamed Lamine Lamali*, Nasreddine Fergani*, Johanne Cohen†, Hélia Pouyllau‡
*Alcatel-Lucent Bell Labs France. France.
†LRI, Univ. Paris-Sud, CNRS, Université Paris-Saclay. France.
†Thales Research & Technology. France.
mohamed_lamine.lamali@alcatel-lucent.com johanne.cohen@lri.fr helia.pouyllau@thalesgroup.com

***Abstract*—Carrier-grade networks comprise several layers where different protocols coexist. Nowadays, most of these networks have different control planes to manage routing on different layers, leading to a suboptimal use of the network resources and additional operational costs. However, some routers are able to encapsulate, decapsulate and convert protocols and act as a liaison between these layers. A unified control plane would be useful to optimize the use of the network resources and automate the routing configurations. Software-Defined Networking (SDN) based architectures, such as OpenFlow, offer a chance to design such a control plane. One of the most important problems to deal with in this design is the path computation process. Classical path computation algorithms cannot resolve the problem as they do not take into account encapsulations and conversions of protocols. In this paper, we propose algorithms to solve this problem and study several cases: Path computation without bandwidth constraint, under bandwidth constraint and under other Quality of Service constraints. We study the complexity and the scalability of our algorithms and evaluate their performances on real topologies. The results show that they outperform the previous ones proposed in the literature.**

***Keywords*—Multi-layer networks; Path computation; Protocol heterogeneity; Unified control plane.**

I. Introduction

Carrier-grade networks generally encompass several layers involving different technologies and protocols. To support some services, such as a Virtual Private Network (VPN), a path across network equipments must be identified and the equipments be configured accordingly. Under stringent requirements of Quality of Service (QoS – e.g., end-to-end delay, geographic zone avoidance, etc.), computing such a path within a single layer is not always possible. Hence, one of the key challenges is to determine the end-to-end path that uses the appropriate *adaptation functions* over the protocols: The mapping from a protocol to another being realized through *encapsulation* (e.g., Ethernet over IP/MPLS [1]), *decapsulation* (the reverse operation) or *conversion* (e.g., IPv4 to IPv6 [2]) functions. Consequently, the path computation process should take into account the adaptation function capabilities of the network equipments in order to ensure *path feasibility*: If a protocol is encapsulated in another one, it must be decapsulated (or unwrapped) further in the path. If several encapsulations are nested, the corresponding decapsulations must occur in the right order. Here, the multi-layer context should be taken in a broad sense: Presence of several protocols and technologies that can be nested, encapsulated, converted, etc.

Preprint of the paper accepted for IEEE INFOCOM 2016.

Dealing with protocol heterogeneity becomes increasingly important nowadays. In addition to the IPv4/IPv6 migration, this heterogeneity appears in tunneling, some architectures (e.g., The Pseudo-Wire architecture [3] allows the emulation – and thus the encapsulation – of lower layer protocols over Packet-Switched Networks), hybrid networks (e.g., National Research and Education Networks – NRENs – which may have optical and IP interconnection points), and last but not least, most carrier-grade networks, which have separate control planes for IP and Transport layers. In all these contexts, a unified control plane would be very useful for optimizing the network resources and reduce operational and management costs.

OpenFlow is a chance to design such a control plane. Some previous works [4], [5] present an OpenFlow-based architecture to achieve this challenge, but they only focus on the convergence of packet and circuit networks. Other works tackle the traffic engineering problem in SDNs but circumscribe it on a single layer [6] or in the IPv4/IPv6 migration context [7]. However, an important problem to solve remains the path computation process in a multi-layer context. Taking into account the adaptation functions is not trivial and classical algorithms such as Dijkstra's one [8] cannot achieve the task as they do not handle these functions.

Here, we design several algorithms to compute shortest paths dealing with protocol changes and adaptation functions.

Our contributions:

1) We widely generalize the model and the polynomial algorithms described in Lamali *et al.* [9], [10] to perform path computation in multi-layer networks (without bandwidth constraint). Our model takes into account all possible types of protocol changes (encapsulation, conversion, etc.) and any additive metric. We drastically improve the algorithm complexity and realize the first implementation, showing their efficiency on two real topologies.
2) For simulation purposes, we empirically study the distribution of adaptation functions over the network nodes and its impact on feasible path existence. We exhibit a phase transition phenomenon, i.e., a gap where the probability of existence of a feasible path hugely increases.
3) We prove that path computation in multi-layer networks under bandwidth constraint is NP-complete even with two protocols and on symmetric graphs,

thus improving a result of Kuipers and Dijkstra [11]. We also obtain results on the complexity of some subproblems: It is polynomial on *Directed Acyclic Graphs* (DAG) and the general problem is not approximable. We propose a new heuristic to resolve the problem and show its efficiency through simulations.

4) We propose the first algorithm to perform path computation in multi-layer networks under several QoS constraints by adapting the Self-Adaptive Multiple Constraints Routing Algorithm (SAMCRA – Van Mieghem and Kuipers [12]) to the multi-layer context. We study its scalability through simulations.

The paper is organized as follows: Section II describes the problem of path computation in multi-layer networks and recalls the related work; Section III formalizes the problem and describes our model of multi-layer network; Section IV proposes algorithms to perform path computation without bandwidth constraint and shows their efficiency through simulations, it also studies the phase transition phenomenon in multi-layer networks; Section V studies the complexity of path computation under bandwidth constraint and proposes heuristic solutions to tackle the problem; Section VI proposes the first algorithm computing paths under additive QoS constraints and studies its scalability; finally, Section VII concludes the paper.

II. Path computation in Multi-Layer Networks

A. Connectivity in multi-layer networks

We aim to present the different concepts of path computation in multi-layer networks through an example. While this example relates to multi-domain multi-layer networks, the underlying problem of path computation is the same as in a single domain network[1]. Figure 1 (inspired by the Inter-Provider Reference Model [14]) depicts a network involving multiple domains and adaptation function capabilities of network equipments: A company owning a Local Area Network (LAN) wishes the Virtual Machines (VMs) of a data-center to be within the same routing domain (for instance through a Layer 2 VPN or a Generic Routing Encapsulation tunnel). Hence, the switches of the LAN and the VMs of the data-center must communicate through Ethernet datagrams and a path has to be determined across the Domains 1 and 2.

On Figure 1, Domains 1 and 2 use IPv6/MPLS-TE technology and are linked by equipments providing Ethernet encapsulation and decapsulation. The Provider Edge (PE) of Domain 1 is linked to the Customer Edge (CE) of the data-center. The adaptation capabilities of each node are shown above it. An example of feasible path would cross the PE of Domain 1 converting IPv4 packets into IPv6 ones. Then it would apply the encapsulation and decapsulation of the border routers of Domains 1 and 2 respectively and the PE of Domain 2 would apply a conversion of IPv6 packets into IPv4 ones. The protocol stacks of the packets at each stage are illustrated at the bottom of Figure 1. As an example of unfeasible path, a direct Ethernet connection between the CE of the data-center and the border router of Domain 1 appears. This configuration leads to a decapsulation of an IPv6 packet from an Ethernet datagram (by the border router of Domain 2) whereas at this stage the datagram encapsulates IPv4 packets.

This example depicts the constraints to comply with when computing a multi-layer (and multi-domain, in this case) path: Being physically linked is not sufficient to establish connectivity. Protocol continuity (by analogy with wavelength continuity in optical networks) must hold and the adaptation functions should occur in the right order. Moreover, feasible paths can involve loops and their subpaths are not necessary feasible [11], [15]. Nowadays, such paths are manually determined and configurations are operated and applied by scripts.

B. Related work

The initial works dealing with protocol and technology heterogeneity circumscribed the problem at the optical layer. For instance, Chlamtac *et al.* [16] described a model and algorithms to compute a path under wavelength continuity constraints. Zhu *et al.* [17] addressed the same problem in WDM mesh networks tackling traffic grooming issues. In [18], Gong and Jabbari provided an algorithm to compute an optimal path under constraints on several layers: wavelength continuity, label continuity, etc.

However, the models of these past works are not adapted to the problem of nested encapsulation and decapsulation capabilities for which a kind of stack mechanism is needed. In [19], Dijkstra *et al.* addressed this issue in the context of the ITU-T G.805 recommendations on adaptation functions. They stressed the lack of solutions on path computation. Kuipers and Dijkstra [11] demonstrated that the problem of path computation with encapsulation and decapsulation capabilities is NP-complete under bandwidth constraint. They proposed a Breadth-First Search (BFS) algorithm that explores all possible paths until finding a feasible one. In [9], [10], Lamali *et al.* demonstrated that the problem is polynomial if the bandwidth constraint is relaxed. Their approach was to model the network as a Push-Down Automaton and to use automata and language theory tools to compute a shortest feasible path, but only considering the number of hops or adaptation functions. More recently, Iqbal *et al.* [20] underlined the need of path computation algorithms in NRENs. They proposed a new matrix-based model for multi-layer networks and algorithms based on k-shortest paths and *LOOK-AHEAD* methods. However, the model deals with technologies[2] instead of protocols. Thus, the nested protocols are not transparent to the nodes. Moreover, the proposed exact algorithm is exponential and can compute only loopless feasible paths.

C. Proposed approach

Our goal is to study the path computation problem in a multi-layer context and to propose efficient algorithms to resolve it. To this end, we focus on three cases: Path computation without bandwidth constraint (by adapting the language theoretic approach of Lamali *et al.* [10]), under bandwidth constraint (by using graph transformation in order to overcome the problem complexity) and under several QoS constraints. The simulations showing the efficiency of our algorithms

[1] The algorithms presented in this paper can be applied in a single-domain or a multi-domain context. For the latter, however, a mechanism for sharing the network information (such as the topology) is needed. This can be done through a PCE for example [13].

[2] A *technology* is an exhaustive description of the protocol stack at some node, e.g., IP over Ethernet over ATM.

