INSTITUT POLYTECHNIQUE DE PARIS

Sur l'optimisation du routage IGP

Soumis le 19 février 2021
écrit par
Paul Régnier (paul.regnier@epfl.ch), Lomàn Vezin (loman.vezin@epfl.ch)
supervisé par
Eric Gourdin (eric.gourdin@orange.com)

***Résumé*—La démultiplication du nombre de services internet proposés aux utilisateurs, particuliers et entreprises, entraîne une augmentation conséquente du volume de trafic sur les réseaux. Notamment suite à la crise du Covid-19, la consommation internet à augmenté de plus de 30%. Une mauvaise gestion de la congestion peut entraîner un ralentissement du trafic, une augmentation de la perte de paquets ou encore la baisse de performance, pouvant tous s'avérer coûteux pour l'opérateur. Nous allons nous placer dans la situation d'un administrateur réseau. A partir de l'information disponible sur ce dernier nous nous proposons de trouver des solutions d'écoulement selon une matrice de demande donnée en minimisant la congestion.**

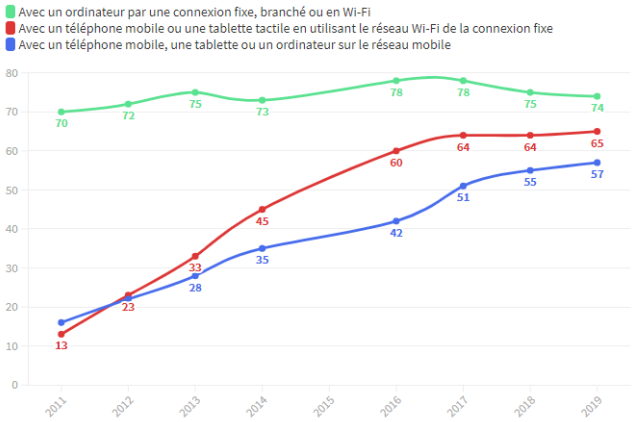

FIGURE 1. Pourcentage des français équipés d'un accès internet selon les appareils. Source : Baromètre du numérique, 2019

I. INTRODUCTION

En tant que candidat pour la future plate-forme de communication multiservices, la technologie IP doit pouvoir répondre à un large éventail d'attentes sur la qualité du service, selon les besoins par différents types de demandes. À cet égard, elle devrait également être en mesure de fournir la souplesse de gestion nécessaire pour contrôler la répartition du trafic sur le réseau afin de maintenir les ressources dans les conditions opérationnelles souhaitées, nécessaires pour garantir la qualité du service. Ainsi, l'ingénierie du trafic (TE), c'est-à-dire la cartographie des flux de trafic sur la topologie physique du réseau existant de façon optimale afin d'atteindre les objectifs opérationnels souhaités est devenue une question importante dans les réseaux IP. Il y a plusieurs approches pour déployer le *trafic engineering* dans les réseaux IP actuels, en optimisant les paramètres pour les décisions de routages ou en utilisant des modèles fixes notamment. On se concentrera sur l'optimisation des paramètres dans ce projet.

II. FORMULATION DU PROBLÈME

Nous nous mettons à la place d'un administrateur réseau cherchant à améliorer le routage de son réseau à partir des informations disponibles. On modélise le réseau en question par un graphe $G = (V, E)$ avec V un ensemble de n *noeuds* modélisant les routeurs et E un ensemble de m *liens* $\{u, v\},\ u, v \in V$, entre ces noeuds. Le graphe est bi-orienté c'est à dire que chaque lien $\{u, v\}$ dispose de deux arcs orientés (u, v). On suppose que les routeurs peuvent communiquer entre eux dans les deux sens. Concrètement l'administrateur réseau dispose pour chaque $e \in E$ de sa capacité $c(e)$ et d'une métrique $w(e)$.

Un flot f sur le graphe G est définit comme solution du système suivant

$$\forall e \in E, \forall i \neq j \in V,$$
$$f_{ij}(e) \geq 0,$$
$$\forall k \in V, \forall i, j \neq k,$$
$$\sum_{e \in \text{ OUT}(k)} f_{ij}(e) - \sum_{e \in \text{ IN}(k)} f_{ij}(e),$$
$$,$$

où IN(k), OUT(k) représentent respectivement les sous ensembles des arcs de E dont k est la source, la destination. La dernière contrainte modélise simplement la conservation du flot à travers le réseau.

