UNIVERSITÉ LAVAL

Faculté des sciences et de génie

IFT-2008 :
Algorithmes et structures de données
Travail Pratique #2

Rapport d'Investigation

Présenté à Madame Kim Rioux-Paradis

Fares Majdoub

Automne 2024

1. **Définition de la question à étudier :**

Suite à l'implémentation de l'algorithme Dijkstra pour trouver le plus court chemin dans le graphe proposé dans le cadre du TP2 , il nous a été demandé d'investiguer trois autres algorithmes de plus court chemin dont un non vu en classe pour mieux adapter le routage à ce type de réseau. Il faut se concentrer sur la complexité temporelle et l'adaptabilité au changement et lors de notre investigation pour optimiser la recherche du chemin le plus court et le moins coûteux. Pour ce faire , il faut étudier la performance en fonction des caractéristiques du sous réseau comme la densité, l'absence des cycles et des poids négatifs . Pour ce faire, nous allons d'abord évaluer les caractéristiques du réseau et identifier les critères essentiels à optimiser. Alors , nous allons commencer par la présentation des algorithmes trouvés , ainsi que leurs avantages et leurs inconvénients. En se basant sur cela , nous allons finalement présenter nos conclusions et nos recommandations en fonction de notre analyse.

2. **Démarches d'investigation**

Nous avons réparti notre démarche d'investigation sur deux phases , la première c'est d'identifier la liste d'algorithmes vue dans le cadre de mon programme d'études et la phase 2 est d'appliquer les critères de sélection sur la liste trouvée . Les critères sont : l'efficacité (complexité temporelle) , le fonctionnement sans poids négatif (le poids sur chaque arc est le délai de transmission entre routeurs , un délai ne peut pas être négatif) , la complexité spatiale et la capacité de gérer les cycles pour prévenir les boucles infinies. Dans le cadre du cours *GLO-2000,* nous avons vu plusieurs types d'algorithmes de routage comme **par vecteur de distance** sous son implémentation dans le protocole **RIP**[1] . Étant simple à implémenter, pas très exigeant niveau matériel et résistant au changement de topologie nous choisissons cet algorithme. Dans d'Autres cours nous avons vu les algorithmes qui nécessitent une heuristique pour parcourir les graphes, ce qui est difficile à conceptualiser dans le cas de routage(l'imprédictibilité des changements dans les délais de transmission) donc nous avons exclu cette famille de notre démarche.Dans le cadre du cours *IFT-2008* on a vu l'algorithme **Floyd Warshall** qui calcule le plus court chemin entre toute paire de nœuds dans le graphe , immunisé au boucle infini ,nous avons gardé ce choix. Nous avons aussi vu l'algorithme de **BellMan Ford** mais on peut implémenter sa version modifiée sans poids négatif ,peut être efficace surtout dans les cas de changement de topologie comme à chaque itération l'algorithme peut ajuster l'information, donc nous avons gardé **BellMan Ford sans poids négatifs .** Finalement ,nous avons cette liste : **Floyd Warshall , BellMan Ford sans poids Négatifs et RIP**, que nous allons étudier en profondeur dans la prochaine section pour définir qui est le mieux adapté globalement.

3. **Application de la démarche d'investigation**

Algorithme de Floyd Warshall : Comme vu en classe , cet algorithme est utilisé afin de trouver les plus courts chemins entre toutes les paires de nœuds dans un graphe pondéré. Quant à son fonctionnement, il utilise une matrice de distance , ou chaque cellule *(i,j)* contient la distance entre le nœud i et j , s'il n'y a pas de connection directe , leur distance est initialisée une valeur infinie. L'algorithme se déroule en n étapes, où n est le nombre de nœuds. À chaque étape, pour chaque paire de nœuds*(i,j)*, il vérifie si le passage par un nœud intermédiaire k permet de réduire la distance actuelle entre i et j. Si c'est le cas, la matrice est mise à jour avec cette nouvelle distance plus courte. À la fin de l'itération, nous aurons la matrice finale de distance contenant le plus court chemin entre chaque paire de nœuds.**Complexité temporelle :** Ayant une boucle externe pour les noeuds intermédiaires K et deux boucles internes pour i et j , la complexité temporelle de cet algorithme est de $\theta(n^3)$.**Complexité Spatiale :** Ayant à stocker une matrice de $n \times n$ pour les distance entre les noeuds , la complexité spatiale de cet algorithme est de $\theta(n^2)$ **Avantages :** En offrant la matrice , nous aurions une vue exhaustive des plus courts chemins dans le réseau. En vérifiant les chemins

