(Presque) Floyd-Warshall sous stéroïdes : (encore) un algorithme de routage pour l'établissement automatique de tunnels

Noureddine Mouhoub, Mohamed Lamine Lamali, Damien Magoni

(Presque) Floyd-Warshall sous stéroïdes : (encore) un algorithme de routage pour l'établissement automatique de tunnels †

Noureddine Mouhoub,[1] Mohamed Lamine Lamali,[1] Damien Magoni[1]

[1] *LaBRI - Université de Bordeaux, 351 cours de la Libération, 33400 Talence, France*

Le routage est un problème fondamental dans les réseaux de communication. Il se base sur des algorithmes de calcul de chemins dans les graphes (Dijkstra, Bellman-Ford, etc.). Ces algorithmes sont étudiés depuis longtemps et très bien compris, du moment qu'ils s'appliquent à un réseau disposant d'un seul protocole de communication. Le problème devient plus complexe dans le cas multiprotocolaire, avec possibilité d'encapsulation de certains protocoles dans d'autres. Cette opération induit des tunnels qui peuvent être imbriqués. Les algorithmes classiques cités ci-dessus ne fonctionnent plus dans ce cas car ils ne peuvent gérer les encapsulations et les piles de protocoles correspondantes (une pile de protocole est la suite des en-têtes des paquets encapsulés). Dans ce travail préliminaire, nous proposons un algorithme *à la* Floyd-Warshall prenant en compte les encapsulations et les conversions de protocoles afin de calculer les chemins les plus courts dans un réseau multiprotocolaire. Pour ce faire, nous étudions la fermeture transitive (concaténation) entre sous-chemins dans le cas où chaque sous-chemin induit une pile de protocoles. Certains sous-chemins ne pouvant pas être concaténés car leur piles de protocoles sont incompatibles. Cet algorithme peut-être déployé en centralisé, ou bien dans un contexte distribué avec mémoire partagée.

Mots-clefs : Tunnels, fermeture transitive, calcul de chemins, encapsulation de protocoles.

1 Introduction et état de l'art

Les réseaux actuels comportent de plus en plus de protocoles de communication différents se basant sur des technologies parfois incompatibles (ex. IPv4/IPv6 pour Internet, RPL dans 6LowPAN, etc.). Il existe deux opérations principales pour permettre la connectivité dans ce contexte : *i)* la *conversion* de protocoles, *ii) l'encapsulation* de protocoles. Ces opérations sont appelées *fonctions d'adaptation*. Un tunnel est un sous-chemin qui commence par une encapsulation et qui se termine par la désencapsulation correspondante. Les tunnels sont parfois imbriqués.

Bien que plusieurs tentatives d'automatisation aient eu lieu (ex. 6to4 [CM01], ISATAP [IET10], etc.), dans les réseaux actuels, il n'existe aucun mécanisme de calcul de chemins (routage) totalement automatisé, *i.e.,* permettant le calcul du chemin le plus court en déterminant automatiquement les nœuds de début et de fin des tunnels. Ceci est dû à la difficulté de concevoir des algorithmes de calcul de chemins prenant en compte les fonctions d'adaptation. En effet, il existe peu de travaux algorithmiques sur ce problème. Dans [KD09], les auteurs proposent un algorithme exponentiel en Brute-Force, *i.e.,* qui explore tous les chemins possibles afin de trouver un chemin faisable. Les auteurs de [LFC18] proposent un algorithme basé sur la théorie des langages permettant de calculer le chemin de bout en bout. Néanmoins, il ne permet pas de calculer les tables de routage, il n'est ainsi adapté qu'au routage source. Le premier algorithme permettant le calcul des tables de routage est décrit dans [LLKC19]. Ce dernier est une généralisation de Bellman-Ford en remplaçant le vecteur de distance par un vecteur de piles de protocoles.