Nous cherchons à délivrer une demande D donnée sous la forme d'une matrice $n \times n$ dont les coefficients vérifient

$$D_{ij} = \sum_{e \in \text{ OUT}(i)} f_{ij}(e) - \sum_{e \in \text{ IN}(i)} f_{ij}(e) \geq 0,$$

où D_{ij} est le volume de la demande partant du noeud i vers le noeud j.

Nous allons chercher à calculer à partir de ces données un *flot* optimal minimisant la *congestion* sur le réseau.

Le flot total sur chaque lien e est défini comme

$$\mathbf{flow}(e, f, D) = \sum_{ij} D_{ij} f_{ij}(e)$$

et la congestion correspond à la congestion maximale par rapport à tous les liens, c'est à dire

$$\mathbf{cong}(f, D) = \max_{e \in E} \frac{\mathbf{flow}(e, f, D)}{c(e)}.$$

Un *routing* f entre les noeuds i et j modélise le transport d'une unité de flot sortant de i et entrant en j. Il laisse les autres noeuds inchangés. Ainsi un *routing* doit naturellement satisfaire les contraintes linéaires suivantes

$$\sum_{e \in \text{OUT}(k)} f_e - \sum_{e \in \text{IN}(k)} f_e = 0, \ \forall k \neq i, j$$
$$\sum_{e \in \text{OUT}(i)} f_e - \sum_{e \in \text{IN}(i)} f_e = 1$$
$$f_{ij}(e) \geq 0, \ \forall e \in E$$

Nous allons dans un premier temps explorer différents protocoles de routage tels que l'IGP, puis dans un second temps nous chercherons à optimiser la métrique du réseau pour minimiser la congestion dans ces protocoles.

III. Le routing IGP

Le routing *IGP* ou *Interior Gateway Protocol* est un protocole de routage utilisé dans les systèmes autonomes. L'*IGP* établit les routes optimales entre un point du réseau et le reste des destinations possibles [2]. Il assure également la convergence des données (i.e. si un routeur s'arrête brutalement, un nouvel itinéraire sera fourni rapidement).

A. L'algorithme

Le protocole s'effectue de cette manière :

— L'administrateur du réseau va devoir attribuer des poids aux liens

— Chaque routeur va calculer l'ensemble des plus courts chemins par rapport à ces poids vers toutes les destinations possibles

— Le trafic est distribué le long de ces plus courts chemins

Il existe plusieurs manières de répartir le trafic le long des plus courts chemins et plusieurs façon de calculer les poids des liens. Citons par exemple le routage *ECMP* ou *Equal Cost Multi-Path* qui sépare le flot de manière égale entre les plus court chemins, voir figure 2.

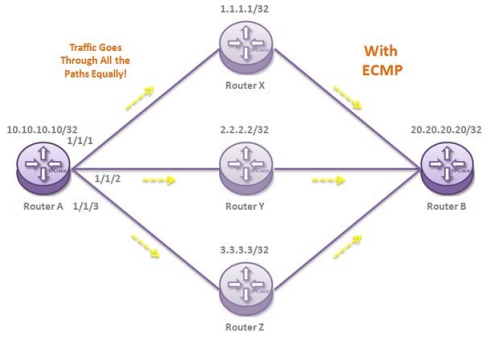

FIGURE 2. Source : IPCisco

Lors de ce projet, nous avons abordé plusieurs manières de router les données le long des plus courts chemins afin de comparer les performances.

B. L'implémentation

Nous utilisons le language python pour implémenter nos algorithmes sur une carte de routeur test représentant l'Europe, voir figure 3 ci dessous. Le réseau test comporte 37 routeurs situés dans les grandes villes européennes et 57 liaisons entre ces derniers. Nous

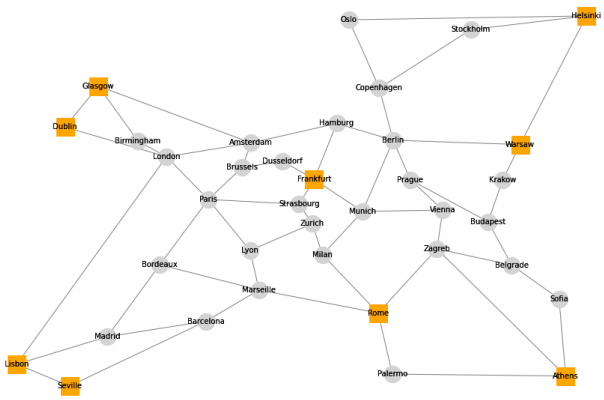

FIGURE 3. Le graphe test sur lequel nous travaillons

cherchons à satisfaire un ensemble de 36 demandes représentatives sur ce réseau.

