R203
Bases des services réseaux

Sami Evangelista
IUT de Villetaneuse
Département Réseaux et Télécommunications
2023–2024

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

- 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 :
  - 3 séances de cours (3 heures)
  - 2 séances de TD (5 heures)
  - 4 séances de TP (13 heures)
  - 1 contrôle (2 heures)
- Évaluation :
  - TP notés (1/3 de la note finale)
  - contrôle (2/3 de la note finale)
- Pour me joindre :
  - physiquement : bureau O108 à l'IUT
  - électroniquement : sami.evangelista at lipn.univ-paris13.fr

Plan

Les services réseaux

- 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**.

Services usuels

- 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

La notion de port

- 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.

Le fichier `/etc/services`

- 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.

Plan

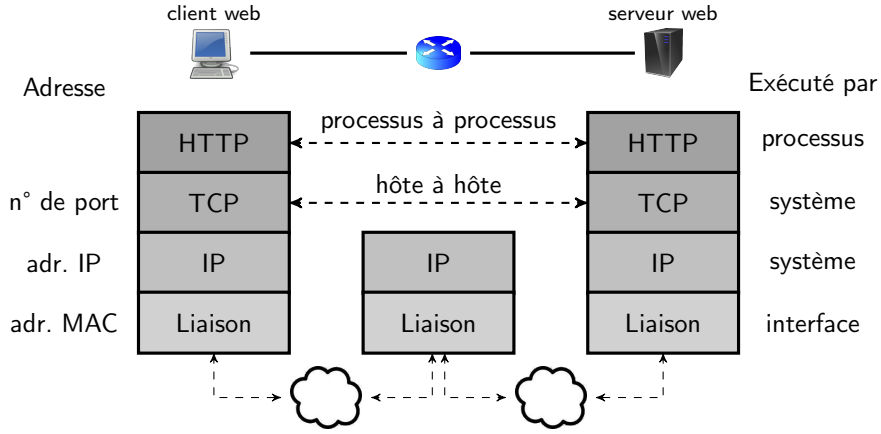
Rôle de la couche de transport

- 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 protocolaire

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



Internet, un réseau non fiable

- 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 ⇒ mémoire des routeurs saturée ⇒ 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

Caractéristiques d'UDP et TCP

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

UDP

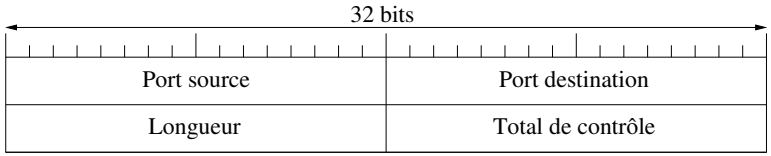
- mode non connecté
- détection mais pas de correction des erreurs de réseau (perte ou déséquencement)
- faible coût en terme de trafic réseau :
  - ajout d'un en-tête UDP (8 o.)

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, ...)

UDP — L'en-tête

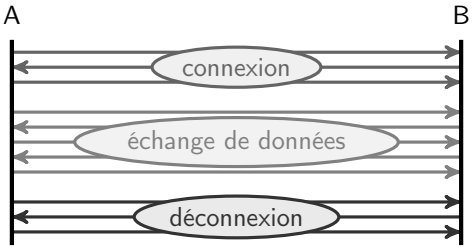
En-tête fixe de 8 octets :



- **Port source** (16 bits) = identifie le processus émetteur
- **Port destination** (16 bits) = identifie le processus destinataire
- **Longueur** (16 bits) = longueur totale du paquet (en-tête + données)
- **Total de contrôle** (16 bits) = code d'erreur calculé sur l'en-tête UDP + une partie de l'en-tête IP

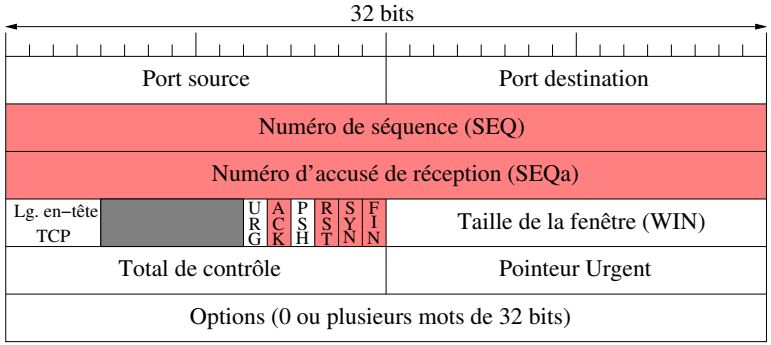
TCP — Déroulement d'une session

- TCP est un protocole en mode connecté :
  - phase de connexion avant tout échange de données
  - phase de déconnexion une fois l'échange terminé
- La phase de connexion sert :
  - à s'assurer que l'autre processus est prêt à communiquer ;
  - et à échanger des informations nécessaires à la suite de l'échange (dans le cas de TCP : des numéros de séquence).
- La phase de déconnexion sert à libérer des ressources (p.ex., de la mémoire allouée pour le contrôle de l'échange).



