

# INF344 – Données du Web

## Internet et HTTP

---

Antoine Amarilli

2 mai 2018

# Vue générale

- Plusieurs **échelles** (locale et globale)
- **Pile** de protocoles
- Messages **imbriqués**

Pour: 01:23:45:67:89:ab

Pour: 12.34.56.78

Page: 1 sur 3

```
<html>
```

```
  <head>
```

```
    ...
```

```
  </head>
```

```
  <body> ...
```

# Le modèle OSI

#	Couche	Exemples	Fonctionnalités
7	Application	<b>HTTP</b> , FTP, SMTP	tâche utilisateur de haut niveau
4	Transport	<b>TCP</b> , UDP, ICMP	sessions, fiabilité, fragmentation
3	Réseau	<b>IPv4, IPv6</b>	routage, adressage, non fiable
2	Lien	Ethernet, 802.11	données fiables, adresses locales
1	Physique	Ethernet, 802.11	échange physique, non fiable

→ Plus la couche est **basse**, plus l'enveloppe est à l'**extérieur**

# Table des matières

Modèle OSI

Couches basses

Couches hautes

HTTP

En-têtes client

En-têtes serveur

Autres aspects

## IP (Internet Protocol), couche 3

- Donner des **adresses** aux machines
- **Router** des paquets entre ces adresses
- Déterminer la position **géographique** d'une machine

	<b>Année</b>	<b>Exemple</b>	<b>Adresses</b>	<b>Trafic</b>
IPv4	1981	208.80.152.201	$\leq 2^{32}$	80%
IPv6	1998	2620:0:860:ed1a::1	$\leq 2^{128}$	20% <sup>1</sup>

- **Network Address Translation** pour pallier la pénurie d'adresses

→ On peut envoyer des messages à une adresse.

1. <https://www.google.com/intl/en/ipv6/statistics.html>, mai 2018

# DNS (Domain Name System), intermède

- Adresses IP **routables** mais difficiles à mémoriser
  - Service pour convertir **www.wikipedia.org** en **208.80.152.201**
  - Hiérarchie : **org**, **wikipedia.org**, **en.wikipedia.org**, etc.
  - Résolution **hiérarchique** ; **gTLDs**, registrars, coûts
  - **Cache** à différents niveaux
  - Problèmes de **sécurisation**
  - **Indirection** :
    - Plusieurs adresses par nom de domaine (services multiples, répartition de charge)
    - Plusieurs noms de domaine par adresse (virtual hosts)
- Qui **gère** le DNS ? Qui a **droit** à un nom de domaine ?
- DNS alternatifs, autres technologies décentralisées (Namecoin...)

# DNS (Domain Name System), intermède

- Adresses IP **routables** mais difficiles à mémoriser
  - Service pour convertir **www.wikipedia.org** en **208.80.152.201**
  - Hiérarchie : **org**, **wikipedia.org**, **en.wikipedia.org**, etc.
  - Résolution **hiérarchique** ; **gTLDs**, registrars, coûts
  - **Cache** à différents niveaux
  - Problèmes de **sécurisation**
  - **Indirection** :
    - Plusieurs adresses par nom de domaine (services multiples, répartition de charge)
    - Plusieurs noms de domaine par adresse (virtual hosts)
- Qui **gère** le DNS ? Qui a **droit** à un nom de domaine ?
- DNS alternatifs, autres technologies décentralisées (Namecoin...)

(Démonstration : résolution DNS avec **dig**.)

# DNS (Domain Name System), intermède

- Adresses IP **routables** mais difficiles à mémoriser
  - Service pour convertir **www.wikipedia.org** en **208.80.152.201**
  - Hiérarchie : **org**, **wikipedia.org**, **en.wikipedia.org**, etc.
  - Résolution **hiérarchique** ; **gTLDs**, registrars, coûts
  - **Cache** à différents niveaux
  - Problèmes de **sécurisation**
  - **Indirection** :
    - Plusieurs adresses par nom de domaine (services multiples, répartition de charge)
    - Plusieurs noms de domaine par adresse (virtual hosts)
- Qui **gère** le DNS ? Qui a **droit** à un nom de domaine ?
- DNS alternatifs, autres technologies décentralisées (Namecoin...)

(Démonstration : résolution DNS avec **dig**.)