Nous utilisons de plus les librairies `NetworkX` pour l'implémentation des graphes et plus tard `Pyomo` pour la résolution de programmes linéaires. Par soucis de simplicité nous implémentons la matrice de demande D comme un graphe de demande K ayant les mêmes noeuds que notre graphe d'étude G et dont les liens représentent les demandes avec leur valeur associée.

Le premier algorithme d'IGP se veut simpliste afin de mieux comprendre les enjeux du problème, nous introduirons plus tard l'ECMP. Notre algorithme calcule tous les plus courts chemins satisfaisant une demande donnée en utilisant l'algorithme de Dijkstra puis répartit le flux de données sur chaque lien e comme la moyenne de ses apparitions parmi les liens les plus courts. [3]

De cette manière, les données sont réparties équitablement sur les plus courts chemins. Par exemple, si il y a 3 plus courts chemins entre le noeud A et le noeud B, alors 33% des données seront envoyées sur le plus court chemin 1, puis 33% sur le plus court chemin 2 et enfin 33% sur le dernier plus court chemin.
Le principal inconvénient de ce protocole est que deux plus courts chemins peuvent partager un lien, lequel se retrouverait donc avec 66% des données dans le cas précédent. Par contre, l'algorithme est très succinct et permet de calculer vite le routage.

Dans cette optique nous proposons notre implémentation de l'algorithme ECMP. Cette dernière se base sur le calcul préalable de tables de routages. Pour une demande donnée on calcule une *forward table* donnant pour chaque noeud u l'ensemble des successeurs v tel que (u, v) est un lien appartenant à un plus court chemin satisfaisant la demande. Suivant le même principe on calcule une *backward table* donnant cette fois l'ensemble des noeuds prédécesseurs. Pour des raisons pratiques nous ajoutons également une source virtuelle s avant le noeud source de la demande et de même nous ajoutons un puits virtuel t pour le noeud puits.

```
def forward_table(G, K, i):
    dem = list(K.edges)[i]
    dem_val = K[dem[0]][dem[1]]['demand']
    a_paths = list(nx.all_shortest_paths(G, dem[0],
    dem[1], weight='weight'))

    table = {}
    for path in a_paths:
        for i in range(len(path)-1):
            if path[i] in table:
              if path[i+1] not in table[path[i]]:
                table[path[i]].append(path[i+1])
            else:
                table[path[i]] = [path[i+1]]
    table[dem[1]] = ['t']
    return table
```

Le code de la table arrière est similaire. Nous pouvons enfin implémenter notre algorithme ECMP.

```
def ECMP(G, K, i):
    # Calcul des tables
    f_table = forward_table(G, K, i)
    b_table = backward_table(G, K, i)
    # Initialisation des constantes
    dem = list(K.edges)[i]
    dem_val = K[dem[0]][dem[1]]['demand']
    flow, load = {}, 0
    # Calcul de la répartition du flux
    while(flow[(list(f_table)[-1], 't')] != 1):
        flow[('s', list(f_table)[0])] = 1
        for nodeA in f_table:
            for nodeB in f_table[nodeA]:
                flow[(nodeA, nodeB)] = 0
        for nodeA in f_table:
            for nodeB in f_table[nodeA]:
                in_flow = 0
                for nodeC in b_table[nodeA]:
                    in_flow += flow[(nodeC, nodeA)]
                flow[(nodeA, nodeB)] =
                in_flow/len(f_table[nodeA])
    del flow[('s', list(f_table)[0])]
    del flow[(list(f_table)[-1], 't')]
    return flow
```

On vérifie bien que le flot de sortie est total, la boucle **tant que** s'exécute au plus deux fois comme lors de la première itération les flots de certains liens, nuls à l'initialisation, doivent se propager dans la somme totale lors de la seconde.

