

R203

Bases des services réseaux

Sami Evangelista
IUT de Villetaneuse
Département Réseaux et Télécommunications
2024–2025

<http://www.lipn.univ-paris13.fr/~evangelista/cours/R203>

Ce document est mis à disposition selon les termes de la licence Creative Commons “Attribution – Pas d’utilisation commerciale – Partage dans les mêmes conditions 3.0 non transposé”.



R203 — Bases des services réseaux

- ▶ Objectifs :
 - ▶ comprendre le rôle de la couche de transport et le fonctionnement des protocoles TCP et UDP
 - ▶ comprendre le rôle et le fonctionnement de services usuels :
 - ▶ DHCP (configuration automatique des hôtes)
 - ▶ SSH (connexion à distance, bases du chiffrement)
 - ▶ HTTP (service web)
 - ▶ configuration de ces services sous linux
- ▶ Volume horaire :
 - ▶ 1 cours introductif
 - ▶ 2 séances de cours-TD
 - ▶ 4 séances de TP
 - ▶ 1 contrôle
- ▶ Pour me joindre :
 - ▶ physiquement : bureau O108 à l'IUT
 - ▶ électroniquement : [sami.evangelista at lipn.univ-paris13.fr](mailto:sami.evangelista@lipn.univ-paris13.fr)

1. Services réseaux
2. La couche de transport
3. DHCP — Configuration automatique des hôtes
4. SSH — Connexion sécurisée
5. Services web

- ▶ leur but : rendre un service à des **clients**
 - ▶ nommage des machines
 - ▶ stockage de fichiers
 - ▶ envoi de mails
 - ▶ streaming vidéo
 - ...
- ▶ repose sur le modèle **client-serveur**
 - ▶ Le client est à l'initiative de l'échange (envoi d'une requête).
 - ▶ Le serveur traite la requête et répond.
- ▶ Au niveau du modèle OSI, ils se situent au niveau des couches 5 à 7.
- ▶ Techniquement, un service
 - ▶ est un **processus**
 - ▶ qui écoute sur un **port**
 - ▶ et utilise un **protocole de transport**.

- ▶ DNS — nommage des hôtes
- ▶ DHCP* — configuration automatique d'hôtes
- ▶ HTTP* — navigation web
- ▶ HTTPS — navigation web sécurisée
- ▶ FTP* — transfert de fichiers
- ▶ SSH* — connexion à distance sécurisée
- ▶ NFS — stockage de fichiers
- ▶ NTP* — horloge
- ▶ SMTP — envoi de messages
- ▶ IMAP — réception de messages

* = vu dans ce module

- ▶ port = identifiant local d'un processus sur l'hôte
- ▶ permet la communication simultanée de plusieurs processus (multiplexage)
- ▶ numéro sur 16 bits ($\Rightarrow$ numéro de port $\in [0, 65535]$)
- ▶ Dans le cadre du modèle client-serveur :
 - ▶ Le serveur écoute sur un port fixe (pour pouvoir être facilement contacté).
 - ▶ Le client utilise un port éphémère.

Les ports systèmes (ou well-known ports)

- ▶ ports de 0 à 1023
- ▶ utilisé par les protocoles les plus répandus (HTTP, SSH, DHCP, ...)
- ▶ Un processus écoutant sur un port système doit avoir les droits root.

Les ports éphémères (ou dynamiques)

- ▶ ports attribués par le système pour une courte durée
- ▶ ports de 32768 à 60999 (sur les versions récentes de linux)
- ▶ utilisation typique :

Lors d'une session TCP ou UDP, le système attribue un port éphémère au client pour la durée de la session.

- ▶ associe des protocoles à un numéro de port + protocole de transport
- ▶ liste gérée par l'IANA (Internet Assigned Numbers Authority)
- ▶ extrait du fichier :

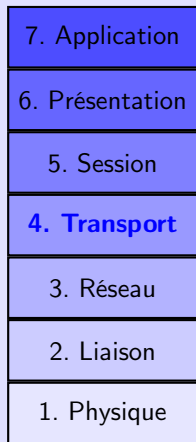
ftp	21/tcp
ssh	22/tcp
domain	53/tcp
domain	53/udp
http	80/tcp

- ▶ Ce ne sont que les ports par défaut !
 - ▶ Un serveur peut écouter sur n'importe quel port ou utiliser un protocole de transport qui n'est pas celui indiqué mais il faut que les clients en soient informés.

