Path Computation in Multi-layer Multi-domain Networks

Mohamed Lamine Lamali, Hélia Pouyllau, Dominique Barth

Path Computation in Multi-Layer Multi-Domain Networks*

Mohamed Lamine Lamali[1], Hélia Pouyllau[1], and Dominique Barth[2]

[1] Alcatel-Lucent Bell Labs France, Route de Villejust, 91620 Nozay, France
{mohamed_lamine.lamali, helia.pouyllau}@alcatel-lucent.com
[2] Lab. PRiSM, UMR8144, Université de Versailles
45, av. des Etas-Unis, 78035 Versailles Cedex, France
dominique.barth@prism.uvsq.fr

Abstract. Carrier-grade networks have often multiple layers of protocols. To tackle this heterogeneity, the Pseudo-Wire architecture provides encapsulation and decapsulation functions of protocols over Packet-Switched Networks. At the scale of multi-domain networks, computing a path to support an end-to-end service requires various encapsulations and decapsulations that can be nested but for which manual configurations are needed. Graph models are not expressive enough for this problem. In this paper, we propose an approach using graphs and Push-Down Automata (PDA) to capture the compatibility among encapsulations and decapsulations along an inter-domain path. They are respectively modeled as pushes and pops on a PDA's stacks. We provide polynomial algorithms that compute either the shortest path in hops, or in the number of encapsulations and decapsulations to optimize interfaces' configuration.

Keywords: Multi-layer networks, Pseudo-Wire, Push-Down Automata

1 Introduction

Most carrier-grade networks are composed of multiple layers of protocols (e.g. Ethernet, IP, etc.). Such layers are administrated by different control and management plane instances. The Pseudo-Wire (PWE3) architecture [2] is a standard which aims to unify control plane functions in such heterogeneous environments allowing multi-layer services (e.g. Layer 2 VPN). It describes encapsulation (a protocol is encapsulated in another one) and decapsulation functions, also called "adaptation functions", emulating services (e.g. Frame Relay, SDH, Ethernet, etc.) over Packet-Switched Networks (PSN, e.g. IP or MPLS).

The management of these functions is achieved within each carrier's network domain: when an encapsulation function is used, its corresponding decapsulation function is applied within the domain boundaries. In large-scale carrier-grade

* This work is partially supported by the ETICS-project, funded by the European Commission. Grant agreement no.: FP7-248567 Contract Number: INFSO-ICT-248567.

networks or in multi-carrier networks, restricting the management of adaptation functions to network boundaries might lead to ignore feasible end-to-end paths and thus to refuse service demands. Hence, the path computation process that precludes the signaling phase must encompass the notion of *encapsulation/decapsulation compatibility*: when a protocol is encapsulated into another at one node, it must be decapsulated at another, and the possibility to nest such functions (e.g. Ethernet over MPLS over SDH). Furthermore, as such function are manually configured on routers' interfaces, minimizing their number would simplify the signaling phase.

The authors of [8, 4] focused on the problem of computing a path in a multi-layer network under bandwidth constraints. In [10], we demonstrated that the problem under multiple Quality of Service constraints is NP-Complete. As a first step in our research, we focus on the problem of finding a path across multiple domains involving compatible - possibly nested - adaptation functions.

In this paper, we demonstrate that this reduced problem can be solved by a polynomial-time algorithm. We consider as an objective function, either the number of adaptation functions to simplify the signaling or the number of nodes to minimize the cost of a path. The proposed approach combines both graph and automata theory: the encapsulation and decapsulation functions are designed as pushes and pops in a Push-Down Automaton (PDA) respectively. To determine the shortest path in adaptation functions, such a PDA is transformed to bypass path segments without such functions. The PDA or transformed PDA is then converted into a Context-Free Grammar (CFG) using the method of [7]. A shortest word, either corresponding to the shortest path in nodes or in adaptation functions, is generated from this CFG.

This paper is organized as follows: sec. 2 recalls the context of multi-layer multi-domain networks and the related work on path computation; sec. 3 provides a formal definition of the problem; sec 4 explains the transformation from a network to a PDA; finally, sec. 5 details the different algorithms computing the shortest path in nodes or in encapsulations/decapsulations.

2 Path computation in Pseudo-Wire networks

In order to mitigate multi-layer issues, some standards define the emulation of lower layer connection-oriented protocols over a PSN (e.g. Ethernet over MPLS, [12]). For instance, the layer 2 frames are encapsulated in layer 3 packets at one network node and decapsulated at another, bursting the OSI model.

The PWE3 architecture [2] assumes an exhaustive knowledge of the network states. This assumption is also used in the multi-layer networking description of [13] and is not valid in a multi-domain context. This issue has been identified by the IETF PWE3 working group. The authors of [1] defined the multi-segment Pseudo-Wire architecture for multi-domain networks. The authors of [3] mention the problem of path determination and suggest the use of the Path Computation Element architecture (PCE) [5], which is adapted to the multi-domain context, to figure it out. It could be a control plane container for the

approach described in this paper, requiring some protocol extensions to add encapsulation/decapsulation capabilities in the coding of the data model used by PCEs.

Related work on path computation. In [4], the authors addressed the problem of computing the shortest path in the context of the ITU-T G.805 recommendations on adaptation functions. They stress the lack of solutions on path selection and the limitations of graph theory to handle this problem. In [8], the authors addressed the same problem in a multi-layer network qualifying it as an NP-Complete problem. The NP-Completeness comes from the problem definition as they allow loops across layers but under a limited bandwidth. They aim to select the shortest path in nodes and provide an exponential-time algorithm accordingly. The model used in [8] is close to a PDA.