Une piste intéressante est le calcul de chemins par *la fermeture transitive* : deux (sous-)chemins sont concaténés pour n'en former qu'un seul. L'algorithme de fermeture transitive le plus connu est celui de Floyd-Warshall. Cette approche ne peut pas être directement appliquée dans notre cas car deux sous-chemins ne sont pas forcément *compatibles* (ex. un premier sous-chemin se terminant avant la fin d'un

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

tunnel alors que le deuxième commence par deux tunnels imbriqués – ceci sur le même nœud). La première partie de notre travail est donc de déterminer dans quels cas il est possible de concaténer deux sous-chemins. En se basant sur ces résultats, nous proposons un algorithme qui calcule le plus court chemin entre toutes les paires de nœuds dans le réseau, ces (sous-)chemins pouvant comporter des tunnels imbriqués et des conversions de protocoles. Pour implémenter cet algorithme, la seule nécessité et que chaque nœud ait accès à la table de routage des autres. L'algorithme peut donc être totalement centralisé ou bien distribué avec mémoire partagée.

2 Modèle et formalisation du problème

Nous reprenons le même modèle et les mêmes notations que dans [LLKC19, LFC18].

Une suite de fonctions d'adaptation $f_0 f_1 f_2 \dots f_k$ induit une pile de protocoles. Par exemple, sur la Figure 1 si on applique la suite $(a \to ab)(b \to b)$ à la pile a [‡], on induit la pile ab [§]. On notera $f(H)$ l'application de la fonction d'adaptation f à la pile H. Parfois, l'application n'est pas possible : si $H = aa$ et $f = (b \to a)$, on ne pourra pas convertir le sommet de pile b en a vu que ce protocole est différent de b. Dans cette situation, nous notons $f(H) = \emptyset$ (pile interdite), ainsi on ne pourra pas aller plus loin, *i.e.,* $f(\emptyset) = \emptyset$ quelle que soit la fonction d'adaptation f. Par exemple, sur la Figure 1, le nœud U_4 ne peut pas appliquer la fonction $\overline{(a \to ab)}$ à la pile b (chemin noir en-dessous). Enfin, la hauteur d'une pile H est notée $h(H)$.

Un sous-chemin est une suite de nœuds, de fonctions d'adaptation et de piles. Pour un sous-chemin donné, on notera H_{in} (resp. H_{out}), la pile que reçoit le nœud de départ (resp. d'arrivée) de ce sous-chemin.

Définition 1. *Un sous-chemin est une séquence $H_{in}U_i f_i U_{i+1} f_{i+1} \dots U_{j-1} f_{j-1} H_{out} U_j$, $(i < j-1)$ où chaque U_k est un nœud, et chaque f_k est une fonction d'adaptation. On dit que ce sous-chemin est valide ssi :*

- *La suite de nœuds $U_i U_{i+1} \dots U_j$ est un chemin dans le graphe $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ et chaque $f_k \in \mathcal{F}(U_k)$,*
- *La suite de fonctions $f_i f_{i+1} \dots f_{j-1}$ appliquée à la pile H_{in} induit une pile de protocoles $H_{out} \neq \emptyset$ dont le sommet est un protocole $y \in In(D)$.*

Définition 2. *Un chemin $\mathcal{P} = H_{in} S f_0 U_1 f_1 U_2 f_2 \dots U_\ell f_\ell H_{out} D$ d'un nœud S à un nœud D est faisable ssi :*

- *$\mathcal{P}$ est un (sous-)chemin valide,*
- *f_0 est une fonction fictive $(x \to x)$ indiquant que S émet le protocole x,*
- *La pile $H_{in} = x$ est de hauteur 1 et la pile $H_{out} = y$ est de hauteur 1 avec $y \in In(D)$.*

Dans la Figure 1, le sous-chemin $H_{in} U_2 (a \to ab) U_3 (b \to b) H_{out} U_4$ avec $H_{in} = a$ (reçu par U_2) et $H_{out} = ab$ (reçu par U_4) est un sous-chemin valide du chemin en vert. Le chemin du haut (en vert) représente un chemin faisable de S à D avec $H_{in} = b$ et $H_{out} = a$. Le chemin du bas (en noir) est non faisable, puisque le nœud U_4 reçoit le protocole b et il ne peut pas désencapsuler le protocole a du b. Formellement, la pile en sortie de U_4 sur ce chemin est la pile interdite $\emptyset$. Par la suite, on omettra de préciser H_{in} et H_{out} dans les chemins faisables car on peut les déduire de f_0 et f_ℓ.