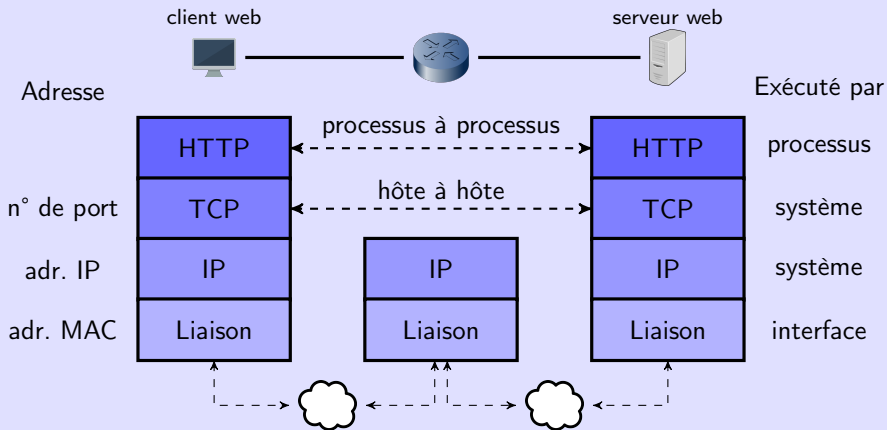
1. Services réseaux
2. La couche de transport
3. DHCP — Configuration automatique des hôtes
4. SSH — Connexion sécurisée
5. Services web

- ▶ communication de bout-en-bout entre processus
- ▶ multiplexage de processus
- ▶ (optionnel) correction des erreurs réseau
- ▶ (optionnel) contrôle des flux
(éviter la congestion du réseau)

protocoles de transport les plus courants : UDP et TCP
(Il y en a d'autres : SCTP, DDP, ...)



- pile mise en œuvre par un client (p.ex. : firefox) et un serveur web communiquant avec HTTP





- ▶ Des paquets envoyés sur le réseau peuvent
 - ▶ être perdus (paquet 3) ;
 - ▶ ou arriver dans le désordre (paquet 4 reçu avant le 2).
- ▶ C'est un réseau de type **best effort** : le réseau fait au mieux mais n'offre aucune garantie de bonne transmission.
- ▶ Sources des erreurs :
 - ▶ problèmes de transmission (bits erronés)
 - ▶ pannes (routeur défectueux, câble débranché, ...)
 - ▶ congestion (trop de paquets circulent sur le réseau $\Rightarrow$ mémoire des routeurs saturée $\Rightarrow$ les routeurs détruisent les paquets qu'il ne peuvent pas traiter)
 - ▶ mauvaise configuration des routeurs (p.ex. : paquets qui tournent en boucle)
 - ▶ paquets entre deux hôtes qui empruntent des chemins différents

- ▶ point commun : utilisation de numéros de port pour le multiplexage

UDP

- ▶ mode non connecté
- ▶ détection mais pas de correction des erreurs de réseau (perte ou déséquencelement)
- ▶ faible coût en terme de trafic réseau :
 - ▶ ajout d'un en-tête UDP (8 o.)

- ▶ point commun : utilisation de numéros de port pour le multiplexage

UDP

- ▶ mode non connecté
- ▶ détection mais pas de correction des erreurs de réseau (perte ou déséquencelement)
- ▶ faible coût en terme de trafic réseau :
 - ▶ ajout d'un en-tête UDP (8 o.)

TCP

- ▶ mode connecté
- ▶ détection et correction des erreurs par
 - ▶ acquittement des paquets reçus ;
 - ▶ et retransmission des paquets (considérés comme) perdus.
- ▶ contrôle de flux
- ▶ coût élevé en terme de trafic réseau :
 - ▶ en-tête TCP + long que l'en-tête UDP (20 contre 8)
 - ▶ paquets nécessaires au contrôle de l'échange (connexion, déconnexion, ...)

- ▶ Une application/un service réseau repose sur un protocole de transport pour l'échange de données.
- ▶ Lequel choisir ? Cela dépend des besoins de l'application.
- ▶ Quelques exemples :
 - ▶ si la fiabilité est le critère principal $\Rightarrow$ on utilise plutôt TCP
 - ▶ transfert de fichiers (HTTP, FTP)
 - ▶ email (SMTP, IMAP)
 - (fiabilité $\Leftrightarrow$ données délivrées sans erreur et sans déséquence)
 - ▶ fiabilité secondaire mais critères temporels forts $\Rightarrow$ on utilise plutôt UDP
 - ▶ téléphonie
 - ▶ streaming

1. Services réseaux
2. La couche de transport
3. DHCP — Configuration automatique des hôtes
4. SSH — Connexion sécurisée
5. Services web

