HAL
open science

Algorithmic and complexity aspects of path computation in multi-layer networks

Mohamed Lamine Lamali, Nasreddine Fergani, Johanne Cohen

To cite this version:

Mohamed Lamine Lamali, Nasreddine Fergani, Johanne Cohen. Algorithmic and complexity aspects of path computation in multi-layer networks. IEEE/ACM Transactions on Networking, inPress, 26 (6), pp.2787 - 2800. hal-01706362v2

HAL Id: hal-01706362

https://hal.science/hal-01706362v2

Submitted on 1 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Algorithmic and complexity aspects of path computation in multi-layer networks

Mohamed Lamine Lamali[1], Nasreddine Fergani[2], Johanne Cohen[3],
[1]LaBRI, Université de Bordeaux. F-33405 Talence cedex. France.
[2]MathWorks. Cambridge, CB4 0HH. UK.
[3]LRI, Univ. Paris-Sud, CNRS, Université Paris-Saclay. 91405 Orsay Cedex, France.

mohamed_lamine.lamali@u-bordeaux.fr johanne.cohen@lri.fr

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 to 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 to automate the routing configurations. Software-Defined Networking based architectures, offer an opportunity 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 we study several cases. If there is no bandwidth constraint, we propose a polynomial algorithm that compute the optimal path. We also give lower and upper bounds on the optimal path length. On the other hand, we show that the problem is NP-hard if there is a bandwidth constraint (or other Quality of Service parameters), even if there is only two protocols and in a symmetric graph. We study the complexity and the scalability of our algorithms and evaluate their performances on real and random topologies. The results show that they are faster than the previous ones proposed in the literature. These algorithms can also have important applications in automatic tunneling.

***Index Terms*—Protocols, Encapsulation, Path computation, multi-layer networks, Complexity theory, automata theory.**

I. Introduction

Carrier-grade networks generally encompass several layers that involve 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, then it must be decapsulated (or unwrapped) further in the path. If several encapsulations are nested, then the corresponding decapsulations must occur in the right order. A path complying with this requirement is said to be feasible. 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.

Dealing with protocol heterogeneity is increasingly important nowadays. In addition to the IPv4/IPv6 migration, this heterogeneity appears in tunneling, some architectures (e.g., The Pseudo-Wire architecture [3]), hybrid networks, 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 for reducing operational and management costs.

OpenFlow, and specifically protocol-agnostic versions such as Protocol-Oblivious Forwarding (POF) [4] or P4 [5], are an opportunity to design such a control plane. Some previous works [6], [7] 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 they circumscribe it to a single layer [8] or to the IPv4/IPv6 migration [9]. 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 (Edsger) Dijkstra's one cannot achieve the task as they do not handle these functions. Here, we design several algorithms to compute minimum cost paths dealing with protocol changes and adaptation functions.

Our contributions:

1) We widely generalize the language theoretic approach of Lamali *et al.* [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 (or weight). We drastically improve the algorithm complexity and show its efficiency through simulations on real and random 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 prob-

ability of existence of a feasible path hugely increases.

3) We provide lower and upper bounds on the length of the shortest feasible path: We show that the shortest feasible path length can be superpolynomial in the network size, but cannot be more than doubly exponential.
4) We prove that path computation in multi-layer networks under bandwidth constraint is NP-hard even with two protocols and on symmetric[1] 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 we show its efficiency through simulations.
5) 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.

Taking into account an additive metric, such as the delay, is important in the shortest feasible path computation. These metrics may depend on the links, but also on the performed adaptation functions along the path. For example, an encapsulation may induce a higher delay than a simple forwarding, a conversion of headers may induce additional computational costs and a delay due to the overhead, etc. Another example is provided by NRENs[2], which have optical and IP interconnection points. The technology choice (optic or IP) may lead to a different path cost, since the technical characteristics (reliability, delay, bandwidth, etc.) of these technologies may differ. However, the weighted problem is challenging. In [10], the shortest protocol sequence corresponding to the shortest feasible path is derived. However, this sequence does not always correspond to the minimum weight feasible path. Thus, the algorithm should not focus on the length of the protocol sequence, but on the way and the cost of its computation. This involves more complex tools (weighted models in language theory). The reason of the efficiency of our solution is that it takes advantage of the nontrivial structure of feasible paths. Having a better view of the structure of the candidate solutions allows to speed up the solution search. Moreover, this structure fits in well-known models of language theory, and there are powerful tools to generate the solutions in these models.

The most important possible applications of our algorithms are the unification of control planes and automatic (nested) tunneling. This paper extends the work published in [13]. It provides the detailed algorithms, together with their proofs of correctness and complexity analysis. It also provides lower and upper bounds on the shortest path length. This has an important impact on the complexity of the problem.

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; Section V provides lower and upper bounds on the shortest path length; Section VI studies the complexity of path computation under bandwidth constraint and proposes heuristic solutions to tackle the problem; Section VII proposes the first algorithm computing paths under multiple additive QoS constraints; Section VIII shows the efficiency of our algorithms through simulations, it also studies the phase transition phenomenon in multi-layer networks; finally, Section IX 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[3]. Figure 1 (inspired by the Inter-Provider Reference Model [15]) 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 Domains 1 and 2.

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 instance of a 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 (i.e., the sequences of encapsulated protocols) 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 frame (by the border router of Domain 2) whereas at this stage the frame encapsulates IPv4 packets. Note that we only considered a unidirectional path. The reverse path from the LAN to the data-center should be computed independently.

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

[1]We define a symmetric directed graph as a graph where a link (U, V) exists if and only if the reverse link (V, U) exists.

[2]National Research and Education Networks.

[3]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 [14].

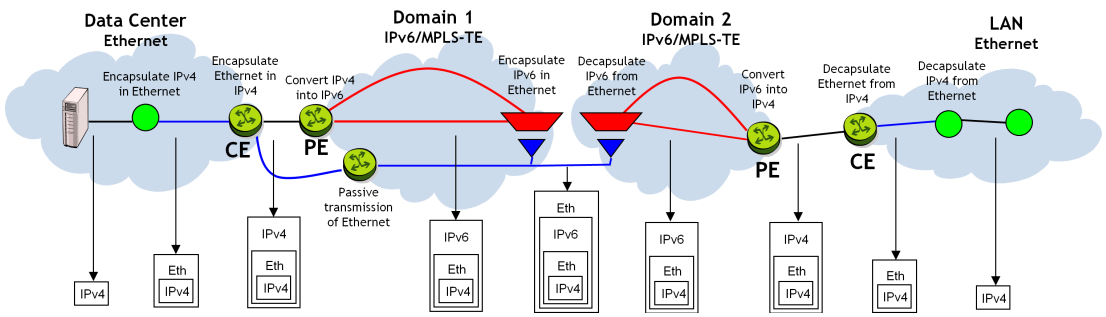


Fig. 1. Carrier-grade network comprising several domains and different layers.

