Notes de cours (ENS Lyon, M1)

Chapitre 1: Routage dans les réseaux

Contents

Introduction générale

Objectifs du cours et présentation du plan général et de l'organisation.

Cours d'**algorithmique**: algorithmes mis en oeuvre dans les réseau, notions assez théoriques. Notamment, un chapitre sur le routage dans les réseaux: étudier quelques algorithmes de routage classiques, leur caractéristiques et introduction de la notion de routage *optimal*.

Aspect réseaux et télécoms: plutôt axé plateformes nouvelle technologie, les réseaux pair à pair, puis les réseaux ad-hoc et sans fil. Algorithmes que l'on utilise pour le routage sur de telles plateformes, algorithmes de broadcast et multicast pour les réseaux sans fil.

Dernière partie sur Internet et le Web, où l'on s'intéressera aux problèmes algorithmiques qui se posent dans ce contexte.

Biblio

La première partie du cours est très proche des problématiques des réseaux, et on peut trouver de la lecture sur ce sujet dans les livres de réseaux classiques (pour le routage dans les réseaux), que vous pouvez trouver à la bibliothèque de l'ENS.

- Andrew Tanenbaum. Réseaux.
- Mischa Schwartz. Telecommunication Networks – Protocols, Modeling and Analysis.
- G. Pujolle, D. Seret, D. Dromard, E. Horlait. Réseaux et Télématique.
- D. Cali, G. Zany. Technologie de l'interconnexion de réseaux.

Les algorithmes pour les plateformes nouvelle technologie sont vraiment des travaux d'actualité, beaucoup de recherche en cours sur le domaine, conférences spécialisées. Lectures et informations en recherchant sur internet des articles récents sur le domaine.

Quelques livres sur les premiers protocoles de pair à pair:

- Andy Oram. Peer-to-Peer: Harnessing the power of disruptive technologies. O'Reilly, 2001.
- F. Berman, G.C. Fox, A.J.G. Hey. Grid computing: Making the Global Infrastructure a Reality. Wiley, 2003.

1 Routage dans les réseaux

1.1 Introduction

1.1.1 Motivation

Principe

Trouver sa route dans un réseau d'entités interconnectées.

Pourquoi?

On dispose souvent de réseaux locaux de types différents (dans une université, une entreprise, ...), mis en place dans divers départements de façon indépendante. Et les usagers des différents réseaux ont besoin d'échanger des données.
Avantage à garder des sous-réseaux physiquement disjoints mais communicants:

- limites du réseau local repoussées
- échanges lents si le réseau est trop étendu
- le risque de collisions sur le réseau augmente avec le nombre de stations reliées
- perturbations au sein d'un sous-réseau limitées au sous-réseau
- problèmes de sécurité et confidentialité adapté à chaque sous-réseau

Comment?

Routage: technique basée sur des adresses de niveau réseau, qui permettent d'aiguiller une trame quelconque émise par un noeud d'un sous-réseau vers un noeud de destination pouvant être situé sur un autre sous-réseau.

Notion de Routeur = élément matériel (et logiciel) permettant d'effectuer la tâche de routage.

Exemple de graphe et exemple de routage: comment aller d'un noeud N_1 à un noeud N_k? (dessiner un graphe).

- N_1 demande comment aller à N_k et on lui répond N_2
- N_2 demande comment aller à N_k et on lui répond N_3
- et ainsi de suite jusqu'à trouver N_k.

De quoi a-t-on besoin localement pour prendre une décision de routage? (calculer la route). Doit on connaître tout le réseau pour pouvoir router?

Problème complexe car distribué à la fois dans le temps et dans l'espace. Calcul distribué de la route: on ne connaît pas à priori la route pour aller en N_k et ce n'est pas un problème. Plusieurs entités permettent de calculer la route. N_1 ne connaît pas nécessairement toute la route, mais au moins le début.
Comment prendre la décision? Dépend des infos locales uniquement:

- ma propre identité
- entête du message
- port d'entrée du message (chaque entité a plusieurs ports)
- informations locales

Dynamicité

Importance des problèmes de reconfiguration des réseaux en cas de panne ou de congestion, fortement lié aux algorithmes de routage. Algorithmes de routage statiques ou dynamiques (adaptatifs).

- Algorithmes adaptatifs difficiles à contrôler.
- Le réseau doit réagir promptement à la moindre défaillance.
- Routage centralisé souvent à exclure (long temps de réponse).
- Adaptation dans l'espace et dans le temps: chaque noeud doit avoir une connaissance aussi complète que possible de l'état global du réseau, mais il faut limiter au maximum l'échange d'informations.

1.1.2 Métriques pour déterminer la performance d'un routage

But du routage: acheminer un paquet de l'émetteur jusqu'au récepteur, en utilisant si possible la meilleure route.

Différentes métriques à considérer: nombre de sauts, capacité des liens, trafic, ...

Une stratégie de routage doit alors être un compromis entre facilité d'administration, robustesse, faisabilité... Comment évaluer la complexité et la performance du routage?

Mesure des performances: critères de délai, débit, coût.

Possibilité de mesures dynamiques: elles ont l'avantage de tenir compte de la charge du réseau. Cependant, cela risque de migrer continuellement la charge entre deux lignes, sans jamais utiliser les deux (on utilise une des lignes, et quand elle devient chargée, on utilise l'autre car elle donne un meilleur routage...) (on y reviendra en présentant l'algo de routage par état de liaison).

On désire que le routage ait les propriétés suivantes:

- Exactitude
- Simplicité
- Robustesse - tolérance vis à vis des défaillances de tous types qui peuvent survenir pendant la durée de vie du réseau: défaillances matériel et logiciel, changement de topologie... L'algorithme de routage doit prendre en compte les modifications de topologie/trafic sans devoir arrêter tous les ordinateurs et réinitialiser le réseau. On retrouve la notion de dynamicité citée plus haut.
- Stabilité - L'algorithme de routage doit converger vers un équilibre.
- Justice (vis à vis des usagers) - satisfaire tous les usagers...
- Optimisation - contradictoire avec la justice par moment, il faut décider ce que l'on veut optimiser et trouver un compromis entre justice (pour chaque connexion individuelle) et optimisation (efficacité globale de ce routage).

Solution de compromis, entre minimiser le délai moyen de traversée des paquets (justice) et maximiser le flux total du réseau (optimisation), consiste à minimiser le nombre de sauts (noeuds traversés) qu'un paquet doit faire, ce qui améliore les délais et réduit la capacité de transport consommée (et donc tend à améliorer le débit). D'où la notion de *facteur d'étirement*.

Facteur d'étirement

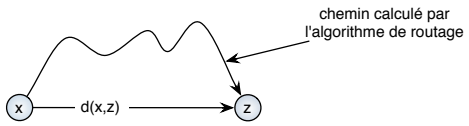
Représentation du graphe du réseau: $G = (V, E)$.

