

# Protocoles dans le Web

## 1 Se connecter au réseau local : DHCP

Un réseau local dispose d'un certain nombre d'adresses IP, et son serveur connaît les adresses libres. Un serveur peut donc attribuer une IP à toute machine désirant s'y connecter. Un client peut désirer une adresse IP spécifique (attribution manuelle), ou alors laisser le serveur lui attribuer une adresse libre (allocation automatique). Aujourd'hui, une telle allocation est dynamique, et valable uniquement pendant la durée d'un bail (quelques heures).

No.	Time	Source	Destination	Protoc	Length	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	364	DHCP Request - Transaction ID 0xe4122f07
2	0.012585	172.21.21.88	172.21.1.114	DHCP	350	DHCP ACK - Transaction ID 0xe4122f07

Client MAC address: IntelCor_37:6f:18 (e0:d4:e8:37:6f:18)	0000	ff ff ff ff ff ff e0 d4	e8 37 6f 18 08 00 45 00	.....7o...E..
Client hardware address padding: 00000000000000000000	0010	01 5e a2 4d 00 00 80 11	00 00 00 00 00 00 ff ff	..^..M.....
Server host name not given	0020	ff ff 00 44 00 43 01 4a	ff 23 01 01 06 00 e4 12	...D.C.J.#.....
Boot file name not given	0030	2f 07 00 00 00 00 00 00	00 00 00 00 00 00 00 00	/.....
Magic cookie: DHCP	0040	00 00 00 00 00 00 e0 d4	e8 37 6f 18 00 00 00 00	.....7o.....
Option: (53) DHCP Message Type (Request)	0050	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Length: 1	0060	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
DHCP: Request (3)	0070	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Option: (61) Client identifier	0080	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Length: 7	0090	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Hardware type: Ethernet (0x01)	00a0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Client MAC address: IntelCor_37:6f:18 (e0:d4:e8:37:6f:18)	00b0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Option: (50) Requested IP Address (172.21.1.114)	00c0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Length: 4	00d0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Requested IP Address: 172.21.1.114	00e0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Option: (12) Host Name	00f0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Length: 15	0100	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
Host Name: LAPTOP-MG4V03CA	0110	00 00 00 00 00 00 63 82	53 63 35 01 03 3d 07 01	.....c..Sc5...=...
Option: (81) Client Fully Qualified Domain Name	0120	e0 d4 e8 37 6f 18 32 04	ac 15 01 72 0c 0f 4c 41	...7o.2...r...LA
Length: 18	0130	50 54 4f 50 2d 4d 47 34	56 4f 33 43 41 51 12 00	PTOP-MG4 V03CAQ...
Flags: 0x00	0140	00 00 4c 41 50 54 4f 50	2d 4d 47 34 56 4f 33 43	..LAPTOP -MG4V03C
A-RR result: 0	0150	41 3c 08 4d 53 46 54 20	35 2e 30 37 0e 01 03 06	Ac<MSFT 5.07....
PTR-RR result: 0	0160	0f 1f 21 2b 2c 2e 2f 77	79 f9 fc ff	!+,./w y...
Client name: LAPTOP-MG4V03CA				

FIGURE 1 – Exemple de trame DHCP interceptée pour la connexion au domaine udcpp

La machine envoie alors une trame avec une adresse IP 0.0.0.0, et un message de requête, en ajoutant des options : dans la figure 1, l'adresse 172.21.1.114 est demandée. Le serveur donne alors sa réponse, positive dans le cas de la figure 2, et communique alors les éléments essentiels à la session : l'adresse IP du client, la durée du bail, l'adresse du routeur, ici 172.21.21.88, et l'adresse du serveur DHCP 193.48.31.11, le masque de sous-réseau...

## 2 Récupérer l'adresse MAC de l'interface : ARP

Les applications ne connaissent que l'adresse logique (IP), mais dans le réseau, c'est l'adresse physique (MAC) qui est utilisée. Il faut donc que la couche IP puisse établir une

	Time	Source	Destination	Protocol	Length	Info
	1 0.000000	0.0.0.0	255.255.255.255	DHCP	364	DHCP Request - Transaction ID 0xe4122f07
	2 0.012585	172.21.21.88	172.21.1.114	DHCP	350	DHCP ACK - Transaction ID 0xe4122f07

Magic cookie: DHCP	0000	e0 d4 e8 37 6f 18 b4 0c	25 e0 40 10 08 00 45 40	...7o... % @...E@
Option: (53) DHCP Message Type (ACK)	0010	01 50 01 00 00 00 7f 11	ca 68 ac 15 15 58 ac 15	·P·...· ·h·...X·
Length: 1	0020	01 72 00 43 00 44 01 3c	64 70 02 01 06 00 e4 12	·r·C·D·< dp·...·
DHCP: ACK (5)	0030	2f 07 00 00 00 00 00 00	00 00 ac 15 01 72 00 00	/·...·...·r·
Option: (58) Renewal Time Value	0040	00 00 ac 15 15 58 e0 d4	e8 37 6f 18 00 00 00 00	·...·X· ·7o·...·
Length: 4	0050	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Renewal Time Value: (10800s) 3 hours	0060	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Option: (59) Rebinding Time Value	0070	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Length: 4	0080	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Rebinding Time Value: (18900s) 5 hours, 15 minutes	0090	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Option: (51) IP Address Lease Time	00a0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Length: 4	00b0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
IP Address Lease Time: (21600s) 6 hours	00c0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Option: (54) DHCP Server Identifier (193.48.31.11)	00d0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Length: 4	00e0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
DHCP Server Identifier: 193.48.31.11	00f0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Option: (1) Subnet Mask (255.255.0.0)	0100	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	·...·...·
Length: 4	0110	00 00 00 00 00 00 63 82	53 63 35 01 05 3a 04 00	·...·ac· Sc5·...·
Subnet Mask: 255.255.0.0	0120	00 2a 30 3b 04 00 00 49	d4 33 04 00 00 54 60 36	·*0;·...I·3·...T`6
Option: (81) Client Fully Qualified Domain Name	0130	04 c1 30 1f 0b 01 04 ff	ff 00 00 51 03 00 ff ff	·0·...·Q·...·
Length: 3	0140	03 04 ac 15 15 58 06 08	c1 30 1f cb c1 30 1d f8	·...·X· ·0·...0·
Flags: 0x00	0150	0f 0b 75 64 63 70 70 2e	70 72 69 76 00 ff	·udcpp· priv·
A-RR result: 255				
PTR-RR result: 255				
Option: (3) Router				
Length: 4				
Router: 172.21.21.88				
Option: (6) Domain Name Server				
Length: 8				
Domain Name Server: 193.48.31.203				

FIGURE 2 – Réponse du serveur.

correspondance entre l'adresse logique et l'adresse physique : c'est la résolution d'adresses, réalisée par l'*Address Resolution Protocol*. Le client effectue une diffusion globale pour que le routeur lui transmette son adresse physique.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.031195	IntelCor_37:6f:...	Broadcast	ARP	42	Who has 172.21.21.88? Tell 172.21.1.114
4	0.034852	PaloAlto_e0:40:...	IntelCor_37:6f:18	ARP	56	172.21.21.88 is at b4:0c:25:e0:40:10
14	0.082659	IntelCor_37:6f:...	Broadcast	ARP	42	Who has 172.21.21.88? Tell 172.21.1.114

> Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)	0000	ff ff ff ff ff ff e0 d4	e8 37 6f 18 08 06 00 01
> Ethernet II, Src: IntelCor_37:6f:18 (e0:d4:e8:37:6f:18), Dst: Broadcast	0010	08 00 06 04 00 01 e0 d4	e8 37 6f 18 ac 15 01 72
> Address Resolution Protocol (request)	0020	00 00 00 00 00 00 ac 15	15 58
Hardware type: Ethernet (1)			
Protocol type: IPv4 (0x0800)			
Hardware size: 6			
Protocol size: 4			
Opcode: request (1)			
Sender MAC address: IntelCor_37:6f:18 (e0:d4:e8:37:6f:18)			
Sender IP address: 172.21.1.114			
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)			
Target IP address: 172.21.21.88			