- ▶ Pour pouvoir communiquer sur un réseau IP un hôte a besoin :
 - ▶ d'une IP et d'un masque de réseau (au minimum)
 - ▶ de l'IP du routeur (ou passerelle) de son réseau
 - ▶ de l'IP d'un serveur de noms (DNS pour la résolution noms d'hôtes → IP)
 - ...
- ▶ informations de configuration obtenues manuellement ou automatiquement

Configuration manuelle

- ▶ L'administrateur remplit des fichiers sur **chaque hôte**.
- ☹ fastidieux
- ☹ source d'erreurs (p.ex., attribuer la même IP à deux hôtes)

Configuration automatique

- ▶ L'administrateur remplit des fichiers sur **un serveur**.
- ▶ Le serveur fournit les informations de configuration aux hôtes (clients).
- ☺ simple, automatique
- ☹ tout repose sur le serveur (⇒ pb en cas de panne)

- ▶ DHCP (Dynamic Host Configuration Protocol)
- ▶ première RFC en 1993 (RFC 1531)
- ▶ évolution du protocole BOOTP
- ▶ protocole de transport utilisé : UDP sur les ports
 - ▶ 67 pour les serveurs et relais (voir plus loin)
 - ▶ 68 pour les clients

Le client

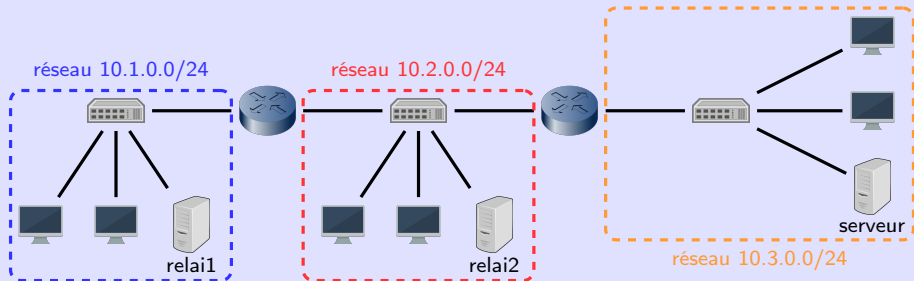
- ▶ hôte configuré automatiquement
- ▶ contacte le serveur pour obtenir un **bail** (+ éventuellement d'autres infos.)
- ▶ un bail = une IP attribuée pour une durée déterminée

Le serveur

- ▶ gère une plage d'IP à distribuer aux clients
- ▶ fournit aussi d'autres infos. (routeur, serveur DNS, ...) sur demande
- ▶ Remarque : on peut avoir plusieurs serveurs sur un réseau.
 - ▶ assure une **tolérance aux pannes** (remplacement d'un serveur défectueux)
 - ▶ nécessite des mécanismes de synchronisation entre serveurs

Le relai

- ▶ intermédiaire entre clients et serveurs situés sur des réseaux différents
(Les requêtes DHCP sont envoyées en diffusion sur le réseau local $\Rightarrow$ elles ne passent pas les routeurs $\Rightarrow$ besoin d'un intermédiaire pour relayer les paquets entre un client et un serveur séparés par un routeur.)



- ▶ Sur les réseaux 10.1.0.0/24 et 10.2.0.0/24, toute requête d'un client passe par le relai qui retransmet au serveur : client → relai → serveur.
- ▶ Toute réponse suit le chemin inverse : serveur → relai → client.
- ▶ Certaines interfaces doivent être configurées manuellement :
 - ▶ celles des routeurs
 - ▶ celles des relais et du serveur

1. Services réseaux
2. La couche de transport
3. DHCP — Configuration automatique des hôtes
4. SSH — Connexion sécurisée
5. Services web

- ▶ SSH = Secure SHell
- ▶ port par défaut : TCP/22
- ▶ offre une connexion sécurisée (chiffrée) sur un hôte distant
- ▶ remplace des protocoles non sécurisés (telnet, rsh, ...)
- ▶ 2 processus interviennent :
 - ▶ le client SSH : celui qui se connecte (avec la commande `ssh`) ;
 - ▶ et le serveur SSH = celui qui accepte la connexion.
- ▶ 2 versions incompatibles de SSH : 1.0 (1995) et 2.0 (2006, RFC4251)
(incompatible $\Leftrightarrow$ le client et le serveur doivent utiliser la même version)
- ▶ Différence majeure : SSHv2 corrige certaines failles de sécurité de SSHv1.

