Bellman-Ford sous stéroïdes : Un algorithme de routage pour l'établissement automatique des tunnels

Mohamed Lamine Lamali[1] [†] et Simon Lassourreuille[1] et Stephan Kunne[2] et Johanne Cohen[2]

[1] *LaBRI - Univ. de Bordeaux, France*
[2] *LRI-CNRS. Université Paris-Sud, Université Paris Saclay, France*

Les tunnels sont omniprésents dans les réseaux d'aujourd'hui. Mais malgré leur importance, on ne dispose toujours pas de protocoles de routage pour les construire automatiquement, la plupart du temps ceux-ci sont établis par scripts ou sont précalculés. Dans ce travail, nous présentons le premier algorithme totalement distribué permettant de construire automatiquement les plus courts chemins comportant des tunnels.
La version complète de cet article est [LLKC19](`https://hal.archives-ouvertes.fr/hal-01987354v1`)

Mots-clefs : Tunnels, encapsulations de protocoles, calcul de chemins, protocoles de routage, algorithmes distribués.

1 Introduction

Dans un réseau multicouche, un tunnel est une portion d'un chemin où le paquet d'un protocole est encapsulé dans un autre. Un tunnel commence par une encapsulation et se termine par la désencapsulation correspondante. Les tunnels peuvent être imbriqués, la suite des en-têtes des paquets formant une *pile de protocole*. Les tunnels sont omniprésents dans les réseaux actuels (Transition IPv4/IPv6, sécurité, VPN, *Onion routing*, etc.). Malheureusement, les protocoles de routage actuels ne sont pas capables de construire automatiquement des tunnels, ceux-ci étant précalculés à l'avance et établis par scripts. Le problème algorithmique sous-jacent est celui du calcul du chemin, et plus particulièrement de décider où encapsuler. Il existe un algorithme polynomial centralisé pour résoudre le problème [LFC18], mais à notre connaissance, aucun algorithme distribué n'a été proposé. Dans cet article, nous proposons le premier algorithme de routage distribué avec établissement automatique de tunnels, c'est-à-dire prenant en compte les encapsulations et les conversions de protocoles. Notre algorithme est une généralisation de l'algorithme de Bellman-Ford distribué, où les vecteurs de distances sont remplacés par des vecteurs de piles. Cet algorithme construit des tables de routage permettant de savoir vers quel voisin envoyer un paquet en fonction de sa destination et de sa pile de protocoles. Nous montrons que la taille des messages est polynomiale, même si un plus court chemin peut être exponentiel. Nous montrons également que l'algorithme converge polynomialement en fonction de la taille du réseau et de son diamètre.

2 Modèle

2.1 Modèle du réseau

Nous reprenons le même modèle que dans [LFC18]. Un réseau multicouche est défini comme un quadruplet $\mathcal{N} = (\mathcal{G}, \mathcal{A}, \mathcal{F}, w)$, où $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ est un graphe orienté symétrique représentant la

[†] Financé par le projet HÉRA ANR-18-CE25-0002.

topologie et où chaque sommet est un routeur. Le nombre de sommets est noté n et le nombre de liens orientés est noté m. L'ensemble $\mathcal{A} = \{a, b, \ldots\}$ représente les protocoles disponibles dans le réseau, leur nombre est noté λ. Chaque routeur est capable d'effectuer certaines opérations (appelées *fonctions d'adaptation*) parmi les suivantes : *Conversion*, un protocole a est converti en un protocole b sans aucun changement dans les protocoles sous-jacents (i.e., déjà encapsulés). Cette fonction est notée $(a \to b)$ (ex. conversion IPv4/IPv6 via NAT-PT). Une retransmission sans changement de protocole est un cas particulier de conversion noté $(a \to a)$. *Encapsulation*, un protocole a est encapsulé dans b. Cette fonction est notée $(a \to ab)$ (ex. encapsulation d'IPv4 dans IPv6). *Désencapsulation*, un protocole a est désencapsulé (ou extrait) d'un protocole b. Cette fonction est notée $\overline{(a \to ab)}$.