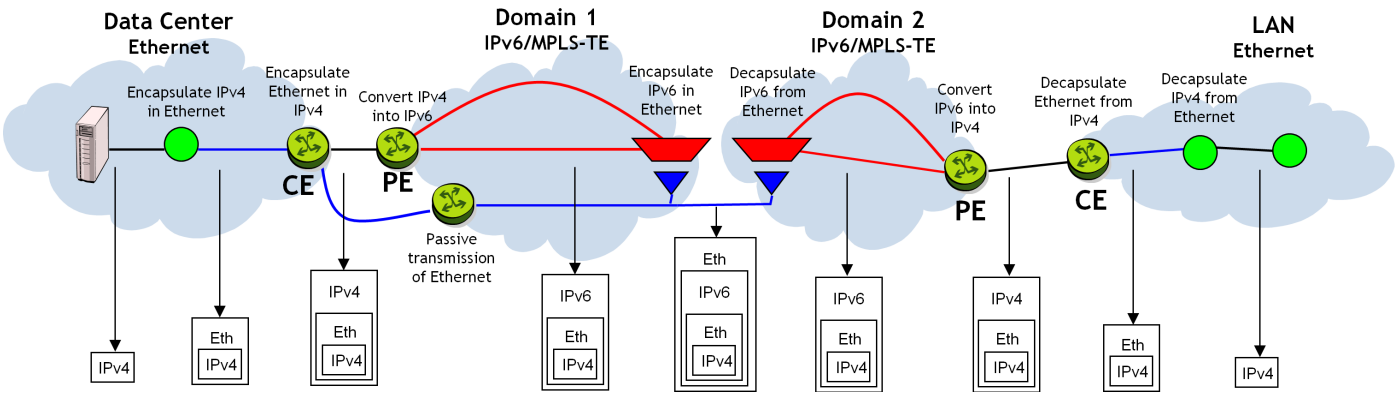

Fig. 1. Carrier-grade network comprising several domains and different layers.

follow a methodology based on the probabilistic distribution of the adaptation functions over the nodes.

III. Model and Problem Formalization

This section describes a mathematical model of multi-layer networks and formalizes the notion of path feasibility.

A. Multi-layer network model

Notation convention. In order to avoid confusion, lowercase letters denote protocols (e.g., a, b, c, x, y) or functions (e.g., f, h, ℓ). Capital letters denote nodes and links (e.g., U, V, E). Finally, calligraphic letters denote sets (e.g., $\mathcal{G}, \mathcal{V}, \mathcal{E}$).

We consider a multi-layer network as a 4-tuple $\mathcal{N} = (\mathcal{G}, \mathcal{A}, \mathcal{F}, h)$ where:

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed graph modeling the network topology. The set of nodes $\mathcal{V}$ models the routers of the network. The set of edges $\mathcal{E}$ models the physical links between the routers.
- $\mathcal{A} = \{a, b, c, \dots\}$ is the set of protocols available in the network, but not necessarily at each router.
- For each node $U \in \mathcal{V}$, $\mathcal{F}(U)$ is the set of adaptation functions available on node U. These functions are:
  - *Conversion*: A protocol a is converted into a protocol b without any change of the possible underlying protocols. This function is denoted by $(a \to b)$. E.g., Wavelength conversion on the optical layer, IPv4 to IPv6, etc.
  - *Passive function*: A protocol a is left as it is. It is a classical retransmission without any protocol change and can be considered as a special case of protocol conversion where $a = b$. Thus it is denoted by $(a \to a)$.
  - *Encapsulation*: A protocol a is encapsulated in a protocol b. It is denoted by $(a \to ab)$.
  - *Decapsulation*: A protocol a is decapsulated from a protocol b. It is denoted by $\overline{(a \to ab)}$.
- $h : \mathcal{V} \times \mathcal{F} \times \mathcal{V} \to \Re_+$ is the *weight* function. The value $h(U, f, V)$ (where $U, V \in \mathcal{V}$ and $f \in \mathcal{F}(U)$) is the cost of using the link (U, V) with the adaptation function f on U. Hence, function h allows representing any additive metric either associated only to the links or to both links and adaptation functions.

B. Path feasibility

Let (S, D) be a pair of nodes in $\mathcal{G}$ corresponding to the source and the destination of the path to be computed. We consider a path from S to D as a sequence of nodes and adaptation functions $Sf_0U_1f_1U_2f_2 \dots U_nf_nD$ where each U_i, $i = 1, \dots, n$, is a node and each f_i is an adaptation function (f_0 being fictitious). A path is *feasible* if:

1) The sequence $SU_1U_2 \dots U_nD$ is a path in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and each $f_i \in \mathcal{F}(U_i)$;
2) Each encapsulated protocol is decapsulated before reaching D according to its encapsulation order and *protocol continuity* must hold (i.e., if the sequence contains a function f_i s.t. $f_i = (a \to b)$, $a, b \in \mathcal{A}$, then $f_{i+1} = (b \to a')$ or $f_{i+1} = (b \to ba')$ or $f_{i+1} = \overline{(a' \to a'b)}$, $a' \in \mathcal{A}$).

Actually, the protocol sequences of feasible paths can be characterized as a well-parenthesized language [10].

IV. Path Computation Without Bandwidth Constraint

This section proposes a polynomial algorithm to resolve the path computation problem without bandwidth constraint and evaluates it through simulations.

A. A polynomial algorithm for path computation

Lamali *et al.* [10] proposed a language theoretic approach to compute a shortest feasible path (involving encapsulations and decapsulations of protocols) in a multi-layer network. The metric considered was the number of hops or of encapsulations in the path. The approach comprises the following steps:

1) Consider the set of protocols as an alphabet and convert the multi-layer network into a Push-Down Automaton (PDA);
2) If the considered metric is the number of encapsulations, transform the automaton in order to bypass passive transitions;
3) Convert the PDA to a Context-Free Grammar (CFG);
4) Compute the shortest word generated by the CFG. It is the protocol sequence of a shortest path;
5) Compute a shortest path from this sequence.

We made several improvements to these algorithms:

- The PDA building is modified in order to support protocol conversion by adding a new transition type;
- The PDA transitions are weighted in order to reflect the weight function. Thus, our algorithm computes the shortest path according to any additive metric (instead of just the number of hops or encapsulations);
- The PDA transformation is no longer useful thanks to the weight function: Simply put $h(U, f, V) = 1$ (where $U, V \in \mathcal{V}$ and $f \in \mathcal{F}(U)$) for all triples where f is an encapsulation, and $h(U, f, V) = 0$ for all other triples. It is also possible to set different weights to each type of encapsulation and minimize the path cost according to these weights;
- The conversion of the PDA into a CFG is adapted: As in [10], each transition from the PDA is converted into a production rule set in the CFG according to a method described in [21]. However, the transition weights are assigned to the corresponding production rules;
- Step 4 is different: Since the production rules are weighted, the goal is no longer to compute the shortest word but the word having the minimum weight derivation tree. This is done thanks to Knuth's algorithm described in [22]. This word corresponds to the protocol sequence of a shortest path to compute;
- The algorithm computing the path matching the protocol sequence is modified in order to take into account the weights.

Due to the lack of space, we cannot detail our improved algorithm. The interested reader can find it (together with its correctness proof and complexity study) in Appendix A.

Additionally to these improvements, the algorithm complexity is drastically decreased. In [10], Step 4 has a complexity of $O(|\mathcal{A}|^8 \times |\mathcal{V}|^7)$ in the worst case, which is the highest complexity in the whole process. Implementing Knuth's algorithm with Fibonacci heaps gives an $O(|\mathcal{Q}| \log |\mathcal{Q}| + |\mathcal{R}|)$ complexity, where $|\mathcal{Q}|$ is the number of nonterminals in the CFG and $|\mathcal{R}|$ is the number of production rules [23]. Since $|\mathcal{Q}| = O(|\mathcal{A}|^3 \times |\mathcal{V}|^2)$ and $|\mathcal{R}| = O(|\mathcal{A}|^5 \times |\mathcal{V}|^2 \times |\mathcal{E}|)$ (see Appendix A), the complexity of the whole process is:

$$O\left(|\mathcal{A}|^5 \times |V|^2 \times |\mathcal{E}|\right)$$

This is a significant improvement compared to the complexity $O(|\mathcal{A}|^8 \times |\mathcal{V}|^7)$ in [10].

B. Simulations

We implemented our algorithm (called PDA) and compared it to a classical BFS approach.

1) Networks used for the simulations and methodology: Large multi-layer topologies are generally not available. Some public ones as the Internet2 network [24] are not large enough to show the scaling of our algorithm. Thus we performed simulations on two topologies described in [25]:

- Topology $T1$ is a simplified version of Time Warner network. It has 41 nodes and 296 directed links.

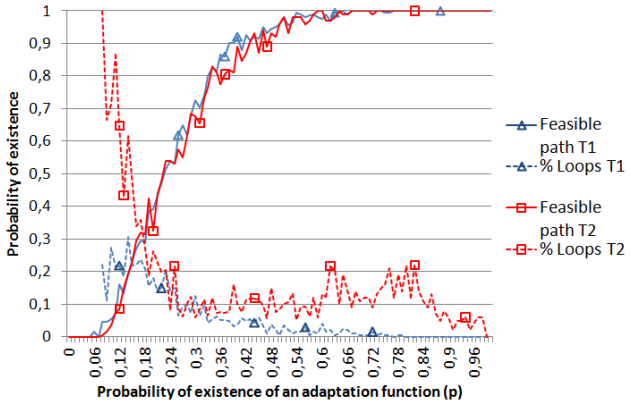

Fig. 2. Probability of existence of a feasible path (and a loop in the shortest one) according to the probability of existence of an adaptation function.

- Topology $T2$ corresponds to the network of Exodus as in 2002. It has 79 nodes and 294 directed links.

Since these topologies are not layered, the adaptation functions are randomly allocated to the nodes. For an alphabet $\mathcal{A}$, there are $3|\mathcal{A}|^2$ possible adaptation functions (for each ordered pair of protocols: a conversion, an encapsulation and a decapsulation). For each node U, each of these adaptation functions is available on U with probability p. The source and the destination nodes are the diameter extremities, which corresponds to 5 (resp. 10) hops for Topology $T1$ (resp. $T2$).

