

Synthèse Introduction aux réseaux

Jean-Philippe Collette

28 janvier 2012

Table des matières

1	Introduction	3
1.1	Internet	3
1.2	Réseau périphérique	3
1.2.1	Réseaux d'accès	3
1.2.2	Supports physiques	6
1.3	Cœur du réseau	7
1.3.1	Circuit switching	7
1.3.2	Packet switching	8
1.3.3	Structure d'Internet	9
1.4	Métriques	9
1.4.1	Délai et pertes	9
1.4.2	Perte (loss)	10
1.4.3	Débit	10
1.5	Modèles et protocole	11
1.6	Histoire	11
2	La couche applicative	12
2.1	Principes des applications réseau	12
2.1.1	Création d'une application réseau	12
2.2	Web et HTTP	13
2.2.1	Type de HTTP	13
2.2.2	Message HTTP	14
2.2.3	Les cookies : état client-serveur	15
2.2.4	Cache web (proxy)	15
2.3	DNS - Domain Name System	16
2.3.1	LocalName Server	17
2.3.2	Enregistrements DNS	18
2.3.3	Messages DNS	18
2.3.4	Ajout d'entrées DNS	18
2.4	Programmation de socket	18
2.4.1	Avec UDP	19
2.4.2	Avec TCP	19
3	La couche de transport	20
3.1	Services de la couche de transport	20
3.2	Multiplexage et démultiplexage	20
3.2.1	Démultiplexage	20
3.3	UDP	21
3.4	Principes de fiabilité lors d'un transfert de données	21
3.4.1	1.0 - Canal fiable	22
3.4.2	2.0 - Canal avec erreurs binaires	22
3.4.3	2.1 - Erreur d'acquis	23
3.4.4	2.2 - Protocole sans NAK	23
3.4.5	3.0 - Canal avec pertes et erreurs	24
3.4.6	Protocole avec pipeline	26
3.5	TCP	28
3.5.1	Structure de segment	28
3.5.2	Timer	28
3.5.3	Fiabilité	29
3.5.4	Contrôle de flux	30
3.5.5	Gestion de la connexion	31

3.6	Principes de gestion de congestion	33
3.6.1	Scénario 1	33
3.6.2	Scénario 2	33
3.6.3	Scénario 3	34
3.6.4	Solutions de contrôle de congestion	35
3.7	Le contrôle de congestion de TCP	35
3.7.1	Efficacité de TCP	36
3.7.2	Équité de TCP	37
4	La couche réseau	38
4.1	Introduction	38
4.2	Circuits virtuels et réseaux de datagrammes	38
4.2.1	Circuits virtuels	38
4.2.2	Réseau de datagrammes	39
4.2.3	Comparaison des deux modèles	40
4.3	Routeurs	40
4.3.1	Ports d'entrée	40
4.3.2	Switching fabrics	41
4.3.3	Ports de sortie	41
4.3.4	Queuing à l'entrée et à la sortie	41
4.4	Protocole IP	42
4.4.1	Format d'un datagramme IP	42
4.4.2	Adressage IPv4	43
4.4.3	ICMP	44
4.4.4	IPv6	44
4.5	Algorithmes de routage	45
4.5.1	Métriques pour le coût	45
4.5.2	Principe d'optimalité	46
4.5.3	Classification des algorithmes de routage	46
4.5.4	Etat de lien - link state	47
4.5.5	Vecteurs de distances - distance vector	48
4.5.6	Comparaison des algorithmes LS et DV	49
4.5.7	Routage hiérarchisé	50
4.6	Routage dans l'Internet	50
4.6.1	RIP	50
4.6.2	OSPF	51
4.6.3	BGP	52
5	La couche lien	54
5.1	Détection d'erreurs	54
5.1.1	Le checksum d'Internet	55
5.1.2	CRC - Cyclic Redundancy Check	55
5.2	Protocole à accès aléatoire	56
5.2.1	Protocole à accès multiple idéal	56
5.2.2	Slotted ALOHA	56
5.2.3	ALOHA pur	57
5.2.4	CSMA	58
5.2.5	CSMA/CD	59
5.2.6	Protocoles Taking turns	59
5.3	Adressage	60
5.4	Ethernet	61
5.4.1	Encodage de Manchester	62
5.5	Switchs et hubs	62
5.6	PPP	64
5.6.1	Trame PPP	65

Chapitre 1

Introduction

1.1 Internet

On a comme composants :

- les hôtes, à la périphérie du réseau
- des liens de communication
- des routeurs, qui permettent la direction de paquets
- des protocoles d'envoi et de réceptions de message. La documentation et la standardisation permettent l'interopérabilité.

Un réseau est cohérent et autonome s'il est administré et indépendant, tout en respectant les standards. Dans son ensemble, Internet n'est pas géré, ses sous-parties le sont.

On peut voir Internet comme un ensemble de services (mails, messagerie instantanée, streaming, VoIP). Ils sont distribués, car échangent des informations entre eux, et peuvent être classés en 2 types :

- les services fiables, orientés connexion, l'envoi du message de la source vers la destination est garanti (TCP) ;
- les services non fiables, où on fait au mieux (UDP).

Un protocole définit le format et l'ordre des échanges entre deux ou plus entités communiquant, c'est le minimum pour l'interopérabilité et l'orchestration des messages (ordre des messages envoyés ou reçus, que faire lors de la réception d'un certain message, que répondre, etc).

1.2 Réseau périphérique

Les systèmes d'extrémité sont les hôtes d'applications qui communiquent entre elles.

Il y a 2 grands modèles de communication :

- modèle client/serveur : un programme client est un programme tournant sur un système d'extrémité et qui envoie des requêtes et reçoit des réponses d'un programme serveur tournant sur un autre système d'extrémité ;
- modèle peer-to-peer : un hôte est client et serveur à la fois. Le problème est que tout peut être mobile et/ou pas toujours en ligne. Un serveur est souvent nécessaire pour un usage minimal (pour savoir qui est en ligne, etc).

1.2.1 Réseaux d'accès

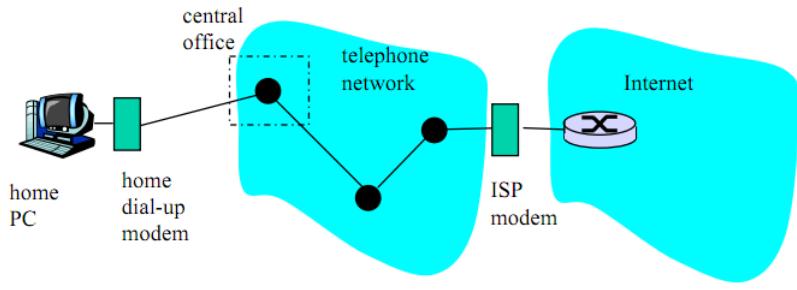
Ce sont les liens physiques qui connectent un système d'extrémité au premier routeur (dit routeur d'extrémité). La plupart des technologies d'accès reprennent des portions du réseau téléphonique traditionnel.

Dans les technologies qui permettent de se connecter, on utilise deux grandes métriques :

- la bande-passante
- le caractère dédié ou partagé

Dial-up modem Accès réseau dédié : le signal est modulé par un modem (home dial-up modem) qui va être démodulé par un autre modem (ISP modem) qui va permettre la transmission du signal sur Internet : le codage du signal est adapté au réseau téléphonique (numérique → analogique → numérique). C'est un modèle orienté connexion car elle est explicite et physique.

La connexion est dite "dial-up" car le programme utilisateur compose le numéro (dial) de l'ISP.

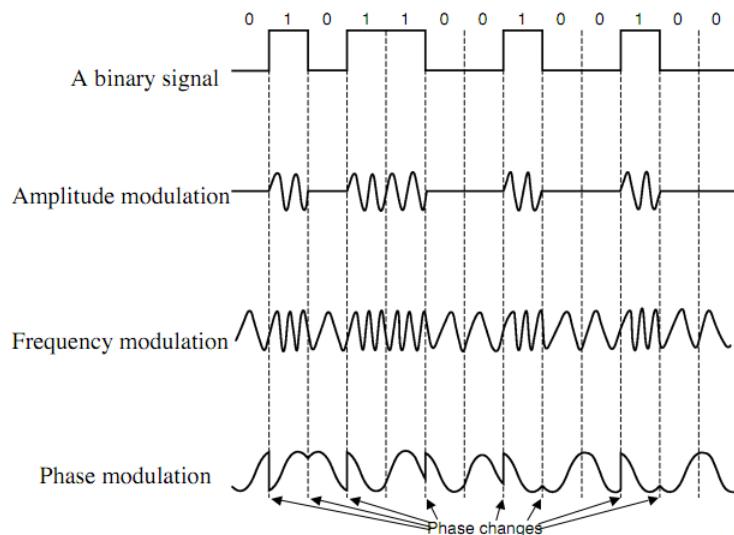


Désavantages :

- Les réseaux téléphoniques n'ont pas beaucoup de bande passante, seulement un spectre de fréquences audibles, pour les voix, entre 0 et 4 kHz. Si la cadence du signal binaire est trop grande, le système ne suit pas.
- Le téléphone est inutilisable.

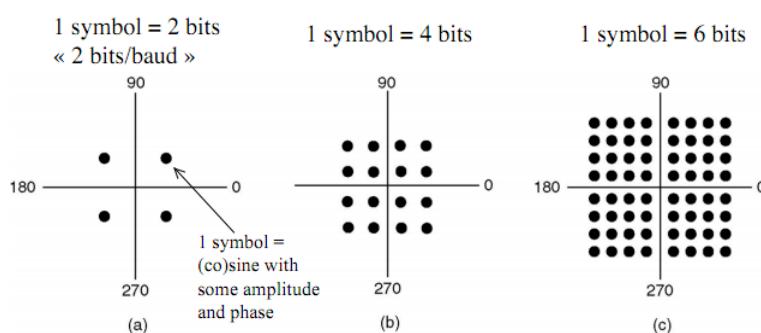
Le signal binaire est transmis grâce à la modulation des composantes des signaux analogiques. Il y a trois types de modulation possible :

- modulation d'amplitude, avec des valeurs hautes et basses
- modulation de fréquence, avec des fréquences tantôt faibles tantôt élevées
- modulation de phase, avec des décalages



Un symbole est une modification du signal. Le baud est l'unité de mesure du nombre de symboles transmissibles par seconde. A ne pas confondre avec le nombre de bits par seconde, qui est un débit d'information.

On peut combiner les modulations de phase et d'amplitude : un symbole (ou baud) peut valoir 2 bits (car il y a 4 possibilités : 00, 01, 10 et 11). Idem pour un symbole de 4 bits et de 6 bits, en jouant sur l'amplitude et la fréquence.



$$\text{Débit de données} = \text{débit de baud} \times \text{nombre de bits par baud}$$

La sinusoïde doit être au plus à 4 kHz, donc période max de 4 millièmes de secondes. Si on réduit trop l'intervalle de transmission (le débit de baud), on ne saura plus retrouver le symbole, la phase et l'amplitude étant incalculables. La limite du nombre de symboles à envoyer est au maximum le double de la période (Niquist) : si H est la bande de fréquence du canal physique.

$$\text{Débit de Baud} \leq 2 \times H$$

On peut toujours augmenter le débit sur le nombre de bits par baud, qui n'est théoriquement pas limité. Cependant, le débit de donnée est limitée par le rapport signal/bruit :

$$\text{Loi de Shannon : débit binaire} \leq H \times \log_2\left(1 + \frac{S}{N}\right)$$

Avec S la puissance du signal et N celle du bruit. Il y a toujours du bruit (thermique).

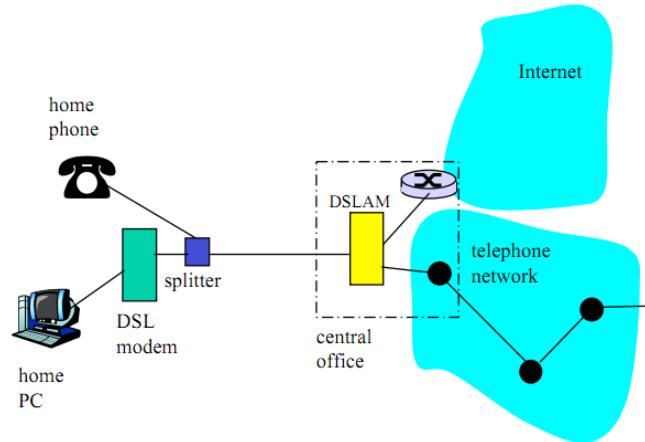
On ne peut augmenter arbitrairement le nombre de bits par baud, car les symboles seront perturbés ; les points risquent de se déplacer sur la grille. Un décodeur effectue un maximum de vraisemblance pour déterminer la position du point, mais s'il n'y a pas assez d'espace il y a un risque d'erreur. De plus, on est conditionné dans un cercle (l'amplitude dépend du nbr de watt à produire, qui n'est pas infini).

Cependant, $\frac{S}{N}$ n'est jamais nul, il y a toujours moyen de transmettre de l'information.

Le débit est aussi limité par la distance à laquelle on se trouve de la centrale téléphonique (atténuation du signal).

DSL (Digital Subscriber Line) Utilisation d'autres bandes de fréquences des lignes téléphoniques, fréquences qui dépendent des distances. Les canaux sont bi-directionnels (full-duplex) : on peut émettre, recevoir et téléphoner en même temps. On peut généralement monter à 1MHz. Généralement,

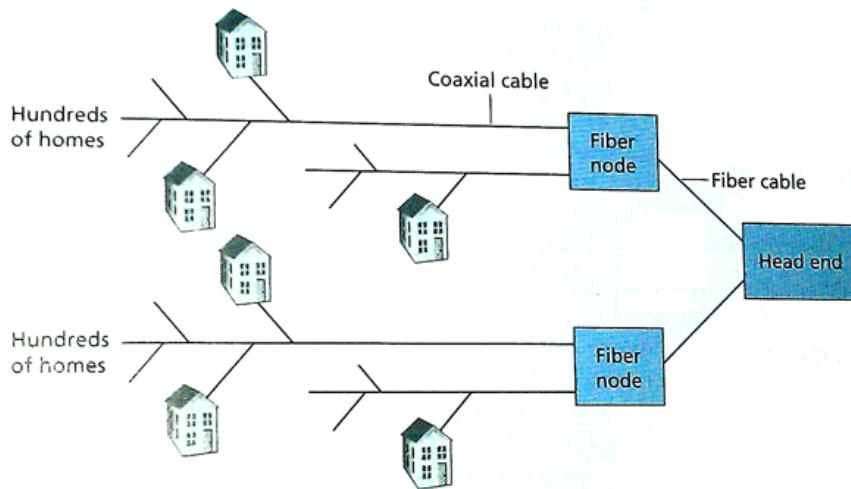
- le flux descendant (download) dans la bande de 50kHz à 1 MHz
- le flux ascendant (upload) dans la bande de 4kHz à 50 kHz
- le flux téléphonique classique, de 0 à 4 kHz



Un DSLAM est un démultiplexeur qui sépare les signaux pour Internet et téléphonique.

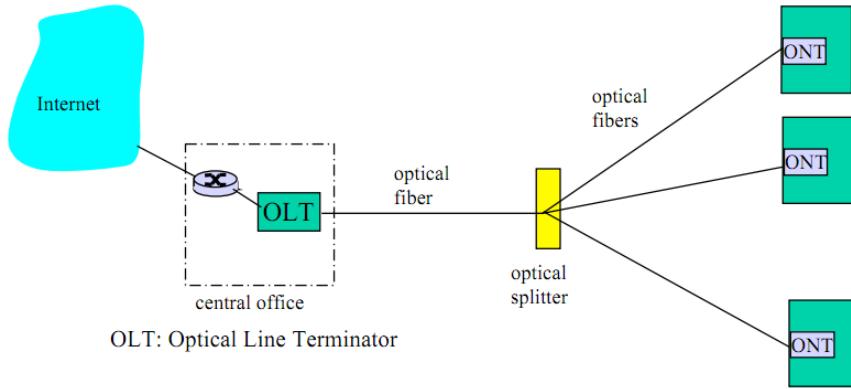
Les avantages par rapport à une connexion dial-up sont que les débits (également asymétriques) sont beaucoup plus élevés et que les utilisateurs peuvent téléphoner et utiliser le réseau en même temps. Il y a constamment une connexion au DSLAM de l'ISP.

Modem câble, réseaux HFC Utilisation du réseau de télédistribution, à base de câbles coaxiaux. Le système est HFC (hybrid fiber coax), c'est-à-dire qu'on utilise en plus de la fibre optique.



L'accès n'est pas point à point mais est partagé entre plusieurs utilisateurs, contrairement aux autres moyens de connexion.

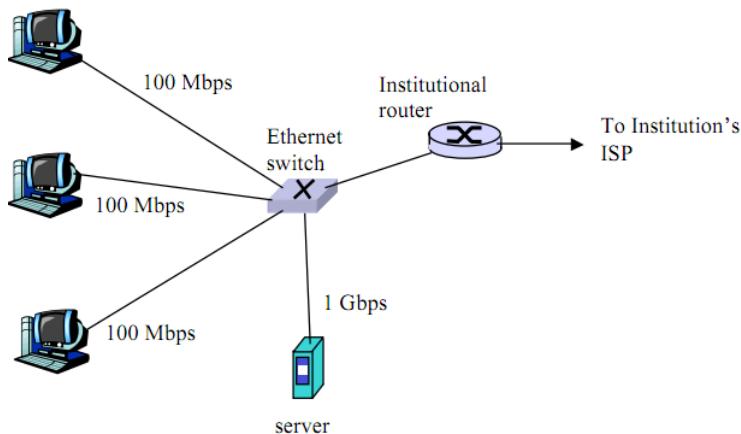
Fiber to the home Idem que modem câble, mais l'arrivée du signal à la maison se fait par fibre optique (au lieu de coax).



Généralement, les fibres sont partagées entre plusieurs résidences, la connexion est partagée. Il y a deux sortes de technologies qui permettent le split entre les clients :

- AON : active optical networks (switched ethernet)
- PON : passive optical networks : chaque maison possède un terminateur de réseau optique (ONT), qui est connecté à un splitter dédié (dans le voisinage). Le splitter combine les signaux en une seule fibre et l'envoie à l'OLT (optical line terminator), qui convertit les signaux optiques en signaux électriques. L'OLT est connecté à Internet via un routeur ; les utilisateurs ont accès avec un routeur connecté à l'ONT.

Ethernet Utilisation de paires torsadées, connectant les utilisateurs à un switch ethernet.



Réseaux à accès sans-fil Réseau partagés, connexion d'un point d'accès à un routeur. Potentiel killer : WiMAX.

1.2.2 Supports physiques

On distingue les supports matériels (guided media), qui propagent les signaux à travers du solide (fibre, coax) et les supports immatériels (unguided media), tels que les ondes radio.

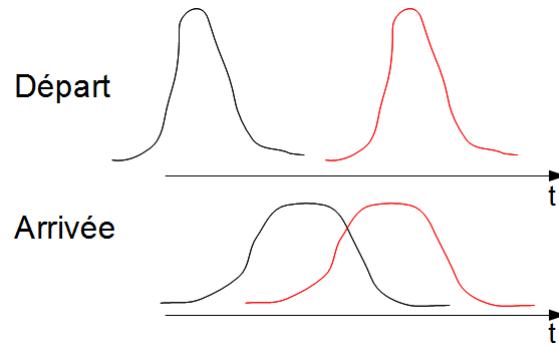
Paire de cuivre torsadée Deux fils de cuivre isolés. Ils sont torsadés pour qu'une surface annule l'induction magnétique d'une surface voisine. Sinon il faut blinder, mais c'est plus cher.

Coax Deux conducteurs de cuivre concentriques (au lieu d'être parallèles comme le fil de cuivre), bidirectionnel. Chaque signal (TV ou digital) est transmis sur une bande de fréquence particulière.

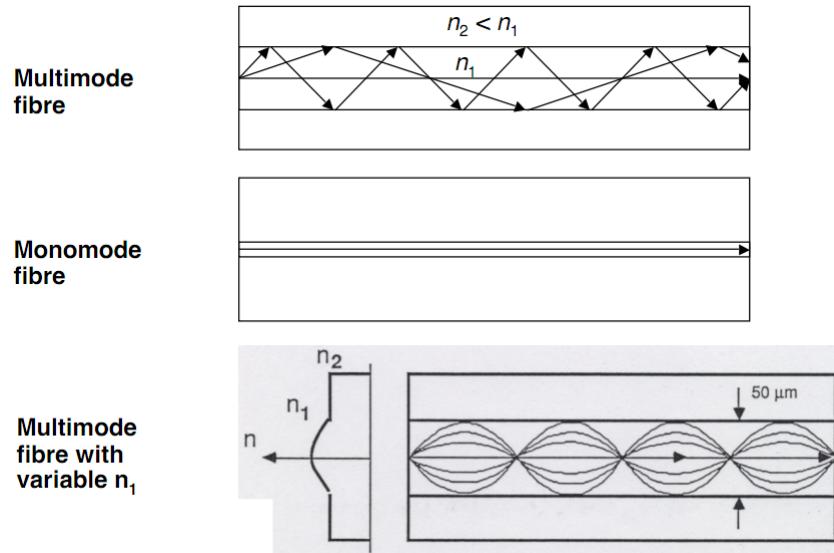
Fibre optique Des pulses de lumières, représentant un bit, sont envoyés à travers une fibre de verre. Support non exposé au bruit électromagnétique.

On utilise deux types de verre pour ne pas perdre des photons, à cause du phénomène de réfraction (qu'il faut éviter), pour obtenir une réflexion totale.

Dans une fibre dite multimode, il y a plusieurs trajectoires de propagation. Il y a dès lors une dispersion de délai : des photons mettent plus de temps à arriver que d'autres. Il y a également un risque de superposition des impulsions. La solution est de décaler les impulsions : plus d'interférences, mais moins de débit.



On peut faire varier n_1 , qui permet de modifier la trajectoire des photons. Les photons qui suivent les courbes sont accélérés (car n_1 plus grand), la courbure est compensée.



Ondes radio Signal porté sur un spectre électromagnétique. Très sensible à l'environnement (réflexion, obstruction, interférence).

Différentes types de lien radio :

- micro-ondes terrestres
- LAN (ex : wifi)
- wide-area (ex : 3G)
- satellite : jusqu'à plusieurs Mbps, mais délai élevé.

1.3 Coeur du réseau

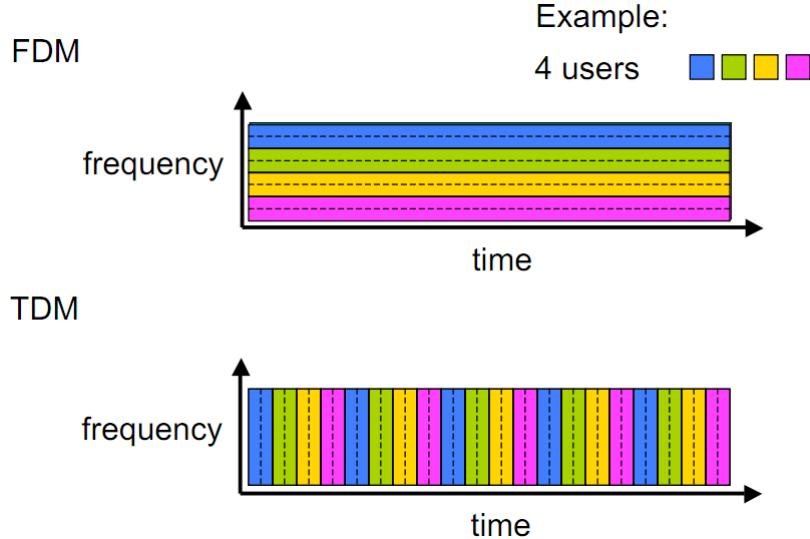
C'est un ensemble de routeurs interconnectés, qui acheminent l'information d'un hôte à l'autre. Deux sortes de transmission :

- switch de circuit : il doit y avoir un circuit de bout en bout du système ; on a un circuit dédié
- switch de paquets : envoi de l'information en morceaux discrets

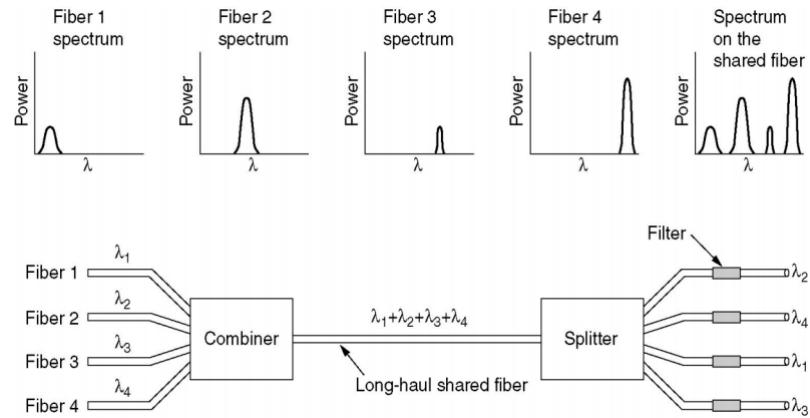
1.3.1 Circuit switching

Il arrive un moment où un lien est partagé entre plusieurs connexions, il y a une allocation statique de ressources. On garantit que la bande-passante et le canal, mais nécessite de pré-processing d'établissement et de relâchement.

On divise le lien physique en morceau en divisant en fréquence (FDM, frequency division multiplexing ; télédistribution) et en temps (TDM, time division multiplexing ; n'est plus utilisé actuellement). Avec la division en fréquence, le hash est restreint mais on l'a tout le temps, tandis qu'avec la division temporelle, le hash est maximal mais le temps est limité.



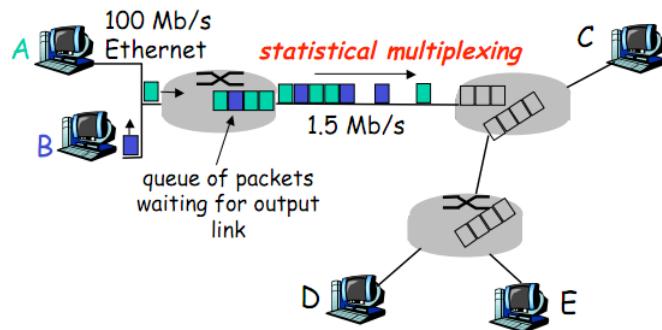
WDM (wavelength division multiplexing) : utilisation des longueurs d'onde pour partager le canal. Tout le circuit est optique (le splitter est un prisme). Les λ_i transitent en parallèle et il n'y a pas d'interférences, car les longueurs d'onde sont différentes. WDM sur une fibre multimode n'est pas avantageux (crée des interférences, pas pratique pour du parallélisme), car on retrouve le problème de dispersion. Plus efficace sur une fibre monomode.



Modèle peu adapté pour des envois des quelques bytes, car on bloque tout un circuit pour peu d'information.

1.3.2 Packet switching

On demande des ressources pour envoyer un paquet de données : les ressources ne sont pas bloquées pendant des longues périodes de temps. Chaque paquet utilise toute la bande passante d'un lien.



Il y a congestion lorsque le réseau surchargé (car trop de paquets), il y mise en attente des paquets à envoyer dans des buffers.

Le multiplexage est statistique : les paquets passent de noeud en noeud ; ils sont analysés, placés en file d'attente, et envoyés. C'est une sorte de TDM, mais il n'y a pas de régularité (alors que le TDM est synchrone).

Diviser en petits paquets permet de diminuer les délais : quand un paquet est envoyé, on peut en préparer un autre ; une série de petits paquets passe plus vite qu'un seul gros.

Le packet switching a supplanté tout car il permet à un plus d'utilisateurs de se connecter au même canal (ex : pour une ligne de 1Mbps, où chaque ligne a un débit de 100kb/s quand elle est active, et ce 10% du temps, 10 utilisateurs

max pour circuit switching. Pour le packet switching, la proba qu'il y ait 35 utilisateurs en même temps est très faible ($(0.1^{35} + 0.1^{34} + \dots + 0.1^{11})$).

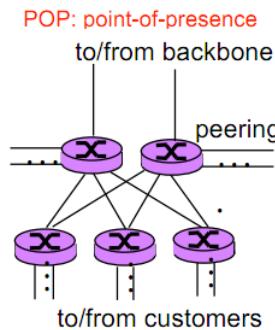
Economiquement intéressant et efficace pour l'envoi de données en rafale, mais nécessite de savoir pour l'utilisateur s'il y a des congestions dans le réseau.

1.3.3 Structure d'Internet

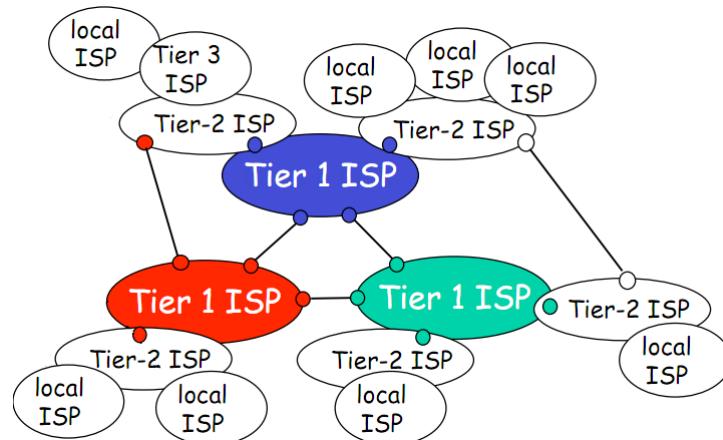
Internet est un réseau de réseaux, qui sont hiérarchisés :

- au coeur, il y a les ISP (Internet Service Provider) (Tiers 1), qui sont interconnectés entre eux (relations de peering). Il y a un genre de troc de trafic ; ils sont tous sur le même pied d'égalité.