L'ensemble de toutes les fonctions d'adaptation disponible dans le réseau est noté $\mathcal{F}$. L'ensemble $In(U)$ (resp. $Out(U)$) est l'ensemble des protocoles que le sommet U peut recevoir (resp. émettre).

Finalement, un coût positif $w(.)$ est associé à chaque triplet (U, f, V), où f est la fonction d'adaptation appliqué par U avant la transmission à V.

2.2 Pile de protocoles et chemin faisable

Une séquence de fonctions d'adaptation induit une pile de protocole. Par exemple, la séquence $(a \to a)(a \to ab)(b \to b)$ induit la pile ab (du bas vers le sommet). Si on reçoit un paquet avec une pile H est qu'on applique une fonction d'adaptation f, $f(H)$ sera la pile résultante. Par exemple, si on applique la fonction $f = (b \to bb)$ à la pile ab, le résultat sera abb. Mais dans certains cas, l'application est impossible. Par exemple, en appliquant la fonction $\overline{(a \to ab)}$ à la pile aba, il y aura un blocage car on essaie d'extraire un a d'un b alors que que le sommet de pile est un a et que le protocole sous-jacent est un b. Dans cette situation on notera $f(H) = \emptyset$, où $\emptyset$ représentera la pile interdite (à ne pas confondre avec une pile vide). On notera $\bar{f}$ la fonction inverse de f. Par exemple, si $f = (a \to b)$ alors $\bar{f} = (b \to a)$. La fonction inverse d'une encapsulation est la désencapsulation correspondante. La pile de protocole H_i induite par une séquence de fonctions $f_0 \ldots f_i$ est récursivement définie comme suit : $H_0 = x$ si $f_0 = (x \to x)$ [‡] et $H_i = f_i(H_{i-1})$

Dans un contexte multicouche, un chemin est noté $Sf_0U_1f_1U_2f_2 \ldots U_\ell f_\ell D$ où chaque U_i, $i = 1, \ldots, \ell$, est un sommet, et chaque fonction d'adaptation f_i est disponible sur le sommet U_i. Un chemin est donc faisable si et seulement si :

1. La séquence $SU_1U_2 \ldots U_\ell D$ est un chemin (au sens classique) dans $\mathcal{G} = (\mathcal{V}, \mathcal{E})$,
2. $H_\ell = x$ et $x \in In(D)$.

Le coût d'un chemin $\mathcal{P} = Sf_0U_1f_1U_2f_2 \ldots U_\ell f_\ell D$ est $w(\mathcal{P}) \stackrel{def}{=} \sum_{i=0}^{\ell} w(U_i, f_i, U_{i+1})$ où $S = U_0$ et $D = U_{\ell+1}$. Un chemin est optimal (ou plus court) s'il est faisable et qu'il minimise $w(\mathcal{P})$. Ainsi, la fonction de coût w peut exprimer n'importe quelle métrique additive sur les sommets et/ou les arcs.

Étant donné un réseau multicouche, notre problème est de calculer le plus court chemin faisable entre chaque paire de sommets (si celui-ci existe). Plus précisément, on veut calculer les tables de routage sur chaque sommet, tel qu'un paquet entre la source et la destination emprunte le chemin le plus court.

3 Un algorithme à vecteurs de piles

Nous avons opté pour la généralisation de l'algorithme de Bellman-Ford (notamment utilisé par le protocole de routage RIP) car il se prête bien à une version distribuée. Cet algorithme consiste, pour chaque sommet, à partager sa table de routage avec ses voisins de façons à construire les chemins les plus courts. On ne peut pas l'utiliser tel quel car dans un contexte multicouche, le chemin optimal dépend de la distance, mais également de la pile de protocoles du paquet à router. Ainsi, enregistrer le prochain voisin et le coût n'est pas suffisant pour construire une table de

‡. f_0 est une fonction fictive signifiant simplement que le sommet de départ émet le protocole x.

routage. Nous avons opté pour un algorithme à *vecteurs de piles*, où la destination, le prochain voisin, le coût *et* la pile de protocoles sont enregistrés dans la table de routage.

