RESEAUX - HLIN611

Licence L3 Informatique

Anne-Elisabeth Baert - baert@lirmm.fr

24 février 2020

1. Chapitre 6 – Routage
   - Introduction au Routage
   - Algorithmes à Vecteurs de Distances
   - Algorithmes à états de liens
   - Autres algorithmes

2. Chapitre 7 : Grandes Applications – Serveurs de Noms
   - Principes de fonctionnement

Le Problème du Routage

Déterminer en fonction de l'adresse réseau du destinataire final d'un datagramme le prochain destinataire.

On peut compléter très légèrement le tableau déjà vu dans le cas d'un hôte quelconque sur un réseau local.

Destination	Contact	Masque	Interface
201.202.203.0	direct	255.255.255.192	eth0
autre	201.202.203.1	0.0.0.0	eth0

Contact indique soit le prochain routeur, soit une destination sur le même réseau.

Problèmes

- Cette table peut prendre des dimensions gigantesques dans le cas d'un routeur censé connaître l'ensemble des destinations de l'Internet ou d'un sous-ensemble.
- Comment construire cette table ?

Routage Statique

Le routage statique constitue une solution simple à la construction de la table : elle est figée et modifiée uniquement par une intervention d'un administrateur.
Cette solution est parfaitement bien adaptée à un réseau local avec un seul routeur assurant la connectivité vers le monde extérieur.

L'utilisation d'une route par défaut permet de passer rapidement le relai d'un hôte à un routeur, d'un routeur à un autre routeur. On comprend mieux pourquoi des incohérences sont possibles. Et il y a pire, par exemple des boucles...

Routage Dynamique

Dans le cas d'un routeur reliant plusieurs réseaux, un routage adaptatif ou dynamique permettra de tenir compte de :

- l'infrastructure des réseaux connectés,
- l'arrivée et la réparation de pannes, la création de nouveaux liens,
- la charge des réseaux (congestions, oscillations),
- de la qualité de service requise, etc.

Une distribution *intelligente* des adresses de réseaux permettrait de réduire la taille des tables de routage (voir routage hiérarchique). Hélas, ce n'est pas le cas, du moins ceci n'a pas été fait systématiquement, dans l'Internet.

Connaissance Partielle et Erreurs

Constat : Dans tous les cas, le résultat de l'algorithme de routage est l'adresse du *suivant*.
On espère que les routeurs sont cohérents entre eux, c'est-à-dire que le *suivant* peut continuer à acheminer correctement le paquet. Sinon, on aura des **erreurs de routage**.
Ceci reste vrai même si un routeur connaît le chemin complet, car on ne peut pas **forcer** une décision sur un **autre** routeur ; sauf cas spécifiques de tests de chemins, ce serait néfaste de le forcer.

Traitement des Erreurs - Un Début

On peut empêcher un paquet de vivre indéfiniment dans l'Internet :
Principe :

- Un champ *durée de vie (ttl)* est attaché à chaque paquet (cf. entête du paquet IP) ; il est initialisée par l'hôte source du paquet ;
- Chaque routeur décrémente la durée de vie ;
- Le routeur qui arrive à une durée de vie nulle ou négative détruit le paquet.

Actuellement, la durée de vie est mesurée en *nombre de routeurs traversés*, dit aussi *nombre de sauts*.

---

ttl- - ;
si *(ttl* $>$ *0)* **alors** router paquet ;
;
sinon expédier (erreur routage) à hôte source ;
;

Classes d'Algorithmes

Plusieurs classes d'algorithmes dynamiques existent en fonction de l'étendue des réseaux reliés.

Globalement, on a besoin de trouver des chemins dans un graphe dynamique. Il faut se poser les questions de