In the problem we consider, we exclude bandwidth constraints as the PCE architecture handles them already and propose a solution for minimizing the number of encapsulations and decapsulations. Our algorithm does not allow loops without adaptation functions, the only loops that may exist involve encapsulations or decapsulations. Thus, minimizing the number of adaptation functions in the path also leads to minimize the number of loops - and to avoid them if a loop-free feasible path with less encapsulations exists. Our contribution is a generalization based on graph and automata theory providing further theoretical assets and a different modeling leading to a polynomial-time algorithm.

Proposed approach. To the best of our knowledge no previous work has considered this problem at the multi-domain scale. It induces to go further domain boundaries allowing *multi-domain compatibility* to determine a *feasible inter-domain path*: when an encapsulation for a given protocol is realized in one domain its corresponding decapsulation must be done in another. Furthermore, we consider two kind of objectives: either the well-known objective of minimizing the number of nodes or the objective of minimizing the number of adaptation functions. This latter is motivated by the fact that it is equivalent to minimize the number of configuration operations, which are often done manually and can be quite complex. To express encapsulations and decapsulations, the network model is converted into a PDA as the stack allows memorizing encapsulations. Hence, our approach encompasses the two shortest path problems either in nodes or in adaptation functions:

1. Convert a multi-domain Pseudo-Wire network into a PDA,
   (a) If the goal is to minimize the number of adaptation functions, transform the PDA to bypass the "passive" functions (i.e. no protocol adaptation),
   (b) else let the PDA as is,
2. Derive a CFG from the PDA or the transformed PDA,
3. Determine the "shortest" word generated by the CFG and
4. Identify the shortest path from the shortest word.

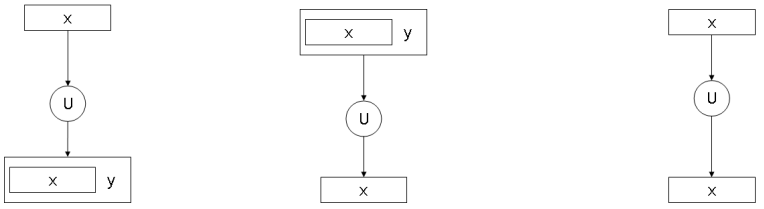
3 Heterogeneous multi-domain network model

A multi-domain network having routers with encapsulation/decapsulation capabilities can be defined as a 3-tuple: a directed graph $G = (\mathcal{V}, E)$ modeling the

routers of a multi-domain network, we consider a pair of vertices (S, D) in G corresponding to the source and the destination of the path we focus on; a finite alphabet $\mathcal{A} = \{a, b, c, \dots\}$ in which each letter is a protocol; an encapsulation or a decapsulation function is a pair of different letters in the alphabet $\mathcal{A}$:

- Figure 1(a) illustrates the encapsulation of the protocol x by the node U in the protocol y;
- Figure 1(b) illustrates that the protocol x is unwrapped by the node U from the protocol y;
- Figure 1(c) illustrates that the protocol x transparently crosses the node U (no encapsulation or decapsulation function is applied). Such pairs are referred as *passive* further in this paper.

We denote by $\mathcal{ED}$ and by $\overline{\mathcal{ED}}$ the set of all possible encapsulation functions and decapsulation functions respectively, (i.e., $\mathcal{ED} \subseteq \mathcal{A}^2$). A subset $P(U)$ of $\mathcal{ED} \cup \overline{\mathcal{ED}}$ indicates the set of encapsulation, passive and decapsulation functions supported by vertex $U \in \mathcal{V}$. We define $In(U) = \{a \in \mathcal{A}\ s.t.\ \exists b \in \mathcal{A}\ s.t.\ (a, b)$ or $\overline{(b, a)} \in P(U)\}$ and $Out(U) = \{b \in \mathcal{A}\ s.t.\ \exists a \in \mathcal{A}\ s.t.\ (a, b)$ or $\overline{(b, a)} \in P(U)\}$. The set $\mathcal{A}(U) = \{(a, a) \in P(U)\}$ is the set of protocols that can passively cross node U.



(a) Encapsulation of protocol x in protocol y, $(x, y) \in P(U)$

(b) Decapsulation of protocol x from protocol y, $\overline{(x, y)} \in P(U)$

(c) Passive crossing, $(x, x) \in P(U)$

Fig. 1. Different transitions when a protocol crosses a node U

Considering a network $N = (G = (\mathcal{V}, E), \mathcal{A}, P)$, a source $S \in \mathcal{V}$, a destination $D \in \mathcal{V}$ and a path $C = S, x^1, U_1, x^2, \dots, U_{n-1}, x^n, D$ where each U_i is a vertex in $\mathcal{V}$ and each $x^i \in \mathcal{A} \cup \overline{\mathcal{A}}$ (where $\overline{\mathcal{A}} = \{\overline{a} : a \in \mathcal{A}\}$).