ssh [user@]hôte

- ▶ ouvre une connexion sécurisée sur
 - ▶ l'hôte (un serveur SSH désigné par son IP ou son nom)
 - ▶ en tant que user (par défaut, le même utilisateur que sur le client).
- ▶ condition : le serveur SSH doit être lancé sur l'hôte
- ▶ option courante :
 - ▶ -p num si le serveur écoute sur un port num $\neq$ 22

```
[sami@debian:/tmp]$ ssh evangelista@test.iutv.fr
evangelista@test.iutv.fr's password:
[evangelista@test.iutv.fr: ]$
```

```
scp source destination
```

- ▶ utilise le protocole SSH pour copier des fichiers à distance
- ▶ source ou destination peut désigner un fichier/répertoire distant :

```
[login@]hôte: [chemin]
```

Par défaut :

- ▶ login = même login que sur le client
- ▶ chemin = répertoire home de l'utilisateur
- ▶ options courantes :
 - ▶ -r (récursif) pour copier des répertoires
 - ▶ -P num si le port utilise sur un port num $\neq 22$

► Examples :

```
scp fic toto@10.1.2.3:      copie fic dans le home de toto
                             sur l'hôte 10.1.2.3
```

```
scp -r ssh.iutv.fr:rep .
```

copie dans le répertoire courant (.) le
répertoire rep se trouvant dans le home
de l'utilisateur sur l'hôte ssh.iutv.fr

- ▶ chiffrer = rendre des données incompréhensibles pour un tiers non autorisé
Alice envoie un message à Bob. Elle ne veut pas que le message puisse être intercepté et compris par Eve.
- ▶ assure la **confidentialité** (un des objectifs fondamentaux de la sécurité)
- ▶ 2 familles d'algorithmes :
 - ▶ chiffrement **symétrique**
 - ▶ chiffrement **asymétrique**

- ▶ Alice et Bob utilisent une **clé partagée** K (nombre, suite de bits, ...).
- ▶ La clé sert à la fois à chiffrer et à déchiffrer.
- ▶ Alice veut envoyer un message m à Bob :
 1. Alice chiffre m avec K et obtient m' .
 2. Alice envoie m' à Bob.
 3. Bob déchiffre m' avec K pour obtenir m .
- ▶ Exemples d'algorithmes symétriques : DES, AES, XOR
- ▶ Avantage : algorithmes rapides
- ▶ Inconvénient : nécessite l'échange de la clé (p.ex., par clé USB)

- ▶ repose sur l'utilisation de deux clés :
 - ▶ une **clé publique** K^{pub} qui peut chiffrer ;
 - ▶ et une **clé privée** K^{priv} qui peut déchiffrer.
- ▶ Principes :
 - ▶ K^{pub} et K^{priv} sont liées par une fonction mathématique.
 - ▶ À partir de K^{pub} il est (quasiment) impossible de trouver K^{priv} .
- ▶ Partage des clés :
 - ▶ K^{pub} peut être distribuée à tous.
 - ▶ K^{priv} ne doit pas être divulguée.
- ▶ Exemples d'algorithmes asymétriques : RSA, DSA, Diffie-Hellman
- ▶ Avantage : seules les clés publiques sont échangées
- ▶ Inconvénient : algorithmes très lents

Contexte

- ▶ Alice veut envoyer un message confidentiel m à Bob.
- ▶ Bob a une clé privée K^{priv}  et une clé publique K^{pub} .

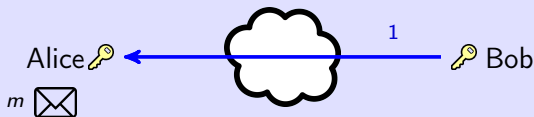
Alice
 m 

Bob

Déroulement

Contexte

- ▶ Alice veut envoyer un message confidentiel m à Bob.
- ▶ Bob a une clé privée K^{priv}  et une clé publique K^{pub} .

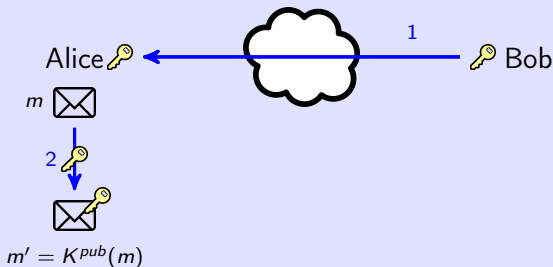


1. Bob envoie sa clé publique à Alice.

Déroulement

Contexte

- ▶ Alice veut envoyer un message confidentiel m à Bob.
- ▶ Bob a une clé privée K^{priv} 🔑 et une clé publique K^{pub} 🔑.

