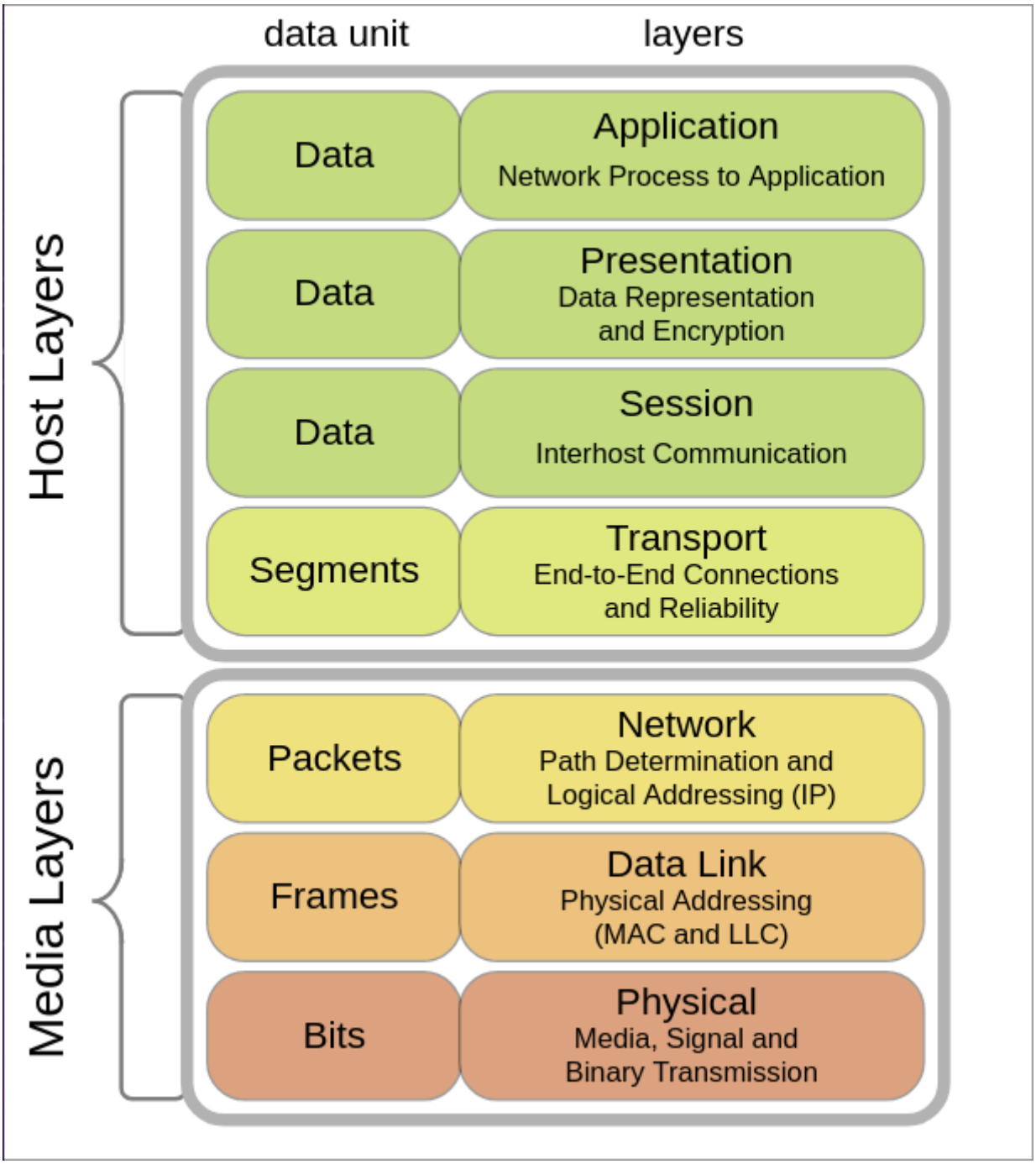
Réseau

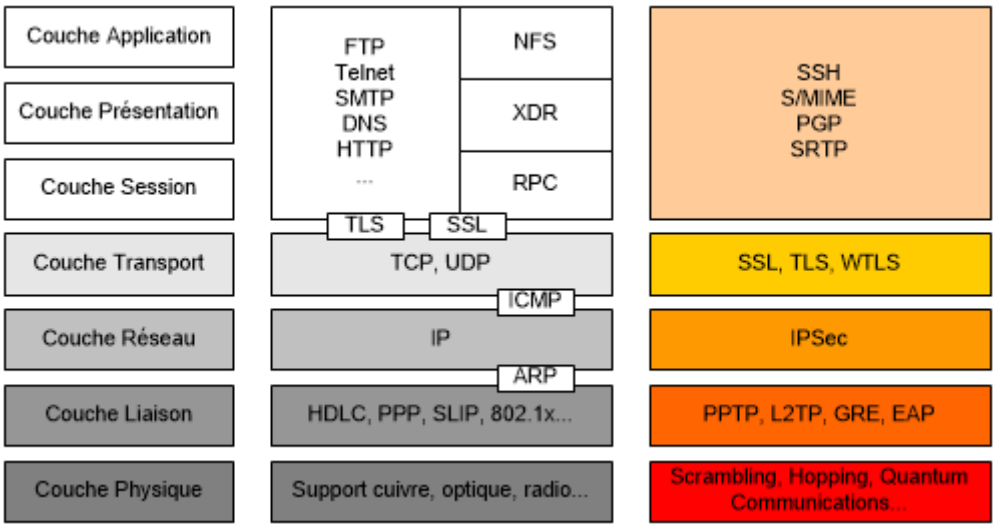
Introduction au modèle OSI

Les modèles

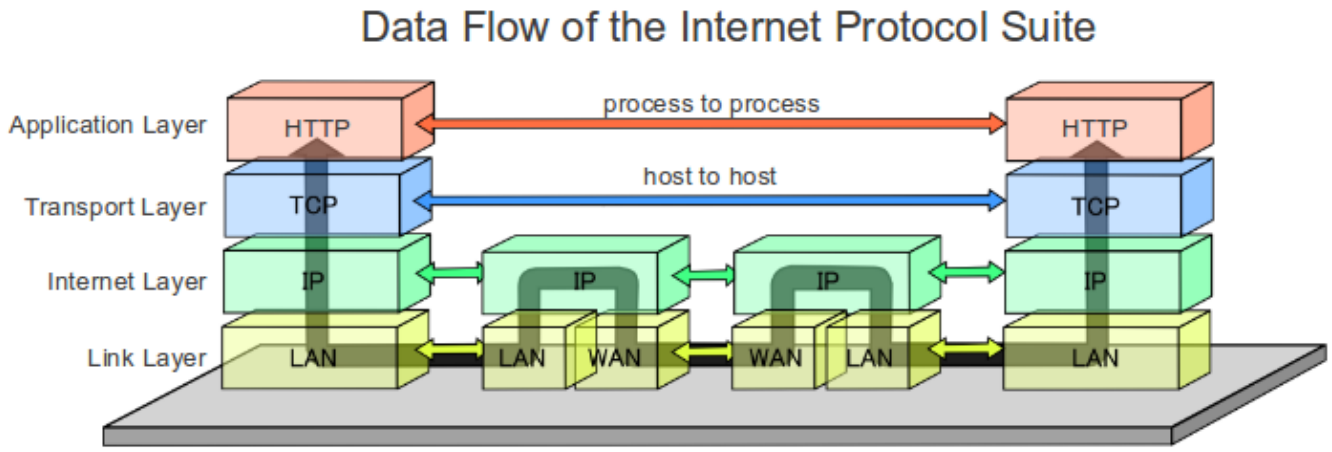


Le modèle OSI est une norme de communication réseau proposée par ISO. Elle met ainsi en relation plusieurs protocoles de communications différents (IP, HDLC, TCP, UDP, HTTP, etc).

Voici par exemple une liste de différents protocoles pour chaque couche du modèle.



Et voici comment une communication entre deux machines s'opère dans ce modèle :



Couche applicative (DNS, HTTP, SMTP, IMAP et POP)

Si on fait abstraction de toutes les couches en dessous de la couche application, on trouve le protocole applicatif. Le protocole applicatif définit comment les données de l'application peuvent être demandées et envoyée (par exemple via HTTP pour des sites internet, IMAP pour recevoir des emails ou encore SMTP pour envoyer des emails).

Le protocole applicatif est le langage utilisé par l'application pour communiquer, il décrit donc la forme des messages et le sens des échanges (définition syntaxique et sémantique).

Pour s'identifier, les applications utilisent un port et une IP (IPv4 ou IPv6), l'IP indique la machine et le port défini l'application émettrice ou destinataire (exemple, 80 pour HTTP, 443 pour HTTPS, 22 pour SSH, 53 pour DNS, etc).

Le DNS

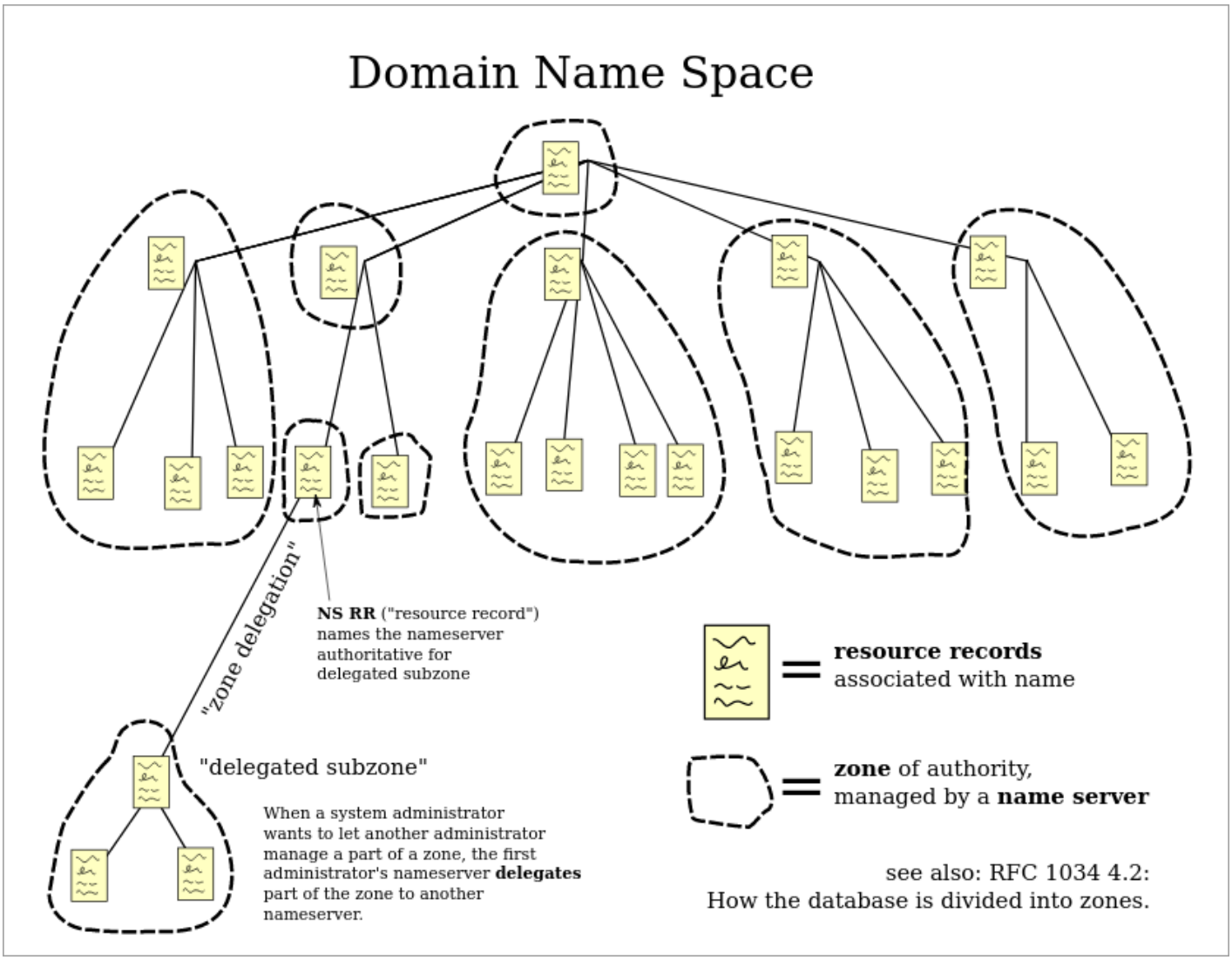
Chaque machine est identifiée au moyen d'une adresse IP (codée sur 32 bits IPv4 ou 128 bits avec l'IPv6).

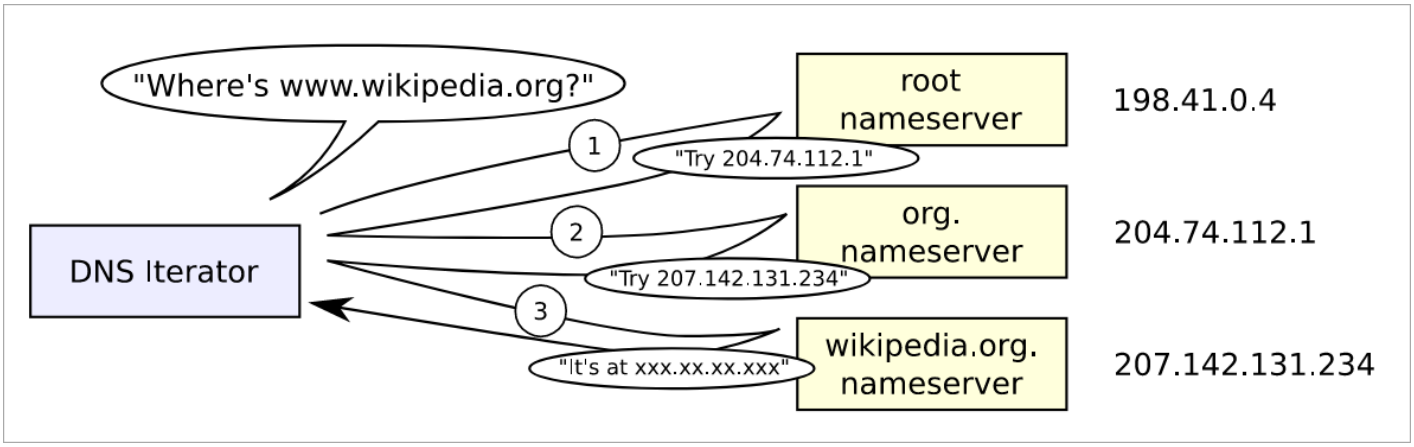
Le but du DNS (Domain Name System) est d'agir comme une sorte d'annuaire, ainsi à la place de devoir retenir des choses tel que `2001:41d0:404:200::597` il suffit de retenir `snowcode.ovh`. Il est donc possible de réserver un nom de domaine (généralement payant).

Une première manière de gérer cela serait d'avoir un fichier texte liant un nom et une adresse IP, par exemple avec `/etc/hosts` qui lie automatiquement `localhost` à l'adresse `127.0.0.1`.

