Applications réseau

Cours 2 : Couche applications

Florian Sikora
florian.sikora@dauphine.fr

LAMSADE

M1 apprentissage

Adapté des slides de Kurose & Ross

Cours 2 : Couche applications

Objectifs

- Comprendre le concept, l'implémentation de protocoles d'applications réseau.
  - Comment se servir de la couche de transport.
  - Paradigme client-serveur vs p2p.
- En savoir plus sur quelques applications populaires (HTTP, FTP...)
- A terme : savoir implémenter une application réseau.

Quizz !

- `http://www.quizzoodle.com/session/27d84913c1eb44119d7b0712735b34f0`

Cours 2 : Couche applications

Applications réseau

- Applications : la raison pour laquelle un réseau existe.
- Q : Des applications réseau ?

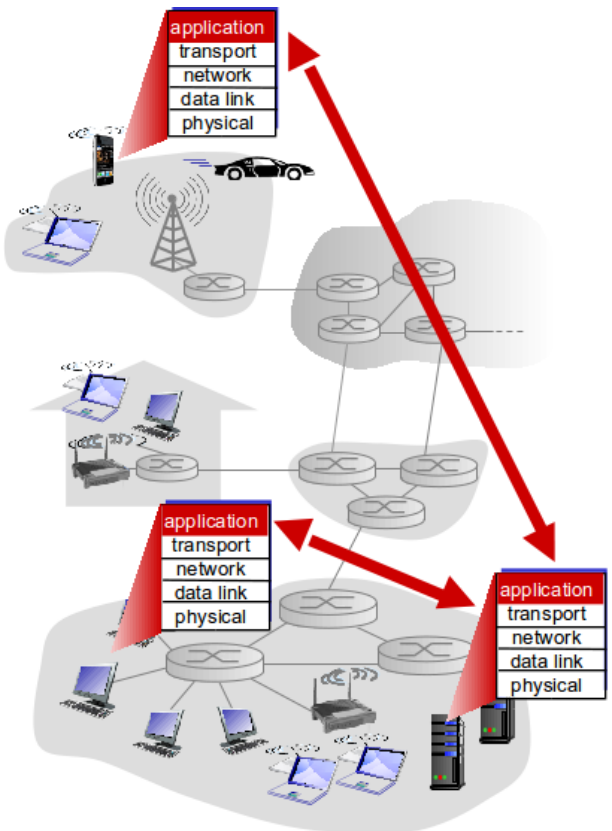
Applications réseau

- Applications : la raison pour laquelle un réseau existe.

- Q : Des applications réseau ?
- E-mail.
- Web.
- Connexion à distance.
- Partage de fichiers en P2P.
- Jeux multi-joueurs.
- Vidéo en streaming.
- Voix sur IP.
- ...

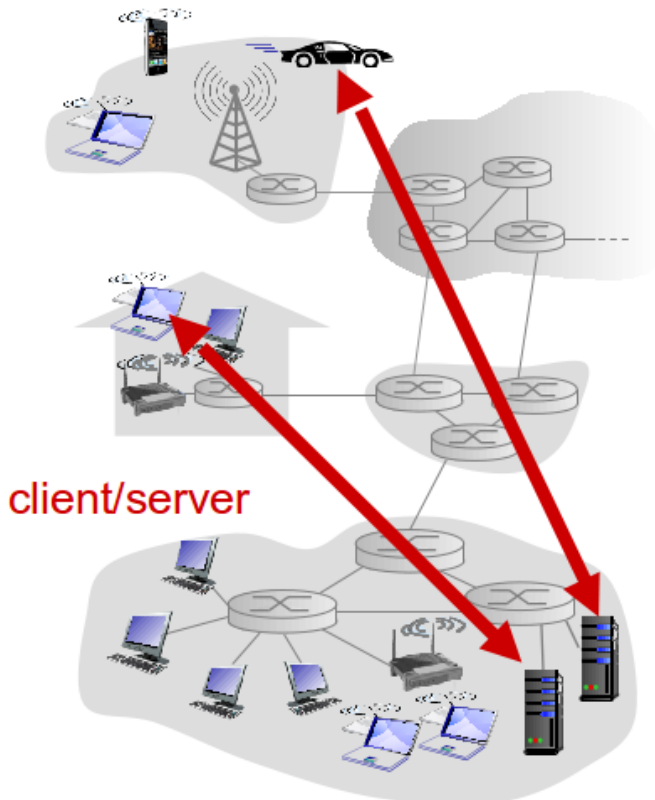
Applications réseau

- Se répartit sur plusieurs systèmes, différents terminaux.
- Plusieurs *processus* communiquant sur le réseau.
  - Web : un navigateur sur un client, un programme sur un serveur Web.
  - Vidéo-conférence...



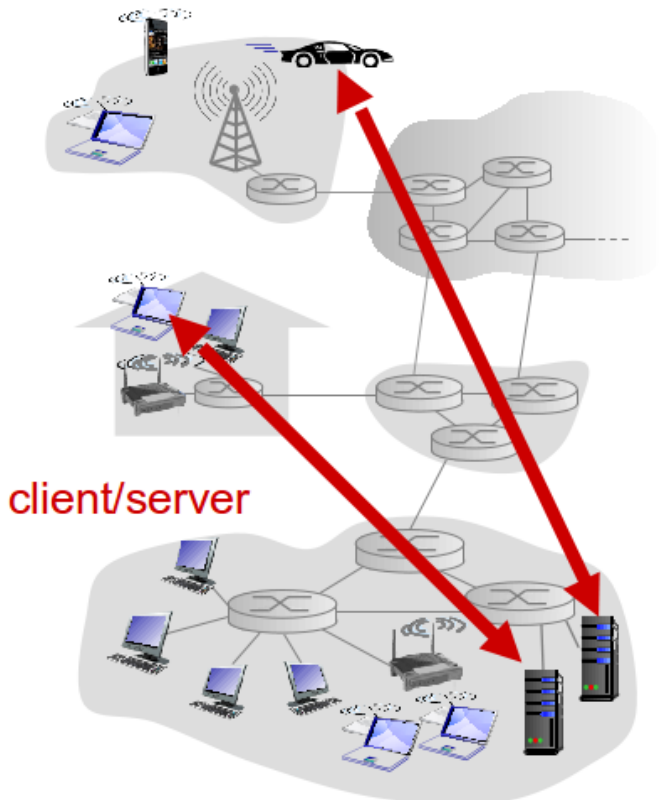
Client-Serveur vs P2P : Architecture Client-Serveur

- Serveur :
  - Un hôte.
  - Adresse IP permanente.
  - Data-center pour augmentation de taille (facebook et la réplication de données...).
- Client :
  - Communique avec un serveur.
  - Peut-être connecté par intermittence.
  - Peut avoir une IP dynamique.
  - Ne communique pas directement avec un autre client.
- Un même hôte peut jouer les 2 rôles (FTP, SSH...)



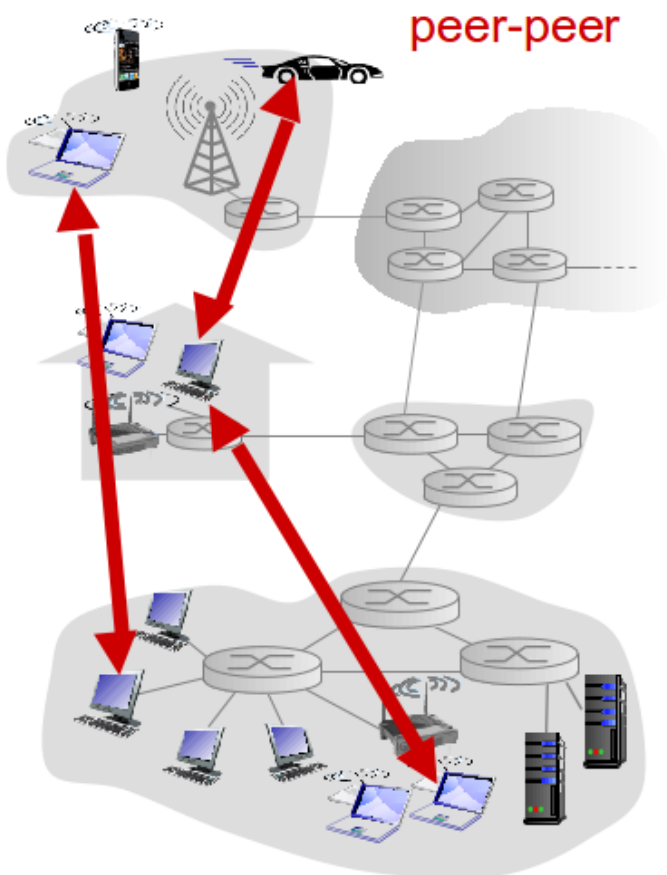
Client-Serveur vs P2P : Architecture Client-Serveur

- Serveur :
  - Un hôte.
  - Adresse IP permanente.
  - Data-center pour augmentation de taille (facebook et la réplication de données...).
- Client :
  - Communique avec un serveur.
  - Peut-être connecté par intermittence.
  - Peut avoir une IP dynamique.
  - Ne communique pas directement avec un autre client.
- Un même hôte peut jouer les 2 rôles (FTP, SSH...)
- Encombrement du serveur et de sa liaison...



Client-Serveur vs P2P : Architecture Peer-to-Peer

- Pas toujours un serveur impliqué.
- Deux terminaux communiquent directement.
- Un “pair” :
  - Demande un service à un autre pair.
  - Propose un service à d’autres pairs.
- “Self-scalability” : un nouveau pair apporte une nouvelle capacité de service en même temps que de nouvelles demandes ($\neq$ client-serveur).
- Pairs peuvent être connectés par intermittence et changer d’adresse IP (implique une gestion difficile).



Communication entre processus

- Processus : “programme” tournant sur un hôte.
- Sur un même hôte, deux processus peuvent communiquer en interne (selon l'OS). Pas ce cours.
- Deux processus sur des hôtes différents communiquent en échangeant des messages.

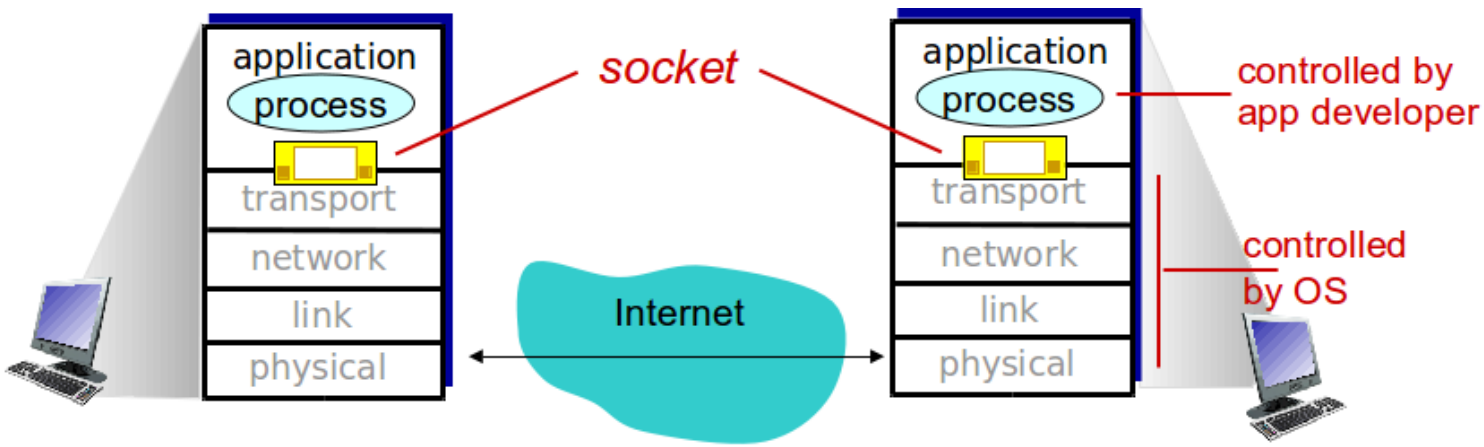
Communication entre processus

- Processus : “programme” tournant sur un hôte.
- Sur un même hôte, deux processus peuvent communiquer en interne (selon l'OS). Pas ce cours.
- Deux processus sur des hôtes différents communiquent en échangeant des messages.

- Client-Serveur :
  - Processus client : processus initiant la communication.
  - Processus serveur : processus attendant d'être contacté.
- Peer-to-peer :
  - Une application P2P à des processus clients et des processus serveurs.

Sockets

- Un processus envoie/reçoit des messages vers/depuis sa *socket*.
- Socket est comme une porte :
  - Le processus envoyant pousse un message par la porte.
  - Le processus envoyant se base sur l'infrastructure de transport de l'autre côté de la porte pour placer le message à la porte du receveur.



Processus d'adressage

- Pour recevoir un message, le processus doit avoir un identifiant.
- Un hôte à une unique adresse IP sur 32 bits (supposons IPv4...).
- Q : Suffisant pour identifier le processus ?

Processus d'adressage

- Pour recevoir un message, le processus doit avoir un identifiant.
- Un hôte à une unique adresse IP sur 32 bits (supposons IPv4...).
- Q : Suffisant pour identifier le processus ?
- Non ! Plusieurs processus peuvent tourner sur un même hôte !

Processus d'adressage

- Pour recevoir un message, le processus doit avoir un identifiant.
- Un hôte à une unique adresse IP sur 32 bits (supposons IPv4...).
- Q : Suffisant pour identifier le processus ?
- Non ! Plusieurs processus peuvent tourner sur un même hôte !
- Un identifiant inclut une adresse IP ET un **numéro de port** (de porte ?), associé à chaque processus.
  - Par exemple, un serveur HTTP par défaut sur le porte 80...
  - Pour envoyer un message HTTP à www.lamsade.dauphine.fr :
    - L'envoyer à l'IP 193.48.71.250
    - Port : 80.

Protocoles d'applications réseau

- Doivent définir :
  - Le **type** de message échangés (requête, réponse...).
  - La **syntaxe** des messages (les champs à remplir, de quel taille...).
  - La **sémantique** des messages (interprétation des champs...).
  - Des règles sur quand et comment envoyer et répondre aux messages.

Protocoles d'applications réseau

- Des protocoles sont **ouverts**.
  - Définis dans les RFC.
  - Autorise une interopérabilité.
  - Exemple : HTTP, SMTP...
- Des protocoles **propriétaires**... Q : Exemple ?

Protocoles d'applications réseau

- Des protocoles sont **ouverts**.
  - Définis dans les RFC.
  - Autorise une interopérabilité.
  - Exemple : HTTP, SMTP...
- Des protocoles **propriétaires**... Q : Exemple ?
  - Skype, (feu) MSN Messenger...
  - Fausse la concurrence, autres logiciels doivent faire de la retro-ingénierie (précédents judiciaires l'autorisant pour interopérabilité).

Services nécessaires à une application réseau

- Plusieurs protocoles de transports existent, doit faire un choix (comme entre avion ou train...).

Services nécessaires à une application réseau

- Plusieurs protocoles de transports existent, doit faire un choix (comme entre avion ou train...).
- **Fiabilité** du transport de données.
  - Certaines ont besoin d'un transport de données fiable à 100% (messagerie, transfert de docs, appli. financières...).
  - Certaines sont tolérantes. Multimédia (audio, vidéo temps réel...)

Services nécessaires à une application réseau

- Plusieurs protocoles de transports existent, doit faire un choix (comme entre avion ou train...).
- **Fiabilité** du transport de données.
  - Certaines ont besoin d'un transport de données fiable à 100% (messagerie, transfert de docs, appli. financières...).
  - Certaines sont tolérantes. Multimédia (audio, vidéo temps réel...)
- **Débit**.
  - Certaines applications requièrent un débit minimal disponible en permanence (multimédia avec encodage à un certain débit...).
  - Certaines utilisent ce qu'elles trouvent (courrier électronique, transfert de fichiers...)

Services nécessaires à une application réseau

- Plusieurs protocoles de transports existent, doit faire un choix (comme entre avion ou train...).
- **Fiabilité** du transport de données.
  - Certaines ont besoin d'un transport de données fiable à 100% (messagerie, transfert de docs, appli. financières...).
  - Certaines sont tolérantes. Multimédia (audio, vidéo temps réel...)
- **Débit**.
  - Certaines applications requièrent un débit minimal disponible en permanence (multimédia avec encodage à un certain débit...).
  - Certaines utilisent ce qu'elles trouvent (courrier électronique, transfert de fichiers...)
- Contraintes de **temps**.
  - Certaines applications ont besoin de faibles délais (temps réel : téléphonie, jeux en ligne...).
  - D'autres peuvent accepter le délai (courrier...).

Services nécessaires à une application réseau

- Plusieurs protocoles de transports existent, doit faire un choix (comme entre avion ou train...).
- **Fiabilité** du transport de données.
  - Certaines ont besoin d'un transport de données fiable à 100% (messagerie, transfert de docs, appli. financières...).
  - Certaines sont tolérantes. Multimédia (audio, vidéo temps réel...)
- **Débit**.
  - Certaines applications requièrent un débit minimal disponible en permanence (multimédia avec encodage à un certain débit...).
  - Certaines utilisent ce qu'elles trouvent (courrier électronique, transfert de fichiers...)
- Contraintes de **temps**.
  - Certaines applications ont besoin de faibles délais (temps réel : téléphonie, jeux en ligne...).
  - D'autres peuvent accepter le délai (courrier...).
- **Sécurité**
  - Cryptage, intégrité des données...

Protocoles transports pour Internet (appronfondi plus tard)

- TCP (Lourd mais fiable).
- UDP (Simple, léger, services minimum).

Protocoles transports pour Internet (appronfondi plus tard)

- TCP (Lourd mais fiable).
- UDP (Simple, léger, services minimum).
  - N'assure pas de fiabilité durant le transport des données entre émetteur et receveur.
  - Pas de contrôle de flux, ordre d'envoi, contrôle de congestion, sécurité...
  - Pas de garantie de temps de transmission.

Protocoles transports pour Internet (appronfondi plus tard)

- UDP (Simple, léger, services minimum).
  - N'assure pas de fiabilité durant le transport des données entre émetteur et receveur.
  - Pas de contrôle de flux, ordre d'envoi, contrôle de congestion, sécurité...
  - Pas de garantie de temps de transmission.
- TCP (Lourd mais fiable).
  - Orienté connexion (échange d'information entre les deux avant le transfert).
  - Fiabilité des données transportées et ordre préservé.
  - Contrôle de flux (émetteur de submerge pas le receveur).
  - Contrôle de congestion (diminution des envois sur surcharge du réseau).
  - Pas de garantie de temps, de débit minimum, de sécurité.

Applications et protocoles

Appli.	Prot. Appl.	Pertes pos-sibles	Débit	Sensib. temps	Prot. Transp.
Transferts	FTP				

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes pos-sibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes pos-sibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP				

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes possibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes possibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	TCP
Web	HTTP				

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes pos-sibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	TCP
Web	HTTP	Non	Flexible	Non	

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes possibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	TCP
Web	HTTP	Non	Flexible	Non	TCP
Streaming Audio/-Vidéo	HTTP (youtube), RTP				

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes possibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	TCP
Web	HTTP	Non	Flexible	Non	TCP
Streaming Audio/-Vidéo	HTTP (youtube), RTP	Tolérant	Audio : >5kbps Vidéo > 10kbps	Oui, msec	

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes possibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	TCP
Web	HTTP	Non	Flexible	Non	TCP
Streaming Audio/-Vidéo	HTTP (youtube), RTP	Tolérant	Audio : >5kbps Vidéo > 10kbps	Oui, msec	TCP ou UDP
Téléphonie Internet	SIP, propriétaire (Skype)				

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes possibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	TCP
Web	HTTP	Non	Flexible	Non	TCP
Streaming Audio/-Vidéo	HTTP (youtube), RTP	Tolérant	Audio : >5kbps Vidéo > 10kbps	Oui, msec	TCP ou UDP
Téléphonie Internet	SIP, propriétaire (Skype)	Oui	Idem	Oui, msec	

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes possibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	TCP
Web	HTTP	Non	Flexible	Non	TCP
Streaming Audio/-Vidéo	HTTP (youtube), RTP	Tolérant	Audio : >5kbps Vidéo > 10kbps	Oui, msec	TCP ou UDP
Téléphonie Internet	SIP, propriétaire (Skype)	Oui	Idem	Oui, msec	Souvent UDP
Jeux multi.					

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes possibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	TCP
Web	HTTP	Non	Flexible	Non	TCP
Streaming Audio/-Vidéo	HTTP (youtube), RTP	Tolérant	Audio : >5kbps Vidéo > 10kbps	Oui, msec	TCP ou UDP
Téléphonie Internet	SIP, propriétaire (Skype)	Oui	Idem	Oui, msec	Souvent UDP
Jeux multi.		Oui	quelques kbps	OUI	

Applications et protocoles

Appli.	**Prot. Appl.**	**Pertes possibles**	**Débit**	**Sensib. temps**	**Prot. Transp.**
Transferts	FTP	Non	Flexible	Non	TCP
e-Mail	SMTP	Non	Flexible	Non	TCP
Web	HTTP	Non	Flexible	Non	TCP
Streaming Audio/-Vidéo	HTTP (youtube), RTP	Tolérant	Audio : >5kbps Vidéo > 10kbps	Oui, msec	TCP ou UDP
Téléphonie Internet	SIP, propriétaire (Skype)	Oui	Idem	Oui, msec	Souvent UDP
Jeux multi.		Oui	quelques kbps	OUI	Souvent UDP

Sécuriser TCP

- Pas de cryptage sur TCP ou UDP.
- Mots de passes en clair à travers Internet...

Sécuriser TCP

- Pas de cryptage sur TCP ou UDP.
- Mots de passes en clair à travers Internet...
- SSL (TSL)
  - Session TCP chiffrée.
  - Authentification client et serveur.
  - Entre la couche Application et Transport (applications “parlent” à TCP via SSL) (HTTPS...)
- Plus à suivre...

Cours 2 : Couche applications

Web et HTTP

- Une page web : des objets.
  - Objets : fichier HTML, images JPG, applette Java...
- Une page web : un fichier HTML de base, faisant références à des objets.
- Chaque objet à une adresse URL (Uniform Resource Locator), du genre `www.legorafi.fr/wp-content/uploads/2012/09/logositeter.png`
  - `www.legorafi.fr` : nom du serveur hébergeant l'objet.
  - `/wp-content/uploads/2012/09/logositeter.png` : chemin de l'objet sur l'hôte.

URL/URN/URI

- URI (Uniform Ressource Indentifier), RFC 2396.
- Exemples :
  - URL (Locator) : donne la localisation sur le réseau.
  - URN (Name) : indépendant de la localisation (censé éviter les liens cassés. Nécessiterai un annuaire de correspondance, difficile à mettre en place, peu utilisé).
- URI identifie selon un *schéma* : http, ftp, mailto, irc...
- Syntaxe générale :

  [<schéma> :]<partie dépendante du schéma>[# fragment]
- Exemples :
  - URN :ISBN :2-7117-8689-7
  - http ://www.wikipedia.org (via http, il y a une ressource sur l'hôte...)