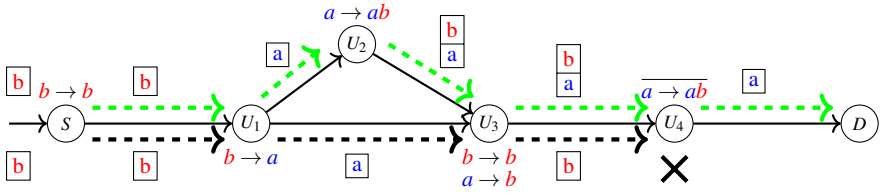


FIGURE 1 – Exemple de sous-chemin valide, chemin non faisable et chemin faisable.

Le coût d'un chemin $\mathcal{P} = S f_0 U_1 f_1 U_2 f_2 \dots U_\ell f_\ell D$ est $w(p) = \sum_{i=0}^{\ell} w(U_i, f_i, U_{i+1})$ où $U_0 = S$ et $U_{\ell+1} = D$. Un chemin plus court est un chemin faisable qui minimise $w(\mathcal{P})$. Le problème qu'on cherche à résoudre est de construire la table de routage (définie dans la section 4) de chaque nœud du réseau, autrement dit calculer tous les plus courts chemins faisables par fermeture transitive des plus courts sous-chemins valides.

‡. Une pile de hauteur 1 signifie qu'il n'y a pas de protocole encapsulé. Ainsi, la notation d'une pile de hauteur 1 et du protocole qu'elle contient est la même. Par abus de notation, on pourra considérer cette pile comme étant un protocole.

§. Par convention, le sommet de pile est le plus à droite.

3 Fermeture transitive de sous-chemins valides

Soit deux sous-chemins valides $H_{in}S\dots H_{out}D$ et $H'_{in}D\dots H'_{out}T$. Le cas le plus simple de concaténation de ces deux sous-chemins est lorsque $H_{out} = H'_{in}$. Ici, la concaténation est directe. La Figure 2, illustre un deuxième cas où $H_{out} = b$ et $H'_{in} = ab$. Cela dit, en considérant les deux piles comme des mots, on peut constater que H_{out} est un suffixe de H'_{in}. Posons $H'_{in} = H.H_{out}$ où "." dénote la concaténation (de piles) et H est la pile qui correspond à H'_{in} privée de son suffixe H_{out}.

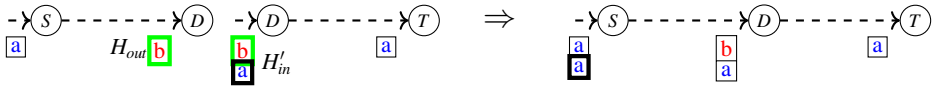


FIGURE 2 – Concaténation possible avec $H_{out} \neq H'_{in}$.

Le troisième cas où la concaténation est possible est symétrique au précédent. A partir de deux sous-chemins valides $H_{in}S\dots H_{out}D$ et $H'_{in}D\dots H'_{out}T$, on peut donc obtenir un sous-chemin valide $H''_{in}S\dots H''_{out}T$ de la manière suivante :

$$(H''_{in}, H''_{out}) = \begin{cases} (H_{in}, H'_{out}) & \text{si} \quad H_{out} = H'_{in} \\ (H.H_{in}, H'_{out}) \text{ où } H = H'_{in} - H_{out} & \text{si} \quad H_{out} \text{ est un suffixe de } H'_{in} \\ (H_{in}, H.H'_{out}) \text{ où } H = H_{out} - H'_{in} & \text{si} \quad H'_{in} \text{ est un suffixe de } H_{out} \end{cases} \tag{1}$$

Si l'algorithme est distribué, il faut que S puisse accéder aux sous-chemins valides partant de D afin de décider si une concaténation est possible, et mettre à jour sa propre table et celle de D, d'où la nécessité d'une mémoire partagée.