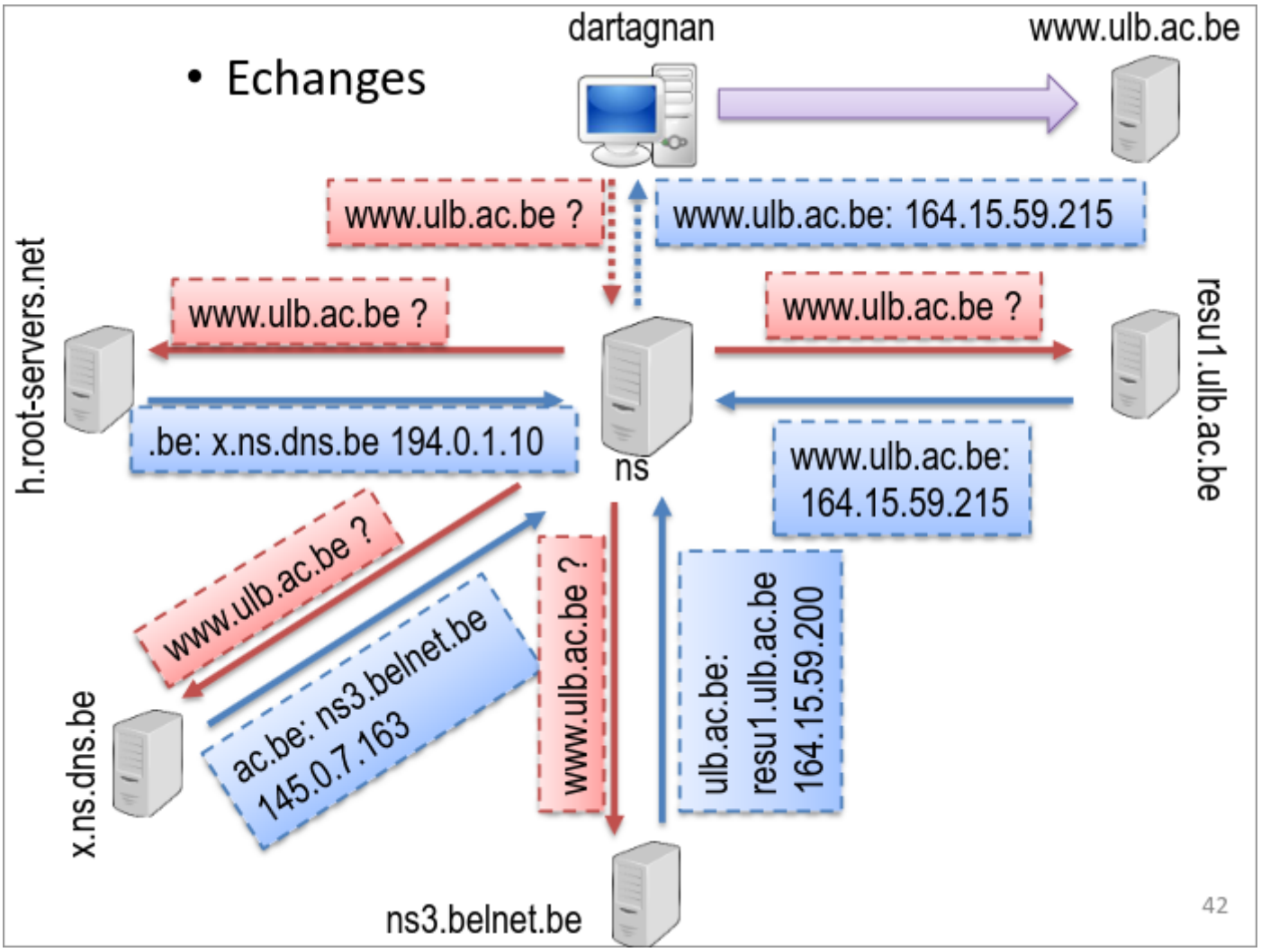
Le problème est qu'il serait impossible de synchroniser une base de donnée contenant tous les noms de domaines de tout le monde sur tous les appareils. On pourrait alors décider de créer une base de donnée centralisée, le problème est alors que tout le monde dépendrait d'un unique serveur pour les noms de domaines.

C'est pour cela qu'il y a plusieurs serveurs liés entre eux. Par exemple si on veut aller sur `swilabible.be`, on va d'abord demander au serveur mondial où se trouve le DNS de `be`, ce dernier peut alors ensuite renvoyer où se trouve `swilabible.be`. Il est donc possible d'avoir plus de niveaux d'imbrications.





Il est donc possible de créer son propre DNS pour gérer ses propres sous domaines.



Le serveur qui va effectuer cette recherche est généralement celui qui est proposé par le réseau local (sauf s'il a été spécifiquement été précisé dans la configuration du système) via le protocole DHCP.

Mail

Les mails utilisent plusieurs protocoles, le SMTP (via le port 25) permet d'envoyer des messages à des serveurs (tel que gmail.com, outlook.com, etc), la réception d'un message (ouvrir sa boite mail) se fait via les protocoles POP3 (si on veut télécharger tous les mails localement) ou IMAP (si on veut ne pas avoir à télécharger tout localement).

Les mails doivent toujours respecter la RFC 1855 afin de pouvoir être bien reçue et comprise par son destinataire.

Pour transmettre une pièce jointe (binaire) en texte, on peut utiliser le base64 qui va encoder le binaire avec des caractères alphanumériques et quelques caractères spéciaux.

Le problème est que les protocoles de mail sont très vieux, ainsi, il est tout à fait possible de prétendre être quelqu'un d'autre, aussi les mails ne sont pas chiffrés et beaucoup de fournisseurs d'accès internet bloquent le port 25.

Il est cependant indispensable de pouvoir supporter les mails, car les emails sont devenus incontournables au fil du temps.

Le web

Le web a été développé au CERN à Genève, derrière le "web" il y a plusieurs éléments (normes, protocoles, applications) :

- HTML (description de documents) ;
- Des serveurs web pour les délivrer (exemple apache ou nginx) ;
- Des clients pour les lire (Firefox, Chrome, etc) ;
- Le protocole HTTP qui permet à ces deux éléments de communiquer (versions allant de 1 à 3).
- Définition de l'URL (protocol://machine:port/chemin), par exemple (https://books.snowcode.ovh:8888/hello.html)

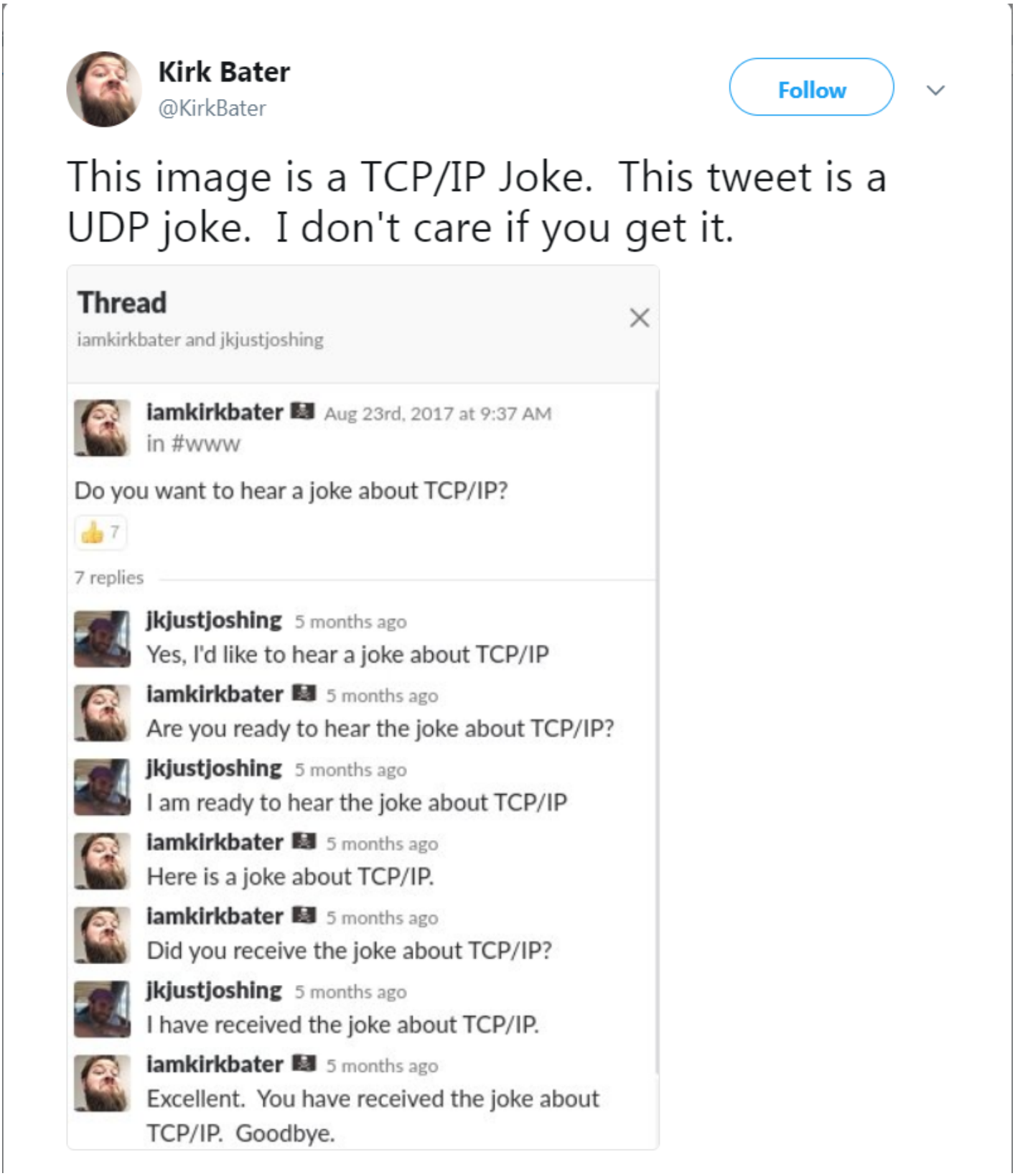
Chaque requête contient une commande (GET, HEAD, POST, PUT, DELETE), un entête (contenant des informations comme un token d'accès pour les cookies, d'où on vient, les métadonnées sur l'appareil, etc), et le corps qui est le contenu de la requête (page ou formulaire).

Ces entêtes peuvent être utilisés pour identifier et tracker des utilisateurs, car leur combinaison permet d'identifier des utilisateurs avec une certaine précision. Cela peut notamment être testé sur amiunique.org.

Couche de transport (UDP, transport fiable et TCP)

La couche applicative repose sur la couche de transport. Cette dernière s'en fout du type de donnée utilisée, cette couche a seulement pour but de transférer les données.

Il existe deux protocoles, le **TCP** (Transmission Control Protocol) qui permet d'envoyer des informations de manière fiables (en vérifiant la bonne réception des "paquets" de données), et l' **UDP** (User Datagram protocol) est un protocole qui envoie les paquets sans se soucier de la bonne réception. Ce dernier, bien que moins fiable, est beaucoup plus rapide.



Il faut donc connaitre le port du programme à contacter, pour cela le système maintient un annuaire liant un numéro de port à une application.

L'UDP (User Datagram Protocol)

40	**320**	Source Port	Destination Port
44	**352**	Length	Checksum
48	**384+**	Data	

Avec l'UDP on va simplement transmettre les données sans se soucier de leur bonne réception. Ainsi pour chaque message (TPDU, Transport Protocol Data Unit) il faut connaitre le numéro de port source (et destination ainsi que la longueur du message et éventuellement un "checksum" permettant de vérifier l'intégrité des informations.

Le protocole UDP est très utilisé pour les applications qui ont besoin d'aller vite, même si cela veut dire de potentiellement perdre des informations. Par exemple pour les jeux massivement multijoueurs, les diffusions en direct de vidéo ou audio, etc.

Le TCP (Transmission Control Protocol)

Le problème avec l'UDP est qu'il n'y a aucune vérification de la bonne réception des paquets ou encore de leur ordre ou de leur intégrité.

Le but du protocole TCP est de garantir l'intégrité des données.

Transfert fiable

TCP est donc un protocole qui implémente le "transfert fiable", nous allons voir ici en quoi consiste le transfert fiable.

Le transfert fiable est une façon de transférer l'information entre un émetteur et un récepteur de telle sorte à pouvoir palier à des pertes, des duplications, des altérations ou un désordre parmi les informations.

Pour cela, chaque TPDU (Transport Protocol Data Unit) contient un "checksum" permettant de vérifier que l'information n'est pas corrompue → protection contre l'altération.

Et lors de chaque réception d'information, le récepteur doit confirmer la bonne réception, si l'émetteur ne recoit aucun acquis de bonne réception avant un certain temps (timer), il considère que l'information est perdue et la renvoi → protection contre la perte d'information.

Si l'acquis lui-même est perdu, l'émetteur va renvoyer l'information et le récepteur va réenvoyer son acquis, car ce dernier a déjà reçu l'information → protection contre la duplication et la perte d'acquis.

Chaque acquis et chaque envoi d'information est donc numéroté, il est ainsi possible de savoir pour chaque acquis à quoi il fait référence. Si un acquis est donc envoyé deux fois, l'émetteur pourra savoir à quelle information chaque acquis fait référence et agir en fonction. S'il envoie une information 1, reçois l'acquis pour 1, puis envois une information 2 et reçois de nouveau un acquis pour 1, il ne prendra pas compte du deuxième acquis → protection contre la duplication d'acquis.

Les acquis et les informations étant ainsi numérotées et allant dans un ordre de croissant. Et puis ce que l'émetteur attend toujours d'avoir reçu une confirmation de bonne réception de chaque partie de l'information, ce protocole assure donc que les informations sont reçues dans le bon ordre → protection contre le désordre.

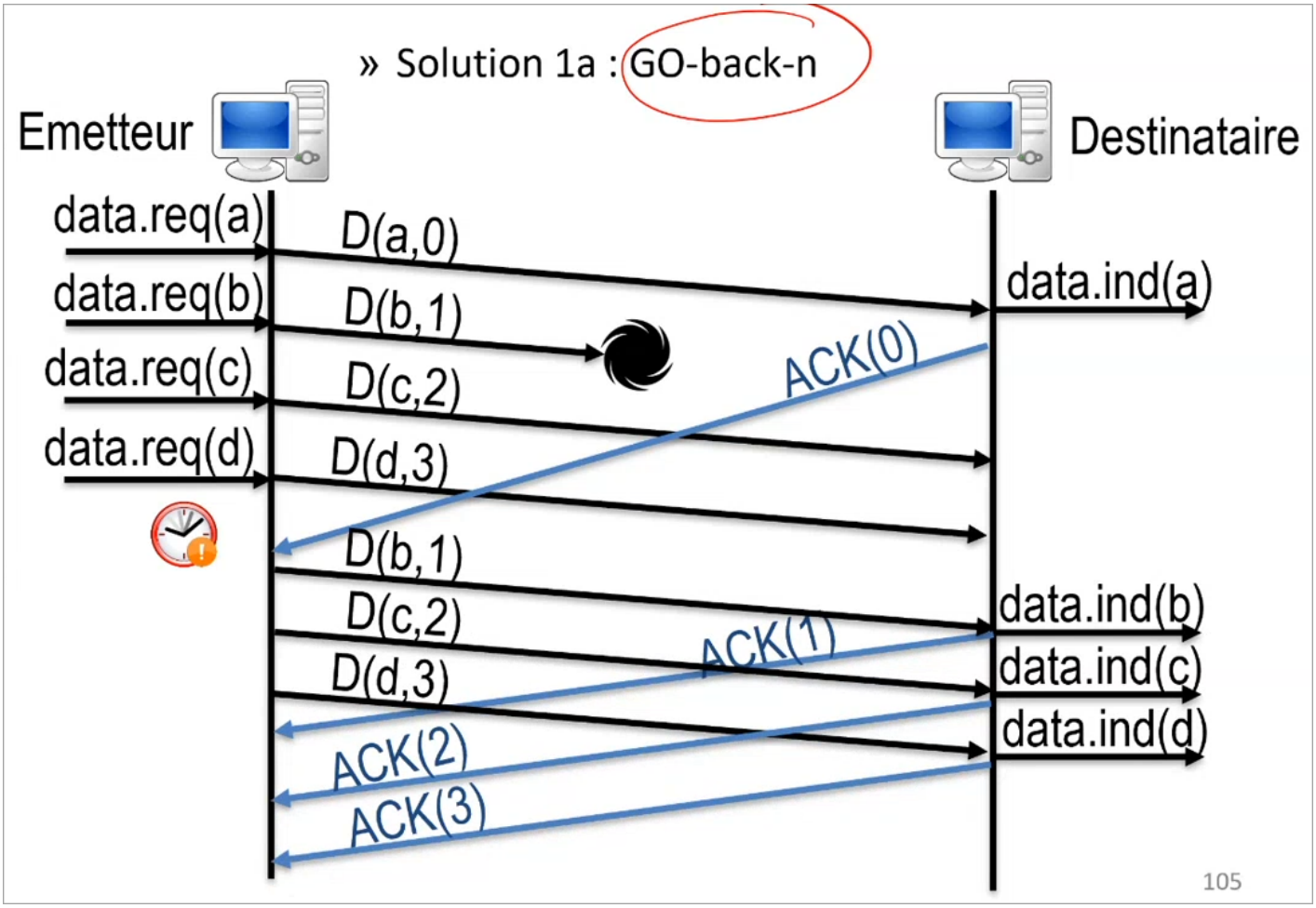
Fenêtre glissante

Les performances de TCP sont bien plus mauvaises que UDP car si le ping est élevé le round-trip time (temps allé-retour) va être très élevé aussi, cela sera donc très lent de tout transmettre. Pour résoudre ce problème, on peut alors utiliser un système de "fenêtre glissante", on va envoyer plus d'information avant d'attendre un acquis (donc moins d'acquis et pas d'envois de trop de données).

La fenêtre défini une série de numéros de séquences qui peuvent être envoyés sans devoir attendre un acquis. Une fois cette fenêtre épuisée, il faut attendre un acquis (pour toute la fenêtre) pour pouvoir recommencer.