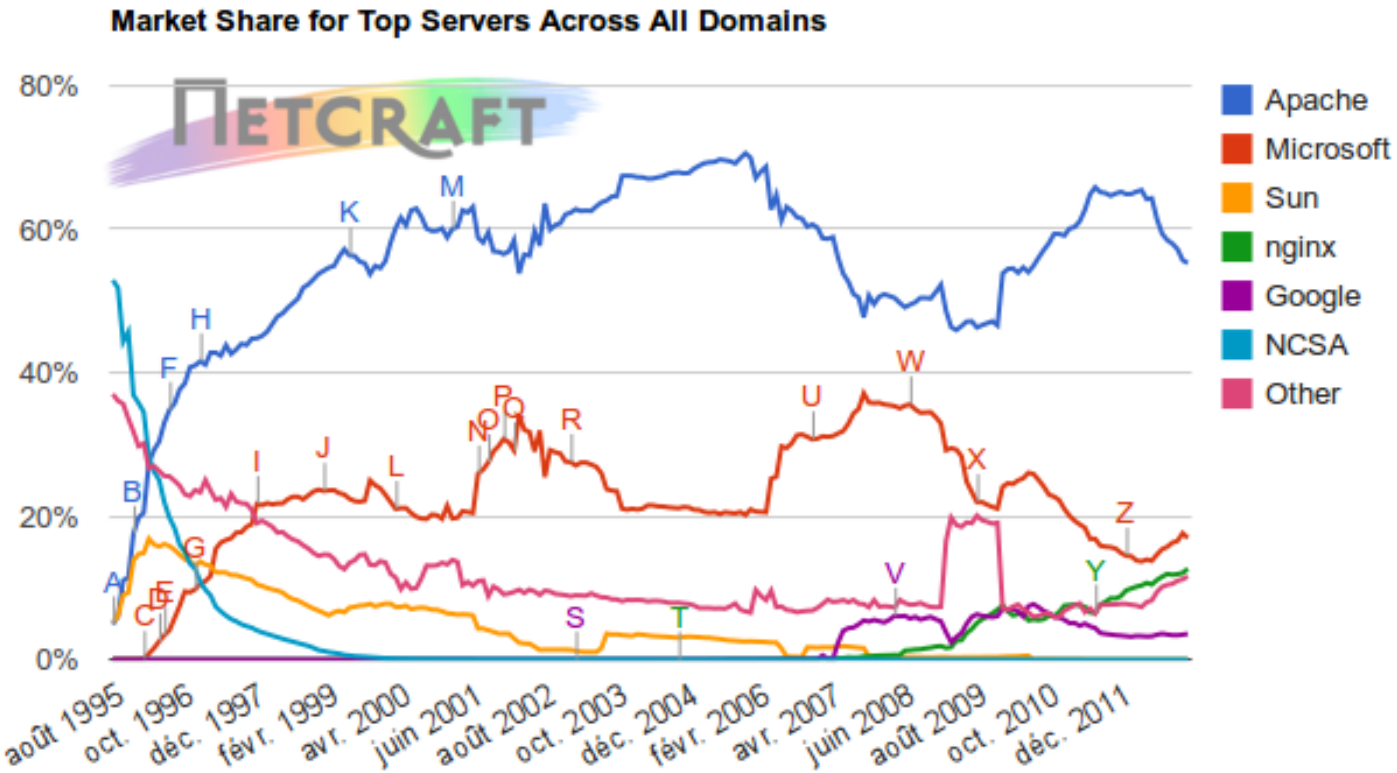
Schéma http

- `http ://<machine> :<port>/<path> ?<query>`
- Port par defaut : 80.
- path : souvent le chemin d'accès sur le système de fichier de l'hôte.
- query : infos supplémentaires...
- Schéma https : idem mais port par défaut 443.

HTTP (Hypertext Transfert Protocol)

- Client/Serveur.
- Client : navigateur qui envoie des requêtes, reçoit (via HTTP) et affiche des objets.
- Serveur : Ordinateur (avec un logiciel) qui envoie (via HTTP) des objets (qu'il héberge) en réponse à des requêtes.

Serveurs Web



Developer	December 2012	Percent	January 2013	Percent	Change
Apache	352,951,511	55.70%	348,119,032	55.26%	-0.43
Microsoft	111,570,010	17.61%	106,619,177	16.93%	-0.68
nginx	76,460,756	12.07%	79,640,472	12.64%	0.58
Google	21,870,614	3.45%	22,573,858	3.58%	0.13

HTTP (suite)

- Utilise TCP (HTTP n'a pas à se soucier de l'intégrité des données.).
  - Le client crée une connexion TCP (création d'une socket) vers le serveur, port 80.
  - Le serveur accepte la connexion TCP du client (s'il le peut...).
  - Échange de messages HTTP.
  - Fermeture de la connexion TCP.

HTTP (suite)

- Utilise TCP (HTTP n'a pas à se soucier de l'intégrité des données.).
  - Le client crée une connexion TCP (création d'une socket) vers le serveur, port 80.
  - Le serveur accepte la connexion TCP du client (s'il le peut...).
  - Échange de messages HTTP.
  - Fermeture de la connexion TCP.
- HTTP est "sans mémoire" (stateless).
  - Le serveur répond aux clients sans stocker d'information relative au client.

HTTP (suite)

- Utilise TCP (HTTP n'a pas à se soucier de l'intégrité des données.).
  - Le client crée une connexion TCP (création d'une socket) vers le serveur, port 80.
  - Le serveur accepte la connexion TCP du client (s'il le peut...).
  - Échange de messages HTTP.
  - Fermeture de la connexion TCP.
- HTTP est "sans mémoire" (stateless).
  - Le serveur répond aux clients sans stocker d'information relative au client.
- Les protocoles avec mémoire sont complexes (mémoire du client et du serveur non identiques si un des deux plante...)

Connexions HTTP

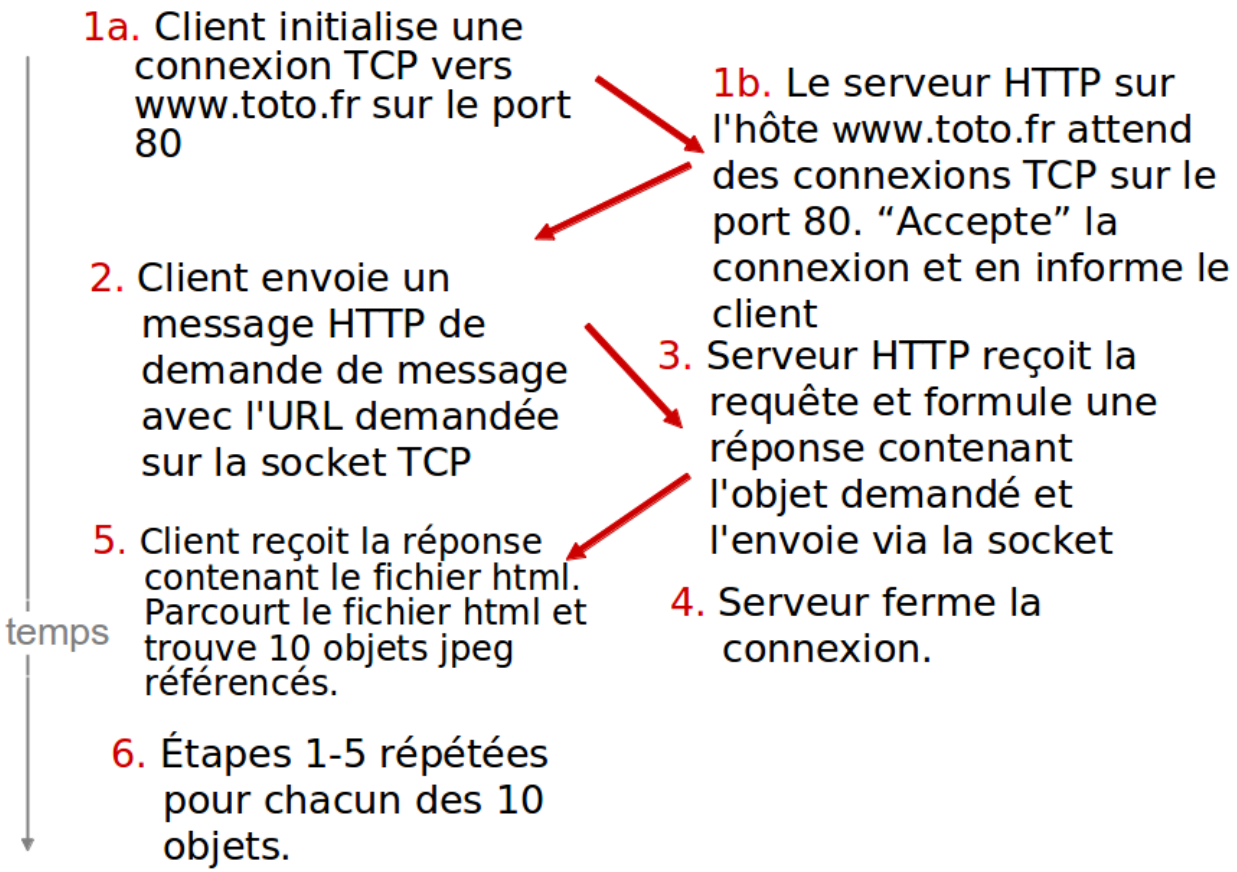
- Deux types de connexions pour HTTP.

- Connexions **non-persistantes** (HTTP 0.9, 1.0, 1.1).
  - Au plus un objet envoyé par connexion TCP, puis fermeture de connexion.
  - Envoi de plusieurs objets = plusieurs connexions.

- Connexions **persistantes** (HTTP 1.1).
  - Plusieurs objets peuvent être envoyés à travers une seule connexion TCP.

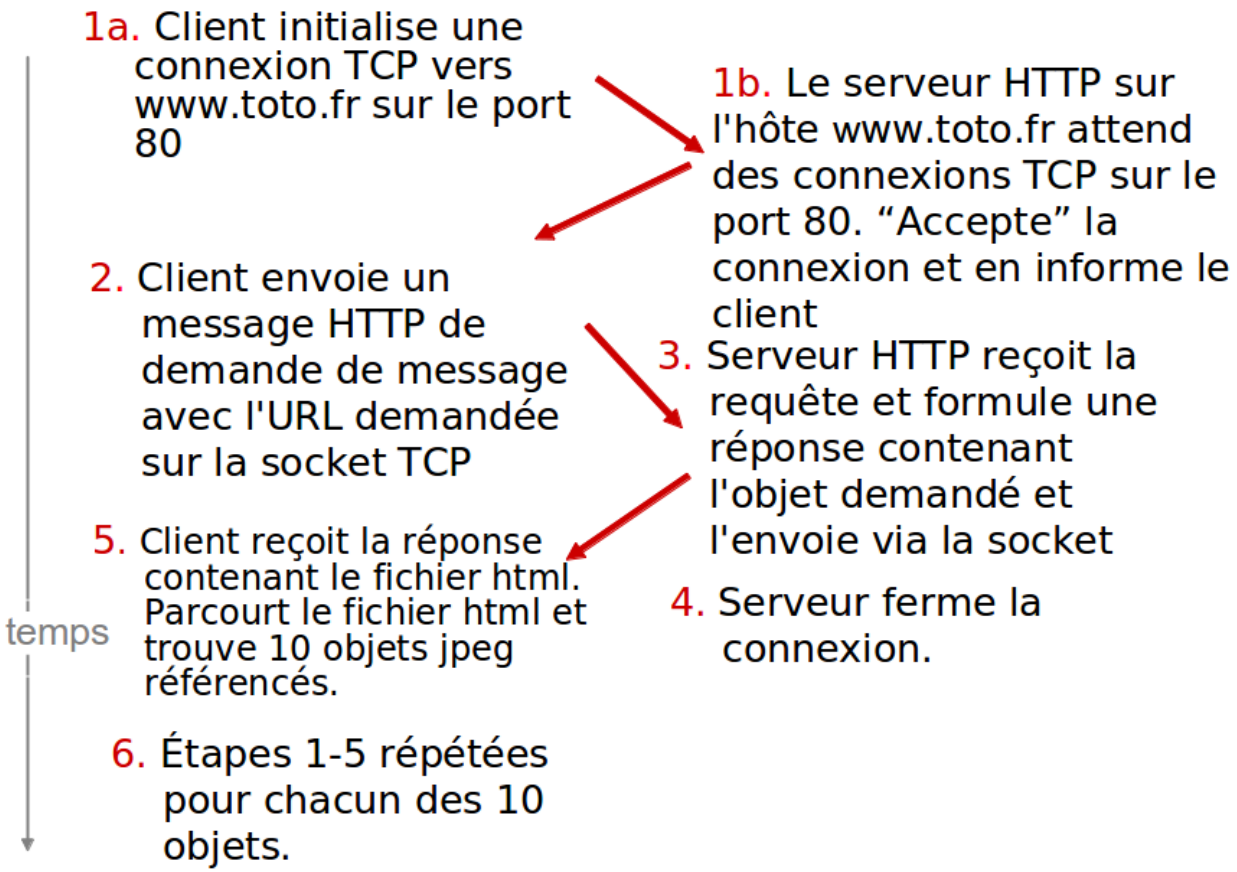
HTTP non-persistant

- Un utilisateur entre l'URL `www.toto.fr/titi/tutu.html`, contenant du texte, et 10 références vers des images jpeg.



HTTP non-persistant

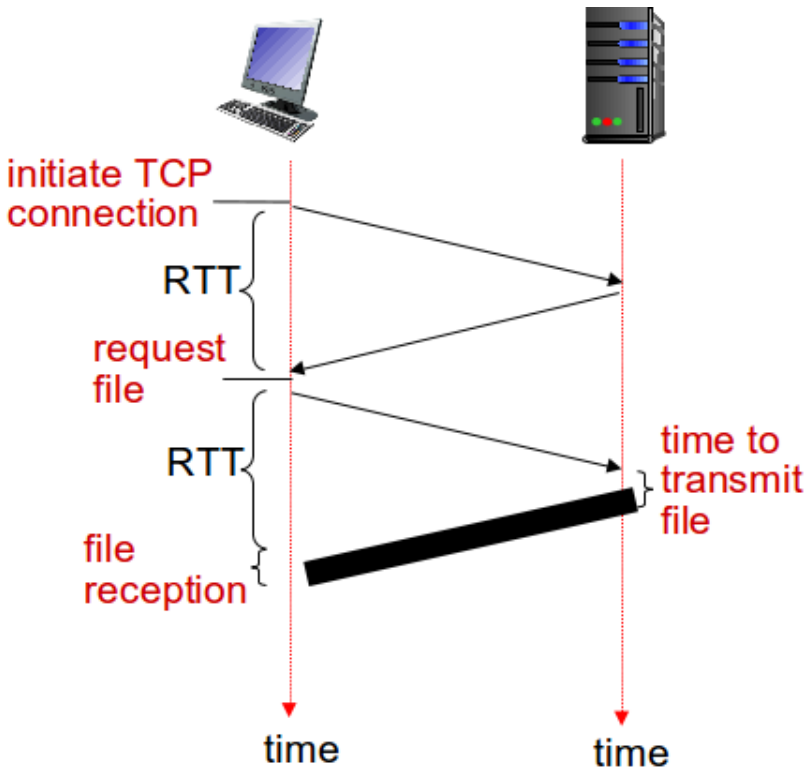
- Un utilisateur entre l'URL `www.toto.fr/titi/tutu.html`, contenant du texte, et 10 références vers des images jpeg.



- 11 connexions établies pour une page.

HTTP non-persistant : temps de réponse

- RTT (**Round Trip Time**) : temps pour un paquet de "voyager" du client au serveur puis de revenir.
- Temps de réponse :
  - 1 RTT pour initier la connexion TCP.
  - 1 RTT pour la requête HTTP et le temps de retour.
  - Le temps de transmission de l'objet.
  - Temps de réponse en non-persistant : $2 \cdot \text{RTT} +$ temps de transmission.



HTTP persistant

- Problèmes du non-persistant :
  - Nécessite 2 RTT par objet à transférer.
  - Allocation de ressources pour chaque connexion niveau client et serveur...
- Les navigateurs modernes ouvrent souvent des connexions en parallèle.

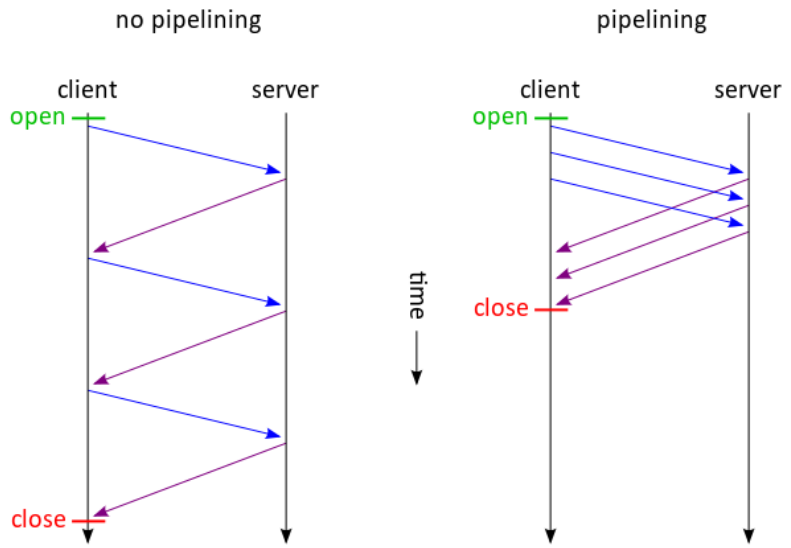
HTTP persistant

- Problèmes du non-persistant :
  - Nécessite 2 RTT par objet à transférer.
  - Allocation de ressources pour chaque connexion niveau client et serveur...
- Les navigateurs modernes ouvrent souvent des connexions en parallèle.
- HTTP persistant :
  - Le serveur laisse ouvert la connexion après avoir envoyé une réponse.
  - Les messages suivants entre le même client et le serveur sont envoyés à travers la même connexion ouverte.
  - 1 seul RTT pour tous les objets.

HTTP persistant

- 2 modes de connexion persistante.

- Sans pipelinage :
  - Client envoie une nouvelle requête uniquement après réception de la requête précédente.
  - Un seul RTT par objet au lieu de 2.

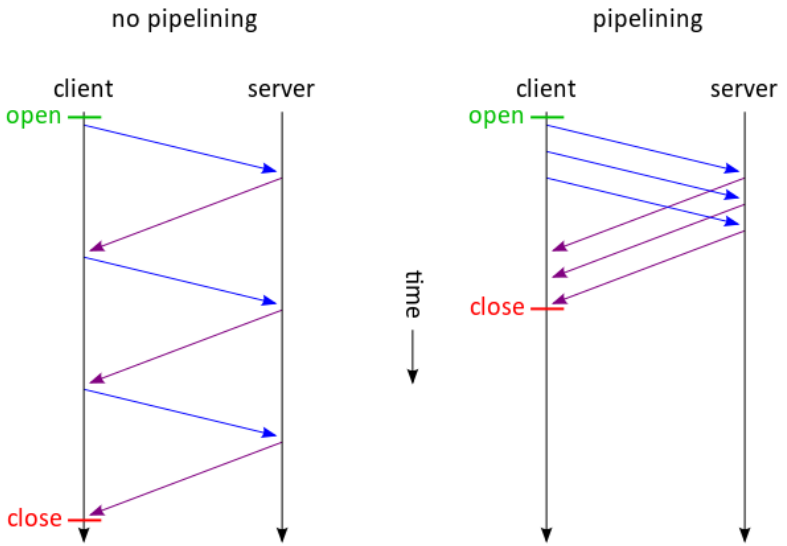


Applet

HTTP persistant

- 2 modes de connexion persistante.

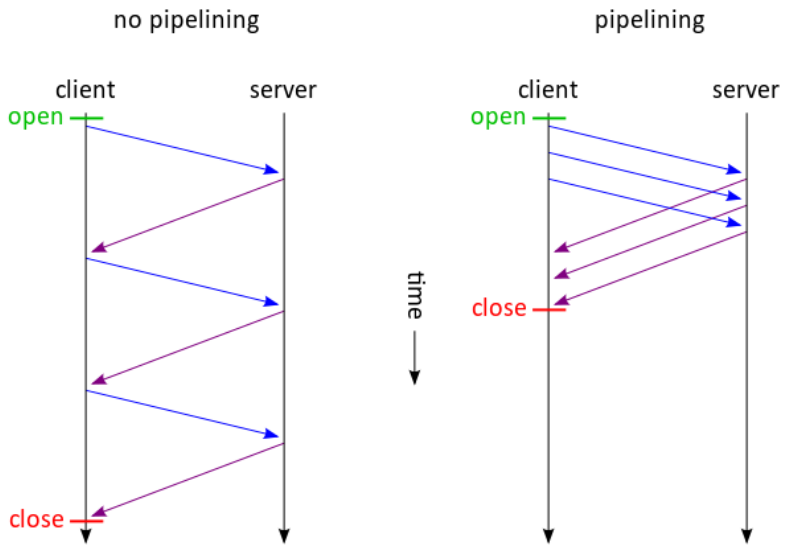
- Sans pipelinage :
  - Client envoie une nouvelle requête uniquement après réception de la requête précédente.
  - Un seul RTT par objet au lieu de 2.
- Avec pipelinage :
  - Le client fait une requête dès qu'il rencontre une référence à un objet.
  - Peut émettre une requête alors que la précédente n'est pas encore satisfaite.



HTTP persistant

- 2 modes de connexion persistante.

- Sans pipelinage :
  - Client envoie une nouvelle requête uniquement après réception de la requête précédente.
  - Un seul RTT par objet au lieu de 2.
- Avec pipelinage :
  - Le client fait une requête dès qu'il rencontre une référence à un objet.
  - Peut émettre une requête alors que la précédente n'est pas encore satisfaite.



- Peu utilisé (conflits avec TCP, le serveur doit envoyer dans l'ordre des requêtes recues : une lenteur peut tout bloquer...).
- Par défaut désactivé sur la plupart des navigateurs.

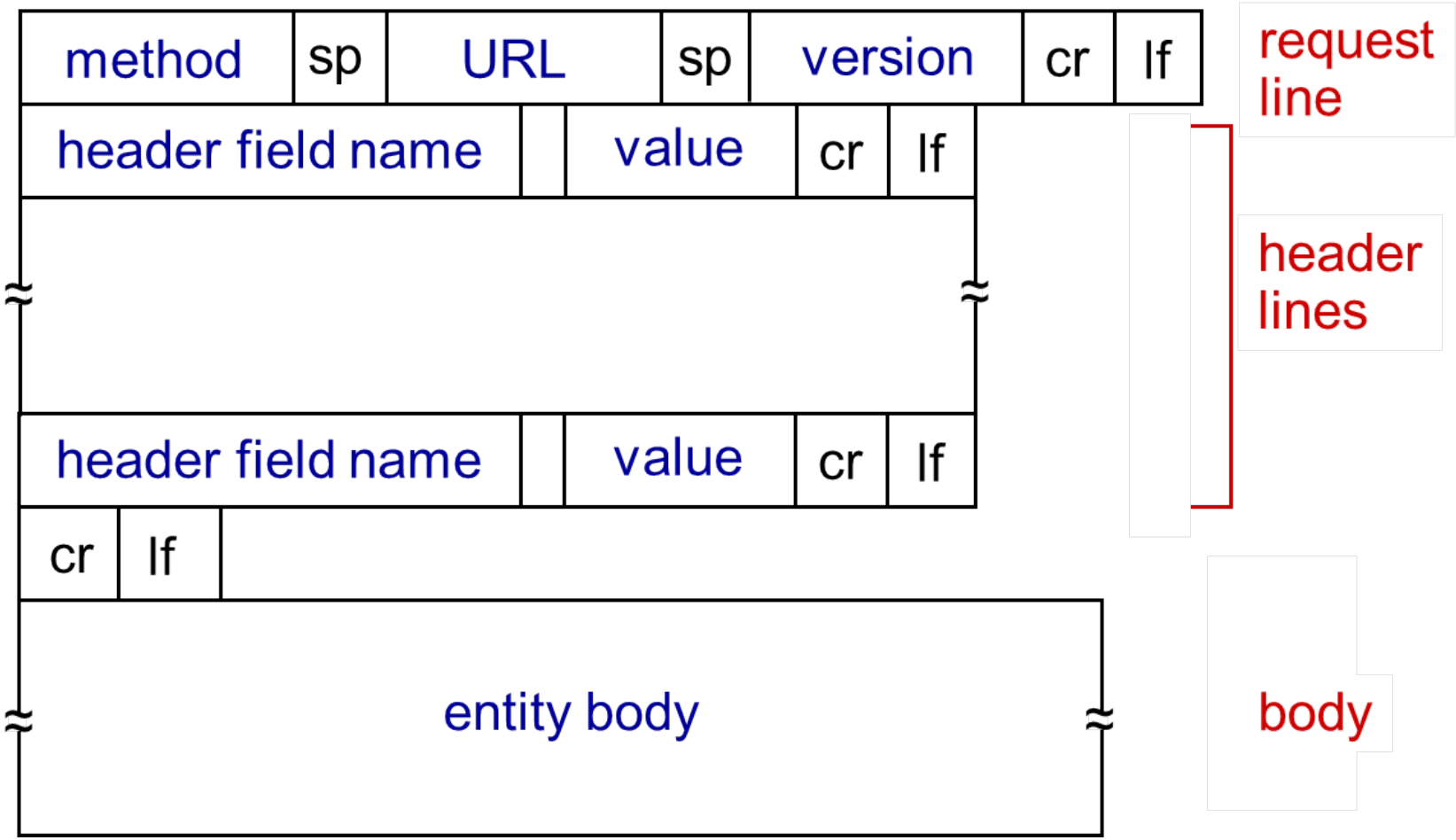
Applet

Messages HTTP

- Deux types de messages HTTP : requêtes et réponses.
- Requête (format **ASCII**) :

- Accept : ce que le client souhaite si possible, sinon par défaut.
- Connection : keep-alive, close...