- Graphe connexe
- A priori, non orienté
- Statique pour commencer (topologie fixée)
- Poids des arêtes = temps pour transférer un message de taille élémentaire. $w : E \rightarrow Q_*^+$

On définit ensuite une fonction de coût d qui définit une métrique, permettant d'évaluer la qualité d'un algo de routage. Par exemple,

$$d(x, y) = \min_{chemin} \sum_{e \in chemin} w(e)$$

C'est une métrique: $d(x, z) \leq d(x, y) + d(y, z)$. Chemin possible de x à z en passant par y, de coût $d(x, y) + d(y, z)$.



Le chemin calculé par l'algorithme de routage a un coût $\rho(x, z)$.

Comment définir le facteur d'étirement?

$$s(x, z) = \frac{\rho(x, z)}{d(x, z)}$$

En gros, on cherche à comparer la longueur du chemin calculé par notre algorithme de routage à la "distance" entre les deux noeuds.

C'est un facteur de performance: trouver un algorithme de routage avec $s = 1$? C'est possible en calculant le chemin de poids minimal pour chaque couple de sommet.

Par la suite on présente plusieurs algos classiques, à la recherche d'un routage optimal. Mais avant, une petite réflexion sur le type de routage...

1.1.3 Type de routage

On peut distinguer deux grandes catégories de routage différentes, suivant que l'on désire établir une connexion de type téléphonique, ou bien si notre réseau fonctionne plutôt comme la poste. Je présente brièvement les 2 approches et je les compare.

Etablir un circuit virtuel

Orientation service avec connexion, pour éviter d'avoir à prendre une décision de routage pour chaque paquet. Lorsqu'une connexion est établie, on choisit

la route entre la source et la destination, puis on la mémorise. Cette route est ensuite utilisée pour tout le trafic qui arrive. C'est l'équivalent du téléphone, lorsque l'on raccroche la connexion est rompue et la route coupée. Une route est appelée un *circuit virtuel*.

Chaque routeur doit tenir à jour une table avec les informations sur comment router pour chaque circuit virtuel, et les paquets doivent contenir le numéro du circuit virtuel. Quelques issues dans la numérotation... Pour les détails cf Tannenbaum.

Sous-réseau datagramme

Par opposition aux circuits virtuels, aucune route n'est établie à l'avance, et chaque paquet est routé de manière indépendante de son prédécesseur. Des paquets successifs peuvent suivre une route différente. C'est un peu l'équivalent du système postal.

Plus de travail dans l'algorithme (prise de décision pour chaque paquet), mais algorithmes plus robustes qui s'adaptent mieux aux défaillances et aux congestions.

Un paquet datagramme doit contenir l'adresse complète de la destination (alors que le paquet de circuit virtuel ne contient que le numéro de circuit).

Comparaison des deux approches

Chaque technique a des avantages/inconvénients, dépend du réseau et du type de trafic... CV: circuit virtuel et DG: datagramme.

- Etablissement d'une connexion: nécessaire uniquement dans le cas du CV
- Adressage: chaque paquet contient les adresses complètes de la source et de la destination pour DG; numéro de circuit pour CV.
- Informations de routage: aucune information du routage des paquets pour DG; pour CV, chaque circuit requiert de la place dans les tables de routage.
- Routage: route initialisée pour CV puis chaque paquet suit cette route; route indépendante pour chaque paquet DG.
- Défaillance d'un routeur: aucune conséquence pour DG; tous les circuits virtuels traversant l'équipement défaillant sont détruits.
- Contrôle de congestion: difficile et complexe pour DG; pour CV c'est facile si l'on dispose de suffisamment de mémoire à l'établissement du CV.

Conclusion et intro de la suite

Objectif de la couche réseau: router les paquets de la machine source à la machine destinataire. Sous-réseau: les paquets accomplissent de nombreux sauts pour atteindre la destination. Réseau sans fil: besoin également de routage si la source et le destinataire ne sont pas dans le même réseau.

Algorithme de routage: décide sur quelle ligne de sortie un paquet entrant doit être retransmis. Sous-réseau datagramme: cette décision est prise pour chaque paquet entrant. Circuits virtuels: décision de routage prise une seule

fois, lors de l'établissement du circuit, puis les paquets suivant ne font que prendre la route établie (routage de session).

Prochaine section: on présente différents algorithmes de routages et on analyse les différentes techniques et leur complexité. On essaie de définir ce qui correspondrait à un routage *optimal* à travers ces algorithmes.

1.2 Notion de routage optimal

1.2.1 Algos centralisés

Il existe des algos de routage **centralisés**: le chemin est calculé par un routeur central. Ce type d'algos présente l'avantage d'être simple, mais il est peu fiable et vulnérable. En cas de défaillance, le routeur central recalcule les tables de routage et renvoie la nouvelle information à chaque noeud. Il est également possible de mettre à jour à certains instants précis les tables de routage, suivant l'état des noeuds (et les noeuds envoient régulièrement un compte-rendu de leur état). Enfin, une troisième variante consiste à procéder aux adaptations de façon asynchrone: un noeud envoie un rectificatif sur son état quand il a beaucoup changé, et le routeur central envoie des nouvelles tables de routage quand la différence est significative.

Les performances de ces méthodes dépendent de la topologie du réseau et du trafic. Il faut que l'adaptation puisse se faire en temps réel. De plus, proposer un algorithme plus sophistiqué comme la 3ème variante peut entraîner une surcharge du réseau par des paquets de contrôle, ce qui peut empêcher le fonctionnement en temps réel. Pourtant, la 3ème variante semble être la meilleure. Les performances relatives de ces algorithmes vont vraiment dépendre.

Nous nous intéressons maintenant aux techniques **distribuées**. Dans ce cas, le chemin est élaboré par chaque routeur. Plusieurs stratégies existent:

1.2.2 Routage par inondation (*flooding*)

Cet algorithme de routage consiste à renvoyer tout paquet reçu à tous ses voisins, mais à éviter d'envoyer deux fois le même paquet. Ceci est généralement contrôlé grâce à un compteur de sauts placé dans l'entête du paquet. Compteur initialisé avec la longueur du chemin séparant la source du destinataire. Si cette longueur est inconnu, plus grand diamètre du réseau.

Inondation sélective: les paquets sont envoyés sur toutes les lignes qui sont approximativement dans la bonne direction (expédier à gauche ce qui vient de la droite...) sauf si la topologie est très particulière.

Algorithme simple, s'il existe un chemin alors le paquet arrive à destination, et le paquet est livré le plus vite possible. Seulement, cela augmente la charge du réseau de façon considérable. A noter que cet algorithme n'est pas adaptatif, c'est un algorithme statique. Algorithme rarement utilisé, mais peut s'avérer intéressant dans certains cas pour sa grande robustesse. Étirement $s = 1$ vu que l'on emprunte le plus court chemin parmi tous les chemins essayés.