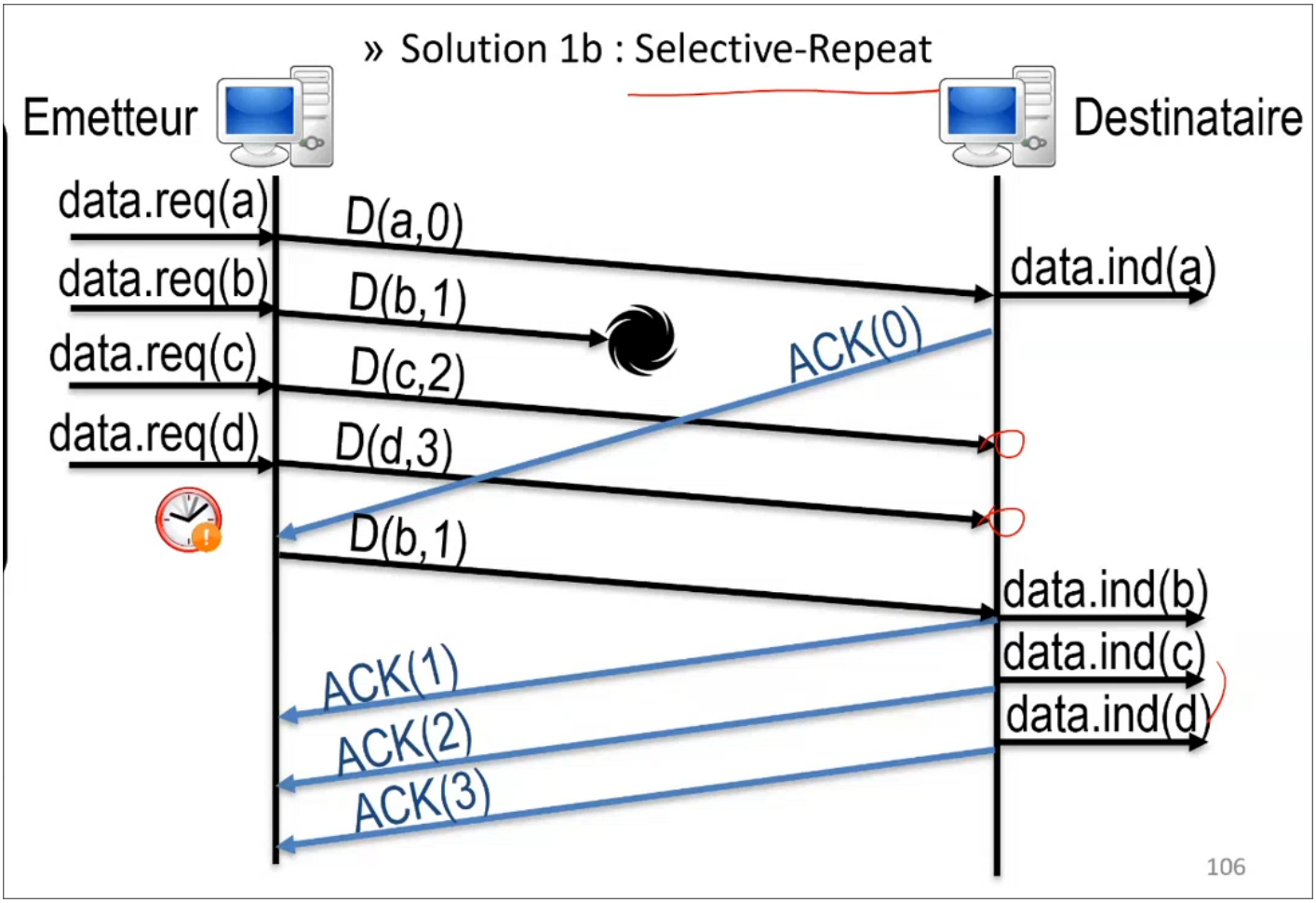
Perte d'information

Lorsqu'une perte d'information survient, il y a plusieurs manières de palier à une perte.

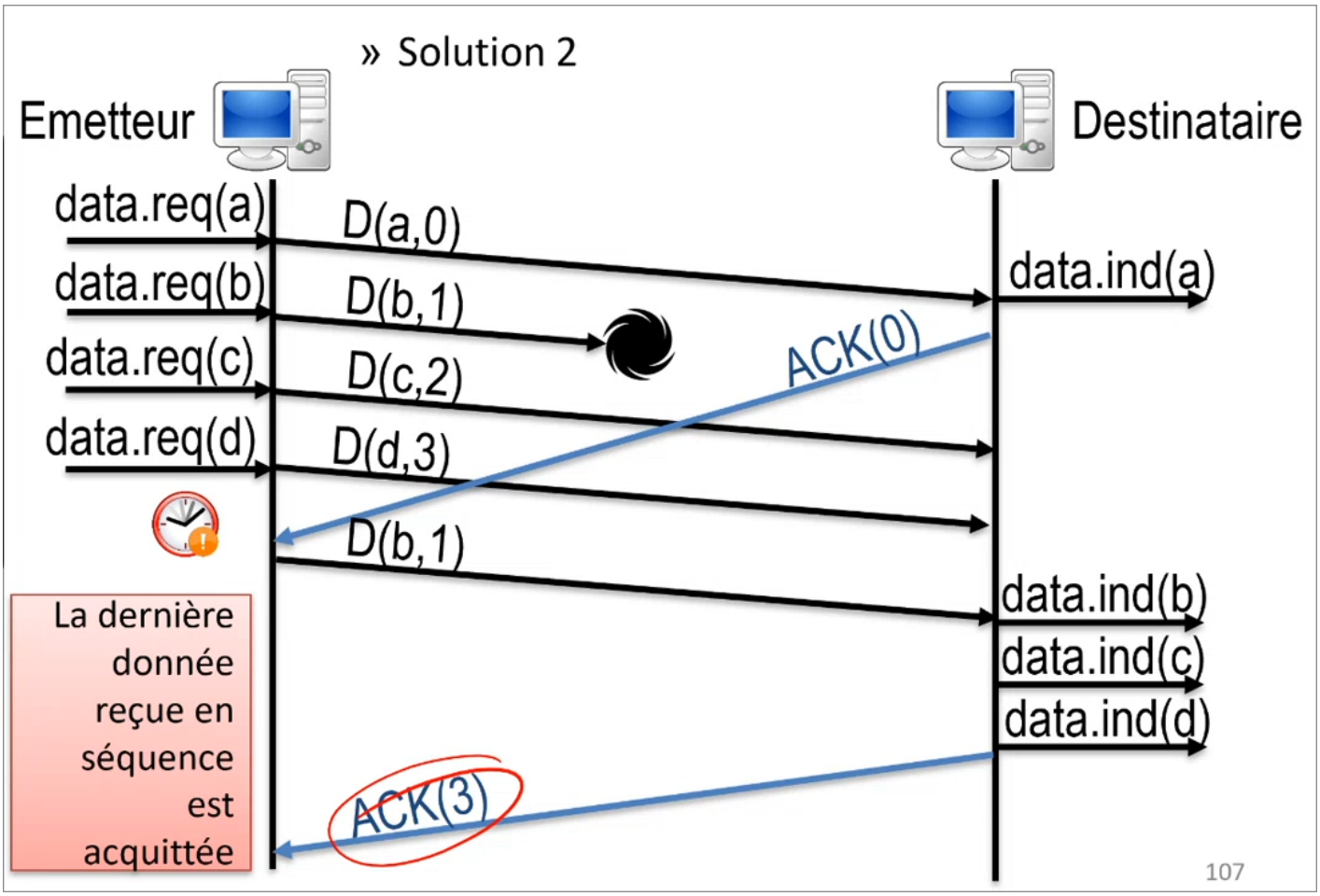


La première, c'est le **go-back-n** dans lequel le destinataire oublie tous les TPDU reçus hors séquence, ainsi s'il doit recevoir 0, 1, 2, 3, mais ne reçois pas 1, il ne va pas tenir compte de 2 et 3 et va dire à l'émetteur par un acquis : "J'ai seulement reçu le TPDU 0". L'émetteur devra alors re-envoyer les TPDU 1, 2 et 3 qui seront alors acquittés par le destinataire.

Cette méthode est avantageuse pour le destinataire, car il n'a pas besoin de retenir les TPDU hors-séquence, mais peu avantageuse pour l'émetteur qui doit tout réenvoyer.



La deuxième méthode est le **Selective Repeat** (plus courante aujourd'hui) qui consiste à garder en mémoire les données hors séquence, si on reprend l'exemple précédent, si on attend de recevoir 0, 1, 2 et 3 et que l'on ne reçoit pas 1, alors on acquitte 0 qui a bien été reçu, l'émetteur envois alors la donnée suivate, 1. Le destinataire va ensuite acquitter tous les autres paquets reçus (1, 2 et 3) qui ne seront donc pas ré-envoyés.



Pour ne pas avoir à acquitter tout un par un, on peut également acquitter la dernière information reçue en séquence (dans ce cas 3), ce qui équivaut à acquitter 1, 2 et 3 d'un coup, ce qui est donc plus efficace.

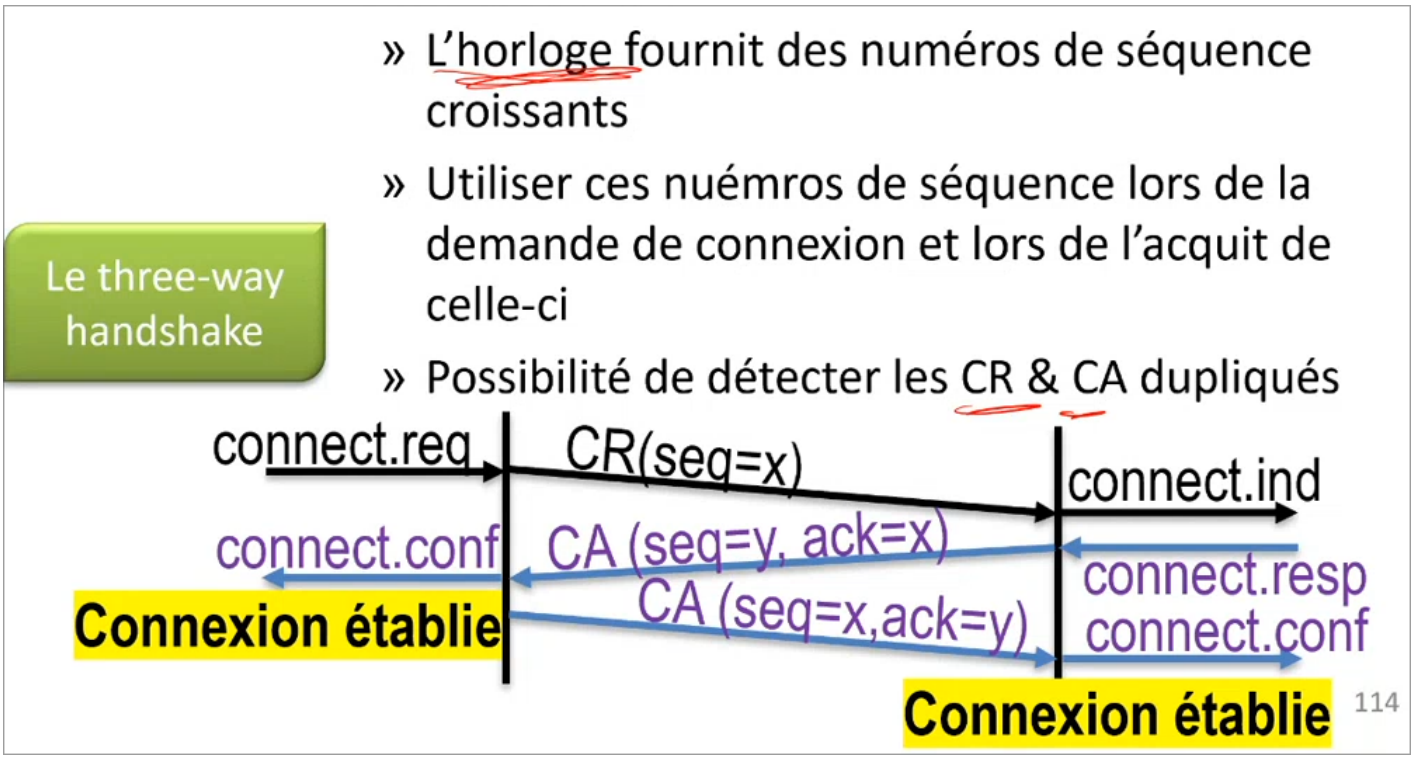
Capacité de traitement variable

Seulement, la capacité de traitement du destinataire peut varier, c'est pourquoi il va préciser dans ses acquis la taille actuelle de la fenêtre, plus la capacité de traitement du destinataire est grande, plus la fenêtre sera grande, et inversement.

À savoir qu'étant donné que les numéros de séquence sont réutilisés, il est possible d'avoir une duplication d'un acquis avec un certain numéro de séquence avec beaucoup de retard. Cela pourrait poser un problème si par hasard le numéro de séquence actuel est justement celui-là. C'est pourquoi la couche réseau (IP) s'occupe de faire un "timeout" sur les paquets, ainsi les paquets trop anciens sont juste oubliés, ce qui règle donc ce problème.

Connexion et déconnexion

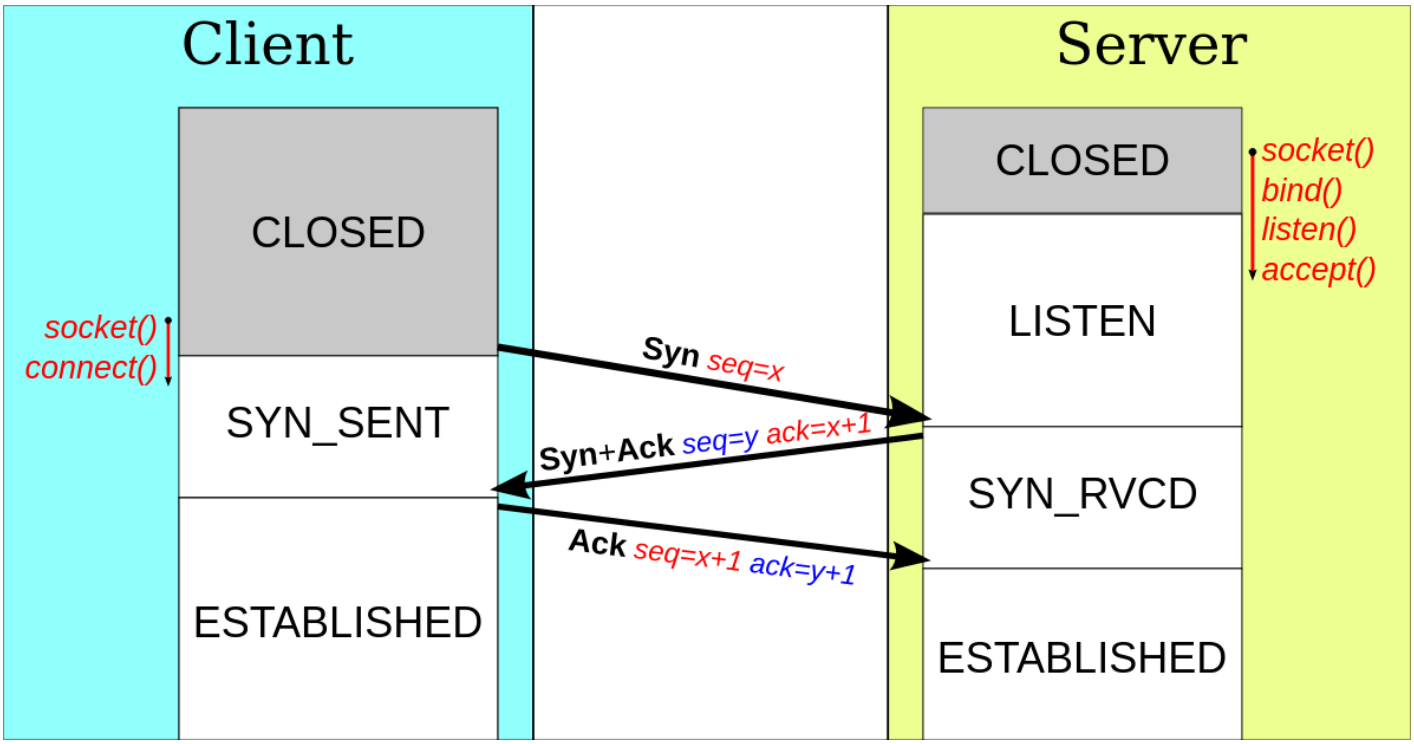
Pour pouvoir commencer à transférer des données, il faut d'abord établir une connexion pour partager des informations initiales. Pour ce faire, on utilise un **three-way handshake**.