→ **On peut envoyer des messages à une machine nommée.**



## TCP (Transmission Control Protocol), couche 4

- IP n'est pas **fiable**
  - TCP fournit des **accusés de réception**
- IP **limite la taille**
  - TCP permet de **fragmenter**
- IP peut **mélanger les paquets**
  - TCP garantit que les paquets arrivent **dans l'ordre**
- IP n'est pas **multiplexé**
  - TCP introduit des **sessions** et des **ports**. (e.g. 80 pour le Web... mais possible de forcer : `http://localhost:8080/`)

# TCP (Transmission Control Protocol), couche 4

- IP n'est pas **fiable**
  - TCP fournit des **accusés de réception**
- IP **limite la taille**
  - TCP permet de **fragmenter**
- IP peut **mélanger les paquets**
  - TCP garantit que les paquets arrivent **dans l'ordre**
- IP n'est pas **multiplexé**
  - TCP introduit des **sessions** et des **ports**. (e.g. 80 pour le Web... mais possible de forcer : `http://localhost:8080/`)

(Démonstration : communication TCP avec netcat.)

# TCP (Transmission Control Protocol), couche 4

- IP n'est pas **fiable**
  - TCP fournit des **accusés de réception**
- IP **limite la taille**
  - TCP permet de **fragmenter**
- IP peut **mélanger les paquets**
  - TCP garantit que les paquets arrivent **dans l'ordre**
- IP n'est pas **multiplexé**
  - TCP introduit des **sessions** et des **ports**. (e.g. 80 pour le Web... mais possible de forcer : `http://localhost:8080/`)

(Démonstration : communication TCP avec netcat.)

→ **On peut avoir un canal de communication avec une machine.**

# Table des matières

Modèle OSI

Couches basses

Couches hautes

HTTP

En-têtes client

En-têtes serveur

Autres aspects

## TLS (Transport Layer Security), couche 5-6

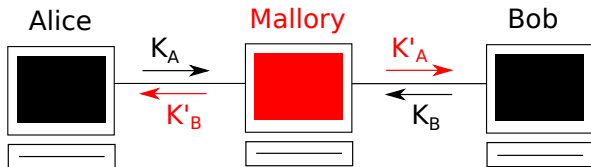
- Échanges en **clair**! Risques (confidentialité, commerce...).
- **Trois garanties** : intégrité, authenticité, confidentialité
- HTTP + TLS = HTTPS. URL en `https://`. Cadenas...
- Cryptographie **asymétrique** : deux machines sûres peuvent communiquer de façon sûre contre des attaquants passifs
- **Confidentialité persistente** (perfect forward secrecy); HSTS
- Cryptographie symétrique pour des raisons de performance
- Ne protège pas les **métadonnées**!
- Possibilité de **fuites** (taille, délai...)
- Impact légal sur la cryptographie

# TLS (Transport Layer Security), couche 5-6

- Échanges en **clair**! Risques (confidentialité, commerce...).
- **Trois garanties** : intégrité, authenticité, confidentialité
- HTTP + TLS = HTTPS. URL en `https://`. Cadenas...
- Cryptographie **asymétrique** : deux machines sûres peuvent communiquer de façon sûre contre des attaquants passifs
- **Confidentialité persistente** (perfect forward secrecy); HSTS
- Cryptographie symétrique pour des raisons de performance
- Ne protège pas les **métadonnées**!
- Possibilité de **fuites** (taille, délai...)
- Impact légal sur la cryptographie

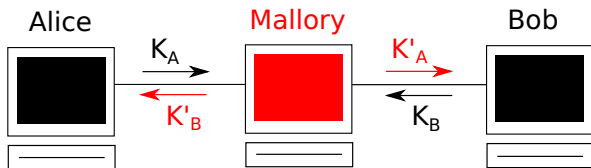
(Démonstration : communication chiffrée avec `ncat -ssl`, tentatives d'interception infructueuses avec `wireshark`.)