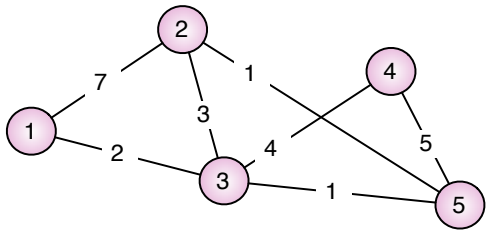
1.2.3 Calcul du plus court chemin

Un autre algorithme statique permettant d'obtenir un étirement de 1 consiste à calculer les plus court chemins $d(x, y)$ pour chaque couple de noeuds du graphe. Un tel algorithme de routage se ramène donc à un problème de calcul de plus court chemin.

Différentes métriques envisageables pour les poids des arcs: nombre de sauts, distance géométrique, temps moyen d'attente et de transmission, ... En général, poids = fonction pondérée de différents facteurs, et on peut modifier la fonction de poids pour adapter l'algorithme à divers facteurs d'optimisation.

Utilisation par exemple de Dijkstra - rappel du cours d'algo L3. Tous les poids sont positifs. Au début, aucun chemin connu, noeuds étiquetés avec la valeur ∞. Etiquettes modifiées au fur et à mesure que l'algorithme progresse, indiquant le meilleur chemin. A chaque étape, le sommet avec la plus petite étiquette est décrété étiqueté de façon permanente et il ne sera plus modifié. Nous repartons de ce noeud pour modifier ses voisins de façon temporaire, et examinons ses voisins.

Exemple sur ce graphe:



Etapes de l'algorithme:

étape / i	1	2	3	4	5
0	(0,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)
1	.	(7,1)	(2,1)	(∞,-)	(∞,-)
2	.	(5,3)	.	(6,3)	(3,3)
3	.	(4,5)	.	(6,3)	.
4	.	.	.	(6,3)	.
d(1,i)	0	4	2	6	3

Complexité en $O(n^2)$ pour un sommet, donc $O(n^3)$ en tout, sous réserve que tous les poids soient positifs ou nuls.

Remarque: pour faire le routage, on considère le problème *à l'envers* car on a besoin de connaître le successeur plutôt que le prédécesseur sur un plus court chemin. On inverse donc la source et la destination, et vu que le graphe n'est pas orienté, on obtient bien le résultat voulu.

Remarque 2: il nous suffit, pour faire le routage au noeud i vers j, de connaître un voisin de i sur un plus court chemin de i à j.

Autre idée: **Floyd-Warshall**.
Pour chaque couple de noeuds (u, v), à chaque étape k,

$$d^{(k)}(u,v) = \min\left(d^{(k-1)}(u,k) + d^{(k-1)}(k,v), d^{(k-1)}(u,v)\right)$$

Programmation dynamique, convergence en $N-2$ itérations maximum. On dit ici qu'un plus court chemin de u à v qui n'utilise que des sommets intermédiaires $\leq k$ est

- soit un chemin qui n'utilise que des sommets intermédiaires $< k$ déjà calculés
- soit un chemin de u à k qui n'utilise que des sommets $< k$ et un chemin de k à v qui n'utilise que des sommets $< k$.

Le coût est ici aussi en $O(n^3)$, et il n'est pas très difficile d'adapter l'algo pour qu'il calcule le successeur pour aller en v depuis u.

1.2.4 Routage dynamique fondé sur le calcul de la distance minimum

Routage à **vecteur de distance**: chaque routeur dispose d'une table de routage précisant pour chaque destination la meilleure distance connue et par quelle ligne l'atteindre (c'est le vecteur de distance). Tables de routage mises à jour par concertation mutuelle entre routeurs voisins qui s'échangent régulièrement des listes de vecteurs.

Algo de **Bellman-Ford**, apprentissage avec des routes en utilisant de l'information agrégée et des échanges locaux, et absence d'état global. La métrique de distance correspond au nombre de routeurs intermédiaires ou autre (cf discussion précédente).

On suppose dans tous les cas qu'un routeur connaît la distance qui le sépare de tous ses voisins. Par exemple pour une métrique de temps d'acheminement, il envoie un paquet test pour mesurer ce temps.

Lemme des plus courts chemins qui dit que si $u \rightsquigarrow w \rightsquigarrow v$ est un plus court chemin, alors $w \rightsquigarrow v$ est aussi un plus court chemin.

Utilisation ensuite de Bellman-Ford: $d(u,v) = \min_w \left(c(u,w) + d(w,v)\right)$ où les w sont les successeurs de u. Programmation dynamique, convergence en $N-1$ itérations maximum.

La version distribuée est la suivante: le noeud w envoie sa valeur de $d(w,v)$ à ses voisins, et le noeud u calcule la meilleure estimation de $d(u,v)$.

Exemple: Une fois toutes les T ms, chaque routeur transmet à ses voisins la liste des longueurs estimées vers chaque destination. De même, il reçoit de ses voisins leur propre liste des longueurs estimées ($d(w,v)$). Et il connaît la distance au voisin ($c(u,w)$). Il peut donc, en faisant le min, connaître la longueur minimale estimée pour aller à une destination donnée. Il peut alors utiliser cette estimation pour mettre éventuellement à jour sa table de routage.

Faire tourner un exemple: graphe et valeurs de mise à jour des tables de routage? Un noeud, ses voisins et les tables de routage de ses voisins, on fait le min ...

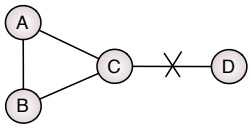
Problèmes de cet algo: propagation des mauvaises nouvelles. La convergence se fait de façon beaucoup trop lente, surtout en cas de mauvaises nouvelles.

Exemple d'un réseau linéaire, distance=1 pour chaque noeud.
A – B – C – D – E
Si A est initialement en panne puis est réparé, en 4 étapes les autres noeuds connaissent le meilleur chemin pour aller à A. B va apprendre dès que A échange ses vecteurs de distance, que A est à sa gauche à une distance 1. La propagation des bonnes nouvelles est rapide.

Par contre, en cas de panne de A, cela va prendre du temps avant que tous réalisent que A est en panne. En effet, B saura qu'il ne peut pas envoyer de paquet directement à A, mais il croira par C que C connaît un chemin pour A (sans réaliser que ce chemin passe par B). On peut faire un tableau récapitulant les distances après chaque échange, la mauvaise nouvelle met du temps à se propager. Ceci est appelé le **problème de la valeur infinie**.

Il existe une solution, dite par **horizon éclaté** ou **coupé**. Dans le vecteur de distance, la distance vers X n'est pas reportée sur la ligne d'arrivée des paquets à faire suivre vers X (valeur infinie). Ainsi, C va *mentir* à B en déclarant que sa distance à A est infinie (car son chemin vers A passer par B). De façon similaire, D dit la vérité à E mais ment à C. Avec cette solution, la mauvaise nouvelle se propage vite.