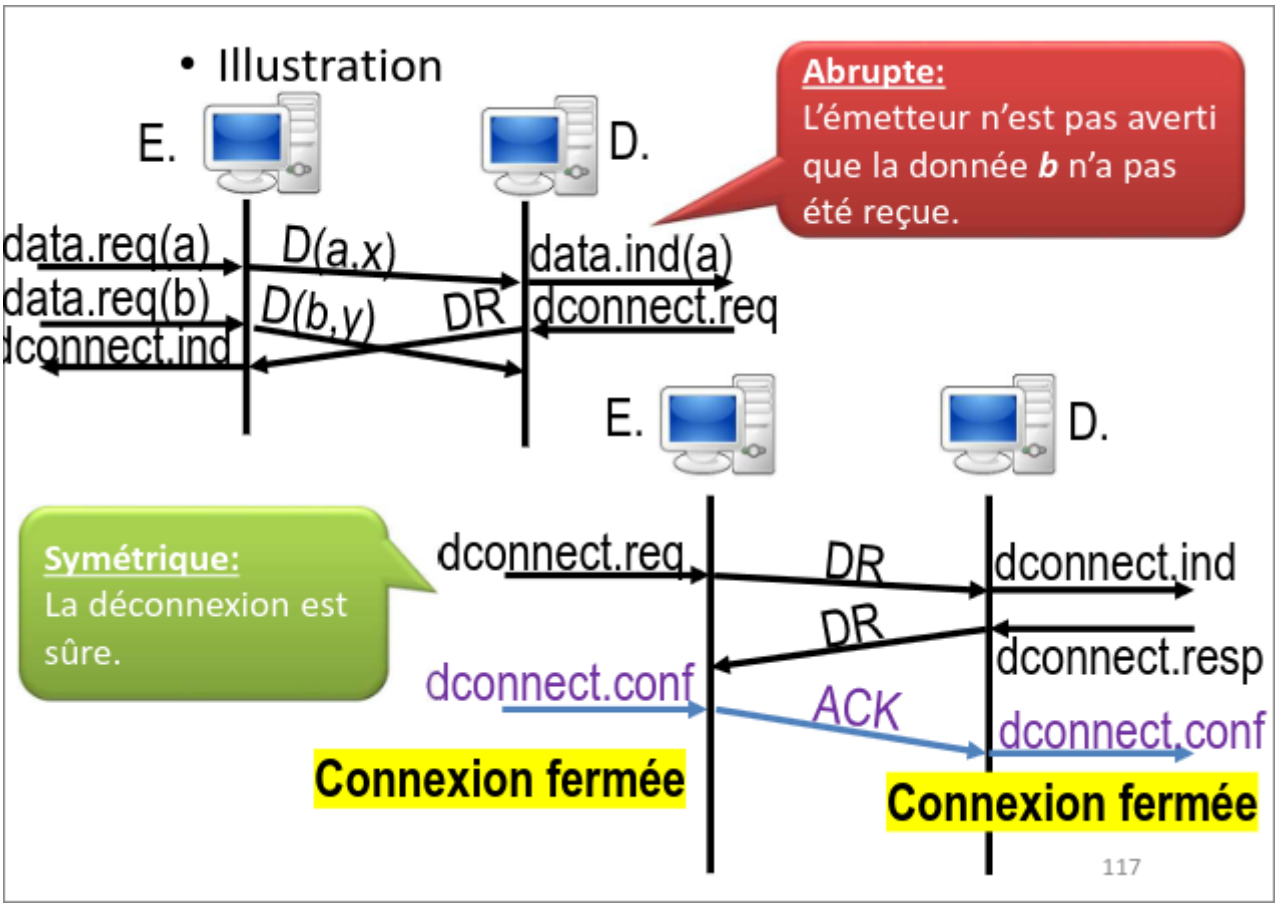
Le client va générer un numéro de séquence (x) et envoyer une demande de connexion au serveur avec ce dernier.

Le serveur va ensuite également générer un numéro de séquence (y) et acquitter la requête, la connexion est alors établie pour le client.

Enfin, le client va acquitter aussi, la connexion est alors établie pour le serveur.



Si une requête est dupliquée, le serveur va envoyer un acquis, mais le client répondra par un `REJECT` pour indiquer que la connexion est refusée, car il n'a pas fait de requête.



Pour ce qui est de la déconnexion, elle peut se faire soit de manière **abrupte**, c'est-à-dire que l'un des deux indique à l'autre "je me casse" et se déconnecte. Le problème, c'est que des données peuvent alors être perdues ou perdre l'information sur la déconnexion.

L'autre méthode est de se déconnecter de manière **symétrique**, autrement dit de manière similaire au three-way handshake. `A` envois à `B` une requête de déconnexion, `B` envois à `A` une requête de déconnexion, `A` acquitte la requête à `B` et se déconnecte (et B fait de même).

Communication bidirectionnelle

Souvent, il arrive que le client et le serveur doivent tous les deux transférer des données, ce qui complique donc un peu les choses.

Ainsi, on peut soit ouvrir deux connexions (une pour client → serveur et une pour serveur → client) mais cela ajoute donc beaucoup de trafic de contrôle et ralenti les choses.

Sinon, on peut utiliser le **piggyback** qui consiste à fusionner les TPDU de contrôle (acquis) et les TPDU de réponse en un seul TPDU ce qui diminue drastiquement donc la quantité de trafic de contrôle.

Implémentation de TCP

Voir sur la page suivante pour la description de l'implémentation du transfert fiable dans le protocole TCP.

TCP : Implémentation du transfert fiable

TCP est un protocole qui implémente le transfert fiable dont on a parlé juste avant. Il comprend 3 phases,

- La phase de connexion qui utilise un three-way handshake
- Le transfert d'informations en utilisant des acquits comme vu précédemment
- La fermeture de connexion

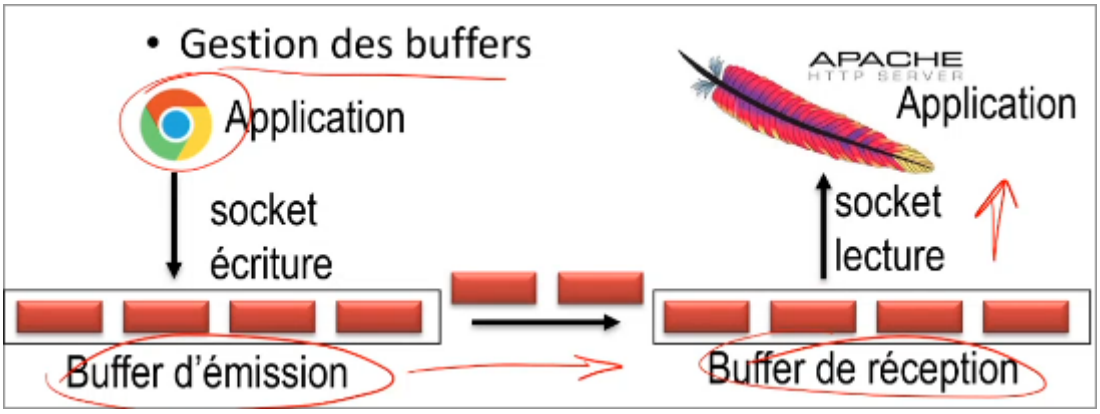
TCP fonctionne également en **unicast**, c'est-à-dire d'un destinataire à un autre et pas à un groupe de destinataire (pas multicast).

Une connexion dans TCP est identifié par quatre informations, le port source, le port de destination, l'IP source et l'IP destination.

TCP définit aussi un MSS (Maximum Segment Size) qui indique la taille maximum des données qui peuvent être dans un TPDU, cela dépends souvent de la couche réseau et de liaison utilisée, par exemple 1500 octets pour Ethernet.

Il y a par ailleurs une extension à TCP qui est utilisée dans la plupart des systèmes d'exploitations qui est le multipath TCP qui consiste à utiliser plusieurs accès réseaux simultanés pour transférer des données plus rapidement (par exemple utiliser la 4G et le Wifi en même temps).

Pour ne pas tout le temps envoyer des choses sur le réseau en permanence, TCP utilise des buffers. Ainsi, lorsque qu'une application veut envoyer des TPDU, il les place dans un buffer d'écriture, une fois plein, les informations sont envoyées. Lors de la réception, les données sont placées dans un buffer de destination, une fois toutes les données reçues, les données sont envoyées à l'application.



TPDU TCP

Un TPDU TCP est composé de plusieurs informations :

- Port source
- Port destination
- Numéro de séquence
- Numéro d'acquittement
- Taille de l'entête
- Des indicateurs (ACK pour acquis, SYN pour demander une connexion, FIN pour terminer une connexion, etc)
- La taille de la fenêtre glissante
- Le "checksum" pour vérifier les données
- Le pointeur de donnée urgente et les options, qui ne servent à rien ou sont facultatives
- Données

Ce système permet donc de faire de la communication bidirectionnelle étant donné qu'il est possible de mettre des données et un acquit dans un même TPDU. On considère en TCP que l'acquit est toujours le prochain numéro de séquence attendu.

Temporisateur TCP

TCP doit également définir un temporisateur, c'est-à-dire mettre un "timeout" au bout duquel, si aucun acquit a été reçu, le(s) TPDU sont considérés comme perdu et doivent être renvoyé.

Ce délai doit donc être plus grand que le RTT (Round-Trip Time) qui est le temps de faire un aller-retour entre un émetteur et une destination. Aussi, si le RTT est très variable, le délai du temporisateur sera plus grand.

La valeur du temporisateur est alors $ RTTmoyen + 4 * RTTvariation $.

Envois des acquits

Il y a plusieurs cas différents d'envois d'acquits en TCP, lors de la réception d'un nouveau TPDU :

- SI on reçoit un TPDU ayant le numéro de séquence attendu ET que toutes les données ont été acquittées jusque-là, ALORS on attend jusqu'à 500 ms ET si aucun TPDU n'arrive au bout de ce délai, ALORS on acquitte le TPDU

- SI on reçoit un TPDU ayant le numéro de séquence attendu MAIS que toutes les données n'ont pas été acquittées jusque-là, ALORS on envoie un acquit pour tous
- SI on reçoit un TPDU ayant un numéro de séquence plus grand que prévu, ALORS on re-duplique l'acquit précédent pour demander le TPDU manquant
- SI on reçoit un TPDU couvrant un trou dans la séquence, ALORS on envoie acquit pour demander le prochain TPDU de la séquence

Fast retransmit

Une amélioration de TCP est le **Fast Retransmit** qui permet de ne pas avoir à attendre le timeout du temporisateur pour renvoyer un ou des TPDU.

Pour ce faire, le destinataire va envoyer des acquits pour tous les TPDU de la séquence, même ceux qui sont perdus. Ensuite, le destinataire va envoyer trois acquits identiques pour le TPDU perdu.

La réception de ces trois acquits est vue comme une demande de ré-envois de ces données par l'émetteur qui n'a ainsi pas besoin d'utiliser son temporisateur dans ce cas. Cela permet donc d'aller beaucoup plus vite.

Gestion des pertes avec TCP

TCP retient uniquement les TPDU qui arrivent en séquence (donc avec les numéros de séquence attendus à chaque fois).

Mais TCP sauvegarde tout de même les TPDU qui arrive hors séquence afin de ne pas avoir à les redemander plus tard.

La fast-retransmit vu plus tôt permet à TCP d'aller plus vite pour demander les données perdues à l'émetteur.

Algorithme de Nagle pour l'émission de TPDU

Il serait fort peu pratique d'émettre des TPDU pour chaque petite donnée, car avoir beaucoup de petit TPDU causerait des problèmes de congestion du réseau.

L'idée est alors de combiner plusieurs TPDU dans un seul, plus gros TPDU. Le fonctionnement de cet algorithme est celui-ci :

1. Le premier octet est envoyé immédiatement
2. Tant que l'accusé de réception n'est pas reçu, on accumule toutes les données suivantes dans un seul TPDU (tampon ou buffer). Lorsque l'acquittent arrive, on envoie le TPDU.
3. On répète la deuxième étape.

Contrôle de flux et de la fenêtre

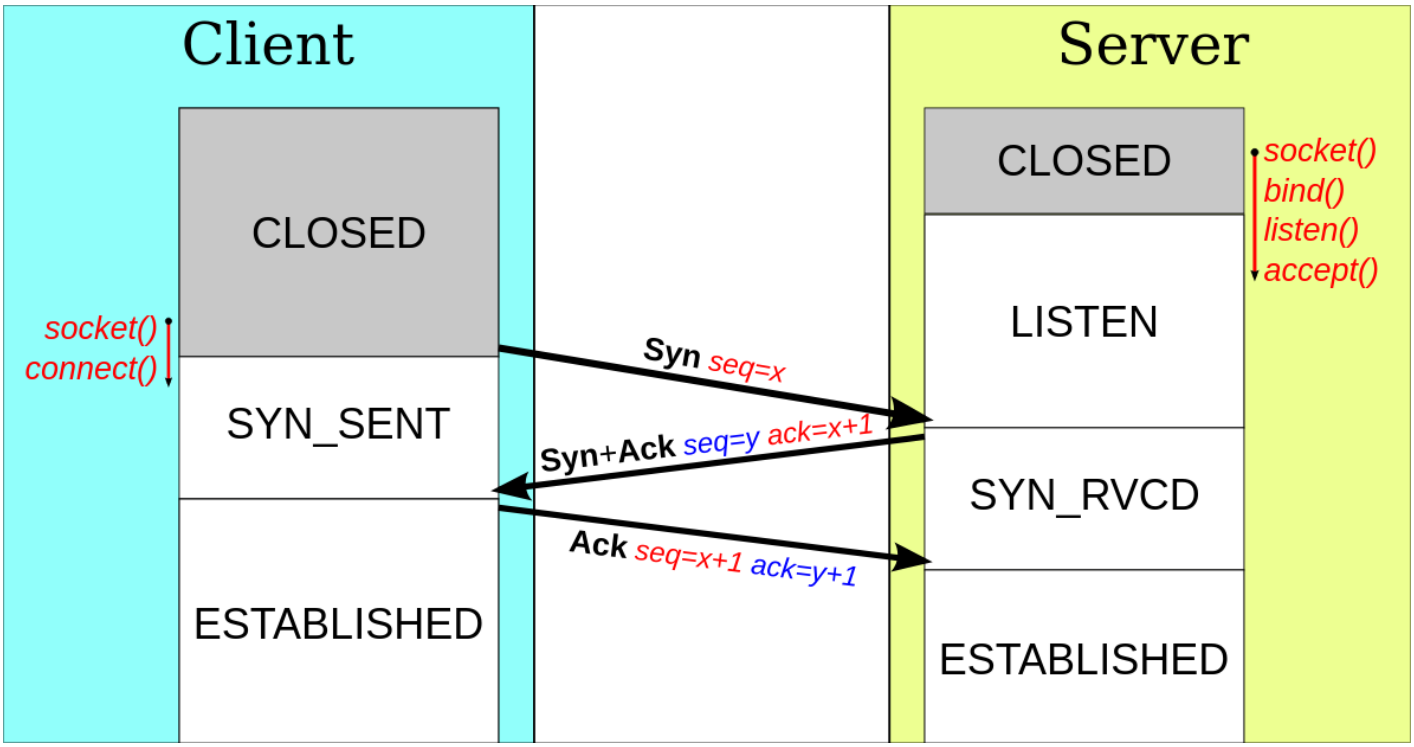
Le destinataire doit indiquer la taille de sa fenêtre (qui symbolise sa capacité de traitement) au fur et à mesure afin de ne pas être submergé de données.

L'émetteur de son côté est obligé d'attendre la réception d'un acquit lorsque la fenêtre est vide avant de recommencer à transmettre.

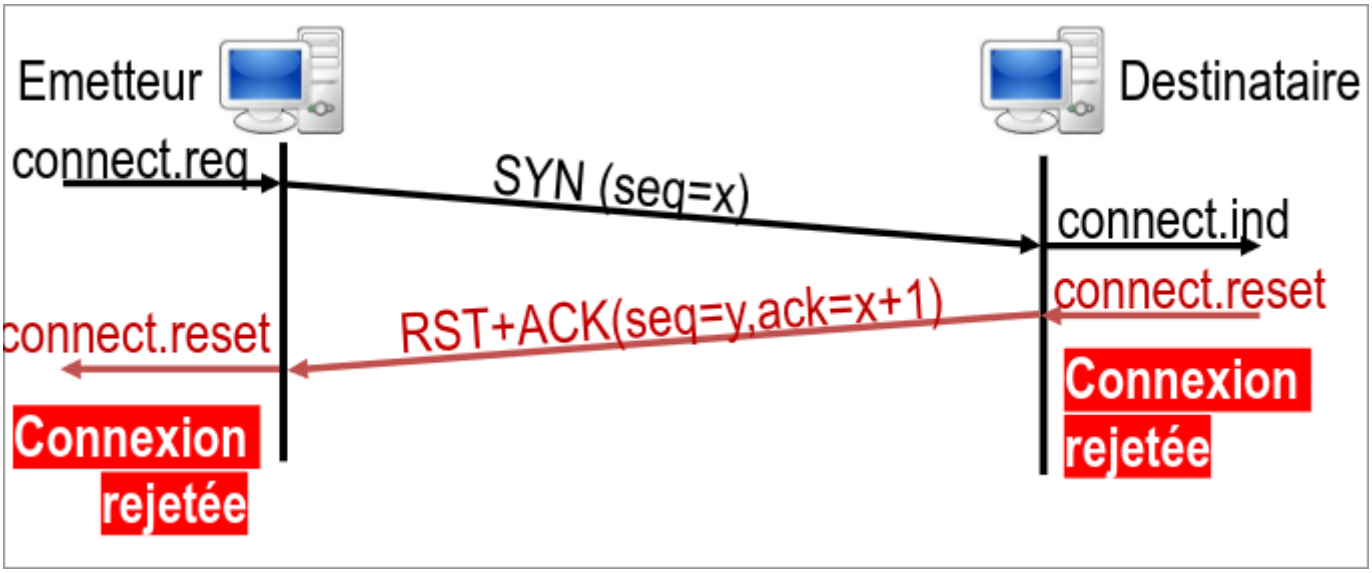
Connexion et déconnexion TCP

Une requête de connexion en TCP se fait avec le flag `SYN`. Le serveur peut ensuite soit accepter en utilisant les flags `SYN` et `ACK`, ou refuser avec `RST` et `ACK`. Si la connexion est acceptée par le serveur, le client envois un `ACK` pour signaler qu'il est prêt à commencer l'échange de données, ou un `RST` et `ACK` pour refuser et annuler la connexion.