functions should occur in the right order. Moreover, feasible paths can involve loops and their subpaths are not necessary feasible [11], [16]. 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.* [17] described a model and algorithms to compute a path under wavelength continuity constraints. Zhu *et al.* [18] addressed the same problem in WDM mesh networks tackling traffic grooming issues. In [19], 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 [20], 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-hard under bandwidth constraint. They proposed a Breadth-First Search (BFS) algorithm that explores all possible paths until finding a feasible one. In [10], Lamali *et al.* proved that the problem is polynomial if the bandwidth constraint is relaxed. They used 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.* [21] underlined the need of path computation algorithms in NRENs. They proposed a new matrix-based model for multi-layer networks and k-shortest paths algorithms to tackle the problem. However, the model deals with technologies[4] 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 follow a methodology based on the probabilistic distribution of the adaptation functions over the nodes.

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

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

A multi-layer network is 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. The number of nodes is denoted by $|\mathcal{V}| = n$ and the number of links is denoted by $|\mathcal{E}| = m$.
- $\mathcal{A} = \{a, b, c, \dots\}$ is the set of protocols available in the network, but not necessarily at each router. The number of protocols is denoted by $|\mathcal{A}| = \lambda$.
- 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. It 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 $a = b$. 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* (or cost) 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 only associated to the links or to both links and adaptation functions.

B. The protocol stack

A sequence of adaptation functions induces a protocol stack. For example, the sequence

$$(a \to a)(a \to ab)(b \to b)(b \to ba)(a \to ab)$$

induces the stack $abab$ (from bottom to top). Let $f_0 \dots f_k$ be an adaptation function sequence. For each $i \leq k$, H_i denotes the protocol stack induced by $f_0 \dots f_i$. The protocol at the top of a stack H is denoted by $Top(H)$ and the protocol just below $Top(H)$ in the stack is denoted by $Top^{-1}(H)$. The "forbidden" stack (that should not be confused with the empty word ϵ) is denoted by $\emptyset$.

More formally, let f be an adaptation function, and let H be a stack and H' the same stack without its top protocol, i.e., $H = H'.Top(H)$[5], where H' is eventually empty. By abuse of language, we will also denote by f the function taking as argument a stack and performing the adaptation function on this stack:

- if $f = (a \to b)$ and $Top(H) = a$, then $f(H) = H'.b$
- if $f = (a \to ab)$ and $Top(H) = a$, then $f(H) = H.b$
- if $f = \overline{(a \to ab)}$ and $Top(H) = b$ and $Top^{-1}(H) = a$, then $f(H) = H'$
- $f(H) = \emptyset$ otherwise.

In fact, $f(H) = \emptyset$ if the adaptation function f cannot handle the stack H (e.g., f is a decapsulation while there is no encapsulated protocol in H). Note that $f(\emptyset) = \emptyset$ for any function in our context. Thus, the protocol stack H_i induced by a sequence of adaptation functions $f_0 \dots f_i$, is recursively defined as following:

- $H_0 = a$ if $f_0 = (a \to a)$
- $H_i = f_i(H_{i-1})$

This allows to give a formal definition of path feasibility.

C. 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 $\mathcal{P} = Sf_0U_1f_1U_2f_2 \dots U_kf_kD$ where each U_i, $i = 1, \dots, k$, is a node and each f_i is an adaptation function (f_0 being dummy). A path $\mathcal{P}$ is *feasible* if:

1) The sequence $SU_1U_2 \dots U_kD$ is a path in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and each $f_i \in \mathcal{F}(U_i)$;
2) The stack induced by the sequence $f_0 \dots f_k$ is $H_k = a$, where a is the protocol emitted by the source S.

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

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

[5]The notation "." stands for a simple concatenation. For example, if $H = abab$ then $H.b = ababb$.

IV. Path computation without bandwidth constraint

This section proposes a polynomial algorithm to resolve the path computation problem without bandwidth constraint.

A. Problem definition

The problem we aim to resolve is to compute the feasible path (if any) of minimum cost from a source S to a destination D.

Problem 1.

$$\min\ h(\mathcal{P}) = \sum_{(U,f,V) \in \mathcal{P}} h(U, f, V)$$

$$s.t.\ \ \mathcal{P} \textit{ is a feasible path from } S \textit{ to } D$$

B. General approach

Lamali *et al*. [10] proposed a language theoretic[6] 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 the multi-layer network as a Push-Down Automaton (PDA);
2) If the considered metric is the number of encapsulations, transform the PDA in order to bypass passive transitions;
3) Convert the PDA to a Context-Free Grammar (CFG);
4) Compute the length of the shortest word generated by the CFG, then generate this word. 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 minimum cost 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.
- The conversion of the PDA into a CFG is adapted in order to assign weights to the production rules;
- Since the production rules are weighted, the goal is no longer to compute the shortest protocol sequence but the one having the minimum weight derivation tree. The derivation tree of a word is the way it is generated by a given grammar, a more formal definition is given in the proof of Proposition 4. This is done thanks to our adaptation of Knuth's algorithm described in [23].
- The computation of the path matching the protocol sequence is modified to take into account the weights.

In addition to these improvements, the algorithm complexity is drastically decreased from $O(\lambda^8 n^7)$ in [10] to $O\left(\lambda^5 n^2 m\right)$.

[6]For the language theoretic terminology, the reader is referred to [22].

This is due to the uselessness of the PDA transformation to bypass passive sequences, since this operation is costly. Moreover, adapting Knuth's algorithm [23] to compute minimum weight derivation tree allows to merge the generation of the shortest (here minimum cost) word and the computation of its length (here its cost).

C. Detailed algorithms

1) Theoretical language aspects of multi-layer paths

Considering a path $\mathcal{P} = Sf_0U_1f_1U_2f_2\dots U_kf_kD$, and its adaptation function sequence $f_0\dots f_k$. Its trace $\mathcal{T}_\mathcal{P} = x_1\dots x_{k+1}$ is defined as the sequence of protocols used along path $\mathcal{P}$. However, the symbols corresponding to the protocols will be modified in order to indicate where the adaptation functions will be performed along the path. If a protocol appears overlined (resp. underlined) in the trace, it means that an encapsulation (resp. a decapsulation) should be performed at this stage. These indications will be used by Algorithm 3 to derive the whole path from the trace. More formally, we introduce the set $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$ and the set $\underline{\mathcal{A}} = \{\underline{a} \mid a \in \mathcal{A}\}$ as alphabets. For each x_i:

- $x_i = a$ then f_i is a conversion;
- $x_i = \overline{a}$ then f_i is an encapsulation;
- $x_i = \underline{a}$ then f_i is an decapsulation;

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 $\mathcal{DE}(U)$.