- $T_C = x^1 \dots x^n$ represents the sequence of protocols which is used over the path C. It is called the *trace* of C. For each x^i:
  - $x^i = a$ and $x^{i+1} = b$ or $\overline{b}$, means that U_i encapsulates protocol a in b ($a, b, \overline{b} \in \mathcal{A} \cup \overline{\mathcal{A}}$)
  - $x^i = \overline{a}$ and $x^{i+1} = b$ or $\overline{b}$ means that U_i unwraps protocol b from a
  - $x^i = a$ and $x^{i+1} = a$ or $\overline{a}$ means that U_i passively transports a
- We denote H_C the sequence $\beta_1, \dots, \beta_n$ obtained from C s.t. for $i = 1..n$:
  - if $x^i = a \in \mathcal{A}$ and $x^{i+1} = b \in \mathcal{A}$ or $x^{i+1} = \overline{b} \in \overline{\mathcal{A}}$ then $\beta_i = (a, b)$
  - if $x^i = \overline{b} \in \overline{\mathcal{A}}$ and $x^{i+1} = a \in \mathcal{A} \backslash \{b\}$ or $x^{i+1} = \overline{a} \in \overline{\mathcal{A}} \backslash \{\overline{b}\}$ then $\beta_i = \overline{(a, b)}$

– We also denote $M_C = \beta'_1, \ldots, \beta'_m$ obtained from H_C by deleting each passive transition β_i s.t. $\beta_i = (a, a)$ and $a \in \mathcal{A}$

Example. Consider the path $C = S, a, U, b, V, \overline{b}, W, a, D$ in the network illustrated by Fig. 2(a). The sequence corresponding to C is $H_C = (a, b), (b, b), \overline{(a, b)}$ and its trace is $T_C = ab\overline{b}a$. Finally, the well parenthesized sequence from C is $M_C = (a, b), \overline{(a, b)}$.

Let ϵ denotes the empty word, and H_C a sequence obtained from a path C as explained above. The following definitions give a formal characterization of the feasible paths.

Definition 1. *A sequence M_C from H_C is* valid *if and only if $M_C \in \mathcal{L}$ where $\mathcal{L}$ is the formal language recursively defined by: $\mathcal{L} = \epsilon \cup ((a, b).\mathcal{L}.\overline{(a, b)}).\mathcal{L}$ for each $(a, b) \in \mathcal{A}^2$.*

Definition 2. *A path C is a* feasible path *in N from S to D if:*

- *$U_1, \ldots, U_{n-1}$ is a path in G*
- *the sequence M_C from C is valid*
- *for each $i = 1..n$:*
  - *if $x^i = a$ and $x^{i+1} = b$ or $\overline{b}$ then $(a, b) \in P(U_i)$*
  - *if $x^i = \overline{a}$ and $x^{i+1} = b$ or $\overline{b}$ then $\overline{(a, b)} \in P(U_i)$*
  - *if $x^i = a$ and $x^{i+1} = a$ or $\overline{a}$ then $a \in \mathcal{A}(U_i)$*

The language $\mathcal{L}$ of valid sequences is known as the *generalized Dyck language* [11]. It is well-known that this language is context free but not regular. Thus, push-down automata are naturally adapted to model this problem.

Example. The multi-domain network illustrated by Fig. 2(a) has 6 routers and two protocols labeled a and b. Adaptation function capabilities are indicated below each node. In this multi-domain network, the only feasible path between S and D is $S, a, U, b, V, \overline{b}, W, a, D$ and involves functions (a, b) on U, (b, b) on V, and $\overline{(a, b)}$ on node W.

Problem definition. As explained above, our goal is to find a feasible multi-domain path. Furthermore, we set as an objective function either the size of the sequence of adaptation functions or the size of the path in number of nodes. Hence, the problem we aim to solve can be defined as follows,

$$\min_C |H_C| \text{ or } |M_C| \text{ s.t. } C \text{ is a feasible path}$$

4 From the network model to a PDA

In this section, we address the conversion from a network to a PDA. The **algorithm 1** takes as input a network $N = (G = (\mathcal{V}, E), \mathcal{A}, P)$ and converts it to a PDA $A_N = (\mathcal{Q}, \Sigma, \Gamma, \delta, S, \emptyset, \{D\})$, where $\mathcal{Q}$ is the set of states of the PDA, Σ the input alphabet, Γ the stack alphabet, δ the transition relation, S the start state, Z_0 the initial stack symbol and D the accepting state, ϵ is the empty string. The automaton A_N from N is obtained as follows:

- Create a state U_x of the automaton for each $U \in \mathcal{V}$ and each $x \in In(U)$, except for S for which create only one state,
- An encapsulation of a protocol x in a protocol y by a node U_x is modeled as a push of the character x in the stack between the state U_x and its successor V_y. It is denoted $(U_x, < x, \alpha, x\alpha >, V_y)$[3] ,
- A decapsulation of y from x by the node U_x is modeled as a pop of the protocol y from the stack. it is denoted $(U_x, < \overline{x}, y, \emptyset >, V_y)$,
- The top of the stack is the last encapsulated protocol,
- If the current state is U_x then the current protocol is x.

Example. Figure 2(b) is an example of output of the algorithm 1. The algorithm transforms the network illustrated by Fig. 2(a) into a PDA of Fig 2(b). For instance, the link (U, V) is transformed into the transitions $(U_a, < a, Z_0, aZ_0 >, V_b)$ and $(U_a, < a, b, ab >, V_b)$ w.r.t. adaptation capabilities of U and V.