TCP — L'en-tête

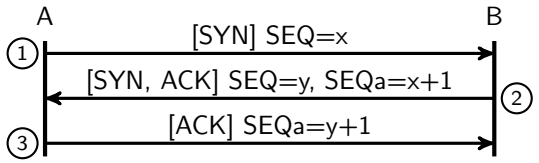
En-tête de 20 octets ou plus :



- **SEQ** (32 bits) = numéro du prochain octet de données envoyé
- **SEQa** (32 bits) = numéro du prochain octet de données attendu
- **ACK** (1 bit) = acquittement
- **RST** (1 bit, ReSeT) = refus de connexion ou déconnexion brutale
- **SYN** (1 bit, SYNchronisation) = demande de connexion
- **FIN** (1 bit, FINalisation) = demande de déconnexion

TCP — Connexion acceptée

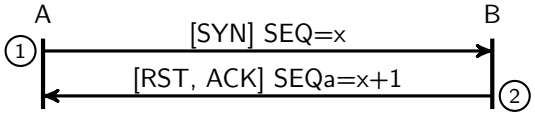
- connexion en 3 temps (three-way handshake)



1. A envoie un paquet
   - de demande de synchronisation (bit SYN=1) ;
   - et contenant un numéro de séquence initial choisi aléatoirement (SEQ=x).
2. B répond par un paquet
   - qui acquitte la demande de A (bit ACK=1 et SEQa=x+1) ;
   - et contient également une demande de synchronisation avec un numéro de séquence initial choisi aléatoirement (SEQ=y).
3. A répond par un paquet qui acquitte la demande de B.

TCP — Connexion refusée

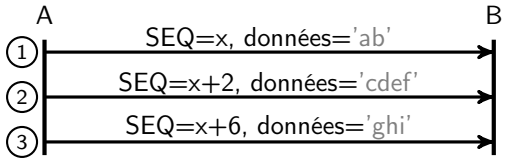
- aucun processus en écoute sur le port de réception du paquet SYN



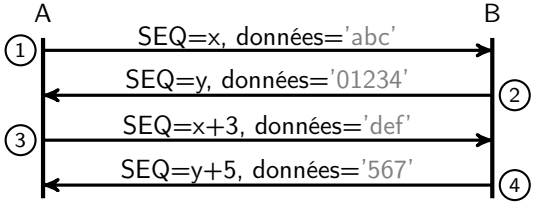
1. A envoie un paquet de demande de synchronisation.
2. B acquitte mais refuse la demande (bit RST=1).

TCP — Échange de données

- Les octets de données à envoyer sont numérotés.
- SEQ est le numéro de séquence du premier octet de données du paquet.
- Après l'envoi d'un paquet de données, le numéro de séquence est incrémenté du nombre d'octets de données envoyés.

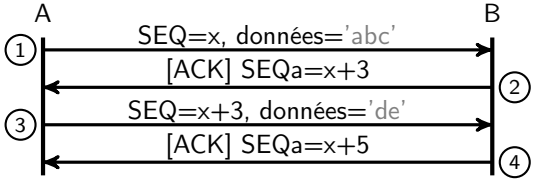


- communication bi-directionnelle ⇒ A et B ont des numéros de séquence indépendants.



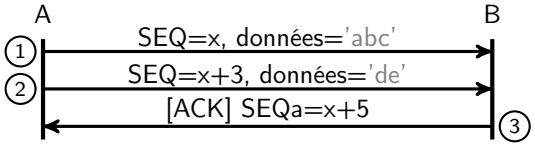
TCP — Acquittements

- Tout octet de données envoyé doit être acquitté.
- Cela se fait grâce au numéro de séquence acquitté (SEQa).



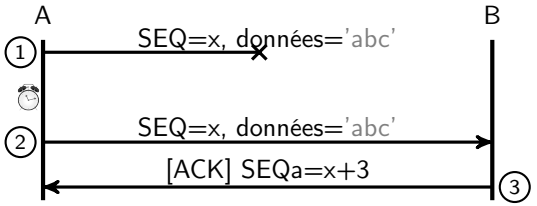
1. A envoie les octets x à $x+2$.
2. B acquitte les octets reçus et indique que le prochain attendu est le $x+3$.
3. A envoie les octets $x+3$ à $x+4$.
4. B acquitte les octets reçus et indique que le prochain attendu est le $x+5$.