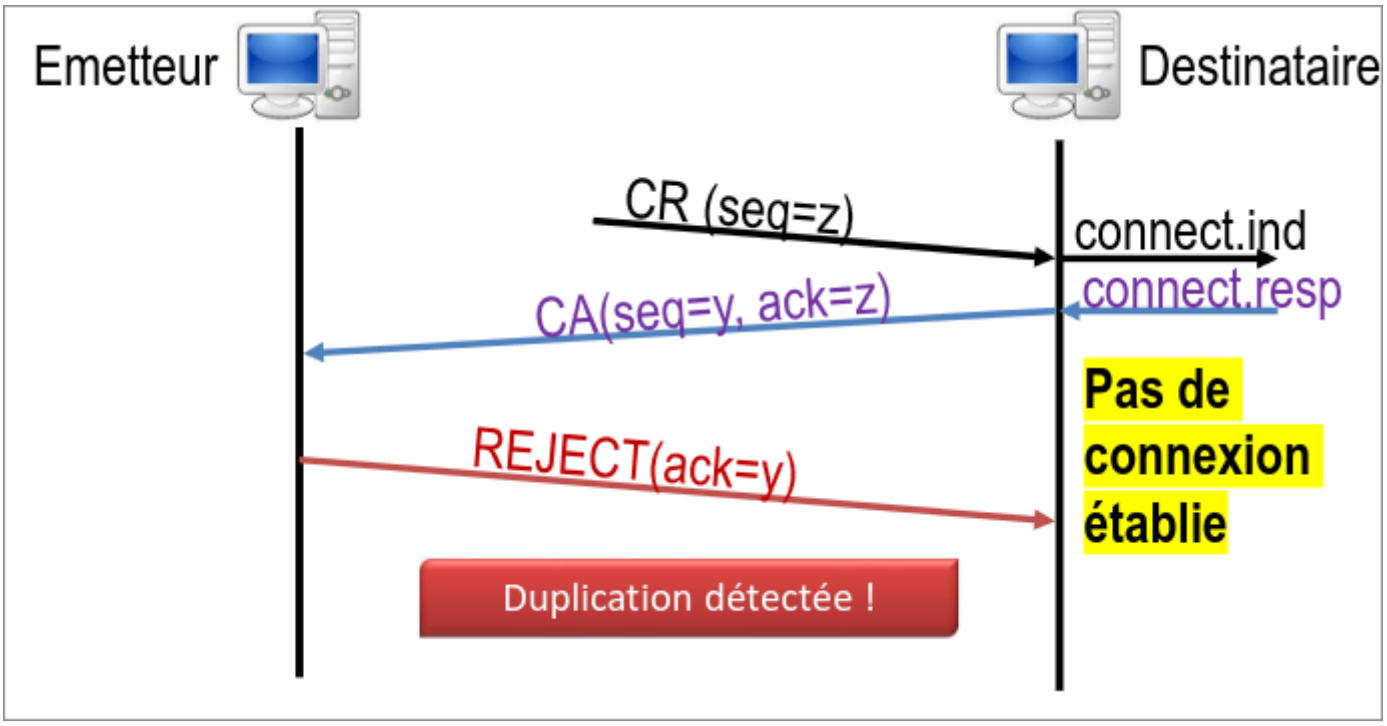
Voici ce qu'il se passe lorsque la connexion est acceptée par le serveur :



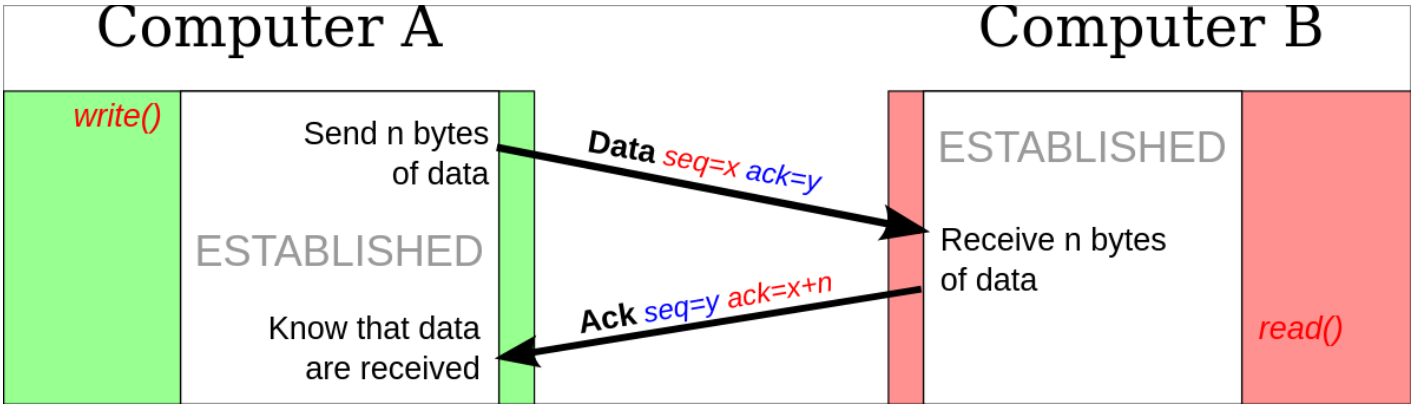
Voici ce qu'il se passe lorsque la connexion est refusée par le serveur :



Et voici ce qu'il se passe lorsque la connexion est invalide et est refusée par le client (compter que REJECT serait en TCP `SYN+ACK`) :

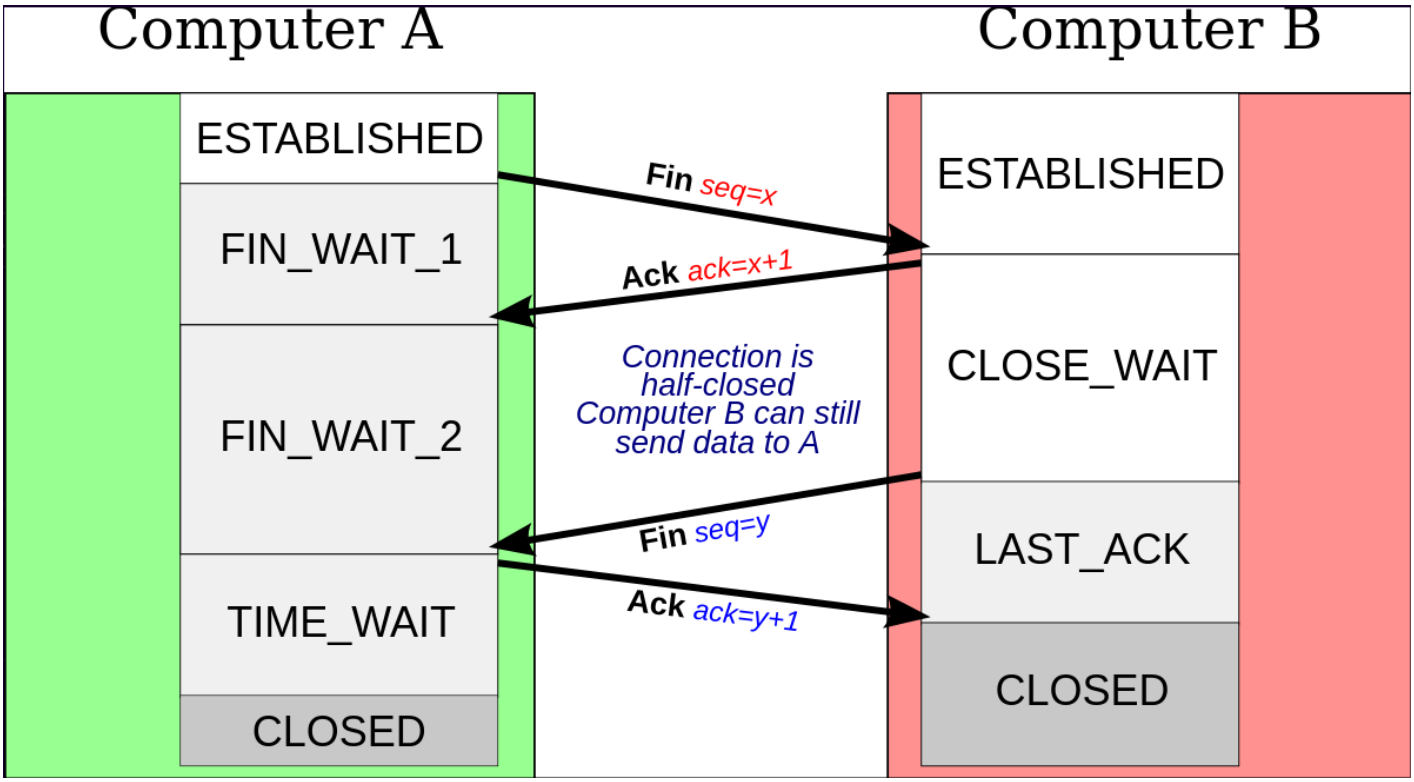


Une fois la connexion établie, un échange de donnée peut donc s'opérer :



Une fois l'échange de donnée terminé, la connexion peut être fermée correctement en utilisant une méthode similaire au three-way handshake utilisé pour la connexion. On envoie un TPDU avec un flag `FIN` pour demander la déconnexion, l'autre partie renvois donc un `ACK` transfert également quelques dernières données, puis envois un `FIN` à son tour qui a une réponse par un `ACK`. La connexion est alors fermée.

Il est aussi possible d'avoir une connexion abrupte, si un `FIN` est envoyé sans attendre d'acquit. Vous pouvez avoir plus de détail en lisant la partie plus tôt sur la déconnexion dans le transfert fiable.



Contrôle de la congestion

Tous les réseaux sur internet ne sont pas égaux, certains sont donc beaucoup plus lents que d'autres. Pour éviter de surcharger ces réseaux avec beaucoup de données, il est donc important de mettre en place un système permettant de limiter cette congestion.

Pour cela, l'émetteur va retenir une "fenêtre de congestion" (qui n'a rien à voir avec la fenêtre glissante).

Cette fenêtre de congestion indique la quantité de donnée qui peut être transmise, elle est mesurée en MSS (maximum segment size, c'est-à-dire la taille maximale d'un TPDU). Au départ, elle est à 1, et à chaque `ACK` reçu, elle augmente de 1. Cette donnée va donc grandir de manière exponentielle. Cette phase est appelée le **slow-start**.

Ensuite, une fois que la fenêtre atteint un certain maximum prédéfini, elle va ensuite grandir de manière linéaire en augmentant de 1 à chaque RTT (Round-Trip Time, donc il faudra attendre que toutes les données envoyées soient acquittées). Cette phase est appelée le **AIMD** (Adaptative Increase Multiplicative Decrease).

Ce qui est fait lorsque des données sont perdues dépends de comment on sait que les données sont perdues :

- Soit, on reçoit un **Fast-Retransmit** (3 acquits dupliqués), la congestion est alors définie comme légère et on définit le maximum de la fenêtre à la valeur actuelle de la fenêtre divisée par 2. Et on définit la valeur de la fenêtre au seuil. On reprend alors en mode

linéaire (AIMD).
- Soit, il y a un **timeout du temporisateur**, la congestion est alors définie comme forte et on définit le maximum de la fenêtre à la valeur actuelle de la fenêtre divisée par 2. Et on met la valeur de la fenêtre à un et on recommence en slow-start (exponentiel).

Couche internet (routage)

La couche internet qui soutient la couche de transport sert à faire acheminer les informations d'une machine source vers une machine destination. Cela se fait cependant sans garantie de fiabilité, c'est pour cela que le protocole TCP est nécessaire.

Certains systèmes, principalement ceux qui transitaient sur le réseau téléphonique, nécessitait une phase d'ouverture de connexion. Ce n'est cependant plus obligatoire aujourd'hui.

Identification des machines

Les machines sont identifiées par des adresses sur 32 bits (IPv4) ou 128 bits (IPv6).

Ces adresses sont écrites généralement sous forme décimale pointée. On regroupe donc les adresses par octets (8 bits) que l'on représente sous forme décimale (pour l'IPv4).

Par exemple :

```
11000001 10111110 01000000 01111100
193    .190    .64     .124
```

En IPv6, les adresses sont codées sur 128 bits, sont représentées en hexadécimal par blocs de 16 bits séparés par `:`. On peut également abréger les `0` consécutifs en utilisant `::`

Ainsi `2001:0bc8:38eb:fe10:0000:0000:0000:0011` devient simplement `2001:0bc8:38eb:fe10::11`

Sous-réseaux

Pour pouvoir se connecter directement à une autre machine, il faut que cette dernière se situe dans le même **sous-réseau**. Chaque sous-réseau est lui-même identifié par une adresse IPv4 particulière.

Pour savoir si machines sont directement connectées sur un sous-réseau donné. Il faut appliquer un **masque de sous-réseau** sur l'**IP du réseau**.

```
IP RES : 11000000.10101000.00000001.00000010 - 192.168.1.2
MASQUE : 11111111.11111111.11111111.00000000 - 255.255.255.0
         -----------------------------------
PREFIX : 11000000.10101000.00000001.00000000 - 192.168.1.0
```

Cela signifie qu'il y a 256 adresses possibles ($2^{(32 - \text{nombre de 1 dans le masque})}$), qui auront toutes un certain préfixe défini plus tôt. Par exemple, 192.168.1.1, 192.168.1.5, 192.168.1.255 sont toutes des adresses faisant partie d'un seul et même sous-réseau 192.168.1.2.

Note : le sous-réseau peut plus simplement être indiqué via la notation <IP RÉSEAU>/<NOMBRE DE 1 DU MASQUE>. Par exemple, plus tôt, on avait un sous-réseau 192.168.1.2/24. Il est aussi bon de noter qu'il y a également une adresse de **broadcast** qui permet de communiquer des paquets à tout le monde dans le réseau.

Routage

La plupart du temps, on communique avec des machines qui sont en dehors de notre réseau direct. Il faut donc connecter les routeurs entre eux et utiliser des algorithmes pour pouvoir acheminer les informations là où il faut.

Routeur

Pour transférer des paquets (unité d'information dans la couche internet), ces derniers transitent par des **routeurs**. Ces derniers sont des relais au niveau de la couche réseau et ont pour but de trouver le meilleur chemin pour faire transiter l'information.

Ainsi, il peut interconnecter des réseaux de natures différentes, le routeur (box) chez vous peut connecter votre réseau sur le réseau téléphonique (xDSL) ou de télédistribution (coaxial).

Le routeur a aussi plusieurs interfaces réseau avec lesquelles il communique, tel que le Wifi, une connexion au réseau de l'ISP (Internet Service Provider tel que Proximus), etc.

Modèles de routage

Il existe deux modèles de routage de l'information :

	Datagrammes	Circuits virtuels
Configuration du connexion	Pas obligatoire	Obligatoire
Adressage	Le paquet contient les adresses source et de destination complètes	Le paquet contient l'identifiant du circuit virtuel.
Informations d'état	Nul autre que la table de routeur contenant le réseau de destination	Chaque numéro de circuit virtuel entré dans la table de configuration, utilisé pour le routage.
Routage	Paquets acheminés indépendamment	Route établie à la configuration, tous les paquets suivent la même route.
Effet d'une panne de routeur	Uniquement sur les paquets perdus lors d'un crash	Tous les circuits virtuels passant par un routeur défaillant sont interrompus.
Contrôle de congestion	Difficile car tous les paquets routés indépendamment.	Simple en pré-affectant suffisamment de buffers à chaque circuit virtuel lors de la configuration, puisque le nombre maximal de circuits est fixe.

Le mode **circuit virtuel** qui va sur base d'une information donnée par l'émetteur déterminer le chemin entre la source et la destination (donc avec phase d'ouverture), ensuite va transférer les paquets sur cette route, une fois terminé une des entités annonce une déconnexion et interrompt le circuit.

Ce mode, bien que simple à comprendre (c'est simplement un circuit), est très peu pratique pour de grands réseaux tels qu'internet, car chaque routeur devrait connaitre tous les autres routeurs et si un routeur ne fonctionne plus, cela pourrait mettre en péril une grande partie du réseau.

