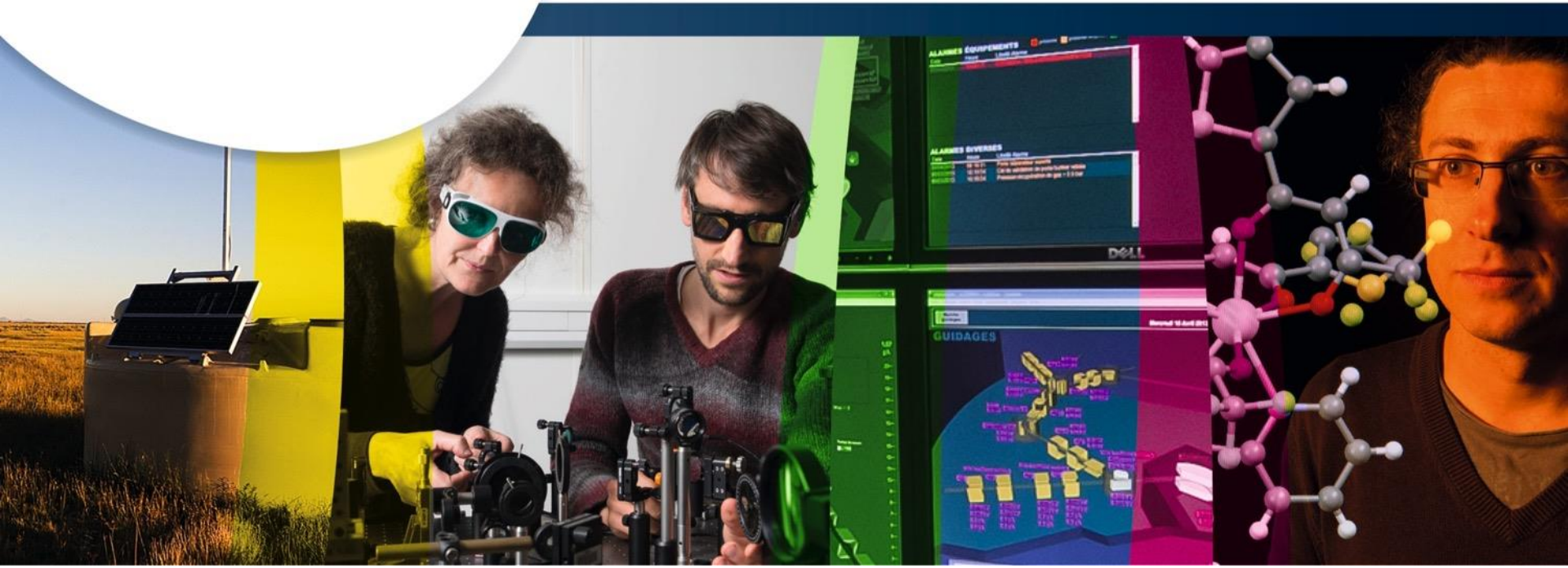
Licence CC BY-NC-SA 3.0 FR

www.cnrs.fr

HTTP et HTTPS



www.cnrs.fr

HTTP(s)

Ce célèbre inconnu

Pourquoi parler de HTTP

www.cnrs.fr

- Pourquoi vous devez connaître HTTP?
  - Propre d'une application web
- Mauvaise compréhension ➔ mauvais design ➔ vulnérabilités
- Nécessaire à la compréhension du top 10 OWASP
- Au cœur des architectures actuelles
  - REST, API, Microservices, [.*]As A Service
- Survoler ce vaste sujet

Histoire courte d'une longue histoire

- **H**yper**t**ext **T**ransfer **P**rotocol

- Histoire longue en 2 secondes
  - **1989** : Création par Tim Berners-Lee au CERN,
  - **1996** : HTTP/1.0, décrit dans la RFC 1945
  - **1997** : HTTP/1.1, normalisé par l'IETF RFC 2616
    - **2014 :** éclatement plusieurs RFC
  - **2012**: groupe de travail sur HTTP/2.0

- Environ 1 milliard de sites web "recensés" en 2016 (NetCraft)