$In(U)$ (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 $\underline{(a \to ab)} \in \mathcal{EN}(U)$ 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 $a \in Out(U)$

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 context-free 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.

2) 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 (resp. Q_j) is the state of $\mathcal{PDA}$ before (resp. after) the transition, $x \in \Sigma \cup \{\epsilon\}$, where ϵ is the empty word, is an input symbol, $\alpha \in \Gamma$ is the symbol that is popped from the top of the stack, and $\beta \in \Gamma^*$ is the symbol sequence that is pushed on the top of the stack.

Remark. WPDAs are more often formalized as 6-tuples $(\mathcal{S}, \Gamma, \mathcal{M}, Q_0, Z_0, \mathcal{S}_F)$ where $\mathcal{M}$, called the *Push-Down transition matrix*. For simplicity, we opt 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 [24].

3) 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 a 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 $\mathcal{PDA} = (\mathcal{S}, \Sigma, \Gamma, \delta, Q_0, Z_0, \{Q_F\}, \omega)$
$\Sigma \leftarrow \mathcal{A} \cup \overline{\mathcal{A}} \cup \underline{\mathcal{A}}$; $\Gamma \leftarrow \mathcal{A}$
Create a single state Q_0 corresponding to node S
Create a dummy final state Q_F
for each node $U \neq S$ in $\mathcal{V}$ and each $x \in In(U)$ **do**
  Create a state U_x
for each state U_x s.t. $(S, U) \in \mathcal{E}$ and each $x \in Out(S)$ **do**
  Create the transition $t = (Q_0, \langle \epsilon, Z_0, Z_0\rangle, U_x)$
  $\omega(t) \leftarrow 0$
for each $x \in In(D)$ **do**
  Create the transition $t = (D_x, \langle x, Z_0, \emptyset\rangle, Q_F)$
  $\omega(t) \leftarrow 0$
for each link $(U, V) \in \mathcal{E}$ s.t. $U \neq S$ **do**
  for each $(x \to y) \in \mathcal{CO}(U)$ **do**
    if $y \in In(V)$ **then**
      for all $\alpha \in \Gamma$ **do**
        Create the transition $t = (U_x, \langle x, \alpha, \alpha\rangle, V_y)$
        $\omega(t) \leftarrow h(U, (x \to y), V)$
  for each $(x \to xy) \in \mathcal{EN}(U)$ **do**
    if $y \in In(V)$ **then**
      for all $\alpha \in \Gamma$ **do**
        Create the transition $t = (U_x, \langle \overline{x}, \alpha, x\alpha\rangle, V_y)$
        $\omega(t) \leftarrow h(U, (x \to xy), V)$
  for each $\overline{(x \to xy)} \in \mathcal{DE}(U)$ **do**
    if $x \in In(V)$ **then**
      Create the transition $t = (U_y, \langle \underline{y}, x, \emptyset\rangle, V_x)$
      $\omega(t) \leftarrow h(U, \overline{(x \to xy)}, V)$

Complexity of Algorithm 1. The complexity of Algorithm 1 is in $O(\lambda^3 m)$. The number of states created is at worst $2 + \lambda(n-1)$. The number of transitions created is in $O(\lambda^3 m)$, which is also the complexity of the whole algorithm.

Proposition 1. *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. Let $\mathcal{P} = Sf_0U_1f_1U_2f_2\dots U_kf_kD$ be a feasible path. 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 $f_i = (x \to y)$
- $t = ((U_i)_x, \langle \overline{x}, \alpha, x\alpha\rangle, (U_{i+1})_y)$ if $f_i = (x \to xy)$
- $t = ((U_i)_x, \langle \underline{x}, y, \emptyset\rangle, (U_{i+1})_y)$ if $f_i = \overline{(y \to yx)}$

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

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

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

Thus the path $Sf_0U_1f_1U_2f_2\dots U_kf_kD$ is feasible in $\mathcal{N}$. □

The weight of a transition sequence is the sum of the weights of each transition (i.e., $\omega(\{t_1, t_2, \dots, t_k\}) = \sum_{i=1}^{k} \omega(t_i)$). The weight 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, a same word may be accepted by different transition sequences. We consider only the transition sequence of minimum weight that accepts w. More formally, $\omega(w) = \min_{t_1,\dots,t_k \in \delta} \omega(\{t_1 \dots t_k\})$ s.t. $\{t_1 \dots t_k\}$ accepts w.

4) From the WPDA to a WCFG

In order to compute the minimum weight trace and its corresponding path, $\mathcal{PDA}$ is converted into a weighted Context-Free Grammar (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 method described in [22]. 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 $O(|\Gamma| \times |\mathcal{S}|^2)$. The number of production rules is bounded by $O(|\delta| \times |\mathcal{S}|^2)$, which is the worst case complexity of Algorithm 2. This corresponds to $O\left(\lambda^5 n^2 m\right)$.

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.

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 $U_x \in \mathcal{S}$ **do**
    Create nonterminal $[Q_0 Z_0 U_x]$ and rule $[Q_0] \to [Q_0 Z_0 U_x]$
  for each transition $(U_x, \langle x, \alpha, \beta\rangle, V_y)$ **do**
    if $\beta = \emptyset$ (pop) **then**
      Create a nonterminal $[U_x \alpha V_y]$ and rule $[U_x \alpha V_y] \to x$
      $\pi(r) \leftarrow \omega(U_x, \langle x, \alpha, \emptyset\rangle, V_y)$
    if $\beta = \alpha$ (conversion transition) **then**
      for each $Q_i \in \mathcal{S}$ **do**
        Create nonterminals $[U_x \alpha Q_i]$ and $[V_y \alpha Q_i]$
        Create the rule $r = [U_x \alpha Q_i] \to x[V_y \alpha Q_i]$
        $\pi(r) \leftarrow \omega(U_x, \langle x, \alpha, \beta\rangle, V_y)$
    if $\beta = x\alpha,\ x \in \Gamma$ (push) **then**
      for each $(Q_i, Q_j) \in \mathcal{S}^2$ **do**
        Create nonterminals $[U_x \alpha Q_j]$, $[V_y \alpha Q_i]$ and $[Q_i \alpha Q_j]$
        Create the rule $r = [U_x \alpha Q_j] \to x[V_y x Q_i][Q_i \alpha Q_j]$
        $\pi(r) \leftarrow \omega(U_x, \langle x, \alpha, x\alpha\rangle, V_y)$

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 $w = \alpha_1 \dots \alpha_k$ (with $\alpha_i \in \{\mathcal{Q} \cup \Sigma \cup \{\epsilon\}\}$) then $\ell(w) = \sum_{i=1}^{k} \ell(\alpha_i)$.
- 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 [23] 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 minimum cost path to compute. 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 $[X] \to \alpha_1 \dots \alpha_n$:
  $\ell([X]) \leftarrow \min\{\ell([X]), \pi(r) + \sum_{i=1}^{n} \ell(\alpha_i)\}$

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 [25], which corresponds to $O(\lambda^5 n^2 m)$.