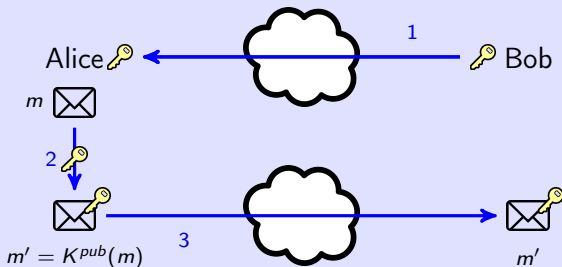


1. Bob envoie sa clé publique à Alice.
2. Alice chiffre m avec la clé publique de Bob et obtient m' .

Déroulement

Contexte

- ▶ Alice veut envoyer un message confidentiel m à Bob.
- ▶ Bob a une clé privée K^{priv} 🔑 et une clé publique K^{pub} 🔑.

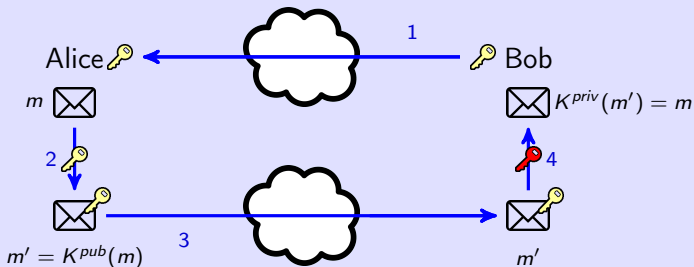


Déroulement

1. Bob envoie sa clé publique à Alice.
2. Alice chiffre m avec la clé publique de Bob et obtient m' .
3. Alice envoie m' à Bob.

Contexte

- ▶ Alice veut envoyer un message confidentiel m à Bob.
- ▶ Bob a une clé privée K^{priv} 🔑 et une clé publique K^{pub} 🔑.



Déroulement

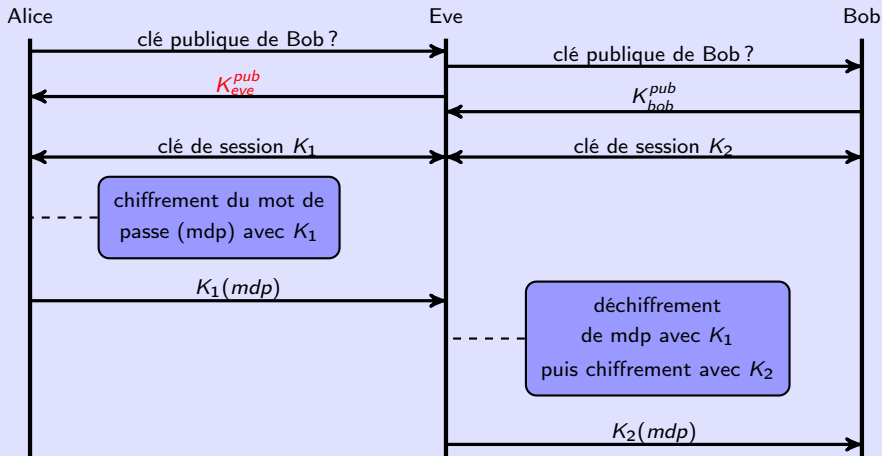
1. Bob envoie sa clé publique à Alice.
2. Alice chiffre m avec la clé publique de Bob et obtient m' .
3. Alice envoie m' à Bob.
4. Bob déchiffre m' avec sa clé privée pour obtenir m .

- ▶ Observation : impossible d'utiliser un seul type de chiffrement
 - ▶ symétrique : comment s'échanger la clé ?
 - ▶ asymétrique : trop lent, marcherait uniquement pour des très petits volumes de données
- ▶ Principe du chiffrement dans SSH :
 - ▶ combiner chiffrement symétrique et asymétrique
 - ▶ pour tirer parti des avantages des deux méthodes :
 - ▶ possibilité d'envoi de clés (en clair) du chiffrement asymétrique
 - ▶ rapidité du chiffrement symétrique
- ▶ Fonctionnement général d'une session SSH :
 1. établissement de la connexion TCP
 2. chiffrement **asymétrique** pour échanger une clé de session
clé de session = clé
 - ▶ **symétrique** ;
 - ▶ choisie **aléatoirement** ;
 - ▶ et **temporaire** (durée de vie = session SSH).
 3. puis chiffrement **symétrique** avec la clé de session pour échanger les données (login, mot de passe, commandes, ...)
 4. fermeture de la connexion TCP

- ▶ MITM = Man In the Middle
- ▶ Un attaquant réussit à intercepter le trafic entre deux hôtes cibles.
- ▶ exemples d'attaques MITM : usurpation ARP/DNS
- ▶ Application à SSH :
 - ▶ L'attaquant se fait passer pour le serveur auprès du client et inversement.