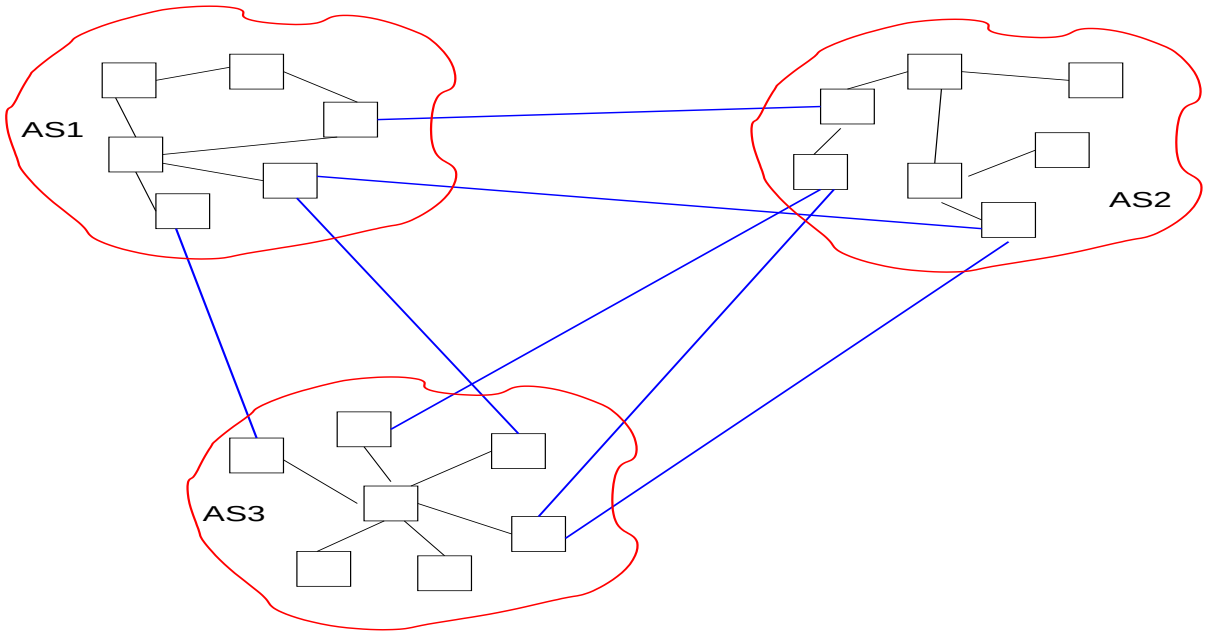
- l'efficacités des algorithmes distribués ou centralisés ;
- convergence, stabilité et cohérence des algorithmes.

Pourquoi plusieurs classes d'algorithmes ? Parce qu'on ne peut plus se contenter d'une organisation centralisée.

Organisation en Systèmes Autonomes de l'Internet

Aujourd'hui, l'Internet est organisé en *Systèmes Autonomes* (AS ci-après), vastes réseaux (en général), administrés chacun par une entité unique.

Chaque AS possède des routeurs *intérieurs* reliant les sous-réseaux entre eux, et des routeurs *extérieurs* reliés à des routeurs extérieurs d'autres AS.



Organisation du Routage dans l'Internet

Chaque AS organise son propre routage interne librement. Plusieurs algorithmes sont connus : RIP, OSPF.

Pour la partie externe, il faut un protocole commun.
Les AS communiquent entre eux par un seul algorithme lié à un seul protocole : aujourd'hui, BGP.

Les routeurs *extérieurs* d'un AS doivent connaître **toutes** les adresses des autres AS afin de constituer un routage cohérent. On insiste sur **toutes**, pas seulement celles des AS et routeurs adjacents.

On pourra voir qu'il est difficile de concilier le fonctionnement extérieur, visible, avec les choix politiques internes. C'est un sujet de recherches actuellement.

Algorithme à Vecteur de Distance RIP

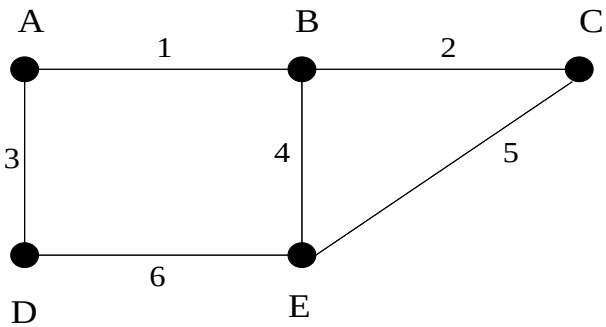
Principes :

- Diffusion d'informations sur le routage à base de la **distance** ; quelle métrique pour exprimer une distance ? le plus fréquent : *nombre de sauts* ;
- Chaque routeur dispose d'une table contenant des triplets (*destination*, *numéro_de_liaison*, *coût*) c'est-à-dire pour telle destination, envoyer sur telle liaison pour tel coût ;
- La table est mise à jour dynamiquement ; il y a diffusion périodique d'informations (*destination*, *coût*)
- Chaque routeur qui reçoit une information la compare au contenu courant de sa table ;
- Si l'information est *meilleure* il la prend ;
- Sinon, il y a des cas où l'on est obligé d'accepter une information fut-elle *moins bonne*, d'autres où on la rejettera.