Algorithm 1 Convert a network to a PDA

Input: a network $N = (G = (\mathcal{V}, E), \mathcal{A}, P)$, a source S and a destination D
Output: push-down automaton $A_N = (\mathcal{Q}, \Sigma, \Gamma, \delta, S, \emptyset, \{D\})$
(1) $\Sigma \leftarrow \mathcal{A} \cup \overline{\mathcal{A}}$; $\Gamma \leftarrow \mathcal{A} \cup \{Z_0\}$
(2) Create a single state for the node S
(3) For each node $U \neq S$ and each protocol $x \in IN(U)$, create a state U_x
(4) For each state U_x s.t. $(S, U) \in E$ and $x \in Out(S)$
    Create a transition $(S, < \epsilon, Z_0, Z_0 >, U_x)$
(5) For each link $(U, V) \in E$ s.t. $U \neq S$ and for each $(x, y) \in P(U)$ and each $\alpha \in \Gamma \backslash \{x\}$
  (5.1) If $x \in \mathcal{A}(U) \cap In(V)$ Create a transition $(U_x, < x, \alpha, \alpha >, V_x)$ {passive trans.}
  (5.2) If $x \neq y$ and $y \in In(V)$ Create a transition $(U_x, < x, \alpha, x\alpha >, V_x)$ {encap.}
(6) For each link $(U, V) \in E$ s.t. $U \neq S$ and for each $\overline{(x, y)} \in P(U)$
  (6.1) If $x \in In(V)$ Create a transition $(U_y, < \overline{y}, x, \emptyset >, V_x)$ {decap.}
(7) Create a fictitious final state D.
(8) For each $x \in In(D)$ and each $\alpha \in \Gamma \backslash \{x\}$ Create a transition $(D_x, < x, Z_0, \emptyset >, D)$

Complexity. Each node U from the graph generates $|In(U)|$ states, except the source node S. A fictitious final state is added. Thus, the number of states is at worst $2 + (|\mathcal{V}| - 1) \times |\mathcal{A}|$ so in $O(|\mathcal{V}| \times |\mathcal{A}|)$. The worst case complexity of **algorithm 1** is in $O(\max((|\mathcal{V}| \times |\mathcal{A}|), (|E| \times ((|\mathcal{A}| \times |\mathcal{ED}|) + |\overline{\mathcal{ED}}|))))$. We assume that the network is connected, so $|E| \geq |\mathcal{V}| - 1$. Since $\mathcal{ED}$ is a subset of $\mathcal{A}^2$, then $|\mathcal{ED}| < |\mathcal{A}|^2$ and $|\overline{\mathcal{ED}}| < |\mathcal{A}|^2$. Thus, the upper bound complexity is in $O(|E| \times |\mathcal{A}|^3)$, which is also an upper bound for the number of transitions.

Proposition 1. *Considering a network $N = (G = (\mathcal{V}, E), \mathcal{A}, P)$, a source $S \in \mathcal{V}$ and a destination $D \in \mathcal{V}$, the language recognized by A_N is the set of traces of* the feasible paths *from S to D in N.*

[3] Note that, even if $x = \overline{a} \in \overline{\mathcal{A}}$, the transition has the form $(U_a, < \overline{a}, \alpha, a\alpha >, V_y)$. Characters in $\overline{\mathcal{A}}$ are only used as input characters. Characters indexing states and pushed characters in the stack are their equivalent in $\mathcal{A}$.

Sketch of Proof. The main idea is to show that a path C is feasible if and only if its trace T_C is accepted by the PDA. Thus, for each feasible path there is a sequence of transitions in the PDA that accepts its trace. And for each accepted word, the sequence of transitions, which accepts this word, corresponds to a feasible path. The complete proof is available in the Appendix online [9].

5 The shortest feasible path

In sec. 4, we provided the method to build a PDA allowing to determine the feasible paths. The next step is to minimize either the number of nodes or the number of adaptation functions. The method to minimize the number of nodes uses directly the PDA as described in sec. 5.1. But, to minimize the number of adaptation functions, as detailed in sec. 5.2, the PDA is transformed in order to bypass the sub-paths without any adaptation function. Then, a CFG derived from the PDA (or the transformed PDA) generates words whose length is equivalent to the number of nodes (or to the number of adaptations). An algorithm browses the CFG to determine the shortest word. Finally, another algorithm identifies the multi-domain path corresponding to this shortest word.

5.1 Minimizing the number of nodes

The number of characters in a word accepted by the automaton A_N is the number of links in the corresponding feasible path (each character is a protocol used on a link). Thus the step of automaton transformation (sec. 5.2) should be skipped. The automaton A_N is directly transformed into a CFG, then the shortest word is generated as described in sec. 5.3. The corresponding feasible path is computed by the algorithm 4 described in sec. 5.3.

5.2 Minimizing the number of adaptation functions

To enumerate only encapsulations and decapsulations in the length of each word (and thus minimize adaptation functions by finding the shortest word accepted), a transformed automaton A'_N in which all sequences involving passive transitions are bypassed must be determined. The length of the shortest word accepted by A'_N is the number of adaptation functions plus a fixed constant.

Let define $\mathcal{Q}_a$ ($a \in \mathcal{A}$) as $\mathcal{Q}_a = \{V_x \in \mathcal{Q} | x = a\}$, and let A^a_N be the *sub-automaton* induced by $\mathcal{Q}_a$. By analogy with an induced subgraph, an induced sub-automaton is a multigraph with labeled edges such that the set of vertex is $\mathcal{Q}_a$ and the set of edges is the set of transitions between elements of $\mathcal{Q}_a$. Since there are only passive transitions between two states in $\mathcal{Q}_a$, all paths in the sub-automaton are passive. Let define $P(U_x, V_x)$ as the shortest path length between U_x and V_x. This length can be computed by any well-known Shortest Path Algorithm. Let $Succ(V_x)$ be the set of successors of V_x in the original automaton A_N, i.e., $Succ(V_x) = \{W_y \in \mathcal{Q} | \exists (V_x, < x, \alpha, \beta >, W_y) \in \delta\}$.