Cette solution a cependant des failles et ne marche pas pour certaines topologies.



Ici, quand C–D tombe en panne, A et B disent à C qu'ils ne peuvent pas atteindre D. C en conclut que D est inaccessible. Par contre, A apprend que B est à distance 2 de D, il en conclut qu'il est à distance 3 en passant par B, et vice versa. Ainsi, la distance de A et B vers D va aller progressivement vers l'infini.

Il n'y a aucune solution qui fonctionne vraiment bien car lorsque X indique à Y qu'il dispose d'une route, Y n'a aucun moyen de savoir s'il se trouve ou non sur cette route.

1.2.5 Routage par état de liaison (Link state routing)

Cet algo remplace le précédent, de convergence trop lente. Utilisé avec diverses variantes.

Tout routeur doit:

- Découvrir ses voisins: envoie d'un paquet par chaque routeur à ses voisins au démarrage, et les voisins répondent avec leur identifiant/adresse réseau.
- Mesurer le temps d'acheminement (la distance) vers chaque voisin.
- Construire un paquet d'information disant ce qu'il vient d'apprendre (voisins et distance)
- Le paquet d'information est envoyé à tous les autres routeurs
- Chaque routeur calcule le plus court chemin vers tous les autres routeurs, avec l'aide de Dijkstra (marque le noeud racine comme définitif, met à jour les voisins, trouve le noeud non définitif adjacent à un noeud définitif qui a le plus court chemin avec la racine; le marque comme définitif, le met à jour et met à jour ses voisins).

Mesure du temps d'acheminement: faut il prendre en compte la charge de la ligne, ou bien se baser uniquement sur le débit? Problème possible: deux sous-réseaux reliés par deux lignes, et la charge va passer d'une ligne à l'autre (lorsqu'une ligne commence à être chargée, l'algorithme dit de passer par l'autre...).

Construction des paquets: décider quand les élaborer. Construction à intervalle régulier ou bien lors d'une coupure de ligne ou défaillance d'un routeur.

Diffusion de l'info: date et fréquence de diffusion des paquets d'info locale. Diffusion par inondation: utilisation de numéro de séquences pour éviter de diffuser 2 fois le même paquet. Information avec Timeout pour l'éliminer si elle n'est pas remise à jour (en cas de plantage d'un routeur).

Place mémoire nécessaire: proportionnelle à kn, n étant le nombre de routeurs et k le nombre de voisins du routeur. Temps de calcul des routes: n^2.

Algorithme de routage très utilisé dans les réseaux modernes, dans divers protocoles (OSPF - Open Shortest Path First; version modifiée dans IP).

1.2.6 Routage optimal

On sait donc faire un routage avec $s = 1$: routage optimal?

Problèmes:

- Le temps de calcul est un peu coûteux au départ, et le problème est assez centralisé avec certaines techniques (lorsqu'un noeud doit connaître la topologie entière du réseau).
- la taille des informations à stocker est importante

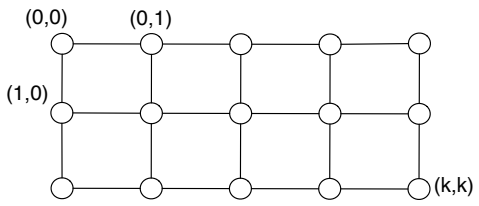
Taille globale? En chaque noeud, pour chaque destination, il faut connaître le noeud successeur. Tableau à n lignes. Coût du stockage du nom: si on a n noeuds, au moins n noms différents, donc un stockage en au moins $\log(n)$ bits par noms, ou même $\log(k)$, k étant le nombre de voisins, comme on l'avait évoqué dans la section précédente. k est le degré du noeud, et on applique une numérotation locale des voisins. La taille totale est donc en $n^2 \log(n)$.

Notation. $\tilde{O}(n^k) = O(n^k \log(n))$: on ne retient que les termes en puissance.

On sait donc faire un stockage global en $\tilde{O}(n^2)$ avec $s = 1$.

1.2.7 Faible coût de stockage

Exemple de routage sur une grille avec numéros: on arrive à un étirement de 1 et un faible coût de stockage.

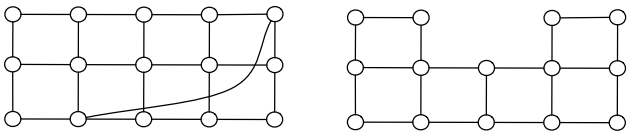


On utilise l'algo suivant (au noeud (i,j), pour router le paquet (x,y)):

$$Route_{(i,j)}(x,y) = \begin{cases} \text{si } x > i, \text{ aller en } (i+1,j) \\ \text{sinon, si } x < i, \text{ aller en } (i-1,j) \\ \text{sinon, si } y > j, \text{ aller en } (i,j+1) \\ \text{sinon, si } y < j, \text{ aller en } (i,j-1) \\ \text{sinon, trouvé} \end{cases}$$

L'information à stocker au sommet (i,j) est uniquement son numéro, en $O(\log(n))$. Le routage se fait en $O(1)$: décision en au plus 4 tests. Etirement sur une grille parfaite: $s = 1$.

Cependant, problème pour grille incomplète ou bien avec sauts:



Algorithmes adaptés à des réseaux *sociaux*, comme on verra plus tard dans le cours, réseaux petit monde, Kleinberg. Grille multidimensionnelle, avec relations logiques (voisin, boulot, activités extra professionnelles...). Routage avec distance= distance dans la grille. Situation comme celle de la figure précédente avec sauts. On transmet au voisin le plus proche de l'objectif dans la grille (on prend les grands sauts pour se rapprocher au plus vite). Routage de longueur polylogarithmique, on gagne beaucoup.

Cas d'une grille avec noms quelconques: définir les coordonnées pour chaque destination possible: $i \rightarrow (x_i, y_i)$.

- x_i codé en $\log X$
- y_i codé en $\log Y$
- Si $X = Y$ (grille carrée), $X = Y = \sqrt{n}$, coût en $\log(n)$.
- $\rightarrow$ en chaque noeud, $n \log(n)$ (coordonnées de chaque destination).
- $\rightarrow$ en tout, coût de stockage en $\tilde{O}(n^2)$, donc on n'a pas gagné grand chose (sauf que c'est la même table à chaque noeud, donc on pourrait utiliser un système de partage...)

1.2.8 Conclusion

Plusieurs facteurs concurrents pour calculer le routage:

- Facteur d'étirement
- Taille des tables locales
- Taille des entêtes des paquets
- Temps de calcul du routage

Si l'on accepte un facteur d'étirement grand et/ou de gros entêtes, on peut avoir des tables locales plus petites (envoyer tout à un noeud donné, le noeud renvoie tout en mettant la route dans l'entête). Si chaque noeud sait router vers

0 et seul 0 sait comment router vers les autres, on a un stockage compact. Mais le facteur d'étirement est arbitrairement mauvais.