- Attaques actives, attaque de l'homme du milieu



- Certificats (et messages d'erreur associés)
- **Qui** sont les autorités de confiance ? ( $\geq 80$  dans Chrome)
- Let's Encrypt <https://letsencrypt.org/>

- Attaques actives, attaque de l'homme du milieu



- Certificats (et messages d'erreur associés)
- **Qui** sont les autorités de confiance ? ( $\geq 80$  dans Chrome)
- Let's Encrypt <https://letsencrypt.org/>

→ On a un canal de communication chiffré entre deux machines



# HTTP (HyperText Transfer Protocol), couche 7

- Le **World Wide Web** (WWW)
- **Protocole** de la navigation Web

→ Pour résumer :

- Une machine **client**
- Un logiciel client : le **navigateur**
- Une machine **serveur**
- Un logiciel serveur : le **serveur Web**
- Un canal de communication fiable et chiffré

# Table des matières

Modèle OSI

Couches basses

Couches hautes

HTTP

En-têtes client

En-têtes serveur

Autres aspects

# Présentation générale

- Standardisé par l'**Internet Engineering Task Force** (IETF) et le **World Wide Web Consortium** (W3C)
- Protocole **documenté** : RFC 2616 (114 pages, 1999, + followups)
- **Extensions** : WebSockets, nouveaux en-têtes...
- Nouvelle version : **HTTP/2**
  - Originellement proposé par Google
  - Documenté dans la RFC 7540 (96 pages, 2015)
  - Meilleure efficacité (compression et multiplexing)
  - Supporté par **26%** des sites Web aujourd'hui<sup>2</sup>
  - Aussi : **QUIC** (remplacement expérimental de TCP)

---

2. <https://w3techs.com/technologies/details/ce-http2/all/all>, mai 2018

# La requête HTTP

- Du **client** vers le **serveur**, connexion TCP (+TLS)

```
GET /wiki/Telecom_ParisTech HTTP/1.1
```

```
Host: en.wikipedia.org
```

→ [http://en.wikipedia.org/wiki/Telecom\\_ParisTech](http://en.wikipedia.org/wiki/Telecom_ParisTech)

**Méthode** Plusieurs possibilités :

GET La plus fréquente

POST Formulaires et effet de bord

HEAD Méta-infos seulement

**autres** PUT, DELETE...

**Chemin** C'est celui de l'URL

**Version** Ici, 1.1 (essentiellement la seule)

**En-têtes** Infos supplémentaires (cf. plus tard)

**Corps** Passer des paramètres avec POST

# La requête HTTP

- Du **client** vers le **serveur**, connexion TCP (+TLS)

```
GET /wiki/Telecom_ParisTech HTTP/1.1
```

```
Host: en.wikipedia.org
```

→ `http://en.wikipedia.org/wiki/Telecom_ParisTech`

**Méthode** Plusieurs possibilités :

GET La plus fréquente

POST Formulaires et effet de bord

HEAD Méta-infos seulement

**autres** PUT, DELETE...

**Chemin** C'est celui de l'URL

**Version** Ici, 1.1 (essentiellement la seule)

**En-têtes** Infos supplémentaires (cf. plus tard)

**Corps** Passer des paramètres avec POST

(Démonstration avec `netcat`)

# La réponse HTTP

- Du **serveur** vers le client, dans la même connexion

```
HTTP/1.1 200 OK
```

```
Content-Type: text/html; charset=UTF-8
```