The algorithm 2 takes as input A_N and computes the transformed automaton $A'_N = (\mathcal{Q}', \Sigma', \Gamma', \delta', S', \emptyset, \{D'\})$. A'_N is initialized with the values of A_N. Then, the algorithm computes the sub-automaton for each character $x \in \mathcal{A}$ (step (1)) and the length values $P(U_x, V_x)$ for each pair of states in the sub-automaton (step (2.1)). Each path between a pair of states is a sequence of passive transitions. If such a path exists (step (2.2)), the algorithm adds transitions to δ' from U_x to each state in $Succ(V_x)$ (steps (2.2.2) and (2.2.3)). These added transitions are the same that those which connect V_x to its successors W_y, but with an input character indexed by the number of passive transitions between U_x and V_x, (i.e., $P(U_x, V_x)$) plus one (indicating that there is a transition sequence which matches with a sequence of protocols $xx \dots x$ of length $P(U_x, V_x) + 1$). The indexed character is added to the input alphabet Σ' (step (2.2.1)).
Example. The algorithm 2 transforms the PDA in Fig. 2(b) into the PDA in Fig. 2(c). the transition $(V_b, < \overline{b_2}, a, \emptyset >, D_a)$ is added to bypass the sequence of transitions $(V_b, < b, a, a >, W_b)$ $(W_b, < \overline{b}, a, \emptyset >, D_a)$.

Algorithm 2 Transform automaton A_N

Input: push-down automaton $A_N = (\mathcal{Q}, \Sigma, \Gamma, \delta, S, \emptyset, \{D\})$
Output: transformed push-down automaton $A'_N = (\mathcal{Q}', \Sigma', \Gamma', \delta', S', \emptyset, \{D'\})$
$\mathcal{Q}' \leftarrow \mathcal{Q}$, $\Sigma' \leftarrow \Sigma$, $\Gamma' \leftarrow \Gamma$, $\delta' \leftarrow \delta$, $S' \leftarrow S$, $D' \leftarrow D$
For each $x \in \mathcal{A}$
  (1) Compute A_N^x
  (2) For each $U_x \in \mathcal{Q}_x$ and each $V_x \in \mathcal{Q}_x$ s.t. $U_x \neq V_x$
    (2.1) Compute $P(U_x, V_x)$
    (2.2) If $P(U_x, V_x) < \infty$ {there is a path between U_x and V_x}
      (2.2.1) Add $x_{P(U_x,V_x)+1}$ and $\overline{x}_{P(U_x,V_x)+1}$ to Σ
      (2.2.2) For each $W_y \in Succ(V_x) \backslash \{U_x\}$ and each $(V_x, < x, \alpha, \beta >, W_y) \in \delta$
        Add the transition $(U_x, < x_{P(U_x,V_x)+1}, \alpha, \beta >, W_y)$ to δ'
      (2.2.3) For each $W_y \in Succ(V_x) \backslash \{U_x\}$ and each $(V_x, < \overline{x}, \alpha, \beta >, W_y) \in \delta$
        Add the transition $(U_x, < \overline{x}_{P(U_x,V_x)+1}, \alpha, \beta >, W_y)$ to δ'

Complexity. Steps (2.2.2) and (2.2.3) are bounded by $O(|\mathcal{Q}| \times |\delta|)$. Step (2.1) (computing the shortest path) is bounded by $O(|\mathcal{Q}_x|^2)$. Since the automaton is connected, $|\mathcal{Q}_x| \leq |\mathcal{Q}| \leq |\delta| + 1$. Thus, the complexity of each iteration in the loop *For* (step (2)) is bounded by $O(|\mathcal{Q}| \times |\delta|)$. The number of iterations of step (2) is in $O(|\mathcal{Q}_x|^2)$. However, a state cannot belong to two different sub-automata. The complexity of the algorithm 2 is in $O(\sum_{x \in \mathcal{A}} |\mathcal{Q}_x|^2 \times |\mathcal{Q}| \times |\delta|)$, which is maximized when $\exists x \in \mathcal{A}$ s.t. $|\mathcal{Q}_x| = |\mathcal{Q}|$ and $|\mathcal{Q}_{x'}| = 0$ for each $x' \neq x$. Thus, the complexity of algorithm 2 is in $O(|\mathcal{Q}|^3 \times |\delta|)$. In the network model, this is equivalent to $O(|\mathcal{A}|^6 \times |\mathcal{V}|^3 \times |E|)$.

Let $L(A_N)$ be the set of words accepted by A_N, and let $L(A'_N)$ be the set of words accepted by A'_N. Let $f : \Sigma' \rightarrow \Sigma^*$ be a function s.t.:

– if $x_i = a_i \in \mathcal{A}'$ then $f(x_i) = \underbrace{aa \dots aa}_{i \text{ occurrences}}$

– if $x_i = \overline{a_i} \in \overline{\mathcal{A}'}$ then $f(x_i) = \underbrace{aa\ldots a\overline{a}}_{i \text{ occurrences}}$

The domain of f is extended to $(\Sigma')^*$:

$$f : (\Sigma')^* \to \Sigma^* \text{ s.t. } w' = x_i^1 x_j^2 \ldots x_k^n \to f(w') = f(x_i^1) f(x_j^2) \ldots f(x_k^n)$$

For simplicity, we consider that x and x_1 are the same character. $f(L(A'_N))$ denotes the set of words accepted by A'_N transformed by f (i.e. $f(L(A'_N)) = \{f(w') \; s.t. \; w' \in L(A'_N)\}$).

It is clear that f is not a bijection ($f(x_i x_j) = f(x_{i+j})$). So to operate the transformation between $L(A_N)$ and $L(A'_N)$, we define $g : \Sigma^* \to (\Sigma')^* \; s.t.$: for each $w = \underbrace{xx\ldots x}_{i \text{ occurrences}} \underbrace{yy\ldots y}_{j \text{ occurrences}} \ldots \underbrace{zz\ldots z}_{k \text{ occurrences}} \in \Sigma^*$, $g(w) = x_i y_j \ldots z_k$.
In other words, $w' = g(w)$ is the shortest word in $(\Sigma')^*$ s.t. $f(w') = w$.