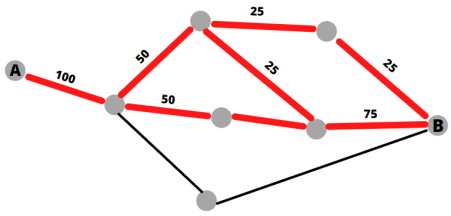

FIGURE 4. Illustration du protocole ECMP

C. L'optimisation

Afin de savoir si un protocole est performant, nous avons comparé la charge maximale sur les arrêtes de notre graphe bi-orienté. Nos deux fonctions python retournent en plus d'un flot un *load*, qui n'est autre que la congestion de l'arrête la plus sollicitée.

En comparant les résultats des deux protocoles, on se rend compte que l'ECMP et le premier modèle sont plutôt similaire, on trouve des *loads* très proches avec plusieurs systèmes de poids.

Avec la première matrice de demande que nous avons utilisé, les *loads* maximaux étaient tous deux de 0.638.

	IGP	**ECMP**
D1	0.638	0.638
D2	0.756	0.545
D3	0.627	0.627
D4	0.435	0.521
D5	0.734	0.734

FIGURE 5. Valeur du *max load* selon différentes demandes

Chaque demande correspond à une entrée non nulle de la matrice D, ou un lien dans le graphe K.

Afin d'avoir un meilleur aperçu des performances de l'ECMP ainsi que motiver l'optimisation des métriques utilisées nous allons à présent regarder la congestion maximale sur l'ensemble du réseau lorsque toutes les demandes sont satisfaites simultanément.

Nous partons dans un premier temps du cas le plus simple où le poids d'un chemin est calculé comme valant le nombre de liens qui le compose. En terme de métriques cela revient à poser

$$w(e) = 1, \quad \forall e \in E.$$

Nous avons alors calculé la congestion maximale satisfaisant l'ensemble des 36 demandes raisonnables données. Il n'est pas surprenant d'observer que ce choix de métrique est loin d'être optimal. En effet si certains liens ne sont absolument pas utilisés, avec un flot nul, certains sont proches d'une congestion de 1 et la congestion maximale dépasse cette valeur. Ainsi la contrainte de capacité n'est pas respectée. Comme nous allons le voir le problème est cependant bien faisable, mieux encore la solution optimale est satisfaisante.

Nous nous proposons alors de chercher à optimiser notre choix de métrique afin de minimiser la congestion lorsque toutes les demandes sont adressées simultanément.

D. Le programme linéaire

Il est possible de formaliser ce problème de minimisation comme un programme linéaire. Soit K notre ensemble de demandes, $G = (V, E)$ notre graphe d'étude. Pour chaque demande $k \in K$ on note s^k le noeud source et t^k le noeud puits, d^k le volume de la demande et pour $e \in E$ on note f_e^k le flot à travers e satisfaisant la demande k.

$$\begin{aligned}
\min \quad & \mathbf{cong}(f, K) \\
\text{s.c.} \quad & \sum_{e \in \mathrm{IN}(t^k)} f_e^k = d^k, && \forall k \in K \\
& \sum_{e \in \mathrm{OUT}(s^k)} f_e^k = d^k, && \forall k \in K \\
& \sum_{k \in K} f_e^k \leq c(e), && \forall e \in E \\
& \sum_{e \in \mathrm{OUT}(v)} f_e^k = \sum_{e \in \mathrm{IN}(v)} f_e^k && \forall k \in K, \forall v \in V \\
& f_e^k \geq 0 && \forall k \in K, \forall e \in E
\end{aligned}$$

Les deux premières contraintes modélisent la résolution de la demande, le routage solution doit satisfaire chaque demande. La troisième est une contrainte de capacité, le routage solution ne peut excéder la capacité des liens, ce que nous observions déjà avec la métrique uniforme. La quatrième est la

contrainte de conservation de flot énoncée précédemment et la dernière impose des flots positifs, un flot négatif n'ayant pas de sens dans le contexte des Télécoms.

On préfère réécrire le programme comme

$$\begin{array}{ll} \min & Z \\ \text{sous contrainte} & \textbf{cong}(f, K) \leq Z \end{array}$$

tout en gardant les contraintes précédentes.