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    (...)
```

- **Code d'état** et **explication**
- **En-têtes**
- **Réponse** (e.g., corps de la page)

**1XX** Information

**2XX** Succès

- 200 : OK

**3XX** Redirection

- 301 : permanente
- 302 : temporaire
- 304 : cf. le cache

**4XX** Erreur client

- 400 : erreur de syntaxe
- 401 : authentification nécessaire
- 403 : interdit
- 404 : non trouvé

**5XX** Erreur serveur

- 500 : erreur interne du serveur

# Chemins et paramètres

- Les chemins sont **hiérarchiques** (séparateur : /)
- Conventions Unix : `https://en.wikipedia.org/./wiki/./`
- Ajout possible de paramètres **non hiérarchiques**
- **Exemple** : `https://www.google.com/search?q=telecom&ie=utf-8&oe=utf-8&client=iceweasel-a`
- Caractères spéciaux **encodés** :  
`https://fr.wikipedia.org/wiki/T%C3%A9l%C3%A9com_ParisTech`



# Table des matières

Modèle OSI

Couches basses

Couches hautes

HTTP

En-têtes client

En-têtes serveur

Autres aspects

- **Plusieurs** sites Web par machine :
  - `fr.wikipedia.org` et `en.wikipedia.org`
- Le nom de domaine est **perdu** après la résolution DNS
- En-tête pour **repréciser** le domaine attendu

Host: `en.wikipedia.org`

- **Plusieurs** sites Web par machine :
  - `fr.wikipedia.org` et `en.wikipedia.org`
- Le nom de domaine est **perdu** après la résolution DNS
- En-tête pour **repréciser** le domaine attendu  
`Host: en.wikipedia.org`

(Démonstration : requête à `en.wikipedia.org` avec différents Hosts.)

- Identifier le **navigateur** utilisé
- Est-ce un **ordinateur** ou un **téléphone**?
- Est-ce un **humain** ou un **robot**?
- Permet de contourner des **bugs**
- Pas de **garanties**!
- Délires historiques (tout le monde dit "Mozilla/5.0")  
User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:17.0)  
Gecko/20130810 Firefox/17.0 Iceweasel/17.0.8

- Identifier le **navigateur** utilisé
- Est-ce un **ordinateur** ou un **téléphone** ?
- Est-ce un **humain** ou un **robot** ?
- Permet de contourner des **bugs**
- Pas de **garanties** !
- Délires historiques (tout le monde dit "Mozilla/5.0")  
User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:17.0)  
Gecko/20130810 Firefox/17.0 Iceweasel/17.0.8

(Démonstration : interception d'une requête de Firefox avec mitmproxy)

## En-tête Accept et Accept-\*

- Plusieurs **versions alternatives** d'une ressource :
  - Type de fichier** Différents formats de page Web...
  - Langue** Anglais, français...
  - Encodage** Compression à la volée
- **Négociation** de contenu
- Indique plusieurs **choix** et des **priorités**
- En pratique, le type de fichier n'est pas vraiment utilisé