2) Phase transition in path feasibility: Depending on the network topology and the adaptation function distribution, there is not always a feasible path. It is interesting to know the probability of a feasible path existence according to probability p in order to set appropriate parameters for the simulations. In case of path existence, knowing the probability that the shortest one involves loops allows comparing the different algorithms (some of them allow loops and others do not). To compute this probability, we performed 200 runs for each value of p and counted the number of times there was a feasible path.

Figure 2 shows the evolution of feasible path existence probability according to p and the proportion of shortest paths that involve loops. Not surprisingly, the probability of feasible path existence grows according to p. On both topologies, the probability of path existence reaches 50% when $p = 0.22$ and follows a phase transition phenomenon. For example, in the interval $p \in [0.10, 0.38]$, the probability of path existence in Topology $T1$ grows from 5% to 90%. This interval is the most suitable to perform simulations. The phase transition phenomenon also holds with more than 2 protocols. The more the number of protocols is high, the more the phase transition is shifted to the left. If there are few feasible paths (for small p), the probability that the shortest ones involve loops is high. However, this probability quickly decreases. For example, for $p > 18\%$, the proportion of shortest paths involving loops is less than 20% in Topology $T1$. The trend of this proportion is not clear in $T2$, however it is less than 21% if $p > 0.22$. The phase transition phenomenon can be seen in [20]. But the results consider only loopless paths and the distribution deals with technologies rather than adaptation functions.

3) Simulation results: Our algorithm is compared to a classical BFS which explores all possible paths until reaching

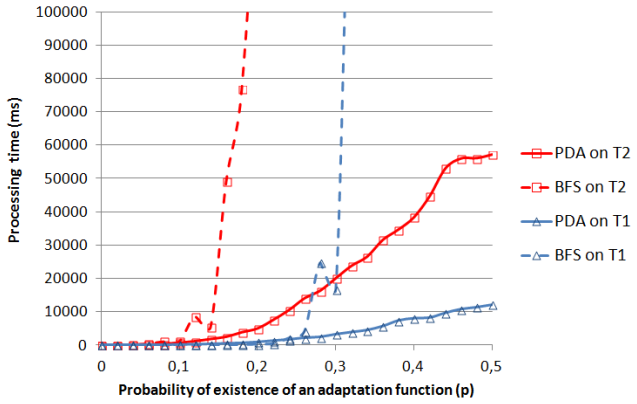

Fig. 3. Comparison of processing time of PDA algorithm and BFS on Topologies $T1$ and $T2$.

the destination. During the exploration process, all *dominated*[3] paths are deleted. BFS can be seen as a version of the algorithm in [11] where the bandwidth constraint is relaxed. The first results showed that BFS algorithm is extremely slow even for small values of p (processing time of the order of several hours). It was impossible to perform a comparison with our algorithm. Due to this tremendous running time, we fixed a maximum length to the explored paths by BFS algorithm. If a path exceeds 10 hops (resp. 14 hops) on Topology $T1$ (resp. $T2$), it is deleted and no more considered. We performed 100 runs for each value of p and averaged the processing time. Figure 3 shows the processing time of PDA algorithm and BFS algorithm on Topologies $T1$ and $T2$ according to the values of p. For small values of p (< 0.22 for $T1$ and < 0.04 for $T2$) BFS algorithm is faster than PDA. However, the processing time of BFS explodes. We cannot put it on Figure 3 because it would be unreadable. For example, the processing time of BFS algorithm on Topology $T2$ for $p = 0.24$ is more than 14 minutes, while that of PDA algorithm is 10 seconds. On Topology $T1$, for $p = 0.38$, the processing time of BFS algorithm is more than 7 minutes, while that of PDA algorithm is 7 seconds. These results show that our algorithm clearly outperforms the BFS approach.

V. Addressing Bandwidth Constraint

This section studies the complexity of path computation under bandwidth constraint and proposes heuristic solutions to resolve the problem.

A. Problem formalization

For Traffic Engineering purposes, a feasible path may be constrained by a minimal bandwidth. But it is possible that feasible paths in a multi-layer network involve loops (i.e., involving the same link several times but using different protocols). It implies that the bandwidth constraint is no longer prunable: Even if the links with not enough bandwidth are deleted by topology filtering prior to path computation, other links can have enough bandwidth if they are selected once but not if more. For example, if a link has a capacity of 10Gbps and the bandwidth constraint is 5Gbps, then this link cannot be crossed more than twice. The (optimization) problem of computing the shortest path in a multi-layer network under bandwidth constraint is defined as follows:

$$\min\ h(\mathcal{P}) = \sum_{(U,f,V)\in\mathcal{P}} h(U,f,V)$$
$$s.t. \begin{cases} \mathcal{P} \text{ is a feasible path between } S \text{ and } D \\ \min_{E\in\mathcal{P}} \dfrac{q_b(E)}{nb(E)} \geq q_b^{min} \end{cases} \quad (1)$$

where $nb(E)$ is the number of times a link E is crossed by path $\mathcal{P}$, $q_b(E)$ is the bandwidth capacity of E and q_b^{min} is the bandwidth constraint.

B. Path computation complexity under bandwidth constraint

The bandwidth constraint impacts the complexity of feasible path computation. In a single-layer network, computing a path under bandwidth constraint is trivial: It suffices to prune all the links without enough bandwidth. This is no longer possible in a multi-layer network. In fact, the decision problem is NP-complete as shown by Kuipers and Dijkstra [11]. But this proof does not work on symmetric directed graphs[4]. However, most communication networks are symmetric. We show that the decision version of the problem remains NP-complete even with two protocols and in a symmetric graph. Consider the following problem:

Problem (1'). Given a multi-layer network $\mathcal{N} = (\mathcal{G} = (\mathcal{V}, \mathcal{E}), \mathcal{A}, \mathcal{F}, h)$, a function assigning to each link $E \in \mathcal{E}$ an available bandwidth $q_b(E)$, a bandwidth constraint q_b^{min} and a pair S and D of nodes in $\mathcal{V}$. Is there a feasible path from S to D satisfying the bandwidth constraint?

Proposition 1: Problem (1') is NP-complete with two protocols even if $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a symmetric directed graph.

Proof: Clearly, the problem is in NP. Thus, we only detail the proof of NP-hardness.

First consider the problem of finding a Hamiltonian path in a symmetric directed graph between two nodes S' and D'. Call this problem SYM-HAM. SYM-HAM is NP-complete (for a detailed proof, see Appendix B).

Now we provide a polynomial reduction from SYM-HAM to Problem (1') restricted to symmetric directed graph and two protocols. Given an instance of SYM-HAM, i.e., a symmetric directed graph $\mathcal{H} = (\mathcal{V}', \mathcal{E}')$ and a pair of nodes (S', D'), we build an instance of Problem (1'), i.e., a network $\mathcal{N} = (\mathcal{G}, \mathcal{A}, \mathcal{F}, h)$ and a pair of nodes (S, D) as following:

Step 1: Splitting the nodes. For each node $U' \in \mathcal{V}'$, four nodes U_1, U_2, U_3 and U_4 are created in $\mathcal{G}$. Links (U_i, U_{i+1}) and (U_{i+1}, U_i) are created for $i = 1, \ldots, 3$. For each link $(U', V') \in \mathcal{E}'$, a link (U_1, V_1) is created in $\mathcal{G}$. This step is illustrated on Figure 4.

Step 2: Adding a tail. $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is augmented by a set $\mathcal{C} = \{C_0, \ldots, C_{n+1}\}$ of nodes ($n = |\mathcal{V}'|$), where $C_0 = S$ is the source node. There are a link (C_i, C_{i+1}) and a link (C_{i+1}, C_i) for $i = 0, \ldots, n$. Moreover, there is also a link

[3] In this context, a path dominates another one if they have the same extremities and the same protocol stack, and the first path is shorter.

[4] A symmetric directed graph is a graph where a link (U, V) exists if and only if the reverse link (V, U) exists.

Node	Adaptation functions
$C_i,\ i = 1 \dots n$	$(a \rightarrow aa)$
C_{n+1}	$(a \rightarrow ab)$
$U_1\ s.t.\ U' \in \mathcal{V}'$	$(b \rightarrow bb), (a \rightarrow ab)$
$U_2\ s.t.\ U' \in \mathcal{V}'$	$\overline{(b \rightarrow bb)}, (a \rightarrow a)$
$U_3\ s.t.\ U' \in \mathcal{V}'$	$\overline{(a \rightarrow ab)}, (a \rightarrow a)$
$U_4\ s.t.\ U' \in \mathcal{V}'$	$\overline{(a \rightarrow aa)}$
X	$\overline{(a \rightarrow ab)}$

TABLE I. THE ADAPTATION FUNCTIONS AVAILABLE ON THE NODES IN THE POLYNOMIAL REDUCTION.

from C_{n+1} to S_1 (the first node resulting from the splitting of S') and conversely. Figure 5 shows this construction. Finally, two nodes X and D are added, as well as the four links $(D_1, X), (X, D_1), (X, D)$ and (D, X) (recall that D_1 is the first node resulting from the splitting of D', see Step 1).

Step 3: Allocating the adaptation functions and available bandwidth. All the links have available bandwidth 1. The bandwidth constraint is set to 1. Thus, any feasible path must cross a link at most once. There is no possible loop. Let the set of protocols be $\mathcal{A} = \{a, b\}$. Node S emits packets of protocol a. For $i = 1 \dots, n$, each node C_i in the tail can encapsulate protocol a in itself. Node C_{n+1} can only encapsulate a in b. For each node $U' \in \mathcal{V}'$, node U_1 can encapsulate any protocol in b. Node U_2 can either decapsulate protocol b from itself or passively transmit protocol a. Node U_3 can either decapsulate protocol a from b or passively transmit protocol a. Node U_4 is able to decapsulate protocol a from itself. Finally, node X can decapsulate protocol a from b. Table I summarizes the allocation of the adaptation functions.