C'est pourquoi la méthode utilisée par internet est le mode **datagramme** où chaque paquet mentionne l'adresse de destination, sur base de cette information les routeurs orientent les paquets vers la destination avec une **table de routage**. Les paquets sont donc transférés individuellement, il n'y a pas besoin de phase de connexion/configuration et le réseau peut grandir beaucoup plus facilement. Il faut toute fois noter qu'il est possible que plusieurs paquets ayant la même source et la même destination peuvent ne pas prendre le même chemin.

Routage statique

Les routes vers les différentes destinations sont définies manuellement par l'administrateur·ice.

Cette méthode est très utilisée, car conceptuellement simple, mais ingérable pour les réseaux de taille importante et variable.

Routage dynamique global

Le routage dynamique global consiste à faire en sorte que chaque routeur dispose d'une carte complète du réseau et déterminer manuellement les chemins les plus courts vers les destinations.

Algorithme état liaisons

Au début, les routeurs échanges des informations entre eux pour pouvoir aider à construire une carte du réseau.

Chaque routeur connait au départ les routes auquel il est relié et les couts (les couts sont calculés sur base de la vitesse de la connexion) associés.

Les routeurs publient ces informations aux autres routeurs afin de construire la carte du réseau. Chaque route est échangée dans un LSP (Link State Packet) qui comprend l'identification du routeur qui annonce ainsi que la destination et son cout associé.

Chaque LSP est à un numéro de séquence associé qui est incrémenté par la source à chaque nouvel envoi. Ainsi, les routeurs peuvent se souvenir des derniers numéros de séquence pour chaque LSP afin de ne pas renvoyer les mêmes LSP en boucle dans le réseau.

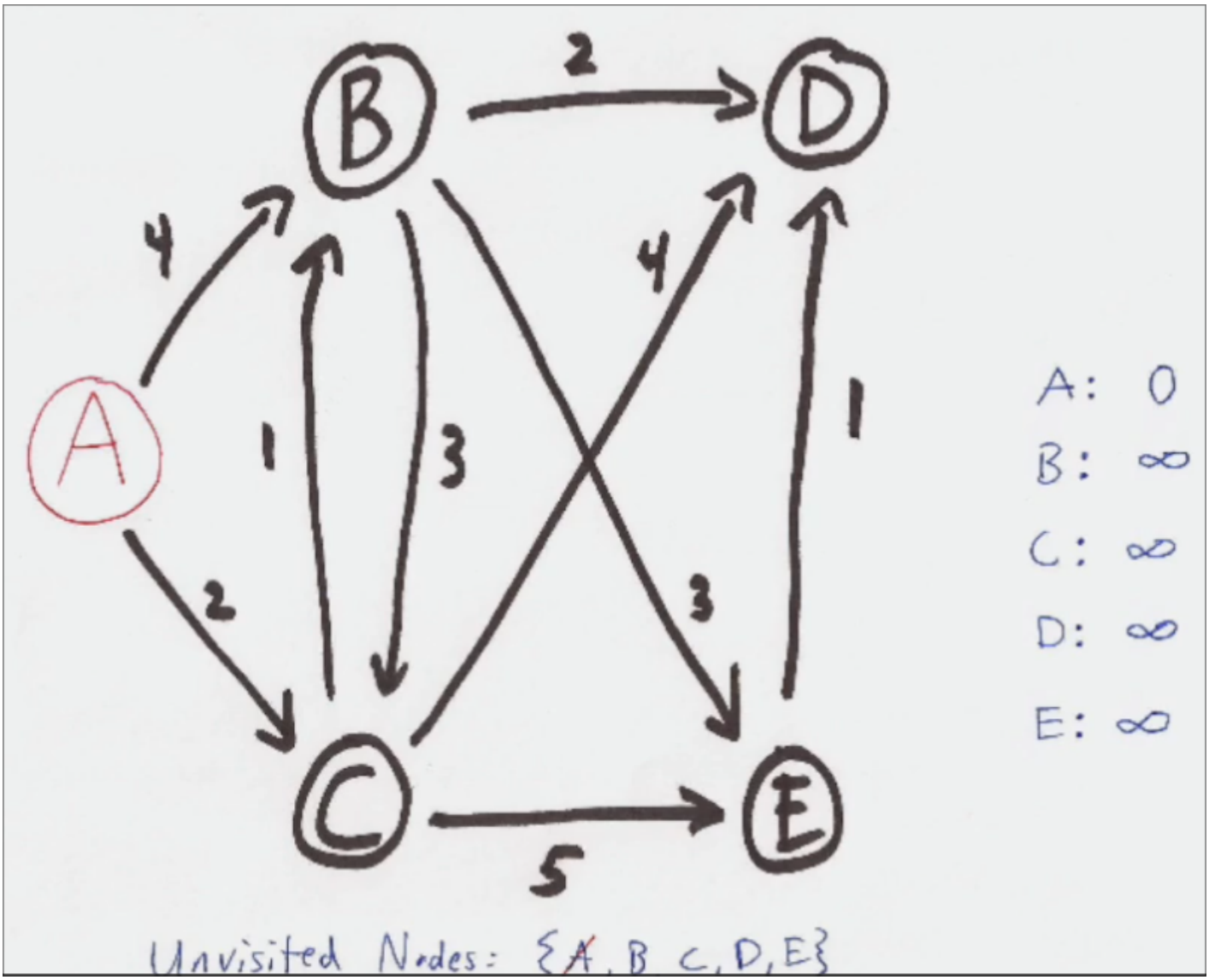
Une fois que la carte du réseau est établie sur base de ces envois, pour chaque requête, on établit le chemin le plus court entre le routeur actuel et la destination. Pour ce faire, on utilise l'**algorithme de Dijkstra**.

L'algorithme de Dijkstra

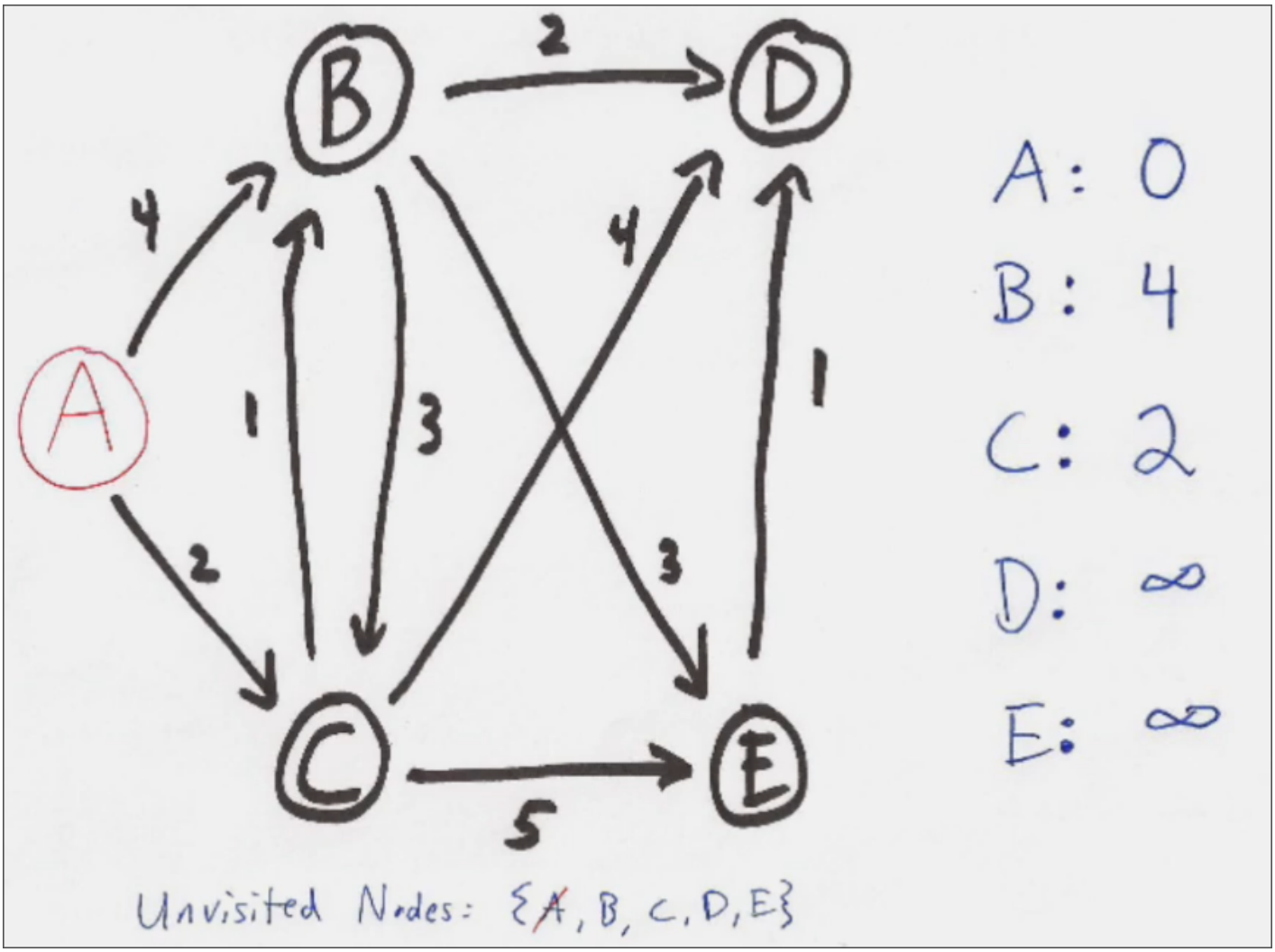
> Les images et les explications de cet algorithme viennent de cette vidéo.

Au départ, on indique que le cout pour atteindre le routeur actuel est 0 et ceux pour atteindre les autres routeurs est infinie, car on n'a pas encore calculé le cout.

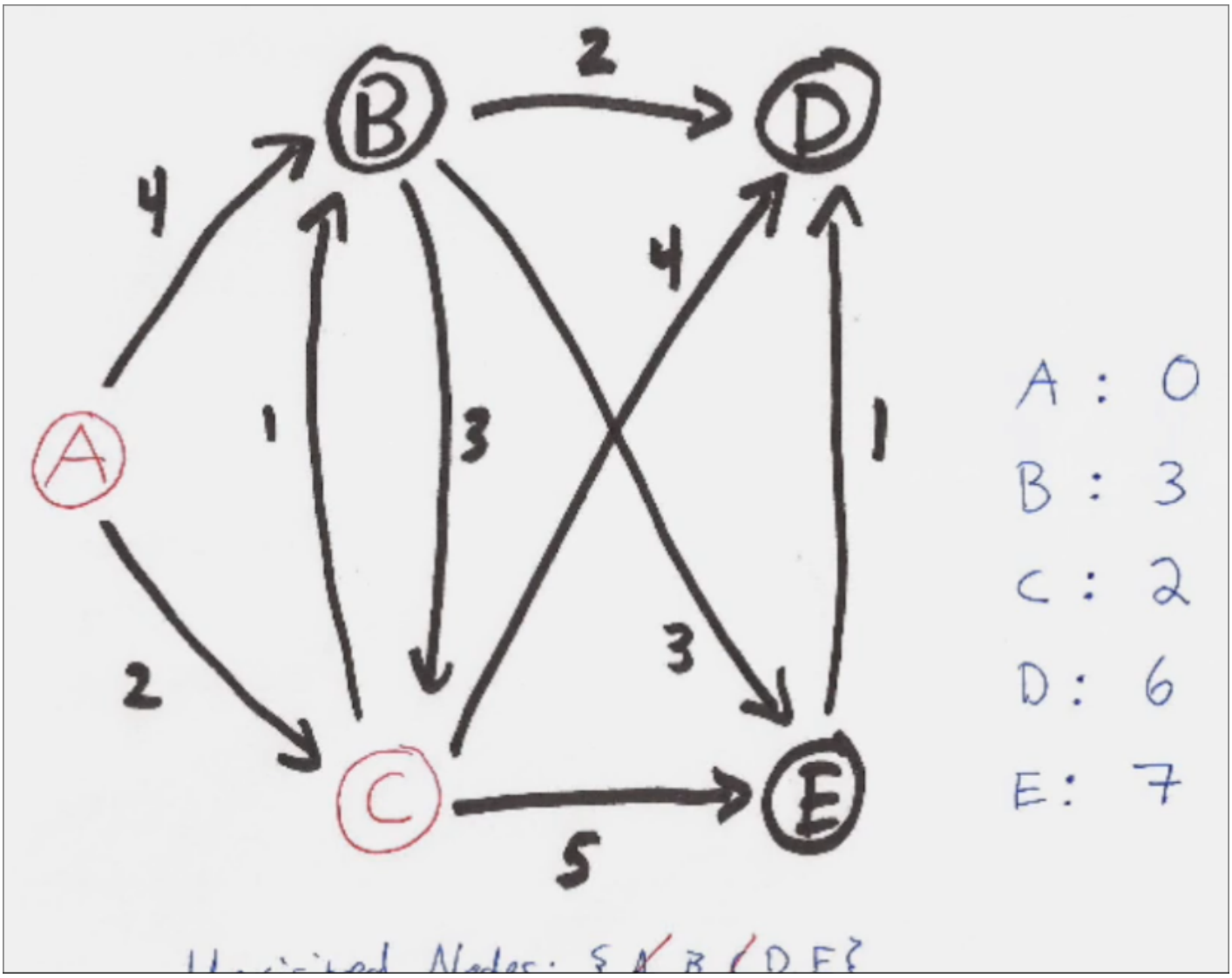
On garde également une liste des routeurs à "visister".



Ensuite, on peut compléter le tableau des couts en indiquant les couts pour atteindre les routeurs voisins du routeur actuel.

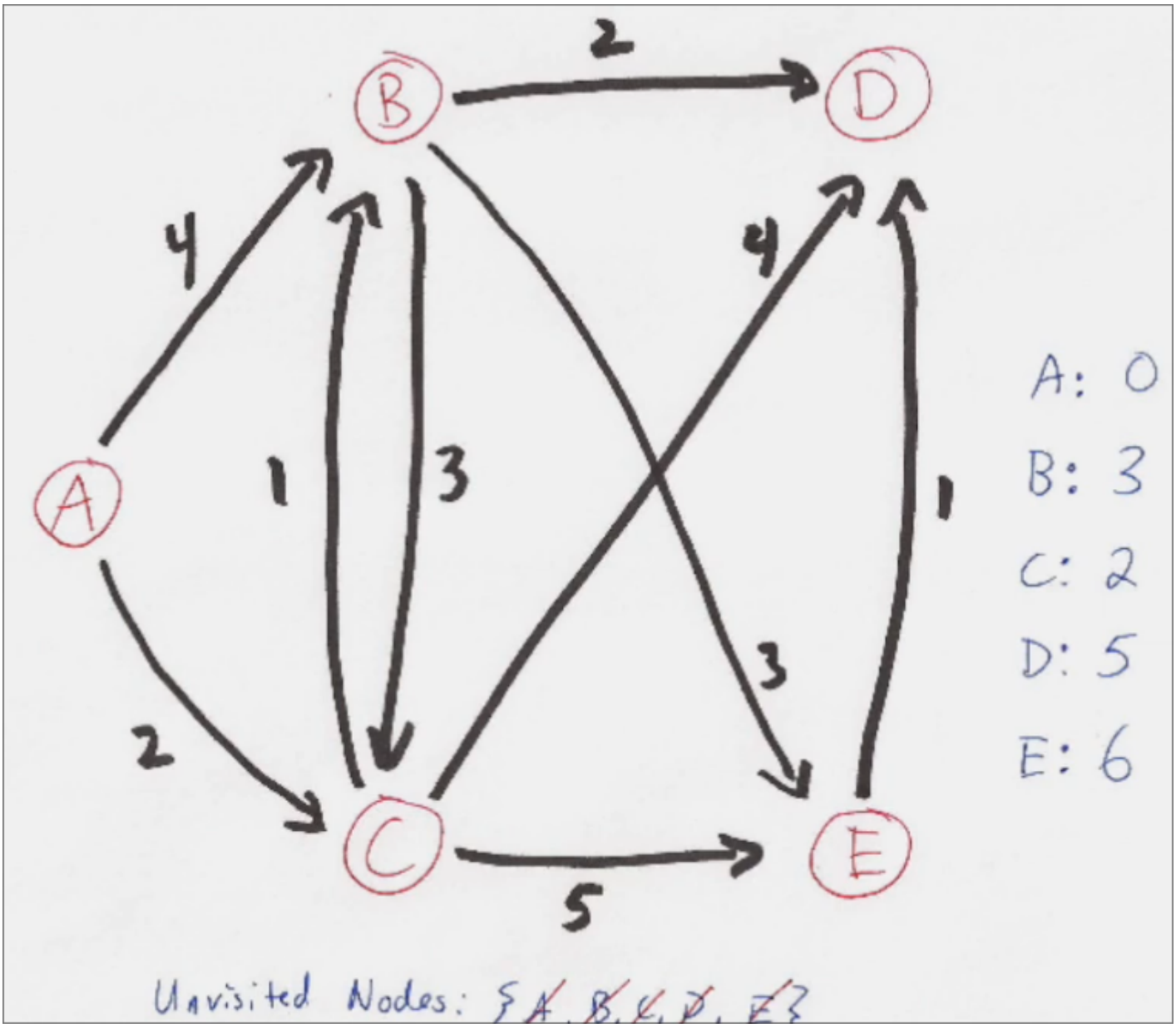


On peut ensuite faire la même chose depuis le routeur ayant le cout le plus bas, dans ce cas-ci, le routeur C.