Algorithme RIP

Données : table de routage ; des doublets $(dest, cout)$ reçus sur une liaison l_{recue}
Résultat : table de routage
pour *toutes les données arrivant* **faire**
  si $dest_{recue}$ *trouvée dans table* **alors**
    si $l_{recue} == l_{table}$ **alors**
      $c_{table} = c_{recue} + 1$;
    sinon
      si $c_{table} > (c_{recue} + 1)$ **alors**
        $l_{table} = l_{recue}$;
        $c_{table} = c_{recue} + 1$;
  sinon
    ajouter $(d_{recue}, l_{recue}, c_{recue} + 1)$ dans table

Vecteurs de Distances - Exemple



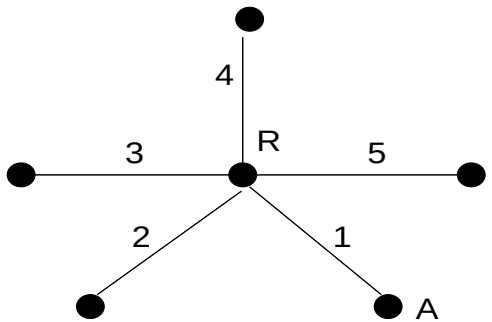
Extrait de *Le routage dans l'Internet*.

Initialisation ;
A diffuse $(A, 0)$ sur les liaisons 1 et 3 ;
B diffuse $(A, 1), (B, 0)$ sur 1, 2 et 4 ;
information perdue sur 4 ;
C diffuse $(C, 0), (A, 2), (B, 1)$ sur 2 et 5 ;
D diffuse $(D, 0), (A, 1)$ sur 3 et 6.

A			B			C			D			E		
→	l	c	→	l	c	→	l	c	→	l	c	→	l	c
A	loc	0	B	loc	0	C	loc	0	D	loc	0	E	loc	0
			A	1	1				A	3	1			
B	1	1				A	2	2						
						B	2	1						
			C	2	1							C	5	1
												A	5	3
												A	6	2

Justifications

Traitement global : Chaque routeur diffuse périodiquement sa table, ensemble de couples (*destination*, *coût*) vers ses voisins adjacents.
Important : Si une nouvelle information sur une destination arrive par la **même** liaison que celle par laquelle on route, alors il faut la prendre, qu'elle soit bonne ou mauvaise.
Pourquoi ? Considérer le point de vue d'un routeur.



Si *A* annonce à *R* qu'une destination *X* est atteinte pour un coût 10 et que le même, *A* annonce ensuite un coût différent (envisager 7 puis 12) pour la même destination, *R* doit accepter ce nouveau coût, sauf si entre temps il a un meilleur chemin.

Remarques

Attention : Nous avons vu **une** simulation possible de la diffusion. L'ordre de diffusion peut être totalement différent, avec des résultats différents sur les tables.

On peut démontrer la convergence de cet algorithme si aucune modification (panne, apparition de nouvelles liaisons) n'arrive. Mais les pannes et modifications sont **fréquentes**. On peut montrer que pour toute panne ou apparition de liaison, l'algorithme converge si un nouvel incident n'a pas lieu avant l'aboutissement de la convergence.

Le défaut de fonctionnement reproché à RIP est résumé ainsi :
Les bonnes nouvelles se propagent vite, les mauvaises se propagent doucement.

Traitement d'une Panne

Principe du traitement d'une panne : annoncer un coût ∞ pour chaque voisin en panne. Cette détection peut se faire par un outil comme `ping` permettant de tester l'existence d'un hôte.

Exemple : Supposons que la liaison 1 tombe en panne. *B* corrige sa table avec le triplet $(A, 1, \infty)$ et la diffuse.

Selon l'ordre de propagation de l'information, on peut constater une convergence rapide, ou des phénomènes connus sous le nom de *comtage à l'infini* :
Supposons que *C* diffuse sa table, **avant** que *B* ne diffuse la sienne. *C* diffuse entre autres informations $(A, 2)$, qui lors du traitement sur *B* va engendrer le triplet $(A, 2, 3)$! ! !

D'autres défauts de fonctionnement de cet algorithme ont été répertoriés (voir bibliographie), accompagnés de solutions plus ou moins heureuses. Elles font l'objet de cours ultérieurs.

Algorithmes à États de Liens - Principes

- Chaque routeur possède la topologie complète du réseau ;
- Deux tâches sont accomplies pour arriver à connaître cette topologie :
  - test d'activité de tous les routeurs adjacents : échanges courts (type `ping`)
  - diffusion périodique de l'état des liens : c'est un compte-rendu des communications possibles. L'état est rediffusée *autant que nécessaire* à tous les routeurs participants, avec horodatage (numéro de séquence) des messages.
- Chaque routeur calcule un plus court chemin vers chacun des autres routeurs.

Exemple : *Open Shortest Path First* (OSPF), est actuellement un algorithme à état de liaisons très utilisé à l'intérieur des systèmes autonomes de l'Internet.