- On peut aussi acquitter plusieurs paquets avec un seul acquittement :



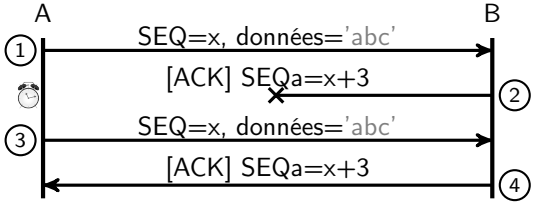
TCP — Temporisations

- utilisation de temporisation pour détecter les pertes de paquets
- Principe : passé un certain délai, si des données n'ont pas été acquittées, on les considère comme perdues et on retransmet.



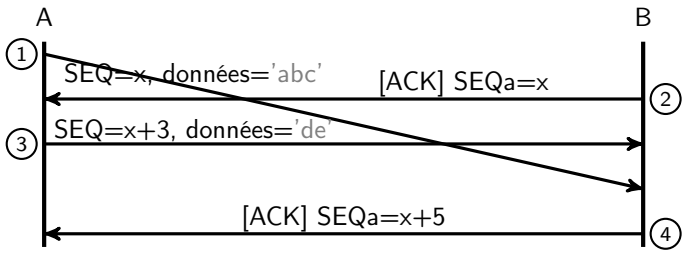
1. A envoie 3 octets de données.
2. Pas d'acquittement dans les temps ⇒ retransmission des 3 octets.

- Autre scénario avec une perte de l'acquittement :



TCP — Déséquencement

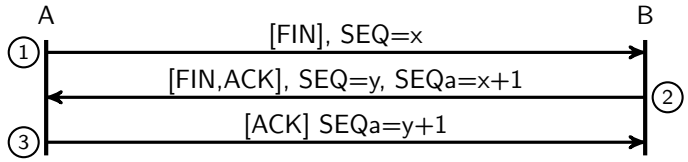
- Possibilité de mémoriser des octets de données non attendus.



1. A envoie 3 octets de données 'abc'.
2. B acquitte mais indique qu'il attend toujours l'octet x.
   Il mémorise les 3 octets.
3. A envoie 2 octets de données 'de'.
4. B acquitte les 5 octets reçus.

TCP — Déconnexion

- Tout octet de données doit avoir été acquitté avant la déconnexion.
- déconnexion en 3 temps



Choix du protocole de transport

- 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 ⇒ on utilise plutôt TCP
    - transfert de fichiers (HTTP, FTP)
    - email (SMTP, IMAP)

    (fiabilité ⇔ données délivrées sans erreur et sans déséquencement)
  - fiabilité secondaire mais critères temporels forts ⇒ on utilise plutôt UDP
    - téléphonie
    - streaming

Plan

3. DHCP — Configuration automatique des hôtes

Configuration des hôtes

- 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)

Le protocole DHCP

- DHCP (**D**ynamic **H**ost **C**onfiguration **P**rotocol)
- première RFC en 1993 (RFC 1531)
- évolution du protocol BOOTP
- protocole de transport utilisé : UDP sur les ports
  - 67 pour les serveurs et relais (voir plus loin)
  - 68 pour les clients

Participants au protocole DHCP

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

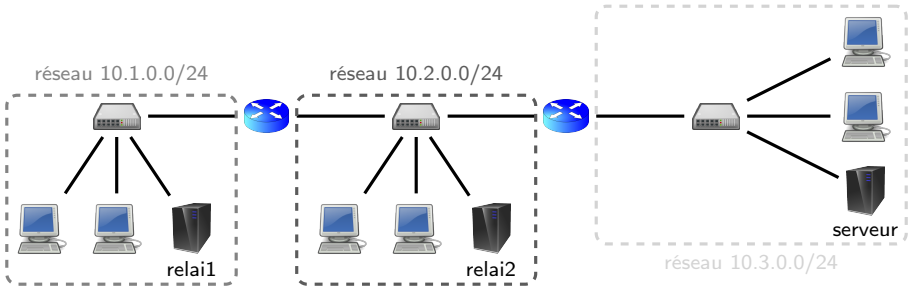
- gère une plage d'IP à distribuer aux clients
- fournit aussi d'autres infos. (routeur, serveur DNS, ...) sur demande
- Remarque : on peut avoir plusieurs serveurs sur un réseau.
  - assure une **tolérance aux pannes** (remplacement d'un serveur défectueux)
  - nécessite des mécanismes de synchronisation 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 ⇒ elles ne passent pas les routeurs ⇒ besoin d'un intermédiaire pour relayer les paquets entre un client et un serveur séparés par un routeur.)