Contexte :

- ▶ Eve (l'attaquant) a réussi à intercepter le trafic entre Alice et Bob.
- ▶ Alice ouvre une connexion SSH sur Bob.



- ▶ Dans l'exemple précédent, l'attaque a fonctionné car Alice a reçu la clé d'Eve au lieu de celle de Bob.
- ⇒ besoin de vérifier les clés reçues !
- ▶ sauvegarde des clés dans `~/.ssh/known_hosts`
- ▶ 3 cas possibles lors de la connexion à un serveur

- ▶ Dans l'exemple précédent, l'attaque a fonctionné car Alice a reçu la clé d'Eve au lieu de celle de Bob.
- ⇒ besoin de vérifier les clés reçues !
- ▶ sauvegarde des clés dans `~/.ssh/known_hosts`
- ▶ 3 cas possibles lors de la connexion à un serveur

Cas 1 : serveur inconnu

⇔ serveur absent du fichier `~/.ssh/known_hosts`

```
$ ssh 10.11.12.13
The authenticity of host '10.11.12.13 (10.11.12.13)' can't be established.
ECDSA key fingerprint is e4:16:7e:9a:52:d3:a3:08:9c:be:12:73:de:e7:55:f5.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.11.12.13' (ECDSA) to the list of known hosts.
root@10.11.12.13's password:
```

- ▶ C'est ce qui arrive lorsqu'on se connecte au serveur pour la première fois.
- ▶ Soit on fait confiance en acceptant la clé (yes);
soit on la refuse (no) et on vérifie la clé reçue auprès de l'administrateur.

Cas 2 : serveur connu + clé valide

⇔ clé reçue = celle associée au serveur dans ~/.ssh/known_hosts

```
$ ssh 10.11.12.13  
root@10.11.12.13's password:
```

► Connexion acceptée par le client.

Cas 2 : serveur connu + clé valide

⇒ clé reçue = celle associée au serveur dans ~/.ssh/known_hosts

```
$ ssh 10.11.12.13
root@10.11.12.13's password:
```

- Connexion acceptée par le client.

Cas 3 : serveur connu + clé invalide

⇒ clé reçue ≠ celle associée au serveur dans ~/.ssh/known_hosts

```
$ ssh 10.11.12.13
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
02:0d:2e:2b:50:19:f9:6d:83:f5:f2:cb:7e:3e:9d:18.
Please contact your system administrator.
```

- Soit c'est une attaque MITM (Someone could be eavesdropping ...); soit l'administrateur a changé les clés du serveur (It is also ...).
- Dans les deux cas : connexion refusée par le client !

1. Services réseaux
2. La couche de transport
3. DHCP — Configuration automatique des hôtes
4. SSH — Connexion sécurisée
5. Services web

- ▶ HTTP = HyperText Transfer Protocol
- ▶ port par défaut : TCP/80
- ▶ protocole de transfert de ressources (web)
- ▶ client HTTP $\approx$ navigateur web
- ▶ utilisé initialement pour le transfert de documents hypertexte (p.ex., HTML) mais peut transférer tout type de document
- ▶ différentes versions :
 - ▶ 1.0 (1996, RFC 1945)
 - ▶ 1.1 (1997, RFC 2068)
 - ▶ 2.0 (2015, RFC 7540)

METH URI HTTP/VERSION

une ligne de requête

Champ1: Valeur1

N lignes d'en-tête

Champ2: Valeur2

Champ3: Valeur3

...

une ligne vide

corps de la requête

(éventuellement vide)

- ▶ METH = méthode. Les plus courantes :
 - ▶ GET = récupérer une ressource
 - ▶ POST = envoyer des données (p.ex., depuis un formulaire HTML)
- ▶ URI = Uniform Resource Identifier = chemin de la ressource demandée
- ▶ VERSION = 1.0, 1.1 ou 2.0
- ▶ Champ1, Champ2, ... : informations supplémentaires sur la requête/le client

- ▶ Host = nom du site web concerné par la requête
 - ▶ indispensable quand le serveur web héberge plusieurs sites
 - ▶ champ obligatoire depuis la version 1.1
- ▶ Connection = options de connexion (voir page 43)
- ▶ Date = Date de l'envoi de la requête
- ▶ Referer = URI du document qui contient un lien vers l'URI demandée
- ▶ User-Agent = infos sur le client web (p.ex., firefox version 1.2.3)
- ▶ Accept = liste des types de media (ou type MIME) acceptés en réponse
- ▶ If-Modified-Since = le serveur ne renvoie la ressource que si sa date de modification est postérieure à celle indiquée dans ce champ