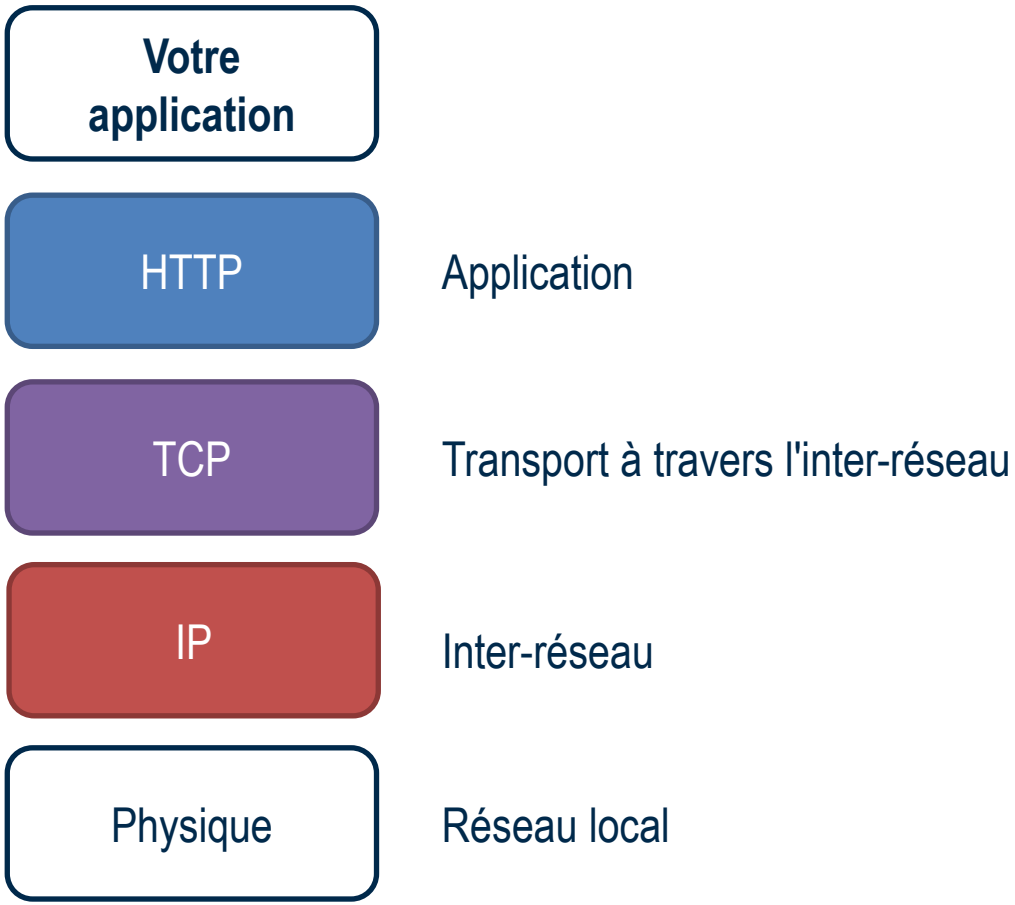
- La version courante est **HTTP/1.1**

www.cnrs.fr