Algorithmes à états de liens - Algo

Données : ensembles (*voisins*, *couts*, *séquence*) ; N : ensemble de nœuds dont le pcc est connu ; $D(v)$: coût parcours vers v

Résultat : nouvelle topologie et arbre des plus courts chemins

```
//Initialisation N={A} ; D(v) infini (tous);
pour tous les nœuds v faire
    si v adjacent alors
        D(v)=coût(A,v);
//mise à jour
répéter
    trouver w extérieur à N tel que D(w) min;
    ajouter w à N;
    pour tout v adjacent à w faire
        D(v)=min(D(v),D(w)+c(W,v));
jusqu'à tous les nœuds v explorés;
```

Autres algorithmes

Ces quelques lignes juste pour dire qu'il existe d'autres types d'algorithmes et d'autres raffinements. Par exemple un algorithme à *état de chemins* (BGP), est utilisé entre systèmes autonomes. Pour d'autres développements voir la bibliographie et la suite de ce cours.

2 Chapitre 7 : Grandes Applications – Serveurs de Noms

- Principes de fonctionnement

Serveurs de Noms - Petit Retour

DNS

L'application *Serveurs de Noms* est aujourd'hui une base fondamentale de l'Internet.

Elle n'est pas seulement utilisée pour la correspondance nom↔adresse des hôtes, mais plus généralement, pour enregistrer des informations d'administration.

RFC

RFC1034 décrit les concepts de base. Suivent un tas de compléments et mises à jour.

Exemple

La messagerie électronique utilise les serveurs de noms pour trouver l'hôte à contacter pour l'acheminement des courriels sur un site.

En effet, que l'adresse électronique d'une personne comporte ou non un nom d'hôte, il y a souvent un (ou quelques) hôte(s) fixé(s) pour l'acheminement dans le domaine de destination.

L'application *serveur de noms* ou DNS, permet de déterminer en fonction d'un nom d'hôte ou du nom de domaine de l'adresse électronique, le serveur à contacter pour la messagerie.

Principes de Fonctionnement

Déjà vu :

- Les conventions de nommage.
- Le fonctionnement de base, consistant à avoir un serveur de noms dans le domaine local, contenant la correspondance noms$\leftrightarrow$adresse des hôtes locaux.
- La requête est acheminée vers un autre serveur (souvent un serveur racine), qui est capable de la refaire suivre.
- Le serveur qui connaît la correspondance répond directement au demandeur.

La partie du système d'exploitation prenant en charge la résolution s'appelle *resolver*. Cet outil existe forcément sur chaque hôte, consulte un fichier de base (*/etc/resolv.conf*), contenant au moins :

- le domaine dans lequel il faut chercher les noms simples (ceux donnés sans le caractère point) ;
- l'adresse IP d'au moins un serveur de noms, en général le serveur de noms local.

Exemple

```
search info.rmatique.fr
nameserver 123.231.111.12
```

- Veut dire que tout nom simple, par exemple *hotel* sera recherché (complété) en tant que *hotel.info.rmatique.fr*. Ensuite, ayant le nom complet, la requête sera expédiée vers 123.231.111.12.
- Le serveur de noms reçoit alors un nom interne ou externe et réagit en fonction de ce qu'il trouve dans sa base.
- S'il ne trouve rien, il doit posséder l'adresse d'au moins un autre serveur de noms (souvent un serveur racine) afin de faire suivre la requête.
- Noter que les requêtes aux serveurs de noms utilisent udp.
- Noter aussi qu'il est indispensable d'avoir l'adresse IP du serveur de noms !

Remarques

- Il est possible de gérer des sous-domaines séparément, c'est-à-dire en fait avoir dans un même domaine, géré par une même autorité administrative, plusieurs serveurs de noms relatifs à plusieurs domaines. Plus généralement, on parlera de *zone d'autorité* pour l'unité gérée par un serveur donné. Mais, un serveur de noms peut gérer plusieurs zones...
- Il est possible aussi d'avoir plusieurs serveurs pour une même zone : penser aux pannes en particulier. Dans ce cas, il y a un serveur dit *primaire* disposant de l'information **origine** et des serveurs *secondaires* disposant d'une **copie**.

Remarques

- Les requêtes sont vulnérables et DNSSEC décrit dans la RFC2535 permet de construire une *chaine de confiance*.
- La mise à jour dynamique (RFC2136) devient indispensable avec la distribution dynamique des adresses avec des logiciels comme DHCP par exemple.

Plus loin

- Chaque serveur gère des informations comme la correspondance *inverse* (obtention d'un nom à partir d'une adresse), le contact messagerie, la durée de vie des informations, les noms multiples d'un même hôte, etc.
- L'application *serveur* s'appelle *named*, qui lit un fichier de configuration de démarrage (*named.conf*).

Plusieurs bases contiennent ensuite toutes les autres informations. La localisation de ces bases est donnée dans le fichier de démarrage.

De même, les serveurs secondaires ou primaires sont nommés dans ce fichier de démarrage.

Conclusion : le point d'entrée pour aller plus loin est : *named* en plus de DNS.