When the value of $\ell([X])$ is updated, the production rule that originates this update is stored as in the classical Dijkstra's algorithm. Then, starting from the axiom, each nonterminal is replaced by the right side of its corresponding production rule, until the generated word only contains terminals. This allows to generate the desired word.

5) Deriving the minimum cost path from its trace

Algorithm 3 generalizes the 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.

Algorithm 3 starts on $nodes[1] = S$ then checks at each step all the links in $\mathcal{E}$ that match the current letter (protocol) in $\mathcal{T}_\mathcal{P}$. If $\mathcal{T}_\mathcal{P} = x_1 x_2 \dots x_k$ ($x_i \in \mathcal{A} \cup \overline{\mathcal{A}} \cup \underline{\mathcal{A}}$), 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 $weights[(U, V), i]$ 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 covered, a classical minimum cost path algorithm from S to D in the graph $(nodes, links, weights)$ computes the minimum weight path.

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

Algorithm 3 Computing the minimum cost path

Input: The network $\mathcal{N}$ and $\mathcal{T}_\mathcal{P}$
Output: The minimum cost 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 $(x_{i-1} \to x_i) \in \mathcal{CO}(U)$ **then**
      Add (U, V) in $links[i]$ and V in $nodes[i+1]$
      $weights[(U, V), i] \leftarrow h(U, (x_{i-1} \to x_i), V)$
    if $x_i \in \overline{\mathcal{A}}$, $x_i \in Out(U)$, $x_i \in In(V)$ and $(x_{i-1} \to x_{i-1}x_i) \in \mathcal{EN}(U)$ **then**
      Add (U, V) in $links[i]$ and V in $nodes[i+1]$
      $weights[(U, V), i] \leftarrow h(U, (x_{i-1} \to x_{i-1}x_i), V)$
    if $x_i \in \underline{\mathcal{A}}$, $x_i \in Out(U)$, $x_i \in In(V)$ and $(x_i \to x_i x_{i-1}) \in \mathcal{DE}(U)$ **then**
      Add (U, V) in $links[i]$ and V in $nodes[i+1]$
      $weights[(U, V), i] \leftarrow h(U, (x_i \to x_i x_{i-1}), V)$
  $i++$
Compute the minimum cost path from S to D in $(nodes, links, weights)$

V. Bounds on feasible path length

In this section, we distinguish the *cost* of a path, i.e., the sum of the weights of the links and the adaptation functions involved in it, from the *length* of the path, i.e., the number of links involved in it. In both cases, links in loops are counted as many times as they are crossed by the path.

By computing the ℓ-value of the axiom $[Q_0]$, Knuth's algorithm decides if a given multi-layer network contains a feasible path between a specified pair of nodes. If $\ell([Q_0]) = \infty$, then there is no feasible path. Otherwise, there is a feasible path of minimum cost $\ell([Q_0])$. Thus, the existence problem is solvable in $O(\lambda^5 n^2 m)$, which depends only on the network parameters. However, to effectively and explicitly compute the feasible path $\mathcal{P}$ of minimum cost (if any), one needs to generate its trace $\mathcal{T}_\mathcal{P}$ and to find the feasible path matching it (Algorithm 3), which are in $O(|\mathcal{T}_\mathcal{P}|)$ and $O(|\mathcal{T}_\mathcal{P}|nm)$ respectively. Note that $|\mathcal{T}_\mathcal{P}|$ is also the length of the feasible path of minimum cost. Thus, the complexity of the whole path computation process is in $O\left(\max\left\{\lambda^5 n^2 m, |\mathcal{T}_\mathcal{P}|nm\right\}\right)$, which is polynomial in the network parameters and in the minimum cost path length. However, to accurately estimate this complexity, bounds on the value of $|\mathcal{T}_\mathcal{P}|$ are needed.

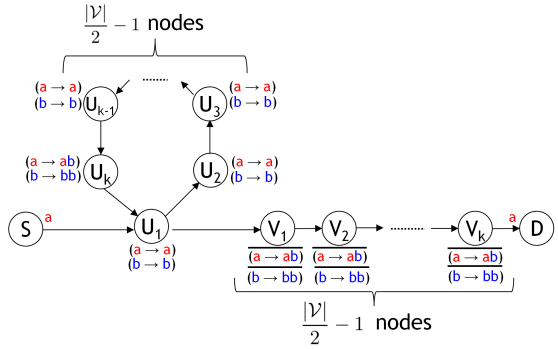


Fig. 2. Example of multi-layer network in which the only feasible path is of length $\Omega(n^2)$.

A. Lower bounds on the shortest feasible path

Obviously, if $\mathcal{P}$ does not involve loops, then $|\mathcal{T}_\mathcal{P}|$ is upper bounded[7] by m. Otherwise, can the length of the minimum cost path be upper bounded by a linear function of n or m?

Unfortunately, the answer is no. Consider the example depicted on Figure 2. The nodes (except S and D) are partitioned into two sets. For convenience and without loss of generality, we suppose that n is even. Nodes $U_1, \ldots, U_k$ where $k = \frac{n}{2} - 1$ form a loop. Each node in this loop is passive (only able to transmit passively any protocol), except the last one which is able to encapsulate any protocol in protocol b. Nodes $V_1, \ldots, V_k$ connect U_1 to D via a directed path. All these nodes are able to decapsulate any protocol from protocol b (thus they delete an occurrence of protocol b from the top of the protocol stack). To reach D, a feasible path should cross the sequence of nodes $V_1, \ldots, V_k$ and delete an occurrence of b at each of these nodes. Thus, when reaching node V_1, the protocol stack of a feasible path should contain k occurrences of b at its top. However, each crossing of the loop $U_1, \ldots, U_k$ adds only one occurrence of b on the top of the stack (at U_k). Therefore, a feasible path should cross the loop k times before reaching V_1. The length of the shortest (and actually the only one) feasible path is $k^2 + k + 2 = \frac{n^2}{4} - \frac{n}{2} + 2$.

Proposition 2 generalizes this result.