FIGURE 3 – Requête ARP du client

### 3 Retrouver l'IP du serveur : DNS

Il est relativement difficile de retenir et de manipuler les adresses IP des différents serveurs qui hébergent leurs sites Internet. Le *Domain Name System* désigne une base de données, fonctionnant à partir d'UDP (port 53), et qui associe chaque adresse IP à une

arp						
No.	Time	Source	Destination	Protocol	Length	Info
3	0.031195	IntelCor_37:6f:...	Broadcast	ARP	42	Who has 172.21.21.88? Tell 172.21.1.114
4	0.034852	PaloAlto_e0:40:...	IntelCor_37:6f:18	ARP	56	172.21.21.88 is at b4:0c:25:e0:40:10
14	0.082659	IntelCor_37:6f:...	Broadcast	ARP	42	Who has 172.21.21.88? Tell 172.21.1.114

> Frame 4: 56 bytes on wire (448 bits), 56 bytes captured (448 bits)	0000	e0 d4 e8 37 6f 18 b4 0c 25 e0 40 10 08 06 00 01
> Ethernet II, Src: PaloAlto_e0:40:10 (b4:0c:25:e0:40:10), Dst: Intel	0010	08 00 06 04 00 02 b4 0c 25 e0 40 10 ac 15 15 58
▼ Address Resolution Protocol (reply)	0020	e0 d4 e8 37 6f 18 ac 15 01 72 00 00 00 00 00 00
Hardware type: Ethernet (1)	0030	00 00 00 00 00 00 00 00
Protocol type: IPv4 (0x0800)		
Hardware size: 6		
Protocol size: 4		
Opcode: reply (2)		
Sender MAC address: PaloAlto_e0:40:10 (b4:0c:25:e0:40:10)		
Sender IP address: 172.21.21.88		
Target MAC address: IntelCor_37:6f:18 (e0:d4:e8:37:6f:18)		
Target IP address: 172.21.1.114		

FIGURE 4 – Réponse du serveur

adresse symbolique. On parcourt l'arbre de l'espace de nommage (voir fig. 5) en partant de la racine, puis on avance vers les domaines de niveau 1, puis vers les sous-domaines, jusqu'à compléter le nommage de la ressource. L'adresse symbolique ne peut pas dépasser 255 caractères, et la RFC 1032 préconise de limiter à 12 caractères le nom d'un noeud.

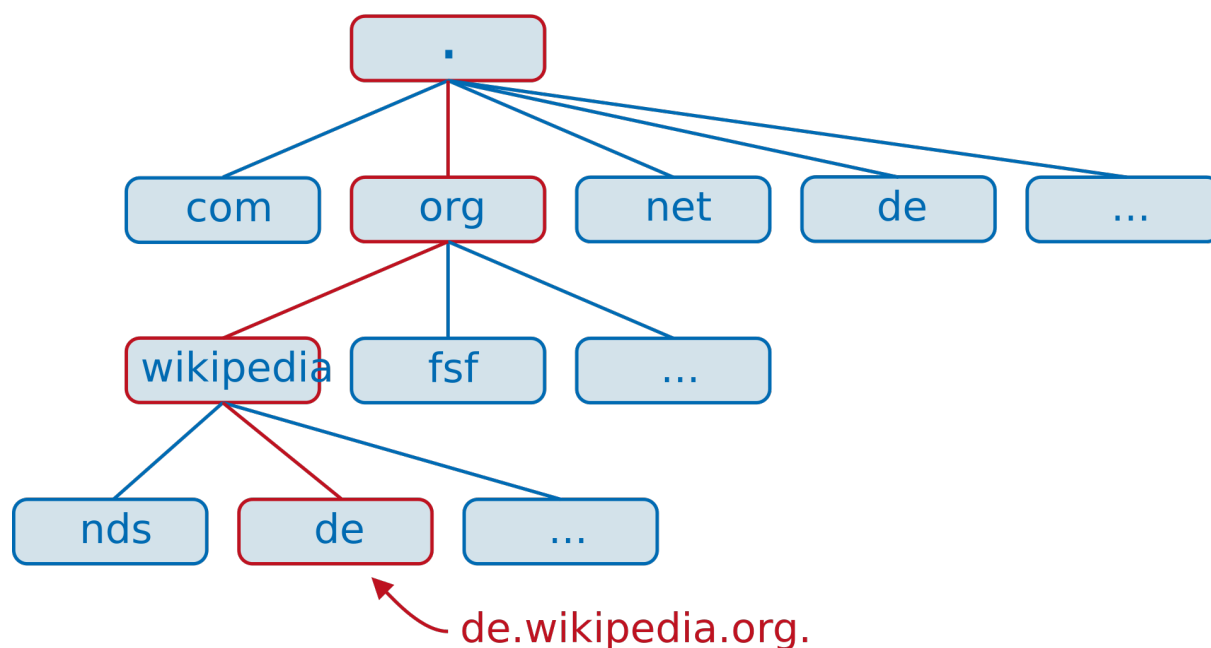


FIGURE 5 – Arborescence de l'espace de nommage.

**Source:** CC BY-SA 2.5, commons.wikimedia.org/w/index.php?curid=556580

## 4 Transporter des données

### 4.1 TCP - Un protocole fiable

TCP est un protocole de communication fiable à flot d'octets orienté connexion.

La connexion entre un client et un serveur s'effectue avec une ouverture en trois temps

(*three-way handshake*). Le point de connexion d'un serveur, nommé *socket*, attend alors une demande de connexion d'un client.

1. Le client, souhaitant se connecter, envoie un segment SYN au serveur qui contient le numéro de séquence à utiliser comme numéro d'initialisation.
2. Le serveur retourne message contenant le message SYN du client, et un accusé de réception ACK.
3. Le client commence à envoyer des données, jusqu'à terminaison de la liaison, qui s'effectue avec un *three-way handshake* contenant le drapeau FIN qui signifie que l'émetteur n'a plus de données à transmettre.

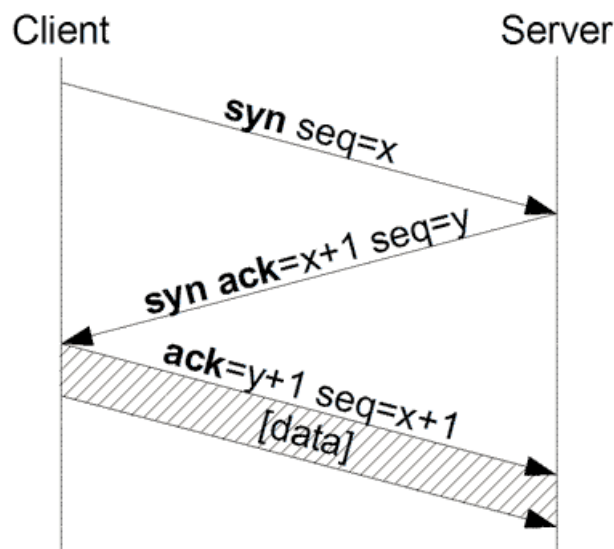


FIGURE 6 – Schéma temporel du Three-way handshake

**Source:** <https://commons.wikimedia.org/wiki/File:Tcp-handshake.png>

Une fois connecté, chaque envoi de paquet doit impérativement être acquitté pour maintenir la connexion et assurer le transfert de l'intégralité des données.

## 4.2 UDP - Un protocole rapide

Quant à UDP, celui est non-fiable : rien dans son en-tête ne permet de réaliser une vérification de l'acquiescement. On dit qu'il est orienté **transaction**. C'est un protocole adapté pour le transfert de données en temps réel : *streaming*, jeux vidéo, etc.