Now, we prove that there is a Hamiltonian path from S' to D' in $\mathcal{H}$ if and only if there is a feasible path from S to D in $\mathcal{N}$ that satisfies the bandwidth constraint. First, assuming that there is a Hamiltonian path from S' to D' in $\mathcal{H}$, we construct a feasible path $\mathcal{P}$ in $\mathcal{N}$ as follows: Starting from S in $\mathcal{N}$, $\mathcal{P}$ crosses the tail and each C_i $(i = 1 \dots n)$ adds an occurrence of protocol a in the stack of encapsulated protocols. Then crossing C_{n+1} adds b as current protocol. Thus, at the end of the tail, there are $n + 1$ encapsulated protocols a (the one emitted by S and n occurrences added in the tail) and the current protocol is b. Following the same node order as in the Hamiltonian path, replace each occurrence of a node $U' \in \mathcal{V}'$ (including S' and D') in the Hamiltonian path by the sequence:

$$U_1(b \rightarrow bb)U_2\overline{(b \rightarrow bb)}U_3\overline{(a \rightarrow ab)}U_4\overline{(a \rightarrow aa)}U_3(a \rightarrow a)\\ U_2(a \rightarrow a)U_1(a \rightarrow ab) \tag{2}$$

Thus, at node U_1 an encapsulation of protocol b occurs, at U_2 protocol b is decapsulated, at U_3 it is decapsulated again, and at U_4 protocol a is decapsulated. Path $\mathcal{P}$ then crosses passively nodes U_3 and U_2, and finally encapsulates protocol b at U_1. Thus, at each time the path crosses a Sequence (2), then one occurrence of protocol a is removed from the protocol stack. Crossing all U_4 s.t. $U' \in \mathcal{H}$ removes all encapsulated occurrences of protocol a except the first one. When the path leaves D_1 to reach node X, the current protocol is b and there is a last occurrence of protocol a which is encapsulated. Finally, node X decapsulates protocol a from protocol b and node D receives protocol a as emitted by S. Thus, $\mathcal{P}$ is a feasible path, and each link is crossed at most once, the bandwidth constraint is satisfied.

Conversely, we show that from any feasible path $\mathcal{P}$ satisfying the bandwidth constraint in $\mathcal{N}$, one can extract a Hamiltonian path between S' and D' in $\mathcal{H}$. A feasible path must cross all nodes U_4 s.t. $U' \in \mathcal{V}'$ in order to decapsulate all occurrences of protocol a encapsulated when crossing the tail. Thus, it involves Sequence (2) for all $U' \in \mathcal{V}'$. By removing the tail part and the nodes X and D from $\mathcal{P}$ and replacing each occurrence of Sequence (2) by the corresponding node U', the resulting path starts from S' and crosses all the nodes in $\mathcal{H}$ before reaching D'. The only problem is the possibility that there are other sequences than Sequence (2) in the remaining path. There are two possible cases:

- An *incomplete* Sequence (2) where U_4 is not reached (e.g., $U_1 f U_2 f' U_3 f'' U_2 f''' U_1$): This cannot happen because such a sequence forbids to reach U_4 later, and thus one encapsulated occurrence of protocol a is never decapsulated and $\mathcal{P}$ cannot be feasible. Such a sequence cannot occur after an occurrence of Sequence (2) on the same nodes because if a node U_i $(i = 2, 3)$ is reached in a Sequence (2) it cannot be reached again due to the bandwidth constraint.
- A sequence $U_1 f V_1 f' W_1$: Let $\mathcal{P}$ be a feasible path from S to D containing a sequence $U_1 f V_1 f' W_1$ (where U_1 and W_1 may be the same node). These three nodes can only encapsulate protocol a or b in protocol b. Thus, after crossing such a sequence, there are three occurrences of protocol b on the top of the protocol stack. However, in network $\mathcal{N}$, there is no possible sequence of nodes and adaptation functions able to decapsulate protocol b three consecutive times. Thus, $\mathcal{P}$ is not feasible.

Thus, if a feasible path exists, then it contains only one occurrence of Sequence (2) for each node $U' \in \mathcal{V}'$. Replacing each Sequence (2) by the corresponding node in $\mathcal{V}'$ induces a Hamiltonian path in $\mathcal{H}$. This concludes the proof. ■

Unfortunately, the previous negative result implies:

Corollary 1: Problem (1) is not approximable (unless $\mathsf{P} = \mathsf{NP}$).

Proof: Since the existence of a feasible path (independently of its cost) is NP-complete to decide, any polynomial approximation algorithm would imply $\mathsf{P} = \mathsf{NP}$. ■

On the other hand, the problem is tractable on some particular topologies:

Corollary 2: Problems (1) and (1') are polynomial if the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a Directed Acyclic Graph (DAG).

Proof: The NP-completeness of Problem (1') results from the fact that the bandwidth constraint is not prunable when feasible paths involve loops. In a DAG, every link is involved at most once in a feasible path due to the absence of cycles. Thus the bandwidth constraint is prunable and the problem can be resolved using the method described in Section IV. ■

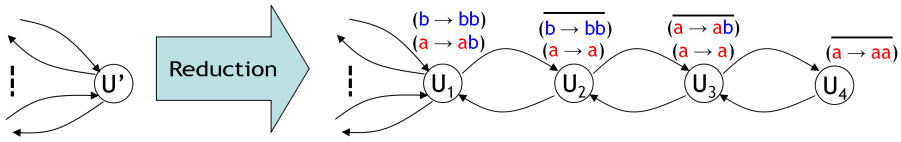

Fig. 4. Reduction from SYM-HAM to feasible path under bandwidth constraint (node splitting).

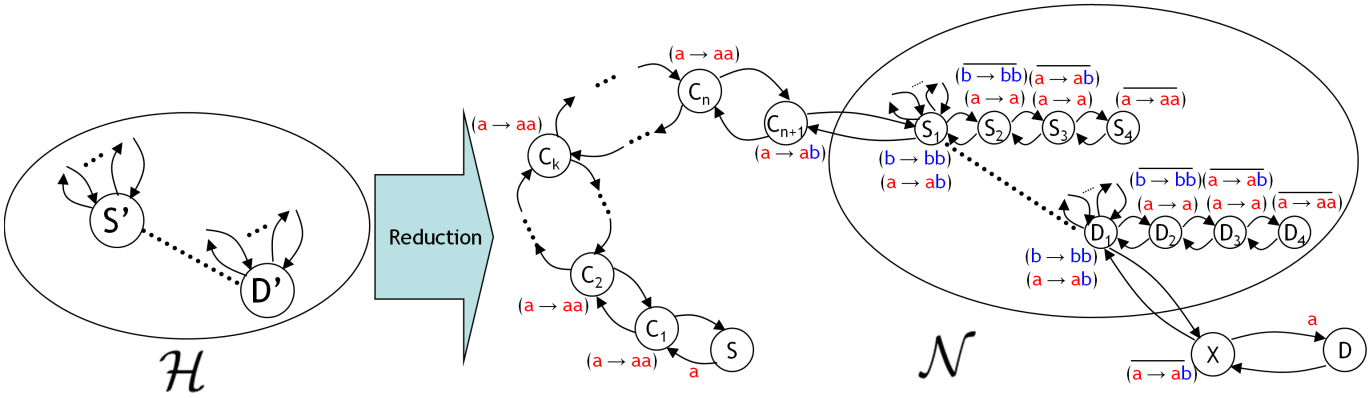

Fig. 5. Reduction from SYM-HAM to feasible path under bandwidth constraint (graph transformation).

C. DAG Heuristic

As seen in Section IV-B2, shortest feasible paths involving loops are infrequent (for $p > 20\%$). Combining this fact with Corollary 2 suggests a heuristic to compute feasible path under bandwidth constraint: Convert the network into a DAG and perform the PDA algorithm to compute a shortest feasible path.

DAG Conversion. The network is converted into a DAG in the following way:

1) Set the number 0 to node S and $|\mathcal{V}| - 1$ to node D (recall that S and D are the extremities of the graph diameter);
2) Perform a BFS algorithm starting from node S and number the nodes in the visit order. The nodes at the same distance from S are visited randomly, thus performing several times this heuristic does not always give the same node numbering and the same DAG;
3) Delete all the links that start at a node and end at a node with a smaller number.

The DAG heuristic is as follows:

1) Convert the network into a DAG;
2) Prune the links without enough bandwidth;
3) Perform the PDA algorithm of Section IV.

D. Simulations

We study the efficiency of the DAG heuristic (called DAG-PDA) and compare it with the algorithm of Kuipers and Dijkstra [11]. The latter is an exact (and thus exponential) algorithm that performs a BFS and explores all the paths that are not dominated and that satisfy the bandwidth constraint. As in Section IV-B3, the BFS algorithm is slow. Thus, we also compare our algorithm with DAG-BFS algorithm, where the network is converted into a DAG before performing the BFS.

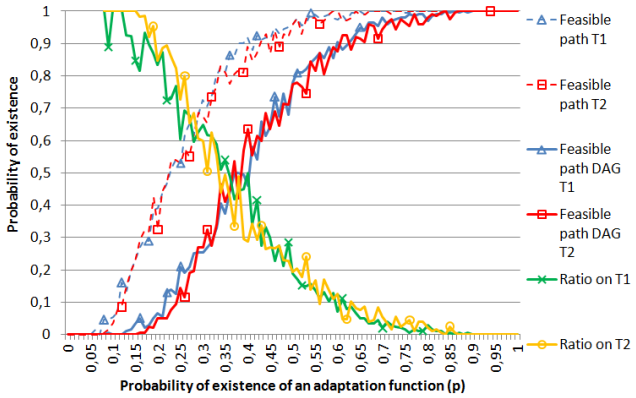

Fig. 6. Probability of feasible path existence before and after DAG conversion on Topologies $T1$ and $T2$.

The simulation conditions (parameters, topology, number of runs, etc.) are the same as in Section IV-B3. The bandwidth capacity of the links is randomly and uniformly selected in the set $\{1, 2, \ldots, 10\}$. The bandwidth constraint is set to 2.