Proposition 2. *For any number of nodes n and any set of protocols $\mathcal{A} = \{a_1, \ldots, a_\lambda\}$, there is a multi-layer network $\mathcal{N}$ with two specified nodes S and D such that the shortest feasible path from S to D contains $\Omega\left(\left(\frac{n}{\lambda}\right)^\lambda\right)$ links.*

Proof. Let $\mathcal{N}$ be a multi-layer network and suppose without loss of generality that $n = \lambda k + 1$ for some integer k. We first construct the underlying graph as following: Create a directed path containing all the nodes where the first one is S and the last one is D. The node $U_{i,j}$ $(0 \leq j < k)$ denotes the $ki + j + 1^{\text{th}}$ node in the path. Thus D is denoted by $U_{\lambda,0}$. Then create a link from node $U_{i,0}$ to S for all $i \leq k - 1$. The adaptation functions are allocated as follows:

- Node S emits protocol a_1 and nodes $U_{0,1}, \ldots, U_{0,k-1}$ are passive and can only transmit any protocol as received,

[7] Since the weights and other possible QoS parameters depend on the links (and the adaptation functions) but not on vertices, paths involving several times the same vertices are not an issue and are considered as loopless.

- Nodes $U_{i,1}, \ldots, U_{i,k-1}$ $(1 \leq i \leq \lambda - 1)$ are only able to deacpsulate protocol a_{i+1} from any protocol,
- Nodes $U_{i,0}$ $(1 \leq i \leq \lambda - 1)$ are only able to encapsulate any protocol in protocol a_{i+1}.

A path must have the sequence a_{i+1}^{k-1} at the top of the protocol stack[8] to cross the nodes $U_{i,1}, \ldots, U_{i,k-1}$ $(1 \leq i \leq \lambda - 1)$ and reach $U_{i+1,0}$. To do so, it must cross the node $U_{i,0}$ $k-1$ times because it is the only node able to add an occurrence of a_{i+1} on the top of the protocol stack. And at each time it reaches $U_{i,0}$ (except the last one), it must cross the link $(U_{i,0}, S)$ again. Let ϕ_i be the shortest path length (in links) to reach $U_{i,0}$. Then reaching $U_{i,0}$ $k-1$ times requires $(k-2)(\phi_i + 1) + \phi_i$ links. Thus, reaching $U_{i+1,0}$ requires $(k-2)(\phi_i + 1) + \phi_i + k$ links. Rearranged, this gives the recurrent equation

$$\phi_{i+1} = (k-1)\phi_i + 2k - 2 \text{ with } \phi_1 = k \qquad (1)$$

Resolving the equation gives $\phi_\lambda = \Omega(k^\lambda)$. Replacing k by $\frac{n-1}{\lambda}$ concludes the proof.

□

The previous result gives an example of a shortest feasible path that is exponential in the number of protocols but polynomial in the number of nodes. However, it is possible to obtain a superpolynomial shortest path with only two protocols. It is well-known that the shortest word generated by a CFG can be of exponential length in the CFG size. On the other hand, there is a polynomial relation between the size of a (W)CFG and an equivalent (W)PDA. This suggests that the shortest feasible path can be exponential. However, the classical algorithm that converts a CFG into a PDA produces a PDA with a fixed number of states (generally only one - see [22] for more details). Thus, the resulting (W)PDA is unlikely to be converted into a multi-layer network. Instead, we build a multi-layer network where the shortest path length is superpolynomial. Proposition 3 details this construction.

Proposition 3. *For any number of nodes n, there is a multi-layer network having 2 protocols with two specified nodes S and D, such that the shortest feasible path from S to D contains $\Omega\left(\sqrt{n}2^{\Omega(\sqrt{n})}\right)$ links.*

Proof. Let $\mathcal{N}$ be the multi-layer network built in the proof of Proposition 2 with λ protocols. We will convert it into a multi-layer network $\mathcal{N}'$ with two protocols a and b. The key idea of the proof is to emulate a protocol a_{i+1} with the sequence of protocols ab^ia. To do so, each node $U_{i,0}$ $(1 \leq i \leq \lambda - 1)$ is replaced by a sequence of $i+2$ nodes $U_{i,0}^0 \ldots U_{i,0}^{i+1}$ where $U_{i,0}^0$ and $U_{i,0}^{i+1}$ are able to encapsulate any protocol in protocol a and the nodes $U_{i,0}^1 \ldots U_{i,0}^i$ are able to encapsulate any protocol in protocol b. Thus, crossing this sequence push ab^ia on the top of the protocol stack. Analogously, each node $U_{i,j}$ $(1 \leq i \leq \lambda - 1$ and $1 \leq j \leq k-1)$ is replaced by a sequence of $i+2$ nodes $U_{i,j}^0 \ldots U_{i,j}^{i+1}$ where $U_{i,j}^0$ and $U_{i,j}^{i+1}$ are able to deapsulate protocol a from any protocol and the nodes $U_{i,j}^1 \ldots U_{i,j}^i$ are able to decapsulate protocol b from any protocol. To cross this sequence of nodes, the top of the stack should contain ab^ia. Now, let ϕ_i be the shortest path length (in links) to reach $U_{i,0}^{i+1}$. The recurrent relation (1) becomes

$$\phi_{i+1} = (k-1)\phi_i + (i+3)k - 2 \text{ with } \phi_1 = k + 2$$

Resolving the equation gives $\phi_\lambda = \Omega(\lambda k^\lambda)$. Replacing λ by $\frac{n-1}{k}$ and setting $k = 3$ gives $\Omega(n3^{\frac{n}{3}})$. However, here n is the number of nodes of $\mathcal{N}$ while the path we exhibit is in $\mathcal{N}'$. The number of nodes in $\mathcal{N}'$ is $n' = O(n^2)$, thus $n = \Omega(\sqrt{n'})$. The shortest feaisible path length in $\mathcal{N}'$ is in $\Omega\left(\sqrt{n'}3^{\Omega(\sqrt{n'})}\right) = \Omega\left(\sqrt{n'}2^{\Omega(\sqrt{n'})}\right)$ □

[8]The notation a^k stands for $\underbrace{aa\ldots a}_{k \text{ times}}$.

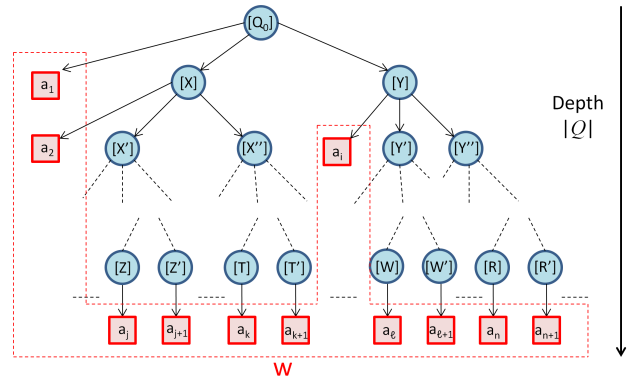


Fig. 3. Derivation tree of a word corresponding to a feasible path.