4 Algorithme pour le calcul des plus courts chemins

Nous présentons ici un algorithme de calcul de chemins en utilisant la fermeture transitive des tables de routage. Chaque ligne d'une table de routage $\mathcal{T}_U$ d'un nœud U est de la forme $(D, H_{in}, w, V, f, H_{out})$ où D est la destination, H_{in} est la pile de protocoles qu'un paquet doit contenir pour parcourir la route de U à D, la valeur w est le coût de cette route, V est le prochain voisin sur cette route (next hop), f est la fonction d'adaptation que U applique à un paquet reçu avant de le transmettre à V, enfin, H_{out} est la pile de protocoles reçue par D sur cette route.

Initialisation. Pour chaque lien (U,V) et chaque $f \in \mathcal{F}(U)$, ajouter la route $(V, H_{in}, w(U,f,V), V, f, H_{out})$ à la table $\mathcal{T}_U$. Le couple de piles $(H_{in}, H_{out}) = (a,b)$ si $f = (a \to b)$, de même, $(H_{in}, H_{out}) = (a, ab)$ si $f = (a \to ab)$, et $(H_{in}, H_{out}) = (ab, a)$ si $f = \overline{(a \to ab)}$.

Construction des tables de routage. Durant cette étape, on vérifie la compatibilité de toutes les combinaisons de sous-chemins valides, *i.e.,* toutes les lignes des tables de routage $(D, H_{in}, w, V, f, H_{out})$ de $\mathcal{T}_S$ et $(T, H'_{in}, w', V', f', H'_{out})$ de $\mathcal{T}_D$, pour un couple de nœuds S et T quelconque. Si les deux routes sont compatibles (selon les conditions de (1)) alors on ajoute la ligne $(T, H''_{in}, w + w', V, f, H''_{out})$ à $\mathcal{T}_S$ (où H''_{in} et H''_{out} sont également définies selon la formule (1)). Lorsqu'un nœud S tente d'ajouter une nouvelle ligne $(T, H''_{in}, w + w', V, f, H''_{out})$ à sa table de routage $\mathcal{T}_S$, il vérifie si le triplet (T, H''_{in}, H''_{out}) est déjà présent dans sa table. Si ce n'est pas le cas, la nouvelle ligne est insérée dans la table. Sinon, le coût $w + w'$ du nouveau chemin est comparé à l'ancien. La table est mise à jour en remplaçant l'ancien chemin par le nouveau si le nouveau coût de celui-ci est inférieur et si la hauteur de H''_{in} (resp. H''_{out}) $\leq \lambda n^2$, car il est prouvé dans [LLKC19] que la pile de protocole d'un plus court chemin faisable (ou valide) ne peut dépasser cette valeur, autrement il existerait un chemin plus court. Selon cette dernière valeur et la ligne 6, si aucune route n'a été ajoutée ou modifiée alors le calcul des tables de routage est fini. Afin d'obtenir tous les chemins faisables, il suffit de prendre les routes où $h(H_{in}) = 1$ et $h(H_{out}) = 1$. Le reste des routes sont des sous-chemins valides et ils sont uniquement nécessaires pour le routage. L'algorithme suivant décrit cette étape.

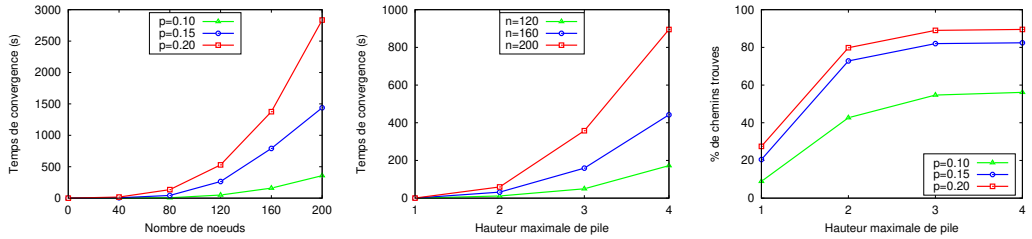
Complexité. Notre algorithme à une complexité en $O(\text{diam}\mathcal{N} \times n \times |\mathcal{T}|^2)$, tels que diam$\mathcal{N}$ est la taille du plus long (en nombre de liens) plus court (en coût) chemin faisable dans $\mathcal{N}$, et $|\mathcal{T}|$ est le nombre de routes qu'une table de routage peut contenir. Ce dernier est en $O(n\lambda^{\lambda n^2})$.