# 5 Requête le serveur : HTTP

## 5.1 Introduction

HTTP est un protocole de la couche Application, au-dessus de TCP, permettant de récupérer des ressources, le plus souvent des fichiers HTML. Il a été conçu pour être lisible par un être humain non-spécialiste de la discipline, et les en-têtes le rend extensible.

C'est un protocole sans état : il n'y a pas de lien entre deux requêtes successives. On pallie au problème en utilisant des **cookies** HTTP, qui permettent au serveur de

# ENTETE TCP

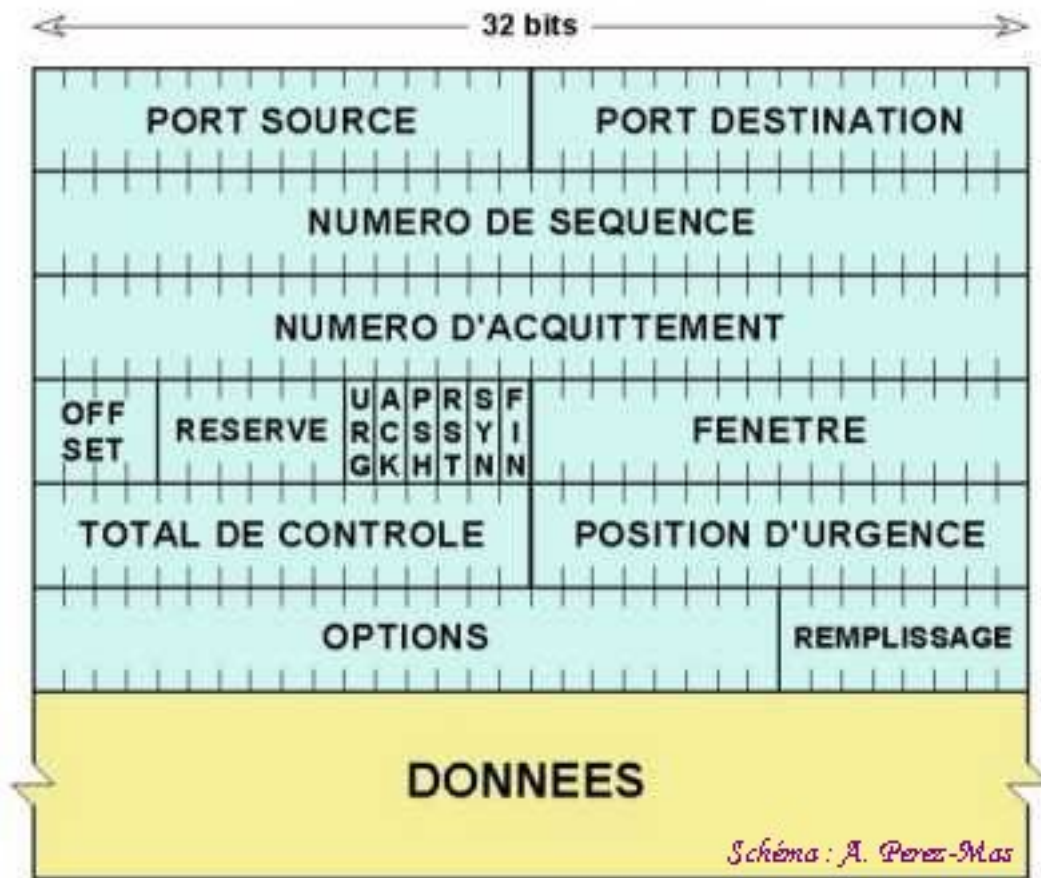


FIGURE 7 – Header TCP

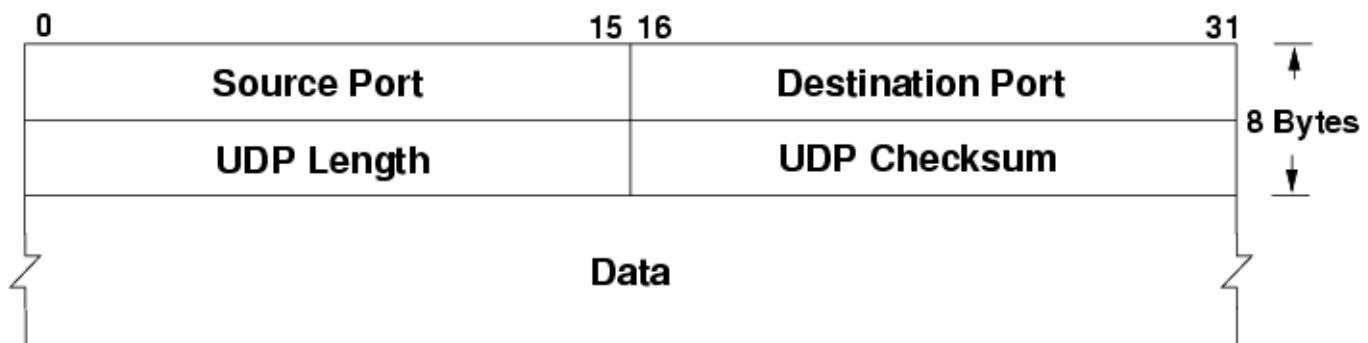


FIGURE 8 – Header UDP

stocker des données dans le navigateur de l'utilisateur, comme par exemple pour garder en mémoire le panier virtuel, ou l'état de connexion à un service, et créer ainsi une session.

L'utilisateur initie la communication avec le serveur en émettant une requête HTTP, ce qui nécessite d'ouvrir une connexion via TCP. On précise que cette connexion prend fin une fois la réponse du serveur traitée, et doit être rouverte pour effectuer une nouvelle requête.

Google a mis en place récemment le protocole QUIC ( *Quick UDP Internet Connec-*

tion) pour accélérer les transactions.

## 5.2 Composition d'une requête

1. La méthode décrit l'action que veut effectuer le client.
  - GET : demande une ressource au serveur.
  - POST : envoie des données à la ressource indiquée, ce qui provoque un changement d'état, ou un effet de bord côté serveur.
  - PUT : remplace la ressource visée par le contenu de la requête.
  - DELETE : supprime la ressource spécifiée.
2. La version du protocole.
3. Des en-tête qui spécifient la nature et l'objet de la requête :
  -

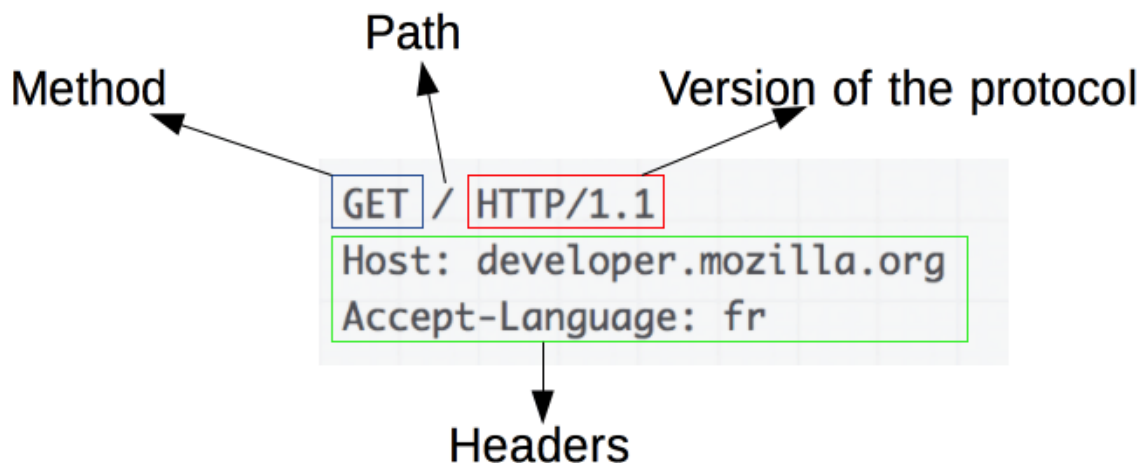


FIGURE 9 – Composition d'une requête GET

**Source:** [developer.mozilla.org/fr/docs/Web/HTTP/Overview](http://developer.mozilla.org/fr/docs/Web/HTTP/Overview)

Cette requête cible une ressource précise, qui est identifiée avec une *Uniform Resource Identifier* (URI), et leur emplacement est déterminé par une *Uniform Resource Locator* (URL).