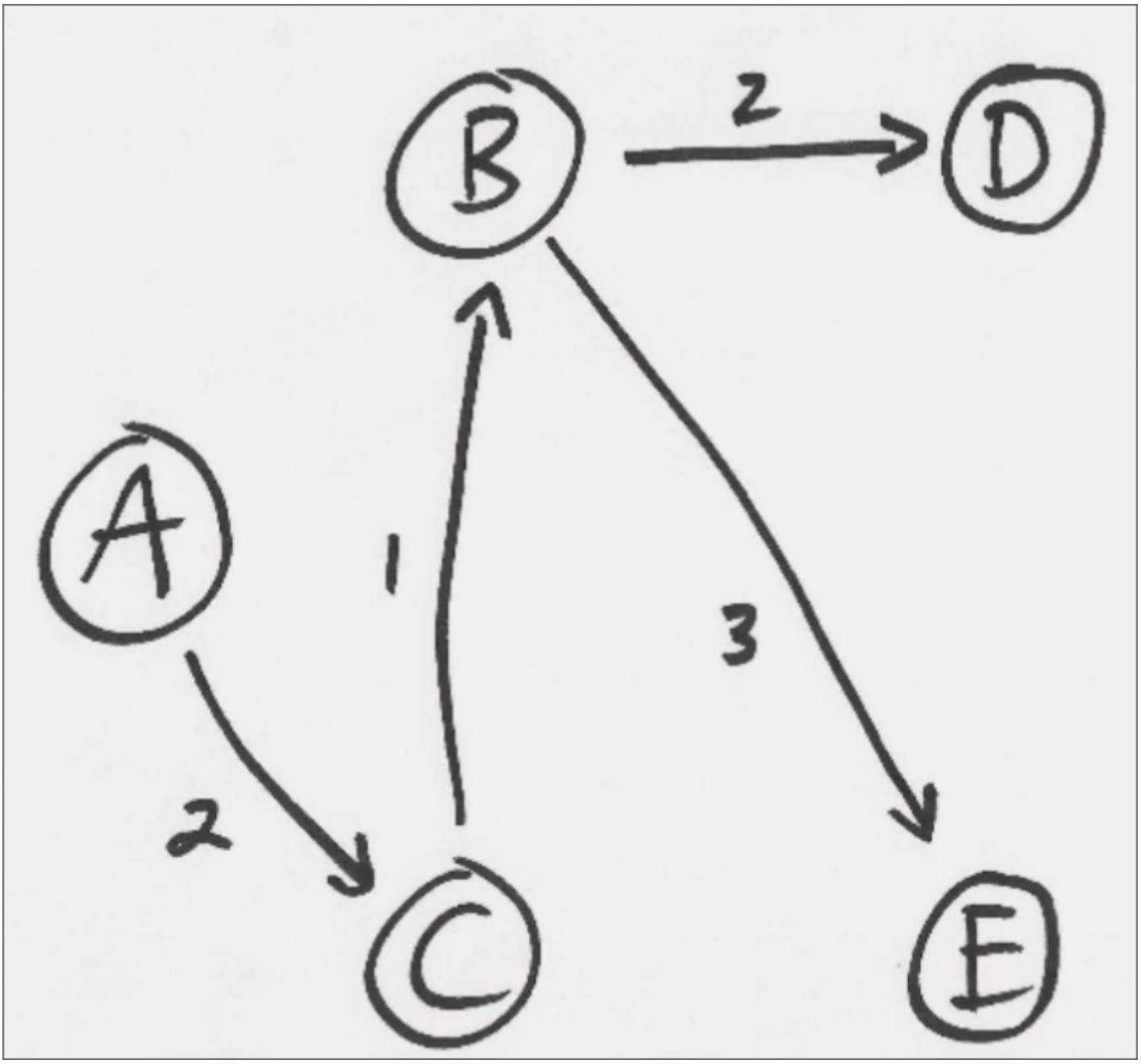


Depuis C, on va ainsi regarder pour chaque routeur voisin de C, quel est le cout pour y accéder. Si le cout total (donc le cout pour atteindre C + le cout pour atteindre le routeur en question) est plus petit que le cout noté précédemment, on met ainsi le cout à jour.

On continue alors le processus jusqu'à avoir fait cette vérification depuis tous les routeurs à visiter.



Une fois cela fait pour tous les routeurs, on peut alors établir quel est le chemin le plus court pour atteindre chaque routeur depuis le routeur courant (A) :



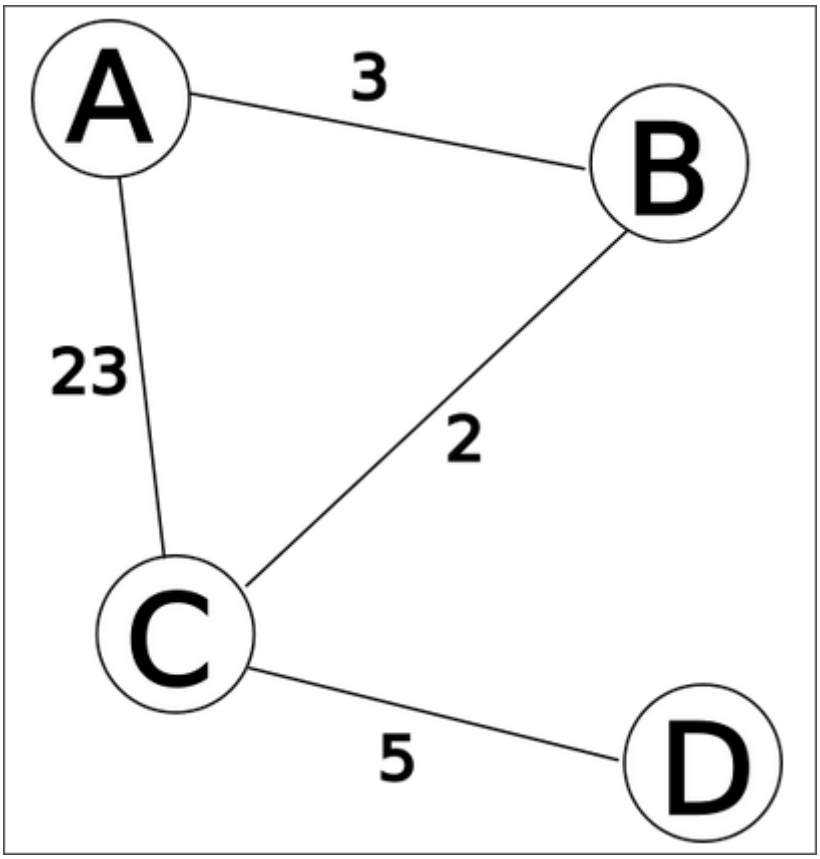
Sur base de cela, nous pouvons donc créer la table de routage. Dans le cas de A, c'est assez simple, car tous les paquets seront envoyés à C.

Routage dynamique décentralisé

À la place d'avoir une carte complète du réseau, on garde simplement une matrice des distances entre les différents routeurs voisins. Il est ainsi possible de savoir, pour chaque routeur cible, quel routeur doit être utilisé (car plus rapide). Ceci a un grand avantage pour de grands réseaux parce qu'il n'est pas nécessaire de connaitre la carte complète du réseau pour pouvoir envoyer des paquets.

Algorithme vecteur de distance

Imaginons le réseau de routeurs suivant :



Au départ, chaque routeur établit une matrice indiquant le cout pour aller à chacun de leur voisin :

T=0

from A	via A	via B	via C	via D
to A				
to B		3		
to C			23	
to D				

from B	via A	via B	via C	via D
to A	3			
to B				
to C			2	
to D				

from C	via A	via B	via C	via D
to A	23			
to B		2		
to C				
to D				5

from D	via A	via B	via C	via D
to A				
to B				
to C			5	
to D				

Ensuite, tous les routeurs envoient ces informations sur le réseau à leur voisin. Ainsi A envois sa matrice à B et C, B l'envois à A et C, C l'envoi à A, B et D et D l'envoi à C.

Sur base de cette information, chaque routeur peut recalculer les couts. Ainsi A va recalculer le cout d'aller à C sur base de la matrice de B et va remarquer qu'aller à C en passant par B est beaucoup plus rapide (3 + 2 = 5 contre 23).

Ils vont ainsi mettre à jour leurs matrices :

T=1

from A	via A	via B	via C	via D
to A				
to B		3	25	
to C		5	23	
to D			28	

from B	via A	via B	via C	via D
to A	3		25	
to B				
to C	26		2	
to D			7	

from C	via A	via B	via C	via D
to A	23	5		
to B	26	2		
to C				
to D				5

from D	via A	via B	via C	via D
to A			28	
to B			7	
to C			5	
to D				

Ensuite, le processus va se répéter jusqu'à ce que la carte de tout le monde soit complète. Ainsi A va par exemple apprendre le cout pour aller à D est seulement de 10 en passant par B (3+7 = 10).

T=3

from A	via A	via B	via C	via D
to A				
to B		3	25	
to C		5	23	
to D		10	28	

from B	via A	via B	via C	via D
to A	3		7	
to B				
to C	8		2	
to D	13		7	

from C	via A	via B	via C	via D
to A	23	5		15
to B	26	2		12
to C				
to D	33	9		5

from D	via A	via B	via C	via D
to A			10	
to B			7	
to C			5	
to D				

Une fois la table de tout le monde complétée, on peut ainsi définir la table de routage assez simplement en regardant par où il faut passer pour atteindre chaque routeur afin d'avoir la distance la plus courte.

Ainsi, la table de A indique par exemple que tous les paquets seront envoyés à B.

Lorsqu'un ou plusieurs liens avec un routeur est down, les routeurs qui le remarquent indique que le lien vers ces routeurs a un cout infini. Il propage alors cette information dans le réseau pour établir à nouveau la matrice et la table de routage de tous les autres routeurs. Cela se fait de manière assez lente ce qui fait que pendant un certain temps certaines destinations pourrait être inaccessibles.

Couche internet (protocoles IPv4 et IPv6)

IP est le protocole de base d'internet et globalement le seul protocole de la couche internet. Il a cependant deux versions majeures, la version 4 et la version 6.

IPv4 vs IPv6

La version 4 d'IP est celle qui est la plus répandue sur Internet, mais qui est toutefois limitée par le nombre d'adresses possibles, car les adresses sont codées sur 32 bits contre 128 bits pour l'IPv6.

Cela signifie que l'IPv4 à moins de cinq milliards d'adresses possibles. IPv6 a comparativement plus de $7x10^{28}$ fois plus d'adresses que l'IPv4.

Aujourd'hui toutes les adresses IPv4 ont été assignées, nous allons voir quels stratagèmes ont été utilisés pour faire en sorte que ce ne soit pas un trop gros problème. Mais il est bon de noter que le futur se trouve dans l'IPv6 et qu'il est donc très important de supporter l'IPv6.

CIDR vs classes

Historiquement, les adresses étaient définies suivant des "classes" allant de A à E. La classe A est équivalente à un masque `/8`, la classe B à un masque `/16`, la classe C à un masque `/24`, la classe D pour le multicast et la classe E pour une utilisation future.

Aujourd'hui, on utilise plus tôt le CIDR (Classless InterDomain Routing) qui consiste à préciser le masque avec un `/<nombre>` à la fin d'une IP (comme on a vu plus tôt). Le CIDR a l'avantage de permettre une configuration des réseaux plus précise et d'être plus simple à comprendre.

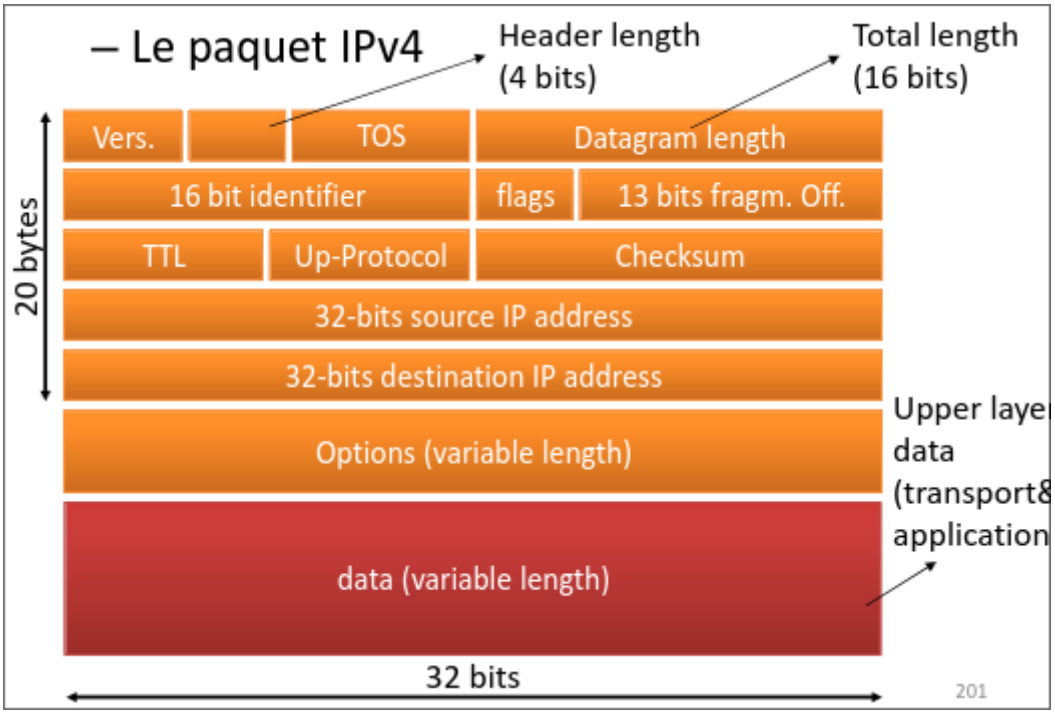
Types d'adresses

Il existe plusieurs types d'adresses :

- Adresses **publiques** qui sont celles utilisées sur Internet et sont achetées à ou allouée par l'IANA (Internet Assigned Number Authority).
- Adresse **loopback** est l'adresse désignant la machine courante, cela permet ainsi d'accéder à un serveur local. Cette adresse est `127.0.0.1` (IPv4) ou `::1` (IPv6) ou `localhost` (qui redirige vers l'adresse IPv4 ou IPv6 dans le fichier d'hôtes).
- Adresses **privées** utilisées sur des réseaux locaux (LAN), elles ne sont pas propagées sur internet et en sont complètement isolées.
- Adresse de **réseau** comme vu plus tôt (préfixe + le reste des bits à 0)
- Adresse de **broadcast** comme vu plus tôt également (préfixe + le reste des bits à 1)

Paquet IPv4

Un paquet IPv4 contient diverses informations :



- La **version** (IPv4 dans ce cas)
- La **longueur d'entête**
- Le "**type of service**" qui indique la priorité du paquet, cette valeur est généralement ignorée
- Le **datagram length** qui indique la longueur totale du paquet sur 16 bits
- L'**identifiant** du paquet
- Des "**flags**" indiquant si le paquet est fragmenté ou non
- Le **fragmentation offset** qui indique le déplacement par rapport au paquet initial
- Le **TTL** (time to live) qui est le temps de survie d'un paquet dans le réseau
- Le **UP-protocol** qui indique le destinataire des données (6 pour TCP, 17 pour UDP)
- Le **Checksum** qui permet de détecter une erreur sur l'entête du paquet, redondant par rapport à TCP

- L'**adresse IP source**
- L'**adresse IP destination**
- Les **options** à ajouter à l'information
- Les données de la couche transport

Fragmentation IPv4

Comme le MSS (Maximum Segment Size) indiquait la taille maximum d'un TPDU de la couche transport, le MTU (Maximum Transert Unit) indique la taille maximale supportée d'un paquet IP, cette taille est imposée par le réseau, donc plusieurs réseaux peuvent avoir des tailles maximales différentes.

Donc si un paquet de longueur 1500 arrive dans un réseau avec un MTU de 600, il va falloir diviser le paquet courant en autres paquets afin de pouvoir le transmettre.

Ainsi le flag permet de savoir si le paquet est fragmenté, l'identifiant permet d'identifier les paquets fragments ensemble. Le fragmentation offset donne l'ordre des fragments. Le dernier fragment a le flag à zéro ce qui indique qu'il n'y a plus de fragments qui suivent.