3.1 L'algorithme

L'initialisation consiste, pour un sommet U, à envoyer à ses voisins $Voi(U)$ l'ensemble des protocoles $In(U)$ qu'il peut recevoir. Envoyer un message (U,H,c) signifie que l'émetteur du message peut joindre la destination U avec la pile H à un coût c. L'initialisation consiste donc à informer ses voisins qu'on peut être joint à coût 0 avec une pile de protocole x, où $x \in In(U)$.

Chaque entrée de la table de routage $\mathcal{T}$ de chaque sommet est un quintuplet (D,H,c,V,f) où D est la destination, H est la pile de protocoles nécessaire pour atteindre D, c est le coût du chemin restant, V est le prochain voisin et f est la fonction d'adaptation que le sommet U doit appliquer au paquet avant de l'envoyer à V. Chaque entrée est indexée par le couple (D,H). Donc $\mathcal{T}(D,H)$ retourne l'entrée (ligne de la table de routage) correspondant à la destination D et à la pile H. Quand un sommet essaie d'ajouter une nouvelle entrée (D,H,c,V,f) à sa table de routage $\mathcal{T}$, il vérifie d'abord que celle-ci n'existe pas déjà. Autrement, il compare le coût de l'ancienne entrée et de la nouvelle, et la modifie le cas échéant. Cette étape correspond à l'Algorithme 1.

Algorithme 1 Ajout d'une entrée à la table de routage du sommet U.

Entrée : Une entrée de la table de routage (D,H,c,V,f)

1: **Si** $(D,H) \notin \mathcal{T}$ **alors**
2: Ajouter (D,H,V,c,f) à la table de routage
3: **Sinon Si** $\mathcal{T}(D,H).cout > c$ **alors**
4: $\mathcal{T}(D,H).cout \leftarrow c$
5: $\mathcal{T}(D,H).next\_hop \leftarrow V$
6: $\mathcal{T}(D,H).fonction \leftarrow f$

L'Algorithme 2 construit les tables de routage de façon distribuée. Quand un sommet U reçoit d'un sommet V le message (D,H,c), U détermine quelle fonction d'adaptation f il peut appliquer à la pile H (lignes 3-5). Ensuite, il calcule le nouveau coût du chemin restant en ajoutant $w(U,f,V)$ au précédent coût (ligne 6). La ligne est ensuite ajoutée selon l'Algorithme 1 (ligne 7). Si la table de routage a été modifiée, il envoie le message (D,H',c') à tous ses voisins, indiquant qu'il peut atteindre la destination D au coût c' avec la pile de protocole H'.

Il faut noter que si f est une désencapsulation $\overline{(a \rightarrow ab)}$, et que U peut joindre D avec une pile aa, alors il peut recevoir la pile aab puis appliquer f. Cela revient à appliquer la fonction inverse $\bar{f}$ avant d'informer ses voisins (ligne 4).

Algorithme 2 Construction de la table de routage du sommet U

1: **Répéter**
2: Recevoir le message (D,H,c) de V
3: **Pour tout** $f \in \mathcal{F}(U)$ **faire**
4: $H \leftarrow \bar{f}(H)$
5: **Si** $H \neq \emptyset$ et $h(H) \leq \lambda n^2$ **alors**
6: $c \leftarrow c + w(U,f,V)$
7: Ajouter l'entrée (D,H,c,V,f) à la table de routage
8: # *en appliquant l'Algorithme 1*
9: **Si** la table de routage a été modifiée **alors**
10: **Pour tout** $W \in Voi(U)$ **faire**
11: Envoyer (D,H,c) à W

Le routage d'un paquet se fait de la manière suivante : on suppose qu'un paquet est un triplet (destination, pile des en-têtes de protocoles, données), par exemple $(D,H,data)$. Quand un sommet

U reçoit ce paquet, il cherche l'entrée correspondant au couple (D,H) dans sa table de routage. S'il n'y a aucune entrée correspondante, le paquet est détruit. Sinon, si l'entrée correspondante est (D,H,c,V,f), il transmet le paquet $(D,f(H),data)$ à V.