- ▶ MIME = Multipurpose Internet Mail Extensions
- ▶ décrit le type d'une ressource retournée par le serveur
- ▶ utilisé par le navigateur web pour savoir comment traiter la ressource reçue
 - ▶ ex : type = audio/mpeg $\Rightarrow$ ouvrir lecteur audio
- ▶ classification sous la forme type/sous-type
- ▶ types usuels :

application/pdf

application/zip

audio/mpeg (fichier mp3)

image/gif

image/png

text/css

text/html

text/plain texte brut (non formaté)

video/mp4

video/webm

HTTP/VERSION CODE MESSAGE

une ligne de réponse

N lignes d'en-tête

Champ1: Valeur1

Champ2: Valeur2

Champ3: Valeur3

...

une ligne vide

*la ressource demandée
(éventuellement vide)*

corps de la réponse

► CODE = code de statut HTTP

► 1xx = information

► 2xx = succès

► 3xx = redirection

► 4xx = erreurs côté client

► 5xx = erreurs côté serveur

► MESSAGE = donne des infos complémentaires au code

- ▶ Connection
- ▶ Date
- ▶ Server = infos sur le serveur web (p.ex., apache version 2.4)
- ▶ Content-Type = type de media de la réponse
- ▶ Content-Length = nombre d'octets dans le corps de la réponse
- ▶ Last-Modified = date de la dernière modification de la ressource

Requête :

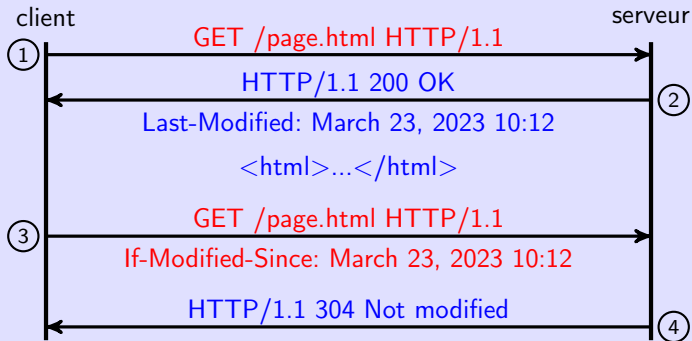
```
GET /page.html HTTP/1.0
Host: example.com
Referer: http://example.com/
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

Réponse :

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Server: Apache/0.8.4
Content-Type: text/html
Content-Length: 59
Last-Modified: Fri, 09 Aug 1996 14:21:40 GMT

<TITLE>Exemple</TITLE>
<P>Ceci est une page d'exemple.</P>
```

Source : wikipedia



- Les champs Last-Modified et If-Modified-Since permettent d'éviter de retélécharger des ressources non modifiées.

⇒ La réponse 4 est vide (un en-tête HTTP mais pas de corps).

Différences majeures :

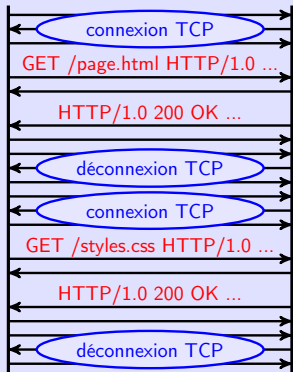
- ▶ connexions persistantes (utilisation du champ Connection)
- ▶ meilleure gestion du cache
- ▶ champ d'en-tête Host obligatoire dans les requêtes
- ▶ négociation de contenu (utilisation du champ Accept)

Exemple : le client télécharge une page `page.html` contenant un lien vers une feuille de style `styles.css`.

HTTP 1.0

- ▶ une connexion TCP par ressource téléchargée

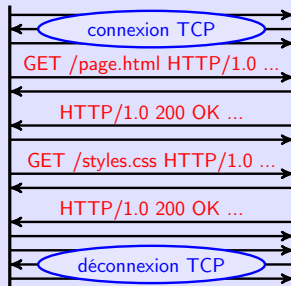
client serveur



HTTP 1.1

- ▶ une connexion TCP pour toutes les ressources téléchargées

client serveur



- ▶ `Connection: keep-alive`

Le client (ou le serveur) est prêt à maintenir la connexion TCP ouverte après le transfert de la ressource.