The following proposition shows the relation between the number of encapsulations in A_N and the size of a word accepted by A'_N.

Proposition 2. *The word accepted by A_N which minimizes the number of pushes is $f(w')$, where w' is the shortest word (i.e., with minimal number of characters) accepted by A'_N.*

Sketch of Proof. There is a linear relation between the length of a word accepted by $L(A'_N)$ and the number of pushes and pops involved to accept it, and the same number of pushes and pops is involved to accept w' (in A'_N) and $f(w')$ (in A_N). The complete proof is available in [9].

5.3 The shortest path as a shortest word

In order to find the shortest word accepted by A_N (resp. A'_N), the CFG G_N such that $L(G_N) = L(A_N)$ (resp. $L(A'_N)$) is computed. Backtracking from terminals to the start symbol, the shortest sequence of derivations is then computed. **From the PDA to the CFG.** The transformation of a PDA into a CFG is well-known. We adapted a general method described in [7] to transform A_N(resp. A'_N) into a CFG $G_N = (\mathcal{N}, \Sigma, S_G, \mathcal{P})$ (resp.$(\mathcal{N}, \Sigma', S_G, \mathcal{P})$) where $\mathcal{N}$ is the set of nonterminals (variables), Σ (resp. Σ') is the input alphabet, S_G is the initial symbol (initial nonterminal) and $\mathcal{P}$ is the set of production rules. Nonterminals are in the form $[UXV]$ where $U, V \in \mathcal{Q}$ (resp. $\mathcal{Q}'$) and $X \in \Gamma$ (resp. Γ').The worst case complexity of this algorithm is in $O(|\delta| \times |\mathcal{Q}|^2)$ (resp. $O(|\delta'| \times |\mathcal{Q}'|^2)$). W.r.t. the definition of the network, the upper bound is in $O(|\mathcal{A}|^5 \times |\mathcal{V}|^2 \times |E|)$ and the number of production rules in the wost case is $1 + |\mathcal{Q}| + (|\delta| \times |\mathcal{Q}|^2)$ (resp. $O(|\mathcal{A}|^8 \times |\mathcal{V}|^5 \times |E|^2)$ and $1 + |\mathcal{Q}'| + (|\delta'| \times |\mathcal{Q}'|^2)$). The number of nonterminals is bounded by $O(|\mathcal{Q}|^2 \times |\mathcal{A}|) = O(|\mathcal{A}|^3 \times |V|^2)$ (resp. $O(|\mathcal{Q}'|^2 \times |\mathcal{A}|) = O(|\mathcal{A}|^3 \times |V|^2)$). The interested reader can refer to [9] for the detailed algorithm.
Example. This method transforms the PDA in Fig. 2(c) into a CFG. Figure 2(d) is a subset of production rules of the obtained CFG. This subset allows generating the shortest trace of a feasible path in the network in Fig. 2(a).

The shortest word generated by a CFG. To find the shortest word generated by G_N, a function ℓ associates to each nonterminal the length of the shortest word that it generates.

More formally, $\ell : \{\mathcal{N} \cup \Sigma \cup \{\epsilon\}\}^*$or $\{\mathcal{N} \cup \Sigma' \cup \{\epsilon\}\}^* \to \mathbb{N} \cup \infty$ s.t.:
- if $w = \epsilon$ then $\ell(w) = 0$,
- if $w \in \Sigma$ or Σ' then $\ell(w) = 1$,
- if $w = \alpha_1 \dots \alpha_n$ (with $\alpha_i \in \{\mathcal{N} \cup \Sigma \cup \{\epsilon\}\}$ or $\{\mathcal{N} \cup \Sigma' \cup \{\epsilon\}\}$) then $\ell(w) = \sum_{i=1}^{n} \ell(\alpha_i)$.

The algorithm 3 computes the value of $\ell([x])$ for each $[x] \in \mathcal{N}$.

Algorithm 3 Compute the values $\ell([x])$ for each nonterminal $[x] \in \mathcal{N}$

Input: $G_N = (\mathcal{N}, \Sigma, S_G, \mathcal{P})$ or $(\mathcal{N}, \Sigma', S_G, \mathcal{P})$
Output: $\ell([x])$ for each nonterminal $[x]$
(1) Initialize each $\ell([x])$ to ∞
(2) While there is at least one $\ell([x])$ updated at the previous iteration do
  (2.1) For each production rule $[x] \to \alpha_1 \dots \alpha_n$ in $\mathcal{P}$
    (2.1.1) $\ell([x]) \leftarrow \min\{\ell([x]), \sum_{i=1}^{n} \ell(\alpha_i)\}$

Proposition 3. *The algorithm 3 terminates at worst after $|\mathcal{N}|$ iterations, and each $\ell([x])$ ($[x] \in \mathcal{N}$) obtained is the length of the shortest word produced by $[x]$.*

Sketch of Proof. During each iteration, at least one $\ell([x])$ is updated to its correct value, until all values are correct. The complete proof is available in [9].

Complexity. The complexity of the algorithm 3 is in $O(|\mathcal{N}| \times |\mathcal{P}|)$ which is $O(|\mathcal{A}|^8 \times |V|^4 \times |E|)$ (resp. $O(|\mathcal{A}|^{11} \times |V|^7 \times |E|^2)$) in the network model.