B. Upper bound on the shortest feasible path length

Proposition 4. *For any multi-layer network $\mathcal{N}$, the shortest feasible path (if any) between two nodes contains at most $O(2^\psi)$ hops, where $\psi = (\lambda+1)\lambda^2 n^2$.*

Proof. We prove that the shortest word (if any) generated by a CFG corresponding to any multi-layer network contains at most $2^{|\mathcal{Q}|} - 1$ terminals, where $|\mathcal{Q}|$ is the number of nonterminals in the CFG. The proof is similar to the one showing bounds on the length of the shortest word generated by a CFG in *Chomsky normal form* [22].

Let $\mathcal{N}$ be any multi-layer network and let (S, D) be the source and the destination of a feasible path in $\mathcal{N}$. Recall that the CFG corresponding to $\mathcal{N}$ and produced by Algorithm 2 contains production rules of the following forms:

1) $[X] \to a$
2) $[X] \to a[Y]$
3) $[X] \to a[Y][Z]$

Where a is a terminal or the empty word ϵ, and $[X]$,$[Y]$,$[Z]$ are nonterminals. Let w be the trace of the shortest feasible path from S to D in $\mathcal{N}$. The word w is then the shortest one generated by the CFG. The derivation tree of w is a tree where the axiom is the root, the internal nodes are nonterminals and the leafs are terminals or the empty word. A nonterminal $[X]$ is the parent of a nonterminal $[Y]$ if the derivation of w contains a production rule where $[X]$ is the left side and $[Y]$ appears in the right side. $[X]$ is the parent of a leaf a in the same way. For example, if the production rule $[X] \to a[Y][Z]$ is used to derive w, then $[X]$ will be the parent of the leaf a and

the nodes $[Y]$ and $[Z]$. Such a tree is a derivation tree of w means that the sequence of leafs of the tree from left to right corresponds exactly to w.

To maximize the size of the derivation tree of w, suppose that it contains only production rules of form 3, except those of the bottom of the tree, which must be in form 1 (otherwise w would contains nonterminals). Furthermore, suppose that in all the production rules of the derivation tree, a is never the empty word. Figure 3 illustrates the shape of such a tree and the way the word w is derived. This is the biggest possible derivation tree for w. The key argument is the following : Each branch in the derivation tree contains each nonterminal of the CFG at most once. If a nonterminal appears twice or more in a branch, one could generate a shorter word by using this nonterminal only once, and this would contradict the fact that w is the shortest word. Thus, the biggest possible derivation tree for w is a complete tree where each internal node is parents of one leaf (terminal) and two internal nodes (nonterminals), except the last node of each branch which is parent of only a leaf. The depth of the tree is at most $|Q|$, and the number of internal nodes is at most $2^{|\mathcal{Q}|} - 1$. Since each internal node gives at most one terminal, then the length of w is at most $2^{|\mathcal{Q}|}-1$. The number of nonterminals in the CFG is at most $|\Gamma| \times |\mathcal{S}|^2 = (\lambda+1)\lambda^2 n^2$ (see Algorithm 2 and its complexity). Thus, the path length is bounded by $2^{(\lambda+1)\lambda^2 n^2} - 1$ □

VI. 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 minimum cost path in a multi-layer network under bandwidth constraint is defined as follows:

Problem 2.

$$\min\ h(\mathcal{P}) = \sum_{(U,f,V)\in\mathcal{P}} h(U,f,V)$$

$$s.t. \begin{cases} \mathcal{P} \text{ is a feasible path between } S \text{ and } D \\ \min_{E\in\mathcal{P}} \dfrac{q_b(E)}{nb(E)} \geq q_b^{min} \end{cases}$$

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-hard as shown by Kuipers and (Freek) Dijkstra [11]. But this proof does not work on symmetric directed graphs. However, most communication networks are symmetric. We show that the decision version of the problem remains NP-hard even with two protocols and in a symmetric graph. Consider the following problem:

Problem (2'). 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 5. *Problem (2') is* NP*-hard with two protocols even if $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is a symmetric directed graph.*

Proof. 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 (a polynomial reduction from the classical Hamiltonian path problem to SYM-HAM is trivial).

Now we provide a polynomial reduction from SYM-HAM to Problem (2') 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 (2'), 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 $i = 1,2,3$. 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, 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 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 $i = 1\ldots,n$, 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.

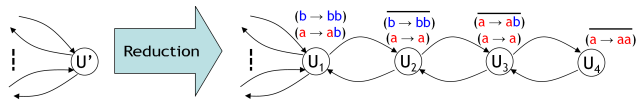


Fig. 4. Reduction from SYM-HAM to feasible path under bandwidth constraint (node splitting).

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:

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

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 an occurence of 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 sequences other than Sequence (2) in the remaining path. There are two possible cases:

- An *incomplete* Sequence (2) where U_4 is not reached (e.g., $U_1fU_2f'U_3f''U_2f'''U_1$): 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 $U_1fV_1f'W_1$: Let $\mathcal{P}$ be a feasible path from S to D containing a sequence $U_1fV_1f'W_1$ (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. □

Remark. In the conference version of this paper [13], Proposition 1 incorrectly states that Problem (2') is NP-complete, and thus being in NP. This claim was based on the supposition that the shortest feasible path length should be polynomial in the size of the network. However, in the light of the results stated in Section V, this claim is false and the shortest path cannot be a polynomial certificate.

Corollary 1. *Problem* (2) *is not approximable.*

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

C. DAG Heuristic

Minimum cost feasible paths involving loops are infrequent (see Section VIII). 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 minimum cost feasible path.

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

1) Set the number 0 to node S and $n-1$ 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.

VII. 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 k additive QoS metrics $(q_1(E), \dots, q_k(E))$ in addition to its available bandwidth $q_b(E)$. These additive metrics can be the delay, logarithm of the packet-loss, etc. Let

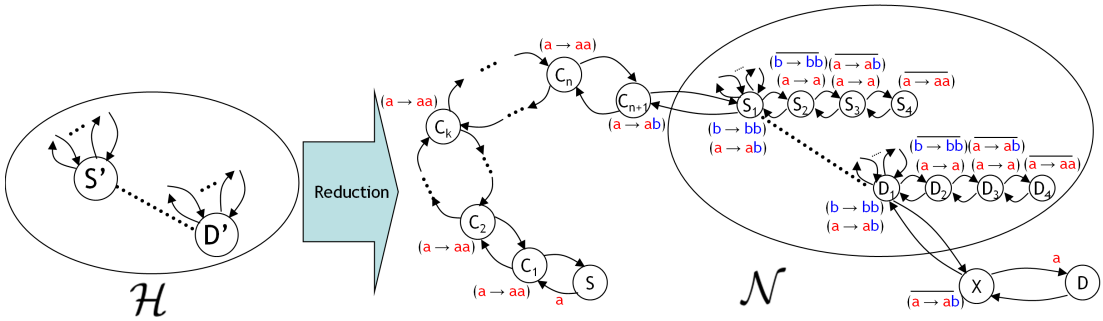


Fig. 5. Reduction from SYM-HAM to feasible path under bandwidth constraint (graph transformation).