Autre solution: stocker le graphe $G = (V, E)$ - coût en $O(|V|+|E|)$, et graphe creux (peu d'arêtes car en général beaucoup de noeuds ont un petit degré). On calcule ensuite le meilleur chemin à chaque fois, et on le traîne dans l'entête.

Il faut donc faire des compromis entre les différentes quantités mises en jeu. En général, les schémas efficaces sont basés sur deux grandes idées:

- On ne veut pas savoir router vers tout le monde (sinon n^2), on choisit donc des noeuds particuliers vers lesquels on sait router.
- A partir de ces noeuds là, on sait router efficacement.

Bornes connues

(non démontré ici)

Si $s < 3$ alors il faut stocker au moins $\tilde{O}(n)$ bits par sommet (au moins $\tilde{O}(n^2)$ en tout). ($\tilde{O}(n^k) = O(n^k \log(n))$).

Inversement, si stocke moins que $\tilde{O}(\sqrt{n})$ par sommet ($\tilde{O}(n\sqrt{n})$ en tout), alors on ne pourra pas faire mieux que $s \geq 5$.

En général on considère que $s \geq 5$ correspond à un algorithme pas assez efficace, et $\tilde{O}(n^2)$ correspond à de trop gros besoins de stockage.

On cherche typiquement des algos avec $s = 3$ et un stockage en $\tilde{O}(\sqrt{n})$ par sommet (ce qui est mieux que de dire $\tilde{O}(n\sqrt{n})$ en tout pour éviter que certains noeuds aient des tables grandes).

Ceci est un compromis optimal vu les contraintes. Il est admissible dans la pratique, et on peut effectivement l'atteindre.

Thorup et Zwick

L'algorithme de Thorup et Zwick marche pour tout graphe, et c'est un algorithme optimal dans le sens des bornes précédentes:

- Information en $\tilde{O}(\sqrt{n})$ bits par sommets, même si le degré du sommet est plus grand que $\sqrt{n}$;
- Etirement $s \leq 3$;
- Entêtes et adresses en $O(\log n)$ bits.

Avant d'étudier cet algorithme (et pour pouvoir l'étudier), on s'intéresse au routage dans les arbres.

Algorithme dans le papier de Thorup et Zwick, lien sur ma page web?

Un point sur ce qu'on s'autorise

On suppose que si une machine veut communiquer avec une autre, elle possède des informations sur cette machine, et donc elle est capable de construire un entête pour la contacter. Attention, les noeuds avec lesquels une machine communique est différent de l'ensemble des noeuds du réseau, on connaît les infos sur les machines avec lesquelles on communique car on les a récupérés.

On désire minimiser:

- La taille de l'entête pour ne pas générer un coût de transport trop important ($O(\log n)$) (et on a vu qu'il faut au minimum $\log n$).
- Les infos stockées dans les routeurs pour pouvoir transformer le message. En effet, c'est là que les ressources matérielles sont critiques.
- Le temps nécessaire pour traiter un paquet qui arrive.

Attention aussi au fait que lorsqu'on compte la mémoire, on compte les bits utilisés, mais toutes les opérations sur un mot machine se font en temps constant $O(1)$ (addition, lecture, multiplication de mots mémoire).

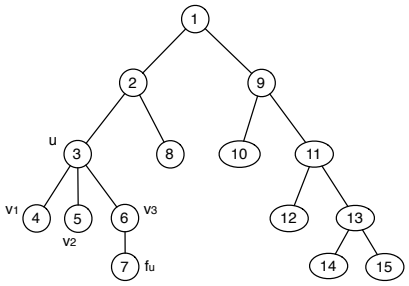
1.3 Routage dans les arbres

On se limite dans un premier temps aux algorithmes de routage dans les arbres, pour introduire l'algo de Thorup et Zwick.

1.3.1 Routage par intervalles

Pas de routage indépendant des noms: on s'autorise à changer les noms des noeuds.

Arbre: on numérote les noeuds selon un parcours en profondeur à partir d'une racine choisie au hasard. Les messages sont destinés à un noeud de numéro entre (1 et n) correspondant au noeud dans l'ordre de profondeur du graphe (c'est là qu'on perd l'indépendance des noms).



Problèmes:

- Comment router dans cet arbre?
- Quelles propriétés utiliser?
- Quelles informations stocker au noeud u?

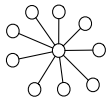
On stocke au noeud u les numéros de ses fils $(v_1, ..., v_k)$ et le numéro du plus grand de ses descendants (f_u). On a $u = v_1 - 1$.

Ainsi,
- les noeuds de $[v_1, v_2 - 1]$ sont joignables par v_1
- les noeuds de $[v_2, v_3 - 1]$ sont joignables par v_2
- ...
- les noeuds de $[v_k, f_u]$ sont joignables par v_k
- les noeuds de $[1, u - 1] \cup [f_u + 1, n]$ sont joignables par le père

La table de routage contient une adresse pour chaque fils et pour le père, donc au noeud u on a une table de taille $(deg(u) + 1) \times \log(n)$.

Algo de routage en u: on doit aller en v. On recherche l'intervalle dans lequel se trouve v, en effectuant une dichotomie sur le tableau suivant:
$[1, u - 1, v_1, v_2, ..., v_k, f_u + 1]$.
Le coût du routage est en $O(\log(deg(u)))$ (coût non constant).
L'étirement de l'algo est 1.

Coût de stockage global en $(\sum deg(u)) \log n = n \log n$: coût acceptable, sauf que certains noeuds peuvent être très chargés si leur degré est important, comme dans le cas d'un serveur centralisé en étoile, donc ce n'est pas acceptable.



1.3.2 Un schéma plus compact

Technique pour limiter les problèmes liés à un nombre de fils important: classification des noeuds en fonction de leur poids et d'un paramètre fixé b ($b = 2$ dans la suite mais on peut généraliser).

En un noeud v, s_v est le nombre de descendants de v dans l'arbre (v inclus).

Soit v' un fils de v. v' est un noeud **lourd** si $s_{v'} \geq \frac{s_v}{b}$, et **léger** sinon.

Lemme: un noeud ne peut pas avoir plus de $b-1$ fils lourds.

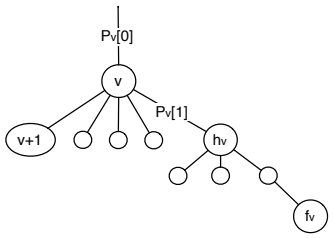
En effet, supposons que v ait b fils lourds. Alors $\sum_{v' \in \text{FilsLourds}} s_{v'} \geq s_v$, or $\sum_{v' \in \text{Fils}} s_{v'} = s_v - 1$, d'où la contradiction qui prouve le lemme.