Un noeud est un POP (point of presence), un centre avec de multiples routeurs. Un graphe est un backbone. Les POP sont reliés entre eux et aux clients, via un ISP plus régional. Il y a également des liens de peering (vers d'autres tiers 1).



- Il y a des plus petits ISP connectés (Tier-2), à un niveau régional. Là, il y a une relation de client/fournisseur (contrairement aux Tiers-1) : les tiers-2 sont des clients des tiers-1. Il peut y avoir des relations de peering entre eux (généralement, à un point de peering, se rassemblent), ce qui est intéressant pour éviter des frais (troc).
- Le dernier niveau de hiérarchie (tier-3) est constitué des ISP locaux.

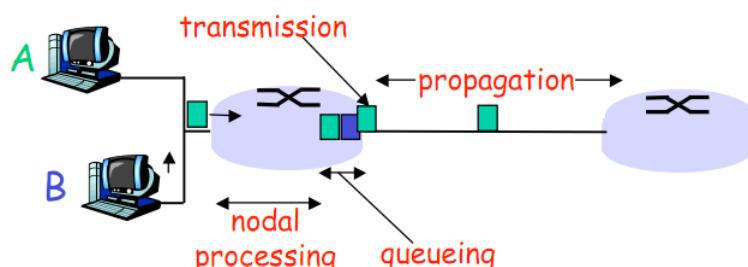


1.4 Métriques

1.4.1 Délai et pertes

Il y a 4 composantes dans le délai :

- délai de processing : vérification du paquet et détermination de la sortie (dépend souvent du CPU).
- délai d'attente : temps d'attente à la sortie d'un lien pour la transmission
- délai de transmission : dépend de la taille du paquet ; ratio entre longueur du paquet (L) et débit binaire (R) : $\frac{L}{R}$
- délai de propagation : dépend de la longueur du câble (d) et de la vitesse de propagation (s) : $\frac{d}{s}$.

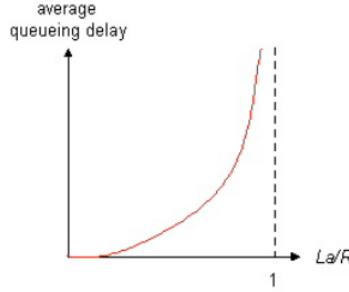


$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

d_{queue} est délai de bufferisation, élément le plus variable.

Délai de bufferisation

Intensité du trafic : $\frac{La}{R}$, avec a le débit d'arrivée dans la file d'attente des paquets en moyenne. C'est un ratio arrivée/sortie, qui croît de manière sur-linéaire.



Délai sur Internet

Le roundtrip time (RTT) est temps aller-retour d'un paquet.

On peut déterminer le chemin que suivent les paquets en envoyant des paquets avec des temps de vie limités (1 pour le premier routeur, 2 pour le deuxième, etc). Le routeur renverra une erreur, qui permettra de l'identifier. On effectue alors une moyenne. Cela permet d'établir la traceroute, soit le chemin de routeurs parcourus. Des routeurs peuvent ne pas répondre à ce genre de demande, ou bien à un certain quota. Généralement entre 15 et 20 sauts.

Cependant, la traceroute n'est pas garantie correcte, les paquets/sondes peuvent suivre des chemins différents (load balancing), ce qui pourrait créer des liens qui n'existent pas. Les routeurs peuvent cependant tenter de conserver le flux.

1.4.2 Perte (loss)

Les files (buffers) précédant un lien ont une capacité finie, et lorsqu'elles sont pleines, les nouveaux paquets sont jetés. Ils peuvent être éventuellement retransmis, mais ce n'est pas garantit.

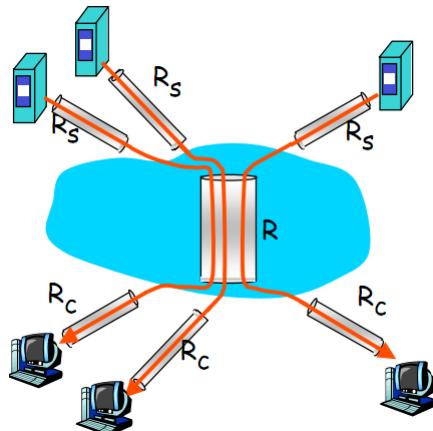
1.4.3 Débit

On peut avoir un débit instantané (point particulier dans le temps) ou moyen (sur des plus longues périodes de temps, dépend des débits relatifs des liens).

Il existe des algorithmes de contrôle de congestion qui permettent aux serveurs de ne pas débiter trop de paquets pour qu'il ne dépasse pas R_c (lien supposé inconnu par le serveur).

Le bottleneck est le maillon faible du réseau, c'est le lien qui bride le plus le débit dans le chemin end-end d'une transmission.

Autre problème : partage des ressources (R), afin de ne pas léser les autres flux. Généralement, ce sont les lignes R_s et R_c qui brident le plus.



1.5 Modèles et protocole

Le défi est de découper le système en couches de manière logique, sans être pénalisé par ce découpage (car trop de découpage engendre des problèmes de performances). Il y a un compromis à trouver entre efficacité et modularité (par couches ou un seul process pour tout).

L'avantage est qu'on peut identifier facilement chaque composante, et que la modularité garantit la facilité de maintenance et de mise à jour.

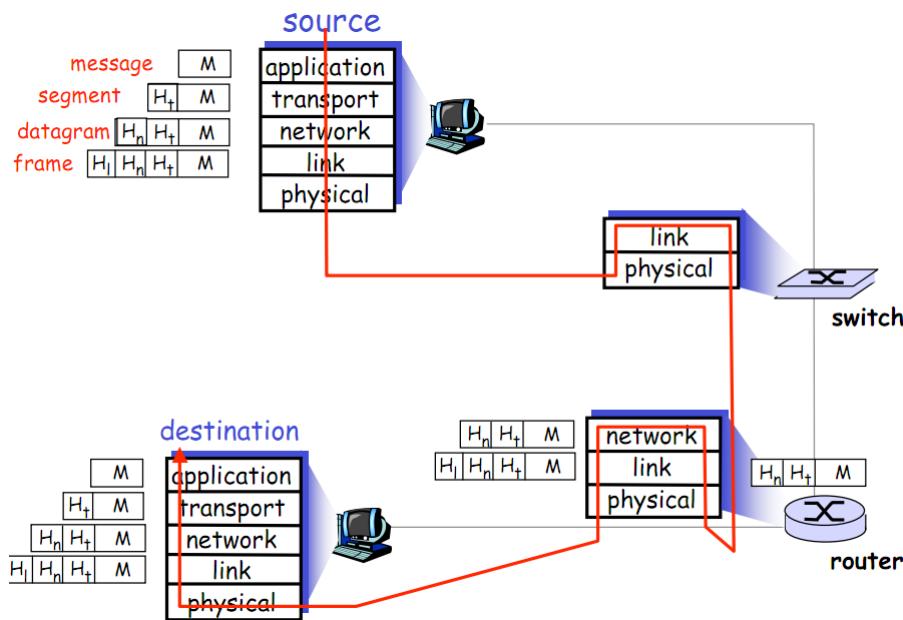
Essentiellement 5 niveaux :

1. les applications (distribuées)
2. le transport : transfert des messages d'une application à une autre. On y trouve TCP et UDP.
3. la couche réseau : transfert des segments de la couche transport d'un hôte source à un hôte destination. On y trouve IP.
4. les liens : transfert des datagrammes de la couche réseau d'un noeud à l'autre par des liens. Selon la nature du lien, il peut y avoir des mesures à prendre (par exemple introduire de la redondance s'il y a beaucoup d'erreurs). Cela peut ne pas être physique
5. la couche physique : dépend intrinsèquement du milieu physique de transmission (on considère les bits et non plus les paquets)

Le modèle ISO compte deux couches en plus : la présentation et la session. Modèle non repris au complet (l'application reprend ainsi les 3 premières couches)

Chaque couche ajoute des en-têtes au paquet qu'il faut transmettre, il y a une encapsulation.

- la couche transport ajoute des informations au message (code d'erreur, des séquences pour remettre tout dans l'ordre)
- la couche réseau ajoute aussi des infos, comme une enveloppe, avec l'adresse d'arrivée.
- la couche link permet de franchir des liens, éventuellement avec ses propres informations locales



Un routeur, quand il réceptionne le paquet, passe les couches à l'envers pour effectuer des vérifications. Puis il repasse par les couches et est envoyé. Ainsi de suite jusqu'à arriver à destination.

1.6 Histoire

Chapitre 2

La couche applicative

2.1 Principes des applications réseau

2.1.1 Création d'une application réseau

Ces applications n'existent que dans les systèmes d'extrémités, elles ne verront pas ce qu'il se passe entre elles lors des échanges. Il n'est donc pas nécessaire d'écrire une application pour un périphérique réseau (pas dans la couche applicative en tout cas), la partie software est très basse et n'exécutent pas des applications utilisateurs.

Grandes architectures

- client-serveur
- P2P

Dans les deux cas d'architecture, un processus peut être de type client (initialise une communication) ou de type serveur (qui attend un contact). Selon le scénario, une application peut être à la fois client et serveur (comme P2P).

En local, les communications entre deux processus se font avec l'OS, mais pour une application distribuée les communications se font par envoi de messages.

Architecture client-serveur Le serveur est toujours en ligne, avec une adresse IP permanente. Actuellement, virtuellement on a un seul serveur, mais physiquement plusieurs machines. Les serveurs sont contenus dans les data-centers.

Les clients communiquent avec le serveur, de manière intermittente ou non, avec une adresse IP dynamique ou non. Ils ne communiquent pas directement entre eux.

Architecture P2P Pas toujours de serveurs, les applications , les peers, dialoguent directement entre elles. Peut être facilement porté à grande échelle (plus il y a de clients, plus le réseau est performant et la charge distribuée), mais il y a de nombreux messages (car pas d'infrastructures), c'est difficilement gérable (une application peut être mobile, changer d'IP), et il y a la nécessité d'une connaissance à priori avant de lancer des requêtes.

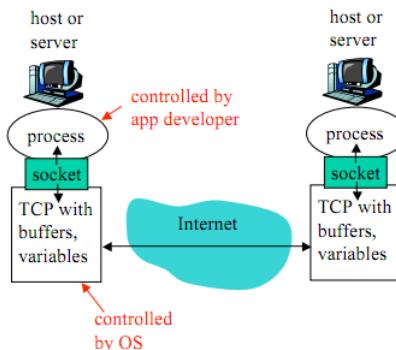
C'est une architecture très difficile à mettre en oeuvre. On a souvent des réseaux hybrides, avec des serveurs qui permettent d'amorcer les requêtes (Skype, messagerie instantanée).

Ex de la messagerie instantanée et Skype :

- client-serveur pour trouver une adresse tierce
- client-client pour la communication

Les sockets

C'est une interface qui permet la communication entre deux processus.



Deux questions à se poser : quel protocole de transport et quels paramètres utiliser.

Adressage de processus

Le premier problème est d'identifier le processus avec lequel on veut communiquer, problème d'adressage. On va assigner une adresse unique à chaque hôte, l'adresse IP. Elle ne suffit pas pour identifier le processus, on lui ajoute alors un numéro de port. Certains ports sont implicites (80 pour web, 25 pour mails).

Protocole

Un protocole applicatif définit

- les types des messages (réponse, requête, ...)
- la syntaxe des messages, quel sont les champs dans les messages et comment les champs sont délimités
- la sémantique, quel est la signification des informations dans les champs
- les règles pour savoir quand et comment les processus envoient et reçoivent des messages

Critères pour le choix d'un service de transport d'une application :

- critère de perte de donnée : certains applications peuvent le tolérer (audio), d'autres non (transfert de fichiers)
- critère de timing : certains applications nécessitent peu de délai (jeu, VOIp)
- critère de bande passante : certains applications nécessitent un minimum de bande passante (multimédia), d'autres non (transfert de fichier, applications "élastiques" qui peuvent s'adapter aux performances)
- critère de sécurité : encryptage, intégrité des données, ...

Les protocoles du domaine public sont spécifiés dans les RFCs (HTTP, SMTP, BitTorrent).

Deux grands choix de services très opposés :

- TCP :
 - orienté connexion : création d'une véritable connexion entre les processus clients et serveurs
 - transport fiable entre les applications
 - contrôle de flux (l'expéditeur n'engorge pas le receveur, il y aura de la place dans son buffer)
 - contrôle de congestion (pour éviter que les files d'attentes des routeurs intermédiaires ne débordent)
 - ne fournit pas des garanties sur les timing, le minimum de bande passante et de sécurité (il faut chiffrer au-dessus)
- UDP :
 - transfert non sûr de données
 - ne fournit pas le connexion de contrôle, de fiabilité, de contrôle de flux, de contrôle de congestion, de timing, de garantie de bande passante et de sécurité.

UDP est encore considéré car il y a au moins l'étape de démultiplexage vers un processus. Il faut compenser son manque de fiabilité au niveau applicatif.

2.2 Web et HTTP

HTTP (hyper-text transfer protocol) est un protocole de la couche applicative et est implémenté par un client et un serveur ; il y a envoi d'une requête (url) par le client et envoi d'une réponse par le serveur.

Avant d'effectuer des requêtes, il faut d'abord établir une connexion TCP entre le client et le serveur. Elle est ensuite fermée quand les requêtes sont terminées.

HTTP est sans état, c'est-à-dire qu'il ne maintient pas d'informations sur les requêtes des clients passées. S'il y avait des état, cela serait très compliqué [...]. Si un client demande plusieurs fois le même objet, il sera renvoyé plusieurs fois.

HTTP ne définit que la manière avec laquelle le serveur et le client communiquent et la structure des segments, pas la manière d'afficher une page web.

2.2.1 Type de HTTP

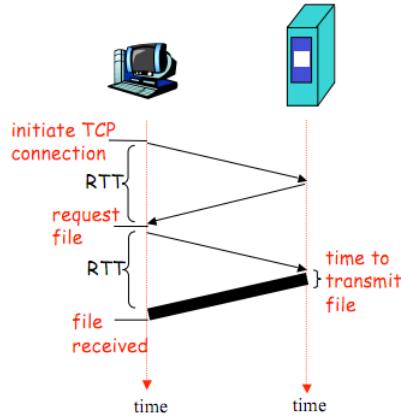
Plusieurs modes de fonctionnement de HTTP :

- HTTP non persistant : au plus un seul objet est envoyé via TCP
- HTTP persistant : plusieurs objets sont envoyés à travers une seule connexion client-serveur

Un RTT est le temps pour qu'un petit paquet soit envoyé par le client et renvoyé par le serveur.

Pour le HTTP non persistant, le temps de réponse comporte

- un RTT pour initialiser la connexion TCP
- un RTT pour la requête HTTP et la réponse
- le temps de transfert du fichier



Le problème est qu'il faut 2 RTT par objet, et qu'il y a chaque fois une réinitialisation de la connexion TCP.

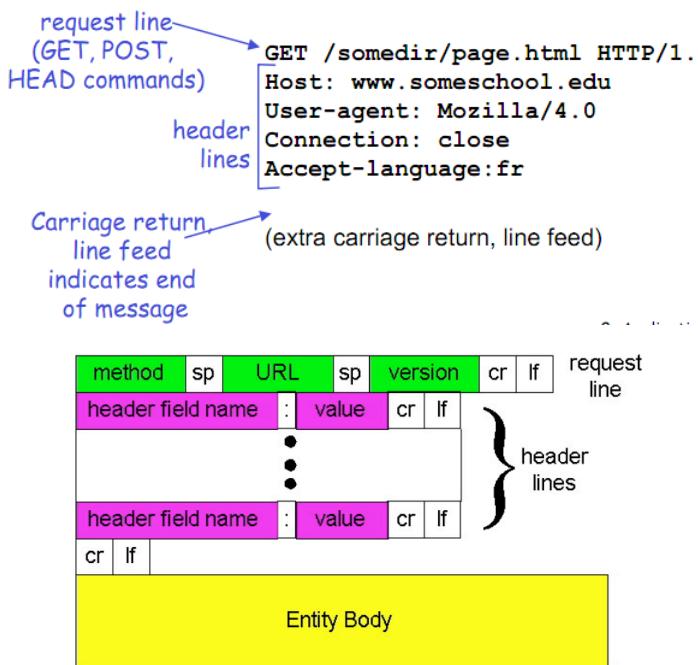
Avec HTTP persistant, la connexion reste ouverte après avoir envoyé la réponse. Du coup, en 1 RTT, on peut envoyer tous les objets demandés.

2.2.2 Message HTTP

Il y a deux types de message : les requêtes et les réponses.

Requêtes

Une requête est lisible (ASCII) et chaque ligne est séparée par un CRLF (carriage return, line feed), la dernière ligne de header en comportant un de plus pour séparer les headers du body.



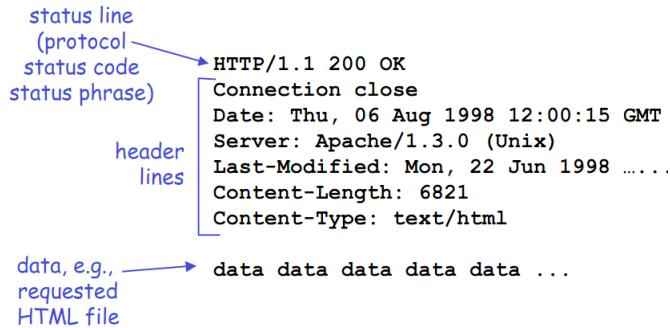
Le nom de l'hôte est économisé dans la ligne de requête car la requête atterrira sur l'hôte en question grâce à TCP, mais est quand même communiqué pour les systèmes intermédiaires de caching.

Il y a plusieurs méthodes pour envoyer de l'information :

- GET, avec les informations dans l'URL (?key=value)
- POST, qui est un GET, mais avec du contenu supplémentaire dans le body de la requête. On peut passer par le GET en étendant l'url.
- HEAD : renvoi une réponse vide, pour vérifier si l'objet existe.
- PUT et DELETE : injection et suppression de contenu à distance.

Il y a d'autres headers, comme `Connection: close|keep-alive`, `User-agent` ou `Accept-language`.

Réponses



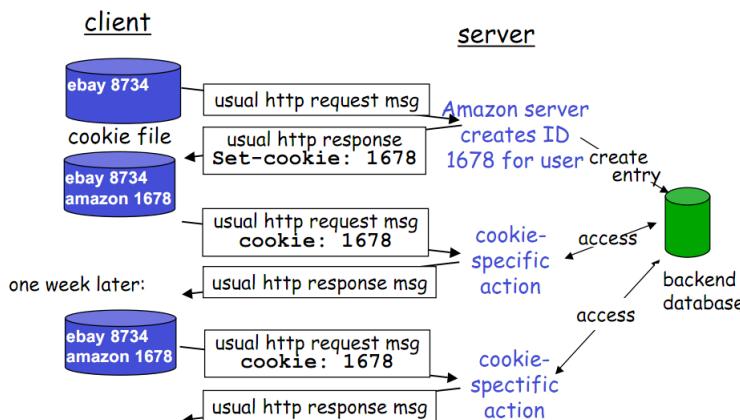
La première ligne de la réponse donne le statutLes principaux :

- 200 : ok ;
- 301 : moved permanently ;
- 400 : bad request ;
- 404 : not found ;
- 505 : HTTP version not supported.

2.2.3 Les cookies : état client-serveur

HTTP ne conserve pas de données, mais des données peuvent être gardées via les cookies chez le client. Le danger est la circulation de données en clair, il vaut mieux passer par une connexion TCP sécurisée.

Un cookie peut être défini par une réponse HTTP et consulté via une requête.

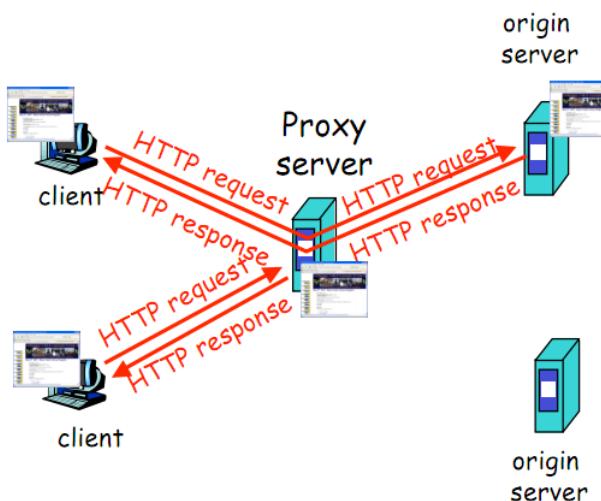


On peut donc garder un état avec un protocole simple, via un message.

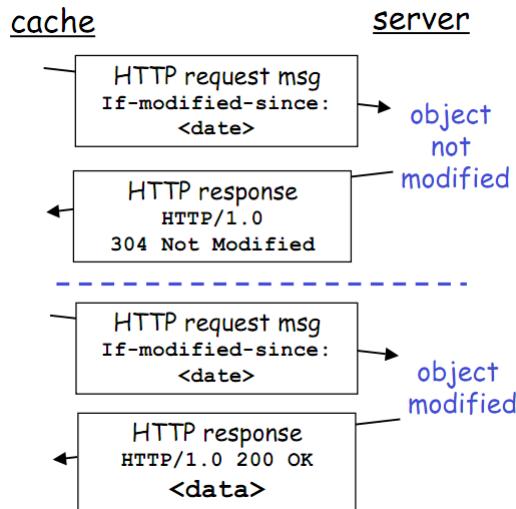
2.2.4 Cache web (proxy)

Ajout d'un état dans un système intermédiaire, pour augmenter les performances : cela permet de soulager les serveurs d'origine s'il y a beaucoup de requêtes, ou les réseaux.

Lors de l'envoi d'une requête HTTP à un cache, ce dernier retourne l'objet s'il est en cache, sinon il demande l'objet au serveur d'origine et ensuite le retourne au client. Il est généralement installé chez l'ISP ou dans une institution.



Il faut veiller à ne pas envoyer un objet qui n'est plus à jour ; à chaque requête, le cache devra avoir accès aux serveurs d'origine pour vérifier la date de dernière modification. Ce n'est pas pénalisant grâce au GET conditionnel : on n'envoie pas l'objet si la version en cache est à jour ; il y a un header dans la requête qui précise la date de dernière modification de l'objet en cache.



Dans la requête HTTP envoyée à un proxy, on re-précise le nom du serveur pour savoir sur quel serveur chercher. Le cache permet donc de

- réduire le temps de réponse si la connexion client-cache est bonne ;
- diminuer le trafic Internet.

2.3 DNS - Domain Name System

Protocole permettant de récupérer l'adresse IP d'un serveur à partir d'un nom symbolique. Les noms doivent être structurés et sont stockés dans une BDD distribuée.

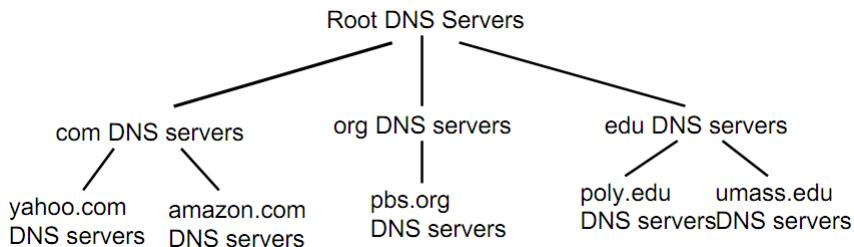
Cette fonctionnalité se situe au niveau applicatif, afin de mettre la complexité à la périphérie du réseau (et garder le cœur simple).

Services DNS :

- traduction nom de machine - ip (indirection)
- alias (nom synonyme pour un autre nom, chaînage) pour les noms de machine ou les serveurs mails
- répartition de charge ; le DNS répartit les requêtes sur différentes machines pour ne pas les surcharger. En pratique, un serveur DNS donne les différentes adresses IP d'un nom de domaine, à la prochaine requête la liste sera décalée. Cela permet d'avoir une réserve d'adresses IP.

Pourquoi le DNS n'est pas centralisé :

- en cas de panne, arrêt complet
- volume du trafic élevé
- les distances de connexion
- maintenance



Au départ, on s'adresse au serveur racine (root DNS server), qui va nous faire descendre dans l'arbre jusqu'à destination, en déléguant la responsabilité de gérer les sous-domaines aux autres serveurs DNS.

Le nombre de serveur DNS est relativement réduit, ce qui n'est pas suffisant pour supporter tout le trafic mondial, on élabore alors des mécanismes pour raccourcir.

La couche la plus haute de serveurs DNS après les serveurs root comporte les TLD (top-level domain) servers ; ils gèrent les .fr, .com, etc.

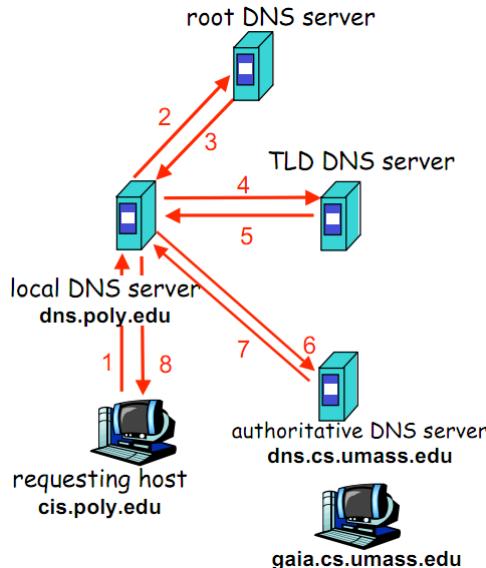
Il y a ensuite les serveurs DNS autoritaires, ceux des organisations concernées ou des ISP, avec les entrées DNS publiques et accessibles à tous.

2.3.1 LocalName Server

Pour une requête DNS, on l'envoie d'abord à un genre de proxyDNS, le local DNS server (pas au root DNS si le proxy connaît l'information). Il faut donc avant tout connaître ce serveur lorsqu'on se connecte, il est communiqué lors de l'établissement d'une connexion grâce au serveur DHCP.

Une demande DNS peut être itérative (la réponse est exacte ou un pointeur vers un autre DNS) ou récursive (réponse complète et exacte).

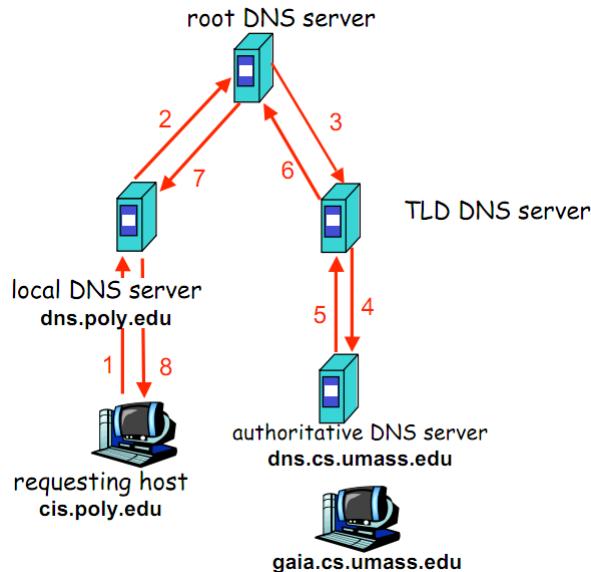
Demande itérative



La demande de l'hôte au local DNS server est récursive, le reste est itératif.

La requête est très vite traitée car il y a un risque de trafic élevé au niveau du local DNS, et donc de caching ; par exemple, si deux demandes de .edu, on passe directement à la requête 4.

Demande récursive



Avantages :

- le local DNS server est moins chargé ;
- le cache s'effectue dans beaucoup plus d'endroits différents, ce qui peut raccourcir le chemin, alors qu'en itératif il faut refaire tout le circuit.

Inconvénients :

- si jamais la chaîne est cassée, le problème est plus compliqué ;
- les serveurs DNS envoient et reçoivent plus de requêtes, et
- bloquent des ressources pendant plus longtemps (car il y a un état à retenir).

Dans les deux cas, il y a du DNS caching : les réponses sont stockées avec un TTL, afin d'accélérer les prochaines requêtes.

2.3.2 Enregistrements DNS

Format d'un Ressource Record (RR) : (name, value, type, ttl).

Le nom est un nom de machine, de domaine, etc, auquel on a une value (alias ou ip). Il y a un type, ainsi qu'un temps de vie (ttl).

Différents types :

- A : name est un nom d'hôte, value une adresse IP. Exemple : (un.domaine.be, 123.456.789.112, A)
- NS : name est un nom de domaine, value est un nom de machine, par exemple le nom de domaine sur DNS du domaine ; (host.be, dns.host.be, NS)
- CNAME : nom est un alias, value le nom canonique. Exemple : (host.be, relay.host.be, CNAME)
- MX : value est le nom du serveur de mail associé au nom de domaine name. Exemple : (host.be, mail.host.be, MX)

Une fois que le mapping DNS est effectué, il est mis en cache. Les serveurs TLD mettent généralement en cache les noms de serveurs, ainsi les serveurs de nom root ne sont pas souvent visités.