Du point de vue implémentation on retrouve la formulation de ces contraintes dans l'extrait de code python suivant.

```
model = ConcreteModel()
model.flow  = Var(model.index2,
domain=NonNegativeReals)
model.flowConserv = Constraint(model.index1,
rule=flowConservation)
model.linkCapa = Constraint(model.links,
rule=linkCapacity)
```

Comme mentionné précédemment nous utilisons la librairie `Pyomo` afin de résoudre ce problème de minimisation. Nous avons choisi ici d'utiliser le solveur `glpk`, un solveur open source faisant partie du *GNU project*. Il peut résoudre des programmes linéaires en utilisant les algorithmes du Simplexe primal et dual ainsi que des méthodes de point intérieur ce qui le rend polyvalent. Nous avons établi notre choix basé sur une étude comparative de Sandia National Laboratories [4].

Les résultats sont cette fois satisfaisants. Avec les poids $w(e)$ optimisés on obtient une congestion maximale de 0.68 et une répartition raisonnable des flots sur l'ensemble du réseau. Il s'agit d'après le solveur de la solution optimale. Bien qu'il s'agisse d'un réseau test simplifié le temps de calcul sur un ordinateur commun est très faible, 0.39 secondes. La mise en place de cette optimisation semble donc toujours raisonnable à plus grande échelle.

Comme le montrent les figures 6 et 7 la répartition du flot est plus uniforme dans la solution optimale que dans le cas d'une métrique constante égale à 1 et la valeur maximale est moindre.

E. Ajout d'une contrainte temporelle

Afin de compléter l'étude de l'optimisation des métriques du réseau nous nous proposons d'ajouter une

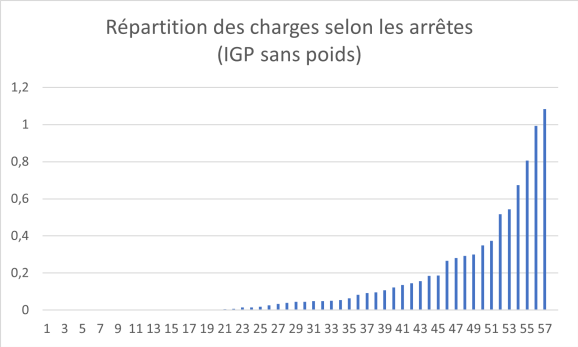

FIGURE 6. Charge exprimée en fraction de 1

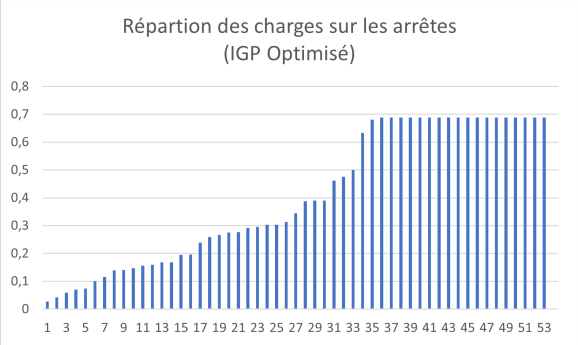

FIGURE 7. Charge exprimée en fraction de 1

nouvelle contrainte, cette fois temporelle.

$$\sum_{e \in E} f_e^k * \delta(e) \leq T_{max} \quad \forall k \in K,$$

où T_{max} est une constante positive symbolisant le temps maximal que peut prendre un paquet avant d'être délivré. $\delta(e)$ est proportionnel à la distance physique entre les deux noeuds du lien e à une normalisation près. La partie gauche est une relation qui nous paraît raisonnable : plus le flot sur un lien est élevé et plus son transit sera long et de même pour la distance séparant ses extrémités.