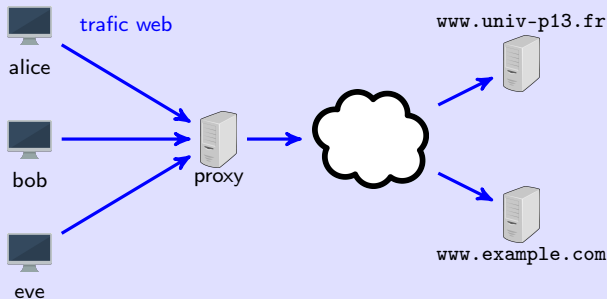
- ▶ `Connection: close`

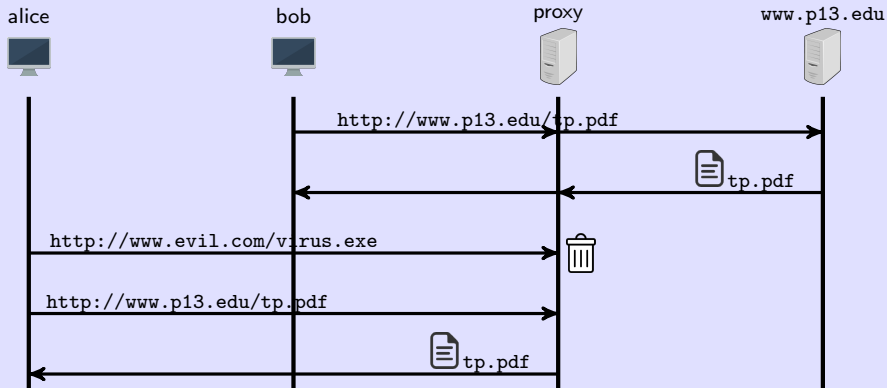
Le client (ou le serveur) veut mettre fin à la connexion TCP après le transfert de la ressource.

- ▶ mettre en cache une ressource = sauvegarder la ressource pour ne pas avoir à recontacter le serveur si elle est redemandée par l'utilisateur
 - 😊 serveur moins chargé
 - 😊 rapidité de chargement pour l'utilisateur
- ▶ Types de cache :
 - ▶ privé : sur le disque de l'utilisateur
 - ▶ partagé (ou public) : sur une machine accessible par plusieurs clients
- ▶ Problème : comment s'assurer que ce qui est dans le cache est bien à jour ?

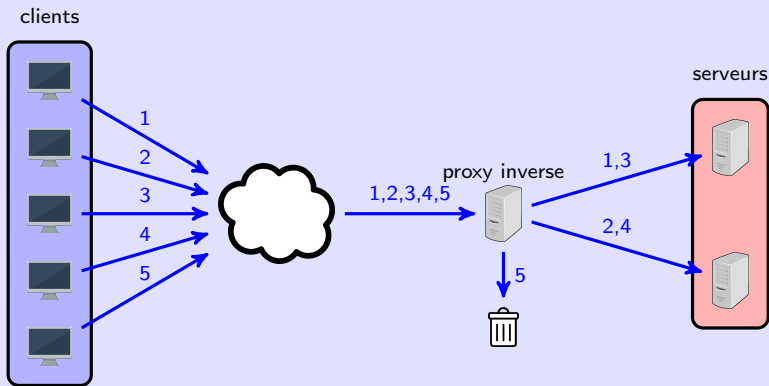
- ▶ Contient une liste de directives séparés par des virgules.
- ▶ Quelques directives :
 - ▶ max-age=N — la réponse est valide (i.e., peut être gardée en cache) pendant N secondes
 - ▶ no-store — ne pas stocker la réponse en cache
 - ▶ no-cache — la réponse peut être sauvegardée dans un cache mais devra être validée par le serveur (p.ex., avec le champ if-modified-since, voir page 40)
 - ▶ private — la réponse doit être sauvegardée dans un cache privé uniquement
 - ▶ public — la réponse peut être sauvegardée dans un cache public
- ▶ Exemple :
Cache-Control: max-age=36000, public, no-cache

- ▶ Un proxy est un **serveur mandataire**.
- ▶ C'est une passerelle de niveau applicatif (au sens OSI).
- ▶ proxy web = intermédiaire entre un navigateur et des serveurs web
- ▶ Le proxy peut améliorer
 - ▶ la **sécurité** (p.ex., bloquer des IP ou l'accès à des sites web malveillants) ;
 - ▶ et les **performances** (p.ex., en mettant en cache des ressources ou en compressant les données).





- ▶ proxy inverse = proxy placé devant des (en frontal de) serveurs
- ▶ quelques avantages :
 - ▶ **sécurité** (p.ex, en détectant et bloquant les attaques ciblant les serveurs)
 - ▶ **équilibrage de charge** (en distribuant un ensemble de requêtes équitabement entre les serveurs)



- ▶ La requête 5 contient du code malicieux détecté par le proxy inverse.