Participants au protocole DHCP — Exemple

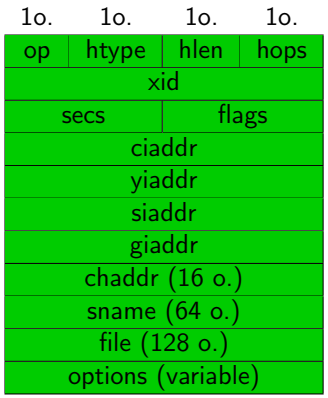


- Sur les réseaux 10.1.0.0/24 et 10.2.0.0/24, toute requête d'un client passe par le relai qui retransmet au serveur : client → relai → serveur.
- Toute réponse suit le chemin inverse : serveur → relai → client.
- Certaines interfaces doivent être configurées manuellement :
  - celles des routeurs
 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

Principaux types de messages

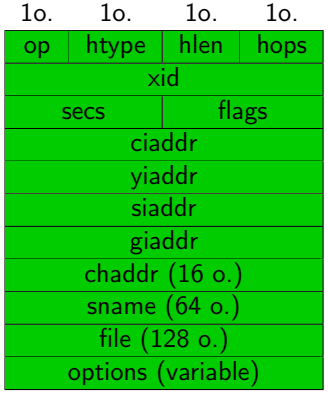
Type	Sens	Contexte(s)
DISCOVER	client→diffusion	demande de bail
OFFER	serveur→client	offre de bail
REQUEST	client→diffusion	acceptation d'une offre
	client→serveur	demande de renouvellement de bail
ACK	serveur→client	REQUEST acceptée
NAK	serveur→client	REQUEST refusée
RELEASE	client→serveur	résiliation du bail (⇔ libération de l'IP)

Structure des messages



- **op** = type d'opération (0 = req., 1 = rép.)
- **htype** = type d'interface (ex : 1 = ethernet)
- **hlen** = longeur des adresses physiques
- **hops** = nombre de relais visités pour traiter le message. initialisé à 0 par le client
- **xid** = identifiant de transaction. choisi aléatoirement par le client. permet au client de reconnaitre les réponses à ses requêtes
- **secs** = secondes écoulées depuis le début du processus de configuration
- **flags** = 1 si le client ne peut pas accepter de paquets unicast (⇒ le serveur sait qu'il doit envoyer les réponses en diffusion)

Structure des messages



- **ciaddr** = client IP address (dans un REQUEST de renouvellement ou un RELEASE)
- **yiaddr** = your IP address (dans un OFFER ou un ACK)
- **siaddr** = server IP address (serveur continuant le processus de conf.)
- **giaddr** = gateway IP address (IP du relai ayant retransmis le message)
- **chaddr** = adresse physique du client
- **sname** = nom du serveur
- **file** = chemin du fichier de démarrage (vide si pas de fichier)

(Par défaut, toutes les IP valent 0.0.0.0.)

Structure des messages — Codage des options

Chaque option est codée sur trois champs :

- code de l'option (1 octet)
- longueur de l'option (1 octet)
- valeur de l'option (selon la longueur)

Par exemple, `03 04 01 02 03 fe` signifie :

- `03` = code de l'option *Router*
- `04` = la valeur de l'option est codée sur 4 octets
- `01 02 03 fe` = valeur de l'option, soit 1.2.3.254

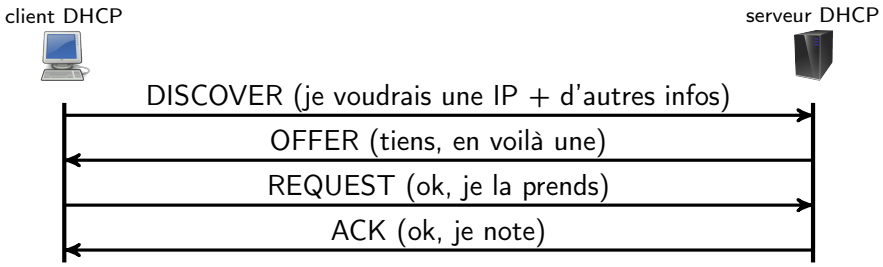
  Rappels :
  - hexadécimal = base 16
  - alphabet hexadécimal = $0, \ldots 9, a, b, c, d, e, f$ ($a = 10, b = 11, \ldots$)
  - donc $fe_{16} = \underbrace{15}_{f} \cdot \underbrace{16}_{\text{base}}{}^{1} + \underbrace{14}_{e} \cdot \underbrace{16}_{\text{base}}{}^{0} = 254$