1) Comparison of the feasibility ratio: Converting the network topology into a DAG deletes some feasible paths in the original network. We measure how much feasible paths are lost by comparing the probability of feasible path existence before and after the DAG conversion according to the probability of existence of adaptation functions (p). Figure 6 shows that the probability of feasible path existence is shifted to the right after the DAG conversion. The ratio $\frac{\text{Probability of feasible path existence in T}i}{\text{Probability of feasible path existence in DAG T}i}$ $(i = 1, 2)$ is clearly decreasing and is less than 50% if $p > 0.34$, which is important but balanced by the improvement of the processing time.

2) Comparison of the processing time: Figure 7 shows the processing time of DAG-PDA, DAG-BFS and BFS algorithms on both topologies according to the probability of existence of

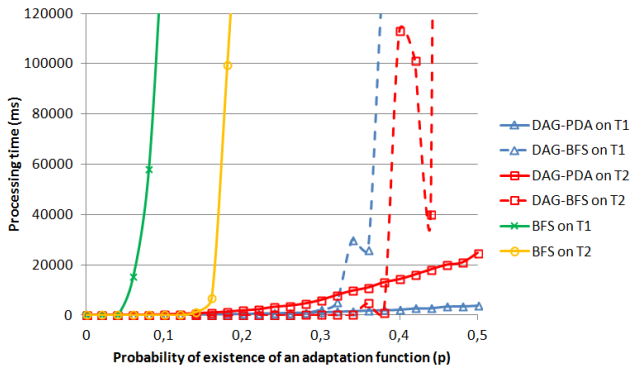

Fig. 7. Comparison of the processing time of DAG-PDA, DAG-BFS and BFS algorithms on Topologies $T1$ and $T2$.

an adaptation function. BFS algorithm is slow even for small values of p. For $p < 0.3$ (resp. 0.4) on Topology $T1$ (resp. $T2$), DAG-BFS is faster than DAG-PDA. Beyond these values, the processing time of DAG-BFS explodes. For example, for $p = 0.5$, the processing time of DAG-BFS is more than 35 minutes on Topology $T1$ and more than 53 minutes on Topology $T2$, while that of DAG-PDA is 3.8 seconds on $T1$ and 24 seconds on $T2$. These results show that the DAG-PDA algorithm is clearly faster when there is a significant number of adaptation functions, but the exponential DAG-BFS algorithm is faster if there are few of them (for small values of p).

VI. Path computation under QoS constraints

A. Multi-constrained feasible path

Let $\mathcal{N}$ be a multi-layer network. Each link $E = (U, V)$ is associated to a set of m additive QoS metrics $q(E) = (q_1(E), \ldots, q_m(E))$ in addition to its available bandwidth $q_b(E)$. These additive metrics can be the delay, logarithm of the packet-loss, etc.

Let $q_b^{\min}$ be the bandwidth constraint and $q^{max} = (q_1^{\max}, q_2^{\max} \ldots, q_m^{\max})$ be a vector of QoS constraints, the problem of computing a shortest feasible path under these constraints is formalized as:

$$\min\ h(\mathcal{P}) = \sum_{(U,f,V)\in\mathcal{P}} h(U,f,V)$$
$$s.t. \begin{cases} \mathcal{P} \text{ is a feasible path between } S \text{ and } D \\ \min_{E\in\mathcal{P}} \dfrac{q_b(E)}{nb(E)} \geq q_b^{\min} \\ \sum_{E\in\mathcal{P}} \left(q_i(E) \times nb(E)\right) \leq q_i^{\max},\ i = 1 \ldots m \end{cases} \quad (3)$$

B. Complexity of multi-constrained feasible path computation

The problem of QoS multi-constrained path computation (on a single layer) is well studied. It is well-known that the decision version associated to this problem is NP-complete, even with 2 additive and/or multiplicative constraints [26]. Van Mieghem and Kuipers [12] gave an exponential time algorithm but showed that the instances that really require an exponential computation time are infrequent. The classical multi-constrained path problem is a particular case of Problem 3, corresponding to the case where there is only one protocol and passive transitions. Thus the decision version associated to Problem 3 is also NP-complete.

C. ML-SAMCRA

As computing a multi-layer path under QoS constraints is NP-complete, any algorithm able to solve this problem is exponential in the worst case (unless P = NP). We propose to adapt the Self-Adaptive Multiple Constraints Routing Algorithm (SAMCRA) to the multi-layer context in order to compute a shortest feasible path under QoS constraints.

SAMCRA is an exact QoS routing algorithm proposed by Van Mieghem and Kuipers [12]. It computes the shortest path under several (additive) QoS constraints but it ignores the feasibility constraint as defined in our paper. SAMCRA has an exponential worst case complexity, but it exhibits a reasonable processing time in practice.

1) The main concepts of SAMCRA: The idea of SAMCRA is to maintain a path list from the source node S to all other nodes until reaching the destination node D. It progressively removes the paths that do not comply with the QoS constraints. The main concepts of SAMCRA are:

- *Non-linear path length:* In SAMCRA, the path length is defined as a non-linear function of the QoS parameters of each link. It reduces the solution space to scan but the algorithm can apply with any metric. Hence, it is not a strict requirement.
- *The k-shortest path algorithm:* The k-shortest path algorithm maintains the list of the paths that are not (yet) removed from the path list.
- *Non-dominance:* A multi-constrained path $\mathcal{P}$ *dominates* another path $\mathcal{P}'$ if $\forall i, \sum_{E\in\mathcal{P}} q_i(E) \leq \sum_{E\in\mathcal{P}'} q_i(E)$ (i.e., if $\mathcal{P}$ is better than $\mathcal{P}'$ for each QoS parameter). A path $\mathcal{P}$ is non-dominated if there is no path which dominates it. The concept of non-dominance induces a partial order over the paths. It avoids the exploration of several paths thus substantially reducing the average complexity of SAMCRA.

The path length definition is not impacted by the multi-layer context and using a linear path length function is not forbidden. The k-shortest path algorithm is not impacted either. However, the concept of dominance must be redefined to meet the path feasibility constraint and to take into account possible loops.

2) Extension of the non-dominance definition: A multi-layer path is characterized by its nodes but also by its protocol stack at the destination node. Thus in the algorithm path list, each path should be stored with its protocol stack at its final node. A multi-layer path can involve the same link several times. Before checking if this path complies with some QoS parameters, the parameters of each link should be multiplied by the number of times this link is involved in the path. The bandwidth constraint is not prunable in multi-layer context, the new non-dominance definition should take it into account.

A path $\mathcal{P}$ dominates a path $\mathcal{P}'$ if the four following conditions are satisfied:

- $\min_{E\in\mathcal{P}} \frac{q_b(E)}{nb_{\mathcal{P}}(E)} \geq \min_{E\in\mathcal{P}'} \frac{q_b(E)}{nb_{\mathcal{P}'}(E)}$

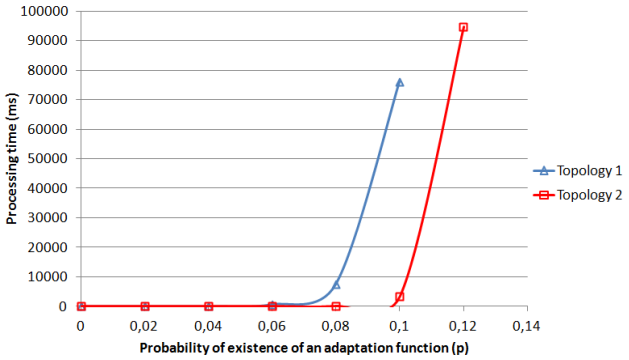

Fig. 8. Processing time of ML-SAMCRA Topologies $T1$ and $T2$.

- $\sum_{E\in\mathcal{P}} q_i(E) \times nb_{\mathcal{P}}(E) \leq \sum_{E\in\mathcal{P}'} q_i(E) \times nb_{\mathcal{P}'}(E)$ $\forall i = 1, \ldots, m$
- $\mathcal{P}$ and $\mathcal{P}'$ have the same final node;
- $\mathcal{P}$ and $\mathcal{P}'$ have the same protocol stack at this node.

Where $nb_{\mathcal{P}}(E)$ (resp. $nb_{\mathcal{P}'}(E)$) is the number of times the link E is involved in path $\mathcal{P}$ (resp. $\mathcal{P}'$). According to this new definition of non-dominance, ML-SAMCRA explores all the possible paths until reaching the destination node with satisfactory QoS parameters. Along the exploration, it removes all paths that are dominated or not feasible.

D. Simulations

We know study the efficiency of ML-SAMCRA through simulations and check if it is as scalable in a multi-layer context as SAMCRA in a single layer context. Figure 8 shows the processing time of ML-SAMCRA on Topologies $T1$ and $T2$ according to the probability of existence of an adaptation function (p). The results show that for $p > 0.08$ (resp. 0.10) on Topology $T1$ (resp. $T2$) the processing time explodes (more than 1 minutes). Clearly, ML-SAMCRA does not scale above these values. There are two reasons:

1) The paths are less comparable in term of the new non-dominance definition: They should have the same protocol stack. As there are less dominated paths, the algorithm complexity increases;
2) Taking into account loops increases the number and the length of the paths, which also increases the algorithm complexity.

So, path computation under QoS constraints in multi-layer networks is more complex than in single layer networks. Thus, exact algorithms are suitable only for small instances.

VII. Conclusion

Most of carrier-grade networks manage their different layers thanks to separate control planes. Designing a unified control plane would allow the network resources to be optimized and the operational management costs to be reduced. One key problem to address is path computation taking into account the protocol heterogeneity and the multi-layer context dealing with encapsulation, conversion and decapsulation of protocols. This paper tackles this issue by partitioning it into three cases: Path computation without bandwidth constraint, under bandwidth constraint and under additive QoS constraints. For the first case, we widely generalized polynomial algorithms in the state of the art and decreased their complexity. Through simulations, we showed that they outperform previous approach in the literature. For the second case, we obtained several time complexity results and proposed efficient heuristics. Finally, we designed the first algorithm to resolve the third case. In future works, we plan to design heuristics to deal with additive QoS metrics, as the exact approach seems to be not scalable. The problem of efficient generation of random topologies being widely open, it would be interesting to analytically study the phase transition phenomenon in order to generate topologies having a suitable number of feasible paths.