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

Problem 3.

$$\min\ h(\mathcal{P}) = \sum_{(U,f,V)\in\mathcal{P}} h(U,f,V)$$

$$s.t. \begin{cases} \mathcal{P} \textit{ is a feasible path from } S \textit{ to } D \\ \min_{E\in\mathcal{P}} \dfrac{q_b(E)}{nb(E)} \geq q_b^{\min} \\ \sum_{E\in\mathcal{P}} \left(q_i(E) \times nb(E)\right) \leq q_i^{\max},\ i = 1 \ldots k \end{cases}$$

The problem of QoS multi-constrained path computation (on a single layer) is well studied and it is well-known that the decision version is NP-complete, even with 2 additive and/or multiplicative constraints [26]. It 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 NP-hard.

B. ML-SAMCRA

Van Mieghem and Kuipers [12] proposed the Self-Adaptive Multiple Constraints Routing Algorithm (SAMCRA) to resolve the problem of multi-constrained path computation on a single layer. It computes paths under several (additive) QoS constraints but it ignores the feasibility constraint as defined in our paper. The algorithm has exponential time worst case complexity but the authors have shown that it exhibits a reasonable processing time in practice. We propose to adapt SAMCRA to the multi-layer context.

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:* 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 only concept that is impacted by the multi-layer context is the non-dominance. It 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 end node. Thus in the algorithm path list, each path should be stored with its protocol stack at its final node. A path can involve the same link several times. Before checking if it 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 and only if:

- $\min_{E\in\mathcal{P}} \frac{q_b(E)}{nb_{\mathcal{P}}(E)} \geq \min_{E\in\mathcal{P}'} \frac{q_b(E)}{nb_{\mathcal{P}'}(E)}$
- $\sum_{E\in\mathcal{P}} q_i(E) \times nb_{\mathcal{P}}(E) \leq \sum_{E\in\mathcal{P}'} q_i(E) \times nb_{\mathcal{P}'}(E)$ $\forall i = 1, \ldots, k$
- $\mathcal{P}$ 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.

VIII. Simulations

A. Path computation without bandwidth constraint

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 [27] are not large enough to show the scaling of our algorithm. Thus, we

performed simulations on two real topologies described in [28] and random scale-free topologies:

- Topology $T1$ is a simplified version of Time Warner network. It has 41 nodes and 296 directed links.
- Topology $T2$ corresponds to the network of Exodus as in 2002. It has 79 nodes and 294 directed links.
- Scale-free topologies are generated according to the Barabsi-Albert model [29]. A complete graph of 10 nodes is created. Then nodes are iteratively added according to a preferential attachment mechanism: A new node is randomly linked to 5 existing nodes. The probability for a node to be selected is proportional to its degree.

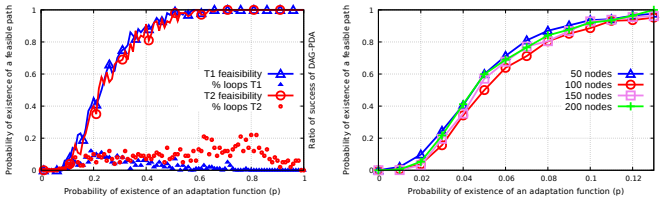
Since these topologies are not layered, the adaptation functions are randomly allocated to the nodes. For a set of protocols $\mathcal{A}$, there are $3\lambda^2$ possible adaptation functions (for each ordered pair of protocols: a conversion, an encapsulation and a decapsulation). For each node U, each 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 minimum cost 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 6a shows the evolution of feasible path existence probability according to p and the proportion of minimum cost 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 minimum cost ones involve loops is high. However, this probability quickly decreases. The phase transition phenomenon can be seen in [21]. But the results consider only loopless paths and deals with technologies rather than adaptation functions.

Figure 6b shows the evolution of feasible path existence probability according to p in a scale-free topology. It seems that the probability is independent from the network size, or varying too slowly depending on it. Whether for 50 or 200 nodes, the probability grows from 0.1% to 99% when $p \in [0.01, 0.15]$. The phase transition is shifted to the left compared to the one observed on Topologies $T1$ and $T2$. This can be explained by the fact that the diameter of a scale-free network is asymptotically $\frac{\log n}{\log \log n}$ [30], which, for the chosen values of n, is much smaller than the diameter of Topologies $T1$ and $T2$.



(a) Real topologies (b) Scale-free topologies

Fig. 6. Probability of existence of a feasible path in real and scale-free topologies according to the probability of existence of an adaptation function.

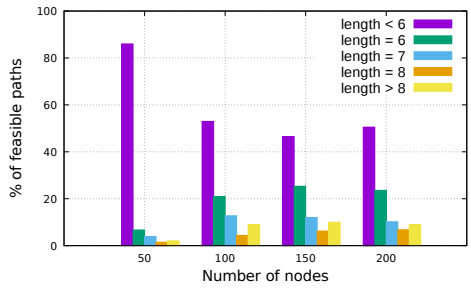


Fig. 7. Feasible path length distribution with $p = 0.05$.

3) Feasible path length distribution

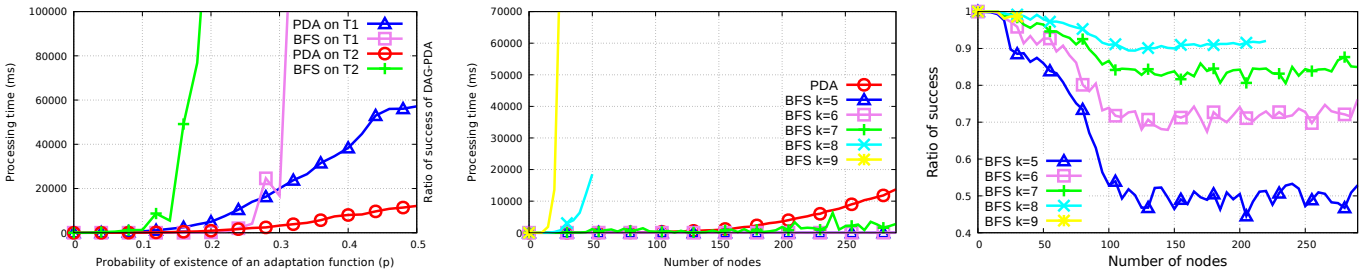
In Section V, we have shown that the shortest feasible path between two nodes may be of superpolynomial length. However, such cases are very specific and are unlikely to appear in practice. We performed simulations on random scale-free topologies, in order to evaluate the effective distribution of feasible path length according to the network size. Figure 7 shows the obtained results averaged over 1000 runs, with $p = 0.05$. For a network of 50 nodes, 86% of the feasible paths have a length ≤ 5, but 2% have a length ≥ 9. For 200 nodes, only 50% of the feasible paths have a length ≤ 5, while 9% have a length ≥ 9. However, all the diameters of the topologies where at most 4. This means that the probability for a shortest feasible path to be more than double the diameter is not negligible.