Structure des messages — Options courantes

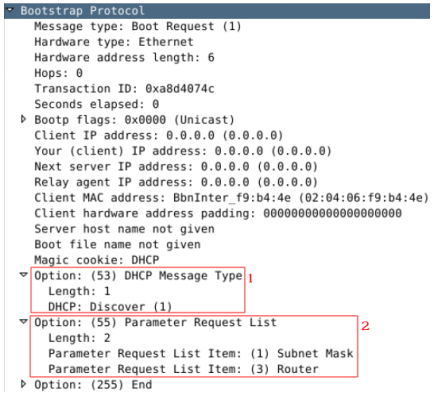
- 53 — **Message type** (obligatoire) — type du message
  1 = DISCOVER 5 = ACK
  2 = OFFER 6 = NAK
  3 = REQUEST 7 = RELEASE
- 255 — **End** (obligatoire, sur 1 octet) — marque la fin des options
- 1 — **Subnet Mask** — le serveur fournit un masque de réseau au client
- 3 — **Router** — le serveur fournit un routeur par défaut au client
- 6 — **Domain Server** — le serveur fournit un serveur DNS au client
- 50 — **Requested IP address** — IP demandée par le client
- 51 — **IP Address Lease time** — durée du bail en sec.
- 54 — **Server identifier** — IP du serveur
- 55 — **Parameter Request List** — codes des options demandées
  Exemple : 37 02 01 03 signifie que le client a demandé deux options :
  - un masque de réseau (option 1 = *Subnet mask*) ;
  - et un routeur (option 3 = *Router*).

Obtention du bail

- Le client obtient une IP et d'autres informations de configuration (masque, routeur, ...) auprès d'un serveur.
- Le serveur mémorise l'IP attribuée pour ne pas l'attribuer à un autre client.
- processus DORA (Discover, Offer, Request, Ack) en 4 temps
- pour obtenir un bail sous unix : `dhclient interface`



Obtention du bail — Message DISCOVER



1 Type du message : 1 = Discover

2 Options demandées au serveur :
  1 = un masque de réseau
  3 = un routeur

Obtention du bail — Message OFFER

1 IP proposée au client

2 IP du serveur qui répond
(option obligatoire dans un OFFER)

3 durée du bail
(option obligatoire dans un OFFER)

4 masque de réseau

(Le client a demandé un routeur mais le serveur n'en a pas à lui fournir.)

Obtention du bail — Message REQUEST


```
Bootstrap Protocol
  Message type: Boot Request (1)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0xa8d4074c
  Seconds elapsed: 0
▸ Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 0.0.0.0 (0.0.0.0)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 0.0.0.0 (0.0.0.0)
  Client MAC address: BbnInter_f9:b4:4e (02:04:06:f9:b4:4e)
  Client hardware address padding: 00000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
▾ Option: (53) DHCP Message Type
    Length: 1
    DHCP: Request (3)
▾ Option: (54) DHCP Server Identifier                    1
    Length: 4
    DHCP Server Identifier: 10.1.0.100 (10.1.0.100)
▾ Option: (50) Requested IP Address                      2
    Length: 4
    Requested IP Address: 10.1.0.2 (10.1.0.2)
▾ Option: (55) Parameter Request List
    Length: 2
    Parameter Request List Item: (1) Subnet Mask
    Parameter Request List Item: (3) Router
▾ Option: (255) End
    Option End: 255
```

1 IP du serveur dont on accepte l'offre (option obligatoire dans un REQUEST pour obtention de bail)
(⇒ si un autre serveur a fait une offre, il sait qu'elle n'est pas retenue)

2 IP demandée
(option obligatoire dans un REQUEST pour obtention de bail)

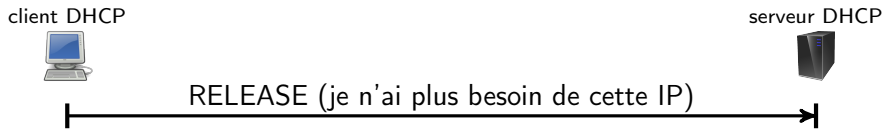
Obtention du bail — Message ACK


```
Bootstrap Protocol
  Message type: Boot Reply (2)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0xa8d4074c
  Seconds elapsed: 0
▸ Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 10.1.0.2 (10.1.0.2)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 0.0.0.0 (0.0.0.0)
  Client MAC address: BbnInter_f9:b4:4e (02:04:06:f9:b4:4e)
  Client hardware address padding: 00000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
▾ Option: (53) DHCP Message Type
    Length: 1
    DHCP: ACK (5)
▾ Option: (54) DHCP Server Identifier                    1
    Length: 4
    DHCP Server Identifier: 10.1.0.100 (10.1.0.100)
▾ Option: (51) IP Address Lease Time
    Length: 4
    IP Address Lease Time: (43200s) 12 hours
▾ Option: (1) Subnet Mask
    Length: 4
    Subnet Mask: 255.255.255.0 (255.255.255.0)
▾ Option: (255) End
    Option End: 255
```

1 mêmes options que dans le OFFER

Résiliation du bail

- Le client prévient le serveur qu'il ne souhaite plus utiliser l'IP qui lui avait été attribuée.
  ⇒ Le serveur peut attribuer cette IP à un autre client.
- pour résilier un bail sous unix : `dhclient -r interface`