Messages HTTP - Format requête



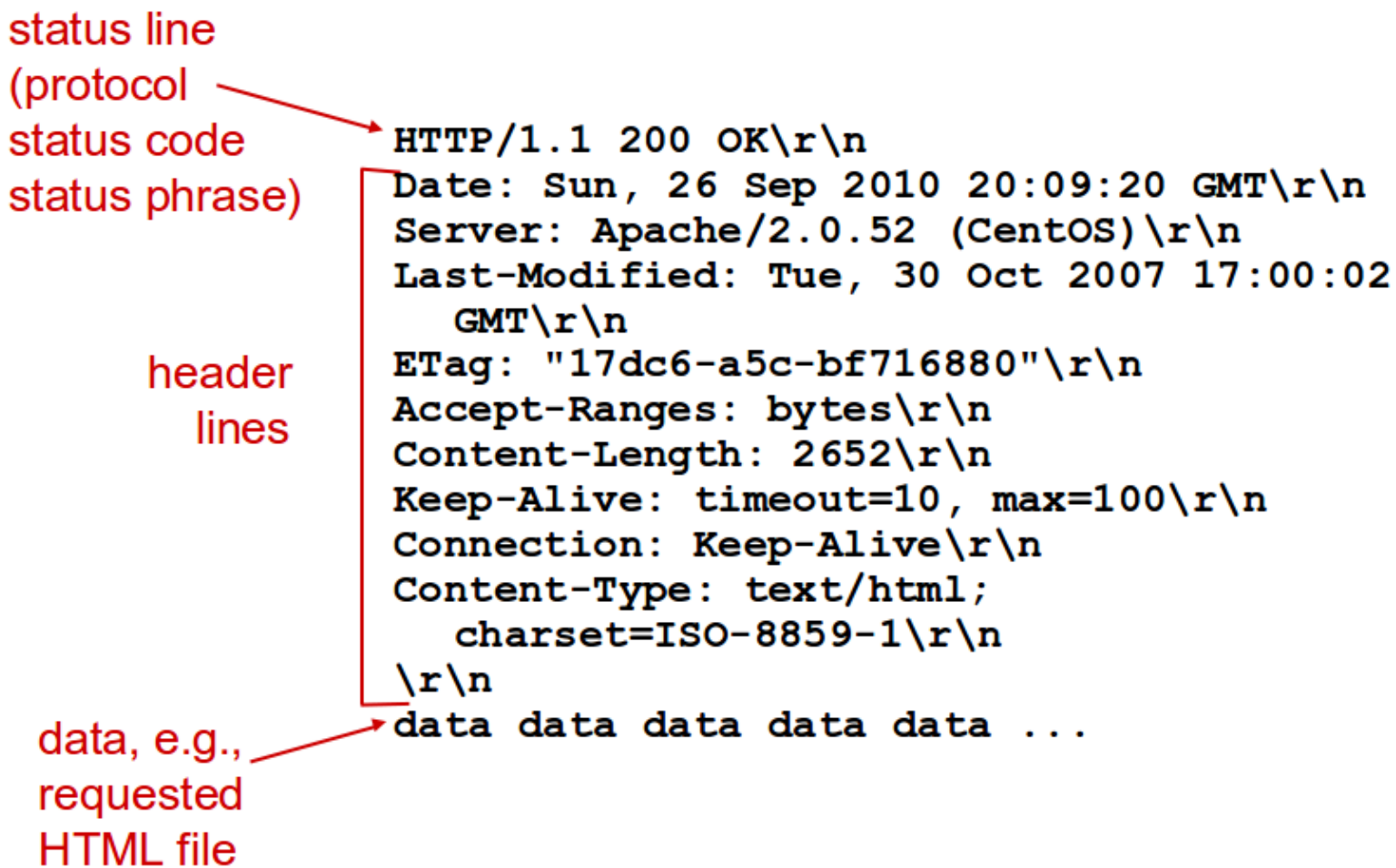
Messages HTTP - Méthodes

- GET (demande de page)
- POST
- Le “body” non utilisé pour GET (exemple précédent), utilisé pour POST.
- Head : comme GET, mais le serveur ne renvoie que l’en tête (débug).
- Uniquement HTTP 1.1 :
  - PUT (upload un objet vers un répertoire du serveur).
  - DELETE (efface un objet sur un serveur).

Messages HTTP - Types méthodes pour requêtes

- Client veut envoyer des données au serveur depuis un formulaire.
- POST :
  - Le body contient les informations des champs.
- GET :
  - Passage des informations par URL.
  - Données passées dans l'URL.
  - Exemple : `http://search.cheezburger.com/?q=lolcat`.
- POST plus "esthétique", taille non limitée (taille d'URL limités par serveur).
- GET peut être "rejoué".

Messages HTTP - Message réponse



Messages réponse : Ligne d'état

- Trois champs.
  1. Version de HTTP.
  2. Code d'état.
  3. Message d'état.
- 1xx : Informations.
- 2xx : Succès.
- 3xx : Redirections.
- 4xx : Erreurs client.
- 5xx : Erreurs serveur.
- `www.w3.org/Protocols/rfc2616/rfc2616-sec10.html` (extrait de la RFC 2616 HTTP).

Messages réponse : exemples états

- 200 OK (requête réussie, objet demandé dans le corps).
- 400 Bad Request (Demande incomprise par le serveur).
- 503 Service Unavailable (Serveur en maintenance ou surchargé).

Messages réponse : exemples états

- L'objet demandé a été déplacé définitivement. La nouvelle adresse est dans le champ Location :. Le navigateur suit.

`https://www.flickr.com/photos/girliemac/sets/72157628409467125/`

Messages réponse : exemples états

- Une authentification est nécessaire pour accéder à la ressource

Messages réponse : exemples états

- Document non trouvé

Messages réponse : exemples états

- Le client a demandé certaines caractéristiques dans ses headers (charset, language...) mais le serveur détecte qu'il ne peut pas répondre à ces conditions. N'arrive pas sur Internet car les navigateurs sont OK avec tout en général.

Messages réponse : exemples états

- Conflit avec quelque chose (par ex., upload d'un fichier plus vieux que le fichier existant...).

Messages réponse : exemples états

- URL ont une limite de taille.

Messages réponse : exemples états

- Dialogue avec une théière RFC 2324

Messages réponse : en-tête

- Champ “Content-length”.
  - Pas obligatoire avant HTTP 1.1 (juste pour vérifier que OK car connexion fermée à chaque réponse).
- Avec connexion persistante (keep-alive) : obligatoire (sauf si sans corps de message ou chunked).
  - Indispensable pour déterminer la fin du message...

HTTP par vous-même

- Testable avec *telnet*.
  - Ouvrir une connexion TCP sur le port 80 d'un serveur web.
    - telnet dauphine.fr 80
  - Tout ce qui est tapé est envoyé au serveur dauphine.fr sur le port 80.
  - Entrer une requête HTTP (exemple minimal mais suffisant).
    - GET /fr HTTP/1.1
    - Host : www.dauphine.fr
    - Deux retours chariots.
  - Le serveur répond (du code html).

HTTP par vous-même

- Testable avec *telnet*.
  - Ouvrir une connexion TCP sur le port 80 d'un serveur web.
    - telnet dauphine.fr 80
  - Tout ce qui est tapé est envoyé au serveur dauphine.fr sur le port 80.
  - Entrer une requête HTTP (exemple minimal mais suffisant).
    - GET /fr HTTP/1.1
    - Host : www.dauphine.fr
    - Deux retours chariots.
  - Le serveur répond (du code html).
- Wireshark donne également les messages envoyés.

Messages réponse chunked

- Indique “chunked” dans l’en-tête à la place de “content-length”.
- Le serveur peut ne pas savoir la taille totale qu’il va envoyer avant le début du transfert (réponses longues).
- Peut envoyer du contenu générer **dynamiquement**.
- La taille (nombre d’octets en hexadécimal) de chaque chunk (morceau) est envoyé avant le morceau lui même, fini par un 0.

Messages réponse chunked

```
telnet www.google.fr 80
GET /search?q=test HTTP/1.1
Host : www.google.fr

HTTP/1.1 200 OK
Transfer-Encoding : chunked

8000
<html><head>...
Premier morceau
3822
Deuxième morceau
4f11
Troisième morceau
0
```

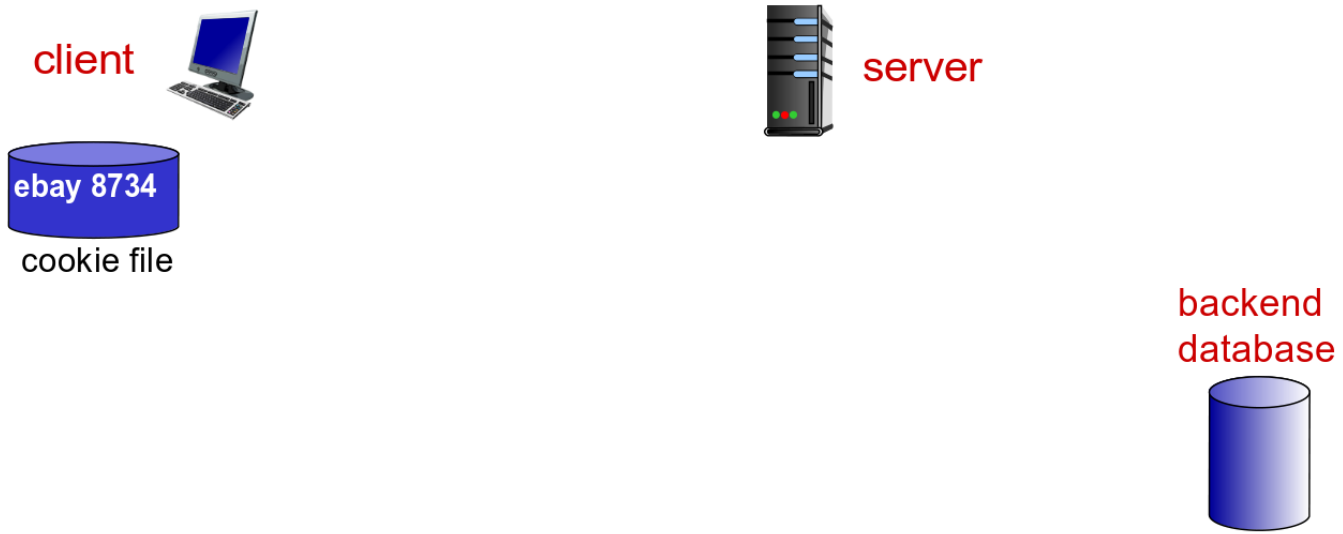
HTTP/2 (2015)

- Basé sur SPDY de Google.
- Compatible avec les versions précédentes.
- Accelération :
  - Compression des headers (ASCII en clair...).
  - Server Push (le serveur peut envoyer au client sans requêtes (des fichiers CSS etc)).
  - Pipelinage avec une seule connexion TCP + suppression de la demande d'envoi dans le même ordre des requêtes qui bloquait le pipelinage.

Cookies

- Vu précédement : serveur HTTP sans mémoire.
- Utilisation de cookies, en 4 étapes :
  1. Le serveur insère une ligne dans l'en-tête HTTP réponse (`Set-Cookie : NAME=VALUE ;...`).
  2. Le client ajoute une ligne dans sa nouvelle requête (dans l'en-tête HTTP : `Cookie : NAME=VALUE ;...`).
  3. Le client stocke un fichier texte (le cookie) géré par le navigateur pour mémoriser les informations.
  4. Le serveur en garde trace dans sa base de données.
- Paires (clé,valeur). (date d'expiration, nom de domaine...)

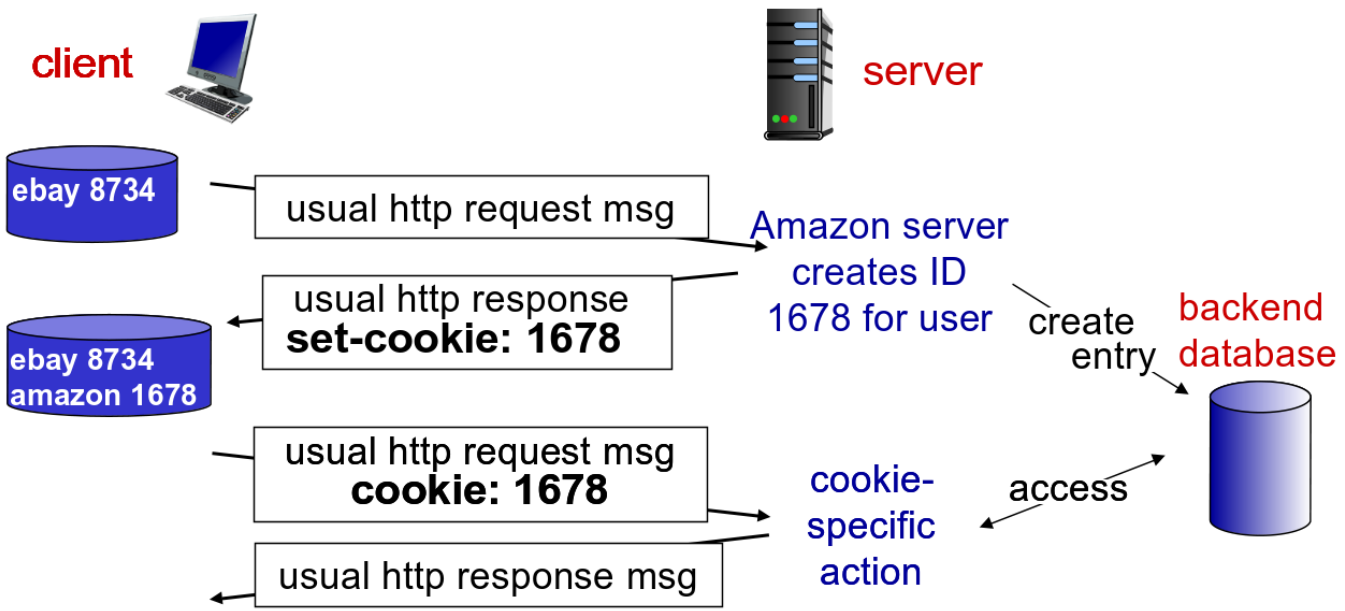
Cookies : Exemple



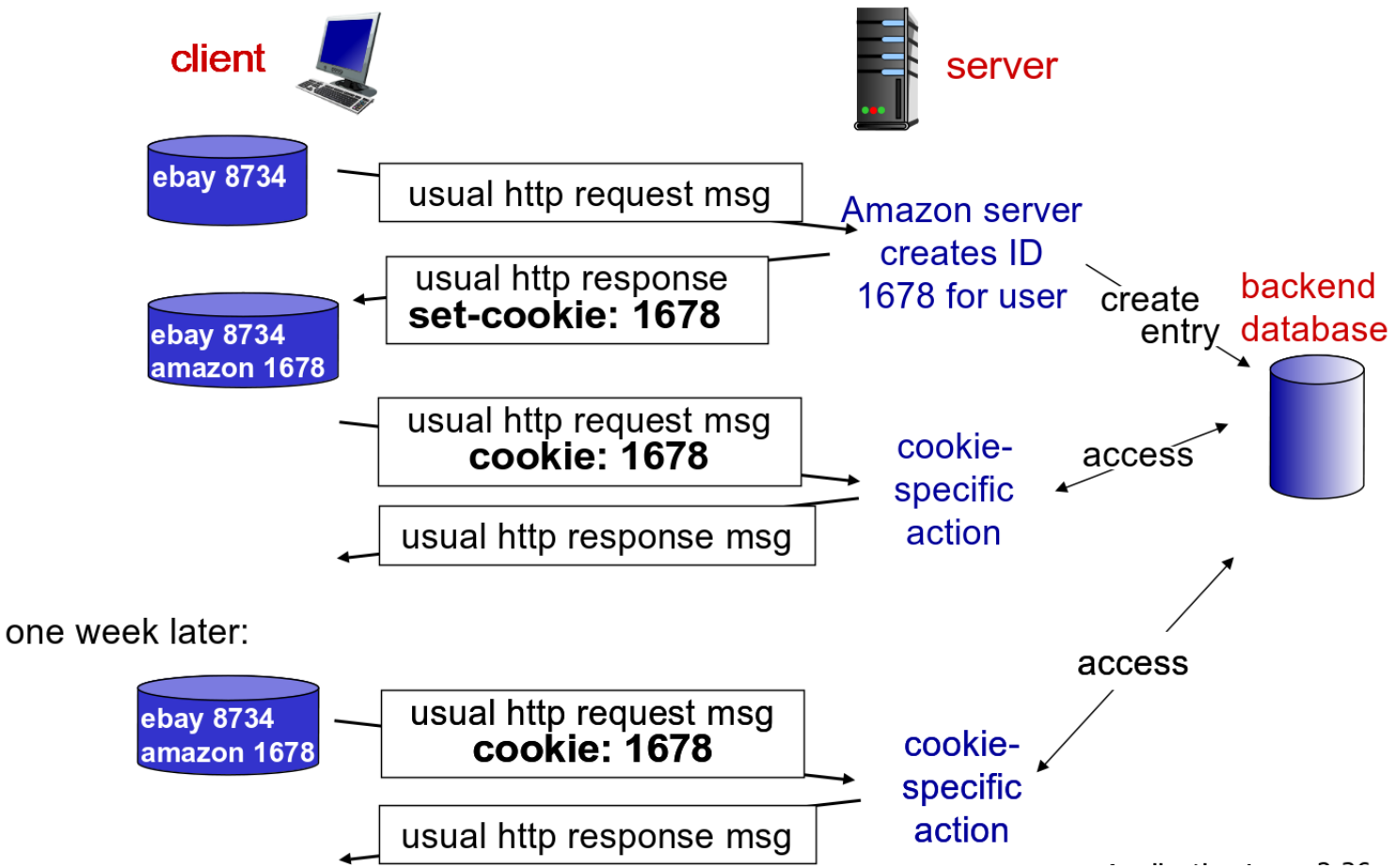
Cookies : Exemple

Cookies : Exemple

Cookies : Exemple



Cookies : Exemple



Cookies HTTP

- Requête HTTP la première fois vers google.

```
GET / HTTP/1.0
Accept-Language : fr
Accept-Encoding : gzip, deflate
Host : www.google.fr
Connection : keep-alive
```

Cookies HTTP

- Réponse du serveur HTTP de Google.

```
HTTP/1.1 200 OK
Content-Length : 1546
Server : GWS/2.0
Date : Wed, 05 Jun 2002 18 :10 :41 GMT
Content-Encoding : gzip
Content-Type : text/html
Cache-control : private
Set-Cookie : PREF=ID=4179fbc62eb90f6c : LD=fr :
   TM=1023300641 : LM=1023300641 : S=EVRhsiQ8sCc ;
   domain=.google.fr ; path=/ ;
   expires=Sun, 17-Jan-2038 19 :14 :07 GMT
```

- Serveur demande la création d'un cookie (envoi en clair).

Cookies HTTP

- Requête HTTP suivante du client.

```
GET / HTTP/1.0
Accept-Language : fr
Accept-Encoding : gzip, deflate
Host : www.google.fr
Cookie : PREF=ID=4179fbc62eb90f6c : LD=fr :
   TM=1023300641 : LM=1023300641 : S=EVRhsiQ8sCc
Connection : keep-alive
```

- Envoie ses cookies au serveur.

Cookies

Site	Name	Content	Expires
ecrans.liberation.fr	_chartbeat2	BwVNg-BhdtCsCqT-e-.1416827274706.141759...	sam. 02 déc. 2017 10:38:24 CET
ecrans.liberation.fr	_ga	GA1.3.1052060318.1376816377	ven. 02 déc. 2016 10:38:25 CET
ecrans.liberation.fr	fofirdId	ffb73b0a-ff0c-440e-92c7-2b854fc63b9c	mar. 24 nov. 2015 12:07:53 CET
ecrans.liberation.fr	tempsLectureUser	41	jeu. 07 avril 2016 00:14:43 CEST
ecrans.liberation.fr	tempsLectureUserMoyenne	41	jeu. 07 avril 2016 00:14:43 CEST
liberation.fr	__cfduid	d1664f3908f16b99ea9b83f2349a33b4014006...	mar. 24 déc. 2019 00:50:00 CET
liberation.fr	__utma	8255535.1052060318.1376816377.137681637...	mar. 18 août 2015 10:59:37 CEST
liberation.fr	_ga	GA1.2.1052060318.1376816377	ven. 01 avril 2016 17:54:18 CEST
liberation.fr	_gaos	.gaos_r=google_d_fr.mc=(no)|(no)|(no).gaos_...	dim. 04 déc. 2016 13:52:18 CET
liberation.fr	_gaost	.rk=.nv=0.r=t_d_co	dim. 04 déc. 2016 13:52:18 CET
liberation.fr	cookie_notified	true	mar. 17 févr. 2015 14:50:32 CET
liberation.fr	optimizelyBuckets	%7B%7D	jeu. 04 avril 2024 12:05:52 CEST
liberation.fr	optimizelyEndUserId	oeu1376816376492r0.7087941346466414	mer. 16 août 2023 10:59:36 CEST

Cookies

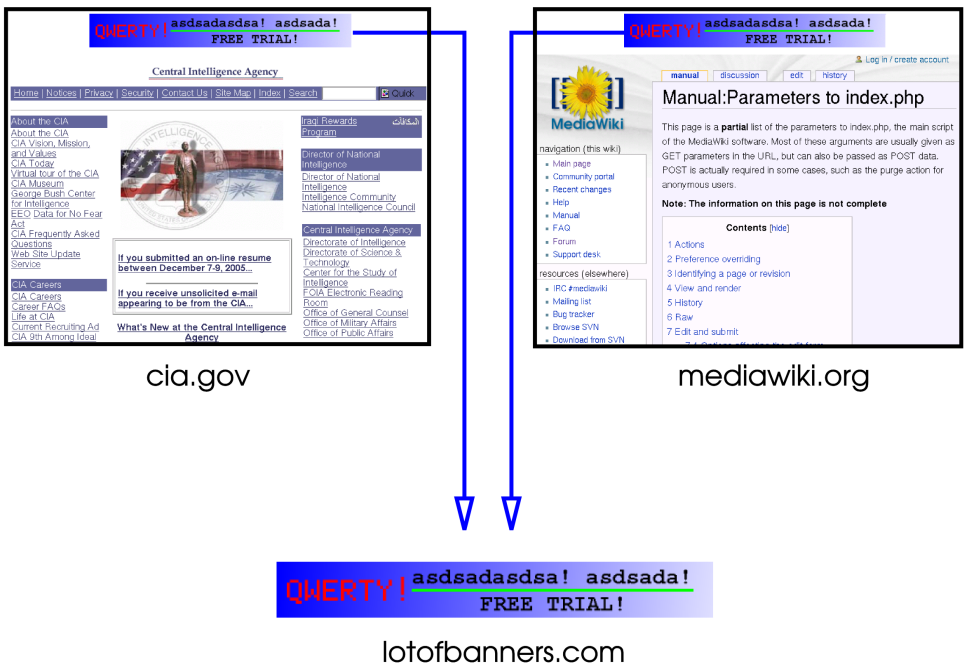
- Utilisés pour :
  - Accès limité.
  - Panier (e-commerce).
  - Recommandations, historique de visite.
  - Garder “l’état” de connexion : messagerie électronique où un cookie est envoyé continuellement pour garder l’identification.

Cookies – problèmes

- Vol de cookies
  - Par accès physique à la machine et récupère les fichiers – avec un LiveCD potentiellement).
  - Par Man-in-the-middle car transmis en clair si pas HTTPS.
- Permet de “replay”.
- Les serveurs ne doivent pas stocker les pwd en clair dans les cookies ! Ni ID de session non aléatoire. Crypter...