Racine de l'arbre = noeud lourd (par convention). On suppose que dans le parcours en profondeur on numérote toujours les noeuds légers avant les noeuds lourds (on réordonne l'arbre au besoin).

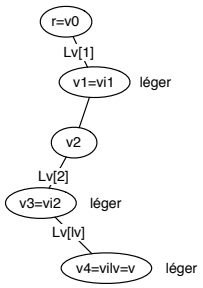
En chaque noeud v on stocke l'information (v, f_v, h_v, P_v) où

- v est le numéro du noeud;
- f_v est le numéro du plus grand descendant (comme dans le routage par intervalle);
- h_v est le numéro du (premier) fils lourd éventuel, sinon $h_v = f_v + 1$;
- P_v contient le numéro de port correspondant à l'arête allant de v à son père ($P_v[0]$), ainsi que le numéro de port pour aller de v à l'éventuel fils lourd (s'il en a un): $P_v[1]$.

Information stockée sur le routeur en $O(\log n)$ (nombre constant de mots).



On regarde le chemin de la racine vers un noeud, et les fils légers rencontrés en route. Combien peut on avoir de noeuds/fils légers sur ce chemin?



Au noeud léger, $s_{v_1} < \frac{s_r}{b}$, puis $s_{v_3} < \frac{s_{v_1}}{b} < \frac{s_r}{b^2}$, et $s_r = n$. En partant de la racine jusqu'à une feuille, après lv noeuds légers on a $s_v = 1 < \frac{n}{b^{lv}}$ d'où, pour $b = 2$, $lv < \log_2(n)$.

Dans l'entête d'un paquet destiné au noeud v, on stocke l'information suivante: (v, L_v) et L_v indique tous les chemins que l'on ne peut pas calculer facilement, à savoir tous les numéros de port pour arriver sur un noeud léger sur la route de r à v. Il y a au plus $\log n$ noeuds légers sur cette route et l'information est de taille $\log n$ (numéro de port, et au plus n voisins), donc entête de taille $\log^2(n)$.
l_v : nombre de noeuds légers au dessus de v (incluant v si v est léger), et
$L_v = (port(v_{i1-1}, v_{i1}), ..., port(v_{il_v-1}, v_{il_v}))$
où les v_{ik} représentent, pour $k = 1..l_v$, les sommets légers sur la route $\{r = v_0, v_1, ..., v_k = v\}$ (cf figure précédente).

Algorithme: Routage du paquet (v, L_v) au noeud u.

1. Si $u = v$ on a trouvé
2. Si $v \notin [u, f_u]$, le paquet est transmis au père en utilisant le port $P_v[0]$
3. Si $v \in [h_u, f_u]$, le paquet doit être transmis au fils lourd (si u n'a pas de fils lourd cet intervalle est vide), sur le port $P_v[1]$.
4. Sinon, v est le descendant d'un des fils légers, et on dispose de l'information de routage dans la table L_v (entête du message): $L_v[l_u]$. l_u nous indique le nombre de noeuds légers au dessus de u, et donc permet de retrouver dans le tableau L_v le port pour aller au fils léger juste en dessous de u.

Ce choix se fait donc en temps constant.

On peut coder efficacement en C pour $b = 2$: le numéro de port s'exprime par la formule:

```
((v>=u && v<h) ? L[l] : P[v>=h && v<=f])
```

(Routage en u d'un paquet v, et on supprime les indices pour la lisibilité). Ici, `L[lv]` représente par convention le numéro de port du lien connectant le routeur v à son hôte. Je rappelle ici qu'une machine est supposée connaître les noeuds avec lesquels elle veut communiquer.

En résumé:

1. Routage en temps constant
2. Taille stockée en chaque noeud: $O(\log n)$

3. Taille de l'entête: $O(\log^2(n))$, et cette taille peut être réduite à $O(\log n)$ avec quelques optimisations.

Détail et optimisation dans l'article de Thorup et Zwick: *"compact routing schemes"* (cf page web du cours).

On s'intéresse maintenant à étendre ce principe pour un algorithme de routage dans des graphes quelconques, en relâchant un peu l'étirement (toujours de 1 pour l'instant).

1.4 Routage dans des graphes quelconques

Principe de Thorup et Zwick (cf article).

1.4.1 Principe

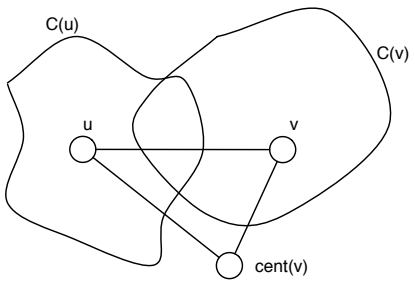
Pour faire du routage avec des plus court chemins, il faudrait avoir pour chaque noeud l'arbre des plus courts chemins, donc n arbres. (Les plus courts chemins sont différents si la source du message change). Or, avec l'algo de Thorup et Zwick, on a vu qu'il fallait stocker $O(\log n)$ informations en chaque noeud, d'où $O(n \log n)$ au total pour les n arbres.

On cherche un stockage en $\tilde{O}(\sqrt{n})$ en chaque noeud, donc on ne peut pas stocker tous les arbres de plus court chemin. On accepte un facteur d'élongation de 3, ce qui permet d'obtenir le résultat voulu.

- On choisit un sous-ensemble des sommets $A \subseteq V$, appelés centres (*centers* pour Thorup, *landmarks* pour Cowen).
- $\delta(u, v)$: distance (pondérée) entre u et v dans le graphe.
- $\delta(A, v) = min\{\delta(u, v) | u \in A\}$: distance entre v et l'ensemble de sommets A.
- Pour chaque $w \in V$, on note $C_A(w) = \{v \in V | \delta(w, v) < \delta(A, v)\}$: ensemble des sommets plus proches de w que de n'importe quel sommet de A. Cet ensemble est appelé le *cluster* de w pour l'ensemble A.
  Si $w \in A$ alors $C_A(w) = \emptyset$, et $C_A(w) \cap A = \emptyset$.
- $cent_A(v)$ représente un sommet de A le plus proche de v. On a donc $\delta(cent_A(v), v) = \delta(A, v)$.

Principe du routage: on s'appuie sur les noeuds centres (de A). Routage de u à v:

1. Si $v \in C_A(u)$, v est dans le cluster de u donc on utilise un plus court chemin de u à v (routage par arbre). L'élongation est de 1.
2. Sinon on passe par $cent_A(v)$, on calcule un plus court chemin de u vers $cent_A(v)$ puis un plus court chemin de $cent_A(v)$ vers v.
   On a alors $\delta(cent_A(v), v) = \delta(A, v) \leq \delta(u, v)$.
   Par inégalité triangulaire et symétrie,
   $\delta(u, cent_A(v)) \leq \delta(u, v) + \delta(cent_A(v), v)$, d'où finalement
   $\delta(u, cent_A(v)) + \delta(cent_A(v), v) \leq \delta(u, v) + 2\delta(cent_A(v), v) \leq 3\delta(u, v)$.