Les couches basses

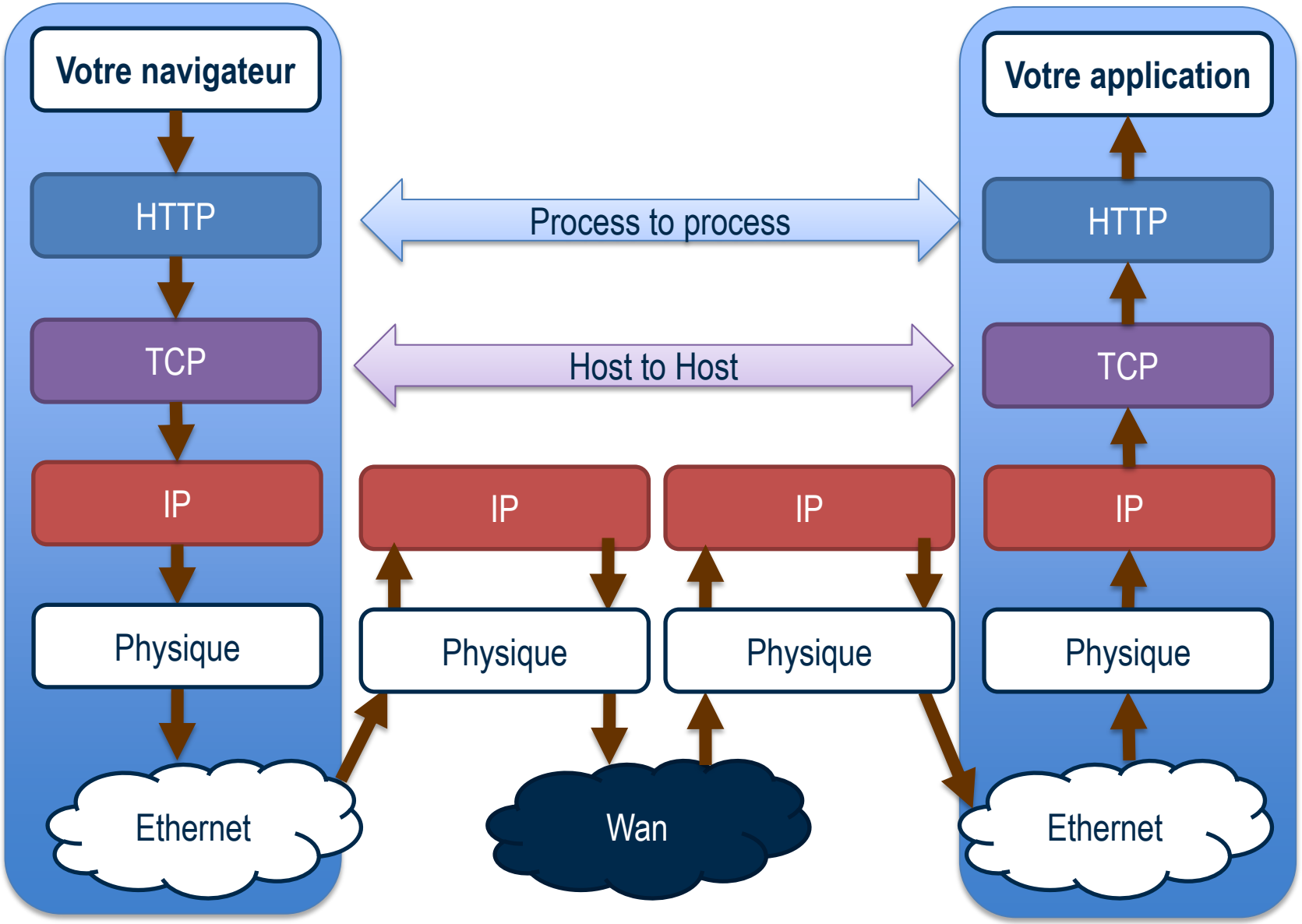
Pile(s) protocolaire(s)