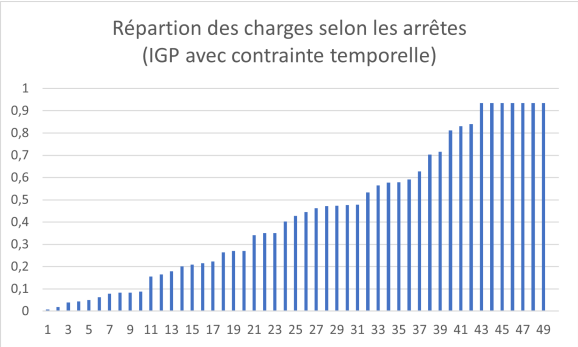

FIGURE 8. Charge exprimée en fraction de 1

Sur python cette contrainte s'écrit

```
model.time = Constraint(expr=sum(
distances[(u,v)]*model.flow[u,v,o,d]
for (u,v) in distances.keys()
for (o,d) in  model.demands) <= T_max)
```

Pour des valeurs de T_{max} trop faibles, conditionnellement aux valeurs choisies des $\delta(e)$, la contrainte est trop restrictive et le programme n'est pas faisable. Pour des valeurs de T_{max} trop élevées la contrainte ne mord pas et n'affecte pas le programme, on retrouve les résultats précédents. Les valeurs les plus intéressantes sont celles entre deux. On observe un résultat intermédiaire entre le routage sans poids, figure 6 et le routage optimal précédent, figure 7. On trouve ainsi par exemple pour $T_{max} = 7000$ lorsque $\delta(e)$ vaut précisément la norme euclidienne de e une congestion maximale de 0.93. Le solveur `glpk` nous indique qu'il s'agit une fois encore de la solution optimale. La figure 8 illustre parfaitement cette idée de transition entre les deux états.

Bien que la contrainte soit purement idéalisée, s'il semble raisonnable pour l'administrateur réseau de vouloir prendre en compte le temps maximal de parcours dans la minimisation de la charge totale ce dernier doit cependant être prudent. Une contrainte temporelle trop forte donnera une congestion maximale plus élevée, voir un programme non faisable. Il faut toutefois noter que si l'effet de la contrainte temporelle joue un rôle important dans notre simulation ce n'est pas nécessairement le cas dans la réalité. Le choix de $\delta(e)$ est discutable, la longueur d'un lien joue-t-elle vraiment un rôle si important lorsque les paquets transitent à la vitesse de la lumière dans de la fibre optique ? On pourrait par exemple penser à une détermination différente de $\delta(e)$, mettant en jeu la qualité du support physique du lien, l'espérance d'aléas pouvant entraîner le ralentissement voir l'interruption du trafic ou encore la priorité dont dispose l'opérateur vis à vis de ses concurrents sur chaque lien e du réseau. Si le choix du facteur f_e^k semble raisonnable en plus d'être pratique il nous faudrait d'avantage de données sur de vrais réseaux pour estimer sa pertinence.

F. Une application possible à des réseaux de taille conséquente

Comme mentionné précédemment les résultats que nous avons obtenus à plus petite échelle semblent suggérer qu'un tel protocole soit applicable à l'échelle d'un réseau Télécom international. Nous avons voulu en savoir plus. Au prix de quelques concessions et d'une perte relative d'efficacité il est possible de trouver un algorithme résolvant de façon satisfaisante le problème en temps polynomial.

La considération individuelle de chaque demande était d'une part dans notre étude une source importante de calculs. L'algorithme de Dijkstra opère déjà en $\mathcal{O}(m + n \log n)$ puis la complexité de notre implémentation peut devenir importante avec le nombre de plus courts chemins. Il semble de plus peu réaliste de redéfinir à chaque instant la métrique du réseau selon les dernières demandes, lesquelles peuvent varier en nombre comme en intensité et, ne parviennent pas de façon discrète et périodique mais bien continue. Nous définissons alors une nouvelle fonction objectif à minimiser, *l'oblivious performance ratio* définie dans [5].

$$\textbf{obliv-perf-ratio}(f) = \sup_D \frac{\textbf{cong}(f, D)}{\textbf{opt}(D)},$$

où le supremum est pris sur l'ensemble des matrices de demande D et $\textbf{opt}(f, D)$ désigne la valeur minimale de la congestion relativement à la demande D obtenue grâce au programme de la partie précédente. On définit alors pour un graphe donné son ratio optimal

$$\textbf{obliv-opt}(G) = \inf_f \textbf{obliv-perf-ratio}(f).$$

Le terme *oblivious* fait donc référence à l'aspect aléatoire de la demande rencontrée par le réseau une fois sa métrique établie. Un résultat édifiant de [5] est le théorème suivant