Résiliation du bail — Message RELEASE


```
Bootstrap Protocol
  Message type: Boot Request (1)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x64e2417d
  Seconds elapsed: 0
▸ Bootp flags: 0x0000 (Unicast)
  Client IP address: 10.1.0.2 (10.1.0.2)                 1
  Your (client) IP address: 0.0.0.0 (0.0.0.0)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 0.0.0.0 (0.0.0.0)
  Client MAC address: BbnInter_f9:b4:4e (02:04:06:f9:b4:4e)
  Client hardware address padding: 00000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
▾ Option: (53) DHCP Message Type
    Length: 1
    DHCP: Release (7)
▾ Option: (54) DHCP Server Identifier                    2
    Length: 4
    DHCP Server Identifier: 10.1.0.100 (10.1.0.100)
▾ Option: (255) End
    Option End: 255
```

1 IP "libérée"

2 IP du serveur qui avait offert le bail (option obligatoire dans un RELEASE)

Plan

4. SSH — Connexion sécurisée

Présentation du service SSH

- 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.

Se connecter à distance : la commande ssh

`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: ]$
```

Copier des fichiers à distance : la commande scp

`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` $\neq 22$
- Exemples :

`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`

Le chiffrement

- 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**

Le chiffrement symétrique

- 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)

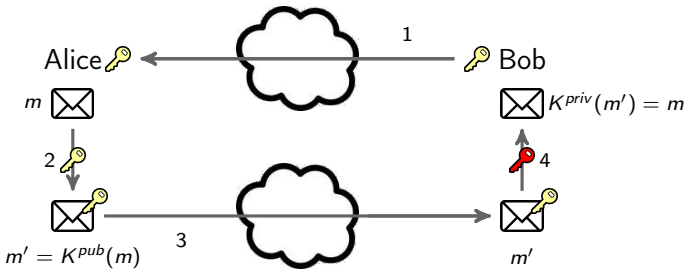
Le chiffrement asymétrique

- 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

Le chiffrement asymétrique — Illustration

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.

Le chiffrement dans SSH

- Observation : impossible d'utiliser un seul type de chiffrement
  - symétrique : comment s'échanger la clé ?
  - asymétrique : trop lent, marcherait uniquement pour des très petits volmues de données
- Principe du chiffrement dans SSH :
  - combiner chiffrement symétrique et asymétrique
  - pour tirer parti des avantages des deux méhodes :
    - 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

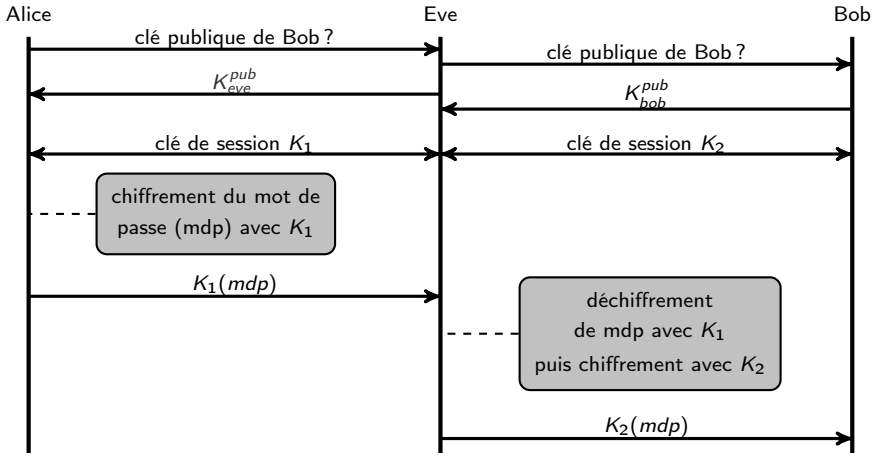
Attaques MITM sur SSH

- 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.

Attaques MITM sur SSH — Exemple

Contexte :

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



Vérification des clés (1/2)

- 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.

Vérification des clés (2/2)

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 !

Plan

5. Services web

Plan

Caractéristiques de HTTP

- HTTP = HyperText Transfer Protocol
- port par défaut : TCP/80
- protocole de transfert de ressources (web)
- client HTTP ≈ 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)

Structure des requêtes HTTP


```
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

Champs de requête usuels

- Host = nom du site web concerné par la requête
  - indispensable quand le serveur web héberge plusieurs sites (voir page 77)
  - champ obligatoire depuis la version 1.1
- Connection = options de connexion (voir page 66)
- 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

Le type de media (ou type MIME)

- 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 ⇒ 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	

Structure des réponses HTTP


```
HTTP/VERSION CODE MESSAGE            une ligne de réponse
Champ1: Valeur1                      N lignes d'en-tête
Champ2: Valeur2
Champ3: Valeur3
...
                                     une ligne vide
corps de la réponse                  la ressource demandée
                                     (éventuellement vide)
```