La pile protocolaire

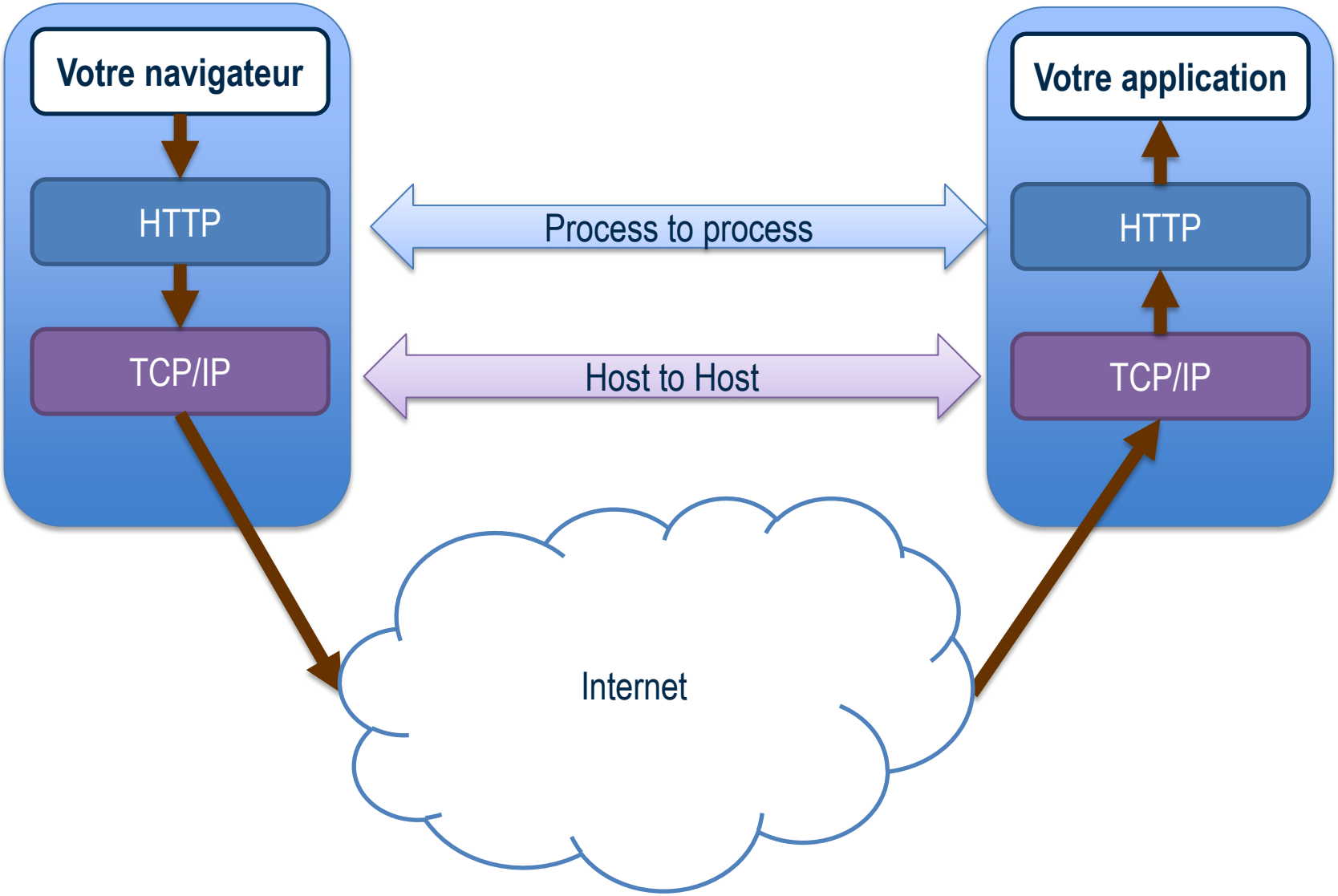


- La correspondance TCP/IP et OSI est sujette à caution

Flux d'informations

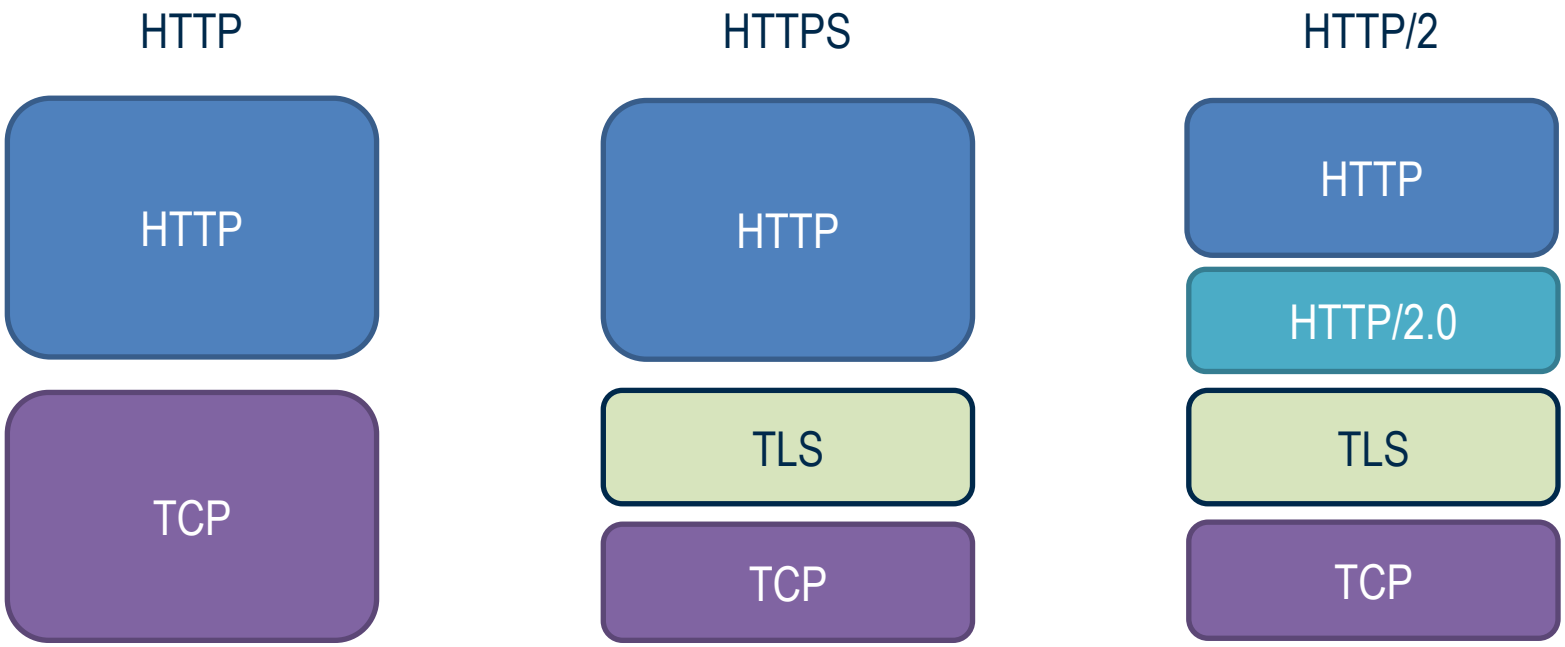


Flux d'informations

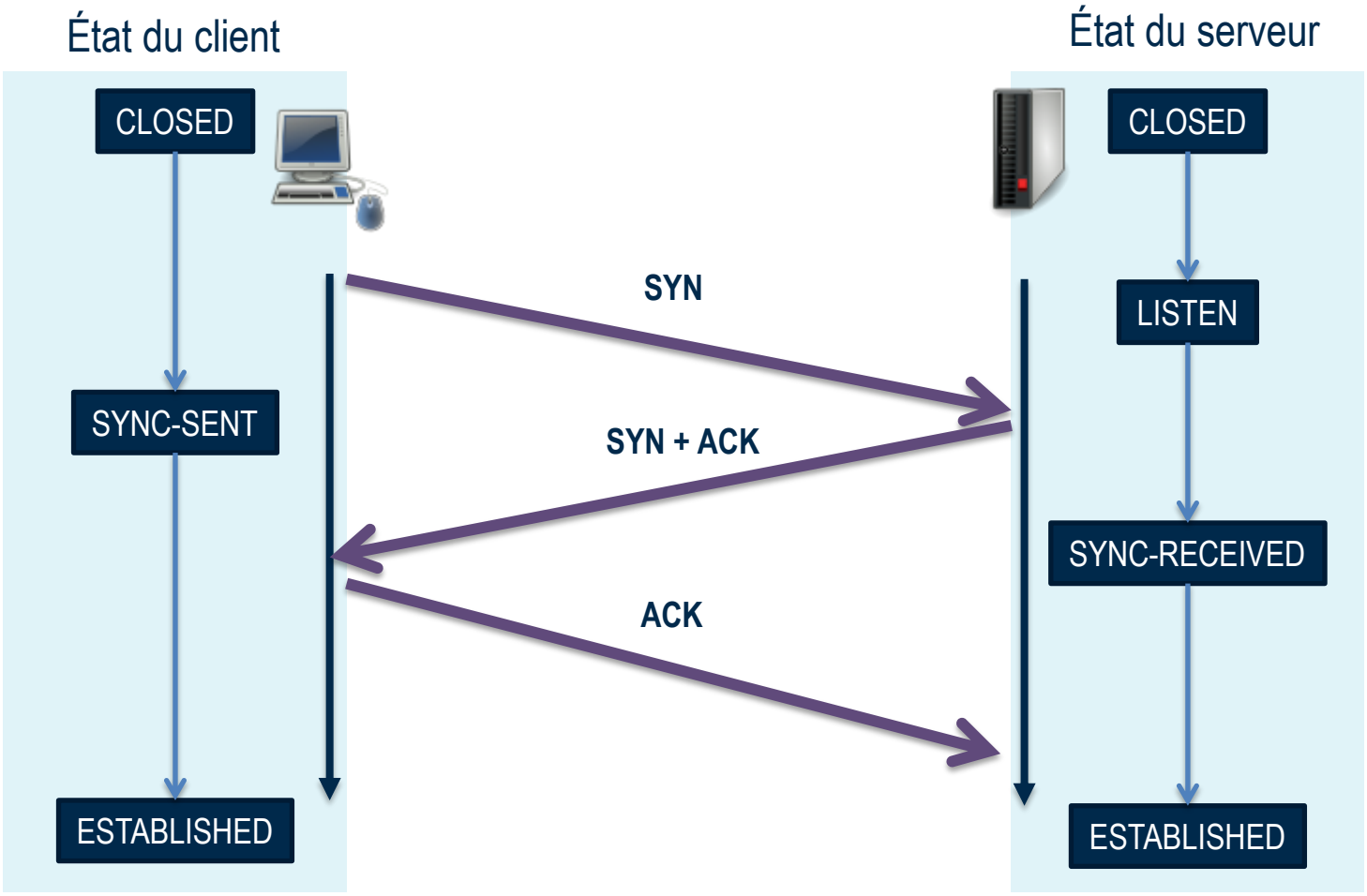


HTTP, HTTPS et HTTP2