References

[1] L. Martini, E. Rosen, N. El-Aawar, and G. Heron, "RFC4448 - Encapsulation Methods for Transport of Ethernet over MPLS Networks," 2008.

[2] F. Baker, X. Li, C. Bao, and K. Yin, "RFC6144 - Framework for IPv4/IPv6 Translation," 2011.

[3] S. Bryant and P. Pate, "RFC3985 - Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture," 2005.

[4] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, "Packet and circuit network convergence with openflow," in *Optical Fiber Communication Conference*. Optical Society of America, 2010.

[5] L. Liu, D. Zhang, T. Tsuritani, R. Vilalta, R. Casellas, L. Hong, I. Morita, H. Guo, J. Wu, R. Martínez *et al.*, "Field trial of an openflow-based unified control plane for multilayer multigranularity optical switching networks," *Lightwave Technology, Journal of*, vol. 31, no. 4, pp. 506–514, 2013.

[6] S. Agarwal, M. S. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*, 2013, pp. 2211–2219.

[7] S. Li, Y. Shao, S. Ma, N. Xue, S. Li, D. Hu, and Z. Zhu, "Flexible traffic engineering: When openflow meets multi-protocol ip-forwarding," *IEEE Communications Letters*, vol. 18, no. 10, pp. 1699–1702, 2014.

[8] E. W. Dijkstra, "A note on two problems in connexion with graphs." *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[9] M. L. Lamali, H. Pouyllau, and D. Barth, "Path computation in multi-layer multi-domain networks," in *Networking (1)*, 2012, pp. 421–433.

[10] ——, "Path computation in multi-layer multi-domain networks: A language theoretic approach," *Computer Communications*, vol. 36, no. 5, pp. 589–599, 2013.

[11] F. A. Kuipers and F. Dijkstra, "Path selection in multi-layer networks," *Computer Communications*, 2009.

[12] P. V. Mieghem and F. A. Kuipers, "Concepts of exact QoS routing algorithms," *IEEE/ACM Trans. Netw.*, vol. 12, no. 5, pp. 851–864, 2004.

[13] A. Farrel, J. Vasseur, and J. Ash, "RFC4655 - A Path Computation Element (PCE)-Based Architecture," 2006.

[14] M. Bocci and S. Bryant, "RFC5659 - An Architecture for Multi-Segment Pseudowire Emulation Edge-to-Edge," 2009.

[15] F. Dijkstra, J. V. der Ham, P. Grosso, and C. de Laat, "A path finding implementation for multi-layer networks," *Future Generation Comp. Syst.*, vol. 25, no. 2, pp. 142–146, 2009.

[16] I. Chlamtac, A. Faragó, and T. Zhang, "Lightpath (Wavelength) Routing in Large WDM Networks," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 5, pp. 909–913, 1996.

[17] H. Zhu, H. Zang, K. Zhu, and B. Mukherjee, "A novel generic graph model for traffic grooming in heterogeneous WDM mesh networks," *IEEE/ACM Trans. Netw.*, vol. 11, no. 2, pp. 285–299, 2003.

[18] S. Gong and B. Jabbari, "Optimal and Efficient End-to-End Path Computation in Multi-Layer Networks," in *ICC*, 2008, pp. 5767–5771.

[19] F. Dijkstra, B. Andree, K. Koymans, J. van der Ham, P. Grosso, and C. de Laat, "A multi-layer network model based on ITU-T G.805," *Comput. Netw.*, 2008.

[20] F. Iqbal, J. van der Ham, and F. Kuipers, "Technology-aware multi-domain multi-layer routing," *Computer Communications*, vol. 62, pp. 85–96, 2015.

[21] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Introduction to automata theory, languages, and computation." Addison-Wesley, 2006.

[22] D. E. Knuth, "A Generalization of Dijkstra's Algorithm," *Inf. Process. Lett.*, vol. 6, no. 1, pp. 1–5, 1977.

[23] M. L. Fredman and R. E. Tarjan, "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, Jul. 1987.

[24] R. Summerhill, "The new internet2 network," in *6th GLIF Meeting*, 2006, available at http://www.internet2.edu/products-services/advanced-networking.

[25] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 231–236. [Online]. Available: http://research.cs.washington.edu/networking/rocketfuel/

[26] Z. Wang and J. Crowcroft, "Quality-of-Service Routing for Supporting Multimedia Applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[27] I. Petre and A. Salomaa, "Algebraic Systems and Pushdown Automata," in *Handbook of Weighted Automata*, M. Droste, W. Kuich, and H. Vogler, Eds. Springer Publishing Company, Incorporated., 2009, ch. 7, pp. 257–290.

APPENDIX A
POLYNOMIAL ALGORITHMS FOR PATH COMPUTATION IN MULTI-LAYER NETWORKS

The sequence of protocols involved in a feasible multi-layer path is a context-free language. Based on this fact, Lamali *et al.* [10] used automata and language theory tools to compute the shortest feasible path in hops or in adaptation functions. We improve their algorithm in order to compute the shortest path according to any additive metric. We also substantially reduce its complexity.

A. Theoretical language aspects of multi-layer paths

Considering a path $\mathcal{P} = Sf_0U_1f_1U_2f_2 \dots U_nf_nD$, let $H_\mathcal{P} = f_1 \dots f_n$ denotes the sequence of adaptation functions along $\mathcal{P}$. Let define as an alphabet the set $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$ and the set $\underline{\mathcal{A}} = \{\underline{a} \mid a \in \mathcal{A}\}$.

$\mathcal{T}_\mathcal{P} = x_1 \dots x_{n+1}$ is the sequence of protocols used along path $\mathcal{P}$. It is called the *trace* of $\mathcal{P}$. For each x_i:

- $x_i = a$ and $x_{i+1} = b$, $\overline{b}$ or $\underline{b}$ means that U_i converts protocol a into b ($a, b, \overline{b}, \underline{b} \in \mathcal{A} \cup \overline{\mathcal{A}} \cup \underline{\mathcal{A}}$)
- $x_i = \overline{a}$ and $x_{i+1} = b$, $\overline{b}$ or $\underline{b}$ means that U_i encapsulates protocol a in b
- $x_i = \underline{a}$ and $x_{i+1} = b$, $\overline{b}$ or $\underline{b}$ means that U_i decapsulates protocol b from a.

Here, some additional definitions are needed. The set of protocol conversions available on node U is denoted by $\mathcal{CO}(U)$. The set of encapsulations available on node U is denoted by $\mathcal{EN}(U)$ and the set of decapsulations available on node U is denoted by $\mathcal{CO}(U)$.

$In(U)$ (resp. $Out(U)$) is the set of protocols that node U can receive (resp. send). More formally:

- If $(a \to b) \in \mathcal{CO}(U)$ then $a \in In(U)$ and $b \in Out(U)$
- If $(a \to ab) \in \mathcal{EN}(U)$ then $a \in In(U)$ and $b \in Out(U)$
- If $\overline{(a \to ab)} \in \mathcal{DE}(U)$ then $b \in In(U)$ and $a \in Out(U)$

Obviously, several paths can have the same trace. The set of traces of the feasible paths in a network $\mathcal{N}$ is a context-free language but it is not regular as the encapsulations and decapsulations should be balanced. In fact, it is a well-parenthesized language, and thus requires a stack to be recognized and computed. PDAs are the classical tools to recognize context-free languages. Using weighted PDAs allows associating a weight to each link and adaptation function in order to model any additive metric.

B. Definition of WPDA

A weighted PDA (WPDA) is a 8-tuple $\mathcal{PDA} = (\mathcal{S}, \Sigma, \Gamma, \delta, Q_0, Z_0, \mathcal{S}_F, \omega)$ where $\mathcal{S}$ is the set of states, Σ is the input alphabet, Γ is the stack symbol set (i.e., stack alphabet) not necessarily different from Σ, δ is the set of transitions, Q_0 is the initial state, Z_0 is the initial stack symbol, $\mathcal{S}_F$ is the set of final (accepting) states and ω is a weight function over the set of transitions (i.e., $\omega : \ \delta \to \Re_+$).

A transition $t \in \delta$ is denoted by $t = (Q_i, \langle x, \alpha, \beta \rangle, Q_j)$, where Q_i is the state of $\mathcal{PDA}$ before the transition, Q_j is the state after the transition, $x \in \Sigma \cup \{\epsilon\}$ is an input symbol, $\alpha \in \Gamma$ is the symbol which is popped from the top of the stack, and $\beta \in \Gamma^*$ is the symbol sequence which is pushed on the top of the stack.

Remark. WPDAs are more often formalized as 6-tuples $\mathcal{PDA} = (\mathcal{S}, \Gamma, \mathcal{M}, q_0, Z_0, \mathcal{S}_F)$ where $\mathcal{M}$, called the *Push-Down transition matrix*, is a matrix over a semiring of formal power series. The input alphabet Σ, the transitions set δ and the weight function ω are expressed by a single entity $\mathcal{M} \in ((\mathcal{R}\langle\langle\Sigma^*\rangle\rangle)^{\mathcal{S}\times\mathcal{S}})^{\Gamma^*\times\Gamma^*}$, where $\mathcal{R}\langle\langle\Sigma^*\rangle\rangle$ denotes the collection of all power series from Σ^* into a semiring $\mathcal{R}$. For simplification purposes, we opted for defining a WPDA as a classical PDA with a weight function over the transition set. For the theoretical foundations of WPDAs, the interested reader can refer to [27].

C. From the graph to the WPDA

Algorithm 1 converts a multi-layer network $\mathcal{N}$ with a specified pair of nodes (S, D) into a WPDA $\mathcal{PDA} = (\mathcal{S}, \Sigma, \Gamma, \delta, Q_0, Z_0, \mathcal{S}_F = Q_F, \omega)$.

Computing a feasible path requires to know the current protocol and the last encapsulated one (in order to know if a decapsulation can be performed). Thus Algorithm 1 creates a state U_x for each node U and each protocol $x \in In(U)$. Being in state U_x indicates that the current protocol is x. The last encapsulated protocol is the one on the top of the stack.