Cookies – problèmes

- Vie privée ?
  - Un site web peut apprendre sur le client (et le revendre...).
  - Analyse de comportement (pub ciblée) :
    - Des publicités placés sur différentes pages mais hébergés sur un même serveur publicitaire peuvent mettre en place des cookies tierce-partie (nom de domaine différent de celui visité) et pister.
  - Firefox et sa politique sur les cookies tiers.



cia.gov mediawiki.org

lotofbanners.com

Proxy cache

- But : répondre au client **sans utiliser le serveur d'origine**.

Proxy cache

- But : répondre au client **sans utiliser le serveur d'origine**.
- Client indique à son navigateur d'accéder au Web via le proxy cache.

Proxy cache

- But : répondre au client **sans utiliser le serveur d'origine**.
- Client indique à son navigateur d'accéder au Web via le proxy cache.
- Le navigateur envoie toutes ses requêtes HTTP au proxy cache.
  - Si l'objet demandé est dans le cache : retourner l'objet.
  - Sinon, le proxy demande l'objet au serveur d'origine, le copie et le retourne au client.

Proxy cache

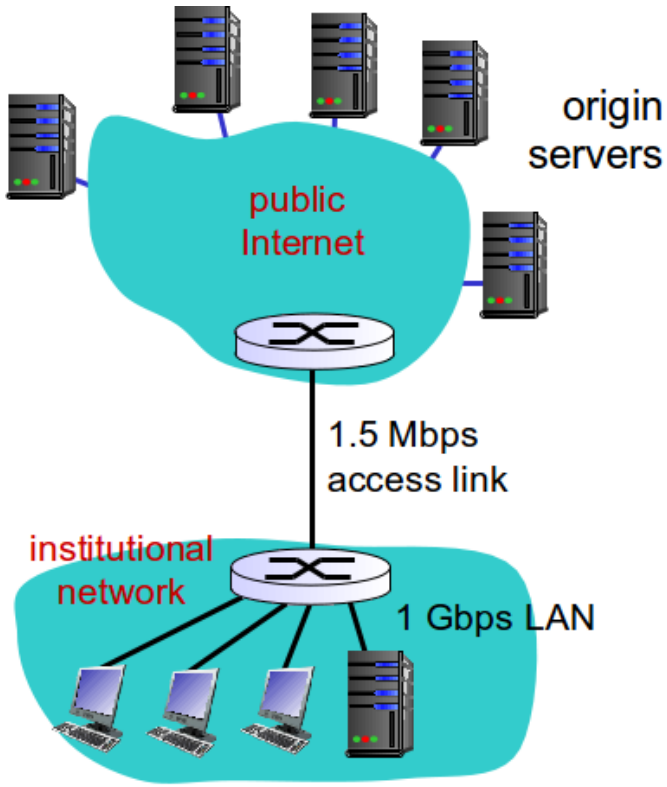
- Un proxy cache agit à la fois comme un client et comme un serveur.
  - Serveur pour le client.
  - Client pour le serveur d'origine.
- Souvent installé par un FAI (et université, entreprise...)

Proxy cache : Pourquoi ?

- Réduire le temps de réponse pour un client.
- Réduire le trafic sur la liaison d'une institution.
- Permet à des producteurs de contenu "pauvres" d'être tout de même efficace (comme pour du P2P...).

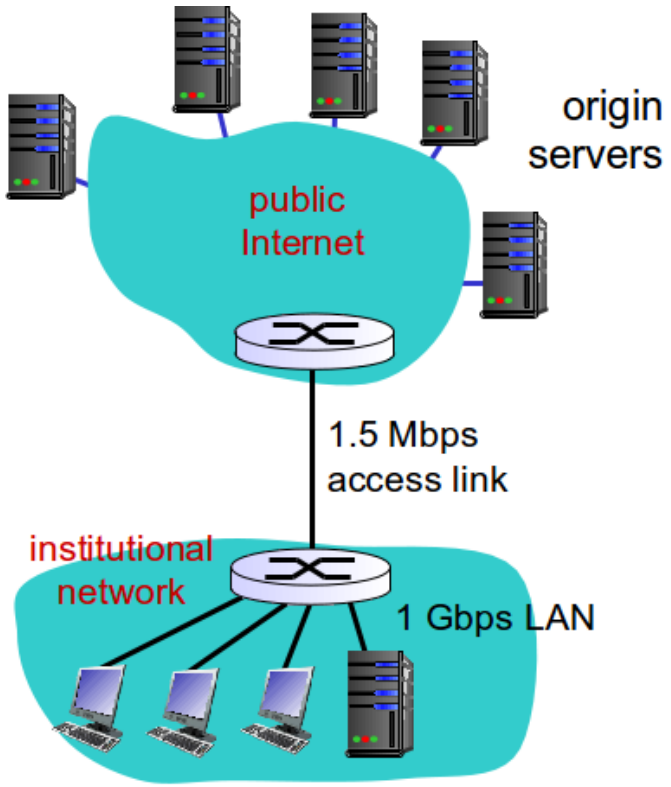
Proxy cache : exemple

- Suppositions :
  - Taille moyenne des objets : 100 Kbits.
  - Nombre moyen de requêtes : 15/sec.
  - RTT entre routeur intitu. et autres routeurs : 2sec.
  - Bande passante LAN : 10 Mbps.
  - Bande passante liaison d'accès : 1.5 Mbps.



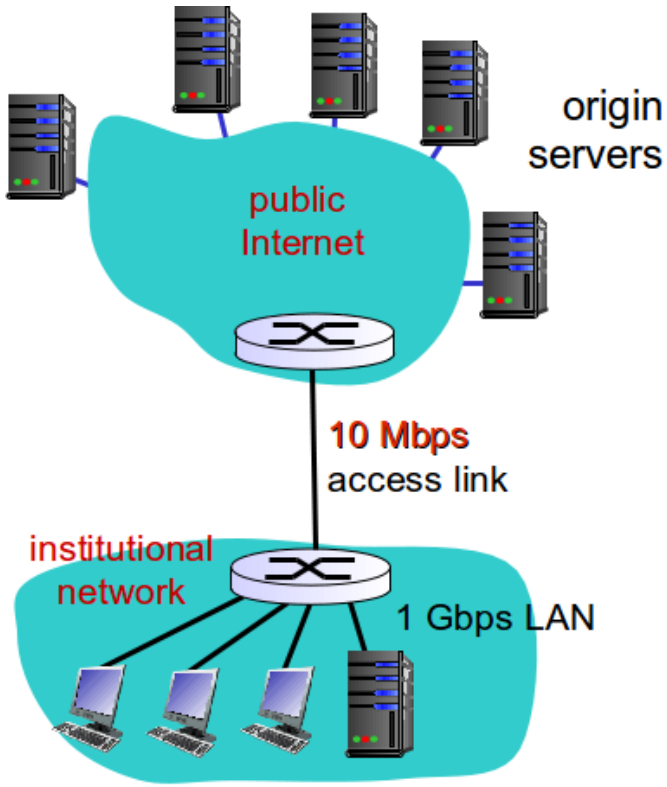
Proxy cache : exemple

- Suppositions :
  - Taille moyenne des objets : 100 Kbits.
  - Nombre moyen de requêtes : 15/sec.
  - RTT entre routeur intitu. et autres routeurs : 2sec.
  - Bande passante LAN : 10 Mbps.
  - Bande passante liaison d'accès : 1.5 Mbps.
- Conséquences :
  - Utilisation du LAN : 15% (# requêtes · taille / bande passante LAN)
  - Utilisation liaison d'accès : **100%**
  - Délai total : délai Internet + délai accès + délai LAN = 2sec + minutes + μs.



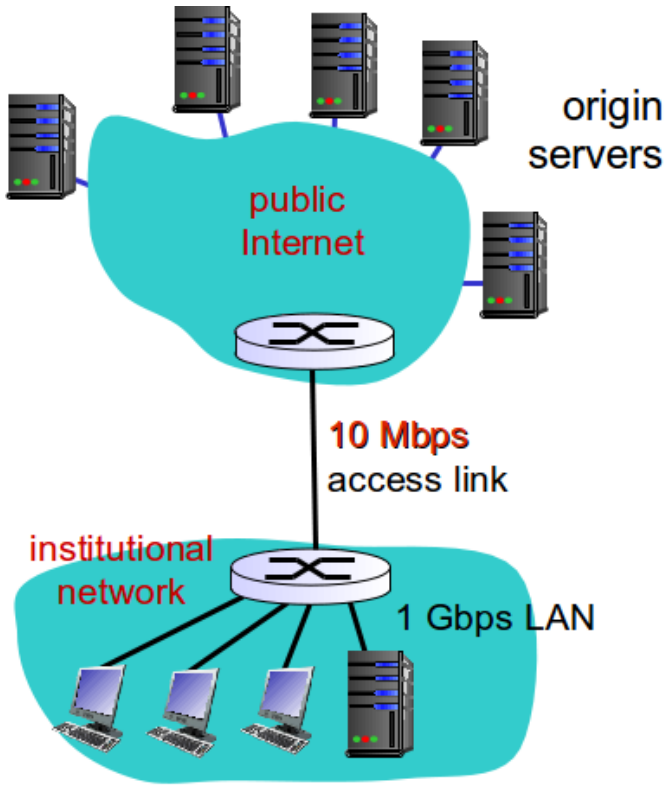
Proxy cache : exemple : Augmenter lien d'accès

- Suppositions :
  - Taille moyenne des objets : 100 Kbits.
  - Nombre moyen de requêtes : 15/sec.
  - RTT entre routeur intitu. et autres routeurs : 2sec.
  - Bande passante LAN : 10 Mbps.
  - Bande passante liaison d'accès : **10** Mbps.



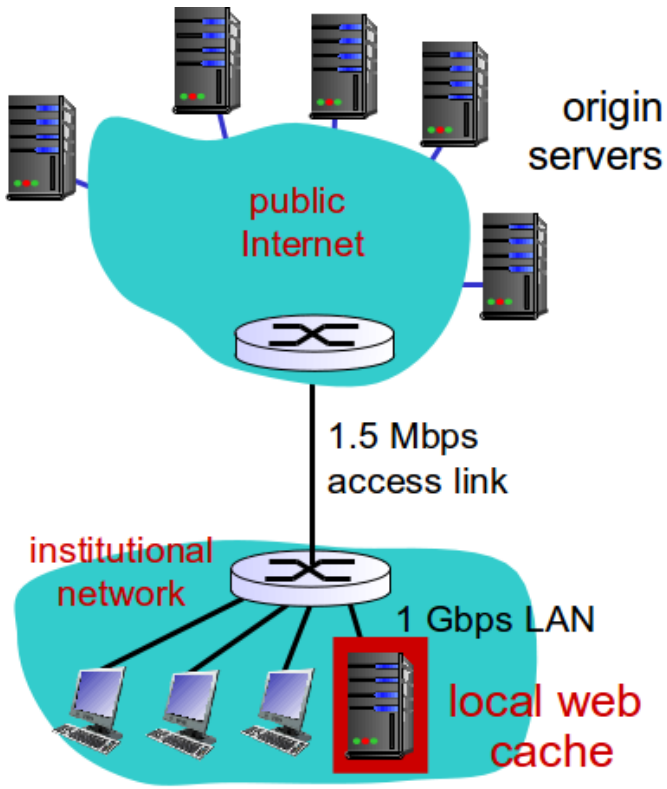
Proxy cache : exemple : Augmenter lien d'accès

- Suppositions :
  - Taille moyenne des objets : 100 Kbits.
  - Nombre moyen de requêtes : 15/sec.
  - RTT entre routeur intitu. et autres routeurs : 2sec.
  - Bande passante LAN : 10 Mbps.
  - Bande passante liaison d'accès : **10** Mbps.
- Conséquences :
  - Utilisation du LAN : 15% (# requêtes · taille / bande passante LAN)
  - Utilisation liaison d'accès : **15%**
  - Délai total : délai Internet + délai accès + délai LAN = 2sec + **msec** + μs.
- Coût : Augmenter la capacité du lien d'accès : cher !



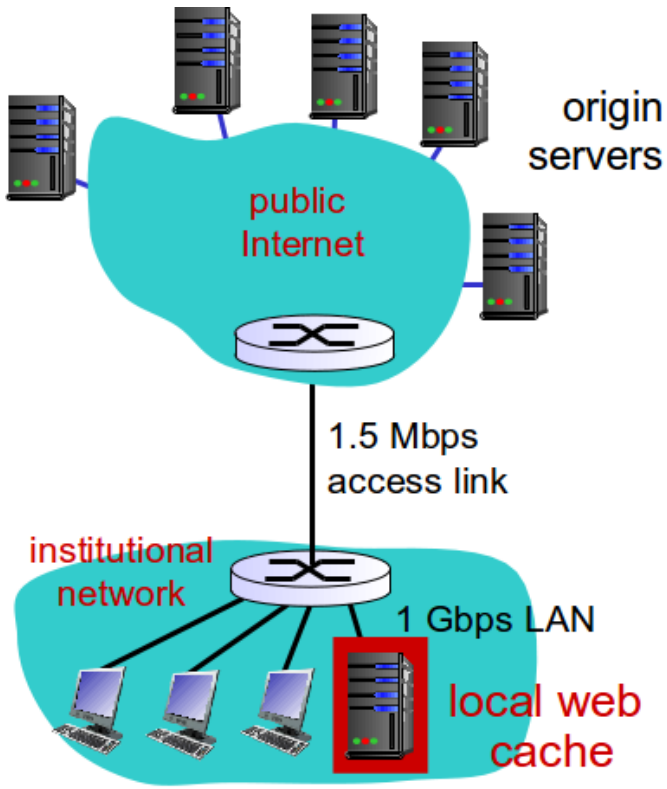
Proxy cache : exemple Proxy Cache

- Suppositions :
  - Taille moyenne des objets : 100 Kbits.
  - Nombre moyen de requêtes : 15/sec.
  - RTT entre routeur intitu. et autres routeurs : 2sec.
  - Bande passante LAN : 10 Mbps.
  - Bande passante liaison d'accès : **1.5** Mbps.



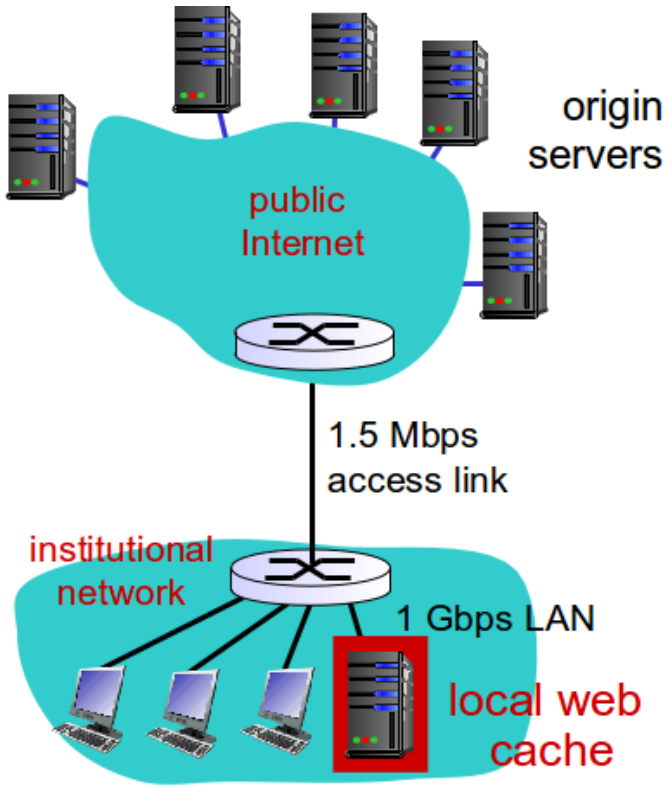
Proxy cache : exemple Proxy Cache

- Suppositions :
  - Taille moyenne des objets : 100 Kbits.
  - Nombre moyen de requêtes : 15/sec.
  - RTT entre routeur intitu. et autres routeurs : 2sec.
  - Bande passante LAN : 10 Mbps.
  - Bande passante liaison d'accès : **1.5** Mbps.
- Conséquences :
  - Utilisation du LAN : 15% (# requêtes · taille / bande passante LAN)
  - Utilisation liaison d'accès : **?**
  - Délai total : **?**
  - Coût : Un serveur cache : pas cher !



Proxy cache : exemple Proxy Cache

- Calcul du délai.
  - Supposons un taux de réussite de 0.4 pour le cache.
  - 40% des requêtes traitées en *ms* (même LAN).
  - Intensité du trafic sur la liaison : passe de 100% à 60% $\rightarrow$ délai de *ms*.
  - Délai total : $0.4 \cdot ms + 0.6 \cdot 2.01$
- Moins cher et plus performant que la solution précédente.

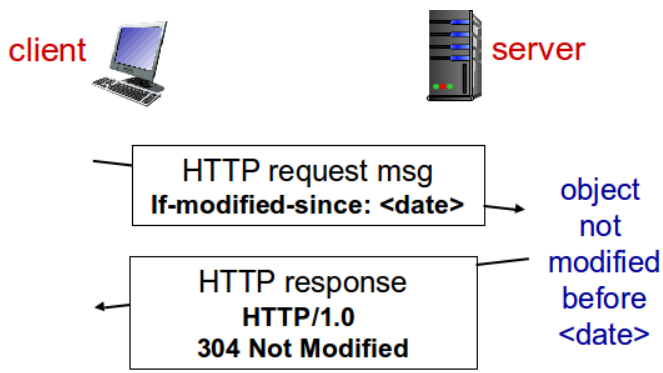


GET conditionnel

- Problème du cache (valable aussi en local) : un objet mis en cache peut être périmé (modifié sur le serveur original).

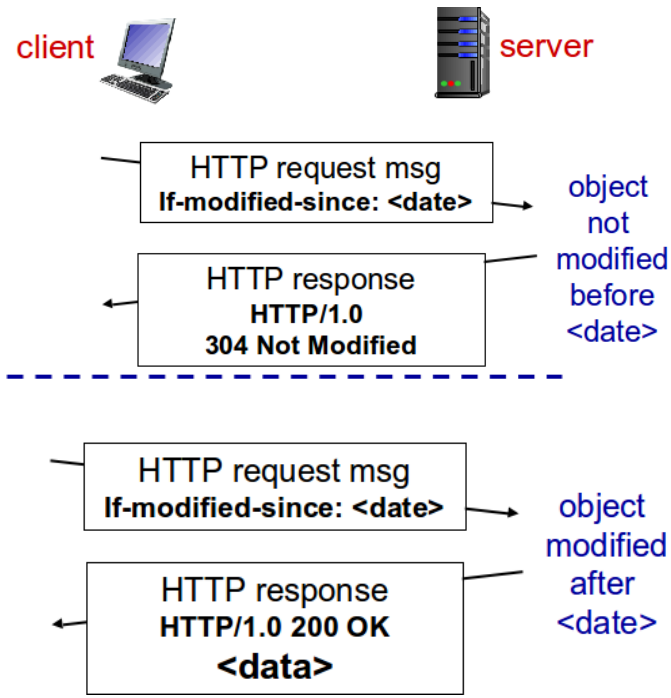
GET conditionnel

- Problème du cache (valable aussi en local) : un objet mis en cache peut être périmé (modifié sur le serveur original).
- Le cache spécifie la date de sa copie dans sa requête avec le champ `If-modified-since : <date>` dans l'en tête.



GET conditionnel

- Problème du cache (valable aussi en local) : un objet mis en cache peut être périmé (modifié sur le serveur original).
- Le cache spécifie la date de sa copie dans sa requête avec le champ `If-modified-since : <date>` dans l'en tête.
- Le serveur renvoie l'objet uniquement s'il possède une version plus récente.



Apparté : le “cloud”

- Terme essentiellement marketing (comme Web2.0).
- Ensemble de services offerts (souvent déjà connus) de manière “cachée” au client (virtualisation, mutualisation...).
- Le client ne s’occupe plus de l’administration (fastidieuse) de serveurs, bdd... mais accède à des services, sans savoir où se trouve telle ou telle infrastructure (datacenters).
- Permet de s’affranchir de fichiers de configuration, de tâches d’administration, de problèmes de scaling...

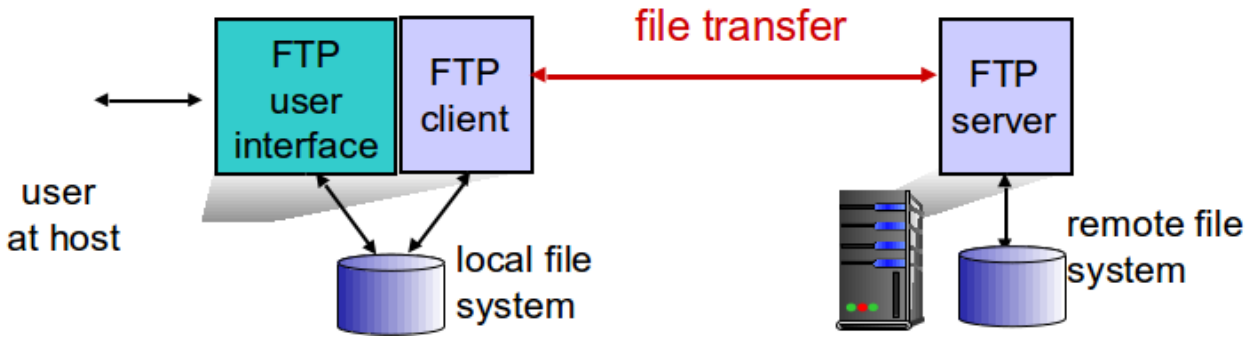
- Vision centralisée en quelque sorte contraire des racines d’internets “libertaires” et décentralisées.

Apparté : le “cloud”

- Officiellement : j’achète, je paye une fois beaucoup ⇒ je loue et paye ce que je consomme.
- Officieusement : j’achète, je paye une fois pour toutes, je remplace quand je veux ⇒ je paye un peu mais tout le temps et longtemps.

Cours 2 : Couche applications

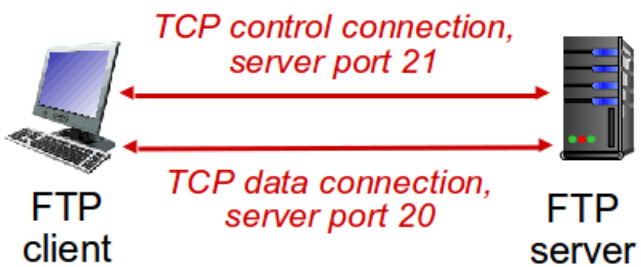
FTP : File Transfert Protocol



- **Transfert de fichier** vers/depuis un hôte distant.
- Modèle client/serveur.
  - Plusieurs clients FTP existent (FileZilla, gFTP...).
  - Plusieurs logiciels serveurs FTP existent.
- RFC 959 (issu de la RFC 765, rendant elle même obsolète la RFC 114, 1971 (plus vieux que HTTP)).
- Serveur par défaut sur le port 21.

FTP : File Transfert Protocol

- Utilise TCP comme HTTP.
- Mais 2 connexions TCP en parallèle (contrôle “hors bande”) :
  - Une de contrôle (identification, commandes...).
  - Une de données : envoi du fichier.
- La connexion de données est ouverte par le serveur vers le client lorsqu'une demande de transfert est reçue.
- Fermée après le transfert d'un fichier - une autre sera ouverte si un autre fichier est demandé.
- Le serveur maintient l'état (répertoire courant, identification...)



FTP - Quelques commandes

- Empruntent la connexion de contrôle.
- Format ASCII.

- USER username (transmet le nom d'utilisateur).
- PASS password (transmet le mot de passe). (FTPS : FTP over SSL...)
- LIST (demande le listage des fichiers contenus dans le répertoire courant sur le serveur distant (envoyé sur la connexion de données)).
- RETR filename (demande le téléchargement du fichier (sur la connexion de données)).
- STOR filename (Enregistre un fichier dans le répertoire distant).