En choisissant A, on dispose alors d'un algorithme d'étirement 3 utilisant cette technique. Cependant,

- Si A est très grand, il faut stocker en chaque noeud l'arbre des plus courts chemins pour tout $a \in A$, ce qui est trop cher (en gros, $O(|A| \log n)$ par noeud).
- Si A est trop petit, alors pour certains noeuds u, $C_A(u)$ risque d'être très grand et on doit stocker en u tous les plus courts chemins vers tous les $v \in C_A(u)$, ce qui est en gros en $O(|C_A(u)| \log n)$.
- La difficulté consiste donc à construire A de la bonne taille, pour avoir $|A|$ en $\tilde{O}(\sqrt{n})$ et $C_A(u)$ en $\tilde{O}(\sqrt{n})$ pour chaque sommet.

1.4.2 Construction de A

On utilise l'algorithme de Thorup et Zwick suivant.

Algorithme **center**(G, s)
$A \leftarrow \emptyset$; $W \leftarrow V$;
tant que $W \neq \emptyset$ **faire**
  $A \leftarrow A \cup$ **sample**(W, s);
  $\forall w \in V,\ C(w) \leftarrow \{v \in V \mid \delta(w, v) < \delta(A, v)\}$;
  $W \leftarrow \{w \in V \mid |C(w)| > 4n/s\}$;
fin
retourner A

Cet algorithme prend en paramètre un graphe pondéré non orienté $G = (V, E)$ et un paramètre $s \in [1, n]$. Le résultat est un ensemble $A \subseteq V$ de taille (en moyenne) $O(s \log n)$, tel que tous les clusters $C_A(w)$, pour tout $w \in V$, sont de taille au plus $4n/s$.

Cet algorithme utilise la sous-procédure **sample**(W, s), qui renvoie un sous-ensemble de W obtenu en sélectionnant chaque élément de façon indépendante, avec la probabilité $s/|W|$ d'être retenu. Ainsi, si $|W| \leq s$, tous les éléments sont retenus et la procédure renvoie W. En moyenne on renvoie s éléments aléatoires de W (si $|W| \geq s$).

L'algorithme rajoute dans l'ensemble A des sommets choisis de façon aléatoire dans l'ensemble W, calcule ensuite les $C_A(w)$, puis recommence en prenant comme ensemble des sommets W les sommets avec un trop grand cluster (de taille $> 4n/s$).

A la fin de l'algorithme **center**, $W = \emptyset$ donc pour tout sommet $w \in V$ on a $|C_A(w)| \leq 4n/s$, ($C_A(w)$ représente les noeuds pour lesquels on doit connaître un plus court chemin depuis w).

A prouver: Terminaison de l'algo?

Nombre d'étapes de l'algorithme? A chaque étape, on ajoute environ s noeuds à A.

Notons par W_i l'ensemble W au début de l'étape i.

$A' =$**sample**(W_i, s) est l'ensemble des éléments rajoutés à A durant cette étape.

La démonstration n'est pas faite formellement ici mais elle est disponible dans le papier de Thorup et Zwick (lien sur la page web du cours).

On montre finalement qu'avec une probabilité 1/2 on a à la fin de l'étape i $|W_{i+1}| \leq |W_i|/2$, et on peut en déduire que l'on termine en moyenne en moins de $2\log n$ étapes.

Vu que l'on rajoute s éléments dans A par étape, on obtient A de cardinal $2s\log n$.

En prenant $s = \sqrt{n/\log n}$, on obtient:

$$\forall w \in V \; C_A(w) < \frac{4n}{\sqrt{n/\log n}} < 4\sqrt{n\log n}$$

$$|A| = O(2\log n \; \sqrt{n/\log n}) = O(\sqrt{n\log n})$$

et même $A \leq 2\sqrt{n\log n}$

1.4.3 Routage

Taille en chaque routeur: au point u, on connaît

- $\forall v \in C_A(u)$, le voisin de u dans un plus court chemin vers v. On stocke donc en chaque noeud u un arbre en $O(\log n)$ pour chaque v, soit un coût de stockage en $O(\log n) \times |C_A(u)|$, ce qui revient à $O(\log n\sqrt{n\log n})$.
- $\forall a \in A$, on a comme information: mon numéro dans l'arbre de a T_a, mon plus grand fils dans T_a, l'indice de mon fils lourd dans T_a, et le nombre de noeuds légers entre a et moi. C'est donc en $O(\log n)$, et ce pour chaque élément de A, d'où une taille en $O(\log n) \times |A| = O(\log n\sqrt{n\log n})$
- D'où un total en $O(\sqrt{n}(\log n)^{\frac{3}{2}}) = \tilde{O}(\sqrt{n})$.

Taille des entêtes des messages: message de u à v

- Si $v \in C_A(u)$, entête $[0, v]$
- Sinon, $[1, cent_A(v), v_{cent_A(v)}, L_v^{cent_A(v)}]$ où $cent_A(v)$ est l'indice du centre de v, ce qui indique l'arbre à utiliser, $v_{cent_A(v)}$ est le numéro de v dans cet arbre, et $L_v^{cent_A(v)}$ est la suite des ports à utiliser pour passer d'un noeud lourd à un noeud léger dans cet arbre.
- Globalement une taille de l'entête en $\log n + \log n + \log^2 n$, mais on a vu qu'en optimisant le $\log^2 n$ peut être ramené à $\log n$. Donc une taille des entêtes en $O(\log n)$.

Algorithme de routage:

- Si $v \in C_A(u)$, sur tout le chemin de u à v, on se rapproche de v, et donc pour tout w sur ce plus court chemin, $v \in C_A(w)$ et donc w sait router au plus court.
- Sinon, on peut router vers v dans l'arbre des plus courts chemins de $cent_A(v)$.
- Le routage se fait en temps constant, avec une élongation 3.

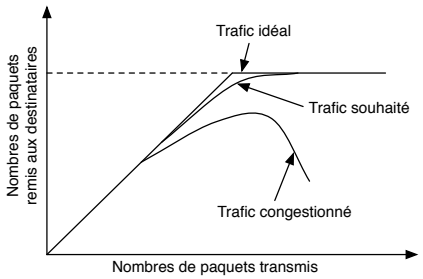
Tous les critères d'optimalité sont donc réunis.

1.5 Congestion dans les réseaux

On aborde maintenant quelques problèmes liés aux routages, liés à la congestion dans les réseaux. On propose aussi des nouveaux algorithmes basés sur ces idées.