Pour un DNS non autoritaire, pour un domaine, on a deux entrées :

- une entrée NS, avec le domaine et le serveur DNS du domaine
- une entrée A, avec l'IP du DNS du domaine (donc l'entrée NS)

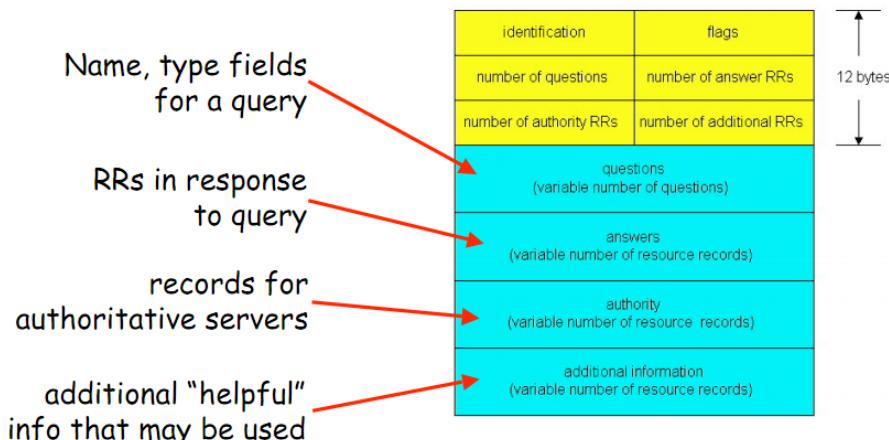
Un serveur autoritaire lui n'aura qu'une entrée, de type A pour l'hôte. Il en faudra une en plus pour un alias mail.

2.3.3 Messages DNS

Les requêtes et les réponses ont la même forme.

Dans le header, il y a un numéro d'identification (que contiendra aussi la réponse). Il y a également des flags (requête ou réponse, récursion désirée, récursion disponible et si la réponse émane de l'autorité). On spécifie aussi le nombre de questions, de réponses, d'enregistrements pour les serveurs de l'autorité et d'infos jugées utiles.

Le corps du message contient les questions, les réponses, les enregistrements et des informations supplémentaires.



Une réponse émane de l'autorité si elle ne vient pas d'un cache (confiance ou non, car risque de pollution DNS)

2.3.4 Ajout d'entrées DNS

Il faut enregistrer les noms dans un registre DNS auprès de sociétés. C'est l'ICANN qui les accréditent.

Lorsqu'on enregistre un domaine, il faut aussi spécifier les IPs des serveurs DNS primaires et secondaires. Par exemple, pour un serveur DNS primaire, on aurait (host.be, dns1.host.be, NS) et (dns1.host.be, 123.456.789.123, A).

Il faudra également une entrée de type A pour un serveur web, et une de type MX pour le serveur mail dans les DNS autoritaires (dns.host.be).

Ce sont les noeuds qui ont une ip, pas un domaine, car un domaine peut compter de nombreuses machines.

2.4 Programmation de socket

C'est une interface demandée par une application, contrôlée par l'OS, et permettant d'envoyer et recevoir des messages avec d'autres applications. Elle peut être de type TCP ou UDP.

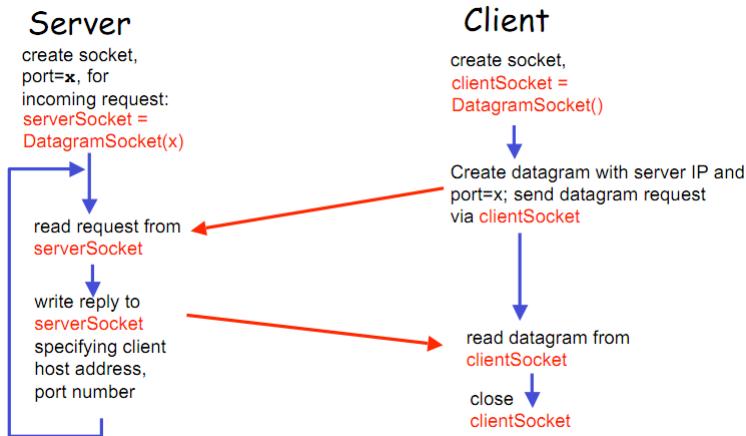
Pour communiquer avec un processus distant, ce dernier doit tourner et avoir un socket atteignable.

2.4.1 Avec UDP

Il n'y a pas vraiment de connexion. Le client attache explicitement l'IP et le port à un paquet. Côté serveur, il faut spécifier qu'on attend sur un port spécifique.

Le socket va ajouter des infos aux requêtes pour qu'elles soient autoportantes (on sait s'il faut répondre ou non, l'ip source, le port source, etc).

Il peut y avoir plusieurs clients (mise en attente si traitement en cours).

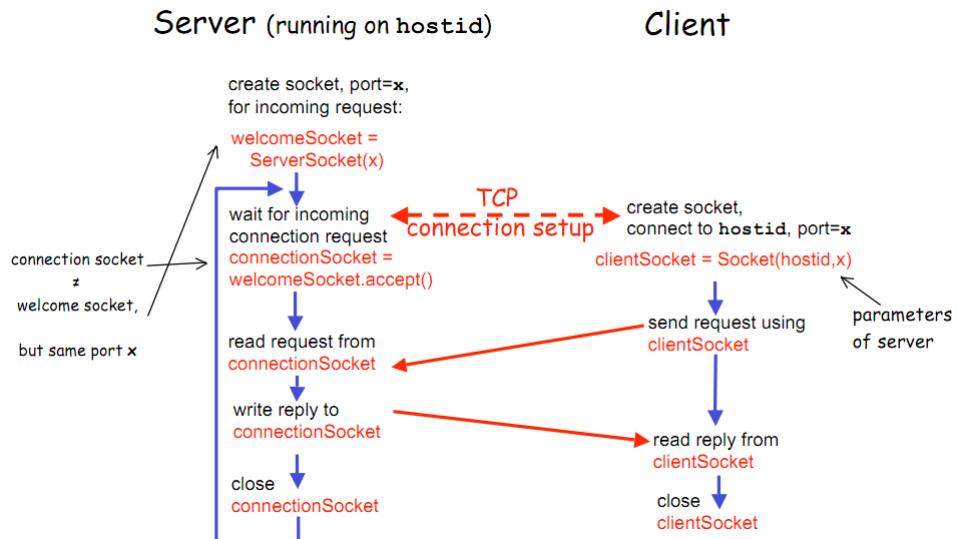


En Java, un socket UDP s'instancie avec `DatagramSocket()`, auquel on passe une instance de `DatagramPaquet`, qui contient l'adresse IP, le port, les données et leur longueur.

Côté serveur, le socket d'accueil s'instancie avec `DatagramSocket(port)`.

2.4.2 Avec TCP

Comme UDP, il est nécessaire que le processus fasse tourner un processus d'écoute d'un socket (welcoming socket). Le client crée un socket aussi qui, contrairement à UDP, est dédié au serveur auquel on veut communiquer (avec UDP, le serveur peut être différent pour le même socket) : il y a un lien explicite. Du coup, une fois le socket créé, plus besoin de re-préciser la destination.



On ne peut pas avoir deux connexions associées au même socket, sinon on ne sait pas vers quel connexion envoyer les informations. La solution est que le welcoming socket crée un nouveau socket et l'associera à la connexion, on a ainsi plusieurs sockets associés au même port.

En Java, le socket client s'instancie avec `new Socket(host, port)`.

Le socket d'accueil est `ServerSocket(port)`, le socket qui prend le relais est généré avec la méthode `accept()` du socket d'accueil.

Chapitre 3

La couche de transport

3.1 Services de la couche de transport

La couche de transport donne une communication logique entre des processus tandis que la couche réseau fournit une communication logique entre des hôtes, des machines.

La couche de transport segmente les messages à envoyer venant de la couche applicative en segments vers la couche réseau. Il réassemble également les segments reçus par la couche réseau et les redirige vers la couche applicative.

Il y a plus d'un moyen de transport ; pour Internet, il y a deux grands protocoles :

- TCP (Transmission Control Protocol), qui fournit un contrôle de flux et de congestion, est fiable et livre dans l'ordre les paquets. Il nécessite un établissement de connexion ;
- UDP (User Datagram Protocol) : livraison non ordonnée, non fiable. On fait au mieux.

Ces deux protocoles ne garantissent pas les délais ni la bande passante.

3.2 Multiplexage et démultiplexage

Pour un hôte receveur, le démultiplexage est le fait de délivrer les segments reçus au bon socket.

Pour un hôte expéditeur, le multiplexage est le fait de collecter les données de plusieurs sockets et de les encapsuler avec un header pour ensuite les passer à la couche réseau

3.2.1 Démultiplexage

Un port est codé sur 16 bits. Les 1024 premiers ports sont réservés (HTTP : 80, FTP : 21, etc).

Seuls les ports source et de destination sont spécifiés dans un segment, or il devrait y avoir les deux ip. Elles ne sont pas spécifiées par la couche de transport, lorsqu'un segment est passé par la couche réseau, on donne à part les deux IP.

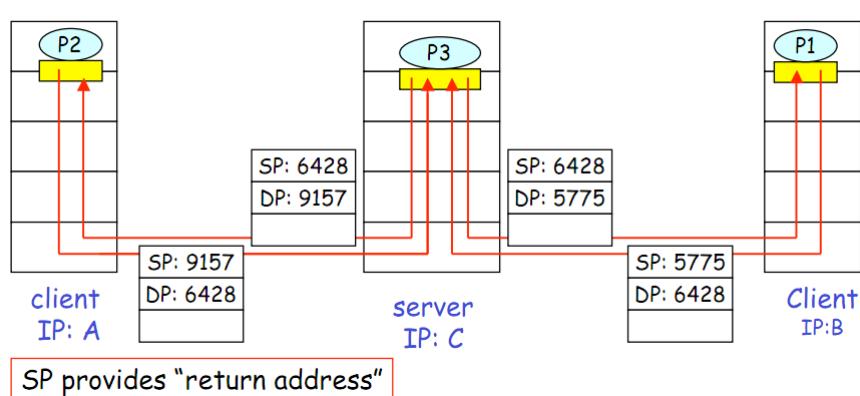
Chaque datagramme a donc une adresse ip source, une adresse ip de destination et un segment de la couche de transport, qui spécifie les ports source et de destination.

UDP Un segment UDP est identifié par le port source et le port destination. On n'utilise que le port de destination pour rediriger le segment.

Un socket UDP est identifié par deux informations : l'adresse IP de destination et le port de destination.

Lorsqu'un hôte reçoit un segment UDP, il regarde le port de destination dans le segment, et le redirige vers le socket associé au port. C'est du démultiplexage sans connexion.