FTP - Queleques réponses serveur

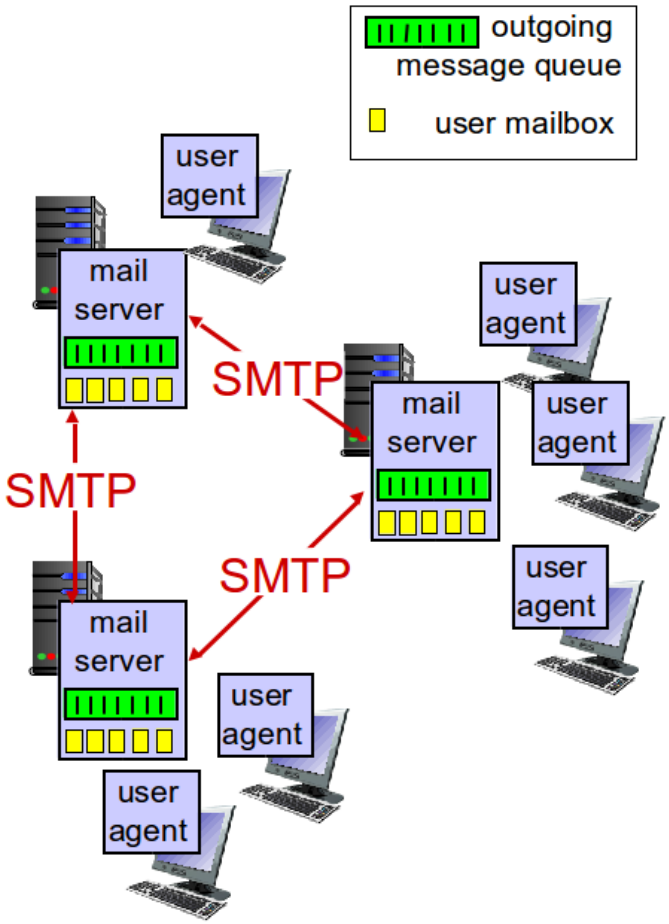
- Serveur répond à l'utilisateur.
- Code + phrase d'état similaire à HTTP.

- 331 Username OK, password required.
- 125 Data connection already open ; transfert starting.
- 425 Can't open data connection.
- 452 Error writing file.

Cours 2 : Couche applications

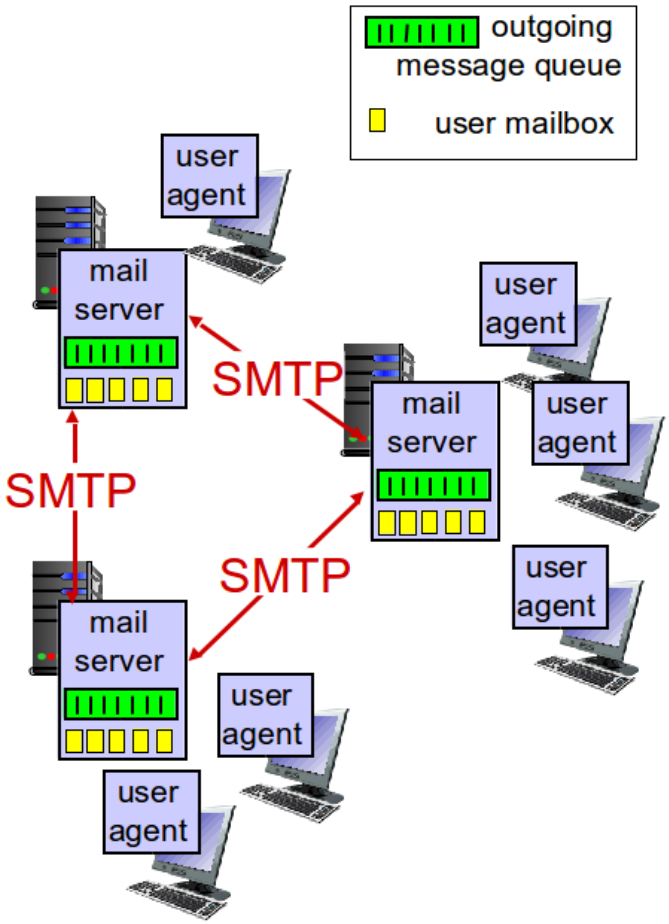
Courrier électronique

- Trois acteurs majeurs :
  - Agent utilisateur.
  - Serveur de messagerie.
  - Simple Mail Transfer Protocol : SMTP (RFC 2821).
- Agent utilisateur (logiciel de messagerie) :
  - Écrire, éditer, lire messages.
  - Thunderbird, client mail iPhone...
  - Messages sortant et entrants stockés sur serveur.



Courrier électronique

- Serveur de messagerie :
  - Boîte aux lettres contenant les messages entrants des utilisateurs.
  - File de messages en attente d'envoi (essai réguliers si echec).
- SMTP :
  - Protocole entre les serveurs mail pour envoyer un message.
    - Client : le serveur qui envoie le mail.
    - Serveur : le serveur qui reçoit le mail.



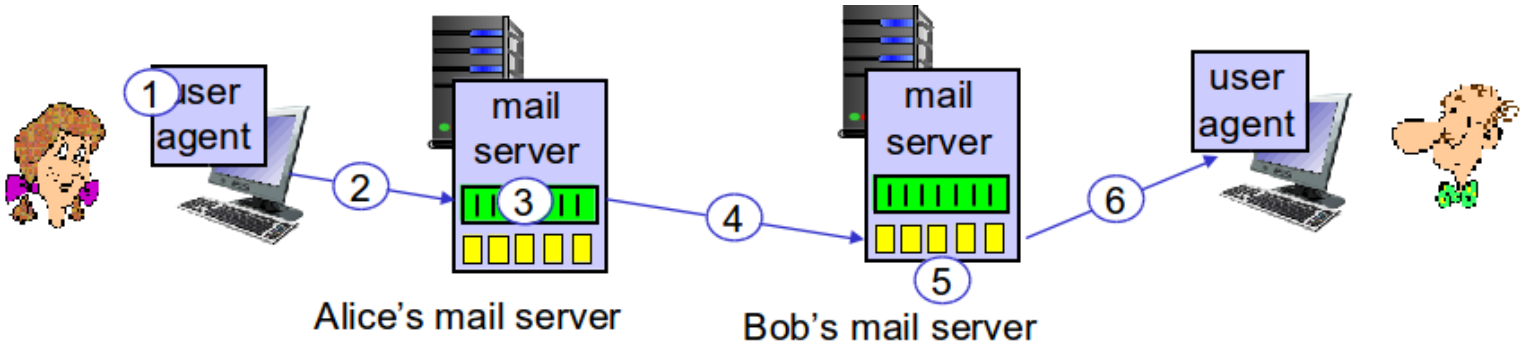
SMTP

- Utilise TCP pour envoyer un message sans pertes d'un client à un serveur. (1982, plus ancien que HTTP)
- Port 25.
- Trois phases :
  - "Handshaking".
  - Transfert.
  - Fermeture.

SMTP

- Utilise TCP pour envoyer un message sans pertes d'un client à un serveur. (1982, plus ancien que HTTP)
- Port 25.
- Trois phases :
  - "Handshaking".
  - Transfert.
  - Fermeture.
- Interactions commandes/réponses (comme HTTP ou FTP).
  - Commandes en ASCII.
  - Réponses avec code d'état et une phrase.
- Messages doivent être en ASCII 7-bits (il fallait épargner la bande passante à l'époque, un peu archaïque). $\rightarrow$ Images doivent être codées !

SMTP - exemple



1. Alice utilise son logiciel de messagerie pour écrire un message à bob@dauphine.fr.
2. Le logiciel d'Alice envoie le message au serveur de mails, qui est placé dans la file.
3. La partie "client" du serveur SMTP d'Alice ouvre une connexion TCP avec le serveur mail de Bob.
4. Le client SMTP envoie le message sur la connexion TCP.
5. Le serveur SMTP de Bob place le message dans sa boîte au lettre.
6. Bob utilise son logiciel pour lire le message.

SMTP - exemple

```
S : 220 mta.dauphine.fr ESMTP Sendmail 8.13.6
C : HELO miage.fr
S : 250 mta2.dauphine.fr
C : MAIL FROM : <francois.hollande@elysee.fr>
S : 250 2.1.0 Ok
C : RCPT TO : <florian.sikora@dauphine.fr>
S : 250 2.1.5 Ok
C : DATA
S : 354 End data with <CR><LF>.<CR><LF>
C : Subject : Blabliblu
C : Les cours
C : endorment...
C : .
S : 250 2.0.0 Ok : queued as A7AD380095
C : QUIT
S : 221 2.0.0 Bye
```

- Testable avec telnet (telnet smtp.dauphine.fr 25, ou telnet smtp.sfr.fr 25).
- Email spoofing facile donc !

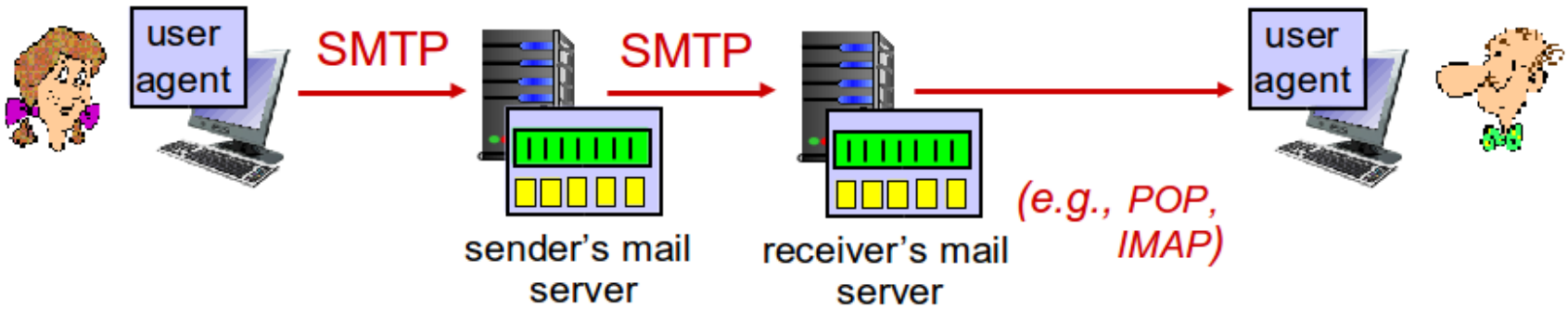
Format de messages

- SMTP : protocole pour l'échange de messages.
- RFC 822 : standard pour le format de messages texte.
- Lignes d'en-tête (comme HTTP...). A ajouter au moment du DATA.
  - To :
  - From :
  - Subject :
  - ...
  - Champs différents des commandes SMTP MAIL FROM et RCPT TO (procédure de présentation de SMTP).
- Le corps du message.
  - Seulement ASCII.

MIME (Multi-Purpose Internet Mail Extensions)

- Standard (donc RFC...) permettant d'utiliser des contenus non ASCII 7bits.
  - Autres langues que l'anglais (accents...).
  - Contenu multimedia (images, vidéos...).
  - ...
- Encode/décode un contenu vers/depuis ASCII.
- Exemples :
  - `=?utf-8?Q?=C2=A1Hola,_se=C3=B1or!?=` → ¡Hola, señor !
  - Content-Type : application/pdf ; name="cv.pdf" Content-Transfer-Encoding : base64 Content-Disposition : attachment ; filename="structural.pdf"
    JVBERi0xLjUKJdDUxdgKNCAwIG9iago8PCAvUyAvR29UbyAvRCAoc2
    bmRvYmoKNyA-
    wIG9iagooKQplbmRvYmoKOCAwIG9iago8PCAvUyAvR29UbyAvRCBbC
    L0ZpdF0gPj4KZW5kb2JqCjExIDAgb2JqIDw8Ci9MZW5ndGggMjE1Ny

Protocoles d'accès à la messagerie



- SMTP : Délivre et stocke le message sur le serveur du destinataire.
- Protocole d'accès à la messagerie : récupérer le message depuis le serveur.
  - POP (Post Office Protocole, RFC 1939). Identification et téléchargement.
  - IMAP (Internet Mail Access Protocol, RFC 1730). Plus de commandes, comme la manipulation de messages sur le serveur.
  - HTTP : gmail, hotmail, yahoo...

POP3

- Phase d'identification.
  - Commandes client (user, pass).
  - Réponse du serveur (+OK, -ERR).

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

POP3

- Phase d'identification.
  - Commandes client (user, pass).
  - Réponse du serveur (+OK, -ERR).

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

- Phase de transaction.
  - list : liste le nombre de messages.
  - retr id : récupère le message id.
  - ...

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3

- Exemple précédent en mode “download and delete”.
  - Impossible de relire un email si on change de client.
- Mode “download and keep” : permet de lire sur plusieurs clients.
  - Stockage et organisation en dossiers en local.
- Pas de mémoire d'une session à une autre.

IMAP (RFC 2060)

- Garde tous les messages à un endroit : le serveur.
- Les utilisateurs peuvent organiser leurs messages dans des dossiers.
- Mémoire de l'état sur plusieurs sessions (conserve le nom des dossiers et leurs contenus.).

Webmails

- 1995-1996 : start-up HoTMaiL.
  - Idée : proposer un service de messagerie gratuit, accessible de partout.
  - 3 employés plein temps, 14 à mi-temps payés en stock options.
  - 100 000 abonnés en 1 mois (marketting viral, pub à chaque courrier (stoppé en 2010)).
  - Racheté par Microsoft en décembre 1997 (12 millions d'abonnés) pour 400 millions de dollars. 330 millions d'abonnés en 2011.
  - Dépassé par gmail en 2012.

Webmails

- 1995-1996 : start-up HoTMaiL.
  - Idée : proposer un service de messagerie gratuit, accessible de partout.
  - 3 employés plein temps, 14 à mi-temps payés en stock options.
  - 100 000 abonnés en 1 mois (marketting viral, pub à chaque courrier (stoppé en 2010)).
  - Racheté par Microsoft en décembre 1997 (12 millions d'abonnés) pour 400 millions de dollars. 330 millions d'abonnés en 2011.
  - Dépassé par gmail en 2012.
- Depuis proposé par d'autres services, les universités, les entreprises...
- Accès au serveur via HTTP, "client" est le navigateur, envoi de courrier via HTTP.
- Échanges entre autres serveurs reste en SMTP.

Cours 2 : Couche applications

DNS : Domain Name System

- Humain : beaucoup d'identifiant.
  - Numéro de sécu., nom, numéro de passeport...
  - Services comme impôts préfèrent un numéro (taille fixe, pas de confusion possible...) pour identifier.
  - Mais utilisation de noms dans la vie de tous les jours.

DNS : Domain Name System

- Humain : beaucoup d'identifiant.
  - Numéro de sécu., nom, numéro de passeport...
  - Services comme impôts préfèrent un numéro (taille fixe, pas de confusion possible...) pour identifier.
  - Mais utilisation de noms dans la vie de tous les jours.
- Hôtes Internet, routeurs :
  - Adresse IP.
  - "Noms" (par ex. `www.dauphine.fr`) utilisés par les humains.
  - Routeurs ont besoin de l'IP (taille fixe, donne l'emplacement...).
  - Utilisateur se rappellent plus facilement d'un nom.

DNS : Domain Name System

- Humain : beaucoup d'identifiant.
  - Numéro de sécu., nom, numéro de passeport...
  - Services comme impôts préfèrent un numéro (taille fixe, pas de confusion possible...) pour identifier.
  - Mais utilisation de noms dans la vie de tous les jours.
- Hôtes Internet, routeurs :
  - Adresse IP.
  - "Noms" (par ex. `www.dauphine.fr`) utilisés par les humains.
  - Routeurs ont besoin de l'IP (taille fixe, donne l'emplacement...).
  - Utilisateur se rappellent plus facilement d'un nom.
- Comment faire correspondre adresses IP et noms (et *vice versa*) ?

DNS : Domain Name System

- Humain : beaucoup d'identifiant.
  - Numéro de sécu., nom, numéro de passeport...
  - Services comme impôts préfèrent un numéro (taille fixe, pas de confusion possible...) pour identifier.
  - Mais utilisation de noms dans la vie de tous les jours.
- Hôtes Internet, routeurs :
  - Adresse IP.
  - "Noms" (par ex. `www.dauphine.fr`) utilisés par les humains.
  - Routeurs ont besoin de l'IP (taille fixe, donne l'emplacement...).
  - Utilisateur se rappellent plus facilement d'un nom.
- Comment faire correspondre adresses IP et noms (et *vice versa*) ?
- Jusqu'en 1982, fichier texte HOST.txt maintenu par Stanford et copié sur chaque hôte...

DNS : Domain Name System

- Service d'annuaire d'Internet.
- Couche application.
- Base de données distribuée, hiérarchie de serveurs de noms.
- Serveurs communiquent entre eux, souvent en UDP, port 53.
- DNS sollicité par d'autres protocoles (HTTP, FTP, SMTP...).

DNS : Domain Name System

- Service d'annuaire d'Internet.
- Couche application.
- Base de données distribuée, hiérarchie de serveurs de noms.
- Serveurs communiquent entre eux, souvent en UDP, port 53.
- DNS sollicité par d'autres protocoles (HTTP, FTP, SMTP...).
  - Un client HTTP demande l'URL `http://www.reddit.com/r/fffffffuuuuuuuuuuuu/`.
  - La navigateur doit connaitre l'adresse IP associée
    - Extrait le nom du serveur.
    - Fait une requête DNS sur ce nom.
    - Reçoit en réponse l'IP correspondante.
  - Connexion TCP (pour HTTP) vers le serveur correspondant à l'adresse IP.

DNS - Services

- Services :
  - Traduction d'adresse.
  - Alias (le serveur Web peut être un nom complexe).
  - Alias de serveur de messagerie (le nom d'hôte du serveur de messagerie gmail pourrai être relay1.west-coast.gmail.com).
  - Répartition de charge (Serveur web répliqué : plusieurs adresses IP avec le même contenu. DNS fait une rotation de l'ordre des adresses IP associées au site).

DNS - Structure

- Pourquoi ne pas centraliser DNS ?

DNS - Structure

- Pourquoi ne pas centraliser DNS ?
  - Fragilité. Si panne, plus d'Internet !
  - Volume de trafic.
  - Éloignement. Certains hôtes devraient traverser le monde...
  - Maintenance, identification...

DNS - Structure

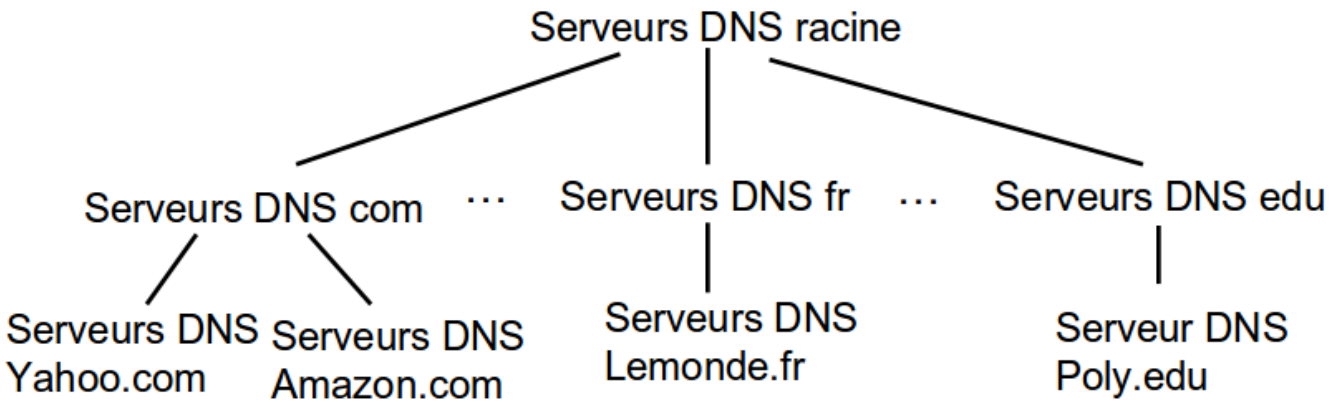
- Pourquoi ne pas centraliser DNS ?
  - Fragilité. Si panne, plus d'Internet !
  - Volume de trafic.
  - Éloignement. Certains hôtes devraient traverser le monde...
  - Maintenance, identification...
- Pas à la hauteur de cette tâche !
- Bon exemple de base de données distribuée à travers Internet.

DNS - serveur local

- Chaque FAI, université... possède un serveur DNS local (adresse accessible dans les options de configuration).
- Serveur “proche” du client, réponse rapide.
- Quand un client fait une requête DNS, elle est d’abord envoyée au serveur DNS local.
  - Le serveur local possède un cache de correspondances nom/IP récentes (peut être non à jour).
  - Agît comme un proxy : devient un client s’il ne peut pas répondre.

DNS - Structure hierarchique

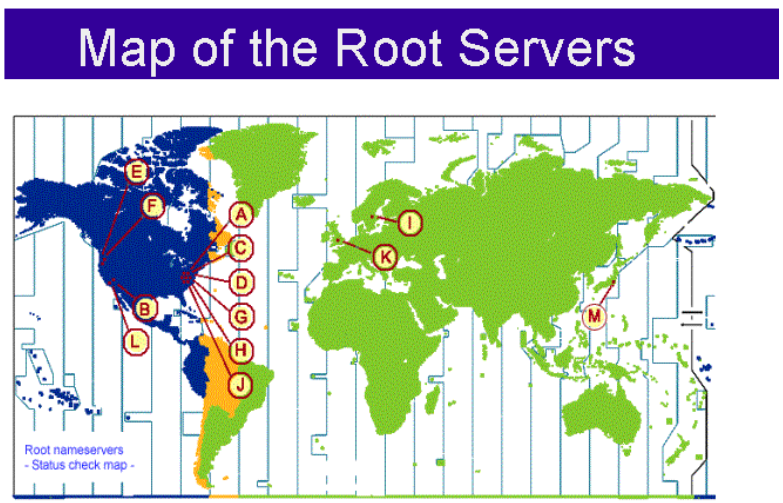
- Grand nombre de serveurs de noms, organisé de manière hiérarchique, distribué dans le monde.
- Aucun n'a la correspondance complète.



- Un client veut l'adresse IP d'`www.amazon.com`.
  - Client envoie une requête au serveur DNS racine pour trouver le serveur DNS com.
  - Client envoie une requête au serveur DNS com pour avoir le serveur DNS amazon.com
  - Client envoie une requête au serveur DNS amazon.com pour avoir l'adresse de `www.amazon.com`.

DNS - serveurs DNS racine

- Interrogé par le serveur DNS local s'il ne peut pas répondre lui-même.
- Contacte un serveur DNS de niveau inférieur s'il ne peut pas répondre.
- 13 serveurs racine dans le monde (10 milliards de requêtes par jour en 2007).



`http://gnso.icann.org`

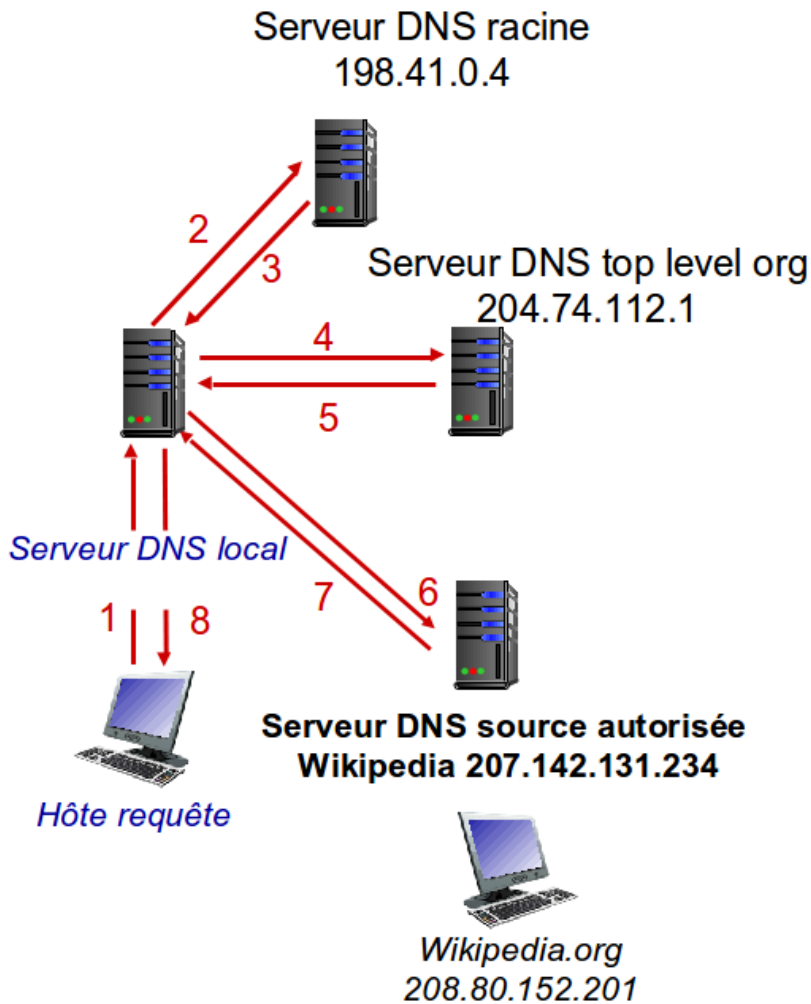
DNS

- Des serveurs DNS “top level” :
  - Responsables dans com, org, net... et des “top level” de chaque pays (fr, uk...)
  - AFNIC gère les .fr...
- Des serveurs DNS “source autorisée”.
  - Un serveur s’enregistre auprès d’un d’entre eux.