Théorème 3.1: Il existe un algorithme en temps polynomial sur le nombre de noeuds n qui, étant donné un réseau G en entrée retourne un routing f tel que

$$\textbf{obliv-perf-ratio}(f) = \textbf{obliv-opt}(G).$$

Il semble très intéressant de minimiser *l'oblivious performance ratio* puisque une faible valeur de ce dernier, proche de 1, implique que pour toute demande D la congestion maximale sur le réseau donnée induite par la métrique établie est proche de sa valeur optimale. Cela permet donc à l'administrateur réseau de calculer rapidement, en temps polynomial, une métrique presque optimale face à n'importe quelle demande. La rapidité

d'exécution de cet algorithme, en temps polynomial, offre une meilleure maintenance dans l'éventualité de pannes réseau, comme la destruction d'un lien entre routeurs. Avant de terminer ce rapport et en lien avec cette notion de flexibilité du réseau nous souhaitons mentionner le *Segment routing*.

IV. Segment Routing

Le segment routing (SR) [1] est une technique de routage qui simplifie l'ingénierie du trafic et la gestion le long du réseau. Il supprime les informations sur l'état du réseau des routeurs et des nœuds de transit dans le réseau et place les informations sur l'état du chemin dans les en-têtes de paquets à un nœud d'entrée.

Comme les informations passent des nœuds de transit au paquet, le routage par segment est très réactif aux changements du réseau, ce qui le rend plus agile et plus flexible que d'autres solutions d'ingénierie du trafic. Les capacités d'ingénierie du trafic permettent à la SR de fournir une qualité de service (QoS) pour les applications et également de cartographier les services du réseau pour les utilisateurs finaux et les applications au fur et à mesure qu'ils traversent le réseau.

A. Comment cela fonctionne ?

Lorsqu'un paquet arrive au nœud d'entrée SR, il est soumis à la politique. Si le paquet remplit les conditions de correspondance pour un chemin SR, le nœud d'entrée SR encapsule le paquet dans un tunnel SR qui traverse un chemin SR, segment par segment.

Chaque segment d'un chemin SR se termine par un nœud d'extrémité de segment. Lorsqu'un paquet arrive à un point d'extrémité de segment, le point d'extrémité examine l'étiquette ou l'en-tête du paquet le plus extérieur pour atteindre le segment correspondant. Il fait ensuite apparaître l'étiquette ou l'en-tête le plus à l'extérieur et fait suivre le paquet au point d'extrémité de segment suivant. Ce processus se poursuit jusqu'à ce que le paquet arrive au point final du segment, qui peut être le nœud de sortie SR.

Lorsqu'un paquet arrive au nœud de sortie SR, ce nœud détermine si le paquet est à la fin de son chemin. Si c'est le cas, le nœud supprime l'information d'en-tête SR et transmet le paquet en fonction de son adresse IP de destination.

Comme les routeurs de transit se contentent de transmettre les paquets sur la base de l'identifiant de segment SR, le SR peut être utilisé pour mettre en correspondance les paquets associés à un utilisateur final ou à une application avec des services de fonction réseau spécifiques. Pour ce faire, il cartographie un chemin vers l'endroit où le service sera appliqué, et fournit des instructions sur le service et des informations supplémentaires sur le chemin de la passerelle de service vers le routeur de sortie du domaine SR.

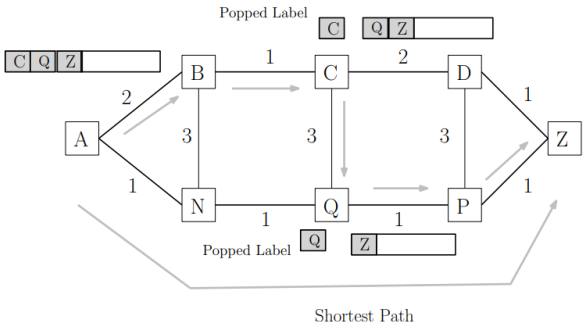

Figure 9. Fonctionnement du segment routing [1]

B. Les avantages

Le principal avantage de SR est sa capacité à simplifier le réseau et à réduire l'utilisation des ressources, ce qui facilite la gestion et l'exploitation de votre réseau.

D'autres avantages rendent le SR souhaitable dans un réseau.