3.2 Convergence

Étant donné qu'un plus court chemin faisable peut contenir des circuits (il peut même être de longueur exponentielle [LFC18]), les piles construites par l'Algorithme 2 peuvent croître indéfiniment. La ligne 5 de cet algorithme impose que la pile ne dépasse pas la hauteur λn^2. Le Lemme 1 montre que cette hauteur est suffisante.

Lemme 1. *S'il existe un plus court chemin faisable entre deux sommets, alors la hauteur de pile de protocoles le long du chemin ne peut dépasser λn^2 protocoles.*

La preuve est liée au lemme de l'étoile. Voir [Sén90] et [AJ12] par exemple. Elle est détaillée dans la version complète de l'article [LLKC19].

Corollaire 1. *Si les coûts sont bornés par une constante, la taille maximale d'un message est $O(\lambda \log \lambda\ n^2)$.*

La Proposition 1 donne le temps de convergence de l'algorithme dans un modèle synchrone. Pour cela, définissons d'abord le diamètre d'un réseau multicouche :

Définition 1. *Le diamètre d'un réseau multicouche $\mathcal{N}$, noté diam $\mathcal{N}$, est le plus long (en nombre de liens) des plus courts chemins faisables entre deux sommets. Plus formellement :*

$$diam\ \mathcal{N} \stackrel{def}{=} \max_{\mathcal{P}\ plus\ court\ chemin\ faisable} |\mathcal{P}|$$

où $|\mathcal{P}|$ est le nombre de liens du chemin.

L'algorithme converge polynomialement en fonction de la taille du réseau et de son diamètre.

Proposition 1. *L'Algorithme 2 calcule les tables de routages correctes après $O(\lambda n^2\ diam\ \mathcal{N})$ rounds.*

La preuve peut être consultée dans la version complète de l'article [LLKC19].

4 Conclusion et perspectives

Dans cet article, nous avons donné le premier algorithme distribué pour le calcul de chemins faisables, et donc l'établissement automatique de tunnels. Cet algorithme est une généralisation de l'algorithme de Bellman-Ford pour la prise en compte des piles de protocoles. Nous avons montré que la taille des messages et polynomiale et que la convergence est également polynomiale, mais en prenant comme paramètre le diamètre. Néanmoins, cet algorithme souffre de plusieurs limitations. Bien que la taille des messages soit polynomiale, leur nombre peut être exponentiel, de même que la taille de la table de routage. Nos travaux futurs se concentreront sur la résolution de ces difficultés.
Remerciements : les auteurs remercient Géraud Sénizergues pour ses explications concernant le Lemme 1.

Références

[AJ12] Antoine Amarilli and Marc Jeanmougin. A proof of the pumping lemma for context-free languages through pushdown automata. *arXiv preprint arXiv :1207.2819*, 2012.

[LFC18] M. L. Lamali, N. Fergani, and J. Cohen. Algorithmic and complexity aspects of path computation in multi-layer networks. *IEEE/ACM Transactions on Networking*, 2018.

[LLKC19] Mohamed Lamine Lamali, Simon Lassourreuille, Stephan Kunne, and Johanne Cohen. A stack-vector routing protocol for automatic tunneling. In *IEEE INFOCOM 2019*, pages 1–9, 2019.

[Sén90] Géraud Sénizergues. A characterisation of deterministic context-free languages by means of right-congruences. *Theor. Comput. Sci.*, 70(2) :213–232, 1990.