DNS - Exemple de résolution

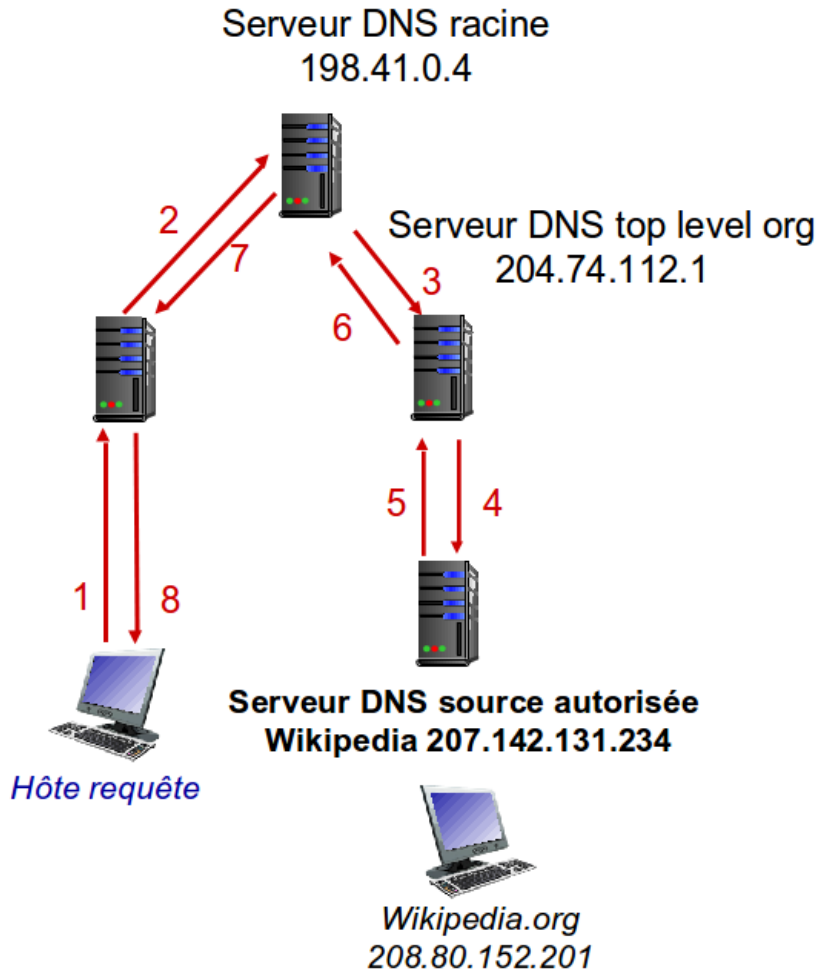
- Itératif ou récursif. Mix possible.

- Itératif :
  - Le serveur DNS répond avec l'adresse d'un autre serveur DNS à contacter.



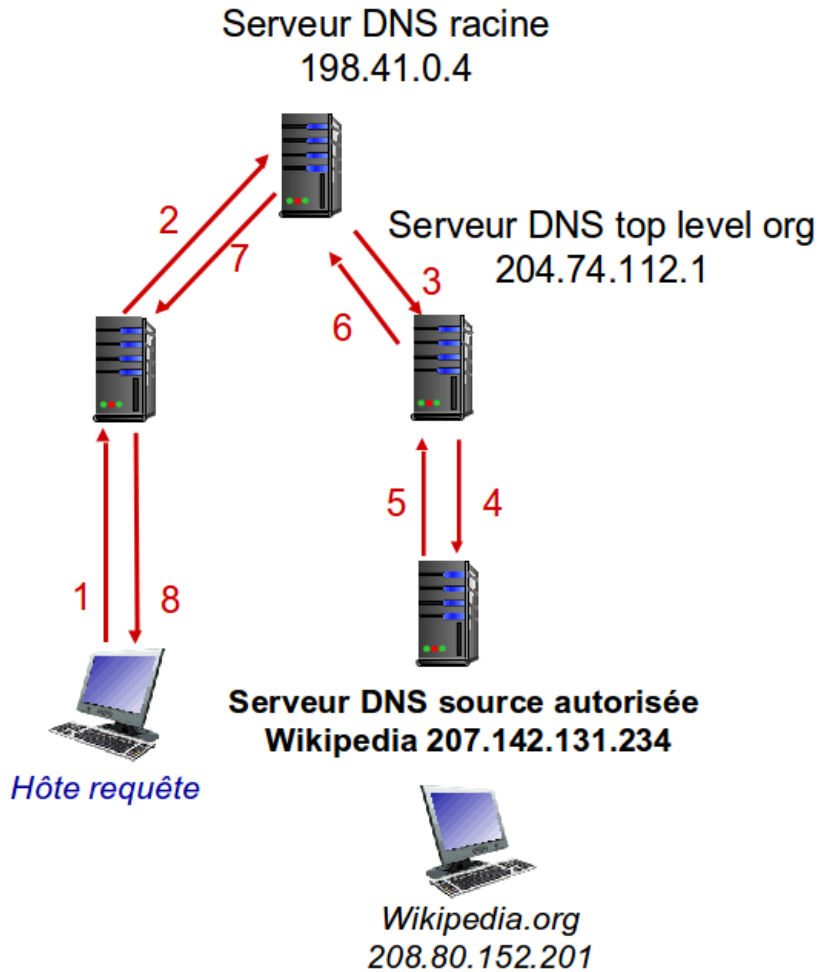
DNS - Exemple de résolution

- Récursif :
  - Le serveur DNS contacté contacte lui même le serveur DNS suivant et répond avec l'adresse IP désirée.
  - Augmente la charge du serveur DNS.



DNS - Exemple de résolution

- Récursif :
  - Le serveur DNS contacté contacte lui même le serveur DNS suivant et répond avec l'adresse IP désirée.
  - Augmente la charge du serveur DNS.



- Cache possible à chaque niveau.

Enregistrements DNS

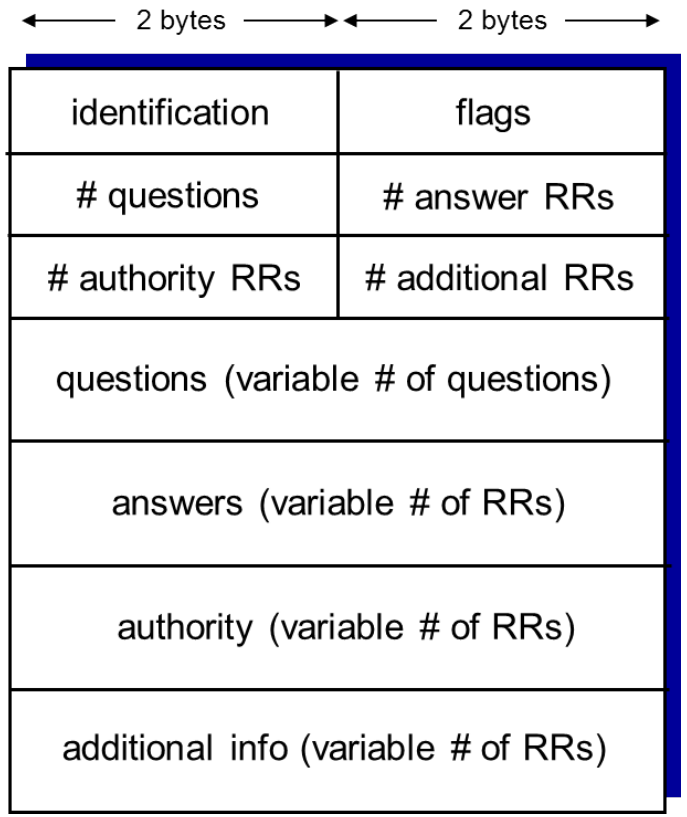
- Les bases de données enregistrent les informations de correspondance (RR : Resource Record) (ipconfig /diplaydns sous windows).
- Format : (Name, Value, Type, TTL)
- TTL : Time To Live.

- Type=A (1) :
  - Name= nom du serveur.
  - Value= adresse IP correspondante.
  - Exemple : (l1.lamsade.dauphine.fr, 193.48.71.250, A)
- Type=NS (2) :
  - Name= domaine.
  - Value= nom d'un serveur DNS source autorisée qui sait obtenir l'IP des serveurs du domaine.
  - Exemple : (toto.com, dns.toto.com, NS)
- Type=CNAME(5) :
  - Name=alias.
  - Value=nom canonique de l'alias.
  - Exemple : (toto.com, relay1.titi.toto.com, CNAME).
- Type=MX (15) :
  - Name=alias.
  - Value=nom canonique d'un serveur de messagerie de l'alias.
  - Exemple : (toto.com, mail.titi.toto.com, MX)
- ...

Messages DNS

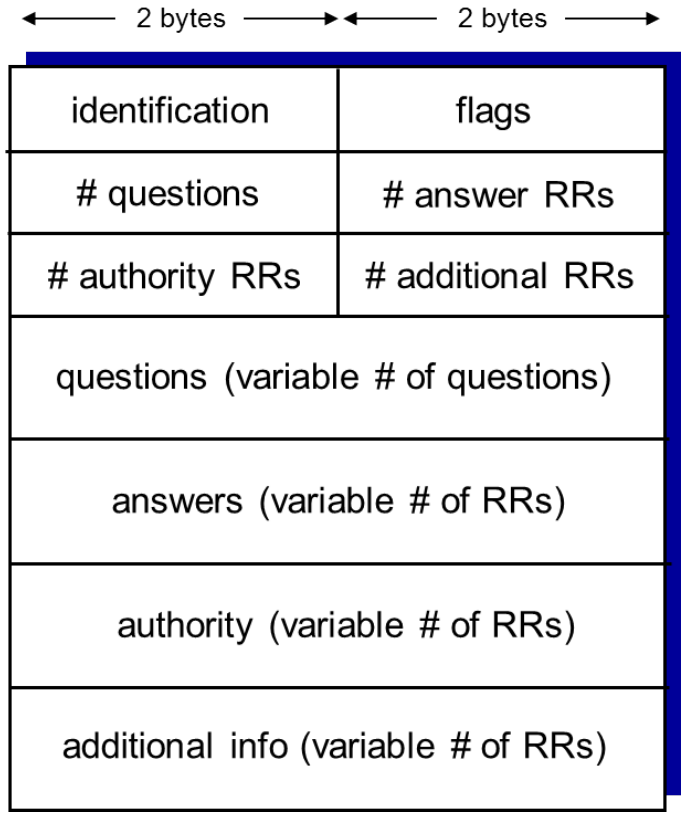
- Requête et réponse avec le même format.

- 12 premiers octets (bytes) : en tête à plusieurs champs.
- Id : nombre sur 16 bits (associer les réponses aux demandes).
- Flags (1 bit chacun (0/1)) :
  - Requête ou réponse.
  - Récursion désirée (requête).
  - Récursion possible (réponse).
  - A l'autorité.
- 4 champs suivants indiquent la taille des 4 champs de données suivants.



Messages DNS

- Questions (demandes) : Nom demandé, type de la demande (A, MX...).
- Réponses : Réponse pour le nom demandé avec le type. Serveur avec plusieurs IP, peut répondre plusieurs IP.
- Autorité : Réponses d'autres serveurs de source autorisée.
- Autres informations potentiellement utiles...



Insérer une information

- Exemple, nouvelle startup DauphineUberAlles.
- Enregistre le nom dauphineuberalles.fr au registraire de nom de domaine associé (AFNIC pour les .fr).
  - Fourni des noms, des adresses IP et 2 DNS de source autorisée (primaire et secondaire).
  - Le registraire ajoute deux RR dans le DNS top level :
    - (dauphineuberalles.fr, dns1.dauphineuberalles.fr, NS)
    - (dns1.dauphineuberalles.fr, 212.212.212.1, A)

Attaques DNS

- Attaques DDoS :
  - Bombarder un serveur DNS racine avec du trafic.
    - Peu susceptible de fonctionner (Anonymous a échoué en 2012).
    - Blacklist possible.
    - Les serveurs DNS locaux ont en cache des top level.
  - Attaquer un top level peut être plus dangereux.
- Attaques redirigées :
  - "Man in the middle" (pirate répond à des requêtes DNS et peut renvoyer le site de votre banque vers le sien...).
  - Empoisonnement (envoie de fausses informations à des serveurs DNS qui mettent en cache).

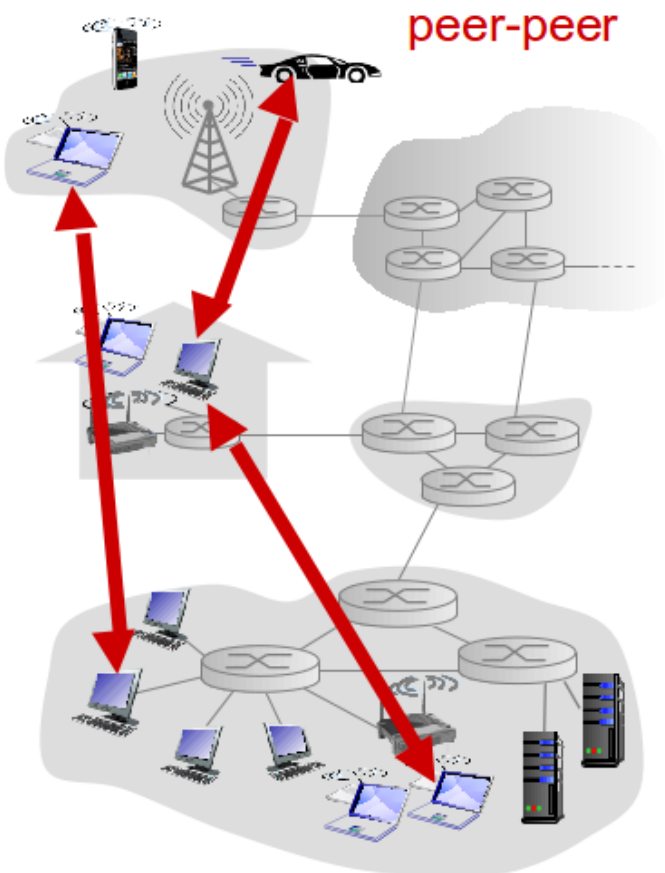
Cours 2 : Couche applications

Architecture

- Hôtes communiquent directement.
- Pairs non toujours connectés et peuvent changer d'adresse.

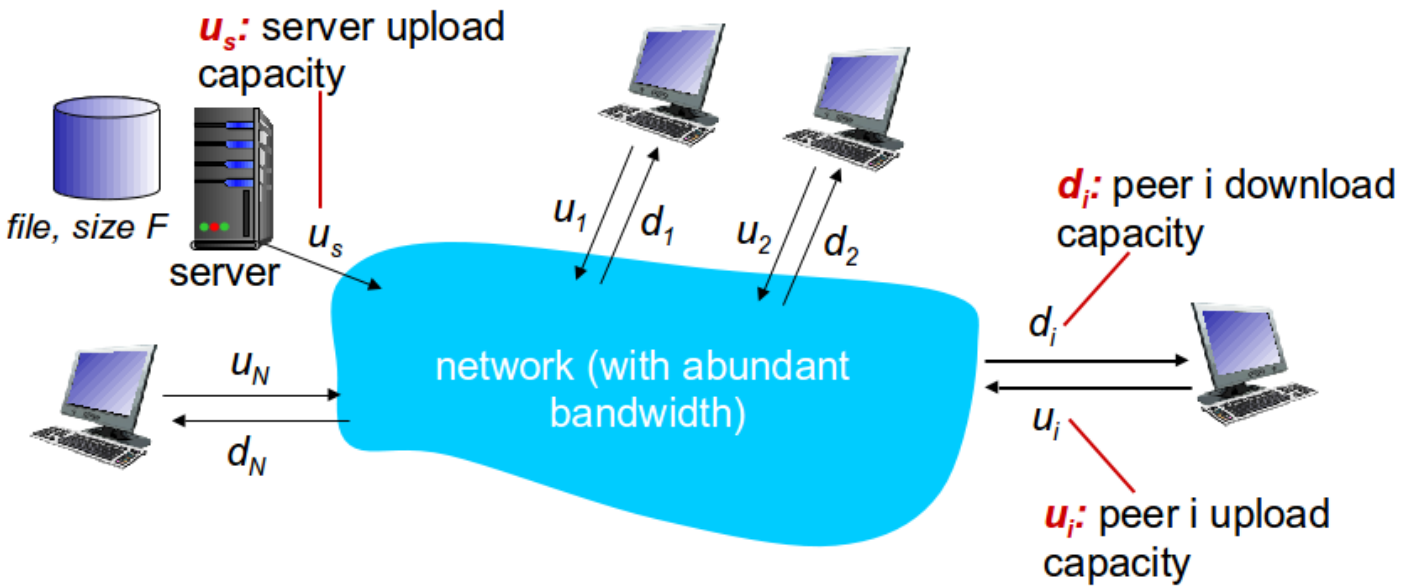
- Distribution de fichiers (BitTorrent...).
- Streaming (vidéo SopCast, musique Spotify...).
- VoIP (Skype...).
- Monnaie électronique (Bitcoin)

- BitTorrent : 21% du download et 60% de l'upload en Europe en 2011 (Chiffres Sandvine).



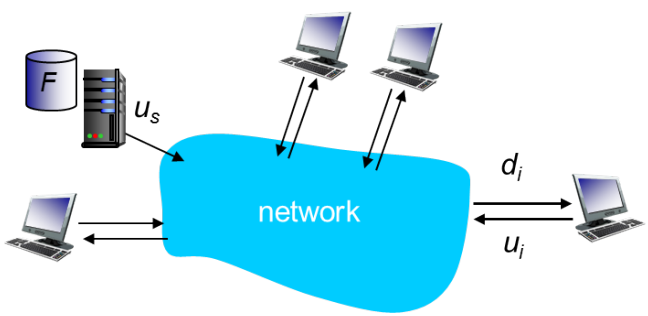
Comparaison

- Q : Temps pour distribuer un fichier de taille F d'un serveur à n pairs.
- La capacité d'upload/download de chaque pair est limité.



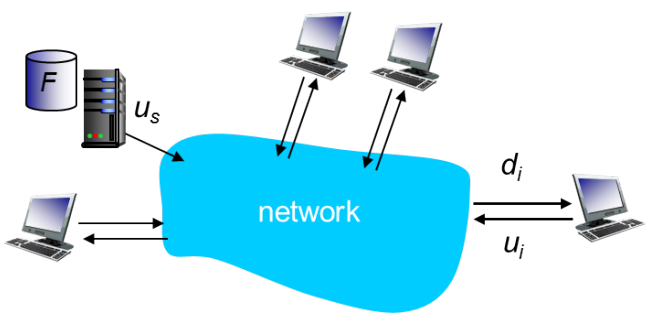
Comparaison - Cas client-serveur

- Transmission du serveur :
  - Doit envoyer séquentiellement n fois F :
    - Temps pour un envoi : F/u_s.
    - Temps pour n envois : nF/u_s.
- Client :
  - Chaque client doit télécharger une copie.
    - d_{min} = plus petite capacité de téléchargement.
    - Plus long temps de téléchargement : F/d_{min}.



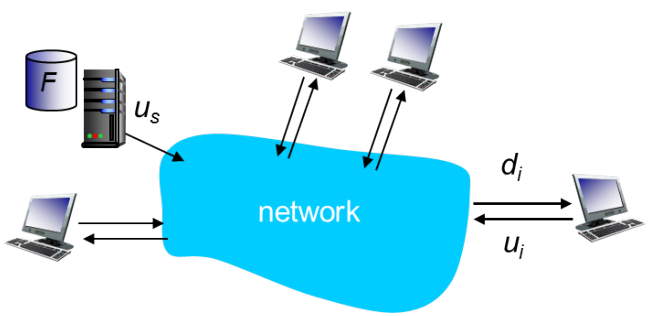
Comparaison - Cas client-serveur

- Transmission du serveur :
  - Doit envoyer séquentiellement n fois F :
    - Temps pour un envoi : F/u_s.
    - Temps pour n envois : nF/u_s.
- Client :
  - Chaque client doit télécharger une copie.
    - d_{min} = plus petite capacité de téléchargement.
    - Plus long temps de téléchargement : F/d_{min}.
- $D_{c/s}$: Temps pour qu'un serveur distribue F à n clients.
- $D_{c/s} \geq \max(\frac{n.F}{u_s}, \frac{F}{d_{min}})$



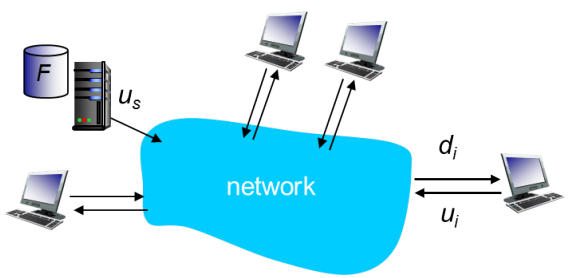
Comparaison - Cas client-serveur

- Transmission du serveur :
  - Doit envoyer séquentiellement n fois F :
    - Temps pour un envoi : F/u_s.
    - Temps pour n envois : nF/u_s.
- Client :
  - Chaque client doit télécharger une copie.
    - d_{min} = plus petite capacité de téléchargement.
    - Plus long temps de téléchargement : F/d_{min}.
- $D_{c/s}$: Temps pour qu'un serveur distribue F à n clients.
- $D_{c/s} \geq \max(\frac{n.F}{u_s}, \frac{F}{d_{min}})$
- Augmente linéairement avec n.



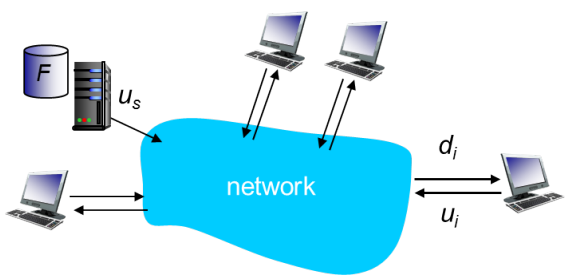
Comparaison - Cas P2P

- Plus compliqué à calculer (qui possède quoi, etc)
- Seul le serveur possède le fichier au début :
  - Temps pour le serveur pour envoyer une seule fois F :
  - F/u_s.
- Client :
  - Chaque client doit télécharger une copie.
    - F/d_{min}
- Au total, les clients doivent télécharger $n.F$ bits.
  - Meilleur taux d'upload possible idéalement sur tout le réseau (limitant la vitesse de téléchargement) : $u_s + \sum u_i$



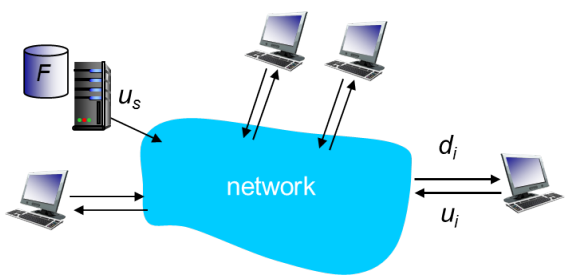
Comparaison - Cas P2P

- Plus compliqué à calculer (qui possède quoi, etc)
- Seul le serveur possède le fichier au début :
  - Temps pour le serveur pour envoyer une seule fois F :
  - F/u_s.
- Client :
  - Chaque client doit télécharger une copie.
    - F/d_{min}
- Au total, les clients doivent télécharger $n.F$ bits.
  - Meilleur taux d'upload possible idéalement sur tout le réseau (limitant la vitesse de téléchargement) : $u_s + \sum u_i$
- D_{P2P} : temps pour distribuer F à n clients en P2P.
- $D_{P2P} \geq \max(\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{n.F}{(u_s+\sum u_i)})$ (borne inf.)