Le SR réduit le nombre de nœuds qui doivent être touchés pour l'approvisionnement et les changements de chemin. Cette action permet au SR d'être plus réactif aux changements du réseau, le rendant plus agile et plus flexible que les autres solutions d'ingénierie du trafic. L'ingénierie du trafic SR fournit une qualité de service aux applications et cartographie les services réseau pour les utilisateurs finaux et les applications lorsqu'ils traversent le réseau. La SR assure la résilience grâce à la restauration de la tête de réseau et à la technologie TI-LFA (*loop-free alternate*) indépendante

de la topologie, qui contribue à la fiabilité des chemins lors des interruptions du réseau (fig. 10).

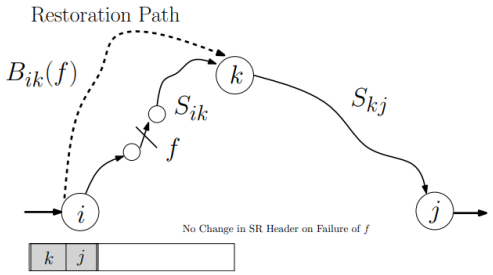

FIGURE 10. Système de restauration des liens defaillants [1]

C. IGP vs SR

Lors de ce projet, nous avons préféré nous concentrer sur l'optimisation des paramètres de routages du protocole IGP. Nous avons seulement survolé le protocole SR afin de comprendre son mode de fonctionnement et ce qu'il peut apporter au domaine du routage. Le protocole SR est néanmoins très intéressant et prometteur, c'est pourquoi nous tenions à en faire mention dans ce rapport.

V. CONCLUSION ET OUVERTURE

Lors de ce projet, nous avons donc pu essayer grâce à des programmes linéaires de minimiser la surcharge d'un grand réseau IP suivant le protocole *IGP*. Il reste encore plusieurs pistes à poursuivre à l'avenir concernant également la résolution de ce problème à l'aide du machine-learning. Une descente de gradient ou Q-Learning pourrait être envisagé. Notamment avec les avancées récentes faites dans le domaine du transport optimal [6]. Cela permettrait alors de se séparer du rôle de l'administrateur qui attribue les poids aux liens. Cependant les réseaux IP sont très sensibles et implémenter un nouveau protocole est très risqué car si le nouveau protocole venait à défaillir, ce serait toute une partie du réseau qui serait paralysé, ce qui donnerait une mauvaise image à l'entreprise de Télécom en charge de ce réseau tout en s'accompagnant d'une perte financière considérable. De plus créer un réseau *test* simulant les demandes réelles est très complexe. Nous possédons donc peu de modèles pour entraîner notre protocole.

RÉFÉRENCES

[1] R. Bhatia, F. Hao, M. Kodialam and T. V. Lakshman, "Optimized network traffic engineering using segment routing," 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, 2015, pp. 657-665, doi : 10.1109/INFOCOM.2015.7218434.

[2] R. Wakikawa, S. Koshiba, K. Uehara and J. Murai, "ORC : optimized route cache management protocol for network mobility," 10th International Conference on Telecommunications, 2003. ICT 2003., Papeete, France, 2003, pp. 1194-1200 vol.2, doi :10.1109/ICTEL.2003.1191606.

[3] B. Fortz, J. Rexford and M. Thorup, "Traffic engineering with traditional IP routing protocols," in IEEE Communications Magazine, vol. 40, no. 10, pp. 118-124, Oct. 2002, doi : 10.1109/MCOM.2002.1039866.

[4] Gearhart, Jared Lee ; Adair, Kristin Lynn ; Durfee, Justin D. ; Jones, Katherine A. ; Martin, Nathaniel Detry, Richard Joseph. Comparison of open-source linear programming solvers., report, October 1, 2013 ; Albuquerque, New Mexico. (https ://digital.library.unt.edu/ark :/67531/metadc868834/ : accessed February 23, 2021), University of North Texas Libraries, UNT Digital Library, https ://digital.library.unt.edu ; crediting UNT Libraries Government Documents Department.

[5] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. 2003. Optimal oblivious routing in polynomial time. In Proceedings of the thirty-fifth annual ACM symposium on Theory of computing (STOC '03). Association for Computing Machinery, New York, NY, USA, 383–388. DOI :https ://doi.org/10.1145/780542.780599

[6] Genevay, Aude, "Entropy-regularized Optimal Transport for Machine Learning", Thèse, 2019-03-13, doi :https ://basepub.dauphine.fr/handle/123456789/20519