## 5.3 Syntaxe d'une URI

1. Le protocole utilisé. Les plus courants sont évidemment HTTP, HTTPS et FTP.
2. L'autorité ou le nom de domaine qui gère l'espace de noms. Il est également possible de donner directement l'adresse IP (pour les tests en local, on utilise *localhost*, soit 127.0.0.1).
3. Le port utilisé pour effectuer le transfert. Par défaut, HTTP utilise le port 80 et HTTPS le port 443, et ils peuvent être omis. Dans le cas où un autre port est utilisé, il faut obligatoirement le mentionner.

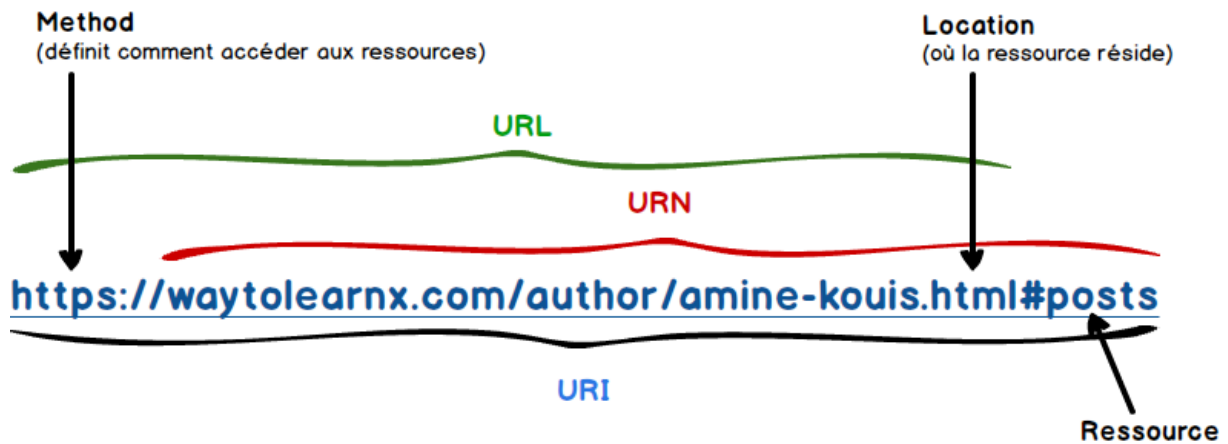


FIGURE 10 – Un exemple d'URL schématisé et légendé.

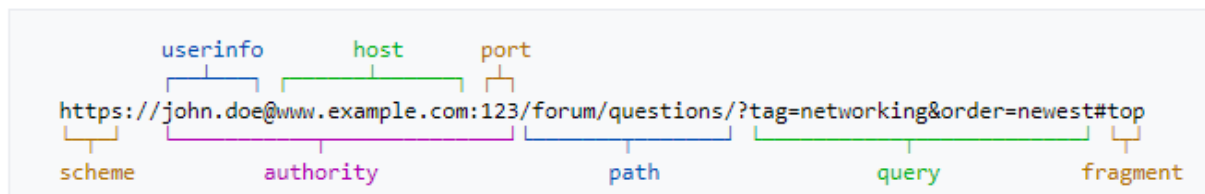


FIGURE 11 – Exemple d'URI légendée.

4. Le chemin de la ressource. Il ne correspond pas obligatoirement au chemin d'accès réel dans le serveur.
5. Des informations de requêtes peuvent être renseignées. Dans ce cas, un ? est inséré et au moins un couple attribut/valeur est inscrit.
6. Une balise ou signet peut être spécifiée, pour que le navigateur se place à un emplacement précis de la page chargée.

## 5.4 Composition d'une réponse

Elle comprend un code de statut, et son message associé. On cite les plus communs :

- 200 : succès de la requête ;
- 301 : redirection permanente ;
- 302 : redirection temporaire ;
- 401 : utilisateur non authentifié ;
- 403 : accès refusé ;
- 404 : ressource non-trouvée ;
- 500 : erreur interne du serveur ;
- 501 : méthode HTTP non-implémentée par le serveur ;
- 503 : service indisponible ou en maintenance.

Lorsqu'une page HTML est reçue, le navigateur l'étudie avec un *parser*, affiche le contenu, et effectue des requêtes pour chaque élément manquant (voir fig. 13). Des éléments légers comme les logos ou fichiers CSS sont mis en cache, du navigateur et du disque



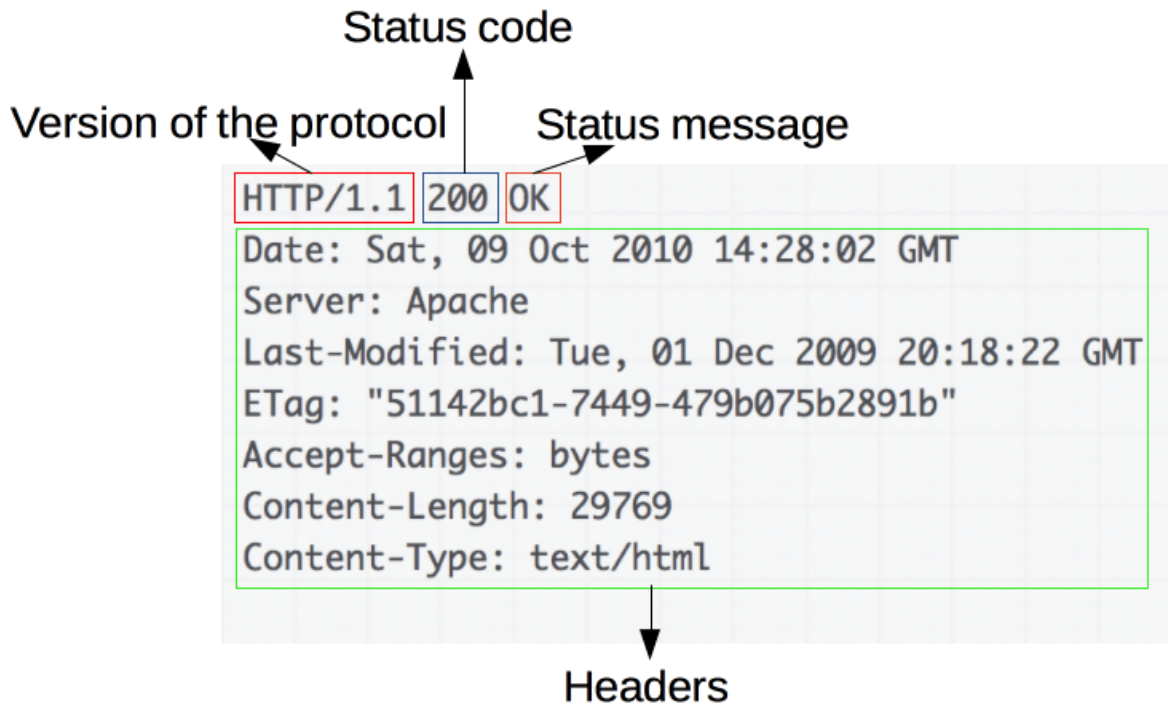


FIGURE 12 – Composition d’une réponse à une requête HTTP

**Source:** [developer.mozilla.org/fr/docs/Web/HTTP/Overview](https://developer.mozilla.org/fr/docs/Web/HTTP/Overview)

dur, ce qui limite le nombre de requêtes à réaliser. Cette pratique est efficace lorsqu’on navigue sur les différentes pages d’un site, qui sont constituées en général avec les mêmes fichiers CSS, et les mêmes éléments graphiques, ou alors lorsqu’on est amené à revenir régulièrement sur les mêmes sites sur une même session, ou alors d’une session à une autre.