There are several algorithms which allow generating a random word of some length from a CFG. The *boustrophedonic* and the *sequential* algorithms described in [6] generate a random labeled combinatorial object of some length from any decomposable structure (including CFGs). The boustrophedonic algorithm is in $O(n \log n)$ (where n is the length of the object) and the sequential algorithm is in $O(n^2)$ but may have a better average complexity. Both algorithms use a precomputed table of linear size. This table can be computed in $O(n^2)$. These algorithms require an unambiguous CFG, but this requirement is only for the randomness of the generation. Recall that our goal is to generate the trace of the shortest feasible path. Thus, we do not take into consideration the randomness and the distribution over the set of shortest traces.

In order to generate the shortest word in $L(G_N)$, the boustrophedonic algorithm uses G_N and $\ell(S_G)$ as input ($\ell(S_G)$ is the length of the shortest word generated by G_N). Thus, the generation of the shortest word w (resp. w') is in $O(|w|^2)$ (resp. $O(|w'|^2)$) including the precomputation of the table.

Example. The algorithm 3 gives $\ell(S_G) = 3$. The boustrophedonic algorithm computes the shortest word with the production rules in Fig. 2(d). The derivation is: $S_G \overset{(1)}{\vdash} [SZ_0D] \overset{(2)}{\vdash} [U_aZ_0D] \overset{(3)}{\vdash} a[V_bZ_0D_a][D_aZ_0D] \overset{(4)}{\vdash} a\bar{b}_2[D_aZ_0D] \overset{(5)}{\vdash} a\bar{b}_2a$

Thus, the shortest word accepted by the transformed PDA is $a\bar{b}_2a$. And the shortest trace of a feasible path is $f(a\bar{b}_2a) = ab\bar{b}a$.

From the shortest word to the path. If the goal is to minimize the number of nodes in the path, the algorithm 4 takes as input the shortest word w accepted by A_N. Otherwise, as w' is the shortest word accepted by A'_N and generated by G_N, by prop. 2, $f(w')$ is the word which minimizes the number of pops and pushes in A_N. In such a case it is the trace T_C of the shortest feasible path C in the network N. It is possible that several paths match with the trace $T_C = w$ (resp. $f(w')$). In such a case, a load-balancing policy can choose a path.

The algorithm 4 is a dynamic programming algorithm that computes C. It starts at the node S and takes at each step all the links in E which match with the current character in T_C. Let $T_C = x^1x^2 \dots x^n$ ($x^i \in \mathcal{A} \cup \overline{\mathcal{A}}$). At each step i, the algorithm starts from each node U in $Nodes[i]$ and adds to $Links[i]$ all links (U, V) which match with x^i. It also adds each V in $Nodes[i+1]$. When reaching D, it backtracks to S and selects the links from D to S.

Example. From the shortest trace $ab\bar{b}a$, the algorithm 4 computes the only feasible path in the network on Fig. 2(a), which is $S, a, U, b, V, \bar{b}, W, a, D$.

Algorithm 4 Find the shortest path

Input: Network N and T_C
Output: Shortest path C
(1) $Nodes[1] \leftarrow S$; $i \leftarrow 1$
(2) While D is not reached do
  (2.1) for each $U \in Nodes[i]$ and each $V \in \mathcal{V}$ *s.t.* $(U, V) \in E$ do
    (2.1.1) If $x^i \in \mathcal{A}$, $x^i \in Out(U)$, $x^i \in In(V)$ and $(x^{i-1}, x^i) \in P(U)$
      (2.1.1.1) Add (U, V) to $Links[i]$ and V to $Nodes[i+1]$
    (2.1.2) If $x^i \in \overline{\mathcal{A}}$, $x^i \in Out(U)$, $x^i \in In(V)$ and $\overline{(x^i, x^{i-1})} \in P(U)$
      (2.1.2.1) Add (U, V) to $Links[i]$ and V to $Nodes[i+1]$
  (2.2) $i++$
(3) Backtrack from D to S by adding each covered link in the backtracking to C.

Complexity. The *while* loop stops exactly after T_C steps, because it is sure that there is a feasible path of length $|T_C|$ if T_C is accepted by the automaton A_N. At each step, all links and nodes are checked in the worst case. Thus, the algorithm 4 is in $O(|T_C| \times |\mathcal{V}| \times |E|)$ in the worst case.

6 Conclusion

In this paper, we presented a new model for heterogeneous networks involving automata theory to provide the shortest path in number in nodes or in encapsulations/decapsulations. The proposed solution can be applied over the Pseudo-Wire architecture [2], but also over other network architectures which involve adaptation functions and protocol changes. The different algorithms of our methodology have polynomial upper-bounds as summarized by Table 1. These

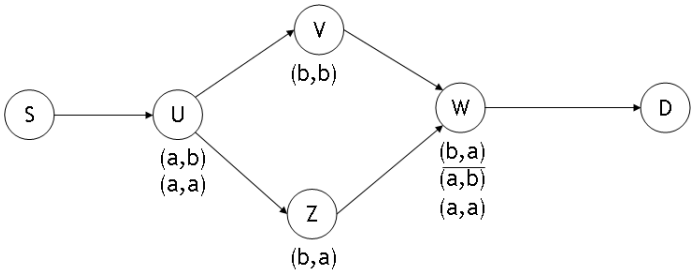
worst-case upper-bounds can be quite high but correspond to unrealistic situations. In future work, we aim to extend our model to support QoS constraints and to define a distributed solution.