[1] https://www.digischool.fr/cours/algorithmes-de-routage-dynamique

possibles via chaque nœud intermédiaire *k*, l'algorithme détecte et élimine les cycles indésirables.qui peuvent avoir lieu si on a des composantes fortement connexes dans notre réseau. **Inconvénients :** Une complexité de $\theta(n^3)$ peut être un peu lourd pour des petits sous-réseaux . En plus , même si on peut détecter les pannes et tout recalculer dans les itérations , ça rend son implémentation encore plus lourde. Sa complexité spatiale de $\theta(n^2)$ est limitante pour les routeurs n'ayant pas beaucoup de mémoire.**Cas d'utilisation:** L'algorithme est particulièrement efficace lorsque les changements topologiques sont rares, car il peut fournir un aperçu global des distances sans nécessiter de mises à jour fréquentes qu'il est plus adapté pour l'étude ou l'analyse des performances d'un réseau contrôlé (université ou espace de travail)comme il permet une vue complète des distances entre toutes les paires de nœuds.

Algorithme de Bellman-Ford sans poids négatifs : Partant d'un nœud source unique, cet algorithme est conçu pour trouver le plus court chemin vers tous les autres nœuds à partir de la source. On commence par initialiser la distance de la source à lui-même à 0 et les distances vers tous les autres nœuds à l'infini, car ils ne sont pas encore atteignables.Ensuite, il effectue *n-1* itérations (où *n* est le nombre de nœuds dans le graphe). À chaque itération, l'algorithme passe en revue toutes les arêtes du graphe. Pour chaque arête entre deux nœuds *(i, j)* de poids *k* , il vérifie si passer par le nœud *i* offre un chemin plus court pour atteindre *j* que celui qu'on connaît déjà. Si c'est le cas, il met à jour la distance pour le nœud *j* avec cette nouvelle distance réduite.En répétant ce processus pour chaque arête et en affinant les distances à chaque itération, on s'assure que tous les plus courts chemins depuis la source sont trouvés au bout des *n-1* passages.

Complexité temporelle : Si on pose que *n* est le nombre de noeuds et que *m* est le nombre d'arêtes , l'algorithme aura une complexité de $\theta(n \cdot m)$. **Complexité Spatiale :** Ayant juste besoin de stocker les distances depuis un noeud source , la complexité spatiales est juste $\theta(n)$.**Avantages :** Peu complexe , comme sa complexité dépend du nombre d'arêtes , dans un sous réseau généralement chaque nœud est connecté à relativement peu d'autres nœuds (comme celui du TP) donc cet algorithme est très efficace dans ces situations. Avec la possibilité de recalculer les distances à chaque itération , il est bon pour les environnements de routage dynamique . **Inconvénients:** Si nous travaillons avec un sous réseau dense , avec beaucoup d'arêtes , l'algorithme devient lent et inefficace. **Cas d'utilisation :** Il est utilisé dans les protocoles de routage dynamique comme il permet de recalculer la distance à chaque itération , utile dans les environnement distribués. Bon pour les réseaux à grande échelle et moins denses pour limiter la complexité.