The conversion functions $(x \to y)$ between node U and node V are turned into transitions $(U_x, \langle x, \alpha, \alpha \rangle, V_y)$ in the WPDA. The encapsulation functions $(x \to xy)$ are converted into pushes of x on the stack $(U_x, \langle \overline{x}, \alpha, x\alpha \rangle, V_y)$ and the decapsulation functions into pops of x from the stack $(U_y, \langle \underline{y}, x, \emptyset \rangle, V_x)$.

Algorithm 1 Convert a network into a WPDA

Input: A network $\mathcal{N} = (\mathcal{G} = (\mathcal{V}, \mathcal{E}), \mathcal{A}, \mathcal{F}, h)$, a source S and a destination D
Output: A WPDA $\mathcal{PDA} = (\mathcal{S}, \Sigma, \Gamma, \delta, Q_0, Z_0, \{Q_F\}, \omega)$
$\Sigma \leftarrow \mathcal{A} \cup \overline{\mathcal{A}} \cup \underline{\mathcal{A}}$; $\Gamma \leftarrow \mathcal{A} \cup \{Z_0\}$
Create $\mathcal{S}$ (the set of states of the WPDA) according to Procedure 1
Build the transition set δ according to:
- Procedure 2 for the set of conversion functions
- Procedure 3 for the set of encapsulation functions
- Procedure 4 for the set of decapsulation functions

Procedure 1 Create $\mathcal{S}$, the set of states of the WPDA

Create a single state Q_0 corresponding to node S
Create a fictitious final state Q_F
For each node $U \neq S$ in $\mathcal{V}$, for each protocol $x \in In(U)$, create a state U_x
for each state U_x s.t. $(S, U) \in \mathcal{E}$, for each $x \in Out(S)$ **do**
  Create the transition $t = (Q_0, \langle \epsilon, Z_0, Z_0 \rangle, U_x)$
  $\omega(t) \leftarrow 0$
end for
for each $x \in In(D)$ **do**
  Create the transition $t = (D_x, \langle x, Z_0, \emptyset \rangle, Q_F)$
  $\omega(t) \leftarrow 0$
end for

Procedure 2 Transform the conversions

for each link $(U, V) \in \mathcal{E}$ s.t. $U \neq S$ **do**
  for each $(x \rightarrow y) \in \mathcal{CO}(U)$ **do**
    if $y \in In(V)$ **then**
      for all $\alpha \in \Gamma$ **do**
        Create the transition $t = (U_x, \langle x, \alpha, \alpha \rangle, V_y)$
        $\omega(t) \leftarrow h(U, (x \rightarrow y), V)$
      end for
    end if
  end for
end for

Procedure 3 Transform the encapsulations

for each link $(U, V) \in \mathcal{E}$ s.t. $U \neq S$ **do**
  for each $(x \rightarrow xy) \in \mathcal{EN}(U)$ **do**
    if $y \in In(V)$ **then**
      for all $\alpha \in \Gamma$ **do**
        Create the transition $t = (U_x, \langle \overline{x}, \alpha, x\alpha \rangle, V_y)$
        $\omega(t) \leftarrow h(U, (x \rightarrow xy), V)$
      end for
    end if
  end for
end for

Procedure 4 Transform the decapsulations

for each link $(U, V) \in \mathcal{E}$ s.t. $U \neq S$ **do**
  for each $\overline{(x \rightarrow xy)} \in \mathcal{DE}(U)$ **do**
    if $x \in In(V)$ **then**
      Create the transition $t = (U_y, \langle \underline{y}, x, \emptyset \rangle, V_x)$
      $\omega(t) \leftarrow h(U, \overline{(x \rightarrow xy)}, V)$
    end if
  end for
end for

Complexity of Algorithm 1. The complexity of Algorithm 1 is in $O(|\mathcal{A}|^3 \times |\mathcal{E}|)$. The number of states created by Procedure 1 is at worst $2+|\mathcal{A}|\times(|\mathcal{V}|-1)$, and the complexity of Procedure 1 is in $O(|\mathcal{A}| \times |\mathcal{V}|)$. The number of transitions created by Procedure 2 and by Procedure 3 is in $O(|\mathcal{A}|^3 \times |\mathcal{E}|)$, which is also an upper bound for their complexity. The complexity of Procedure 4 is bounded by $O(|\mathcal{A}|^2 \times |\mathcal{E}|)$.

Proposition 2: A path $\mathcal{P}$ in a network $\mathcal{N}$ is feasible if and only if its trace $\mathcal{T}_\mathcal{P}$ is accepted by $\mathcal{PDA}$.

Proof: Consider a feasible path $\mathcal{P} = Sf_0U_1f_1U_2f_2 \dots U_nf_nD$. By construction, for each 3-tuple (U_i, f_i, U_{i+1}) there is a transition:

- $t = ((U_i)_x, \langle x, \alpha, \alpha \rangle, (U_{i+1})_y)$ if $f_i = (x \rightarrow y)$
- $t = ((U_i)_x, \langle \overline{x}, \alpha, x\alpha \rangle, (U_{i+1})_y)$ if $f_i = (x \rightarrow xy)$
- $t = ((U_i)_x, \langle \underline{x}, y, \emptyset \rangle, (U_{i+1})_y)$ if $f_i = \overline{(y \rightarrow yx)}$

This transition recognizes the i-th letter of the trace $\mathcal{T}_\mathcal{P}$. It is easy to show by induction that $\mathcal{T}_\mathcal{P}$ is accepted by the automaton.

Conversely, if a trace $\mathcal{T}_\mathcal{P}$ is accepted by a transition sequence $t_1 \dots t_n$ where each $t_i = ((U_i)_x, \langle x, \alpha, \beta \rangle, (U_{i+1})_y)$. Then there is an adaptation function:

- $f_i = (x \rightarrow y) \in \mathcal{CO}(U_i)$ if $t_i = ((U_i)_x, \langle x, \alpha, \alpha \rangle, (U_{i+1})_y)$
- $f_i = (x \rightarrow xy) \in \mathcal{EN}(U_i)$ if $t_i = ((U_i)_x, \langle \overline{x}, \alpha, x\alpha \rangle, (U_{i+1})_y)$
- $f_i = \overline{(y \rightarrow yx)} \in \mathcal{DE}(U_i)$ if $t_i = ((U_i)_x, \langle \underline{x}, y, \emptyset \rangle, (U_{i+1})_y)$

Thus the path $Sf_0U_1f_1U_2f_2 \dots U_nf_nd$ is feasible in $\mathcal{N}$. ■

The weight of a path $\mathcal{P} = Sf_0U_1f_1U_2f_2 \dots U_nf_nD$ is defined as the sum of the weights of its links and its adaptation functions. It is denoted by $h(\mathcal{P}) \stackrel{def}{=} \sum_{i=1}^{n} h(U_i, f_i, U_{i+1})$ with $U_{n+1} = D$.

We define the weight of a transition sequence as the sum of the weights of each transition (i.e., $\omega(\{t_1, t_2, \dots, t_n\}) = \sum_{i=1}^{n} \omega(t_i)$). The weigh of a word w, denoted by $\omega(w)$, is the weight of the transitions that accept w in $\mathcal{PDA}$. But as $\mathcal{PDA}$ may be nondeterministic, it is possible that several transition sequences accept the same word. Thus we consider only the sequence of transitions of minimum weight that accepts w. More formally, $\omega(w) = \min_{t_1,\dots,t_n \in \delta} \omega(\{t_1, \dots t_n\})$ s.t. $\{t_1, \dots t_n\}$ accepts w.

Lemma 1: If $\mathcal{PDA}$ accepts the trace $\mathcal{T}_\mathcal{P}$ of a path $\mathcal{P}$, then $\omega(\mathcal{T}_\mathcal{P}) = h(\mathcal{P}^*)$, where $\mathcal{P}^*$ is the path of minimum weight having $\mathcal{T}_\mathcal{P}$ as trace.

Proof: By definition, $\omega(\mathcal{T}_\mathcal{P}) = \omega(\{t_1, \dots, t_n\})$, where $\{t_1, \dots, t_n\}$ is the transition sequence with minimal weight which accepts $\mathcal{T}_\mathcal{P}$. From $\{t_1, \dots, t_n\}$, it is possible to build the path $\mathcal{P}^*$ (inversing the conversion in Algorithm 1) such that $\mathcal{T}_{\mathcal{P}^*} = \mathcal{T}_\mathcal{P}$ and $h(\mathcal{P}^*) = \omega(\mathcal{T}_\mathcal{P})$.

Suppose that $\exists \mathcal{P}'$ s.t. $\mathcal{T}_{\mathcal{P}'} = \mathcal{T}_\mathcal{P}$ and $h(\mathcal{P}') < \omega(\mathcal{T}_\mathcal{P})$, then it is possible to build from $\mathcal{P}'$ a sequence of transitions that

corresponds to the links and adaptation functions involved in $\mathcal{P}'$ (as in Algorithm 1). Let this sequence be $t'_1 \dots, t'_n$. The weight of each transition t'_i corresponds to the weight of an adaptation function associated to a link in $\mathcal{P}'$. The weight of $t'_1 \dots, t'_n$ is then less than $\omega(\mathcal{T}_\mathcal{P})$, and by Proposition 2, this sequence accepts $\mathcal{T}_\mathcal{P}$. This is inconsistent with the definition of $\omega(\mathcal{T}_\mathcal{P})$. ■

D. Computing the minimal weight trace

In order to compute the minimum weight trace and its corresponding path, $\mathcal{PDA}$ is converted into a weighted Context-Free Grammar (WCFG).

1) From the WPDA to a WCFG: A WCFG is a CFG with a weight function over the set of production rules. The conversion of a PDA into a CFG is well-known. The conversion of a WPDA into a WCFG is done in the same way, in addition the weight of each transition is assigned to the corresponding production rules (called rules in Algorithm 2) in the WCFG.

Algorithm 2 is an adaptation of the general method described in [21]. It converts $\mathcal{PDA}$ into a WCFG $\mathcal{CFG} = (\mathcal{Q}, \Sigma, [Q_0], \mathcal{R}, \pi)$ where:

- $\mathcal{Q}$ is the set of nonterminals,
- Σ is the alphabet or set of terminals (the same as the WPDA input alphabet),
- $[Q_0]$ is the initial symbol (initial nonterminal, or axiom),
- $\mathcal{R}$ is the set of production rules,
- $\pi : \mathcal{R} \to \Re_+$ is the weight function over the set of production rules.