1.5.1 Définition de la congestion

Nombreux paquets présents dans le sous-réseau → dégradation des performances: phénomène de **congestion**. Si le trafic augmente trop vite, les routeurs ne peuvent pas faire face, ils perdent des paquets. Lié à la file d'attente des routeurs, et à la technologie de transport utilisée (TCP...). Le problème tend à empirer, et avec un trafic élevé les performances s'écroulent complètement, la quasi-totalité des paquets ne sont plus délivrés à leur destinataire.



Principal facteur de congestion: saturation de la file d'attente sur un routeur. Paquets détruits s'il n'y a pas la place de les stocker. Solution: agrandir la file d'attente. Cependant, file d'attente de longueur infinie empire encore le phénomène de congestion, car le temps mis par un paquet pour atteindre la tête de file devient trop important. Paquets non acquittés retransmis, ce qui augmente la charge sur le réseau et la congestion.

Autres sources de congestion: processeurs internes aux routeurs de faible performance. La file d'attente grandit alors même si le routeur a une capacité de transmission suffisante. Même problème pour les lignes à faible bande passante.

La congestion s'entretient elle même et devient de plus en plus forte. Ainsi, si un routeur ne dispose pas de buffer libre, il ignore les nouveaux paquets qui arrivent, et l'émetteur du paquet dépasse le temps limite d'attente d'un accusé de réception. Il retransmet donc systématiquement le paquet, éventuellement plusieurs fois de suite. Il ne peut donc pas libérer de place de son côté pour recevoir de nouveaux paquets, et la congestion se propage en amont, comme un embouteillage de voitures.

Principes du contrôle de congestion

Résolution du problème lors de la conception du système, pour s'assurer qu'il n'y aura pas congestion. Aucune correction possible lorsque le système est en cours de fonctionnement.

Contrôle basé sur une rétroaction: surveiller le système pour détecter quand et où la congestion apparaît, puis envoyer cette information là où les actions doivent être prises. Le comportement du système est alors ajusté pour corriger le problème. Problème: paquets de contrôle que l'on envoie pour signaler la congestion amplifient le phénomène de congestion en augmentant le trafic juste au moment où l'on cherche à le réduire... Autre problème: éviter l'oscillation du système en trouvant la bonne fréquence de rétroaction (il ne faut pas réagir trop vite, mais pas trop lentement non plus).

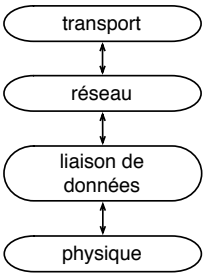
Réaction à une congestion: augmenter les ressources ou diminuer le trafic.

Circuits virtuels: contrôle de congestion dans la couche réseau. Réseaux datagrammes: contrôle à la fois dans la couche réseau et dans la couche transport.

1.5.2 Algorithmes de contrôle de congestion

Prévention de la congestion

Elle peut se faire à différents niveaux. La couche *liaison de données* se situe juste au dessus de la couche *physique*, mais en dessous de la couche *réseau*. la couche *transport* est au dessus de la couche *réseau*.



La **couche liaison de données** fractionne les données d'entrée de l'émetteur en trames de données, transmet les trames en séquence et gère les trames d'acquittement. Elle se sert de la couche physique, et doit offrir à la couche réseau une liaison exempte d'erreurs. Le contrôle de congestion peut se faire à ce niveau, au niveau de la politique de retransmission: vitesse de réaction et comportement de l'émetteur face à des paquets transmis dont les accusés de réception sont hors délais. Différentes politiques au niveau de l'accusé de réception peuvent aussi influer la congestion, si chaque paquet est acquitté immédiatement les paquets d'acquittement créent une surcharge de trafic. La politique de contrôle de flux peut aussi influer.

Au niveau de la **couche réseau**, le choix entre circuit virtuel et datagramme influe sur la congestion, certains algos ne marchent que pour le cas du circuit virtuel. On peut influer ici sur la politique de mise en attente des paquets, la gestion des files d'attente des routeurs, le mode de distribution (priorités, circulaire), la destruction des paquets (file pleine). L'algorithme de routage

joue également un rôle primordial en étalant de façon optimale le trafic. Il faut également bien gérer la durée de vie des paquets (combien de temps on conserve un paquet avant de le détruire).

Enfin, on peut également trouver des voies d'action au niveau de la **couche transport**, qui se charge de découper les données, les passer à la couche réseau, et s'assure que tous les morceaux arrivent correctement de l'autre côté. Protocoles de transport: TCP, UDP... Les voies d'action sont les mêmes que pour la couche liaison, sauf qu'il est plus difficile d'estimer les intervalles de temps hors délais dans la gestion des paquets. En effet, le temps de transit dans un réseau est beaucoup moins prévisible que celui sur un support de transmission entre deux routeurs. Trop court: paquets inopportuns retransmis. Trop long: congestion réduite mais temps de réponse plus long à chaque paquet perdu.

Canalisation du trafic

Problème de congestion lié au fait que le trafic est aléatoire et souvent par rafales ou saccades incontrôlées. Une méthode pour prévenir la congestion consiste à pousser les ordinateurs source à fournir leurs données à un rythme uniforme.

Algorithme du seau percé: seau d'eau dont le fond est percé d'un trou, l'eau s'écoule goutte à goutte. Peu importe le débit d'arrivée de l'eau dans le seau, l'écoulement se fait à vitesse constante. Plus d'eau dans le seau: l'écoulement s'arrête. En revanche, si le seau déborde, l'eau déborde (et se perd) mais l'écoulement par le trou reste contant.

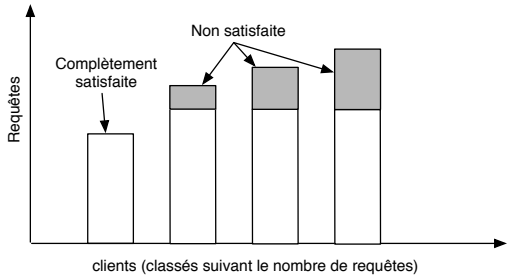
1.5.3 Max-Min Fairness

Partage équitable des demandes des ressources:

- Les petits utilisateurs ont tout ce qu'ils veulent
- Les gros utilisateurs se partagent éventuellement ce qu'il reste

Plus formellement, les ressources sont allouées aux utilisateurs dans l'ordre grandissant des demandes. Personne ne reçoit plus que ce qu'il a demandé, et les utilisateurs avec des demandes non satisfaites se partagent les ressources restantes.

Exemple:



Max-min pondéré, qui rajoute des priorités aux clients.
Regarder éventuellement plus d'infos sur
`http://www.cs.berkeley.edu/~kfall/EE122/lec26/`

1.5.4 Algorithme de flot maximum

C'est traité en cours/TD de graphe en L3, et dans le Cormen: méthode de Ford-Fulkerson. Ce sont des techniques algorithmiques qui servent pour les réseaux.