Algorithme par vecteur de distance (Protocole RIP)[2]: Chaque routeur dans le réseau ou RIP est implémenté conserve un vecteur de distance (ou appelé table de routage) , qui indique le nombre de saut nécessaire pour atteindre chaque routeur dans le réseau. Le vecteur est mis à jour en fonction des informations reçues des voisins.Au début , chaque routeur initialise sa table de routage indiquant la distance (nombre de sauts) vers chaque destination connue , toutes les 30 secondes , chaque routeur envoie son vecteur de distance à ses voisins directs . En recevant ce vecteur , le routeur fait ces calculs pour savoir s'il peut atteindre une destination dans un chemin plus court(moins de sauts) . Si c'est le cas , la table de routage est mise à jour . Pour éviter les boucles infinies , on a une limite de 15 sauts et un certain temps à vivre(TTL), après ça un paquet est détruit dans le réseau pour libérer le trafic non nécessaire et pour éviter la corruption du paquet. **Complexité temporelle :** Avec un nombre maximal de 15 sauts seulement , à chaque itération ça nous prends *n* opération pour mettre à jour la table de routage qui nous donne une complexité de $\theta(n)$

Complexité Spatiale : Chaque routeur ne stocke que les les noeuds atteignable en 15 sauts ou moins , donc avec *K* représentant ces derniers , la complexité sera de $\theta(n \cdot K)$,avec *n* le nombre total des

[2] Notes de cours GLO-2000

nœuds et $K < n$. **Avantages :** Simple et léger niveau mémoire et temporelle pour les routeurs , surtout avec des ressources limitées . Il est immune au changement topologique comme il met à jour les vecteurs périodiquement donc il peut s'ajuster . Avec la limite de 15 sauts , on prévient les boucles infinies et avec le TTL on ajoute une autre couche de protection. **Inconvénients :** lenteur causé par l'attente des mises à jours des tables puis le calcul pour les ajuster, si la topologie change souvent , ceci peut causer des problèmes et peut surcharger le trafic.. Les 15 sauts limitent un peu la portée du réseau. **Cas d'utilisation :** pour les LAN , il offre une simplicité et une fiabilité . Il ne consomme pas trop de ressources , parfait pour un équipement avec des capacités limitées.

4. **Analyse des données recueillies**

- **Voici un tableau encapsulant les résultats trouvés dans la section précédente**

	Floyd-Warshall	**Bellman-Ford**	**RIP**
Complexité temporelle	$\theta(n^3)$	$\theta(n \cdot m)$	$\theta(n)$
Complexité Spatiale	$\theta(n^2)$	$\theta(n)$	$\theta(n \cdot K)$
Efficacité face aux changements Topologiques	**Faible**	**Bonne**	**Moyenne**
Efficacité en fonction de la densité	**Bonne**	**Moyenne**	**Faible**
Simplicité de mise en oeuvre	**Complexe**	**Simple**	**Très simple**

Donc selon notre recherche , Floyd Warshall est adapté aux analyses statiques , dans les environnement sables ou les réseaux sont denses ou une vue de tous les chemins est nécessaire. Bellman-Ford est excellent pour les réseaux dynamiques et distribués , il est adapté aux sous réseaux IP avec une topologie qui change souvent . L'algorithme par vecteur de distance (Protocole RIP) est bon pour les sous réseaux de petites à moyenne taille , moins adaptés aux réseaux ou la topologie change souvent comme ça cause des délais mais parfait pour un faible usage de ressources.

Conclusions

Finalement le choix du meilleur algorithme à part Dijkstra et ses sous-jacents dépend vraiment des cas d'utilisation précis ou du besoin spécifique du client. S'il s'agit d'un LAN dans une école primaire , RIP est parfait surtout pour son faible usage de ressources . Si on cherche à étudier tout les chemins possibles dans un sous réseau Floyd-Warshall est parfait mais peut être exigeant pour les ressources vu les complexités temporelle et spatiales élevées. Cependant, si nous devons choisir un juste milieu ça sera Bellman-Ford sans les poids négatifs est parfait pour le réseaux dynamiques , excellent pour les sous-réseaux IP avec une topologie qui change souvent ou chaque noeud n'est pas directement connecté à un grand nombre de voisins.