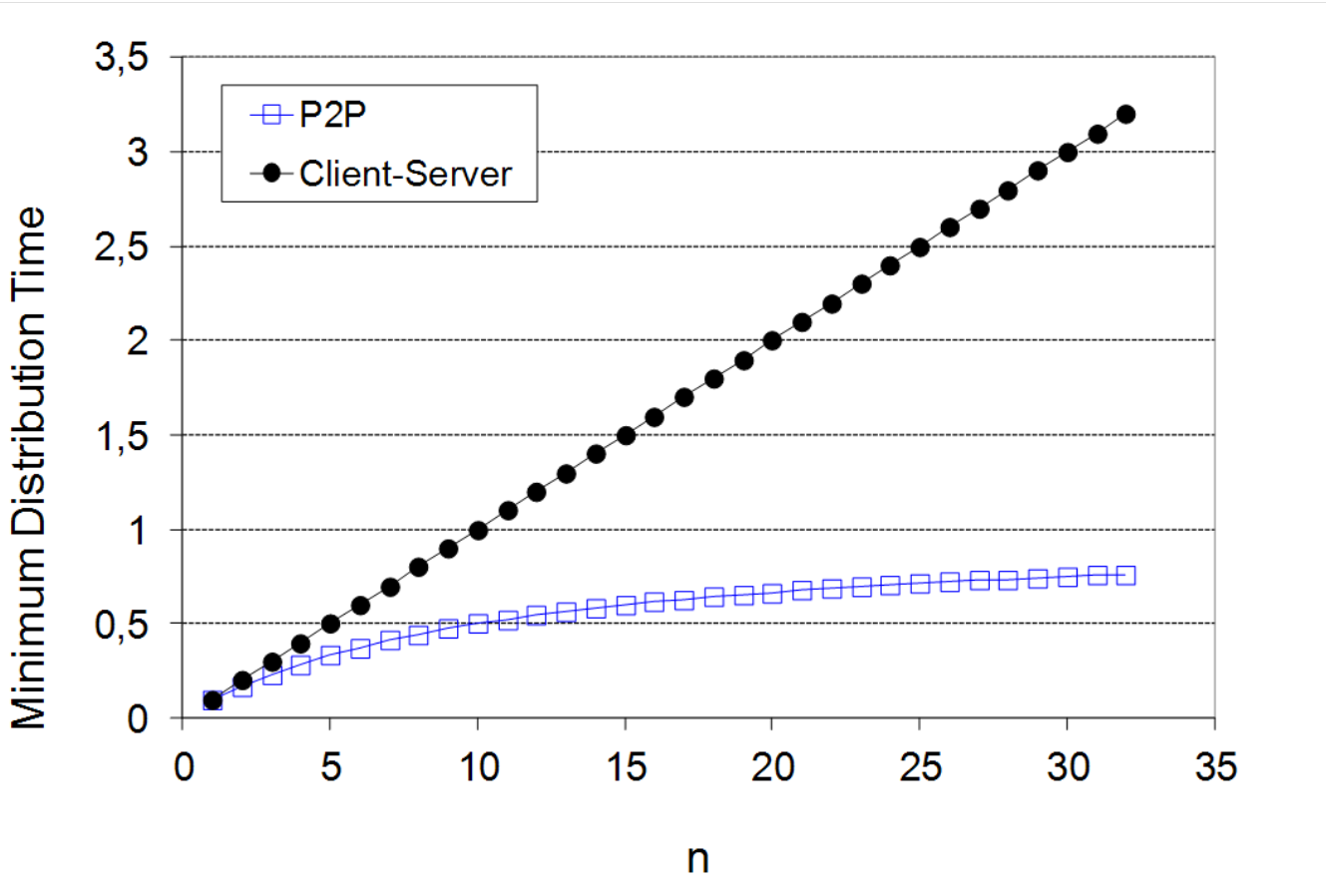
Comparaison - Cas P2P

- Plus compliqué à calculer (qui possède quoi, etc)
- Seul le serveur possède le fichier au début :
  - Temps pour le serveur pour envoyer une seule fois F :
  - F/u_s.
- Client :
  - Chaque client doit télécharger une copie.
    - F/d_{min}
- Au total, les clients doivent télécharger $n.F$ bits.
  - Meilleur taux d'upload possible idéalement sur tout le réseau (limitant la vitesse de téléchargement) : $u_s + \sum u_i$
- D_{P2P} : temps pour distribuer F à n clients en P2P.
- $D_{P2P} \geq \max(\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{n.F}{(u_s+\sum u_i)})$ (borne inf.)
- Augmentation linéaire avec n... mais $\sum u_i$ aussi !



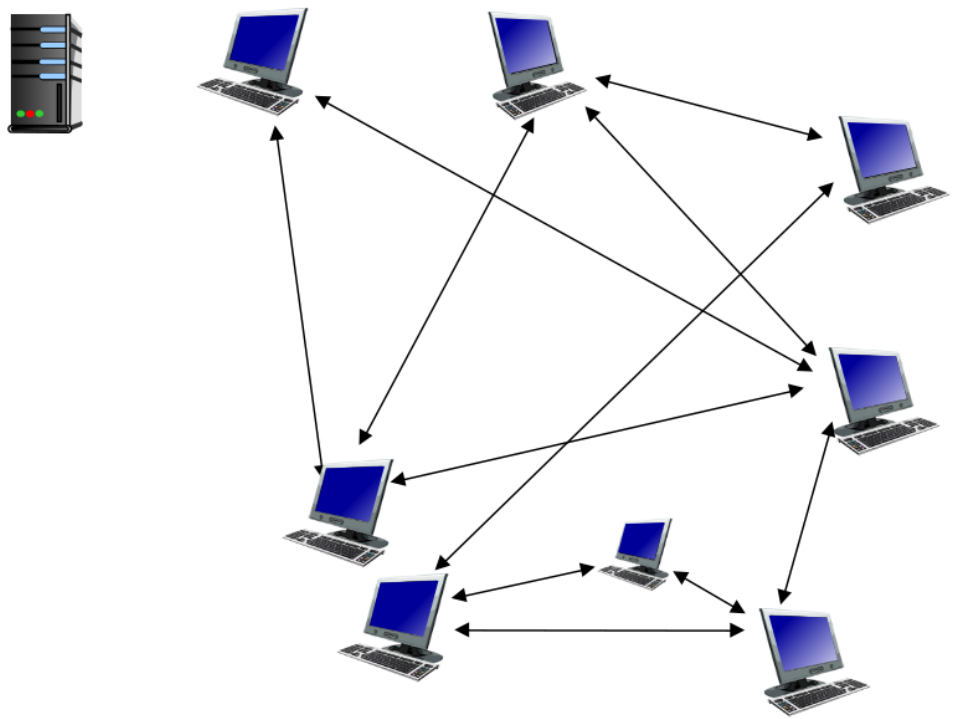
Comparaison

- Upload d'un client : u.
- $F/u = 1$ heure.
- $u_s = 10 \cdot u$.
- $d_{min} \geq u_s$.



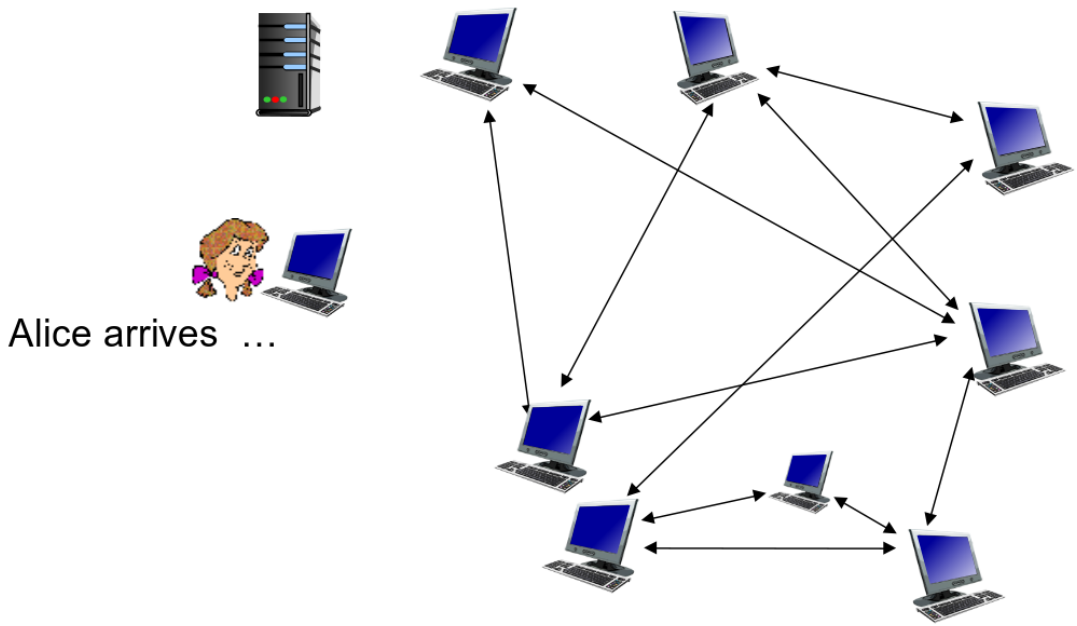
P2P : Bittorent

- Fichiers (ex : distribution linux) divisés en “chunks” (tranches) (ex. 256 Kb).
- Pairs recoivent/envoient des chunks.
- Tracker : serveur connaissant quels pairs ont des chunks du fichier.



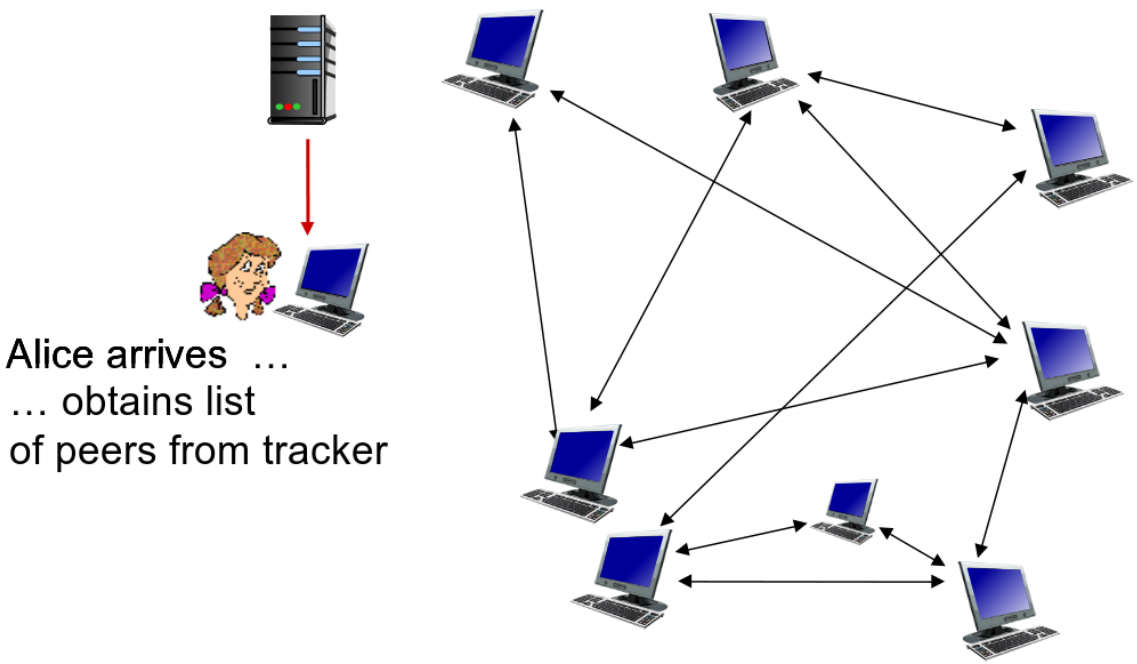
P2P : Bittorent

- Fichiers (ex : distribution linux) divisés en “chunks” (tranches) (ex. 256 Kb).
- Pairs recoivent/envoient des chunks.
- Tracker : serveur connaissant quels pairs ont des chunks du fichier.



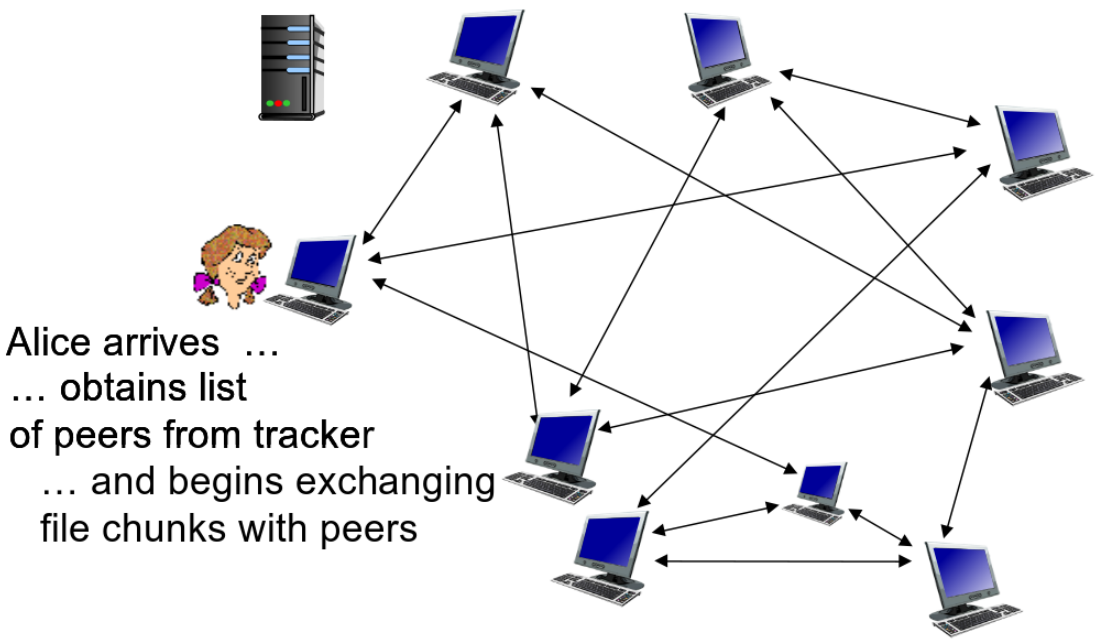
P2P : Bittorent

- Fichiers (ex : distribution linux) divisés en “chunks” (tranches) (ex. 256 Kb).
- Pairs recoivent/envoient des chunks.
- Tracker : serveur connaissant quels pairs ont des chunks du fichier.



P2P : Bittorent

- Fichiers (ex : distribution linux) divisés en “chunks” (tranches) (ex. 256 Kb).
- Pairs recoivent/envoient des chunks.
- Tracker : serveur connaissant quels pairs ont des chunks du fichier.



P2P : Bittorent

- Pair rejoint le réseau :
  - N'a pas de chunk, mais va les accumuler au cours du temps depuis d'autres pairs.
- Pair télécharge, et upload des chunks à d'autres pairs.

P2P : Bittorent

- Pair rejoint le réseau :
  - N'a pas de chunk, mais va les accumuler au cours du temps depuis d'autres pairs.
- Pair télécharge, et upload des chunks à d'autres pairs.
- Une fois le fichier entièrement téléchargé, il peut (égoïstement) partir, ou (altruistement) rester.

P2P : Bittorent - demander un chunk

- À un temps donné, différent pairs ont différentes tranches du fichier.
- Périodiquement, Alice demande aux pairs une liste des tranches qu'ils possèdent.
- Alice demande les chunks qui lui manque, les plus rares en premier.

P2P : Bittorent - envoyer un chunk

- Tit-for-Tat (Coopération-réciprocité-pardon / donnant-donnant).
- Alice envoie des chunks aux pairs qui lui en envoient le plus rapidement.
  - Les autres ne recoivent rien.
  - Ré-évalue régulièrement.
- Périodiquement, Alice choisit aléatoirement un pair pour lui envoyer des chunks.
  - Lui offre la possibilité de rejoindre le top.

P2P : Bittorent - envoyer un chunk

- Tit-for-Tat (Coopération-réciprocité-pardon / donnant-donnant).
- Alice envoie des chunks aux pairs qui lui en envoient le plus rapidement.
  - Les autres ne recoivent rien.
  - Ré-évalue régulièrement.
- Périodiquement, Alice choisit aléatoirement un pair pour lui envoyer des chunks.
  - Lui offre la possibilité de rejoindre le top.
- Gros taux d'upload : trouve de meilleurs partenaires : fichier obtenu plus rapidement.

P2P : Annuaires

- Un pair veut un fichier : nécessité d'un annuaire (on ne peut pas interroger tous les pairs).

Nom Fichier	IP
Ubuntu 13.10.iso	64.25.10.44
Star wars.avi	74.25.14.85
Booba feat La Fouine.zip	78.14.25.2
...	...

P2P : Annuaires

- Plus facile de travailler avec des valeurs que des noms.
- Fonction de hachage des noms (en fait, du fichier binaire).
- Clé = hash(clé originale).
- Annuaire : table de hachage !

Clé	IP
454675	64.25.10.44
464467	74.25.14.85
854517	78.14.25.2
...	...

P2P : Annuaires

- “Avant” (Napster...), serveur central répertoriant les adresses des pairs possédant des fichiers.
- Vérifie que les pairs sont toujours connectés (ping, connexion TCP...)
  - Sensible à la panne.
  - Goulet d'étranglement.
  - Légalité...

P2P : Annuaires

- "Avant" (Napster...), serveur central répertoriant les adresses des pairs possédant des fichiers.
- Vérifie que les pairs sont toujours connectés (ping, connexion TCP...)
  - Sensible à la panne.
  - Goulet d'étranglement.
  - Légalité...
- Pour un réseau totalement décentralisé : l'annuaire doit être également réparti !
  - Chaque pair possède une partie de l'annuaire.
    - Chaque pair est responsable de tous les fichiers qui...
  - Nécessité de redondance.

DHT

- Distributed Hash Table.
- Comporte des entrées (clé,valeur).
  - Ex : (Numéro sécu, nom de famille), (id fichier, IP)...

DHT

- Distributed Hash Table.
- Comporte des entrées (clé,valeur).
  - Ex : (Numéro sécu, nom de famille), (id fichier, IP)...
- Distribution des entrées sur les millions de pairs.
- Un pair fait une requête DHT avec une clé.
  - DHT retourne la valeur associée à la clé.
- Un pair peut aussi insérer une entrée.

DHT

- Comment assigner une entrée à un pair ?
- Idée :
  - Convertir chaque clé (nom de fichier) en un entier.
  - Assigner un entier à un pair (Ex : `clé = hash(‘‘Ubuntu 12.10’’)`).
  - Stocker l'entrée (clé, valeur) chez le pair qui est le “plus proche” de la clé.

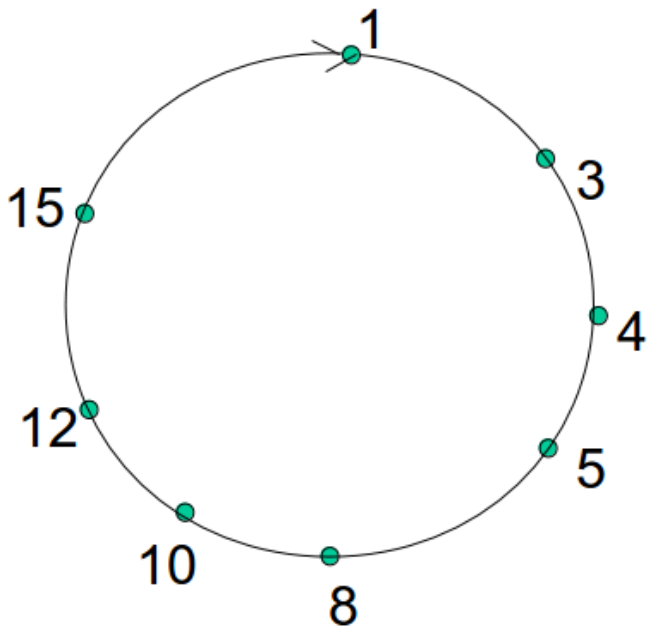
DHT - Assigner

- Règle : assigner la clé au pair qui à l'identifiant le plus proche.
- Convention : plus proche = successeur immédiat.

DHT - Assigner

- Règle : assigner la clé au pair qui à l'identifiant le plus proche.
- Convention : plus proche = successeur immédiat.
- Par ex : $n = 4$, pairs : $1, 3, 4, 5, 8, 10, 12, 14$;
  - clé = 13 : pair successeur est 14.
  - clé = 15 : pair successeur est 1.

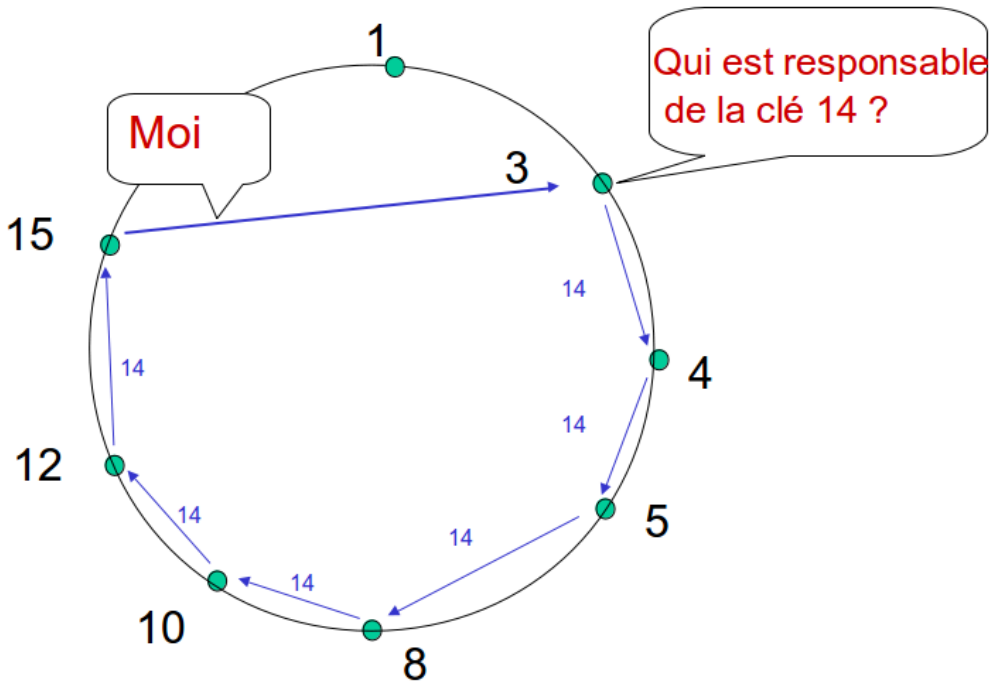
DHT circulaire



- Chaque pair ne connait uniquement que (l'adresse IP de) son prédécesseur et son successeur immédiat.
- Chaque pair ne connait que 2 adresses, même si le réseau est constitué de millions de nœuds.

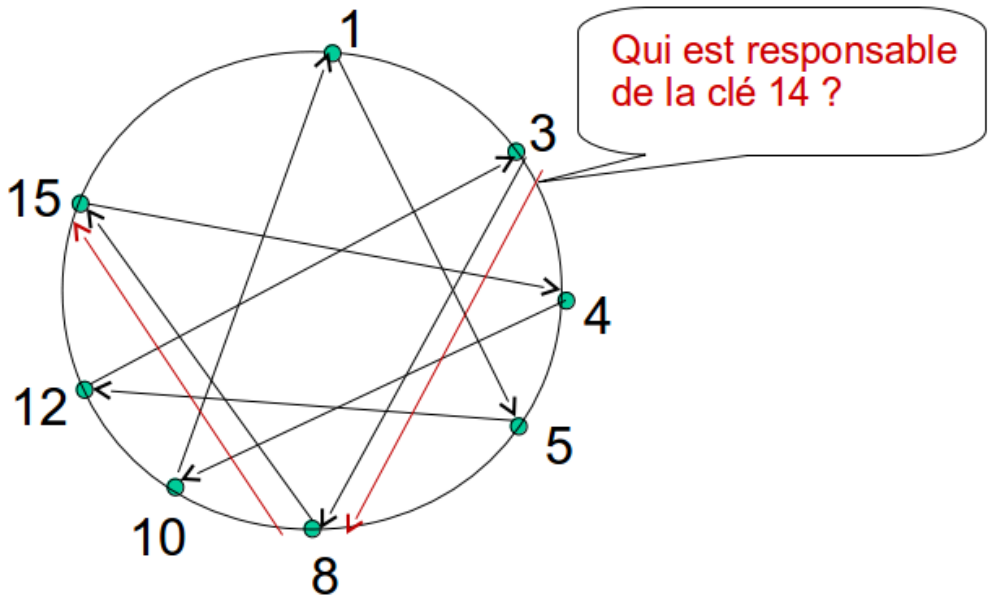
DHT circulaire

- "Plus proche" défini comme le "plus proche successeur".