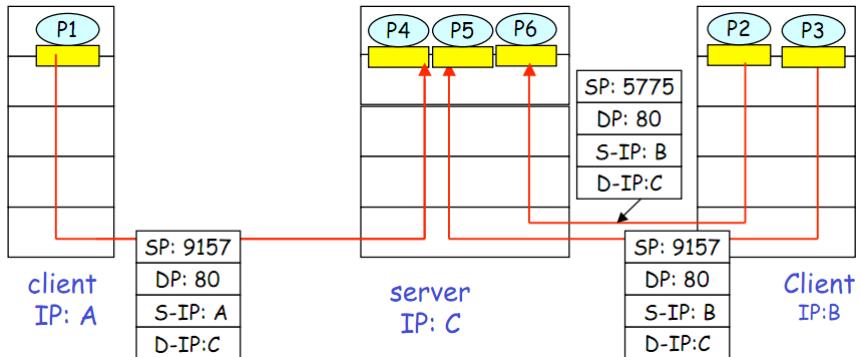
```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



TCP Un socket TCP est identifié par 4 informations : les IPs source et destination et les ports source et destination. L'hôte receveur utilise les quatre informations pour rediriger le segment vers le socket approprié. C'est un démultiplexage orienté connexion.

L'adresse IP est nécessaire pour pouvoir différencier les sockets attachés au même port, notamment quand ils sont générés par un welcoming socket. De plus, cela permet de parer au cas où deux clients auraient le même port source.

Ainsi, les serveurs web ont différents sockets sur le port 80 pour connecter chaque client. Les connexions HTTP non persistantes auront différents sockets à chaque requête



3.3 UDP

Il faut UDP au minimum pour le démultiplexage, et pour ne pas utiliser nécessairement TCP et avoir ses ralentissements. On fait au mieux, en contrepartie les segments UDP peuvent être perdus ou délivrés dans le désordre

Raisons d'utilisation d'UDP :

- pas de connexion établie, moins de délai
- très simple, il n'y a pas d'état chez l'expéditeur ou le receveur
- le header des segments est léger
- pas de contrôle de congestion, UDP peut aller aussi vite qu'on veut

C'est un protocole orienté sans connexion, car il n'y a aucun établissement entre le receveur et l'expéditeur UDP, et chaque segment UDP agit indépendamment des autres.

UDP est généralement utilisé dans les applications multimédia en streaming, qui sont tolérantes à la perte et sensibles au débit. Il est utilisé aussi pour DNS et SNMP (network management).

Pour garantir un minimum de fiabilité, au niveau de l'application, on ajoute au segment un checksum pour vérifier si le paquet est arrivé en totalité ou non, pour ainsi ne pas propager l'erreur. C'est similaire l'utilisation d'un bit de parité, où on garantit qu'il y a toujours un nombre pair de 1. Ce n'est pas totalement fiable, vu qu'on peut échanger des bits ou diminuer le nombre pair de 2.

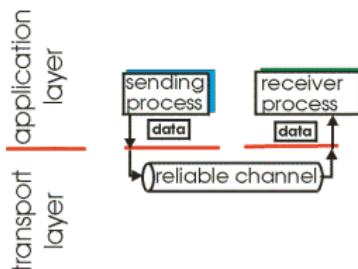
On va supposer qu'il peut toujours y avoir une erreur qui passe, quelque soit le nombre de bits de contrôle et les moyens de vérification, car le segment peut être modifié pour que la vérification soit correcte.

Un test facile est de sommer les bits, et de complémer le somme pour former le checksum. Ainsi, quand le receveur récupère le segment, en sommant tous les bits, en tombant sur 0 il sera vérifié. Ce n'est pas à 100% fiable, par exemple +1 pour un nombre et -1 pour l'autre.

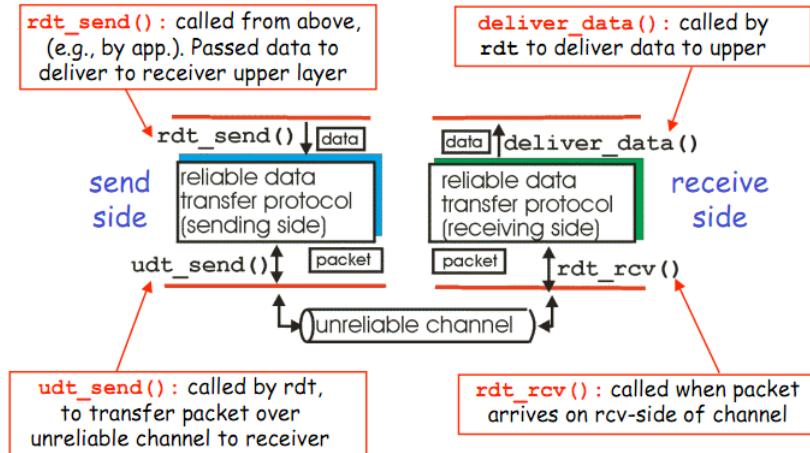
Ce léger test permet un temps de calcul rapide. Les erreurs résiduelles ne sont que peu probables, car auparavant il y a déjà des vérifications et des corrections dans les couches inférieures, ou bien parce qu'il est rare d'avoir des erreurs dans un protocole aussi léger.

3.4 Principes de fiabilité lors d'un transfert de données

La fiabilité est importante dans les couches applicatives, de transport et de lien. Le service fourni est un canal fiable entre les deux applications.

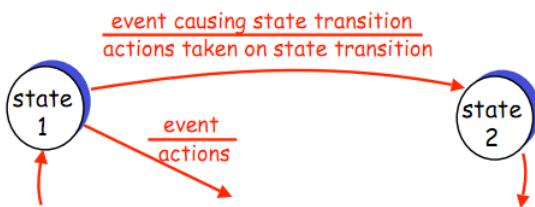


On va définir 4 fonctions pour l'implémentation du service.



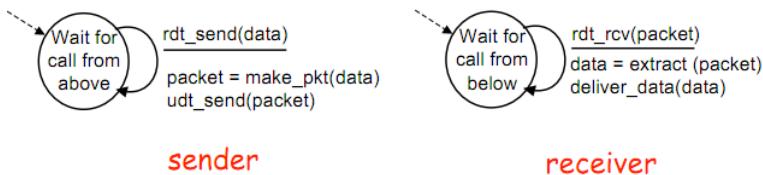
Les caractéristiques du canal non fiable vont déterminer la complexité du protocole de transfert de donnée fiable (rdt - reliable data transfer).

On va représenter le protocole par une machine d'état finie.



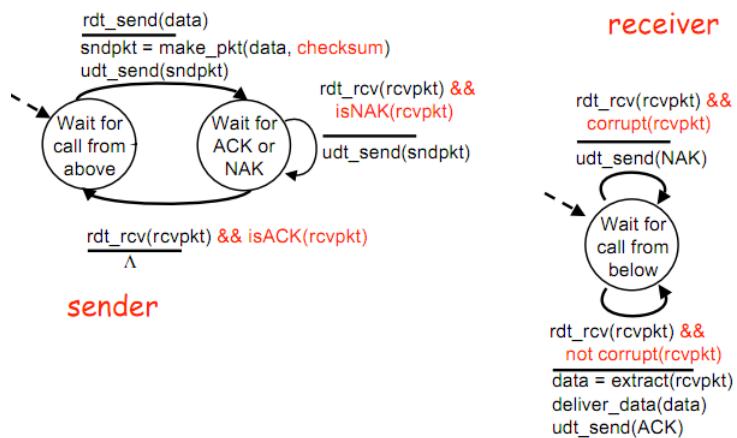
3.4.1 1.0 - Canal fiable

On suppose qu'il n'y a pas d'erreur dans les bits, ni de perte de paquet.



3.4.2 2.0 - Canal avec erreurs binaires

Introduction d'un checksum dans le paquet. On renverra un paquet de confirmation (NAK ou ACK) pour savoir s'il faut renvoyer le paquet ou non.



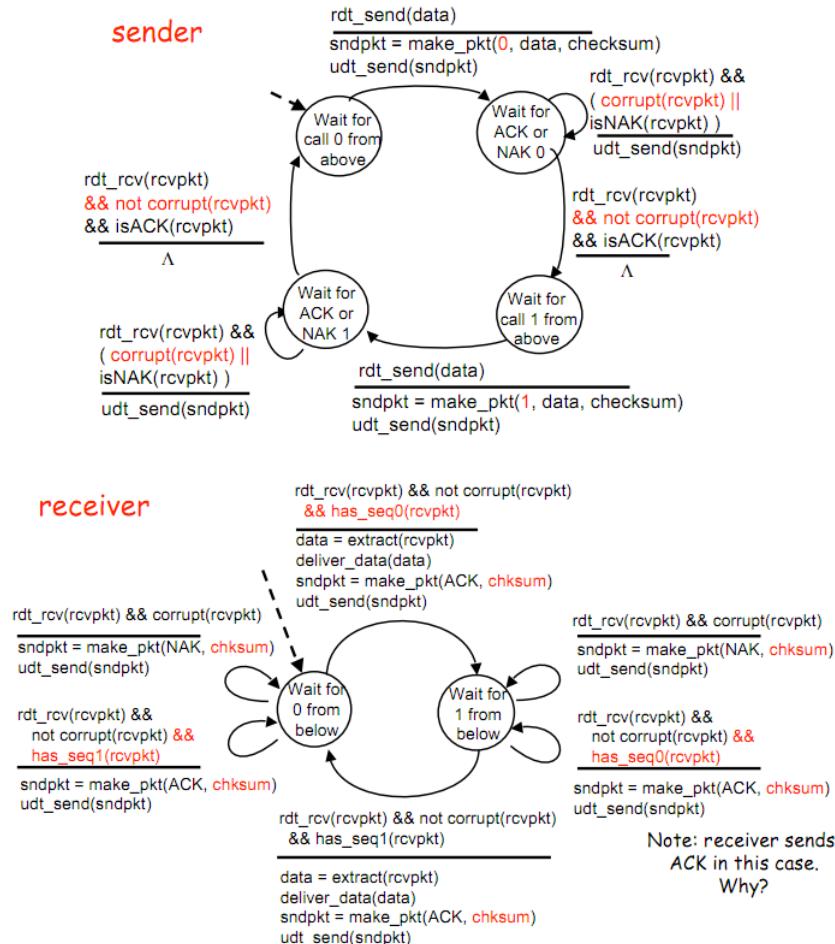
Ainsi, l'expéditeur envoie un paquet, et attend la réponse du récepteur.

Les protocoles de ce type sont dit **stop-and-wait** : l'expéditeur n'envoie aucun nouveau segment tant que le récepteur ne les a pas confirmés.

3.4.3 2.1 - Erreur d'acquis

Le problème est qu'on pourrait ne pas savoir comment interpréter la confirmation, car le message pourrait être erroné, ce qui peut conduire des doublons de paquet ; un ACK ou un NAK corrompu sera détecté avec le checksum et jet, mais l'envoyeur ne sait pas s'il doit retransmettre ou non.

La solution est d'insérer dans les paquets un numéro de séquence, et d'acquitter ce numéro. Ainsi, les paquets doublons seront jetés.

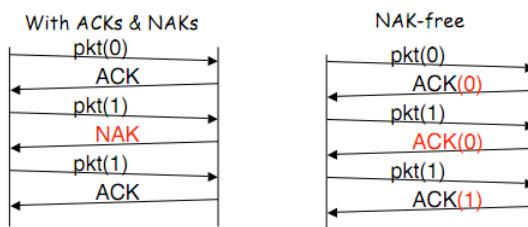


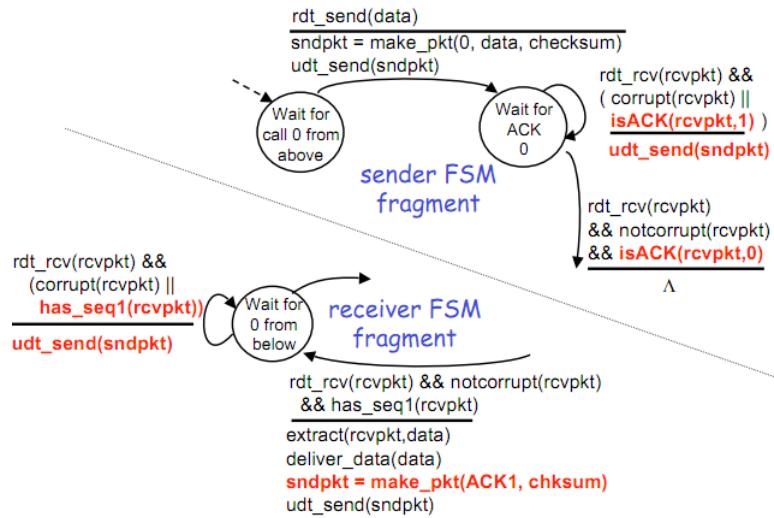
Le récepteur envoie un ACK s'il reçoit un paquet non corrompu qui a une mauvaise séquence car cela signifie qu'il n'a pas reçu l'ACK précédent, alors que le récepteur a bien eu le paquet.

Dans le cas de l'envoi/réception d'un et un seul paquet la fois, un identifiant modulo 2 suffit amplement, vu qu'il y a une alternance.

3.4.4 2.2 - Protocole sans NAK

Suppression des NAK, et lorsqu'on ACK, on ACK le dernier numéro qui était correct.

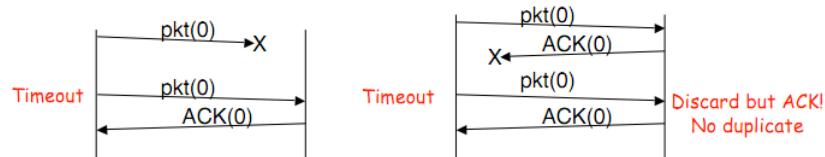




3.4.5 3.0 - Canal avec pertes et erreurs

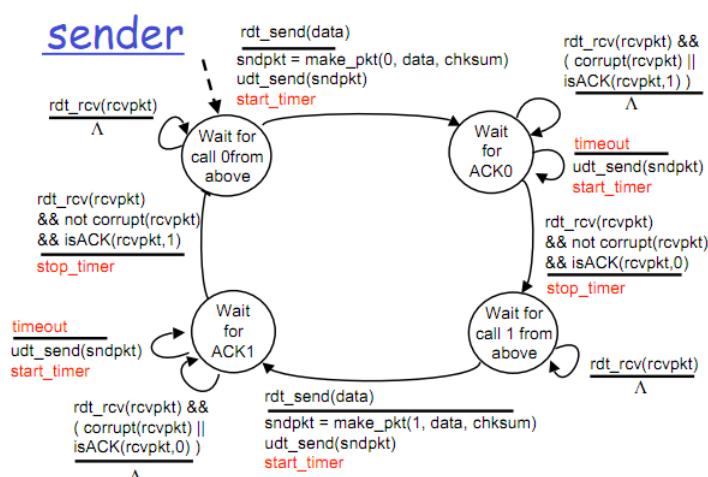
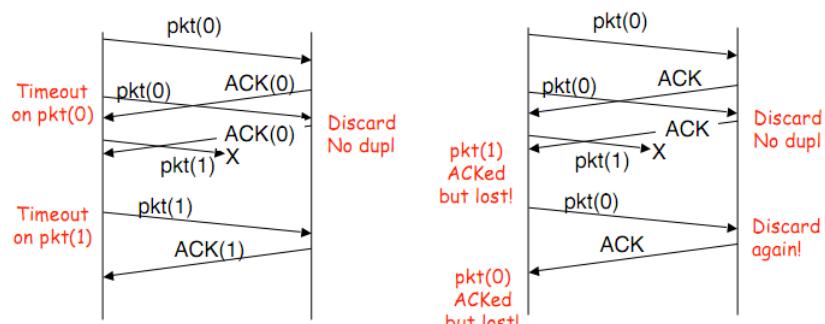
On suppose un réseau avec des erreurs et des pertes de messages.

La solution typique est un timeout pour le ACK ; lorsque le temps s'est écoulé et si on n'a pas reçu d'accusé de réception, on renvoie le paquet. La transmission peut être longue, au point que le timeout se déclenche : il y a risque de doublon, mais l'alternance évite ce problème.

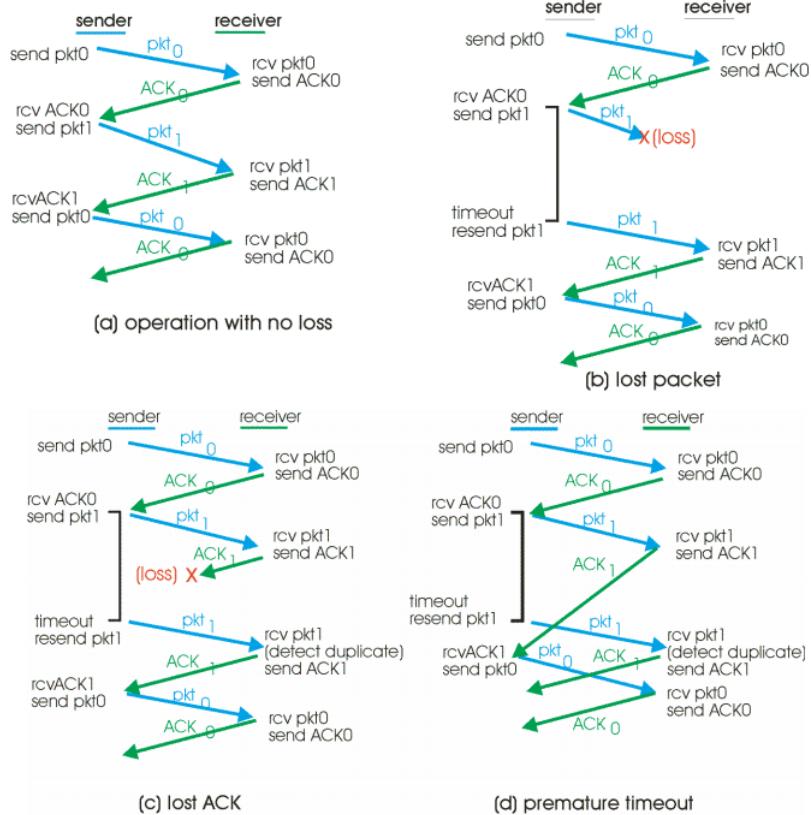


On a 4 éléments : checksum, numérotation paquet, numérotation accusé et timeout.

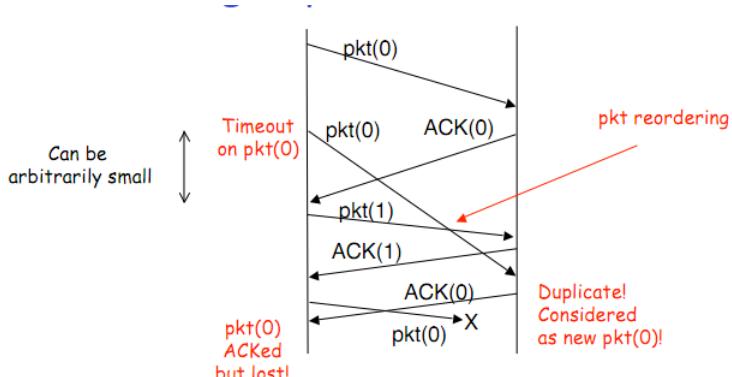
Dans une certaine mesure, on peut supprimer le numéro d'accusé ; lorsqu'on reçoit un paquet erroné, on n'envoie aucun accusé et on fait comme si le paquet n'était jamais parvenu. Cependant, il peut y avoir des doublons qui peuvent survenir et faire perdre des paquets.



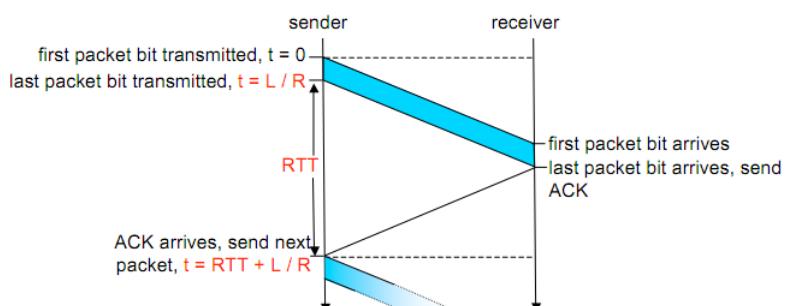
Ce protocole est appelé Alternating-bit protocol (1969).



Le protocole est correct si on suppose que les paquets envoyés sont reçus dans l'ordre, c'est-à-dire qu'un paquet envoyé avant un autre ne le dépasse jamais. Une solution est d'augmenter la durée des timeouts (mais problèmes de performances), une autre (meilleure) est d'utiliser plus de numéros.



Performances du stop-and-wait RTT : round-trip time, temps de propagation du paquet et de réception de l'accusé.



$\frac{R \cdot RTT}{2}$ = produit débit-délai. R.RTT représente le nombre de bits en transit dans le réseau, ce que l'on peut "stocker" dedans.

Il peut en effet y avoir un décalage ; on peut avoir tout envoyé du côté de l'émetteur, mais n'avoir encore rien reçu chez le récepteur.

On a l'utilisation chez l'expéditeur :

$$U_{\text{sender}} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{1}{1 + \frac{R.RTT}{L}}$$

Cependant, vu le caractère stop-and-wait, l'utilisation est très faible, le réseau physique est sous-exploité.

- Example: 1 Gbps link, 15 ms end-to-end prop. delay, 1KB packet:

$$T_{\text{transmit}} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8\text{kb}}{10^{*}9 \text{ bps}} = 8 \mu\text{sec}$$

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- 1KB pkt every 30 msec \rightarrow 33kB/sec throughput over 1 Gbps link

3.4.6 Protocole avec pipeline

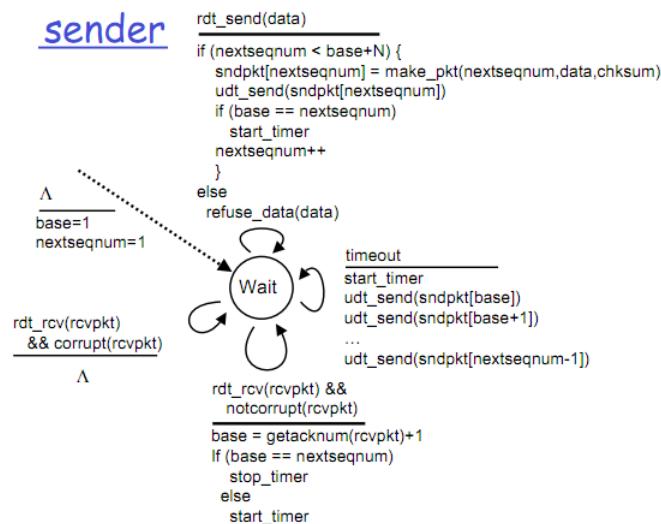
Pour augmenter les performances, on introduit du parallélisme, avec un pipeline : les paquets sont envoyés l'un après l'autre, sans attendre les accusés.

Deux grandes catégories de protocole de pipelining : le go-back-N et le selective repeat.

Go-Back-N

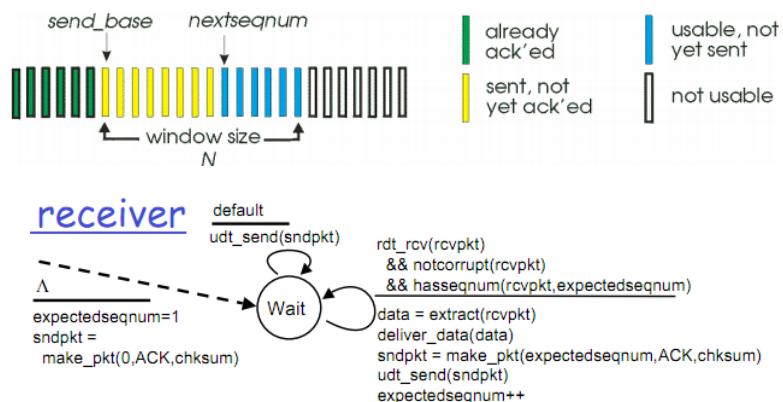
- l'émetteur envoie jusqu'à N paquets non accusés dans le pipeline, numérotés dans la fenêtre temporelle.
- le récepteur est supposé peu élaboré, il n'a qu'un seul buffer. Du coup, s'il y a une perte, tout ce qui suit est ignoré. Il envoie des accusés cumulatifs ("tout ok jusqu'au paquet X").
- l'émetteur a un timer, pour attendre jusqu'au dernier paquet, et renvoie ceux qui se sont perdus à partir du dernier accusé reçu.

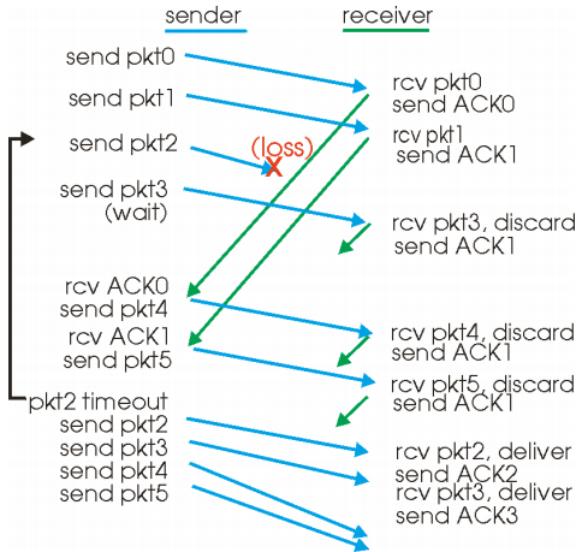
Pas optimal s'il y a des pertes dans le réseau.



Quelle taille de fenêtre maximum utiliser ? On ne dépasse jamais K comme numéro, sinon risque de confusion entre les numéros des paquets.

On ne peut pas aller jusque K pour N , la limite est $K - 1$.

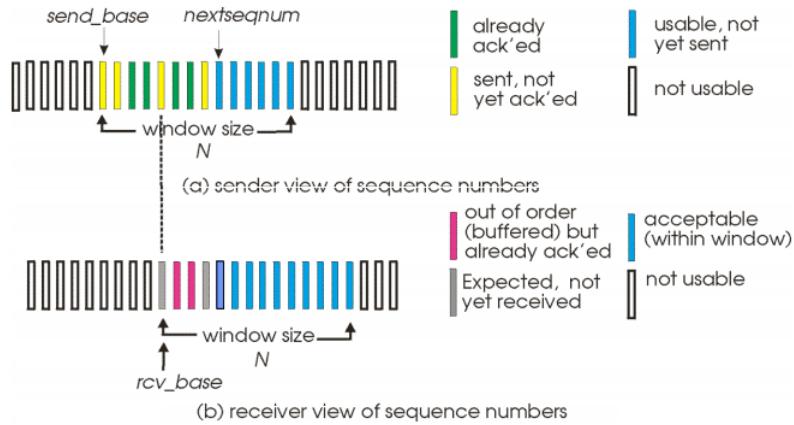




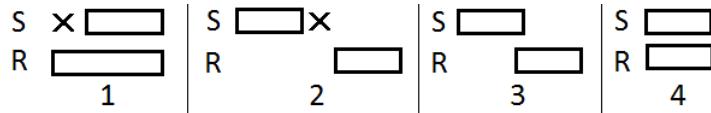
La taille de fenêtre maximale $N \leq K - 1$.

Selective Repeat

- idem pour l'émetteur
- le récepteur maintient un buffer et envoie des accusés de réception individualisés. S'il y a une perte, tout est bufferisé jusqu'à recevoir le paquet perdu.
- l'émetteur a un timer pour chaque paquet. S'il expire sans qu'on ait reçu un accusé, on renvoie le paquet.



Plusieurs situations extrêmes sont possibles.



1. impossible, car l'émetteur a envoyé et confirmé des paquets qui n'ont pas été reçus par le récepteur. Idem pour dessin 2 : le récepteur a reçu et confirmé des paquets que le sender n'a pas encore envoyés.
3. Possible dans le cas où le récepteur a accusé les paquets, mais où l'émetteur n'a encore rien reçu comme confirmation. Cas limite et pris en compte dans "pkt n in [rcvbase - N, rcvbase - 1]", pour que l'émetteur ne soit pas indéfiniment bloqué.
4. situation initiale.

Avec une fenêtre de taille $\frac{K}{2}$, pas de problèmes car les fenêtres de réception et d'émission ne se chevauchent pas.

Taille de fenêtre maximale

Si N_s est la taille de la fenêtre de l'émetteur, N_r celle du récepteur et K la taille de la séquence, il faut en général que $N_s + N_r \leq K$.

- Go-back-N : $N_r = 1$, $N_s \leq K - 1$
- Selective repeat : $N_s = N_r \leq \frac{K}{2}$
- Alternating-bit : $K = 2$, $N_s = N_r = 1$

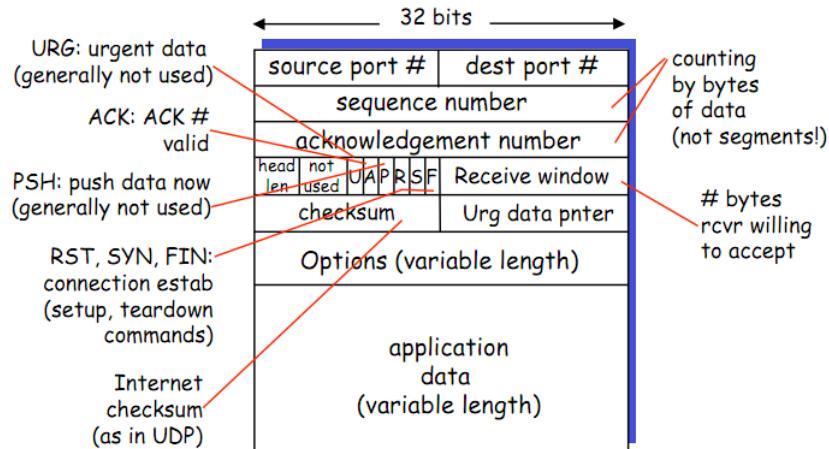
3.5 TCP

Protocole point-à-point (un émetteur et un récepteur), qui identifie tous les bytes par un numéro (protocole byte stream). Il y a du pipelining, avec une mémoire tampon et un contrôle de flux (afin que le récepteur ne soit pas surchargé). Il est full-duplex, la connexion est bidirectionnelle et les deux entités peuvent envoyer/recevoir en même temps. On définit le MSS, la taille maximale de segment.

Protocole orienté connexion. La fenêtre continuera à glisser, mais sa taille sera modifiée par TCP, en fonction de l'application.

Le MSS est la quantité maximale de donnée venant de la couche applicative dans le segment, et non la taille maximale d'un segment TCP incluant les headers. Il est généralement déterminé par la trame (de couche lien) la plus large qui peut être envoyée par l'expéditeur, autrement dit le MTU (maximum transmission unit). Avec cette taille, on est sûr qu'un segment TCP rentrera dans une seule trame de la couche lien.

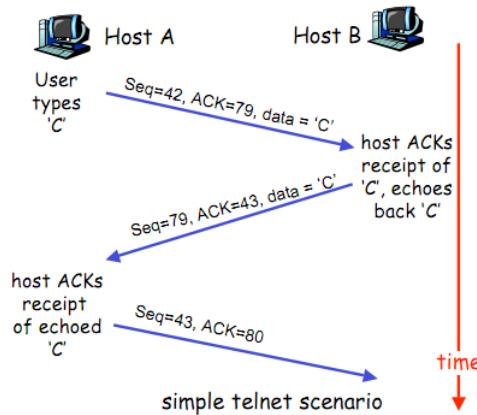
3.5.1 Structure de segment



Lors de la réception d'un byte, on le récrit (confirmation visuelle de la réception). Le ACK est cumulatif (comme go-back-N).

Le numéro de séquence n'est pas celui de la séquence de segment, mais celui des bytes transmis. Par exemple, pour un fichier de 3200 bytes et pour un MSS de 1000 bytes, on aura les numéros de séquence 0, 1000, 2000 et 3000.

L'aknowledgement number est le prochain numéro de séquence attendu. Il est cumulatif ; par exemple, si pour un MSS de 1000 bytes on a les numéros 0 et 2000, le numéro d'ACK sera 1000. Le prochain numéro sera 2000 ou 3000, selon que l'implémentation jette les segments désordonnés ou les garde (ce qui est généralement le cas).



Si des segments arrivent dans le désordre, le protocole ne définit volontairement rien, c'est à l'implémenteur de gérer ce cas. Le fait de rien garder peut être utile selon les cas (par exemple beaucoup de connexions TCP).

3.5.2 Timer

Le timer porte sur le plus ancien byte non acquitté. Il faut qu'il soit plus long que le RTT, mais pas trop court (sinon il y aura des retransmissions inutiles) ni trop long (sinon réaction lente face à des pertes de segments).

Le problème est délicat pour TCP : il faudrait connaître la moyenne et l'écart-type des délais, afin d'estimer le RTT. TCP va effectuer des mesures et adapter dynamiquement son timer. Les mesures se font en envoyant un timestamp dans un paquet.

Les délais peuvent fortement varier, on va donc effectuer une moyenne exponentielle pondérée en temps réel de l'estimation.

$$\text{RTT estimé} = (1 - \alpha) \cdot \text{RTT estimé} + \alpha \cdot \text{RTT échantillon}$$

Généralement, $\alpha = 0.125$. La variance sera aussi estimée, de la même façon.

A cette estimation on ajoute une marge de sécurité. Ainsi, plus il y a des grandes variations dans l'estimation du RTT, plus cette marge sera grande.

On estime donc de combien le RTT échantillon dévie du RTT estimé.

$$\text{RTT dévié} = (1 - \beta) \cdot \text{RTT dévié} + \beta |\text{RTT échantillon} - \text{RTT estimé}|$$

Généralement, $\beta = 0.25$. On a enfin l'intervalle d'un timeout :

$$\text{Intervalle timeout} = \text{RTT estimé} + 4 \cdot \text{RTT dévié}$$

3.5.3 Fiabilité

TCP propose donc un service de transfert fiable, par dessus le service non fiable d'IP. Les segments sont pipelinés et lors de la réception d'un segment, les données sont retransmises.

S'il y a un timeout, on renvoi le segment qui l'a causé (mais seulement celui-là, alors que le go-back-n renvoi tout ce qu'il y a après) ; il n'y a qu'un seul timer de retransmission, qui correspond au plus vieux segment non acquitté.

Voici ce qui se passe en fonction d'un évènement :

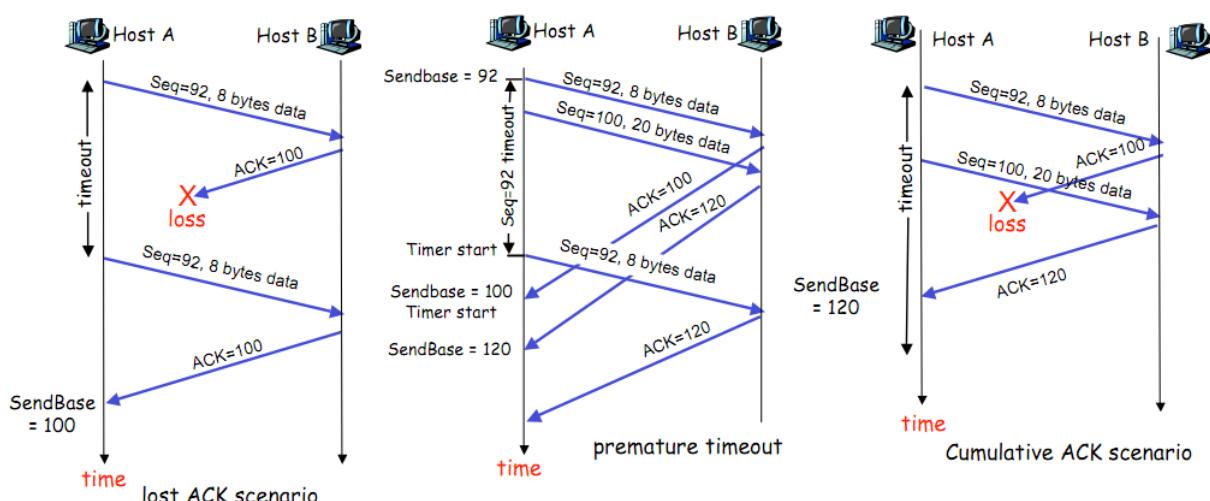
- Réception de données d'une application
 1. création d'un segment avec un numéro de séquence
 2. démarre le timer si ce n'est déjà pas le cas

- Timeout
 1. retransmission du segment qui a causé le timeout
 2. redémarrage du timer

- Réception d'un ack : s'il y a des segments précédents non acquittés,
 1. mettre à jour ce qu'on sait acquitté
 2. démarrer le timer s'il y a des segments en circulation

Une différence par rapport au go-back-N est que ce dernier acquitte le dernier paquet reçu dans l'ordre, alors que TCP acquitte la prochaine séquence attendue.

Quelques scénarios :



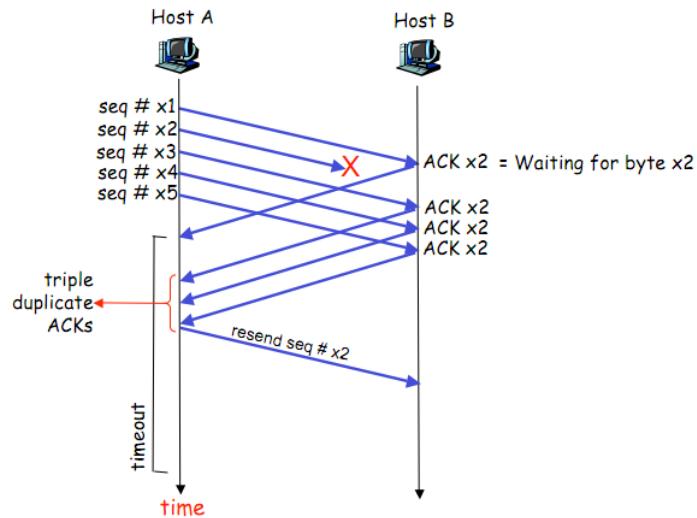
Génération d'ACK TCP :

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq #. Gap detected	Immediately send duplicate ACK , indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap

La période de timeout peut être parfois longue, ce qui entraîne un long délai avant le renvoi d'un paquet perdu. On introduit la retransmission rapide, qui consiste à renvoyer un segment avant que le timer n'expire.

La détection se fait sur des doublons d'acquis : vu que l'expéditeur envoie plusieurs segments à la fois, il recevra plusieurs acquis. Si un segment est perdu, il aura nécessairement des doublons d'acquis.

Ainsi, s'il reçoit trois fois le même ACK pour le même segment, on supposera que le segment d'après est perdu, et il sera donc renvoyé.

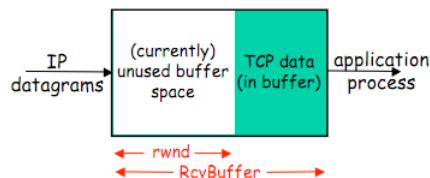


Les retransmissions sont donc déclenchées lorsque le timer arrive à expiration ou lorsqu'il y a des doublons d'acquis.

3.5.4 Contrôle de flux

Le but du contrôle de flux est que l'expéditeur ne sature pas le buffer du récepteur en lui transmettant trop de données trop vite. C'est un service qui adapte la vitesse d'envoi de l'application expéditrice en fonction de la vitesse d'absorption de l'application réceptrice.

Le côté expéditeur de la connexion TCP possède un buffer de réception, qui peut prendre du temps pour être lu par les applications. On va supposer que TCP rejette les segments dans le désordre



Le but de TCP est de ne jamais dépasser un buffer, il doit donc toujours vérifier

$$\text{Dernier byte reçu} - \text{Dernier byte lu} \leq \text{Taille du buffer de réception}$$

Initialement, la taille de la fenêtre est la taille du buffer. L'espace non utilisé dans le buffer définira la taille de la fenêtre de réception :

$$\text{Taille de la fenêtre} = \text{Buffer de réception} - (\text{Dernier byte reçu} - \text{Dernier byte lu})$$

Le récepteur va donc informer l'émetteur de l'espace inutilisé de son buffer en incluant rwnd dans le header des segments qu'il lui envoie. L'émetteur va alors limiter le nombre de bytes non acquittés à rwnd.

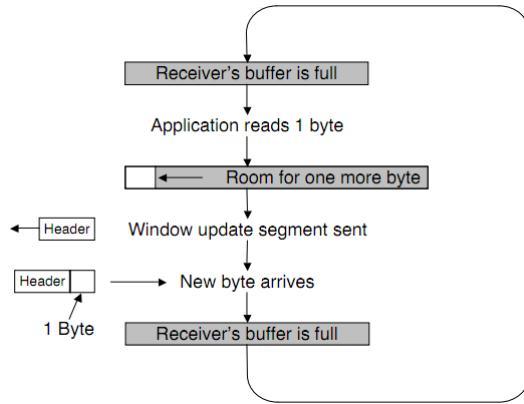
L'émetteur va garder comme variable le dernier byte envoyé et le dernier byte acquitté. La différence donne la quantité de données non acquittées qu'il a envoyé. Cette différence doit être inférieure à la taille de la fenêtre :

$$\text{Dernier byte envoyé} - \text{Dernier byte acquitté} \leq \text{Taille de la fenêtre}$$

Des problèmes surviennent quand les données passent par les sockets un byte à la fois ; un segment d'un byte est envoyé par RTT.

Pour éviter ce trafic élevé pour peu de données, on utilise l'algorithme de Nagle, qui permet du groupage de bytes à envoyer. Cela permet de se protéger d'émetteurs qui envoient byte par byte. Ainsi, on envoie le premier byte, on bufferise le reste en attendant l'ACK puis en envoie tout.

Cela ne suffit pas pour se protéger de mauvais receveurs, qui lisent le buffer byte par byte (et qui renverrait chaque fois un message).



Pour y remédier, il y a l'algorithme de Clarke : si le buffer est à moitié vide ou s'il y a de la place pour un segment de taille maximum, on avertit l'émetteur (et non plus à chaque byte lu)

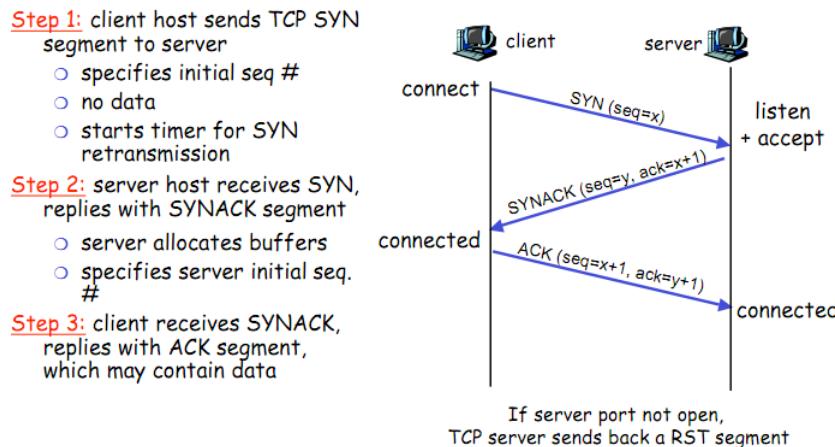
Autre problème : soit un émetteur n'ayant rien à envoyer à un récepteur dont le buffer est plein. L'émetteur a donc une taille de fenêtre nulle.

Si le récepteur vide (ou fait de la place) le buffer et qu'il n'a rien à envoyer à l'émetteur, ce dernier ne saura jamais qu'il peut envoyer des données.

Pour parer cette éventualité, on impose que l'émetteur continue à envoyer des segments d'un byte quand la taille de fenêtre du récepteur est nulle. Ainsi, s'il y a des ACKs, l'émetteur aura une taille de fenêtre à jour.