Complexity of Algorithm 2. The number of nonterminals is bounded by $O(|\Gamma| \times |\mathcal{S}|^2)$ (as each nonterminal is in the form $[Q_i x Q_j]$ with $Q_i, Q_j \in \mathcal{S}$ and $x \in \Gamma$. The number of production rules is bounded by $O(|\delta| \times |\mathcal{S}|^2)$. Thus the worst case complexity of Algorithm 2 is bounded by $O(|\delta| \times |\mathcal{S}|^2)$. This corresponds to $O\left(|\mathcal{A}|^5 \times |\mathcal{V}|^2 \times |\mathcal{E}|\right)$.

2) The minimum weight derivation tree: Generating the minimum weight trace (and then the minimum weight path) requires to build its derivation tree. Let $[X]$ be a nonterminal, we define $\ell([X])$ as the sum of the weights of the productions needed for, starting from $[X]$, deriving a word in Σ^*. Thus $\ell([Q_0])$ is the weight of the minimum weight trace.

The function is $\ell : \{\mathcal{Q} \cup \Sigma \cup \{\epsilon\}\}^* \to \mathbb{N} \cup \{\infty\}$ s.t.:

- if $w = \epsilon$ or $w \in \Sigma$ then $\ell(w) = 0$,
- if $w = \alpha_1 \dots \alpha_n$ (with $\alpha_i \in \{\mathcal{Q} \cup \Sigma \cup \{\epsilon\}\}$) then $\ell(w) = \sum_{i=1}^{n} \ell(\alpha_i)$.
- Let $r_1 = [X] \to \gamma_1, r_2 = [X] \to \gamma_2, \dots, r_k = [X] \to \gamma_k$ be the set of production rules having $[X]$ as left part. Then $\ell([X]) = \min\{\pi(r_1) + \ell(\gamma_1), \dots, \pi(r_k) + \ell(\gamma_k)\}$

Knuth's algorithm [22] can be adapted to compute the minimum weight derivation tree of a grammar. This corresponds to the weight of $\mathcal{T}_\mathcal{P}$, where $\mathcal{P}$ is the shortest path to compute.

Algorithm 2 Convert a WPDA into a WCFG

Input: $\mathcal{PDA} = (\mathcal{S}, \Sigma, \Gamma, \delta, Q_0, Z_0, \{Q_F\}, \omega)$
Output: $\mathcal{CFG} = (\mathcal{Q}, \Sigma, [Q_0], \mathcal{R}, \pi)$

```
Create the axiom [Q_0]
for each state U_x ∈ S do
    Create the nonterminal [Q_0 Z_0 U_x]
    Create the rule [Q_0] → [Q_0 Z_0 U_x]
end for
for each transition (U_x, ⟨x, α, β⟩, V_y) do
    if β = ∅ (pop) then
        Create a nonterminal [U_x α V_y]
        Create the rule r = [U_x α V_y] → x
        π(r) ← ω(U_x, ⟨x, α, ∅⟩, V_y)
    end if
    if β = α (conversion transition) then
        for each Q_i ∈ S do
            Create nonterminals [U_x α Q_i] and [V_y α Q_i]
            Create the rule r = [U_x α Q_i] → x[V_y α Q_i]
            π(r) ← ω(U_x, ⟨x, α, β⟩, V_y)
        end for
    end if
    if β = xα, x ∈ Γ (push) then
        for each (Q_i, Q_j) ∈ S^2 do
            Create nonterminals [U_x α Q_j], [V_y α Q_i] and [Q_i α Q_j]
            Create the rule r = [U_x α Q_j] → x[V_y x Q_i][Q_i α Q_j]
            π(r) ← ω(U_x, ⟨x, α, xα⟩, V_y)
        end for
    end if
end for
```

The adapted algorithm maintains a list of production rules and updates the $\ell[X]$ according to the formula above. The sketch of the algorithm is as follows:

- Initialize $\ell([X])$ to ∞ for each nonterminal $[X]$
- For each production rule $[X] \to \alpha_1 \dots \alpha_n$ update $\ell([X])$ as follows:
  $\ell([X]) \leftarrow \min\{\ell([X]), \pi(r) + \sum_{i=1}^{n} \ell(\alpha_i)\}$

The algorithm terminates when all the $\ell[X]$ have the right value and no additional update is possible. Implementing this algorithm with Fibonacci heaps leads to a $O(|\mathcal{Q}| \log |\mathcal{Q}| + |\mathcal{R}|)$ complexity [23], which corresponds to $O(|\mathcal{A}|^5 \times |V|^2 \times |\mathcal{E}|)$.

With the correct values of $\ell[X]$, it is trivial to generate the word with the minimum weight derivation tree.

E. Deriving the shortest path from its trace

Algorithm 3 is a generalization of an algorithm proposed in [10]. It takes as input the minimum weight trace $\mathcal{T}_\mathcal{P}$ accepted by $\mathcal{PDA}$ and computes the path $\mathcal{P}$ that matches it[5].

Algorithm 3 starts on $nodes[1] = S$ then checks at each step all the links in $\mathcal{E}$ which match the current letter (protocol) in $\mathcal{T}_\mathcal{P}$. If $\mathcal{T}_\mathcal{P} = x_1 x_2 \dots x_n$ ($x_i \in \mathcal{A} \cup \overline{\mathcal{A}} \cup \underline{\mathcal{A}}$), then at each step i, the algorithm starts from each node U in $nodes[i]$ and adds to $links[i]$ all the links (U, V) which match x_i. Each V is added in $nodes[i+1]$. The value $weights[(U, V), i]$

[5] It is possible that several paths match the trace. In this case the path can be chosen randomly or according to a load-balancing policy.

is the cost of using link (U, V) at step i. It corresponds to the weight $h(U, f_i, V)$ where f_i is the adaptation function used at step i. When the trace $\mathcal{T}_\mathcal{P}$ is completely covered, a classical shortest path algorithm from S to D in the graph $(nodes, links, weights)$ computes the minimum weight path.

Algorithm 3 Computing the shortest path

Input: The network $\mathcal{N}$ and $\mathcal{T}_\mathcal{P}$
Output: The shortest path $\mathcal{P}$
$nodes[1] \leftarrow S$; $i \leftarrow 2$
while The trace is not completely covered **do**
for each $U \in nodes[i]$, $V \in \mathcal{V}$ *s.t.* $(U, V) \in \mathcal{E}$ **do**
if $x_i \in \mathcal{A}$, $x_i \in Out(U)$, $x_i \in In(V)$ and $(x_{i-1} \rightarrow x_i) \in \mathcal{CO}(U)$ **then**
Add (U, V) in $links[i]$ and V in $nodes[i+1]$
$weights[(U, V), i] \leftarrow h(U, (x_{i-1} \rightarrow x_i), V)$
end if
if $x_i \in \overline{\mathcal{A}}$, $x_i \in Out(U)$, $x_i \in In(V)$ and $(x_{i-1} \rightarrow x_{i-1}x_i) \in \mathcal{EN}(U)$ **then**
Add (U, V) in $links[i]$ and V in $nodes[i+1]$
$weights[(U, V), i] \leftarrow h(U, (x_{i-1} \rightarrow x_{i-1}x_i), V)$
end if
if $x_i \in \underline{\mathcal{A}}$, $x_i \in Out(U)$, $x_i \in In(V)$ and $\overline{(x_i \rightarrow x_i x_{i-1})} \in \mathcal{DE}(U)$ **then**
Add (U, V) in $links[i]$ and V in $nodes[i+1]$
$weights[(U, V), i] \leftarrow h(U, \overline{(x_i \rightarrow x_i x_{i-1})}, V)$
end if
end for
$i++$
end while
Compute The shortest path from S to D in $(nodes, links)$

Complexity of Algorithm 3. The complexity of Algorithm 3 is bounded by $O(|\mathcal{T}_\mathcal{P}| \times |\mathcal{V}| \times |\mathcal{E}|)$ in the worst case.

APPENDIX B
PROOF THAT SYM-HAM IS NP-COMPLETE

Problem SYM-HAM. Given a directed symmetric graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a pair of nodes (S, D), is there a Hamiltonian path from S to D in $\mathcal{G}$?

Proposition 3: SYM-HAM is NP-complete.

Proof: First, it is clear that SYM-HAM is in NP. Thus, we prove its NP-hardness by providing a polynomial reduction from the Hamiltonian path problem in *undirected* graphs to SYM-HAM. Consider an undirected graph $\mathcal{H} = (\mathcal{V}', \mathcal{E}')$ and a pair of nodes (S', D'). It is NP-complete to know whether there is an undirected Hamiltonian path between S' and D'. The reduction builds an instance of SYM-HAM as follows: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \mathcal{V}'$. For each undirected edge (U, V) in $\mathcal{H}$, create the directed links (U, V) and (V, U) in $\mathcal{G}$.

Let $\mathcal{P}' = SU_1U_2 \dots U_nD$ be a Hamiltonian path in $\mathcal{H}$. For each edge (U_i, U_{i+1}) in $\mathcal{H}$, one can take the corresponding directed link (U_i, U_{i+1}) in $\mathcal{G}$ and construct a Hamiltonian path in $\mathcal{G}$.

Now let $\mathcal{P}' = SU_1U_2 \dots U_nD$ be a (directed) Hamiltonian path in $\mathcal{G}$. By replacing each link (U_i, U_{i+1}) by the corresponding undirected edge (in $\mathcal{H}$), one obtains a path visiting all the nodes exactly once in $\mathcal{H}$ (as $\mathcal{G}$ and $\mathcal{H}$ have the same set of nodes). Thus, the obtained path is a Hamiltonian path in $\mathcal{H}$.

So $\mathcal{H}$ admits an undirected Hamiltonian path between S' and D' if and only if $\mathcal{G}$ admits a directed Hamiltonian path from S to D. ■