- `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

Champs de réponse usuels

- 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

Exemple d'échange HTTP

Requête :

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

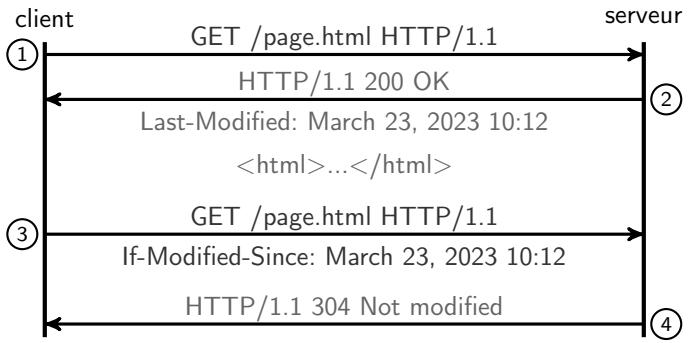
Réponse :

```
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

Utilisation des champs de date de mise à jour



- 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).

De HTTP 1.0 à HTTP 1.1

Différences majeures :

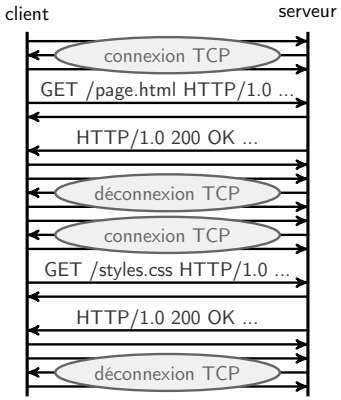
- connexions persistantes (utilisation du champ Connection)
- meilleure gestion du cache
- champ d'en-tête Host obligatoire dans les requêtes
- négociation de contenu (utilisation du champ Accept)

Connexions persistantes

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

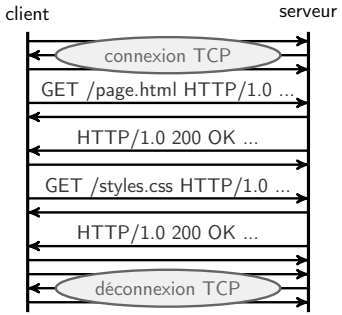
HTTP 1.0

- une connexion TCP par ressource téléchargée



HTTP 1.1

- une connexion TCP pour toutes les ressources téléchargées



Connexions persistantes — Le champ Connection

- `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.

Le cache

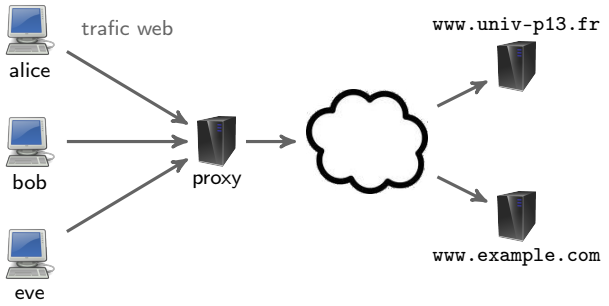
- mettre en cache une ressource = sauvegarder la ressource pour ne pas avoir à recontacter le serveur si elle est redemandée par l'utilisateur
  - ☺ serveur moins chargé
  - ☺ rapidité de chargement pour l'utilisateur
- Types de cache :
  - privé : sur le disque de l'utilisateur
  - partagé (ou public)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

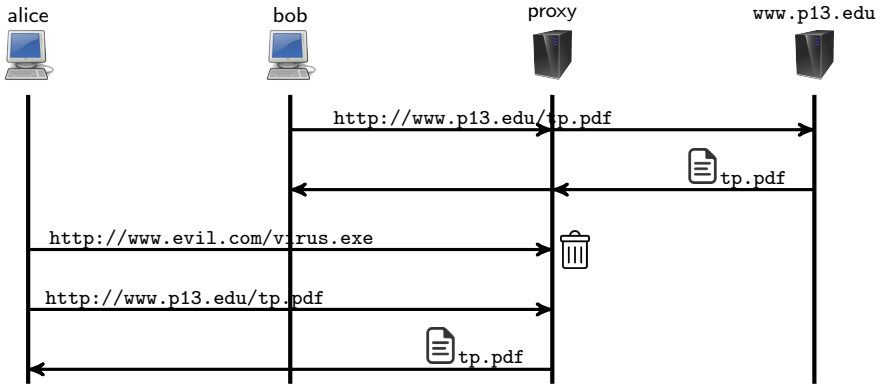
Gestion du cache : le champ Cache-Control

- 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 63)
  - private — la réponse doit être sauvegardée dans un cache privé uniquement
  - public — la réponse peut être sauvegardée dans un cache public
- Exemple :

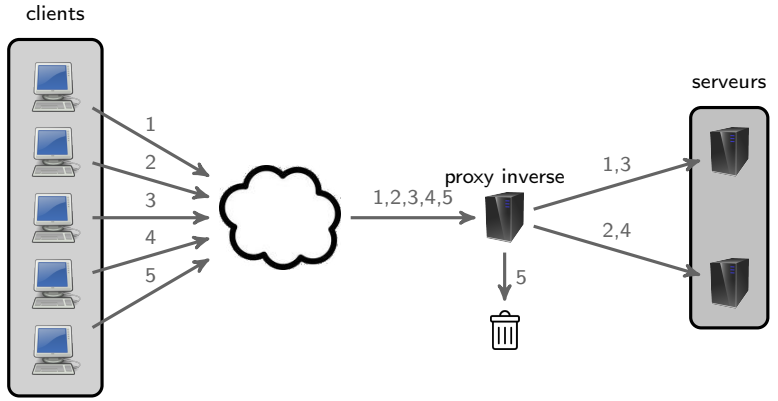
```
Cache-Control: max-age=36000, public, no-cache
```

- 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 ameliorer
  - 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 (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 équitablement entre les serveurs)



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

`https://httpd.apache.org/`

- première version publique en avril 1995
- logiciel libre
- maintenu par la fondation Apache
- environ 20% des serveurs web en février 2024 sont des serveurs apache (source : `https://news.netcraft.com`)
- serveur web multi-sites (ou hôtes virtuels)
- architecture modulaire (un module = une extension du serveur web)
  - exemples de modules :
    - modules d'exécution de scripts php, python ou cgi (voir page 78)
    - module d'authentification
    - module de support des sessions
  - Les modules peuvent être chargés au lancement d'apache.

Fichiers et répertoires principaux (tous dans `/etc/apache2/`) :

- `apache2.conf` — fichier de configuration principal
- `mods-available` — répertoire des fichiers de configuration des modules disponibles
- `mods-enabled` — répertoire des fichiers de configuration des modules activés (liens symboliques vers les fichiers de `mods-available`)
- `sites-available` — répertoire de fichiers de configuration des hôtes virtuels disponibles
- `sites-enabled` — répertoire des fichiers de configuration des hôtes virtuels activés (liens symboliques vers les fichiers de `sites-available`)

(valable sur debian ≥ 9)

Tous ces fichiers contiennent des **directives** qui modifient le comportement du serveur.

- DocumentRoot — répertoire contenant les ressources du serveur