Step	Algorithm	Upper-Bound Complexity	
		Minimizing hops	Minimizing enc./dec.
Network to PDA	Algorithm 1	$O(\|E\| \times \|\mathcal{A}\|^3)$	$O(\|E\| \times \|\mathcal{A}\|^3)$
PDA to trans. PDA	Algorithm 2	/	$O(\|\mathcal{A}\|^6 \times \|\mathcal{V}\|^3 \times \|E\|)$
Trans. PDA to CFG	Algorithm 5[9]	$O(\|\mathcal{A}\|^5 \times \|\mathcal{V}\|^2 \times \|E\|)$	$O(\|\mathcal{A}\|^8 \times \|\mathcal{V}\|^5 \times \|E\|)$
Shortest word length	Algorithm 3	$O(\|\mathcal{A}\|^8 \times \|\mathcal{V}\|^4 \times \|E\|)$	$O(\|\mathcal{A}\|^{11} \times \|\mathcal{V}\|^7 \times \|E\|^2)$
Shortest word	Boustrophedonic[6]	$O(\|w\|^2)$	$O(\|w'\|^2)$
Trace to shortest path	Algorithm 4	$O(\|T_C\| \times \|\mathcal{V}\| \times \|E\|)$	$O(\|T_C\| \times \|\mathcal{V}\| \times \|E\|)$

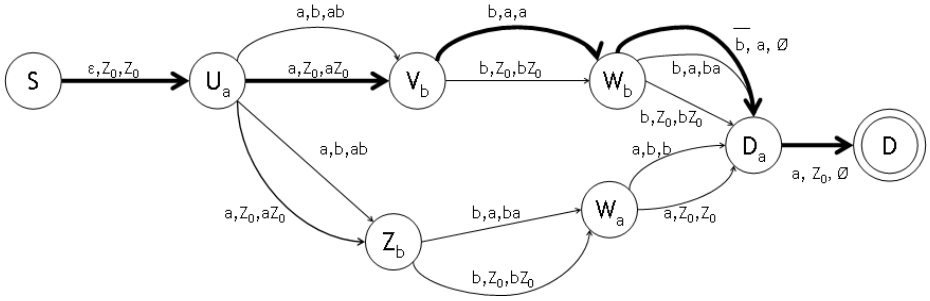
Table 1. Algorithms and their complexities

References

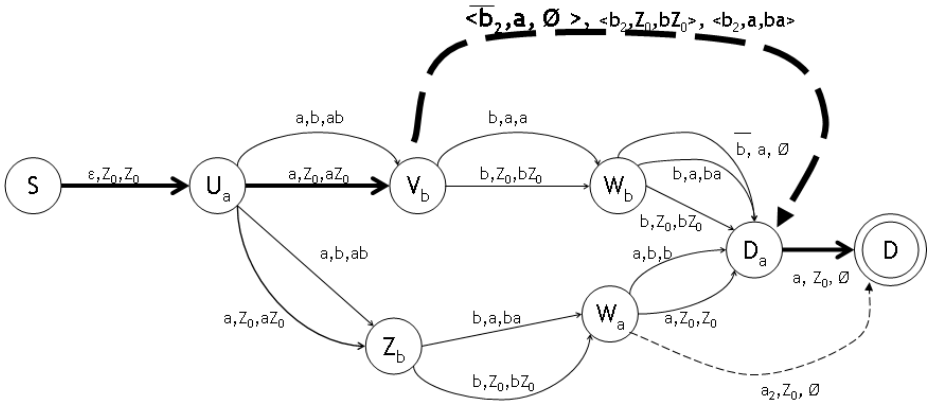
1. M. Bocci and S. Bryant. RFC5659 - An Architecture for Multi-Segment Pseudowire Emulation Edge-to-Edge, 2009.
2. S. Bryant and P. Pate. RFC3985 - Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture, 2005.
3. H. Cho, J. Ryoo, and D. King. Stitching dynamically and statically provisioned segments to construct end-to-end multi-segment pseudowires. `http://www.ietf.org/id/draft-cho-pwe3-mpls-tp-mixed-ms-pw-setup-01.txt`, 2011.
4. 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.
5. A. Farrel, JP. Vasseur, and J. Ash. RFC4655 - A Path Computation Element (PCE)-Based Architecture, 2006.
6. Ph. Flajolet, P. Zimmermann, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 1994.
7. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Longman Publishing, 2006.
8. F. A. Kuipers and F. Dijkstra. Path selection in multi-layer networks. *Computer Communications*, 2009.
9. M.L. Lamali, H. Pouyllau, and D. Barth. Appendices to "Path computation in multi-Layer multi-domain Networks". `https://www.ict-etics.eu/fileadmin/documents/publications/scientific_papers/report.pdf`, 2011.
10. M.L. Lamali, H. Pouyllau, and D. Barth. End-to-End Quality of Service in Pseudo-Wire Networks. In *ACM CoNEXT Student Workshop*, 2011.
11. Jens Liebehenschel. Lexicographical Generation of a Generalized Dyck Language. *SIAM J. Comput.*, 2003.
12. L. Martini, E. Rosen, N. El-Aawar, and G. Heron. RFC4448 - Encapsulation Methods for Transport of Ethernet over MPLS Networks, 2008.
13. K. Shiomoto, D. Papadimitriou, JL. Le Roux, M. Vigoureux, and D. Brungard. RFC5212 - Requirements for GMPLS-based multi-region and multi-layer networks (MRN/MLN), 2008.



(a) Network



(b) Corresponding PDA



(c) Transformed PDA

(1) $S_G \rightarrow [SZ_0D]$
(2) $[SZ_0D] \rightarrow \epsilon[U_aZ_0D]$
(3) $[U_aZ_0D] \rightarrow a[V_bZ_0D_a][D_aZ_0D]$
(4) $[V_bZ_0D_a] \rightarrow \overline{b_2}$
(5) $[D_aZ_0D] \rightarrow a$

(d) Subset of G_N which generates T_C

Fig. 2. Example