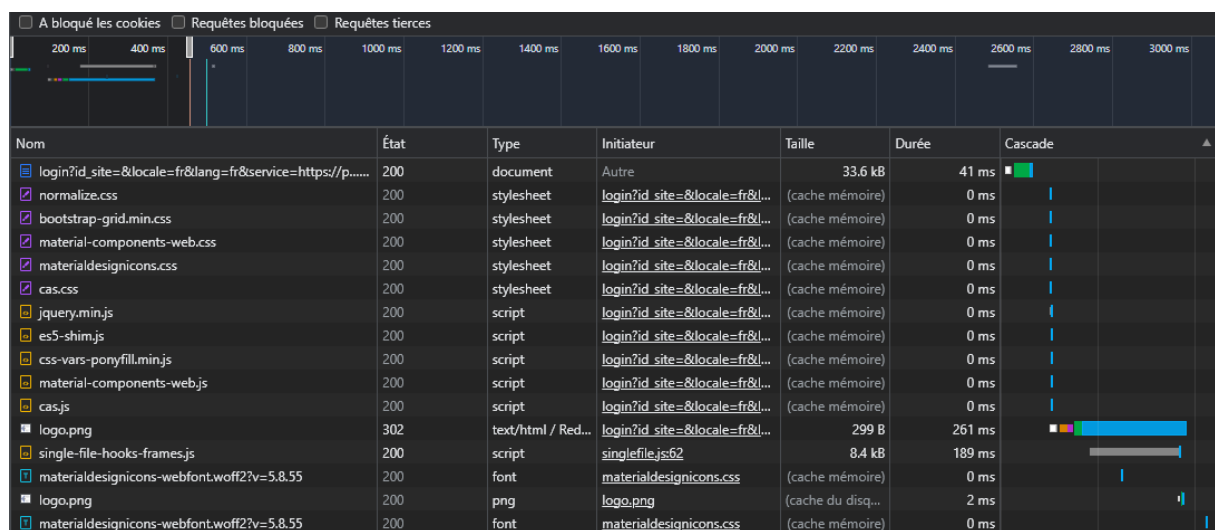


FIGURE 13 – Ensemble des requêtes effectuées dans l’ordre chronologique pour accéder à la page d’authentification de l’Université.

L’en-tête indique également le type de la ressource envoyée, sous le standard *Multipur-*



pose Internet Mail Extensions, abrégé en MIME. Le plus courant est `text/plain` pour les fichiers texte. On note donc que ce n'est pas l'extension du fichier qui est communiquée.

L'IANA<sup>1</sup> constitue le registre officiel pour les types MIME.

## 5.5 Les requêtes HTTP via Javascript

Référence : [developer.mozilla.org](http://developer.mozilla.org)

Nous avons vu précédemment que du côté de l'utilisateur, on pouvait requêter des pages HTML entières. L'inconvénient est qu'il n'est pas possible de récupérer seulement une partie de la page de cette manière : en effectuant une requête sur une page HTML, la page précédente est écrasée par la nouvelle, et tous les fichiers nécessaires sont également requêtés<sup>2</sup>.

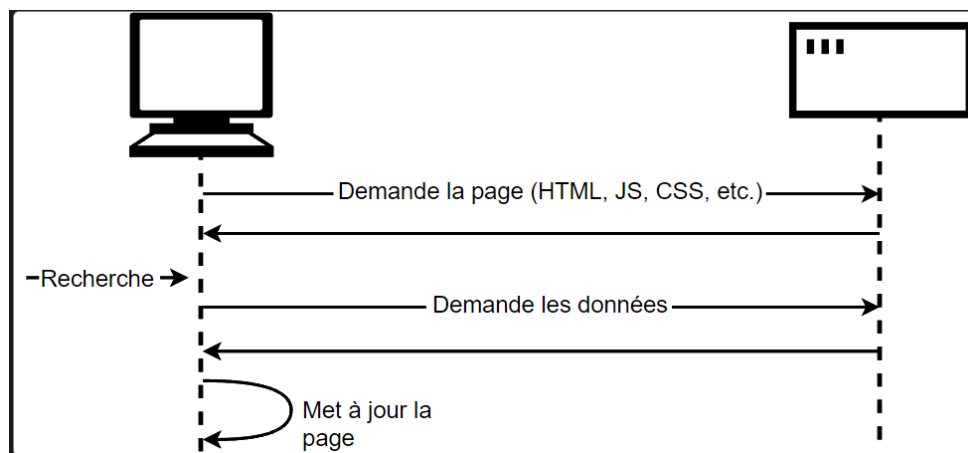


FIGURE 14 – Transaction souhaitée : on met à jour la page sans la redemander.

Javascript dispose de la librairie `fetch` pour réaliser une telle opération. Exemple présenté par Mozilla ci-dessous.

```
1 // On appelle `fetch()` en lui passant l'URL.
2 fetch(url)
3 // fetch() renvoie une promesse. Lorsque nous aurons reçu
4 // une réponse du serveur, le gestionnaire then() de la
5 // promesse sera appelé avec la réponse
6 .then((response) => {
7 // Le gestionnaire lève une erreur si la requête a échoué.
8 if (!response.ok) {
9   throw new Error(`Erreur HTTP : ${response.status}`);
10 }
11 // Sinon, si la requête a réussi, le gestionnaire récupère
12 // la réponse sous forme de texte en appelant response.text(),
13 // Et renvoie immédiatement la promesse renvoyée par response.text().
14 return response.text();
15 })
16 // Quand response.text() a réussi, son gestionnaire `then()` est
17 // appelé avec le texte et nous copions celui-ci dans la boîte
18 // poemDisplay.
19 .then((text) => {
```

1. <https://www.iana.org/assignments/media-types/media-types.xhtml>
2. normalement le navigateur s'occupe de les mettre dans un cache...

```

20     poemDisplay.textContent = text;
21 })
22 // On intercepte les éventuelles erreurs et on affiche un message
23 // dans la boîte `poemDisplay`.
24 .catch((error) => {
25     poemDisplay.textContent = `Erreur lors de la récupération du vers : ${
        error}`;
26 });

```

On réalise donc une requête HTTP, mais le résultat est traité par Javascript et non par le navigateur directement. On utilisera en général du JSON pour effectuer les transactions de données.

## 6 Sécuriser la transmission de données : TLS/SSL

### 6.1 Principes

Initialement *Secure Socket Layer*, puis mis à jour en devenant *Transport Layer Security*, ce protocole a pour objectif de sécuriser les échanges sur le réseau, permettant par exemple de réaliser des transactions bancaires.

TLS/SSL, actuellement en version 3.1, assure les services suivants :

- **confidentialité**, assurée par les algorithmes à **chiffrement symétrique** de blocs comme DES, 3DES ou AES ;
- **intégrité**, assurée par *Message Authentication Code* (MAC) basée sur les **fonctions de hachage** comme MD5 ou SHA-1 ;
- **authentification**, assurée à partir de certificats.

Dans la représentation par couche, TLS/SSL vient au-delà de TCP.

Un échange se déroule en deux phases :

1. Après une requête du client, le serveur envoie son certificat, la signature du certificat et la liste des algorithmes cryptographiques qu'il souhaite négocier.
2. Le client vérifie la validité du certificat avec la clef publique des certificats des autorités de certification ;
3. Si le certificat est valide, le client génère une clef de chiffrement symétrique, ou **clef de session**, qu'il chiffre avec la clef publique du certificat, et la transmet au serveur.
4. Le serveur la déchiffre avec sa clef privée.
5. Le client et le serveur échangent avec la clef de session qui n'est connue que par eux. A ce moment, la connexion TLS est établie.
6. A la fin de la connexion, le serveur révoque la clef de session.