```
Accept: text/html,application/xhtml+xml,
```

```
application/xml;q=0.9,*/*;q=0.8
```

```
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
```

- HTTP 1.0 **fermait** la connexion après une requête
- Inefficace, vu que **TCP** permet de gérer ça!
- En-tête **Connection** pour indiquer si la connexion doit/va rester ouverte ou non (client et serveur)
- **Réutiliser** la connexion après pour d'autres requêtes
- **HTTP 1.1** : la connexion reste ouverte par défaut
- **Timeout** pour éviter de gaspiller des ressources
- **Pipelining** : envoyer **plusieurs requêtes**, recevoir les réponses dans l'ordre
  - Support mauvais en pratique, pas activé par les navigateurs
  - Avec HTTP/2, **multiplexing** : requêtes dans le **désordre**

- Indique de **quelle page** le navigateur vient
- Transmis après clic sur un **lien** ou soumission d'un **formulaire**
- Pas de **garanties!**
- Quid de la **vie privée?**

Referer: `https://en.wikipedia.org/wiki/Telecom_ParisTech`



- Demander uniquement une **certaine partie** du fichier
- Utile pour reprendre un **téléchargement interrompu** !
- **Anecdote** : en 2011, une faille pour planter le serveur Web le plus utilisé (Apache) en demandant une grande quantité de parties chevauchées (CVE-2011-3192.)

Range: bytes=0-42

# Table des matières

Modèle OSI

Couches basses

Couches hautes

HTTP

En-têtes client

En-têtes serveur

Autres aspects

- Identifie le **serveur**
- (Toujours) pas de **garanties!**

- Identifie le **serveur**
- (Toujours) pas de **garanties!**

(Démonstration : réponse à la requête Firefox avec `mitmproxy`.)

## En-tête Content-Type et Content-Length

- Plusieurs **types de ressources** sur le Web : pages Web, mais aussi images, texte, PDF, multimédia...
  - Nécessité d'identifier le **type** (quel document, quel format?)  
**Internet media type**, historiquement Multipurpose Internet Mail Extensions (MIME)
- Plusieurs encodages (formats de représentation du texte) :
  - Nécessité de préciser l'encodage pour certains documents
- Afficher une **barre de progression** ?
  - Préciser la **longueur** à l'avance

Content-Type: text/html; charset=UTF-8

Content-Length: 4242

# Encodages

Un encodage va **numéroter** les caractères et **encoder** les numéros

**ASCII** 128 caractères. Pas d'accents. 7 bits

**ISO-8859** 128 caractères en plus pour compléter ASCII; 1 octet  
Français : ISO-8859-1 alias Latin-1

**Unicode** Numéroter **tous** les caractères (> 100 000)

**UTF-32** Représenter les caractères Unicode; 4 octets

**UTF-8** Compatible avec ASCII; longueur variable

---

	Encodage ASCII		Encodage Latin-1		Encodage UTF-8	
	nb	représentation	nb	représentation	nb	représentation
A	65	01000001	65	01000001	65	01000001
é	-	-	233	11101001	233	11000011 10101001
ñ	-	-	-	-	324	11000101 10000011

---

→ En pratique : **utiliser UTF-8**

# Table des matières

Modèle OSI

Couches basses

Couches hautes

HTTP

En-têtes client

En-têtes serveur

Autres aspects

# Authentication HTTP Basic

- Mécanisme simple de **contrôle d'accès**
- Le **serveur** :
  - répond à une requête avec **401 Not Authorized**
  - identifie un **domaine** d'authentification avec l'en-tête  
`WWW-Authenticate: Basic realm="foo"`
- Le **client** :
  - encode **réversiblement** login et mot de passe : *encoding*
  - **renouvelle** la requête avec l'en-tête :  
`Authorization: Basic encoding`

→ Aucune **sécurité** si on n'utilise pas HTTPS!



# Authentification HTTP Digest

- Le serveur envoie un **nonce** (message aléatoire) avec le domaine
  - Le client envoie un haché **non réversible** du nonce concaténé aux informations d'authentification
  - Le mot de passe ne **circule pas** en clair
  - Le haché utilisé est **MD5** : vieux et pas forcément sûr
  - Basic et Digest sont **difficiles à distinguer** pour l'utilisateur
- Authentification HTTP **peu utilisée** car peu flexible

- **Proxy** (serveur mandataire) : effectue (ou relaie) des requêtes Web pour le compte d'autrui
- Peut s'utiliser côté **client** ou côté **serveur**
- Nombreuses utilisations :
  - **Filtrer** ou **censurer** le Web (employeurs, écoles, régimes totalitaires, contrôle parental, etc.)
  - Conserver un **journal** de l'activité
  - Conserver un **cache** (cache commun à plusieurs utilisateurs)
  - **Anonymisation** (cacher la véritable origine de la requête)
  - Autres **modifications** : traduction, etc.
- Avec SSL, c'est **plus difficile** !

- **Sauvegarder** le résultat d'une requête pour la réutiliser
- Plusieurs caches possibles : proxies, ou le navigateur
- Serveur :

`Cache-Control` Indiquer si on peut conserver en cache :

`public, private, no-cache/no-store`

`Expires` Date d'**expiration**

`ETag` Identifiant de **version**

- Client :

`Pragma: no-cache` Interdire aux **proxies** d'utiliser leur cache

`If-Modified-Since` Si modifié depuis une certaine **date**

`If-None-Match` Si l'identifiant de **version** a changé

# Cookies

- Pas de **sessions** en HTTP
- Le serveur peut demander au client de **mémoriser une valeur** :  
`Set-Cookie: nom=valeur; option1; option2 :`
  - `expires` date d'**expiration** (peut être lointaine)
  - `domain` limite la **portée** (domaine, sous-domaine)
  - `path` limite la **portée** (sous-chemin)
  - `secure` seulement sur SSL
  - `httpOnly` pas lisible par JavaScript
- Le client **fournit la valeur** avec chaque requête :  
`Cookie: nom=valeur`
- Le client peut **lire** et **modifier** des cookies (pas de garanties) et les **effacer** (vie privée)

- Stocker un **identifiant de session** opaque. (Interception?)
- Garder l'utilisateur **connecté** sur de longues périodes
- **Traquer** un utilisateur de façon unique (même sans compte)
- Risques :
  - **Sécurité** Il ne faut pas donner un cookie au mauvais site!
  - **Vie privée** Cf autre cours

- Matériel de cours inspiré de notes par Pierre Senellart et Georges Gouriten
- Merci à Pierre Senellart pour sa relecture