- $O(n)$ requêtes pour résoudre une requête avec n pairs.

DHT circulaire avec raccourcis

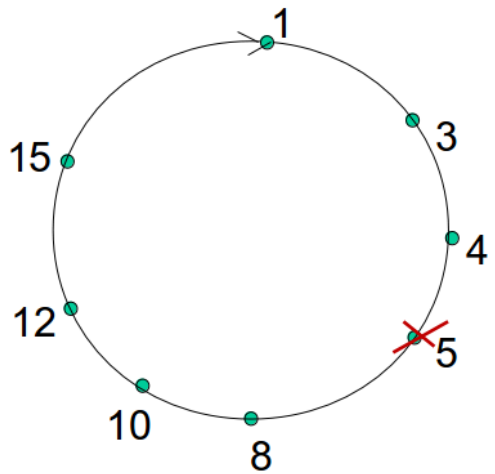


- Chaque pair garde trace de l'IP de son prédécesseur, son successeur et son raccourci.
- Réduit le nombre de messages envoyés.
- Si $O(\log n)$ voisins (raccourcis), $O(\log n)$ messages dans la requête.

DHT - perte de pairs

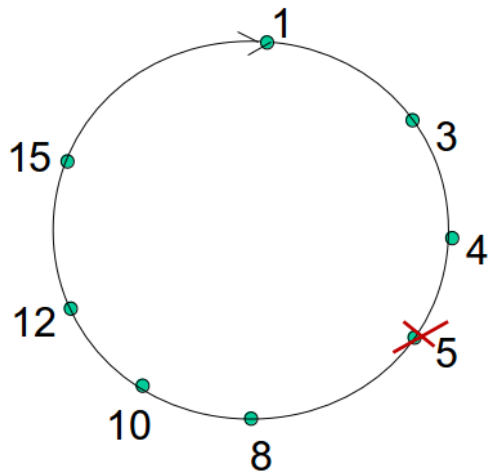
- Pairs peuvent partir et venir.
- Chaque pair connaît l'adresse de ses 2 successeurs.
- Chaque pair "ping" régulièrement ses 2 successeurs.
- Si le 1er est mort, choisit le 2ème comme nouveau 1er.

DHT - perte de pairs



- Pair 5 quitte le réseau.
  - Pair 4 détecte le départ de 5.
  - Fait de 8 son successeur immédiat.
  - Demande à 8 qui est son successeur immédiat.
  - 4 fait du successeur immédiat de 8 son second successeur.
  - N'affecte que le voisinage ($\neq$ hachage classique).

DHT - perte de pairs



- Pair 5 quitte le réseau.
  - Pair 4 détecte le départ de 5.
  - Fait de 8 son successeur immédiat.
  - Demande à 8 qui est son successeur immédiat.
  - 4 fait du successeur immédiat de 8 son second successeur.
  - N'affecte que le voisinage ($\neq$ hachage classique).
- Et si 14 rentre ?
  - Mise à jour des listes.
  - Responsable des clés 13 et 14, "déleste" 15.

DHT

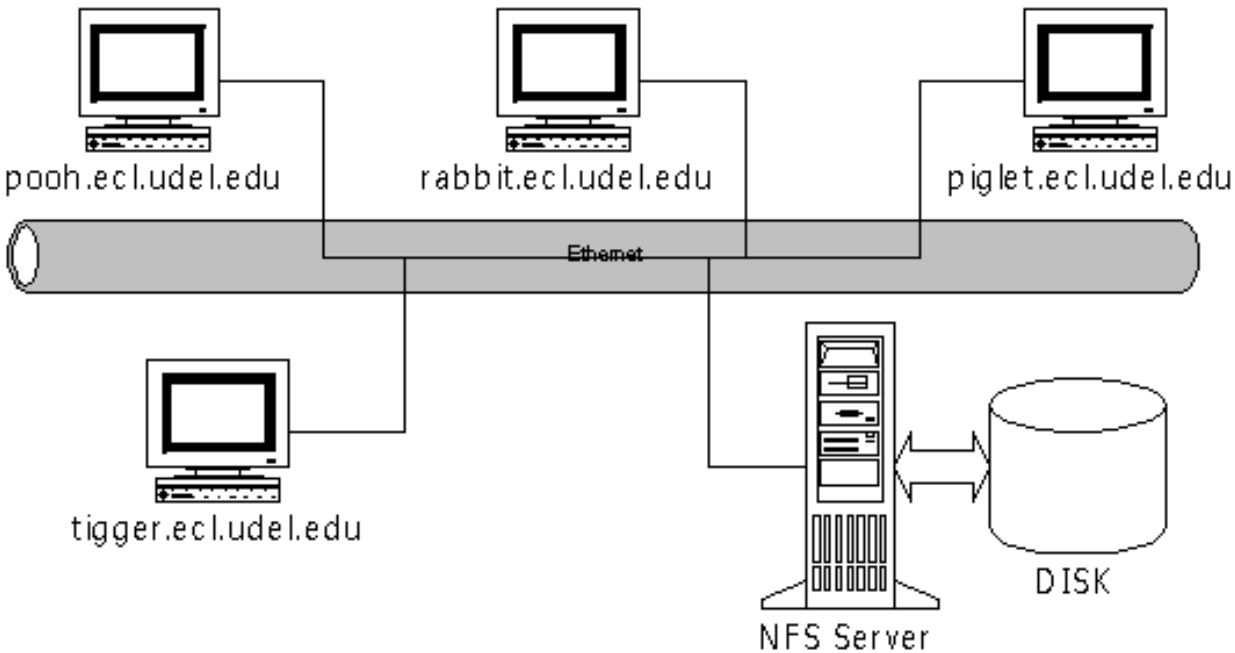
- Plus large que le P2P ! BDD distribuée (DNS, VoIP...).
- Intéressant :
  - Plus il y a de monde, plus il y a de capacité.
  - Auto-gestion, plutôt robuste à la perte.
- Un système pas parfais !
  - Doit connaitre par un moyen certains nœuds pour y rentrer...
  - Problème de sécurité si certains nœuds sont evil. Difficile de vérifier l'intégrité.
  - Balance des nombre de clés, mais quid de la balance selon popularité ?

Cours 2 : Couche applications

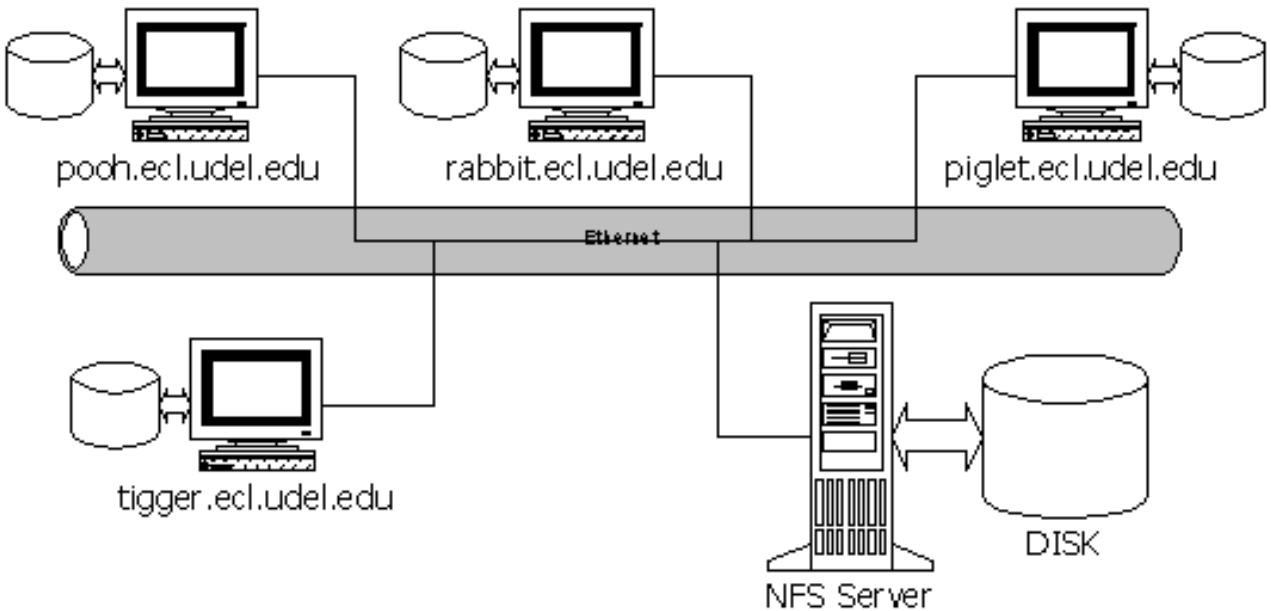
NFS

- Network File System.
- Protocole (couche Application) permettant de rendre transparent l'utilisation de fichiers répartis sur différentes machines, via un réseau.

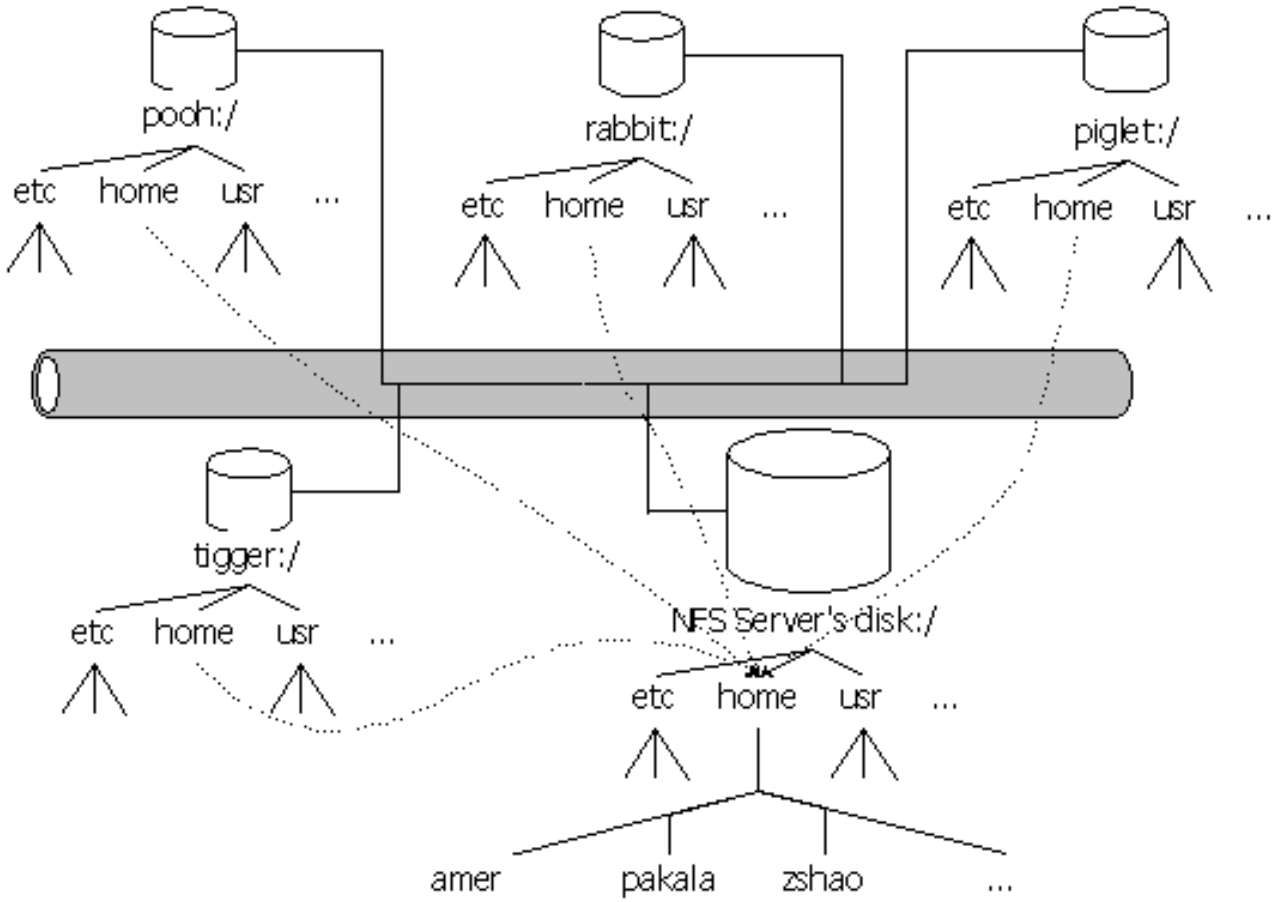
NFS - Postes de travail sans disque



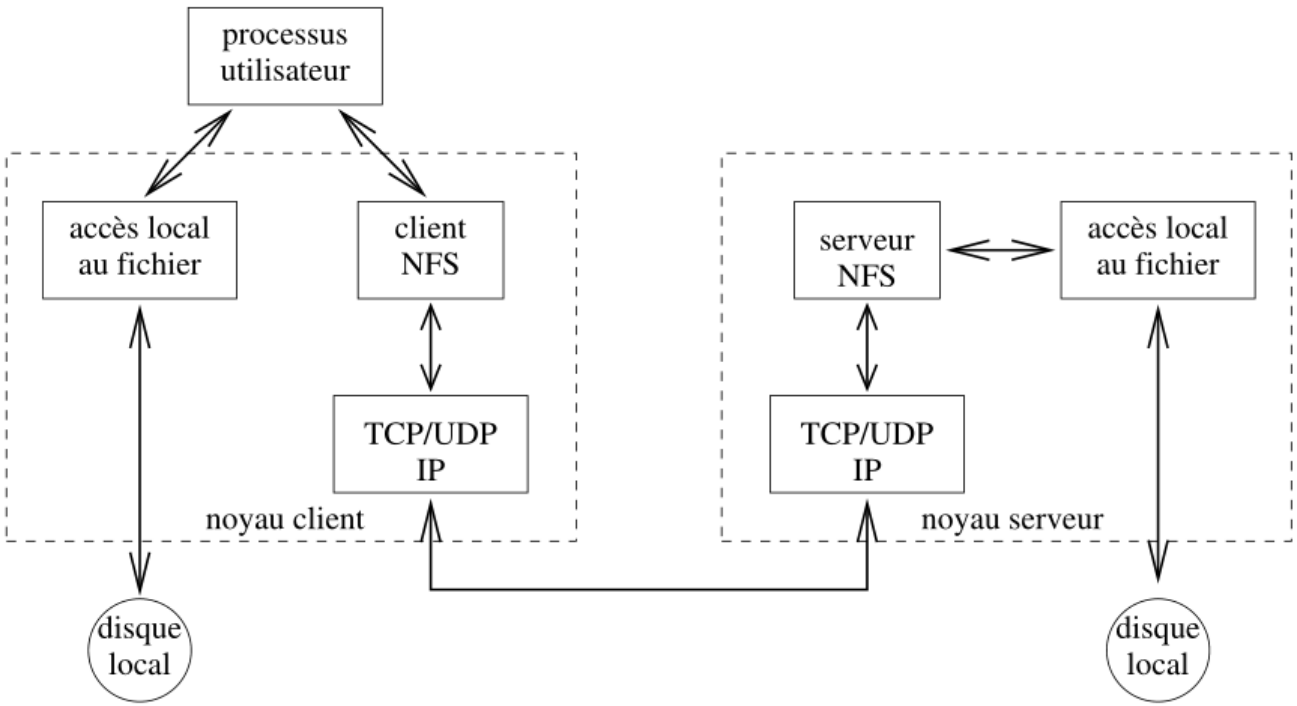
NFS - Plusieurs postes de travail pour un même utilisateur...



NFS - Plusieurs postes de travail pour un même utilisateur... vue logique



NFS - Fonctionnement



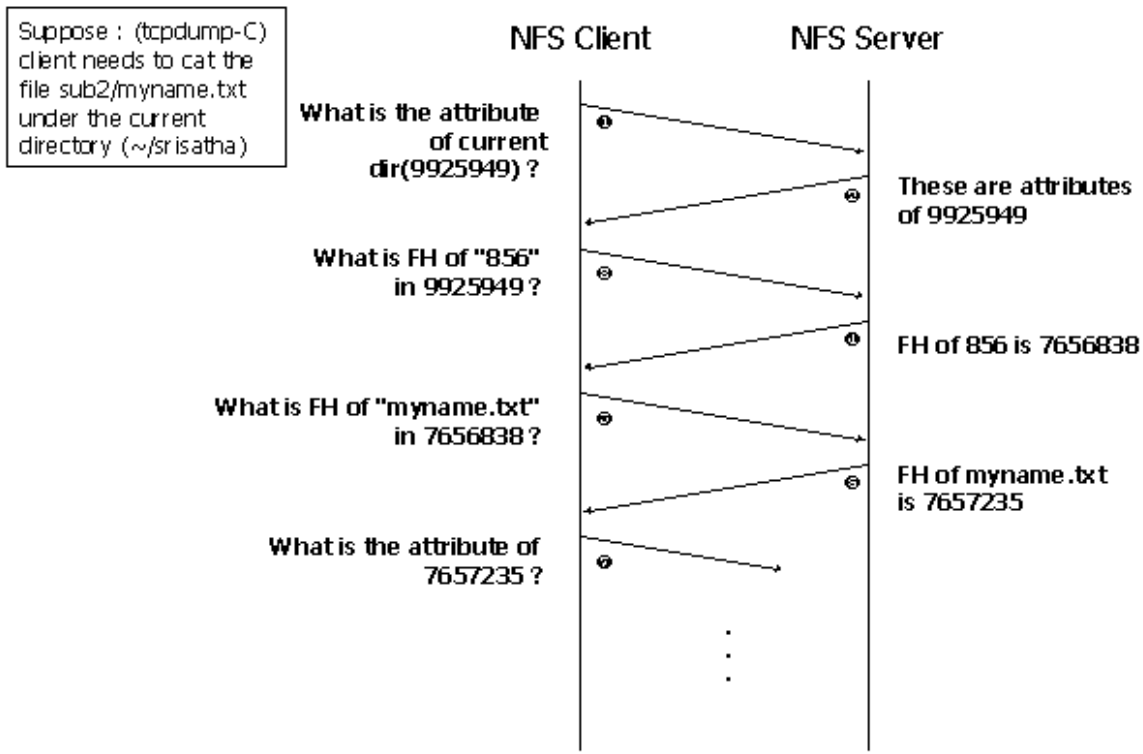
NFS - Fonctionnement

- Manipulation de fichiers en système :
  - Manipulation de “handle” (descripteurs), de chemins, déplacement, suppression...
- Descripteur opaque pour le client.

NFS - Fonctionnement

- Selon ce qui est demandé, le système d'exploitation fait la demande soit au système de fichier local, soit au client NFS.
- Le client NFS fait alors une requête au processus NFS de la machine serveur.
- Communication entre eux pour obtenir le descripteur (pas le fichier !)...
- Différent d'un protocole de transfert de fichiers.
  - Système de fichier distant, de manière transparente.

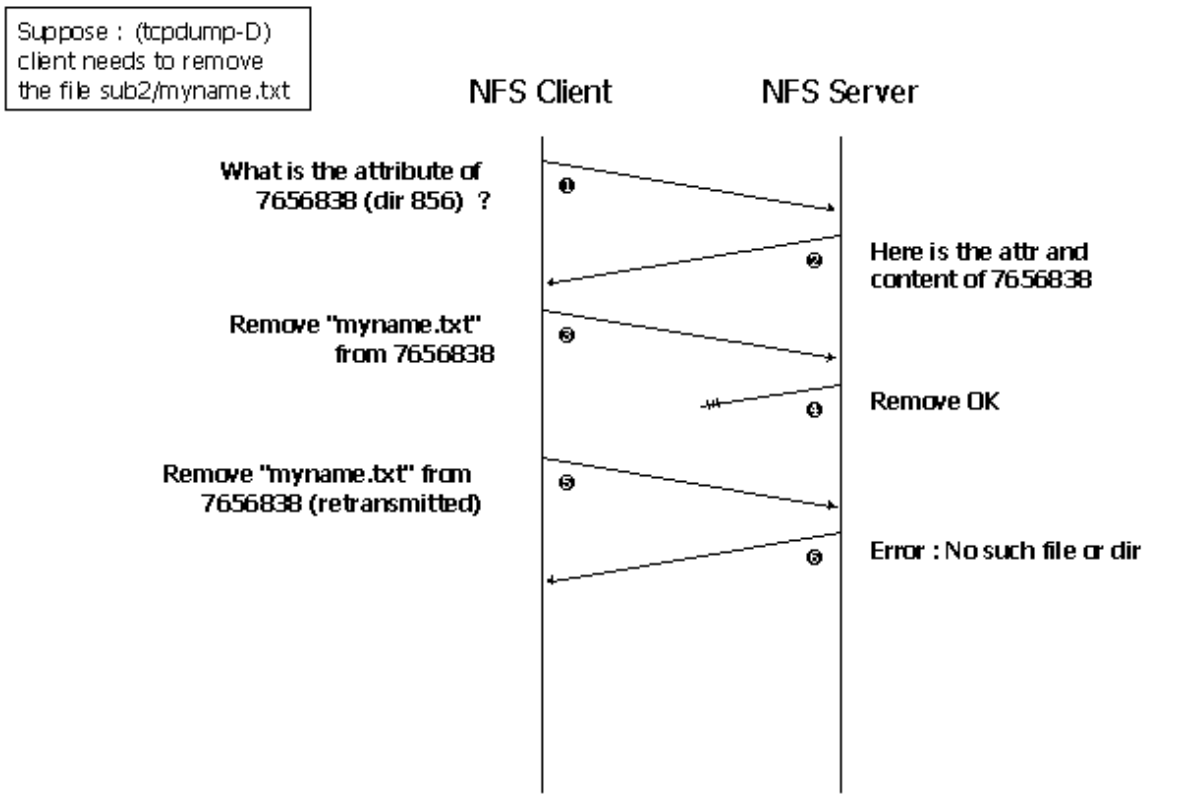
NFS - Exemple communication



NFS - Sans état

- Le serveur est sans état, il ne maintient aucune information vis à vis de client.
- S'il y a un problème (probable car sur le réseau), si le serveur plante, rien n'est perdu.
  - Le client retransmet jusqu'à recevoir une réponse.
- Serveur idempotent : même résultat si même requête.

NFS - Exemple perte



NFS - UDP ou TCP ?

- Premières version de NFS en UDP.
  - Essentiellement en réseau local.
  - Performances importantes pour transparence : TCP rajoute des en-têtes...
- Versions plus récente sur le réseau non local, utilise TCP.

Pour conclure...

- Architecture des applications réseau :
  - Client/serveur.
  - P2P.
- Service requis par certaines applications :
  - Bande passante, délai, fiabilité...
- Le modèle de transport sur Internet (plus à suivre) :
  - Fiabilité orienté connexion : TCP.
  - Non fiable, datagrammes : UDP.
- Quelques protocoles :
  - HTTP, FTP, SMTP/POP/IMAP, DNS, BitTorrent/DHT...

Pour conclure...

- Notion de protocoles.
- Échanges typiques requête/réponse très stricts.
  - Le client demande une information ou un service.
  - Le serveur répond avec des données, un code d'état...
- Format de messages :
  - En-têtes :
    - Champs donnant de l'information sur les données.
  - Données :
    - Information communiquée.

Pour conclure... Notions clés !

- Contrôle vs messages :
  - “Dans la bande” vs “hors bande”.
- Centralisé vs décentralisé.
- Sans état vs avec état.
- Transferts de messages fiables vs non fiables (plus dans la suite).

Capiche ?

http://www.quizzoodle.com/session/
170168d3dcb1483cadd4185b2e6368d2