Le serveur peut également demander au client de s'authentifier. Ce dernier envoie alors son certificat en même temps que la clef de session, et le serveur procède alors comme pour le client à la validation du certificat du client.

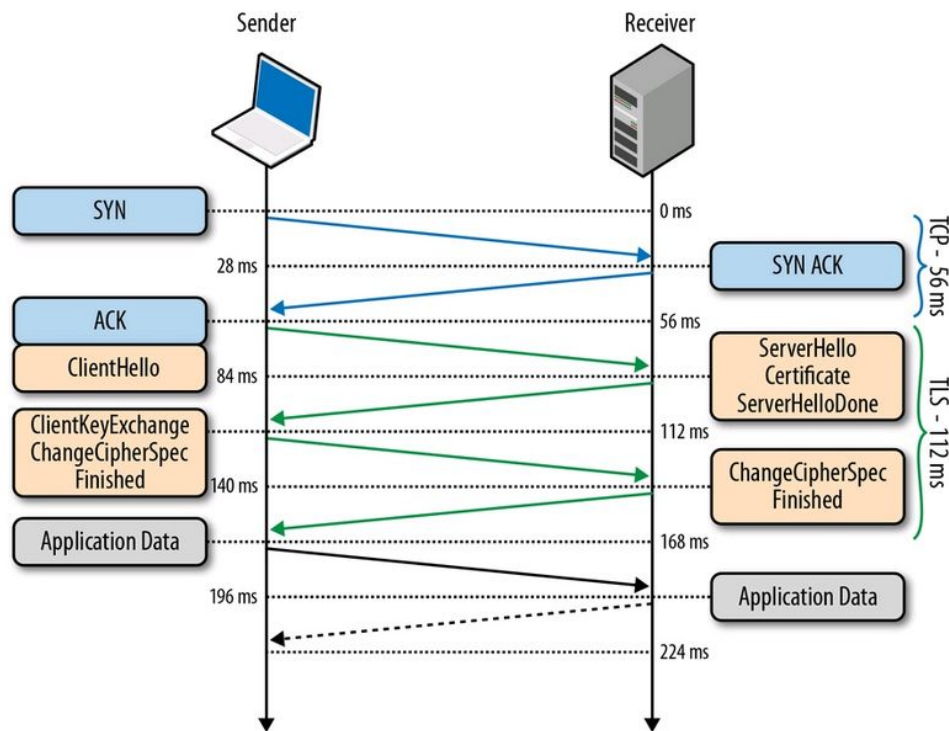


FIGURE 15 – Diagramme d'initialisation d'échange TLS

## 6.2 Cryptographie

**Définition.** La **cryptographie asymétrique** repose sur l'utilisation d'un couple de clefs publique/privée : la clef publique peut être utilisée par tout le monde pour crypter un message, et l'envoyer au détenteur de la clef privée, qui est le seul moyen de déchiffrer le message.

Ainsi, pour un serveur HTTP, il faudra donc générer un couple de clefs pour pouvoir initialiser une connexion sécurisée avec ses clients. Pour TLS v1.3, l'algorithme Diffie-Hellman est employé. Le client encode sa requête avec la clef publique, et le serveur est le seul à pouvoir déchiffrer son contenu. Une clef de chiffrement symétrique (commune aux deux interlocuteurs) est alors convenue, puis utilisée.

**Définition.** Une **fonction à sens unique** est une fonction dont on calcule facilement l'image d'une valeur, mais difficilement l'antécédent d'une valeur.

**Définition.** Une **fonction de hachage** est une fonction qui associe des données de taille arbitraire à une valeur de taille fixe. Le résultat est appelé *hash*, *valeur de hachage*, *signature*, *condensat*.

## 6.3 Authentification forte

Si TLS/SSL permet d'établir une connexion sécurisée entre un client et un serveur, elle ne garantit pas immédiatement que le client est bien celui qu'il prétend être. La méthode décrite précédemment, à savoir la génération par le client d'un couple clef privée/clef publique est efficace, mais n'est pas répandue : les certificats sont stockés dans le navigateur, et les utilisateurs possèdent aujourd'hui plusieurs appareils. De plus, la génération des-dits certificats est loin d'être aisée pour un utilisateur lambda.

On lui préfère une authentification à deux facteurs (2FA), consistant à présenter deux preuves d'identité distinctes au mécanisme d'authentification. En général, on utilise le système classique nom d'utilisateur et mot de passe, auquel on ajoute une identification physique (clef, puce, biométrie), ou un mot de passe à usage unique, en lien avec un appareil unique qu'une bonne partie des utilisateurs peut posséder : une adresse mail vérifiée, ou encore mieux, un téléphone portable, recevant un SMS ou se connectant à une application tierce.

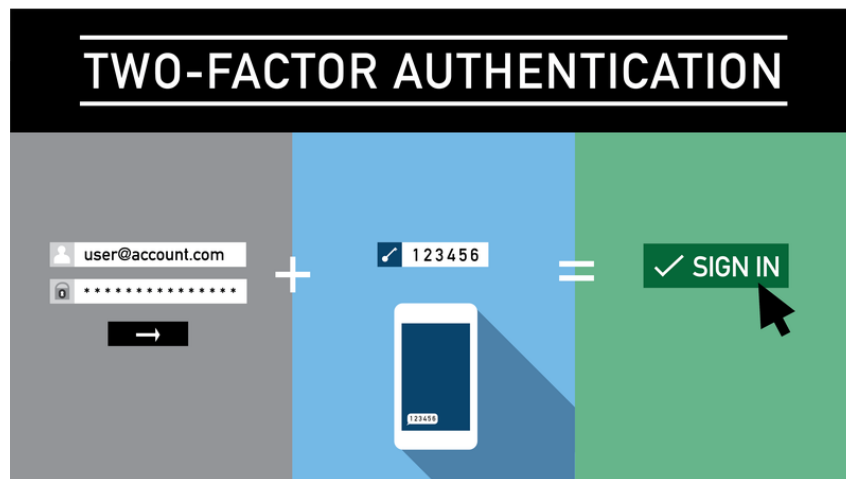


FIGURE 16 – Illustration d'une authentification à 2 facteurs avec un téléphone.

Source: [mshelton.medium.com/two-factor-authentication-for-beginners-b29b0eec07d7](https://mshelton.medium.com/two-factor-authentication-for-beginners-b29b0eec07d7)

## 6.4 Authentification faible

On décrit ci-dessous les étapes d'une authentification classique avec cookie :

1. L'utilisateur écrit un nom d'utilisateur et un mot de passe, et l'envoie par l'intermédiaire du navigateur au serveur.
2. Le serveur vérifie dans la base de données la validité des identifiants. Dans le cas positif, il génère une session à partir des identifiants, et un identifiant de session unique, qu'il renvoie au client.
3. Le client reçoit l'ID de session, et le sauvegarde dans un cookie. **A chaque fois** qu'il demande l'accès à une ressource protégée, il communique l'ID de session dans la requête. Cela évite donc d'avoir à réécrire les identifiants à chaque changement de page.
4. Le client met fin à la session en indiquant au serveur qu'il se déconnecte. Le serveur révoque alors l'ID de session et la détruit. Le client doit alors communiquer à nouveau ses identifiants et être validé par le serveur pour accéder à nouveau au contenu protégé.