```
DocumentRoot /var/www/html
```

- DirectoryIndex — page d'accueil d'un répertoire

```
DirectoryIndex index.html
```

- Include — inclusion d'un autre fichier de configuration

```
Include un_autre_fichier.conf
```

- User — utilisateur système qui lance le service (généralement, www-data)

```
User www-data
```

- ErrorDocument — document renvoyé par le serveur en cas d'erreur

```
ErrorDocument 404 /errors/404.html
```

- ErrorLog — fichier de journal des erreurs apache

```
ErrorLog /var/log/apache2.log
```

Les hôtes virtuels

- un hôte virtuel ≈ un site web hébergé par le serveur
- fichiers de configuration dans `/etc/apache2/sites-available`
- Pour déclarer un hôte virtuel :

```
1 <VirtualHost ip:port>
2    ServerName nom.du.site
3    ...
4    </VirtualHost>
```

1. IP et port de réception des requêtes (* = n'importe laquelle/lequel)
2. définit le nom du site
3. on place ici les directives spécifiques à cet hôte.

- Exemple de deux sites hebergés accessibles par le port 80 :

```
1 <VirtualHost *:80>
2    ServerName www.truc.com
3    DocumentRoot /var/www/html/truc
4 </VirtualHost>
5 <VirtualHost *:80>
6    ServerName www.machin.com
7    DocumentRoot /var/www/html/machin
8 </VirtualHost>
```

Les scripts CGI

- CGI = Common Gateway Interface
- méthode d'interface entre un serveur Web et des programmes générateurs de contenus
- par extension, un script exécuté par le serveur qui renvoie du contenu HTTP
- Un script CGI peut être écrit dans n'importe quel langage (python, bash, C, java, ...).

Les scripts CGI — Sortie

Un script CGI **doit** écrire sur la sortie standard :

1. une partie de l'en-tête du message HTTP renvoyé au client
   - Le type de contenu renvoyé (champ `Content-Type`) doit obligatoirement apparaître.
2. une ligne vide
3. le corps de la réponse HTTP (éventuellement vide)

La sortie d'un CGI correspond donc à un bout du message HTTP renvoyé par le serveur au client (partie de l'en-tête + corps de la réponse).

Les scripts CGI — Exemple


```
#!/bin/bash

echo "Content-Type: text/html"
echo ""
echo "<html>"
echo "<head><title>Informations</title></head>"
echo "<body>...</body>"
echo "</html>"
```