3.5.5 Gestion de la connexion

On doit initialiser des variables lors de l'établissement d'une connexion, par exemple le numéro de séquence du premier byte. TCP ne commence pas systématiquement au même numéro pour chaque connexion. Cela s'effectue lors des premières connexions, lors de la création des sockets.



La particularité de TCP est qu'on ferme chaque sens de connexion indépendamment. On évite ainsi les pertes.

Closing a connection:

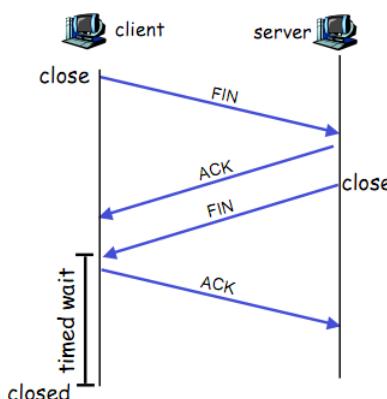
```
client closes socket:  
clientSocket.close();
```

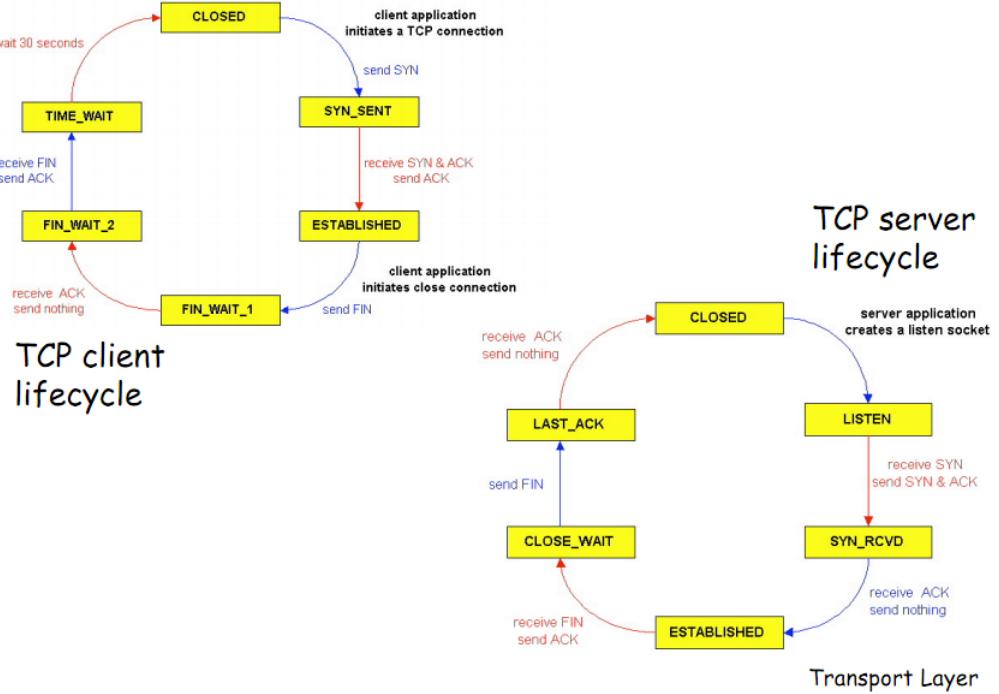
Step 1: client end-system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. When all data sent, closes connection, sends FIN

Note: two directions closed separately

Symmetric/graceful release

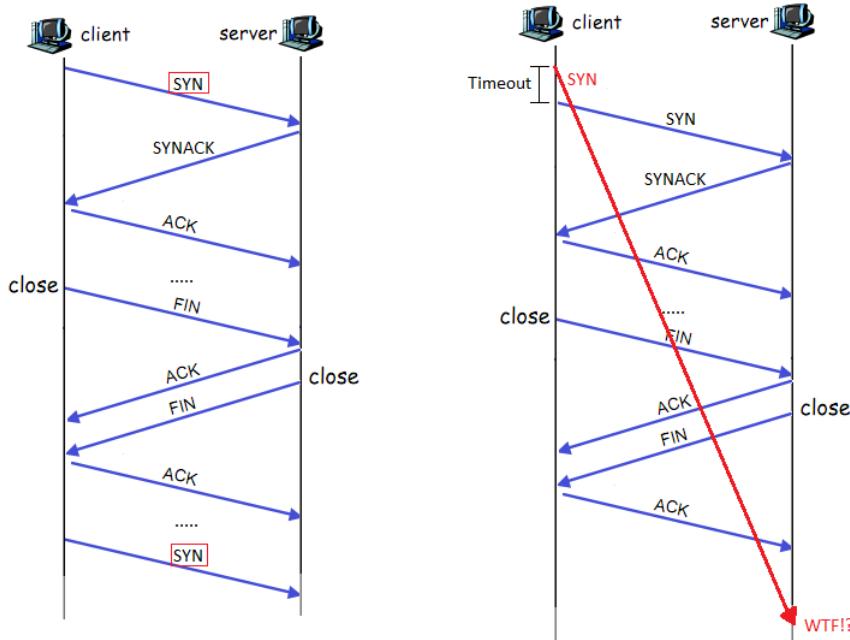




Si tous les ACK se perdent, on peut perdre la fin de la connexion ; pas de solution connue.

Il vaut mieux attribuer des numéros de démarrage différents, sinon il peut y avoir un risque de confusion avec des anciens messages qui circuleraient sur le réseau.

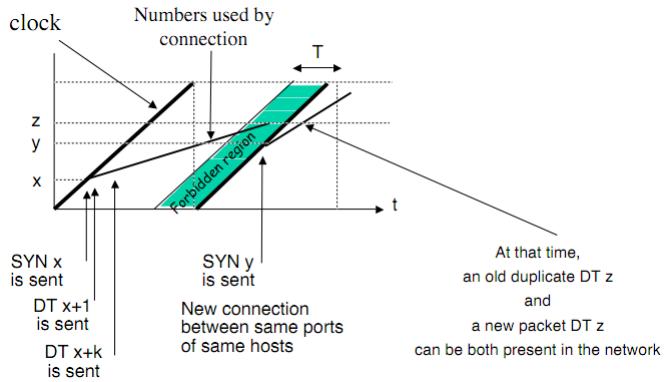
Exemple d'un SYN qui peut être reçu après la fermeture d'une connexion : cela peut conduire à des ambiguïtés dans une autre connexion. Il peut s'agir aussi d'un SYN fantôme, qui a été envoyé au début, mais qui est reçu bien plus tard.



Le risque disparaît si le paquet a un temps de vie court.

En pratique, on introduit un délai entre deux connexions TCP avec les mêmes IP et les mêmes ports, afin d'être sûr que tous les paquets du réseau "meurent".

On utilise alors des horloges, des compteurs de k bits avec k suffisamment grand pour tenir compte du temps de vie du paquet, et en définissant une zone interdite (on s'interdit une entrée par le haut et par le bas).



Il faut être sûr que l'horloge soit suffisamment rapide pour ne pas brider le débit. La valeur de la clock sera le numéro de séquence. T est le temps de vie d'un paquet.

Le rejet des paquets peut venir d'une attaque, qui chercherait notamment à trouver le numéro : l'augmentation de la robustesse se fait au détriment de la sécurité

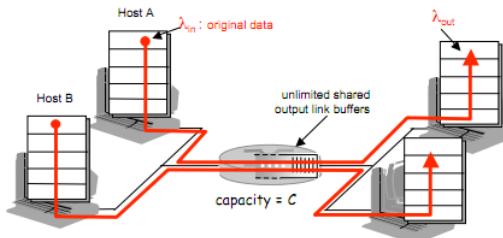
Il peut y avoir aussi des inondations de SYN (SYN flood attack) : cela consiste à envoyer de nombreux SYNs sans continuer le 3-way handshake. Le serveur sera mis à mal avec de nombreuses connections TCP à moitié ouvertes et jamais utilisées.

3.6 Principes de gestion de congestion

La gestion de congestion permet d'éviter de surcharger le réseau (alors que le contrôle de flux évite de surcharger les buffers d'un des protagonistes de la connexion). Des manifestations de congestions sont des pertes de paquets (les buffers des routeurs débordent) et des longs délais (vu que les buffers des routeurs sont remplis)

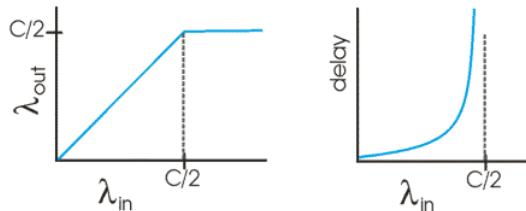
3.6.1 Scénario 1

Supposons que des hôtes envoient des données à un débit de λ_{in} bytes par seconde à travers un lien de capacité C .



Dans ce scénario, on a un routeur dont les buffers sont supposés de taille infinie.

A gauche on a le débit du récepteur (per-connection throughput). On serait tenté d'envoyer à un débit de $\frac{C}{2}$, mais le graphe de droite montre que plus on s'en rapproche plus les délais sont longs.

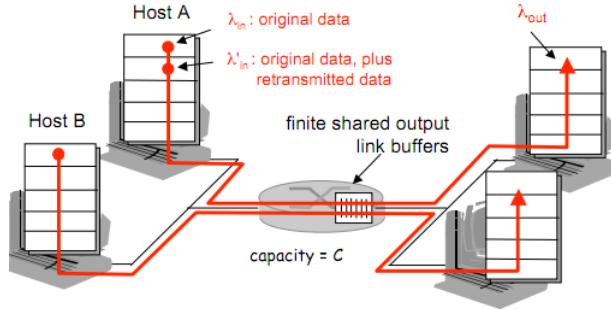


Même dans un scénario idéal, on a une cause de congestion : **les délais sont plus grands à mesure que le débit se rapproche de la capacité du lien.**

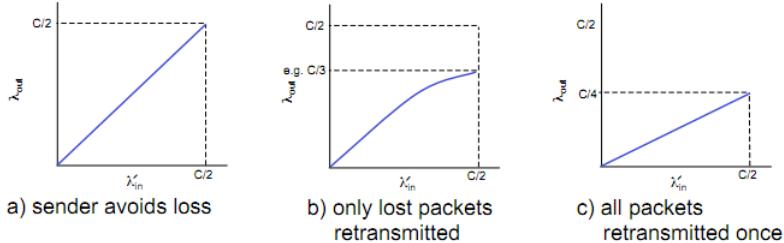
3.6.2 Scénario 2

Supposons que les buffers des routeurs sont finis, donc des paquets seront perdus quand ils seront pleins. Supposons également que les connexions sont fiables.

De ce fait, on va distinguer deux débits : λ_{in} le débit d'envoi des données dans le socket par l'application et λ'_{in} le débit auquel sont envoyés les données ET les retransmissions (offered load).



Plusieurs scénarios sont possibles :

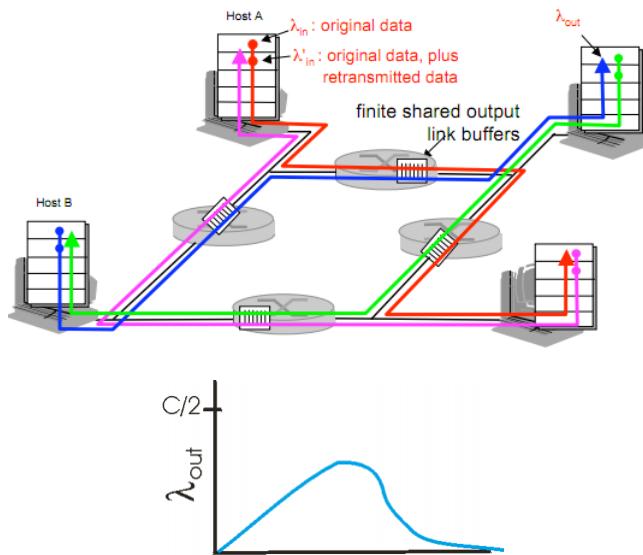


- a) Il n'y a pas de retransmissions
- b) Supposons qu'un tiers des données est retransmis. On a donc, pour une capacité maximale $0.5C$, $0.333C$ données originales et $0.166C$ données retransmises.
On a un autre problème d'un réseau congestionné : **l'émetteur doit retransmettre pour compenser les paquets perdus à cause des dépassements de buffer.**
- c) Supposons que le timeout expire trop tôt, c'est-à-dire que le paquet n'est pas perdu mais fortement retardé. Il y a donc des retransmissions supplémentaires qui ne servent à rien.
Autre conséquence d'un réseau congestionné : **des retransmissions inutiles font que le routeur utilise le lien pour des copies inutiles d'un paquet.**

3.6.3 Scénario 3

Supposons des routeurs aux buffers finis et avec de multiples chemins.

On définit le throughput comme la moyenne de messages correctement délivrés. Le goodput est quant à lui le nombre moyen de bits utiles délivrés sur une période de temps.



Dans le premier routeur du flux rouge, une partie des paquets sera perdue, et sera écrasée par le flux vert dans le second routeur. A partir d'une certaine valeur, on utilise de plus en plus de ressources (λ'_in) pour une efficacité faible (λ_{out}).

On a un nouveau symptôme d'un réseau congestionné : **quand un paquet est perdu sur un chemin, la capacité de transmissions qui a été utilisée par chacun des liens pour amener le paquet à sa perte a été gaspillé.**

3.6.4 Solutions de contrôle de congestion

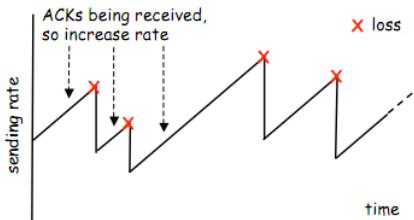
Il y a deux grandes familles de solutions pour le contrôle de congestion :

- end-end congestion control (choix de TCP) : on observe les délais et les pertes, et on règle le débit (sans se fier au réseau) ;
- network-assisted congestion control : on compte sur les routeurs pour qu'ils "marquent" les paquets pour signaler un début de congestion (choke packet).

3.7 Le contrôle de congestion de TCP

TCP va sans cesse aller à la limite de ce qui est possible dans le réseau. Il effectue un feedback implicite : si on reçoit des acquis, on est sûr qu'il n'y a pas de congestion ; s'il y a des pertes (donc pas d'acquittement), on n'est pas sûr que c'est une congestion (peut être un retard, une erreur binaire ou une congestion), mais on va quand même supposer que ça en est.

TCP va utiliser la technique du "probing for bandwidth", soit augmenter le débit en monitorant l'arrivé d'acquis, jusqu'à avoir des pertes. S'il y en a une, il diminue le débit et recommence. La difficulté est de trouver les coefficients de montée et de descente du débit.



On va introduire une autre fenêtre, une fenêtre de congestion : le nombre de byte qu'on peut injecter sera borné par cette fenêtre (comme pour la gestion de flux). La taille de la fenêtre est exprimée en MSS (maximum segment size).

Le débit sera fonction de la taille de la fenêtre :

$$\text{Débit} = \frac{\text{Taille de la fenêtre}}{\text{RTT}}$$

Pour rappel, les pertes sont détectées par les timers ou 3 doublons d'ACK. S'il y a des doublons, le receveur reçoit nécessairement les segments de l'émetteur (ce qui est bon signe). Si un timer expire, soit l'ACK perdu, soit les messages envoyés ne sont jamais arrivés, on est alors dans une situation grave (un timer ne devrait jamais connaître de congestion).

Si un timer expire, on réduit la fenêtre de congestion à 1. Si on a des doublons d'acquis, on diminue la fenêtre de moitié.

Si on reçoit des acquis (non doublon), on augmente la taille de la fenêtre : on l'agrandit d'abord exponentiellement (on la double ; slowstart phase), ensuite linéairement pour éviter les congestions lorsque la taille de la fenêtre dépasse un seuil. Pour augmenter exponentiellement, TCP va ajouter 1 à la fenêtre de congestion à chaque paquet reçu.

On a intérêt à augmenter vite s'il y a de la bande passante dans le système, car le slowstart est vraiment lent (au début).

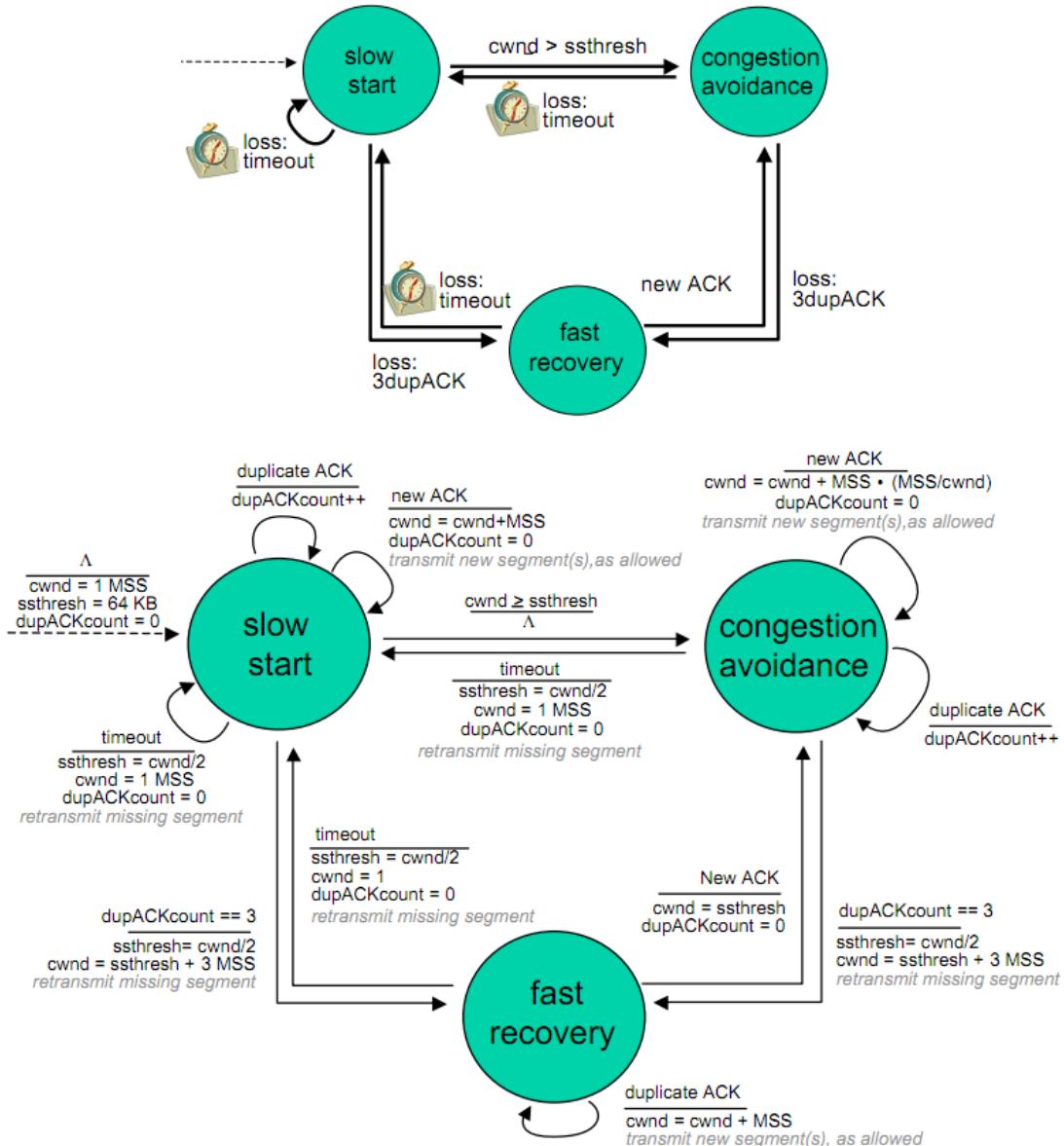
S'il y a congestion, le seuil est défini comme la moitié de la fenêtre avant de la réduire.

Dans la phase de gestion de congestion, lors de la réception d'un ACK, on augmente la taille de la fenêtre par une fraction de MSS, pour qu'au bout d'un RTT on augmente de 1 :

$$\text{Fenêtre de congestion} = \frac{\text{MSS}}{\text{Taille de la fenêtre}}$$

Ce mode de fonctionnement est dit AIMD (Additive Increase Multiplicative Decrease), car on augmente la fenêtre de congestion d'un MSS par RTT lors de la réception d'un ACK, et on la divise par deux lors d'une perte (ne provenant pas d'un timeout).

Le fast recovery est la retransmission du temps perdu, on y reste tant qu'il n'est pas acquit.

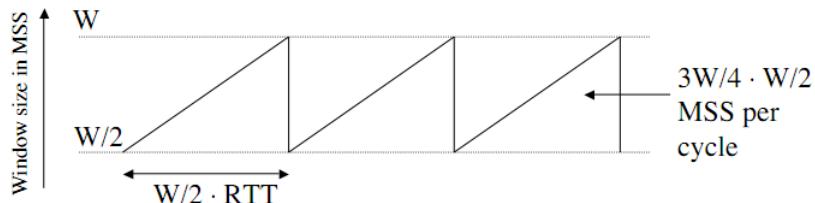


On a $+3$ MSS quand on va de congestion avoidance fast recovery, car on sait qu'on a reçu 3 messages qui suivent ce qu'on a envoyé pour pouvoir continuer d'avancer. Dès que le paquet incriminé est renvoyé et acquit, on effectue un saut (car les messages qui suivent sont consommés), on peut alors diminuer la taille de la fenêtre. C'est un phénomène transitoire.

3.7.1 Efficacité de TCP

Si on ignore le slow start, soit W la taille de la fenêtre lorsqu'une perte se produit. A ce moment, le goodput est de $\frac{W}{RTT}$. Juste après la perte, la taille de la fenêtre est divisée par deux ($\frac{W}{2}$), le goodput aussi ($\frac{0.5W}{RTT}$).

Le goodput moyen est donc de $\frac{0.75W}{RTT}$, la quantité de segments envoyés avant une perte est $\frac{3W}{4} \cdot \frac{W}{2}$.



On a p le taux de perte ; le nombre de MSS par cycle est

$$\frac{3W}{4} \frac{W}{2} = \frac{3W^2}{8} = \frac{1}{p}$$

$$\Leftrightarrow W = \sqrt{\frac{8}{3p}}$$

Le goodput en MSS est la taille de la fenêtre divisée par RTT :

$$\frac{\text{Fenêtre}}{\text{RTT}} = \frac{3W}{4RTT} = \sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}}$$

Le goodput moyen, en bps, est donc

$$\sqrt{\frac{3}{2}} \frac{MSS}{RTT\sqrt{p}}$$

A cause de l'algorithme de congestion, si le taux de perte augmente, le débit diminue.

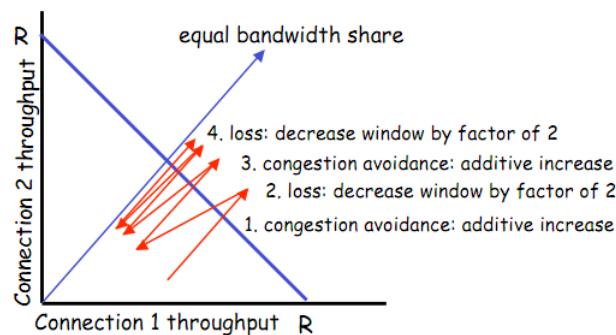
Ce n'est pas valable s'il y a beaucoup de perte, car on fait l'hypothèse qu'on perd un paquet par cycle. De plus, le timer peut faire quitter le steady state, ce qui complique la formule.

La taille du MSS est discutée lors de l'établissement de la connexion (le minimum de ce que les entités peuvent supporter). Le protocole impose une taille minimale.

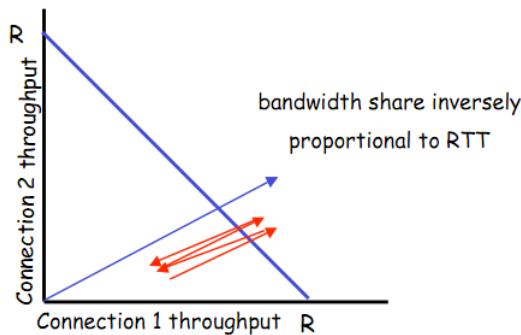
TCP n'aura pas de bonnes performances dans les réseaux très haut débit sur des longues distances. Il faudra des fenêtres très grandes, mais vu que l'augmentation est linéaire (car TCP ne connaît pas l'environnement), il faudra du temps pour profiter de toute la bande passante. De plus, il faudra que p soit très petit pour bénéficier de toute la vitesse.

3.7.2 Équité de TCP

TCP est équitable si les RTT sont les mêmes.



Si les RTT sont différents, celui qui aura le plus grand sera lésé, ce qui est normal vu la formule ; le débit est inversement proportionnel au RTT.



Ce n'est pas aberrant de donner plus de débit à une connexion qui va plus loin, car on doit consommer plus de ressources sur plus de liens.

Si un flux UDP et un flux TCP sont en compétition dans un buffer, UDP n'adaptera pas son débit tandis que TCP le fera. Ils vont tous les deux subir des pertes, mais TCP va vite lâcher car UDP est capable de tout prendre ; TCP va constamment diviser son débit par 2. Tant que les connexions TCP sont majoritaires, pas de problèmes

On peut multiplier les connexions TCP pour chaque fois augmenter le débit

→ l'équité bonne ou pas bonne selon les points de vue

Chapitre 4

La couche réseau

4.1 Introduction

C'est une couche se situant aux extrémités dans chaque hôte, et que les dispositifs intermédiaires sont capables de comprendre.

Du côté de l'expéditeur, cette couche encapsule les segments (venant de la couche de transport) dans des datagrammes. Du côté du receveur, ces datagrammes sont délivrés à la couche de transport.

Le forwarding, au niveau d'un routeur, est le déplacement d'un paquet d'une entrée à la sortie appropriée, via une table d'acheminement, qui se construit par un protocole de routage.

Le routage, au niveau d'un routeur, permet de découvrir la topologie du système et de déterminer la route que le paquet doit suivre. La décision à prendre doit être cohérente par rapport à tout le réseau, pour éviter les boucles ou les conflits. C'est le routage qui va générer la table de forwarding.

Dans certains réseaux, il y a un établissement d'une connexion (pas pour Internet), c'est-à-dire que tous les routeurs vont établir un chemin dans le réseau.

Cette couche réseau fait le lien entre deux machines, alors que la couche transport connecte des processus entre eux via des sockets.

Plusieurs services peuvent être proposés par la couche :

- Les datagrammes individuels garantissent la livraison ...
- ...et la livraison avec moins de 40 ms de délai
- Les flux de datagrammes garantissent la livraison dans l'ordre,
- un minimum de bande passante, et
- des restrictions sur les changements entre deux paquets.

La couche réseau d'Internet, IP, propose un seul service, celui du best-effort : on fait de son mieux pour acheminer le paquet à destination. Il en existe bien d'autres (CBR, ABR) qui proposent plus de services.

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

4.2 Circuits virtuels et réseaux de datagrammes

Comme la couche de transport avec TCP et UDP, la couche réseau propose un service avec ou sans connexion. Un réseau de datagrammes est un service sans connexion, tandis qu'un réseau virtuel est un service avec connexion.

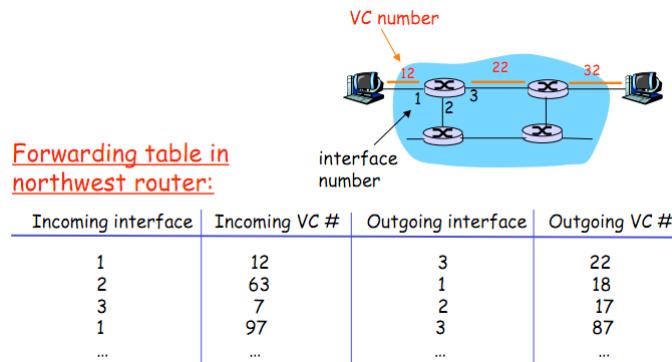
C'est similaire aux services de la couche de transport, mais

- le service se fait d'hôte à hôte,
- le réseau est d'un type ou de l'autre (pas les deux), et
- l'implémentation se fait dans le cœur du réseau et dans les systèmes d'extrémités (on n'a pas de couche transport dans les routeurs).

4.2.1 Circuits virtuels

Chemin tracé dans le réseau de façon à avoir certaines performances et la possibilité d'actions sur le réseau durant l'acheminement. On a un état par destination. Ces chemins se comportent comme des circuits téléphoniques.

On attribue un numéro à chaque lien du chemin à parcourir (VC number), et qui peut changer en cours de route (s'il y a une connexion qui porte le même numéro dans le même noeud). Les routeurs maintiennent l'information de l'état de connexion.

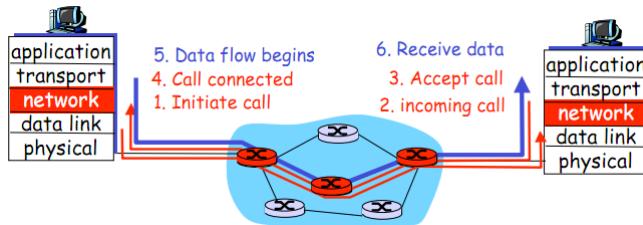


Dans le header du datagramme, on va préciser le VC number du lien à utiliser. Il sera remplacé à chaque passage dans un noeud, en fonction de la table d'acheminement.

Le numéro de paquet change car

- c'est plus simple si on permet plusieurs nombres ; si le nombre était unique, les routeurs devraient communiquer entre eux pour le déterminer, ce qui ralentit et complexifie le réseau ;
- cela permet de garder réduit le champ VC number.

Si un lien est cassé, il faut rétablir un circuit, les paquets n'arriveront pas destination.



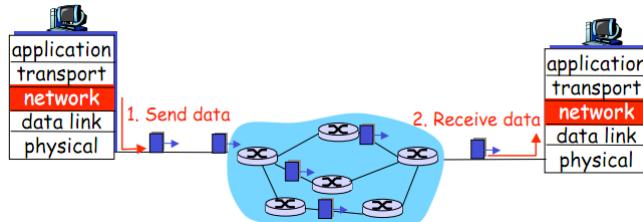
Le transfert se fait donc en 3 étapes :

1. VC setup : la couche de transport contacte le réseau en spécifiant l'adresse du récepteur, et attend qu'il crée le circuit virtuel. La couche va déterminer le chemin, les VC numbers et va ajouter dans la table de forwarding de chaque routeur du chemin une entrée.
2. transfert de données.
3. VC teardown : on informe le réseau qu'on veut terminer le circuit virtuel. Les tables de chaque routeur sont mises à jour.

Les protocoles de signalisation sont utilisés pour installer, maintenir et démonter un circuit virtuel. C'est aussi utilisé dans d'autres réseaux (ATM, frame-relay, X.25), mais n'est plus utilisé dans l'Internet d'aujourd'hui.

4.2.2 Réseau de datagrammes

Il n'y a aucun établissement de connexion, chaque noeud répartit les paquets entre les lignes. Chacun des paquets est autoportant, ils doivent avoir l'adresse de destination.



Si un lien est cassé, on peut le rediriger pour quand même l'acheminer.

La table d'acheminement peut être très grande (une entrée par adresse) ; pour économiser, on peut spécifier des ranges de valeurs pour les acheminements, en spécifiant des préfixes.

Si plusieurs solutions dans le forwarding conviennent, on va choisir celle qui a le plus long préfixe en commun.

4.2.3 Comparaison des deux modèles

Pour les datagrammes (utilisés pour Internet) :

1. échange de données entre des ordinateurs. C'est un service "élastique", il n'y a pas de requis au niveau du temps
2. les systèmes périphériques sont "intelligents" (des ordinateurs).

Ils peuvent s'adapter, appliquer un contrôle et récupérer d'erreur. Ainsi, le réseau est simple à l'intérieur, et complexe à la périphérie.

3. il y a plusieurs types de liens, avec des caractéristiques différentes, ce qui rend l'uniformité du service difficile.

Pour les réseaux virtuels (ATM) :

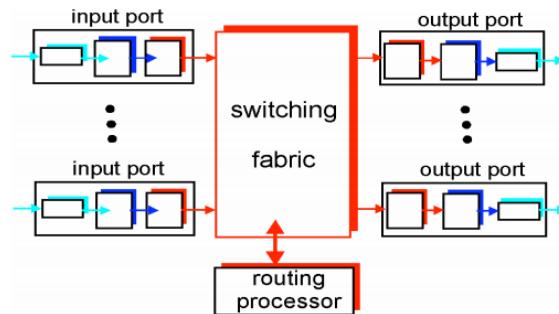
1. évolution de la téléphonie
2. adapté aux conversations humaines, avec des timing stricts et des garanties de fiabilité. C'est nécessaire pour les services qui sont garantis.
3. les systèmes d'extrémité sont simples (comme des téléphones), la complexité est à l'intérieur du réseau.

Ce sont des modèles duals.

4.3 Routeurs

Le rôle du routeur est d'acheminer les paquets d'une de ses entrées à la bonne sortie, en fonction de sa table d'acheminement.

Le processeur est généralement utilisé pour générer les structures de données (calcul de routes, recherche du chemin le plus court, etc) et acheminer des datagrammes d'un lien d'entrée vers un lien de sortie.

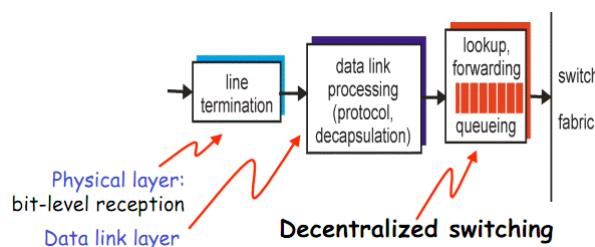


Un routeur est composé de 4 éléments :

1. des ports d'entrée, chargés des fonctionnalités physiques et de celles de la couche lien ;
2. un switching fabric, qui connecte les ports d'entrée et de sortie (sorte de réseau dans le réseau)