4) Simulation results on real topologies

Our algorithm is compared to a classical BFS which explores all possible paths until reaching the destination. During the exploration process, all *dominated*[9] paths are deleted. BFS can be seen as a version of the algorithm in [11] where the bandwidth constraint is relaxed. The first results has shown 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 longer considered. We performed 100 runs for each value of p and averaged the processing time.

Figure 8a shows the processing time of PDA and BFS algorithms on Topologies $T1$ and $T2$ according to the values

[9]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.



(a) Processing time (b) Processing time (c) Ration of success of BFS

Fig. 8. Processing time of PDA and BFS on scale free topologies and ratio of success of BFS according to the network size.

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 hugely increases. We cannot put it on Figure 8a 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 10s. On Topology $T1$, for $p = 0.38$, the processing time of BFS is more than 7 minutes, while that of PDA is 7s.

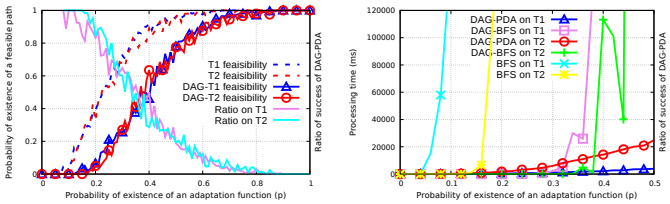
5) Simulation results on random scale-free topologies

Unlike fixed real topologies, random scale-free topologies allow to study the algorithm efficiency according to the network size, which is the goal of this section. As in the previous case, simulating BFS without path length restriction is impossible due to a prohibitive processing time. Instead, we performed simulations of BFS with maximum path length k from 5 to 9. Paths exceeding this length are deleted and no longer considered. Moreover, BFS is only required to find a feasible path, not necessarily the minimum cost one.

Figure 8b shows the processing time of PDA and BFS with different maximum path lengths according to the network size (number of nodes). Figure 8c shows the proportion of feasible paths found by the different BFSs according to the network size. The probability p is fixed to 0.05. All the results are averaged over 1000 runs. The processing time of BFS with $k = 9$ is tremendous and the algorithm is stopped when the network size reaches 30 nodes. For instance, the processing time of the latter is 177s for 30 nodes. For a number of nodes ≥ 100, BFSs with $k = 5$, 6, and 7 are faster than PDA. For instance, when the network size is 160, the processing time of PDA is 1261ms while the processing time of BFS $k = 5$ (resp $k = 6$ and $k = 7$) is 1.23ms (resp. 9.78ms and 1005ms). However, as shown in Figure 8c, when the network contains feasible paths, BFS with $k = 5$ finds only 47% of them. This proportion is 83% for $k = 7$. When the network size grows and k is fixed, this proportion tends to 0% since the probability that a feasible path has length $\leq k$ tends to 0. The advantage of PDA is that it is an exact algorithm, it always finds the optimal path when it exists, while the BFS approach lacks both efficiency and correctness since it can miss existing feasible paths.

B. Path computation under bandwidth constraint

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 VIII-A4, 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. 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.



(a) Phase transition (b) Processng time

Fig. 9. Phase transition (before and after DAG conversion) and comparison of the processing time of DAG-PDA, DAG-BFS and BFS.

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 9a shows that the probability of feasible path existence is shifted to the right after the DAG conversion. The ratio $\frac{\text{Probability of feasible path existence in DAG T}i}{\text{Probability of feasible path existence in T}i}$ $(i = 1, 2)$ 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 9b shows the processing time of DAG-PDA, DAG-BFS and BFS algorithms on both topologies according to the probability of existence of 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 hugely increases. 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).

C. Path computation under several QoS constraints

We studied the efficiency of ML-SAMCRA through simulations and checked if it is as scalable in a multi-layer context as SAMCRA is in a single layer context. The results have shown that for $p > 0.08$ (resp. 0.10) on Topology $T1$ (resp. $T2$) the processing time hugely increases (more than 1 minutes). Clearly, ML-SAMCRA does not scale above these values. The main possible reason is that 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. The second possible reason is that taking into account loops increases the number and the length of the paths, which in turn increases the algorithm complexity. Thus, path computation under QoS constraints in multi-layer networks is more complex than in single layer networks. Exact algorithms are only suitable for small instances.

IX. Conclusion

Most of carrier-grade networks manage their different layers thanks to separate control planes. A unified control plane would optimize the network resources and reduce the operational management costs. 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 have shown that they are faster than the previous approaches in the literature. We also provided lower and upper bounds for the shortest path length. For the second case, we obtained several 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 is 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] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *HotSDN*. ACM, 2013, pp. 127–132.

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[6] 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.

[7] L. Liu *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.

[8] S. Agarwal, M. S. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *INFOCOM*, 2013, pp. 2211–2219.

[9] 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.

[10] M. L. Lamali, H. Pouyllau, and D. Barth, "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] M. L. Lamali, N. Fergani, J. Cohen, and H. Pouyllau, "Path computation in multi-layer networks: Complexity and algorithms," in *INFOCOM*, 2016, pp. 1–9.

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

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

[16] 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.

[17] 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.

[18] 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.

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

[20] 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," *Computer Networks*, vol. 52, no. 10, pp. 1927–1937, 2008.

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

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

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

[24] I. Petre and A. Salomaa, "Algebraic Systems and Pushdown Automata," in *Handbook of Weighted Automata*, M. Droste, W. Kuich, and H. Vogler, Eds. Springer-Verlag Berlin Heidelberg, 2009, ch. 7, pp. 257–290.

[25] 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.

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

[28] 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/

[29] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[30] B. Bollobás and O. Riordan, "The diameter of a scale-free random graph," *Combinatorica*, vol. 24, no. 1, pp. 5–34, 2004.

Acknowledgment

The authors would like to thank Hélia Pouyllau for her participation to the previous version of this work. We also

thank the three referees for their feedback and suggestions. The first author was partially supported by the HÉRA project, funded by The French National Research Agency. Grant no.: ANR-18-CE25-0002.