## 7 Espace de stockage

Référence : [developer.mozilla.org](https://developer.mozilla.org)

Il est possible de stocker des données dans le navigateur. Javascript propose une API pour réaliser ces opérations, avec deux espaces différents :

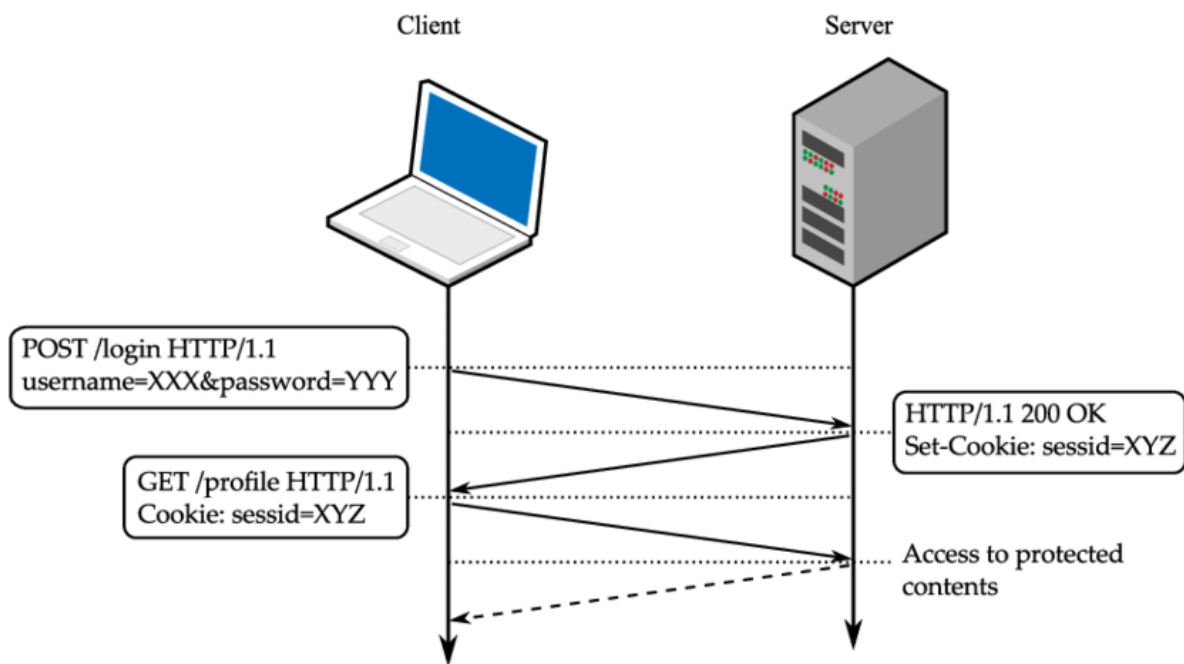


FIGURE 17 – Schéma d’une authentification avec cookie

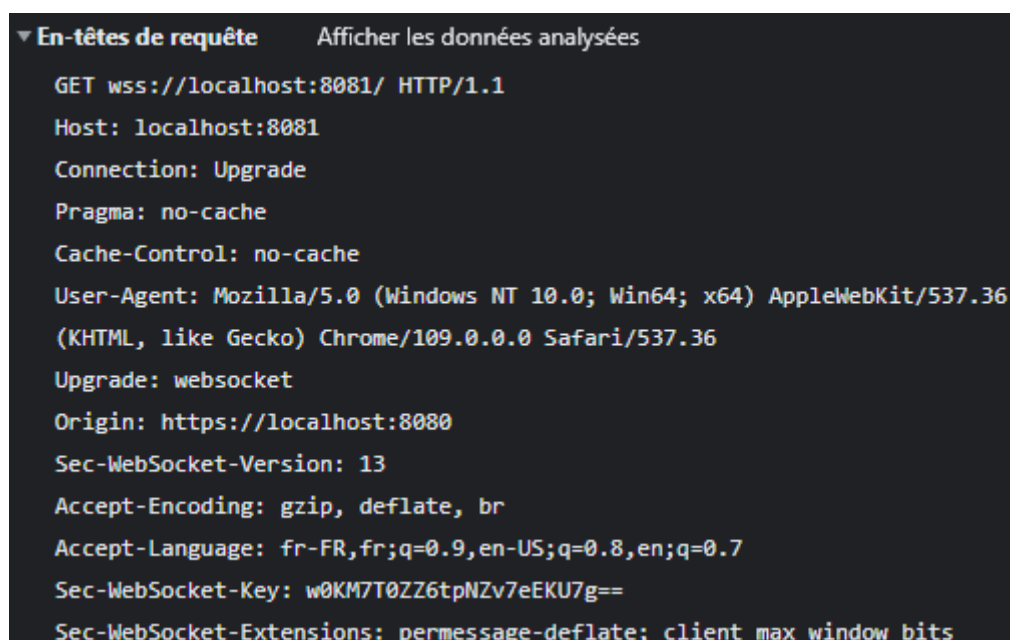
**Source:** Surviving the Web : A Journey into Web Session Security - Scientific Figure on ResearchGate. Available from : [https://www.researchgate.net/figure/Cookie-based-User-Authentication\\_fig1\\_314289445](https://www.researchgate.net/figure/Cookie-based-User-Authentication_fig1_314289445) [accessed 4 Feb, 2023]

- **sessionStorage**, qui est valable le temps d’une **session** : jusqu’à ce que le navigateur soit fermé
- **localStorage**, qui comme son nom l’indique, stocke localement les données sur l’ordinateur, ce qui rend les données persistantes même après fermeture.

## 8 Communication bidirectionnelle avec WebSocket

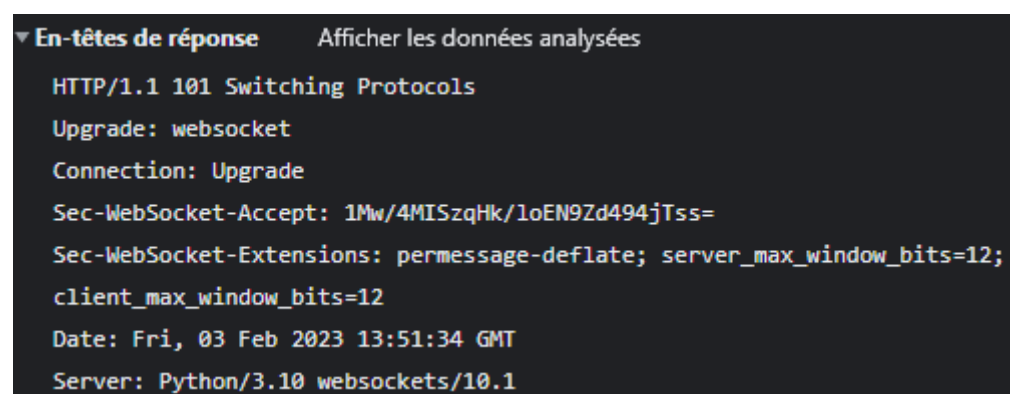
Le WebSocket est un standard récent défini par la RFC 6455 qui consiste à créer des canaux de communication bidirectionnels par-dessus une connexion TCP entre un navigateur web et un serveur web. Le protocole a été normalisé en 2011, donc relativement récent.

La connexion démarre en envoyant une requête HTTP au serveur Websocket, en spécifiant qu'on souhaite *améliorer* cette connexion.



```
▼ En-têtes de requête    Afficher les données analysées
GET wss://localhost:8081/ HTTP/1.1
Host: localhost:8081
Connection: Upgrade
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36
Upgrade: websocket
Origin: https://localhost:8080
Sec-WebSocket-Version: 13
Accept-Encoding: gzip, deflate, br
Accept-Language: fr-FR, fr;q=0.9, en-US;q=0.8, en;q=0.7
Sec-WebSocket-Key: w0KM7T0ZZ6tpNZv7eEKU7g==
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits
```

FIGURE 18 – *Header* de la requête d'*upgrade* du client



```
▼ En-têtes de réponse    Afficher les données analysées
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: 1Mw/4MISzqHk/1oEN9Zd494jTss=
Sec-WebSocket-Extensions: permessage-deflate; server_max_window_bits=12;
client_max_window_bits=12
Date: Fri, 03 Feb 2023 13:51:34 GMT
Server: Python/3.10 websockets/10.1
```

FIGURE 19 – *Header* de la réponse du serveur

Une telle connexion permet au serveur d'envoyer des informations au client de sa propre initiative, sans effectuer de requête. Ainsi, tout type d'application nécessitant une transmission en temps réel, comme un jeu en ligne, une discussion instantanée, une application de diffusion de résultats en direct (sports, bourse, etc.) bénéficiera grandement d'une ouverture de WebSocket.