3. des ports de sortie
4. un processeur de routage, qui exécute les protocoles de routage. Cela n'a rien à voir avec le traitement en temps réel que l'on applique à chaque paquet reçu (qui se fait dans le switching fabric ou dans les ports d'entrée/sortie).

4.3.1 Ports d'entrée

Lieu où s'appliquent les dernières couches. On doit être capable de décoder et reconnaître un train binaire, qui peut être propre à un lien. Chaque paquet est encapsulé dans une trame particulière, qui dépend des fonctionnalités voulues (récupération d'erreur, etc.).



La partie datalink analyse, jette le paquet éventuellement si abîmé, et transmet dans un buffer d'entrée où on va retrouver le paquet ip tel quel.

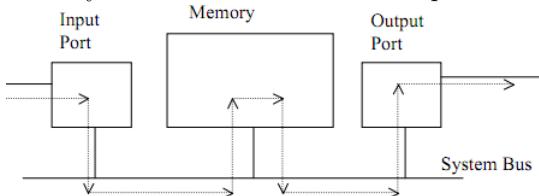
La fonctionnalité de look-up (table de forwarding) se trouve dans chaque interface, la table n'est pas stockée dans une autre mémoire, ainsi on peut fonctionner à la vitesse de la ligne.

Il y a une mise en attente si les datagrammes arrivent plus vite que le rythme de forwarding.

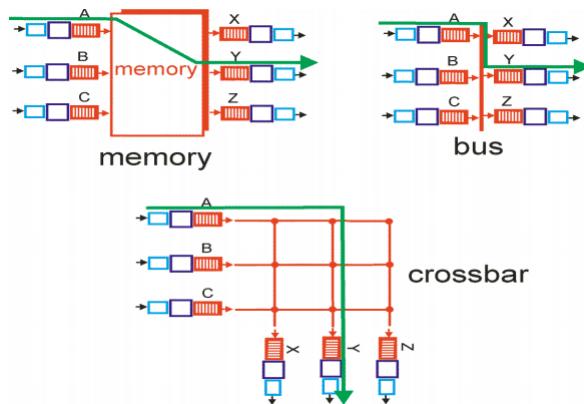
4.3.2 Switching fabrics

Trois types d'implémentation :

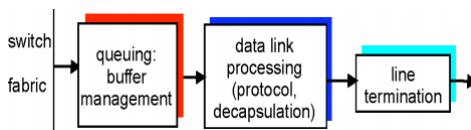
- avec de la mémoire : stockage du paquet en mémoire, basé sur l'architecture de von Neuman.
- Simple à mettre en place (n'importe quel ordinateur avec deux interfaces), mais pas très performant (le bus étouffe les performances et il y a un double accès mémoire pour chaque datagramme) ;



- avec un bus : un seul passage sur un bus, sur lequel on estampille le paquet pour le diriger vers la bonne sortie. On est toujours bloqué par le débit brut du bus, de plus l'accès au bus est séquentiel ;
- avec matrice de commutation : introduction de parallélisme, avec des interrupteurs que l'on peut activer ou désactiver. Si on veut copier le paquet, il suffit d'activer plusieurs commutateurs sur une même ligne. Il y a un minimum d'ordonnancement à avoir, si jamais plusieurs paquets sont redirigés vers la même sortie. De plus, il faut activer les commutateurs. L'ordonnanceur doit être un minimum intelligent, car les paquets ont des tailles variables, ce qui fait que les commutateurs doivent être ouverts/fermés plus ou moins longtemps.



4.3.3 Ports de sortie



Fonctions symétriques à celles des entrées.

On pourrait implémenter plusieurs files d'attente, pour chaque type de paquet, ou bien une politique d'ordonnancement différente de FIFO.

Lorsqu'on a décidé du paquet à envoyer, on l'encapsule (dans le datalink), puis on le binarise.

4.3.4 Queuing à l'entrée et à la sortie

Des buffers sont présents aux entrées et aux sorties d'un routeur. C'est eux qui jettent les paquets quand ils sont pleins.

La taille des buffers est généralement définie pour absorber au mieux un RTT de flux.

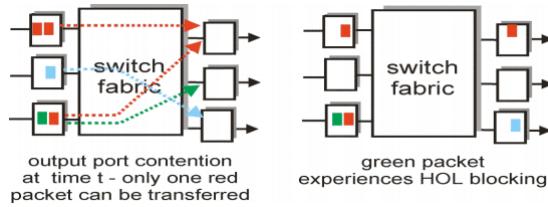
On prend en général, pour N flux TCP, un buffer de taille égale

$$\frac{RTT.C}{\sqrt{N}}$$

Prendre simplement $RTT.C$ est surestimé, car plus il y a de flux TCP plus le signal sera lisse (grâce notamment au contrôle de congestion).

On peut en avoir dans les ports de sortie comme dans les ports d'entrée, dans le cas où le switch fabric est trop lent que pour pouvoir absorber tout le flux.

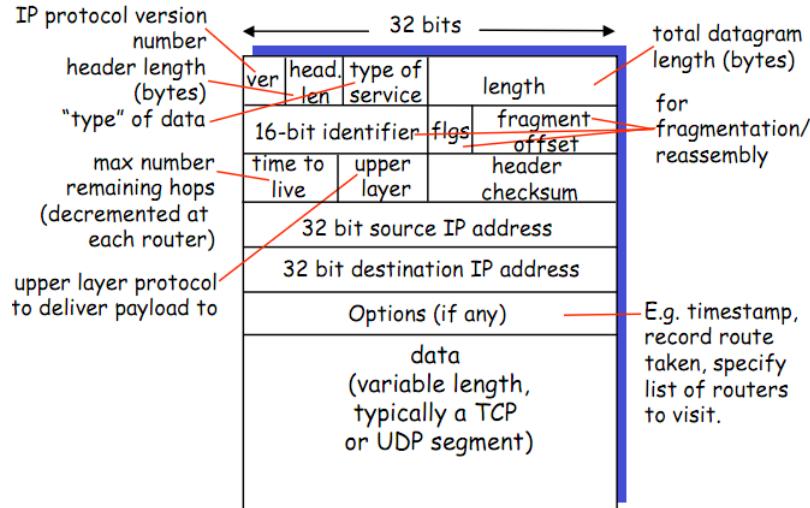
HOL (head-of-the-line) blocking : si un paquet en tête de file est bloqué, tous ceux avant le sont aussi. Il faudrait aller plus loin dans l'analyse des paquets du buffer, mais cela compliquerait l'ordonnanceur.



4.4 Protocole IP

Le checksum ne s'occupe que de l'entête, le checksum du body étant géré par l'entête TCP ou UDP.

4.4.1 Format d'un datagramme IP



Codé sur 32 bits. Il y a un header de 20 bytes pour TCP et un autre de 20 bytes pour IP.

Fragmentation et réassemblage

Le MTU (Maximum Transmission Unit) est la taille de paquet maximum qu'une trame de la couche lien peut transporter (ex d'Ethernet : 1500 bytes). Vu que sur un trajet d'un paquet il peut y avoir plusieurs types de liens et de protocole, le MTU varie.

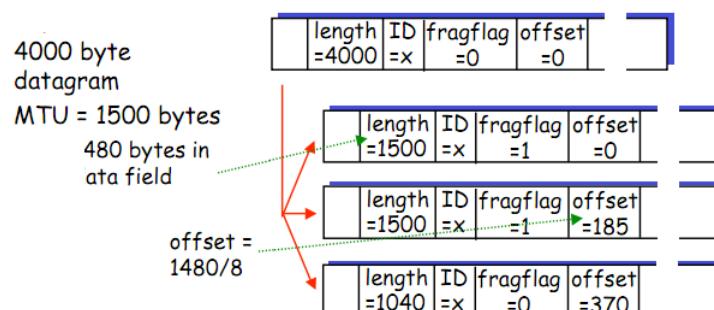
La fragmentation consiste à découper un "gros" paquet en plus petits blocs, tous autoportant. C'est le système à la fin du trajet qui s'occupera de reformer le paquet initial. On inclut dans le header des informations permettant d'identifier et de rassembler dans l'ordre les fragments. Cette segmentation est transparente pour la couche de transport.

Un fragment peut être refragmenté ; la difficulté est de rassembler dans l'ordre des paquets qui ont été fragmentés plusieurs fois.

Le fragflag est à 0 si le paquet n'est pas fragmenté ou s'il est le dernier fragment. offset permet de placer le fragment au bon endroit, il lève aussi l'ambiguïté de fragflag (si fragflag = 0 et offset = 0, ce n'est pas un fragment, sinon c'en est un et c'est le dernier).

Chaque morceau possède un ID pour pouvoir grouper les segments. length compte la longueur du message ET des entêtes.

La quantité de données dans les fragments doit être un multiple de 8 bytes (sauf le dernier fragment). L'offset qui est spécifié est ainsi en unité de 8 bytes.



On présume qu'on ne sera jamais contraint à fragmenter en paquets plus petits que 8 bytes.

Lorsqu'on fragmente un fragment, il n'y en aura qu'un seul qui aura un 0 car on reprend le fragflag.

On aimerait éviter de fragmenter pour éviter les surcharges. C'est aussi pour éviter les surcharges que les routeurs ne réassemblent pas les paquets. Le problème est de connaître le MTU minimum du chemin

On utilise pour cela des paquets ICMP, avec un flag indiquant de ne pas fragmenter. Les routeurs doivent pouvoir envoyer des messages ICMP (on pourrait ne pas vouloir le faire pour éviter des dénis de service (ou bien on place une borne)).

Si la source reçoit un message d'erreur ICMP, alors elle réessaiera avec un paquet plus petit que le MTU indiqué dans le paquet ICMP de retour.

Le problème est qu'il faut que les routeurs retournent des messages ICMP. De plus, la congestion pourrait jeter les messages ICMP, et la route pourrait changer, ce qui engendre de toute façon de la fragmentation.

Problèmes de la fragmentation :

- c'est une charge en plus pour les routeurs
- des attaques DOS sont possibles, par exemple en envoyant des fragments sans commencer à 0, ou des fragments qui se chevauchent.

4.4.2 Adressage IPV4

La plupart du temps, une adresse IP (sur 32 bits) identifie une interface physique (et non pas un système) : soit un hôte, soit l'interface d'un routeur. Toutes les IP dans une même zone vont avoir le même préfixe, pour faciliter le routage ; le protocole IP permet d'amener le paquet au dernier routeur.

Sous-réseaux

Dans une adresse IP, on distingue la partie subnet (les bits de grand ordre) et la partie hôte (les bits de petit ordre).

Un subnet est l'ensemble des interfaces qui ont la même partie subnet de leur IP. Ces interfaces peuvent s'atteindre physiquement sans l'intervention d'un routeur.

Pour déterminer les subnets, il suffit de détacher chaque interface de son hôte ou de son routeur, et de créer des réseaux isolés.

CIDR

CIDR (Classless InterDomain Routing) est le format d'ip a.b.c.d/x, où x est le nombre de bits de subnet dans l'adresse.



DHCP

Pour récupérer une adresse IP, on peut la hardcoder dans un fichier système, ou bien en adresser une dynamiquement depuis un serveur. DHCP va fournir cette adresse, en plus de démarrer certains services de bases.

Généralement, les routeurs possèdent un serveur DHCP. Le modèle impose qu'il y ait un serveur DHCP dans tous les subnets.

4 messages :

1. message de découverte : le client envoie une requête de découverte en broadcast (tous les serveurs DHCP le recevront). La requête est envoyée à partir d'une adresse non routable (0.0.0.0).
2. message de réponse : le serveur DHCP envoie une proposition d'adresse, que l'on peut garder pendant une certaine durée (il faudra la renouveler).
3. message de request : choix parmi les propositions d'ip.
4. message d'ack

Les messages sont tous en broadcast, on sait constamment qui il y a sur le réseau. Il y a de plus une détection rapide si jamais une IP est utilisée plus d'une fois.

Le serveur DHCP ne retourne pas qu'une adresse IP : il retourne l'adresse du routeur, le nom et l'IP du serveur DNS, le masque du réseau.

Ce sont les ISP qui fournissent des portions de leur espace d'adresses. Ces ISP obtiennent ces adresses auprès de l'ICANN (Internet Corporation for Assigned Names and Numbers), qui alloue des adresses, gère les DNS et assigne des noms de domaine.

NAT - Network Address Translation

Il faut s'arranger pour que des adresses non routables restent locales, cachées (très probablement déjà utilisées). L'avantage est qu'on peut changer d'ISP sans changer le subnet.

Le routeur est à la fois client et serveur DHCP : client pour récupérer une adresse ip, serveur pour le subnet.

La motivation est donc d'avoir un réseau local avec juste une seule adresse IP, pour tous les périphériques. Ainsi, un routeur NAT doit

- pour les datagrammes sortant, remplacer l'adresse IP et le port source par l'adresse IP NAT et un nouveau port.
Ainsi, les réponses reviendront vers le serveur NAT.
- se rappeler, dans une table de traduction NAT, du mapping de chaque adresse source et port vers l'adresse NAT et du nouveau port.
- pour les datagrammes entrant, remplacer l'adresse NAT et le nouveau port par ce qui est stocké dans la table NAT.

C'est une sorte de firewall : si un paquet veut atteindre une machine qui n'a pas d'entrée NAT, il restera bloqué au routeur. Il y a une ouverture lors de l'envoi d'un paquet du LAN.

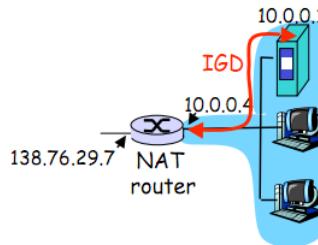
Problèmes si le paquet est crypté.

Le NAT est controversé, car les routeurs ne devraient entrer en jeu que jusqu'à la couche 3 (en modifiant les ports, il y a un accès la couche de transport). De plus, cela viole la condition end-to-end ; les NAT doivent être pris en compte dans le design d'une application réseau. Ces raccourcis d'adresse devraient prendre fin avec IPv6, vu l'espace d'adresse disponible.

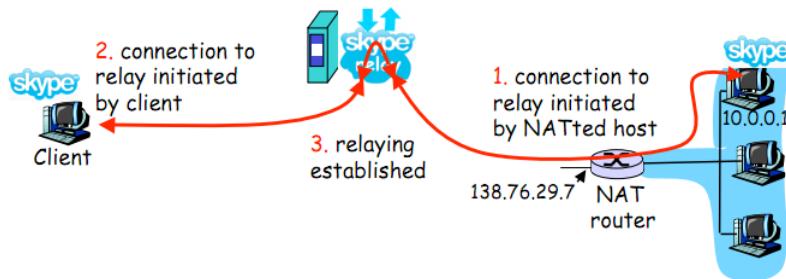
On rencontre cependant un problème lorsqu'un client veut traverser le NAT sans qu'un hôte derrière ne l'ait sollicité (ex d'un serveur).

Solutions :

1. configurer statiquement la table NAT
2. Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Cela permet aux hôtes derrière le NAT d'avoir des IP publiques et d'ajouter et de supprimer des mapping de port. C'est une configuration automatique de la table NAT.



3. utiliser un relai, exemple de skype : le routeur acceptera ce qui vient des serveurs skype, mais pas directement de l'autre interlocuteur.



4.4.3 ICMP

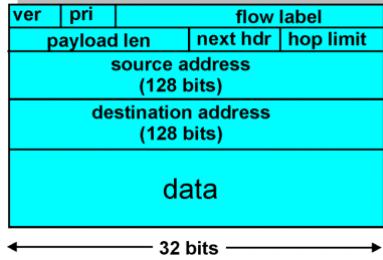
ICMP (Internet Control Message Protocole) est un protocole reportant les erreurs à l'aide de paquets ICMP. Il est utilisé par les hôtes et les routeurs pour communiquer des informations réseau.

Il envoie des informations sur ce qui a généré l'erreur. Il peut être utile pour autre chose (ex : ping, ou MTU pour une fragmentation). Traceroute les utilise pour déterminer une route, car ils contiennent des informations sur le routeur.

4.4.4 IPv6

La motivation était le fait que l'espace des adresses de 32 bits sera bientôt complètement alloué. De plus, le format de header aide les traitements et le forwarding et facilite le QoS.

Le header d'un datagramme IPv6 est de 40 bytes, et ne permet pas la fragmentation.

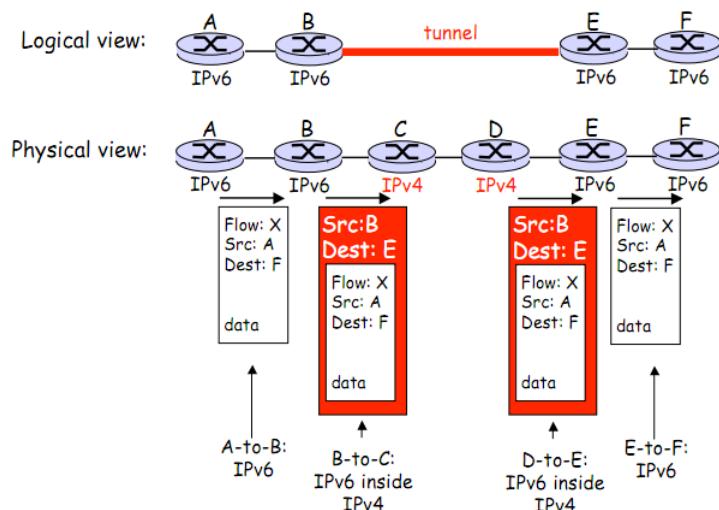


On trouve un champ qui détermine la priorité d'un datagramme dans un flux. Le flow label permet d'identifier les datagrammes d'un même flux, même si le concept de flux n'est pas bien défini. Le champ next header identifie le protocole à qui livrer le contenu, ou d'autres options.

Le checksum a été complètement retiré, ce qui réduit le temps de traitement ; la responsabilité est reportée à la couche de transport et à la couche lien. On définit également ICMPv6.

Pour le passage d'IPv4 à IPv6, tous les routeurs ne sauraient pas être mis à jour en même temps, et aucun "flag day" n'a été défini.

Pour remédier à ce problème, on utilise le tunneling : un datagramme IPv6 est incorporé dans un datagramme IPv4 pour qu'il puisse traverser les routeurs IPv4.



4.5 Algorithmes de routage

Les algorithmes de routage permettent donc de générer la table d'acheminement des routeurs. Le système est trop grand pour être fédéré globalement, c'est à un niveau local que le routage est calculé.

Chaque hôte est attaché directement à un routeur, que l'on appelle le routeur par défaut, ou le first-hop router. Le but est de trouver un chemin de coût minimal entre le routeur par défaut d'un émetteur et celui d'un récepteur.

Le routage consiste à générer un graphe de la topologie du réseau et à chercher des chemins à l'intérieur. A chaque lien, on assigne un coût qu'il faudra minimiser. Le fait de changer le coût des liens modifie à coup sûr les chemins.

4.5.1 Métriques pour le coût

Les coûts peuvent être définis de manière à optimiser le réseau. Il est pour cela nécessaire de connaître la matrice de trafic (TM) : pour chaque paire de noeuds (i, j) , $TM(i, j)$ est la quantité de trafic entrant par le noeud i et sortant par le noeud j .

Il y a plusieurs manières de définir les métriques.

Minimiser la charge moyenne des liens Pour assurer un minimum de hop dans le routage, on peut mettre le coût de tous les liens à 1. Cela minimisera aussi la charge des liens et le processing dans les noeuds, mais ça ne garantit pas un délai minimum ni une congestion minimale.

En effet, on a

$$\text{Score} = \text{charge moyenne des liens} = \frac{\sum_{i \in \text{links}} \text{charge}_i}{N}$$

Minimiser la charge moyenne équivaut à minimiser la somme de toutes les charges, on peut donc enlever le dénominateur.

De plus, router un nouveau flux de débit R pour un chemin P augmentera le score :

$$\text{Augmentation de la charge} = \sum_{i \in P} R = R \times \text{nombre de hops dans } P$$

Ainsi, minimiser la charge d'un lien équivaut à minimiser le nombre de hop, et pour cela il suffit d'avoir la métrique statique 1 pour chaque lien.

Minimiser l'utilisation moyenne d'un lien

$$\text{Score} = \text{Utilisation moyenne du lien} = \frac{\sum_{i \in \text{links}} \text{utilisation}_i}{N} = \frac{\sum_{i \in \text{links}} \frac{\text{charge}_i}{\text{capacité}_i}}{N}$$

C'est équivalent à minimiser la somme de toutes les utilisations des liens. Router un nouveau flux de débit R selon un chemin P va augmenter le score :

$$\text{Augmentation de l'utilisation} = \sum_{i \in P} \frac{R}{C_i} = R \times \sum_{i \in P} \frac{1}{C_i}$$

Cela équivaut donc à chercher le chemin qui minimise $\sum_{i \in P} \frac{1}{C_i}$. Pour y parvenir, on peut assigner la métrique $\frac{1}{C_i}$, la capacité inverse. Cela a un sens, car on somme des temps de transmission.

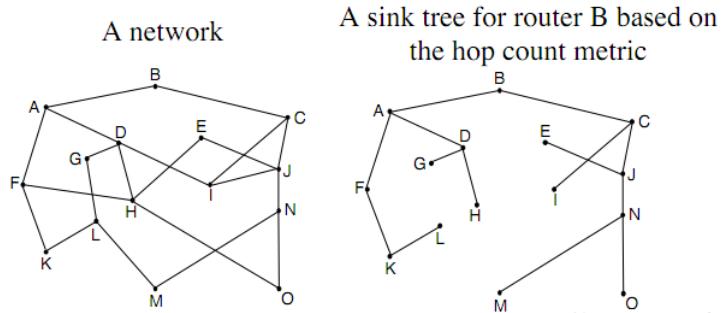
Autres métriques

- métrique sur le délai d'un lien : on minimise le délai, mais cela implique plusieurs composantes : le temps de propagation, le temps de transmission (taille du paquet / capacité du lien) et le temps d'attente (qui varie et dépend de la charge, difficile à prendre en compte)
- coût administratif : chaque métrique est calculée pour optimiser un certain score, par exemple pour mieux balancer la charge, mais ça devient dépendant de la matrice de trafic
- n'importe quelle quantité sommable, c'est à dire que le coût d'un chemin est la somme des coûts des liens.

4.5.2 Principe d'optimalité

Si un routeur J est dans le chemin optimal pour aller du routeur I à un routeur K , alors le chemin optimal de J à K suit la même route.

En conséquence, l'ensemble des routes optimales est un arbre recouvrant dont la racine est la source. Le spanning tree est individuel et associé à un noeud, il n'est pas généralisable pour tous les autres noeuds. (ex : O → H)



4.5.3 Classification des algorithmes de routage

L'information peut être

- globale : tous les routeurs ont la topologie complète et les informations de coût des liens ; ce sont les algorithmes à état de lien ;
- décentralisée : les routeurs ne connaissent que les voisins physiquement connectés et les coûts pour aller vers ces voisins. Le processus est itératif, il y a un échange d'informations entre les voisins ; ce sont les algorithmes à vecteur de distances.

L'algorithme peut être

- statique : les routes changent doucement au cours du temps
- dynamique : les routes changent plus rapidement ; il y a une mise à jour périodique et une réponse aux changements des coûts des liens.

4.5.4 Etat de lien - link state

Protocole le plus répandu ; le réseau est considéré comme un graphe. Chaque noeud est un routeur, qui connaît bien son voisinage. Il va former un paquet qui l'identifie et qui va décrire l'état de ses liens direct, et va diffuser ce paquet dans la topologie (diffusion en broadcast).

Les paquets contiennent les distances entre les voisins, selon une métrique. Tout le monde possède tous les états des liens, ce qui permet de reconstruire le graphe.

A partir de ce graphe, on cherche le plus court chemin avec l'algorithme de Dijkstra.

Algorithme de Dijkstra

Fréquemment utilisé dans les routeurs. Il va chercher les chemins de moindre coût pour atteindre tous les noeuds. On sait ainsi par quelles interfaces de sortie on doit passer ; tout est stocké dans la table d'acheminement.

On obtient alors un spanning tree, qui permettra de calculer la table d'acheminement, et qui indique pour chaque destination le point de sortie.

Notations :

- $c(x, y)$ est le coût d'un lien du noeud x au noeud y . Vaut ∞ s'ils ne sont pas voisins
- $D(v)$ la valeur courante du coût du chemin de la source vers la destination v
- $p(v)$ le noeud précédent dans le chemin depuis la source jusqu'à v
- N' l'ensemble des noeuds fixés, qui appartiennent au chemin de moindre coût

Si on exécute l'algorithme dans un noeud u :

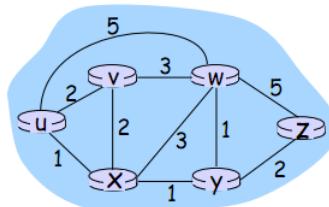
```