Eviter les boucles

Pour éviter que des paquets ne bouclent dans le réseau, on utilise la valeur TTL qui indique le temps de vie du paquet en routeur parcouru. Ainsi, la valeur est décrémentée par chaque routeur et si un paquet arrive avec un TTL de un ou inférieur, le paquet est jeté.

ICMPv4

ICMP (Internet Control Message Protocol) permet de "diagnostiquer" certains problèmes réseau via PING et TRACEROUTE.

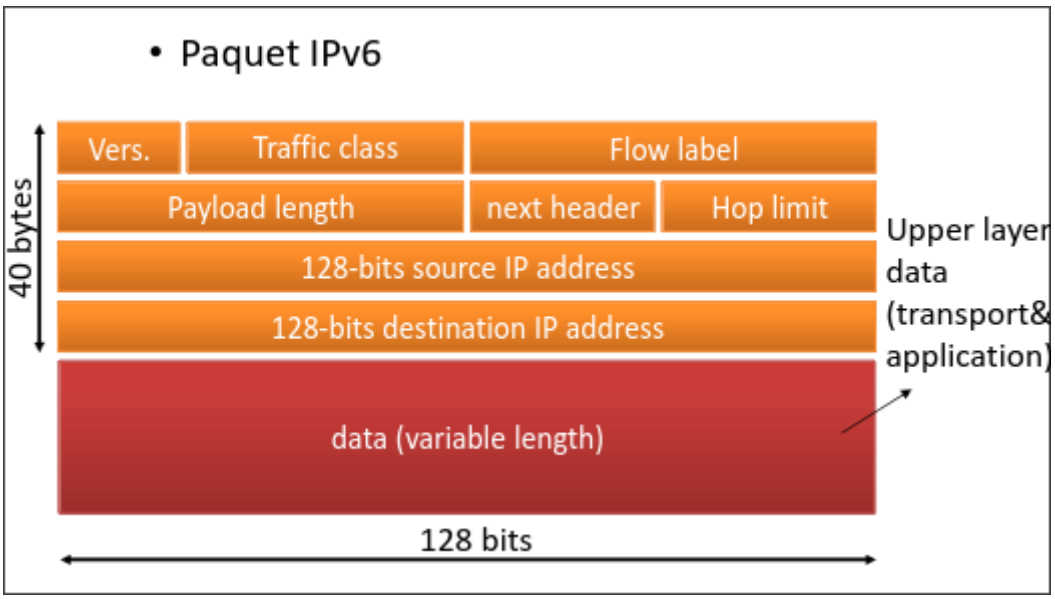
Ping permet de savoir si une machine est capable de recevoir ou répondre au niveau IP, très utile pour vérifier les connexions réseaux, mais souvent limité par les firewalls pour empêcher de pouvoir "tester" les machines du réseau.

Traceroute permet de déterminer le chemin d'une source vers une destination. Cela envoie des paquets IP avec un TTL croissant en partant de 1, qui va donc être rejeté par le routeur qui va ainsi renvoyer un message ICMP `time-exceeded` avec un identifiant du routeur, ce qui permet ainsi de connaitre tous les intermédiaires.

Il est parfois même possible, en utilisant certains sites internet, de retrouver les localisations des routeurs par lequel la requête est passée.

Paquet IPv6

IPv6 comme dit précédemment a pour but d'augmenter le nombre d'adresses disponible, améliorer les performances, intégrer de la sécurité, supporter les nouvelles applications en temps réel, facilité la configuration.



Un paquet IPv6 est composé de :

- La **version** (ici IPv6)
- Le **Traffic Class** permettant de marquer les paquets pour obtenir une *qualité de service particulière*
- **Flow label** permettant d'étiqueter un paquet afin de grouper des paquets faisant partie d'un "flow" commun tel qu'un live vidéo par exemple
- Le **payload length** indiquant la taille des données
- Le **next header** qui mentionne une option à traiter ou pour indiquer la couche supérieure à recevoir des données (UDP ou TCP)
- Le **hop limit** qui indique le total de routeurs que le paquet peut traverser, lorsque cette valeur arrive à 0 le paquet est jeté. Similaire au TTL en IPv4
- L'**adresse IP source**
- L'**adresse IP destination**
- Les données de la couche de transport

Différence avec l'IPv4

Il y a donc certaines différences entre les paquets IPv6 et les paquets IPv4 :

- Le checksum n'existe plus, car redondant par rapport à la couche de transport ou d'accès réseau. Cela permet d'améliorer les performances, parce que les routeurs n'ont plus

besoin de vérifier cette valeur
- Disparition des options de taille variable, ces derniers peuvent être placés dans un entête particulière
- Disparition de la fragmentation au niveau des routeurs, IPv6 oblige un MTU de 1280 octets, les paquets peuvent donc être fragmentés. Cela est un grand avantage pour les routeurs qui n'ont plus besoin de fragmenter les paquets, c'est ainsi un gain de performance.

Types d'adresses IPv6

- Adresses **locale-lien** est attachée à l'interface et a une portée limitée au LAN, et ne traverse pas les routeurs. Ce sont des adresses de type `FE80::/10`
- Adresses **locale-unique** sont des adresses attachées à une interface qui peuvent traverser des routeurs mais non utilisable sur internet. Elles sont de type `FC00::/7`
- Adresses **globale-unique** sont des adresses globales, uniques sur Internet et attribuée par le service d'accès Internet. Elles sont de type `2000::/3`
- Adresses **multicast** désignant un groupe de receveurs. Elles sont de type `FF00::/8`
- Adresses **anycast** permettant d'interagir avec n'importe quelle adresse d'un groupe donné (le plus proche selon la politique de routage)

En IPv6, une machine a donc plusieurs adresses, par exemple une adresse globale-unique, une adresse locale-lien et une adresse IPv4 globale.

ICMPv6

ICMPv6 est une évolution d'ICMP pour l'IPv6. Ce protocole permet plusieurs choses :

- Elle permet une gestion des groupes multicast (IGMP en IPv4),
- Retrouver une adresse physique (MAC) en fonction d'une IP (ARP en IPv4),
- Neighbor discovery qui permet de déterminer les adresses local-lien des voisins, routeurs, etc afin de savoir si un voisin est accessible ou non

Elle permet aussi la signalisation des erreurs (destination unreachable, packet too big, time exceeded, parameter problem) ou d'autres messages d'information (ping, gestion de groupes multicast, neighbor discovery).

Neighbor Discovery Protocol (ND)

Le ND est un protocole utilisé pour découvrir les voisins et réaliser de l'auto-configuration des interfaces réseaux (construire l'adresse locale-lien de façon à ce qu'elle soit unique sur le LAN).

Contrairement à DHCP (Dynamic Host Configuration Protocol), qui sert aussi à configurer les hôtes, les adresses ne sont pas distribuées, mais construites. Il est cependant aussi possible de distribuer des adresses en IPv6 en utilisant DHCPv6.

ND prévoit 5 types de messages :

- Sollicitation de routeur, vérification d'un routeur
- Réponse de routeur, un routeur annonce sa présence
- Sollicitation de voisin, vérification d'un voisin
- Réponse de voisin, un voisin indique sa présence
- Redirection, redirige vers un autre routeur

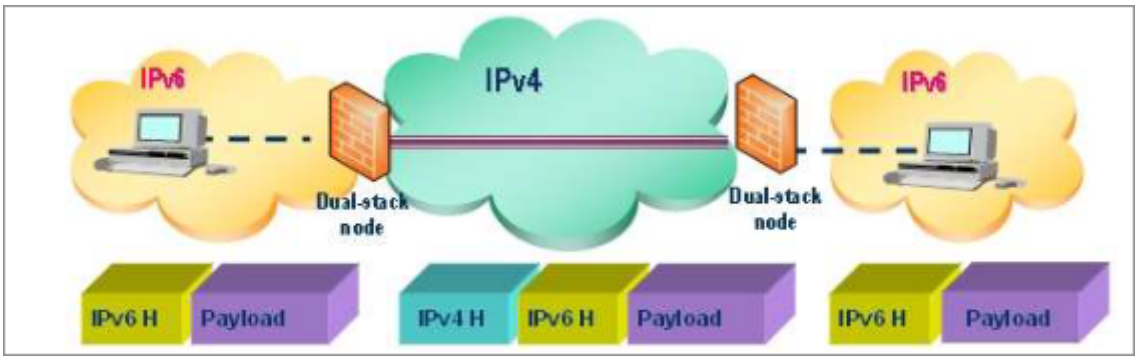
Neighbor solicitation/adverisement permet d'obtenir l'adresse physique (MAC) d'un voisin et de vérifier s'il est accessible.

Transition vers IPv6

La transition vers la version 6 du protocole IP doit être progressive, car il est impossible d'imposer un changement brutal sur plusieurs milliards d'appareils d'un coup. De plus, les couts pour la transition peuvent être importants parce qu'il faut un logiciel et/ou matériel adapté.

Il existe plusieurs dispositifs pour faire une transition vers l'IPv6 graduelle

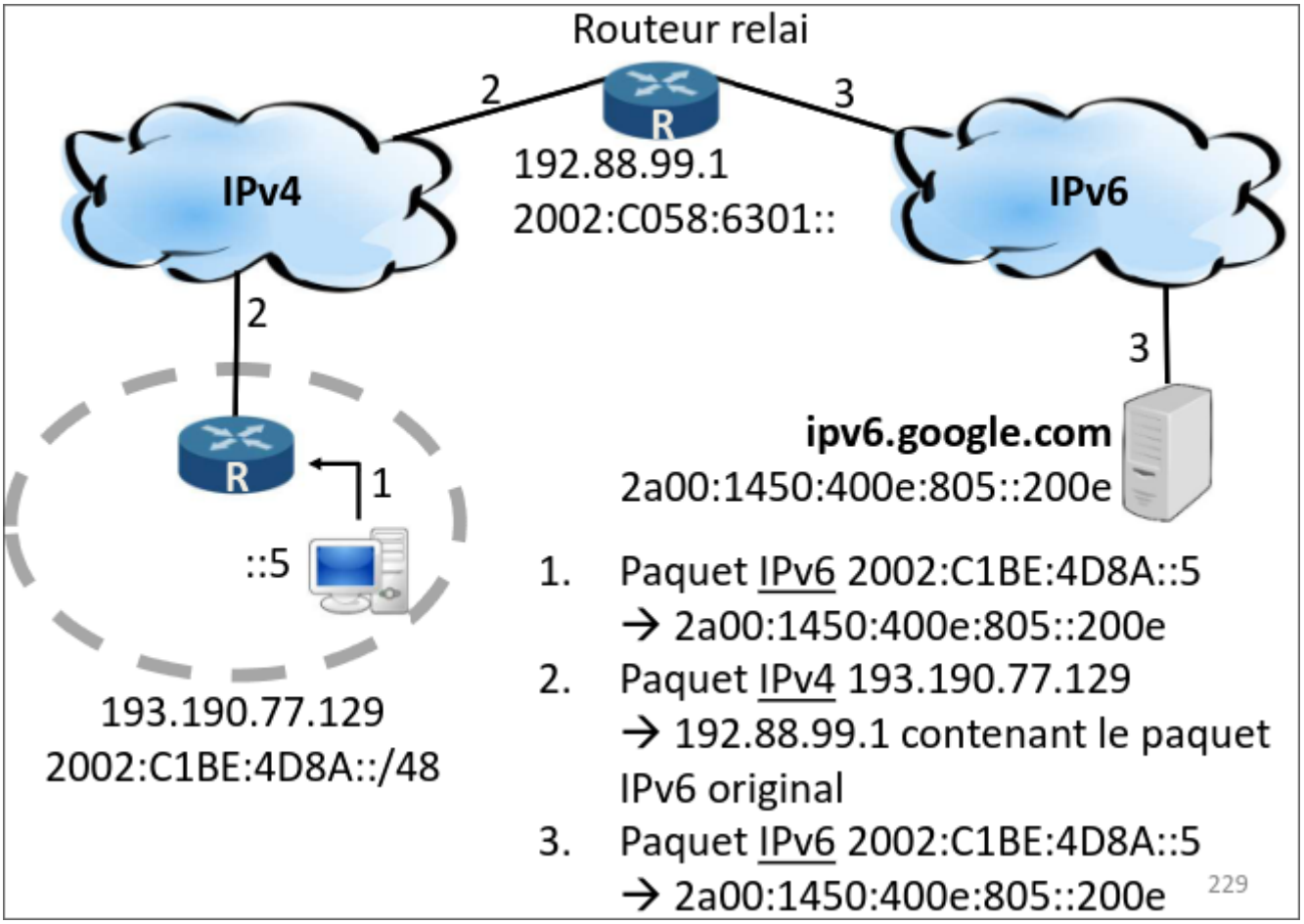
Tunnel-brokers



L'idée ici est de créer un tunnel entre le réseau du client et du fournisseur de service IPv6 (par exemple un site internet). Cela consiste à mettre des paquets IPv6 dans des paquets IPv4.

Un client va donc s'abonner aux services d'un tunnel-broker (par exemple Sixxs), et va envoyer ses paquets IPv6 dans des paquets IPv4 à ce tunnel-broker. Ce dernier va ensuite récupérer les paquets IPv4 pour les envoyer dans le réseau IPv6.

6to4



Une autre idée est d'utiliser des "tunnels automatiques" tel que des routeurs 6to4 qui vont faire le pont entre l'Internet IPv6 et l'Internet IPv4.

Ainsi les paquets IPv6 vont être encapsulés dans des paquets IPv4 et envoyé au routeur relais le plus proche. Ce dernier va ensuite envoyer le paquet IPv6 dans le réseau IPv6 et faire la même chose dans le sens inverse.

Tous les routeurs relais sont identifiés par l'adresse Anycast `192.88.99.1`. Ici le tunnel se fait automatiquement, il n'y a donc pas besoin de s'abonner à un service particulier par exemple.

L'IPv6 d'origine renseignée dans le paquet IPv6 commence toujours par `2002:` pour l'IPv6 et est suivi de l'adresse IPv4 publique. Cela permet ainsi, en utilisant une adresse IPv4 publique, de créer une adresse IPv6 unique.

Le 6to4 a cependant quelques problèmes, par exemple, il est assez difficile de dimensionner les routeurs 6to4 et il est difficile d'assurer une bonne connectivité. Surtout que si un ISP crée un routeur 6to4, il sera également obligé de gérer les clients d'autres ISPs.

Déploiement dual-stack

Le déploiement dual-stack est la méthode de transition la plus "pure" vers l'IPv6 puis ce qu'elle consiste en un fournisseur d'accès Internet qui transforme son infrastructure de manière à supporter nativement à la fois IPv4 et IPv6 (d'où le nom "dual-stack").

Ainsi, le réseau du client, ainsi que le réseau de l'ISP supporte tous les deux l'IPv6 et l'IPv4.

Le problème avec le dual-stack c'est que cela demande souvent une refonte importante de tout le réseau, ce qui n'est parfois pas possible.

6rd

6 Rapid Deployment est une technologie développée par Free. Elle consiste à implémenter une sorte de 6to4 à l'intérieur du réseau de l'ISP.

Ainsi les clients ont des routeurs 6rd qui vont placer les paquets IPv6 dans des paquets IPv4 et va l'envoyer à un routeur 6rd qui va ensuite propager les paquets IPv6 sur le réseau IPv6.

Les machines sont identifiées par des adresses IPv6 locale au réseau de l'ISP, il n'y a donc pas besoin de préfixe comme ce serait le cas pour 6to4 et il n'est ainsi pas nécessaire de modifier tout le réseau existant.

Le 6rd a permis à Free de déployer l'IPv6 à 1.5 million de clients en seulement 5 semaines.

Pallier les problèmes de l'IPv4 en attendant l'IPv6

En attendant que l'IPv6 soit vraiment répandue, il est possible de tout de même palier aux limites d'adressage de l'IPv4.

L'utilisation du CIDR permet par exemple d'affiner les espaces IP adressés et ainsi éviter d'assigner trop d'adresses à une entité.

Le **NAT** (Network Address Translator) permet de partager une adresse IP publique entre plusieurs centaines de machines. C'est ce qui est généralement utilisé pour tous les réseaux domestiques.

Le DHCP (Dynamic Host Configuration Protocol) permet de distribuer les adresses au moment de la connexion et ainsi d'en avoir moins pour un certain groupe d'utilisateur.

Fonctionnement du NAT

Le NAT fonctionne en modifiant le port et l'IP source des paquets. Ainsi, lorsqu'un paquet veut sortir du réseau local, le NAT va remplacer l'IP locale par l'IP publique, et le port source par un autre. Quand la réponse du serveur arrive, le NAT va regarder quel IP locale et quel port source correspond au port source transformé, va de nouveau modifier le paquet et l'envoyer à la bonne machine du réseau.

Le routeur peut également parfois faire office de firewall.