Le socket peut générer 4 événements, que l'application doit écouter :

- `open`, qui signifie que la connexion est établie ;



- `message`, qui signale la réception de données ;
- `error`, qui est explicite ;
- `close`, qui signale que la connexion WebSocket est désormais fermée.

Son utilisation est similaire aux sockets Berkeley et Unix, elle sera détaillée dans la mise en application.

## 9 Exercices d'analyse

### Exercice 1 — Analyse de fonctionnement

Nom	État	Type	Initiateur	Taille
<input checked="" type="checkbox"/> <code>common.css</code>	304	stylesheet	Autre	95 B
<input checked="" type="checkbox"/> <code>index.css</code>	304	stylesheet	Autre	95 B
<input checked="" type="checkbox"/> <code>infor.css</code>	304	stylesheet	Autre	95 B
<input type="checkbox"/> <code>collect?v=2&amp;tid=G-53G9RWSYBH&amp;gtm=45je33f0&amp;p=96926...2F%...</code>	204	ping	<a href="#">js?id=G-53G9RWSYBH:370</a>	17 B
<input type="checkbox"/> <code>swift.php?langue=mot_corse&amp;param=FRANCESE%20TALIAN...DAAU...</code>	200	xhr	<a href="#">common.js:28</a>	17.3 kB
<input type="checkbox"/> <code>swift.php?langue=mot_corse&amp;param=FRANCESE%20TALIAN...AAUT...</code>	200	xhr	<a href="#">common.js:28</a>	17.3 kB
<input type="checkbox"/> <code>swift.php?langue=mot_corse&amp;param=FRANCESE%20TALIAN...AUTO...</code>	200	xhr	<a href="#">common.js:28</a>	24.0 kB
<input type="checkbox"/> <code>swift.php?langue=mot_corse&amp;param=FRANCESE%20TALIAN...UTOR...</code>	200	xhr	<a href="#">common.js:28</a>	15.1 kB
<input type="checkbox"/> <code>swift.php?langue=mot_corse&amp;param=FRANCESE%20TALIAN...TORJ...</code>	200	xhr	<a href="#">common.js:28</a>	13.8 kB
<input type="checkbox"/> <code>swift.php?langue=mot_corse&amp;param=FRANCESE%20TALIAN...ORI%...</code>	200	xhr	<a href="#">common.js:28</a>	13.5 kB

Parolla à traduce  
manghj

**manghjà magnà manghià**  
manger; empiéter

**manghjacristu magnacristu manghiacristu manghja cristu**  
bigot à la langue bien pendue

Note : `xhr` pour `XMLHttpRequest`

Sur la page [www.adecec.net/infor/index.php](http://www.adecec.net/infor/index.php), il est possible d'effectuer des traductions entre français et corse. Un utilisateur écrit ici 6 lettres, et une liste de mots et définitions correspondantes apparaît. Il est également possible de cliquer dessus (cela n'effectue **aucune requête**) pour avoir plus de détails (traduction en anglais et italien, ethymologie, synonymes, etc.).

1. D'après vos connaissances, les captures d'écran, et le texte descriptif, que se passe-t-il si on tape un caractère ? Que reçoit-on comme résultat de chaque requête ?
2. Donner l'ordre de grandeur de la quantité d'information transmise pour obtenir la traduction de ce mot, et critiquer ce résultat.
3. Après une étude<sup>3</sup> des statistiques utilisateurs, 2% des requêtes amènent à une consultation du détail d'un mot. En prenant en compte ce résultat, et les réponses aux questions précédentes, proposer une amélioration de cette page pour qu'il y

3. Totalement inventée pour cet exercice.

ait moins de données transmises pour une recherche. *On ne demande pas ici un travail exhaustif mais uniquement conceptuel.*

## Exercice 2

Nous allons essayer d'analyser le fonctionnement du site de l'université à l'aide de l'inspecteur du navigateur. On répondra aux questions sous la forme d'un rapport, en regroupant dans un ou plusieurs paragraphes les éléments de réponse.

1. Ouvrir l'inspecteur puis aller sur la page d'authentification de l'Université. Si vous êtes déjà connecté, déconnectez-vous.
  1. Quelle est l'adresse IP du serveur de l'Université ?
  2. Y a-t-il une ou plusieurs requêtes pour charger tous les éléments de la page ? (CSS, JS, images, etc.)
  3. Essayer de se connecter avec des identifiants incorrects. Comment sont transmis les identifiants au serveur ? Comment le serveur nous répond-il ? Quel est le code de retour ?
  4. Tous les éléments précédemment reçus sont-ils de nouveau téléchargés ?
  5. Se connecter avec les bons identifiants. Fermer l'onglet, puis retourner sur la page d'authentification. Que se passe-t-il ?
  6. Fermer et relancer le navigateur, puis retourner sur la page d'authentification. Que se passe-t-il ? Pourquoi ?
  7. Se déconnecter à nouveau, ouvrir sur deux onglets différents la page d'authentification. S'authentifier sur un onglet, puis actualiser l'autre onglet. Que se passe-t-il ? Pourquoi ?
  8. Y a-t-il des cookies suspects dont le nom commence par `_ga` ? Rechercher leur provenance et présenter l'intérêt de leur utilisation.
  9. Quel est le domaine de la page de connexion ? Celui de la page contenant les outils de l'ENT ?
10. Consulter le site de CAS<sup>4</sup>. Rédiger un paragraphe expliquant le fonctionnement de l'authentification aux services de l'université, ainsi qu'à des services tiers, comme celui d'Outlook. On mettra en valeur les cookies utilisés (en particulier les cookies TGC, et celui commençant par ST).

*En ouvrant l'inspecteur sur la page d'authentification, cocher la case "Conserver le journal". Cela permettra de consulter les requêtes réalisées sur le domaine `auth.univ-corse.fr`.*

## Exercice 3 — Projet - Compteur interactif

Pour appliquer l'ensemble des informations présentes dans ce cours, nous allons réaliser un compteur interactif : un serveur distribue une page sur laquelle est affiché un compteur, qu'on peut augmenter ou diminuer en appuyant sur des boutons. La valeur du compteur est stockée dans le serveur.

Pour faciliter la création du serveur, on utilisera la librairie FastAPI pour Python. Celle-ci nous permettra de donner une structure fixe aux JSON qui seront reçus et transmis.

### Création du serveur

---

4. [https://www.esup-portail.org/consortium/espace/SSO\\_1B/cas/](https://www.esup-portail.org/consortium/espace/SSO_1B/cas/)

1. Installer la librairie, et créer un répertoire dans lequel on crée un premier fichier python `main.py` qui fera tourner le serveur.
  2. Créer un dossier `static` dans lequel on placera tous les fichiers HTML, CSS, JS et autres à distribuer au client.
  3. Créer la page HTML contenant le compteur, et deux boutons `+` et `-`. Chaque bouton provoque l'exécution d'une requête AJAX vers le serveur, qui augmente ou diminue un compteur.
  4. Ouvrir deux navigateurs différents, et réaliser des requêtes sur l'un d'entre eux. Pourquoi l'autre navigateur ne voit pas l'actualisation du compteur ?
5. Ajouter une méthode qui s'appelle toutes les 5 secondes et qui envoie une requête demandant la valeur du compteur.

**Une solution possible : le *polling***

Le *polling* consiste à envoyer périodiquement des requêtes.

**Une solution en temps réel : les *Websockets***

Les Websockets permettent une communication bi-directionnelle entre le client et le serveur. Ce dernier peut alors envoyer un message au client sans attendre une requête.