Algorithme : Construction de la table de routage d'un nœud S.

(1) **Répéter**
(2) **Pour tout** $(D, H_{in}, w, V, f, H_{out}) \in \mathcal{T}_S$ **faire**
(3) **Pour tout** $(T, H'_{in}, w', V', f', H'_{out}) \in \mathcal{T}_D$ **faire**
(4) Calculer (H''_{in}, H''_{out}) en fonction de (H_{in}, H_{out}) et (H'_{in}, H'_{out}) *# selon la formule* (1)
(5) **Si** $h(H''_{in}) \leq \lambda n^2$ *et* $h(H''_{out}) \leq \lambda n^2$ **alors**
(6) Ajouter la route $(T, H''_{in}, w + w', V, f, H''_{out})$ à la table de routage $\mathcal{T}_S$
(7) *# si elle n'existe pas ou si la précédente a un coût supérieur*

5 Simulations

Nous avons effectué des simulations pour mesurer le temps de convergence et le % de chemins faisables trouvés en fonction de la taille du réseau n, de la probabilité p d'existence d'une fonction d'adaptation sur un nœud et de la hauteur maximale de pile de protocoles h_{max}. Dans la figure 3a, si $h_{max} = 3$ et $p = 0.10$ alors le temps de convergence de l'algorithme est d'environ $6s$ (resp. $159s$) dans un réseau ayant 80 (resp. 160) nœuds, tandis que pour $p = 0.20$, le temps de convergence est environ $135s$ (resp. $1376s$). La figure 3b montre l'impact de h_{max} sur le temps. Pour $n = 200$ et $p = 0.10$, le temps de convergence est d'environ $59s$ (resp. $895s$) si $h_{max} = 2$ (resp. 4). La figure 3c montre que dans un réseau de 200 nœuds et avec $p = 0.20$, l'algorithme trouve environ 27%(resp. 90%) de paires de nœuds reliés par un chemin faisable si h_{max} est fixée à 1 (resp. 3). Ces simulations montrent que le temps de convergence et le % de chemins trouvés dépendent fortement de la probabilité p (le nombre de fonctions par nœud) et de la hauteur maximale de pile h_{max} (le nombre de protocoles encapsulés). Ceci montre l'influence de certains paramètres du problème.



(a) Temps de convergence ($h_{max} = 3$). (b) Temps de convergence ($p = 0.10$). (c) % de chemins trouvés ($n = 200$).

FIGURE 3 – Temps de convergence de l'algorithme et % de chemins faisables trouvés.

6 Conclusion et perspectives

Nous avons présenté un algorithme de calcul de chemins avec établissement automatique de tunnels. Ce dernier est presque une généralisation de l'algorithme Floyd-Warshall en faisant uniquement la fermeture transitive entre les (sous)-chemins avec piles de protocoles compatibles. Parmi les travaux futurs, citons d'une part, l'évaluation de notre algorithme dans un contexte distribué avec mémoire partagée, et d'autre part, l'optimisation de l'opération de fermeture transitive afin d'accélérer le calcul des tables.

Références

[CM01] B. Carpenter and K. Moore. RFC 3056 : Connection of IPv6 Domains via IPv4 Clouds, 2001.

[IET10] IETF. RFC 5579 : Transmission of IPv4 Packets over Intra-Site Automatic Tunnel Addressing Protocol Interfaces, 2010.

[KD09] Fernando Kuipers and Freek Dijkstra. Path selection in multi-layer networks. *Comput. Commun.*, 32(1) :78–85, jan 2009.

[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*, 26(6) :2787–2800, 2018.

[LLKC19] M. L. Lamali, S. Lassourreuille, S. Kunne, and J. Cohen. A stack-vector routing protocol for automatic tunneling. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1675–1683, 2019.