1 Initialization:
2   N' = {u}
3   for all nodes v
4     if v adjacent to u
5       then D(v) = c(u,v)
6       else D(v) = ∞
7
8 Loop
9   find w not in N' such that D(w) is a minimum
10  add w to N'
11  update D(v) for all v adjacent to w and not in N' :
12    D(v) = min( D(v), D(w) + c(w,v) )
13  /* new cost to v is either old cost to v or known
14  shortest path cost to w plus cost from w to v */
15 until all nodes in N'

```

Exemple.

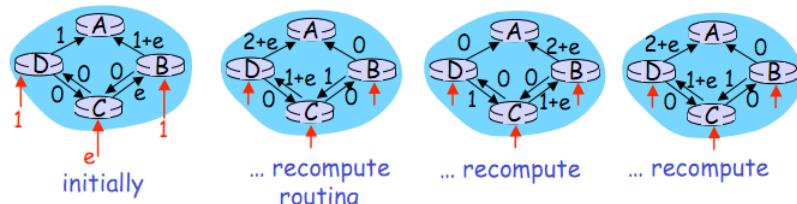
Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u		2,u	5,u	1,u	∞
1	ux		2,u	4,x		2,x
2	uxy		2,u	3,y		4,y
3	uxyv			3,y		4,y
4	uxyvw					4,y
5	uxyvwz					



Naïvement, l'algorithme est quadratique si on parcourt chaque fois tous les noeuds à chaque itération. En maintenant la liste triée, on a une exécution en $n \log n$ (avec une file priorité).

L'algorithme est bon si les métriques sont statiques. Si les métriques changent, c'est-à-dire si le coût d'un lien varie (par exemple le trafic), il faudrait chaque fois tout recalculer vu que les plus courts chemins risquent de changer.

De plus, des oscillations sont possibles lorsque les coûts des liens dépendent du trafic. Par exemple :



Une solution est de s'assurer que les routeurs n'exécutent pas l'algorithme en même temps.

→ avoir des métriques dynamiques peut être dangereux, une modification en temps réelle n'est pas conseillée.

On diffuse globalement son estimation locale.

4.5.5 Vecteurs de distances - distance vector

Historiquement c'est le premier, il n'est plus utilisé sauf dans des petits réseaux Il est basé sur la connaissance des voisins d'un noeud, et non du graphe en entier. On utilise l'équation de Bellman-Ford pour converger vers la bonne valeur.

C'est un algorithme

- itératif, car il continue tant qu'on ne reçoit pas de l'information des voisins

- asynchrone, car il se déclenche lors de la réception d'informations

- distribué, car il reçoit des informations de ses voisins et renvoie ses propres résultats

On définit $d_x(y)$ comme le coût du chemin de moindre coût qui va de x à y . On a alors, si le minimum est appliqué à tous les voisins v de x .

$$d_x(y) = \min_v \{c(x, v) + d_v(y)\}$$

Soient

- $D_x(y)$: le moindre coût estimé de x à y

- $c(x, v)$: pour un noeud x , le coût pour accéder à ses voisins v

Un noeud x maintient deux vecteurs :

- un vecteur de distances, où figurent tous les noeuds du réseau : $\mathbb{D}_x = [D_x(y) : y \in N]$

- les vecteurs de distances de ses voisins : $\mathbb{D}_v = [D_v(y) : y \in N]$

L'idée de base est la suivante :

- chaque noeud envoie périodiquement son vecteur de distances estimé à ses voisins

- quand un noeud x reçoit un nouveau vecteur de distances d'un de ses voisins, il met à jour son propre vecteur en utilisant l'équation de Bellman-Ford :

$$D_x(y) \leftarrow \min_v \{c(x, v) + D_v(y)\} \text{ pour chaque noeud } y \in N$$

- dans des conditions normales, $D_x(y)$ converge vers le moindre coût $d_x(y)$

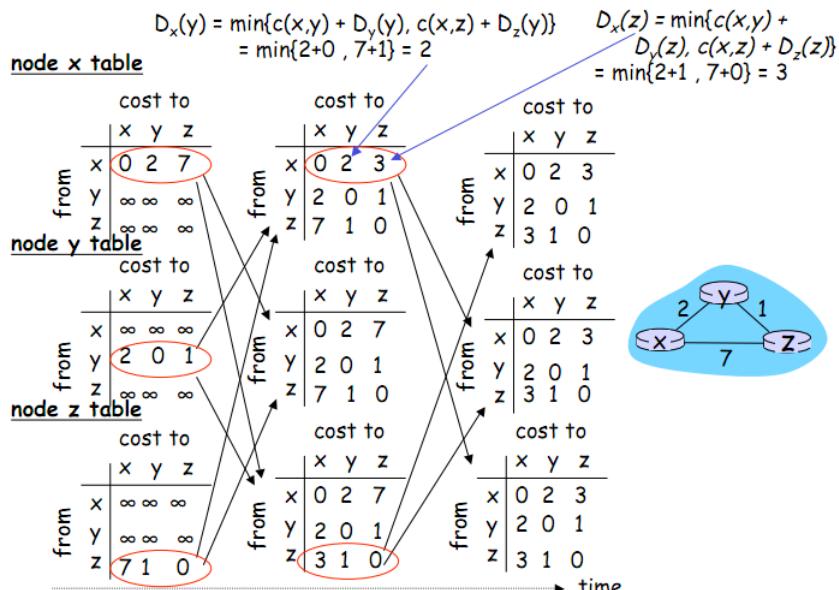
- asynchrone : lors d'un changement de topologie ou d'un coût, il y a un message de mise à jour des vecteurs de distance envoyé aux voisins.

- distribué : chaque noeud ne réagit que quand son vecteur de distance change.

Pour que ça converge, il faut que les coûts ne soient pas négatifs. Le pire serait un cycle négatif, car l'algorithme va boucler vu que le cycle améliore la distance.

On diffuse localement (aux voisins) une connaissance globale (les estimations).

Exemple.

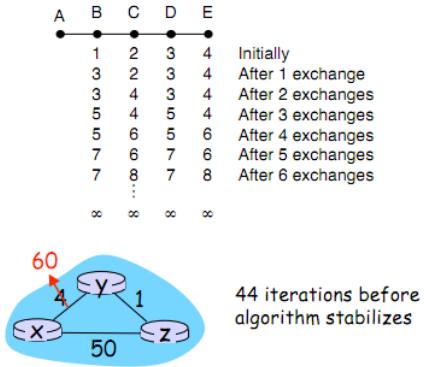


Il y a autant de colonnes que de noeuds dans le réseau, et autant de lignes que de voisins.

L'algorithme se met bien à jour si une métrique diminue, tous les noeuds convergeront rapidement.

Par contre, si une métrique augmente (ou si un noeud tombe en panne), cela va dégénérer. Pour atteindre la connaissance qu'un noeud n'est plus accessible, il faudra du temps. Idem si une métrique augmente : il y a des incrémentations d'une unité jusqu'à ce que ça se stabilise, après de nombreuses itérations et messages).

link A-B is down, $c(B,A) = \infty$



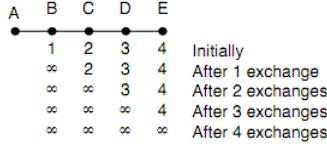
Il y a 44 itérations car :

- y saura que pour accéder à x, il y a 60, mais il peut passer par z qui peut y accéder en 5. Donc $D_y(x) = 6$
- z va détecter le changement et choisira $\min 6 + 1, 50 = 7$
- y va détecter le changement et choisira $\min 7 + 1, 60 = 8$
- ...
- z va choisir $\min 51, 50 = 50$, au lieu de passer par y il utilisera directement son lien de poids 50. Il sera stabilisé
- y détectera le changement et $d_y(x) = 51$

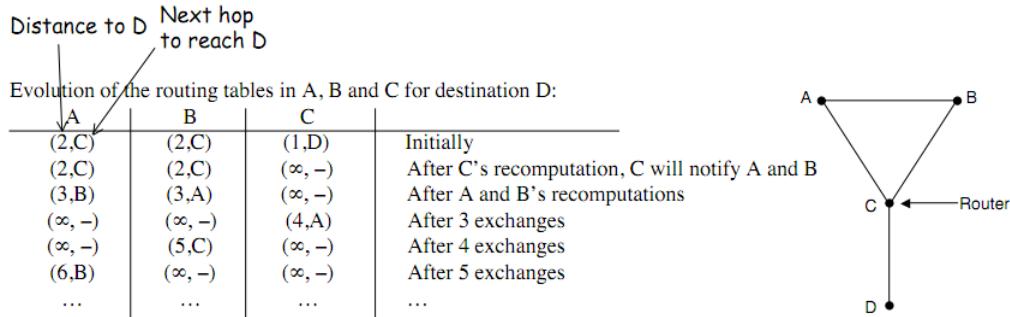
Il y a une boucle de routage entre y et z.

Une correction est possible, le poisoned reverse, appelé aussi le split-horizon : on diffuse des "mensonges" entre les routeurs. Un noeud ne diffuse un mensonge à un noeud que si, pour accéder à un autre noeud, il doit passer par ce noeud qui passe par lui-même pour accéder à l'autre noeud. Pour le savoir, en plus de donner leur vecteur de distances, les noeuds voisins vont indiquer leur next-hop.

Par exemple, C va indiquer à B que sa distance par rapport à A est infinie. Ainsi, B n'ira pas vers A via C.



Cependant cela risque toujours de cycler, par exemple si le lien CD tombe en panne.



C va détecter que le lien est cassé, il passe sa distance à ∞ et notifie A et B. A voit qu'il peut passer par B et B par A pour aller en D, ils mettent à jour leurs distances.

C reçoit les mises à jour et voit qu'il peut passer par A. Il met à jour sa distance et la propage. Dans le même temps, A et B détectent qu'il y a un problème (car pour accéder à D, ils passent par l'autre noeud qui passe par eux-mêmes), le poisoned reverse est activé et ils mentent sur leur distance à D.

B voit qu'il peut passer par C pour aller à D, il met à jour sa distance en passant par C. A voit qu'il peut passer par C, mais que C passe par A, donc il ment encore. C voit qu'en A la distance à D est ∞ de même qu'en B, sa distance devient ∞ aussi.

A voit qu'il peut passer par B, il met à jour sa distance. B voit que la distance de C est ∞ , idem pour A, donc sa distance reste ∞ . C voit qu'il peut passer par B, mais qui lui-même passe par C. Il ne peut passer par A car sa distance est ∞ , du coup sa distance reste ∞ .

4.5.6 Comparaison des algorithmes LS et DV

En complexité de messages :

- LS : avec n noeuds et E liens, $\mathcal{O}(nE)$.

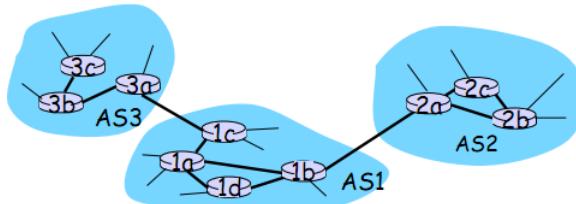
- DV : les échanges se font uniquement entre les voisins, le temps de convergence peut varier très fort.
- En vitesse de convergence
- LS : algorithme en $\mathcal{O}(n \log n)$, avec la nécessité de $\mathcal{O}(nE)$ messages. Il peut y avoir des oscillations
- DV : le temps de convergence varie. Il peut y avoir des boucles dans le routage, tout comme des problèmes de comptage l'infini.
- En robustesse, ce qui se passe lorsqu'un routeur tombe en panne.
- LS : un noeud peut prévenir d'un coût d'un lien incorrect. Chaque noeud calcule sa propre table
- DV : un noeud peut prévenir d'un coût de chemin incorrect. Chaque table d'un noeud est utilisée par les autres, l'erreur se propage à travers le réseau

4.5.7 Routage hiérarchisé

Le routage doit être hiérarchisé, sinon il y aurait des millions d'entrées à stocker dans les tables de routage des modems, et les messages envoyés pour les calculs pollueraient la bande passante.

De plus, chaque administrateur système pourrait vouloir contrôler le routage de son propre réseau

On définit un AS comme un système autonome (ISP, opérateur réseau, etc), un domaine où il y a un administrateur qui décide du protocole de routage pour un ensemble de routeurs.



On distingue le routage intra-domaine (intra-AS) et le routage inter-domaine (inter-AS). Il y aura des routeurs particuliers, à la périphérie d'un AS, qui font le lien avec un AS voisin : ce sont des passerelles (gateway).

Le routage intra-domaine peut être de n'importe quel type, l'administrateur réseau fait comme il lui plaît. Le routage inter-domaine par contre doit être le même, sinon il n'y aurait aucune coordination entre les AS.

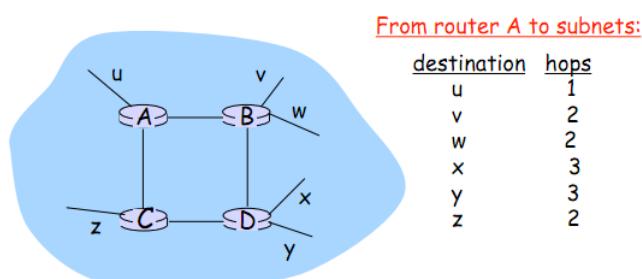
Pour un routage interne à un AS, seul le protocole intra-domaine sera utilisé. Pour un routage externe, il faudra les deux, car il faudra trouver le point de sortie.

4.6 Routage dans l'Internet

Pour le routage intra-AS, il y a RIP, OSPF, IS-IS (intermediate system to intermediate system) et IGRP (interior gateway routing protocol).

4.6.1 RIP

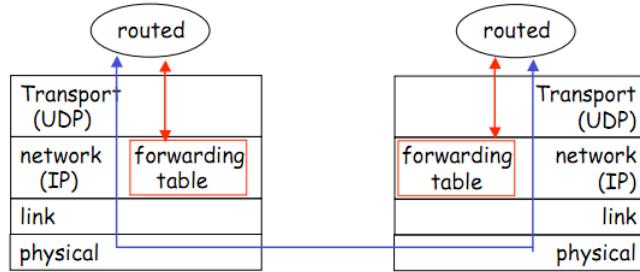
RIP (Routing Information Protocol) est un algorithme de vecteur de distances. Les liens sont toujours unitaires, et on définit qu'il y aura maximum 15 hops, 16 hops étant considérés comme infini. 0 hop signifie que ce n'est pas accessible. S'il n'y a pas de hop à effectuer, au minimum 1 hop enregistré dans la table.



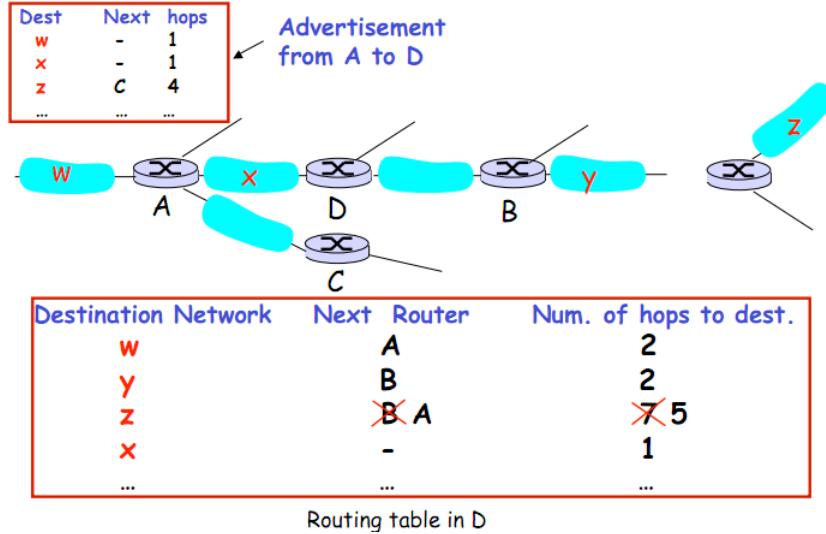
Les vecteurs de distances sont échangés entre les voisins toutes les 30 secondes via des Response Message, appelés aussi advertisement.

Si un lien ou un voisin ne répond pas après 180 secondes, il est considéré comme déconnecté et un advertisement est envoyé. Le poison reverse est utilisé pour éviter des boucles.

Les tables de routage RIP sont gérées par la couche applicative par un processus (un daemon, route-d). Les advertisements sont envoyées dans des paquets UDP.



Une table RIP est une table de routage qui contient les vecteurs de distances et la table d'acheminement.



4.6.2 OSPF

OSPF (Open Shortest Path First) est un algorithme à état de lien public. La topologie du réseau est connue dans tous les noeuds.

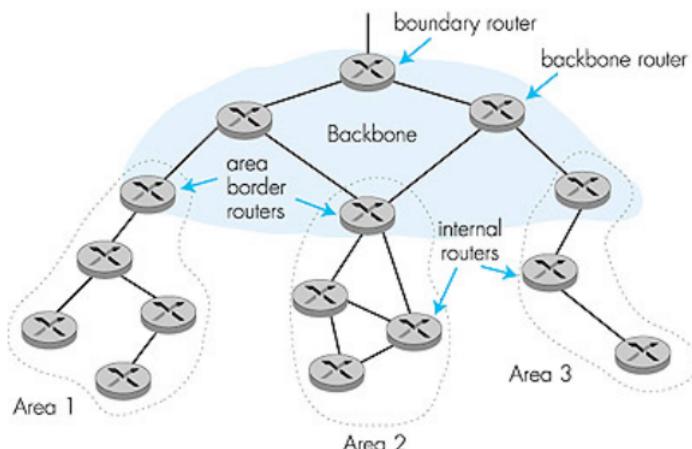
Un advertisement OSPF comporte une entrée pour chaque voisin du routeur qui l'a envoyé. Ils sont disséminés dans l'AS entier via du flooding. Les messages OSPF sont envoyés directement via IP, sans passer par TCP ou UDP.

Avantages par rapport à RIP :

- sécurité : tous les messages OSPF sont authentifiés ;
- il est permis d'avoir plusieurs chemins de moindre coût ;
- pour chaque lien, on peut avoir plusieurs métriques (par exemple on cherche à minimiser la charge ou le délai) ;
- multicast ;
- OSPF hiérarchisé dans des grands domaines.

Les mises à jour se font toutes les 30 minutes, il y a envoi d'un paquet d'état de lien.

OSPF hiérarchisé



On distingue deux niveaux : les local area et le backbone. Chaque message à état de lien ne s'applique que dans les local area ; chaque noeud connaît le plus court chemin vers les réseaux de toutes les local area.

On distingue 3 types de routeur :

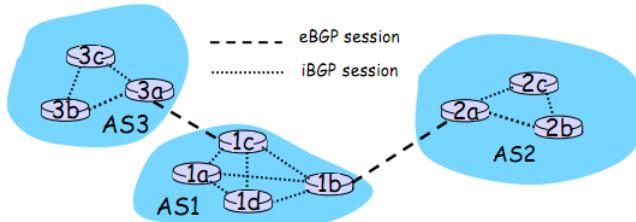
- les routeurs au bord des area : ils synthétisent les distances des réseaux dans leur zone et les affichent aux autres routeurs de bord de zone
- les routeurs backbone, qui lancent OSPF en se limitant au backbone
- les routeurs de bord (boundary routers), qui sont connectés à d'autres AS.

4.6.3 BGP

BGP (Border Gateway Protocol) est le standard pour le routage inter-AS. Il permet de montrer au reste d'Internet que des réseaux existent et fournit un moyen à chaque AS de

- obtenir les informations d'accessibilité des AS voisins
- propager les informations d'accessibilité à tous les routeurs internes de l'AS
- déterminer les bonnes routes en se basant sur ces informations et la politique de l'AS

Les paires de routeurs (BGP peers) échangent des informations à travers des connexions TCP semi-permanentes, ce sont des sessions BGP.

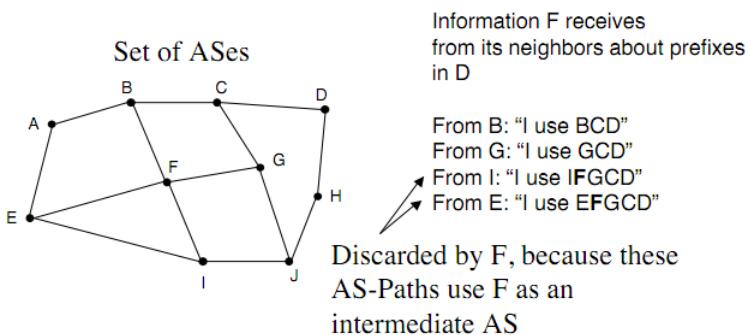


Quand un AS informe un autre de préfixes, il promet qu'il peut router n'importe quel datagramme qui correspond à un de ces préfixes.

Lorsqu'un préfixe est reçu par un routeur de bord à travers une session eBGP, il va distribuer le nouveau préfixe à tous les routeurs de l'AS à travers des sessions iBGP, et il va prévenir les autres AS auxquels il est connecté des nouvelles informations d'accessibilité. Chaque routeur va créer une entrée pour le préfixe dans sa table d'acheminement.

- Lorsqu'un préfixe est envoyé, il est accompagné d'attributs qui forment la route ; il y a deux attributs importants :
- AS-Path : c'est la liste des AS qu'il faut traverser. BGP est un algorithme à chemin de vecteur ; AS-Path est propagé, alors que dans un protocole DV seule la distance est envoyée
 - NEXT-HOP : indique quel routeur interne à l'AS utiliser pour atteindre le prochain AS, le next hop (car il peut y avoir plusieurs chemins)

Un routeur gateway, lorsqu'il reçoit des informations de routage, peut pratiquer une politique qui en accepte ou en décline. Ainsi, quand un AS se voit dans l'AS-Path, il le jette directement pour ne pas créer de boucle. C'est plus puissant que le poisoned reverse et c'est possible grâce à cet AS-Path.



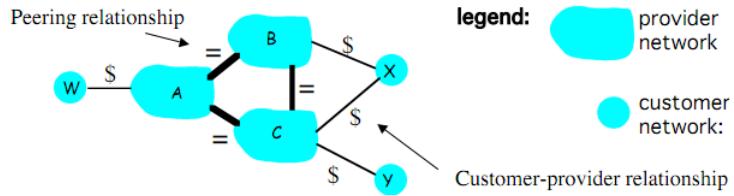
Un routeur peut apprendre qu'il y a plus d'une route pour un préfixe, il faut donc en choisir une. Il y a des règles d'élimination :

1. une politique de l'AS, qui mettrait un attribut en avant
2. le plus court AS-Path
3. hot potato routing : on choisit le chemin où le routeur de next-hop (pour passer au prochain AS) est le plus proche
4. critères additionnels

BGP envoie des messages en utilisant TCP :

- OPEN : ouvre une connexion TCP à un peer et authentifie l'émetteur
- UPDATE : affiche un nouveau chemin, pour ne mettre à jour qu'une route en particulier et ne pas tout renvoyer
- KEEPALIVE : garde une connexion ouverte en l'absence d'UPDATEs. Il permet aussi d'acquitter les messages OPEN
- NOTIFICATION : rapporte des erreurs dans les messages précédents. Est aussi utilisé pour fermer une connexion.

Politique d'importation et d'exportation de BGP : décider qui on choisit lorsqu'on reçoit des routes et ce qu'on désire montrer ou pas. Exemple de politique :



- X n'affichera pas qu'il peut router des paquets de B vers C en passant par lui-même.
- A prévient B du chemin AW et B prévient X du chemin BAW. B ne devrait pas prévenir C qu'il connaît BAW, car B n'a pas d'avantages à router BCAW, vu que C et W ne sont pas des clients de B. B va alors vouloir forcer C à router vers W via A ; B ne veut router que depuis et vers ses clients.

Pourquoi distinguer le routage inter et intra-AS :

- politique :
 - intra-AS : il n'y a qu'un seul administrateur, il n'y a pas de politique à appliquer
 - inter-AS : un AS veut contrôler comment le trafic est routé et qui traverse son réseau
- échelle : le routage hiérarchique permet de préserver la taille des tables et de diminuer le trafic de mise à jour
- performance :
 - intra-AS : peut se focaliser sur les performances
 - inter-AS : la politique prédomine sur les performances

Chapitre 5

La couche lien

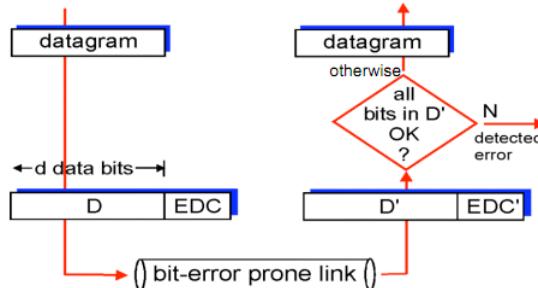
La couche lien s'occupe de la connexion d'un noeud à un autre via les interfaces. Elle offre les services suivants :

- la mise en trame et l'accès à un canal
- le transport fiable entre deux noeuds
- le contrôle de flux
- la détection et la correction d'erreurs (causées par le bruit)
- half- et full-duplex.

Cette couche est implémentée dans les NIC, c'est un mélange de software et d'hardware. Toutes les couches supérieures sont software.

5.1 Détection d'erreurs

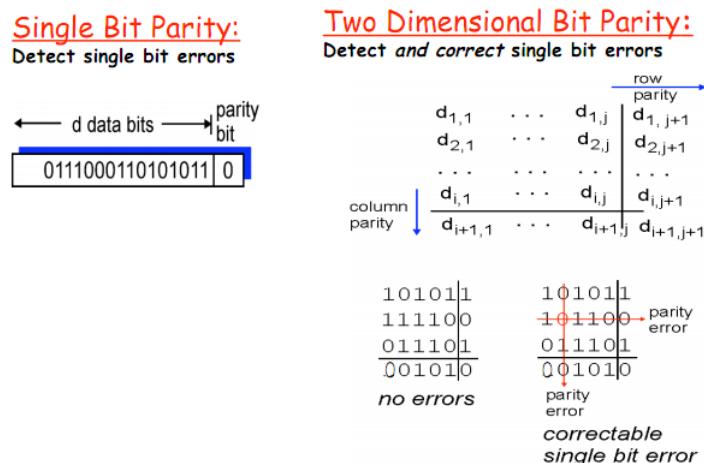
On transmet une trame accompagnée d'un code de détection d'erreur, des bits de redondance. Parfois, on inclut suffisamment de données pour permettre des corrections. Les données sont protégées par cette vérification, mais cela peut aussi inclure les headers.



Un code de détection d'erreur n'est pas toujours fiable, par exemple le bit de parité est correct s'il y a deux modifications de bits, ou si le bit de parité est lui-même changé.

Une erreur peut toujours s'y glisser ; en incluant de plus grands champs d'ED(C) (Error Detection (and Correction)) on en repère plus et mieux. Il y a toujours au minimum un bit d'overhead pour détecter une erreur.

On peut intégrer un système de réparation naïf avec des bits de parité pour du texte en deux dimensions.



Une forme carrée est la plus optimale. Pour n bits, les côtés font \sqrt{n} ; l'overhead en plus est donc proportionnel à \sqrt{n} .

Le mieux que l'on puisse faire, pour détecter ou corriger un bit erroné, se fait en $\log_2 n$.

5.1.1 Le checksum d'Internet

Le but est donc, dans la couche de transport, de détecter des erreurs dans les segments.

L'émetteur traite le segment comme des séquences de bits. Il génère le checksum en prenant la somme en complément à 1 du contenu des segments, et le place dans le champ de checksum.

Le récepteur fera la même somme et comparera à ce qui est contenu dans le champ checksum.

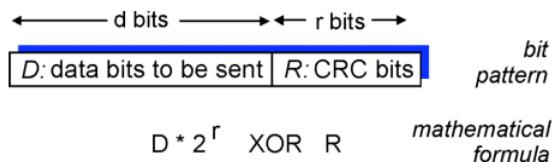
Le checksum d'Internet n'est pas plus performant qu'un bit de parité. Il se trouve dans la couche de transport car elle est implémentée en software, il faut donc un code facile à calculer. Le CRC est en hardware, ce qui rend l'exécution rapide.

5.1.2 CRC - Cyclic Redundancy Check

On voit les données comme un nombre binaire D . On choisit un générateur G de $r + 1$ bits.

On introduit un code R de r bits au début de la trame de façon à ce que toute la trame soit divisible par G en modulo 2.

Le récepteur connaîtra G et n'aura qu'à diviser $\langle DR \rangle$ par G en modulo 2 : si le résultat n'est pas nul, il y a une erreur.



Ce code peut détecter des séquences d'erreurs $\leq r$ bits et est largement utilisé en pratique.

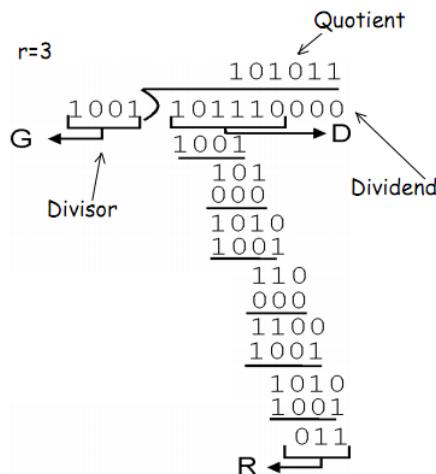
Le R peut se déduire facilement :

$$D \cdot 2^r \text{ XOR } R = nG \Leftrightarrow D \cdot 2^r = nG \text{ XOR } R$$

Après avoir appliqué un XOR à gauche et à droite. Cela équivaut à diviser $D \cdot 2^r$ par G en base 2 et à en prendre le reste. Dans cette division on ne tient pas compte des restes et des emprunts ; faire + ou - revient au même. On obtient alors

$$R = \text{reste}\left(\frac{D \cdot 2^r}{G}\right)$$

Exemple



On prend souvent une vue polynomiale pour les bits ; par exemple, $1101 \Leftrightarrow x^3 + x^2 + 1$. On a alors que la trame transmise

$$T(x) = D(x) \cdot x^r - R(x)$$

doit être divisible par $G(x)$ pour qu'elle soit correcte.

Supposons qu'une erreur $E(x)$ a été intégrée à la trame : elle vaut alors $T(x) + E(x)$, le récepteur calculera $\frac{T(x) + E(x)}{G(x)}$.

Le reste sera alors égal à $\frac{E(x)}{G(x)}$; si $E(x)$ n'est pas un multiple de $G(x)$, alors l'erreur sera détectée. De ce fait, $G(x)$ ne doit pas être choisi n'importe comment.

Situations où une erreur passe inaperçue : $G(x) = x^3 + 1$, $E(x) = x^3 + 1$ ou $E(x) = x^4 + 1$, car $G(x)$ divisible par ces nombres.

Il faut toujours un $G(x)$ avec un nombre pair de termes, car il détectera automatiquement les cas où il y a un nombre impair de bits d'erreur dans la trame (si $x = 1$ dans $G(x) = x^{16} + x^{12} + x^5 + 1 = (x+1)P(x)$, zéro à gauche et à droite. $E(x) = x^2 + x + 1$ n'est pas divisible par $x+1$). On fait au moins aussi bien qu'un bit de parité.

Le bit de parité est un cas particulier où $G(x) = x + 1$ (ou $G(x) = x$) ; $r = 1$, donc G doit être d'ordre 1.

Généralement, les erreurs ne sont pas isolées, elles se font en rafale localisée dans une zone de n bits. La rafale commence et se termine par un bit erroné (les bits entre peuvent être corrects ou non). Toute rafale de n inférieure à l'ordre de $G(x)$ est détectable.

Soit $E(x) = E'(x)x^k$, avec un entier k et $E'(x)$ la rafale d'erreurs. Si le degré de G est r et celui de $E' < r$, un polynôme de degré inférieur ne sera jamais divisible, du coup d'erreur sera toujours détectable.

$$\frac{E(x)}{G(x)}$$

C'est divisible par une polynôme de 16 bits, par exemple $G(x)$ lui-même.

5.2 Protocole à accès aléatoire

C'est une famille de protocoles permettant l'accès à une ressource partagée, ici un canal de communication. Il y a une collision quand un noeud reçoit deux ou plusieurs signaux en même temps.

5.2.1 Protocole à accès multiple idéal

Il remplirait les critères suivants :

- quand un seul noeud veut transmettre, il le fait à un débit R
- quand M noeuds veulent transmettre, chacun le fait à un débit moyen de $\frac{R}{M}$ (équité)
- complètement décentralisé, c'est-à-dire qu'il n'y a pas un noeud spécial qui coordonne les transmissions ni de synchronisation ni de slots.
- simplicité

On distingue 3 classes de protocoles :

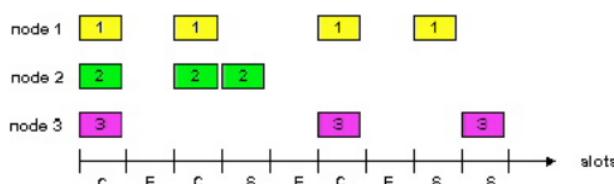
- le partitionnement du canal : on a une division du canal (slots temporels ou de fréquence), et on alloue chacune de ces divisions à un noeud
- les protocoles à accès aléatoire : le canal n'est pas divisé, ce qui génère des collisions dont on peut récupérer. Ces protocoles spécifient comment détecter les collisions et les récupérer.
- les protocoles "taking turns" : les noeuds peuvent utiliser le canal à tour de rôle

Une division temporelle ou par fréquence n'est pas optimale, car dans les deux cas il y a des slots qui restent inutilisés.

5.2.2 Slotted ALOHA

On suppose que

- toutes les trames ont la même taille
 - le temps est divisé en slots de même taille, et suffisamment long pour transmettre une trame
 - les noeuds sont synchronisés
 - si 2 ou plus noeuds transmettent dans un slot, tous les noeuds peuvent détecter la collision
- Lorsqu'un noeud reçoit une trame à envoyer, il la transmet dans le prochain slot.
- S'il n'y a pas de collision, le noeud attend une nouvelle frame à envoyer.
 - S'il y a collision, le noeud retransmet la trame dans le prochain slot avec une probabilité p , et ce jusqu'à y arriver sans collision.



Avantages :

- un seul noeud utilise tout le canal pour transmettre
- tout est décentralisé, il suffit juste d'une synchronisation
- simple

Inconvénients :

- avec les collisions, il y a des slots gaspillés
- des slots sont inutilisés
- les stations ne se rendent pas compte qu'elles créent des interférences ;
- nécessité d'une synchronisation

Calcul de l'efficacité

On va supposer N noeuds avec plusieurs trames à envoyer, chacun transmet dans un slot avec une probabilité p . La probabilité pour qu'un noeud ait accès à un slot est $p(1 - p)^{N-1}$.

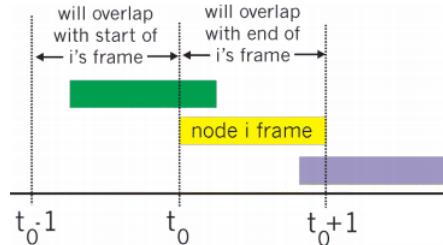
La probabilité pour que tous les noeuds ait accès est $Np(1 - p)^{N-1}$.

Pour maximiser l'efficacité avec N noeuds, on a $p^* = \frac{1}{N}$. Pour tous les noeuds, il faut prendre la limite de l'expression $Np^*(1 - p^*)^{N-1} = (1 - \frac{1}{N})^{N-1}$ avec N qui tend vers l'infini, ce qui donne $\frac{1}{e} = 0.37$.

Donc au mieux 37% des transmissions sont utiles dans le canal.

5.2.3 ALOHA pur

Il n'y a pas de slots ni de synchronisation. Quand une trame arrive, elle est transmise immédiatement.



La probabilité de collision augmente, car une frame envoyée au temps t_0 peut entrer en collision avec les frames dans l'intervalle $[t_0 - 1, t_0 + 1]$.

Calcul de l'efficacité

$$P(\text{succès}) = P(\text{le noeud transmet}).P(\text{pas de transmission dans } [t_0 - 1, t_0]).P(\text{pas de transmission dans } [t_0, t_0 + 1])$$

$$= p(1 - p)^{N-1}(1 - p)^{N-1} = p(1 - p)^{2(N-1)}$$

La probabilité de succès pour tous les noeuds est $Np(1 - p)^{2(N-1)}$. Si on prend un p optimal et N qui tend vers l'infini, on obtient $\frac{1}{2e} = 0.18$, soit 18% d'efficacité.

Efficacité par rapport au trafic moyen : soit $G = pN$ la moyenne de trafic agrégé (ou demandé) par unité de temps. C'est le nombre de tentatives de transmissions par unité de temps ; N stations tentent d'envoyer une frame avec une probabilité p à chaque unité de temps.

Pour le slotted ALOHA, l'efficacité est

$$Np(1 - p)^{N-1} = G\left(1 - \frac{G}{N}\right)^{N-1}$$

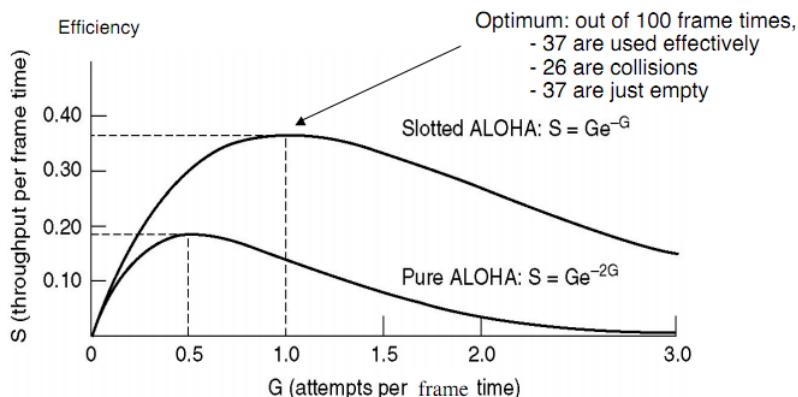
Si N est très grand, pour un G donné, l'efficacité tend vers $G.e^{-G}$

Pour l'ALOHA pur, on a l'efficacité

$$Np(1 - p)^{2(N-1)} = G\left(1 - \frac{G}{N}\right)^{2(N-1)}$$

Si $N \gg$, cela tend vers Ge^{-2G} .

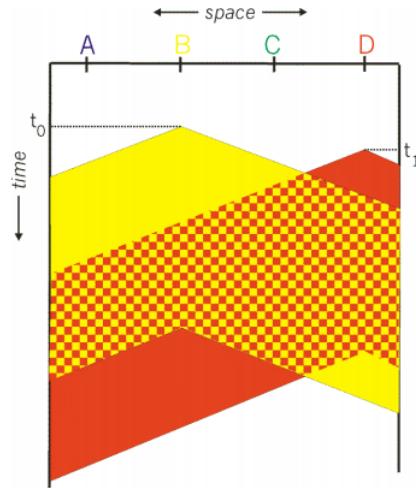
Si G est plus petit que 1, on a des bons résultats.



5.2.4 CSMA

Amélioration de ALOHA : on écoute le canal avant d'émettre. S'il semble libre, on émet ; s'il est occupé, on reporte la transmission (LBT - listen before talking).

Une collision peut toujours se produire lorsque le canal devient libre : toutes les stations vont tenter d'émettre, alors qu'elles ne s'entendent pas. Tous les paquets transmis sont gaspillés ; la distance et le délai de propagation sont à prendre en compte.



On peut définir deux sortes de CSMA :

- CSMA non persistant : si le canal est occupé, on réessaie plus tard
- CSMA persistant (ou p-persistant) : si le canal est occupé, on écoute jusqu'à ce qu'il soit libre

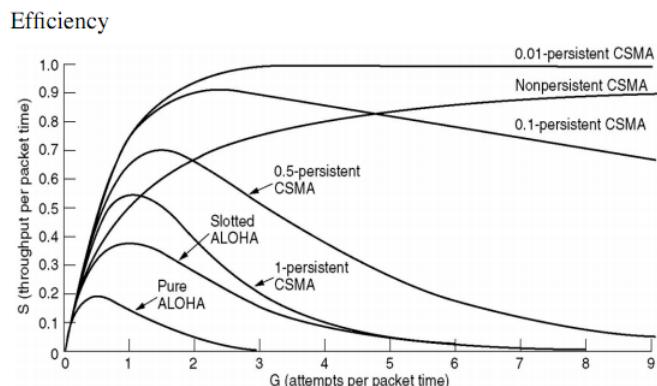
p-persistent CSMA:

```

While true do
    if channel is free
    then {
        with probability p: immediate transmission; or
        with probability 1-p: stay idle during at least propagation time ( $\tau$ )
    else listen until the channel is freed.

```

C'est un équilibre entre l'efficacité et le délai : cela introduit un délai inutile dans les charges faibles, mais l'efficacité est meilleure à des plus grandes charges.



Soient

- B le débit du canal
- F la taille de la trame
- L la longueur du canal
- c la vitesse de propagation
- τ le délai de propagation : $\frac{L}{c}$
- T le délai de transmission : $\frac{F}{B}$

τ est aussi le temps de contention ; une collision est possible au début d'une transmission, s'il n'y en a pas eu après un temps τ , il n'y en aura pas.

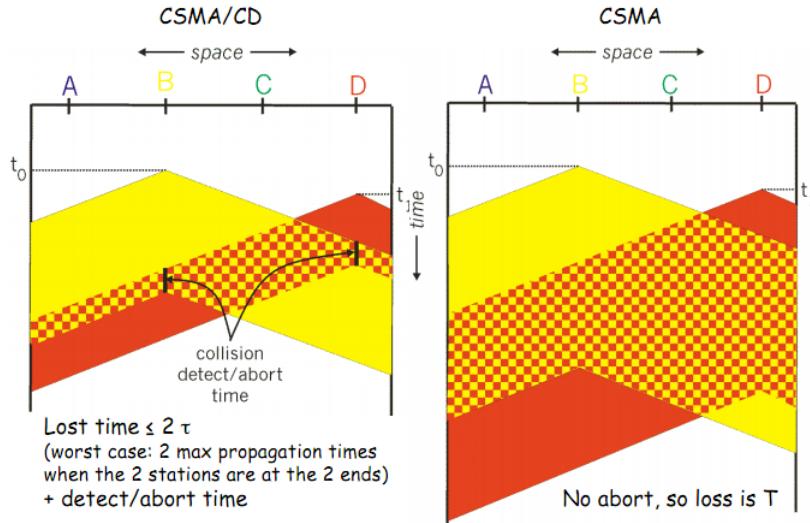
$$\alpha = \frac{\tau}{T} = \frac{BL}{cF}$$

Il vaut mieux ne pas couper les trames, car il faudra récupérer le canal autant de fois qu'il y a de coupures. En transmettant tout d'un coup, on ne demande la ressource qu'une fois.

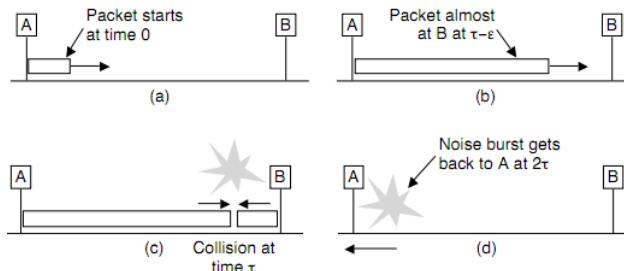
Lorsqu'on a acquis la ressource, τ est le temps pendant lequel il peut y avoir une collision, car les bits ne se sont pas propagés complètement. Après, pendant la durée $T - \tau$, il n'y a plus aucun risque de collision. Réduire les risques de collisions (donc augmenter l'efficacité du système) revient à minimiser τ . Le fait de segmenter la trame n'est pas une bonne idée, sinon il y a autant de risques de collision que de coupures.

5.2.5 CSMA/CD

CSMA avec une détection de collision : la station s'écoute elle-même, et si l'écoute est différente de ce qu'on émet, alors il y a collision. C'est facile sur les réseaux câblés, il suffit de comparer la force des signaux, mais c'est plus compliqué en sans-fil.



La trame doit avoir une longueur minimale, sinon une station peut émettre sans se rendre compte qu'il y a eu collision.



Il faut donc que $T > 2\tau$, donc que $\frac{F}{B} > 2\frac{L}{c}$, ce qui conduit à

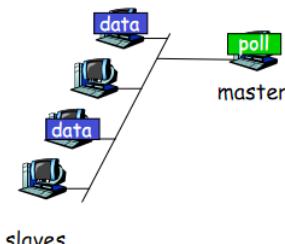
$$F_{\min} = 2 \frac{BL}{c} = \pm BL 10^{-8} \text{ bits}$$

Ethernet a choisi une longueur de 64 bytes, soit 512 bits (avec des marges supplémentaires à cause des autres délais).

5.2.6 Protocoles Taking turns

Protocoles tentant d'avoir les avantages des protocoles à accès aléatoire et des protocoles à partitionnement de canal.

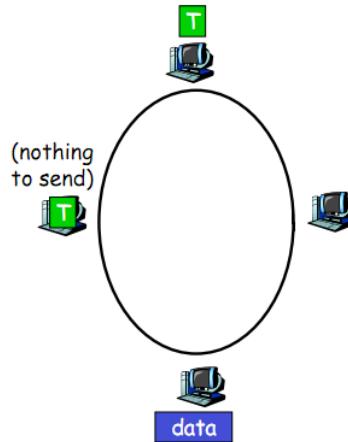
Polling



Un noeud maître invite des noeuds esclaves à transmettre tour à tour. C'est un système généralement utilisé lorsque les périphériques esclaves sont simples, "dumb".

Les problèmes sont le surplus qu'entraînent les headers de polling, la latence, et le fait qu'il n'y a qu'un seul point critique (le maître).

Token passing



Un jeton de contrôle est passé d'un noeud à l'autre de manière séquentielle. Il s'agit d'une trame avec un contenu particulier, qui permet à celui qui la reçoit de transmettre sur le canal.

Les problèmes sont les overheads, la latence, et la présence d'un seul point critique (le jeton, qui peut être perdu ; il faudra décider qui en relance un).

5.3 Adressage

Un adressage est nécessaire où il y a de multiples destinataires à l'écoute.

Une adresse MAC (ou LAN ou physique ou Ethernet) ne sert pas à localiser un hôte (car on sait déjà où il se trouve via l'IP), c'est un identifiant. Chacune de ces adresses MAC est sur 48 bits et est unique. Elles ne dépendent pas de la localisation, donc elles ne changent pas.

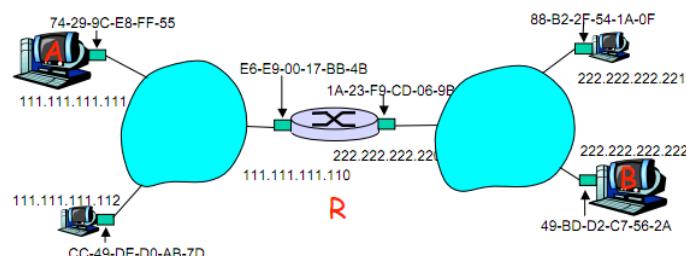
A titre de comparaison, les IPs sont des adresses hiérarchiques qui permettent surtout d'atteindre le dernier routeur, soit le subnet de destination. Elles peuvent changer dans le temps, elles ne sont donc pas portables.

Il faut un protocole pour pouvoir déterminer une adresse MAC à partir d'une adresse IP, on utilise ARP (Address Resolution Protocol) : une table ARP contient un mapping d'adresses IP-MAC, avec un TTL (time-to-live). Seules les IP du subnet s'y trouvent.

Supposons qu'on veut envoyer une trame à un hôte dont on ne connaît pas l'adresse MAC :

1. On envoie une trame broadcast (FF-FF-FF-FF-FF-FF) qui contient l'adresse IP cherchée, et en indiquant l'adresse MAC source.
2. Lorsque l'hôte possédant l'IP recherchée la recevra, il répondra avec son adresse MAC.

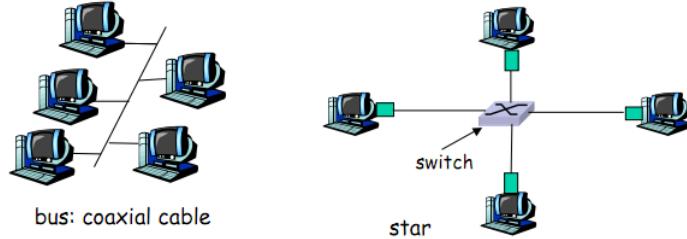
La table ARP permet de faire du caching, pour ne pas redemander tout le temps l'adresse MAC. L'envoi de la trame en broadcast permet de découvrir une association IP-MAC chez tous les hôtes, cela met à jour/ajoute une entrée à la table ARP. C'est un système plug-and-play, il n'y a aucune intervention d'un administrateur dans la création d'une table ARP.



Lorsque les hôtes se trouvent sur deux subnets différents, lorsqu'on envoie la trame, l'adresse MAC source est celle de A et celle de destination celle du routeur de sortie, et pas celle de B, sinon B serait sur le subnet. Il faut viser le routeur de sortie, qui va décapsuler et remplacer les adresses MAC : l'adresse source sera l'adresse MAC du routeur, la destination sera celle de B. A détecte que le destinataire n'est pas dans le subnet grâce au masque réseau et à l'adresse IP de destination. Au niveau des requêtes ARP (avant d'envoyer la trame), il faut demander l'adresse MAC du routeur ; elles sont locales à un sous-réseau.

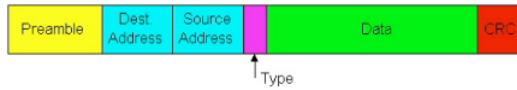
5.4 Ethernet

Au début de son utilisation, une topologie autour d'un bus était très populaire. Tous les noeuds étaient dans le même domaine de collision. Maintenant, une topologie en étoile prévaut : on a un switch central, et chaque machine utilisent un protocole Ethernet séparé. Les noeuds n'engendent ainsi pas des collisions entre eux.



Les raisons du succès d'Ethernet :

- un des premiers protocoles
- simple
- même débits que les autres technologies
- bon marché



Une trame Ethernet contient

- les adresses MAC source et de destination
- le CRC
- les données
- des bits de préambule pour synchroniser les clocks de l'émetteur et du récepteur
- le type de protocole de couche supérieur à qui livrer la trame (généralement IP)

L'Ethernet

- est sans connexion, il n'y a aucun handshaking entre le récepteur et l'émetteur
- non fiable, il n'y a pas d'ACK ou de NAK. S'il y a des trous dans les flux de trames, ils seront rebouchés avec TCP, sinon par les applications.
- utilise CSMA/CD sans slot :

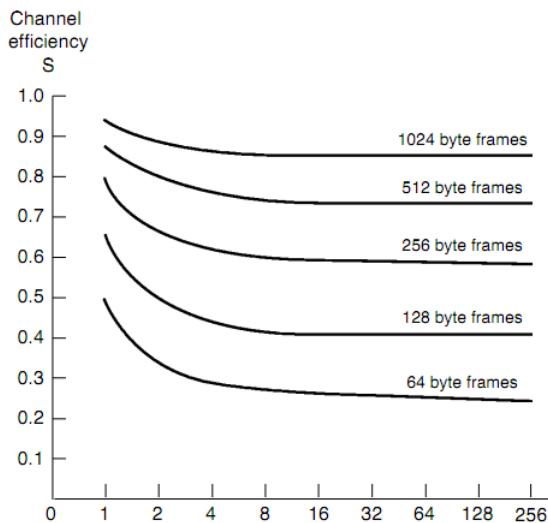
1. lors de la réception d'un datagramme, la NIC (Network Interface Card) crée une trame
2. si la NIC constate que le canal n'est pas occupé, il transmet la trame. Sinon, il attend que le canal se libère, ensuite il transmet
3. si la NIC transmet la trame entière sans collision, son travail est terminé
4. si la NIC détecte une collision lorsqu'il transmet, il s'arrête et envoie un signal de jam pour s'assurer que tout le monde détecte bien la collision
5. après l'annulation, la NIC entre dans le mode de backoff exponentiel : après m collisions rencontrée pour transmettre une même trame, la NIC choisit un k aléatoire dans $[0, \dots, 2^m - 1]$. Il attendra $k \times$ le temps pour transmettre 512 bits et retournera à l'étape 2.

L'exponential backoff tente ainsi d'estimer la charge ; plus celle-ci est élevée (donc plus il y a des collisions), plus le délai d'attente risque d'être long. C'est une sorte de CSMA p-persistant, avec un p adaptatif.

Soit α le nombre moyen de slots avant d'arriver à transmettre. On a l'efficacité S :

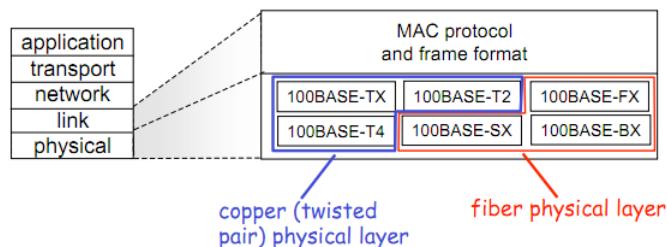
$$S = \frac{T}{T + \alpha \times 2\tau} = \frac{1}{1 + \alpha \times \frac{2\tau}{T}} = \frac{1}{1 + \alpha \times \frac{2BL}{cF}}$$

Pour N grand, α converge vers e pour un CSMA p-persistant avec un p^* idéal de $\frac{1}{N}$.
Donc pour un N très grand, l'efficacité ne peut pas être meilleur que $\frac{1}{1+e \times \frac{2\tau}{T}}$



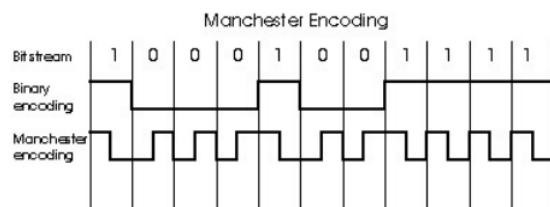
Ethernet est baseband, c'est à dire que le signal n'est pas envoyé dans une autre bande de fréquences, il est envoyé tel quel.

Il y a plusieurs standards pour les liens Ethernet, alors que le protocole MAC et le format des trames est le même. Il y a différentes vitesses (2/10/100/1000 Mbps) et différents supports physiques (fibre, câble).

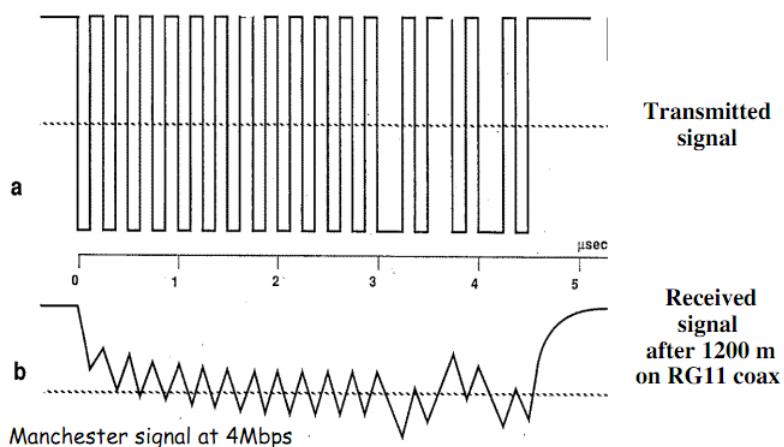


5.4.1 Encodage de Manchester

Généralement utilisé par Ethernet.



Chaque bit possède une transition. Les clocks des noeuds expéditeur et récepteur peuvent ainsi se synchroniser, il n'y a pas besoin d'une clock globale parmi les noeuds. Cet encodage agit au niveau de la couche physique.



5.5 Switchs et hubs

Un hub n'agit que sur la couche physique; pour chaque trame reçue, il va la répéter sur tous les périphériques connectés. Il agit comme un bus. Il régénère le signal (au cas où il serait affaibli), il n'y a aucune "intelligence".

Il n'y a pas de buffering des trames, ni de CSMA/CD. Des collisions sont donc possibles.

Un commutateur (switch) est plus sophistiqué, il agit sur la couche 2. Il réceptionne les trames, les stocke, vérifie si elles sont complètes, et les transmet au bon destinataire. Il est invisible pour les hôtes.

Contrairement aux hubs, il a un rôle actif. Ils sont plug-and-play et apprennent par eux-mêmes, ils n'ont pas besoin d'être configurés. Il n'y a plus aucune collision si les liens sont fullduplex (2 paires torsadées pour chaque sens par exemple).

Il y a éclatement des domaines de collision, qui fait que la probabilité d'en avoir une est très faible.

Le switch a besoin d'une table d'acheminement, qui map les adresses MAC aux interfaces (\neq table ARP). Il y a de l'auto-apprentissage, il va lui-même la créer ; chaque fois qu'il reçoit une trame, il peut associer l'adresse MAC source à l'interface d'arrivée. Si l'adresse de destination n'est pas dans la table, on envoie la trame à tout le monde. Il y a donc par moment des inondations avec un switch.

Une trame peut devoir être renvoyée dans l'interface par laquelle elle est entrée dans le switch (par exemple avec un hub connecté à cette interface). Dans ce cas le switch jette la trame.

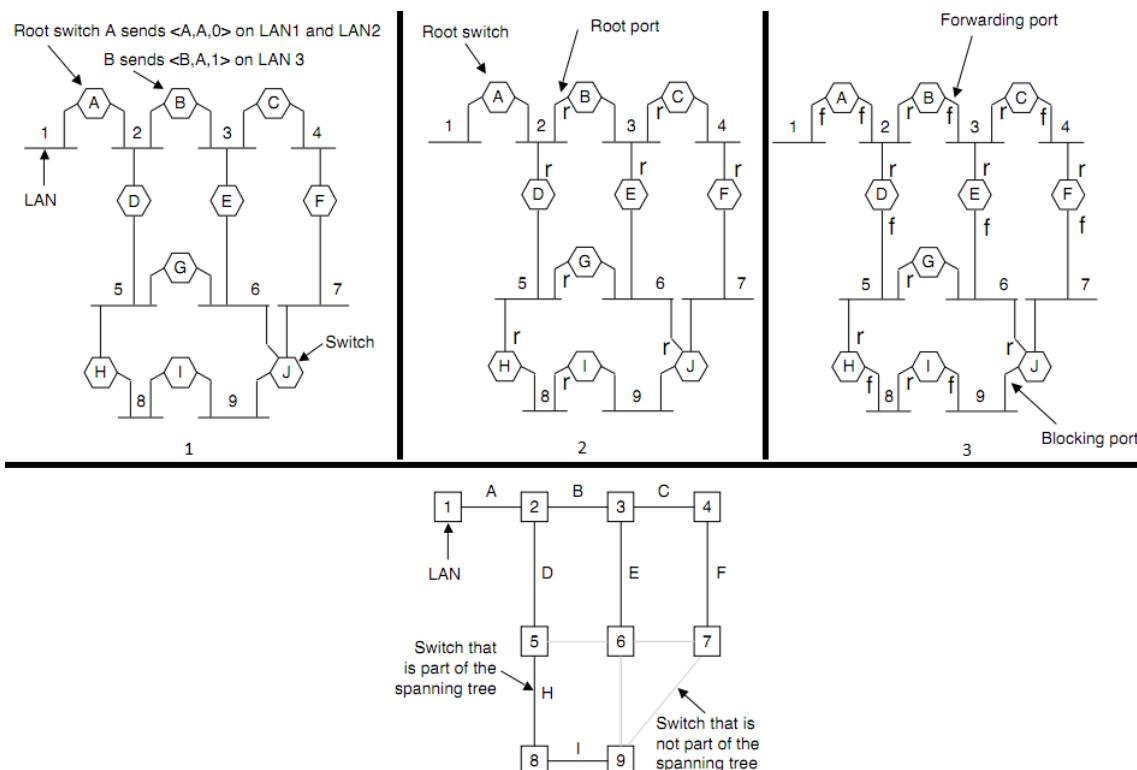
Ce qui se passe quand un switch reçoit une trame :

1. record link associated with sending host
 2. index switch table using MAC dest address
 3. if entry found for destination
 - then {
 - if dest on segment from which frame arrived
 - then drop the frame
 - else forward the frame on interface indicated
- else flood**
- forward on all but the interface
on which the frame arrived*

L'algorithme fonctionne car il n'y a pas de cycle, mais cela diminue la robustesse.

Pour pouvoir utiliser des cycles, il faut construire un spanning tree sur la topologie réelle. On utilise pour cela des paquets BPDU (Bridge Protocol Data Unit) et on procède ainsi :

1. on détermine le switch racine, le root id. On utilise pour cela des paquets BPDU qui contiennent un triplet (id du switch source, la racine supposée, la distance à la racine)
2. on construit l'arbre ; grâce aux flux de BPDUs, un switch peut connaître sa distance à la racine et quel port y mène en prenant la plus petite distance.
3. on décide si les ports non root seront bloquant ou de type forwarding



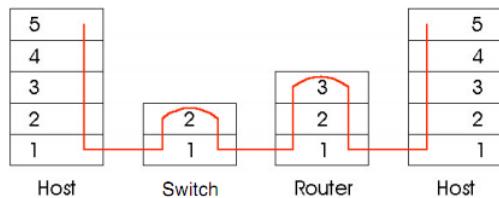
Le résultat est un spanning tree, qui n'est pas nécessairement optimale (il l'est seulement pour la racine). Il est le même pour toutes les paires source-destination.

Certains switchs ou liens peuvent ne pas être repris dans le spanning trees. Ils peuvent être utiles en cas de défaillance, c'est pour ça que les ports bloquants doivent quand même écouter les BPDUs (mais ne pas laisser passer le reste).

Comparé à un routeur :

- les routeurs font partie de la couche réseau, les switchs de la couche lien
- les routeurs effectuent du routage et de l'acheminement IP et implémentent des algorithmes de routage.

Les switchs maintiennent des tables d'acheminement MAC, implémentent du filtrage, de l'apprentissage et des algorithmes de spanning tree.



	<u>hubs</u>	<u>routers</u>	<u>switches</u>
traffic isolation	no	yes	yes
plug & play	yes	no	yes
optimal routing	no	yes	no
cut through	yes	no	yes <i>Some do</i>

Avantages des switchs :

- il y a élimination des collisions
- les liens physiques peuvent être de plusieurs types
- auto-gestion : si un switch tombe en panne, la reconfiguration est automatique

	Pour	Contre
Switch	<ul style="list-style-type: none"> - plug-and-play - filtrage et forwarding à haut débit 	<ul style="list-style-type: none"> - il faut prévenir les cycles avec un spanning tree non optimal - gros trafic ARP si le réseau est grand - non protégé contre le broadcast abusif
Routeur	<ul style="list-style-type: none"> - pas de cycle - le chemin est optimal - firewall 	<ul style="list-style-type: none"> - pas plug-and-play - il y a plus de temps de processing (3 couches à traiter)

Un routeur n'est pas plug and play car il faut définir les subnets, le plan d'adressage.

Un périphérique est cut through si la propagation est directe, c'est-à-dire pas de modèle store-and-forward. Le hub traite directement les bits. Le routeur non, car il y a un traitement. Certains switchs peuvent le faire, mais en général non.

5.6 PPP

C'est un protocole point à point, sans broadcast, avec un émetteur et un récepteur. Il n'y a pas de collision ni de problème d'accès ni de MAC (Media Access Control).

Une implémentation de PPP doit posséder les caractéristiques suivantes :

- il encapsule des datagrammes de la couche lien dans des trames. Il peut prendre des trames de n'importe quel type et est capable de démultiplexage.

- bit transparency : on ne doit retrouver nulle part la séquence de bits qui sert de flag de début et de fin de la trame.
- détection d'erreur, pas de correction
- connection liveness : sorte de battement de cœur, permet de vérifier que le lien fonctionne toujours.
- négociation d'adresse de couche réseau : les points peuvent apprendre/configurer chaque adresse des autres réseaux

PPP ne requiert pas

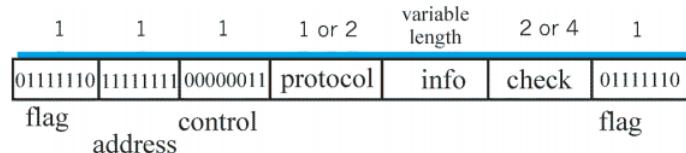
- de correction/récupération d'erreur
- de contrôle de flux
- de livraison des trames dans l'ordre
- de supporter les liens multipoints (ex : polling)

La récupération d'erreur, le contrôle de flux et le réordonnancement des données est délégué aux couches supérieures.

5.6.1 Trame PPP

Une trame PPP contient les éléments suivants :

- des flags, pour délimiter le début et la fin d'une trame
- une adresse (inutile, vu qu'il n'y a qu'un récepteur)
- des bits de contrôles qui ne font rien
- le protocole auquel il faut délivrer la trame (PPP-LCP, IP, etc)
- les informations à transmettre
- un champ de check, pour de la détection d'erreur



La transparence des données (bit transparency) requiert que l'on puisse quand même transmettre le flag 01111110 dans les données. En l'état, la trame serait coupée et inutilisable.

Il y a deux solutions :

- le byte stuffing : on ajoute un flag supplémentaire après celui dans la trame.
Le récepteur quand à lui, lorsqu'il détectera les 2 bytes 01111110 consécutifs, jettera le premier et continuera la réception des données.
- le bit stuffing : l'émetteur ajoute un 0 systématiquement lorsqu'il y a cinq 1 successifs, même si ce n'est pas le fagnon.
Le récepteur va supprimer tout zéro qui se trouve après cinq 1. S'il y a six 1 consécutifs, il vérifiera si c'est un flag.