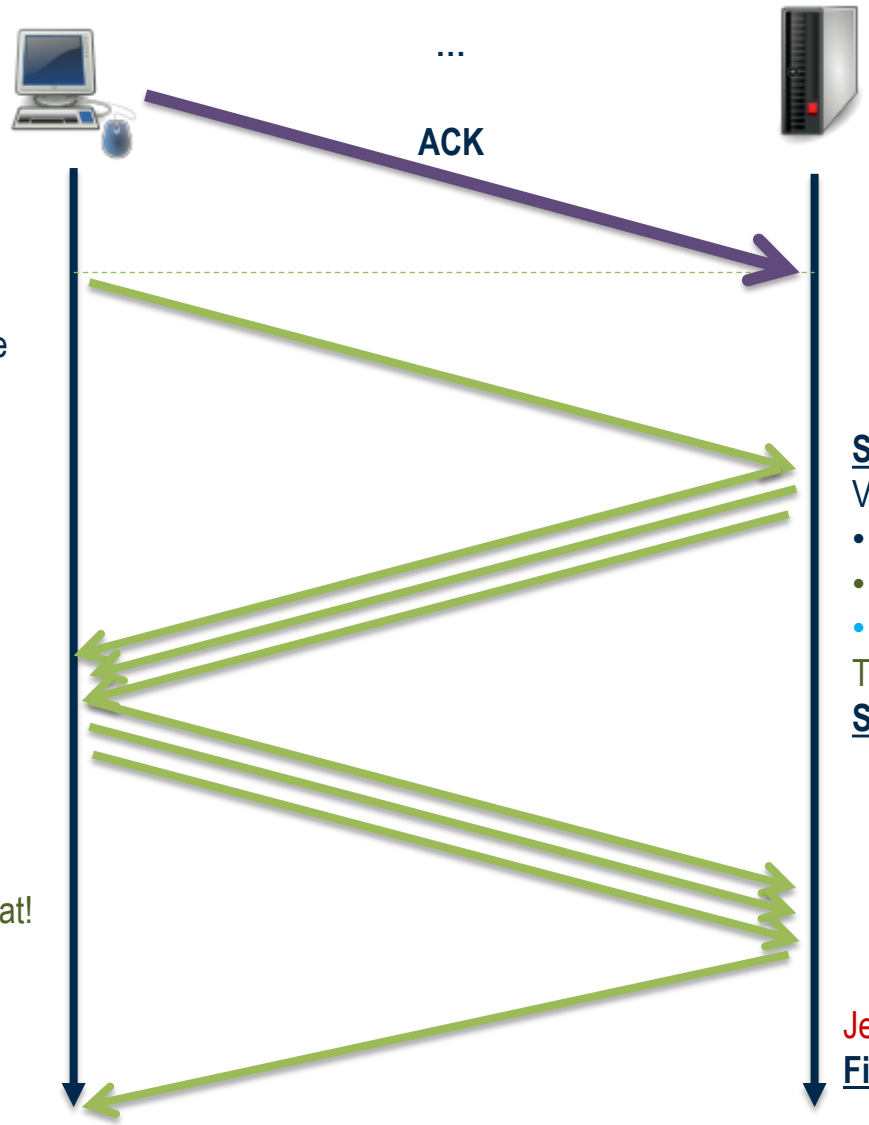
- TLS: Transport Layer Security
- HTTP/2 n'est pas le replacement de HTTP/1 !



Le Three-Way Handshake TCP



Le Handshake TLS



ClientHello
Voici
- ma version de TLS maximale
- les algorithmes supportés

ServerHello
Voici
- Ma sélection dans tes choix
- Mon certificat
- Une clef qui te sera utile

T'as un certificat ?
ServerHelloDone

Voici
- mon certificat
- Un mix de clefs
- De quoi vérifier mon certificat!

Je bascule en mode chiffré
Fin

Je bascule en mode chiffré
Fin

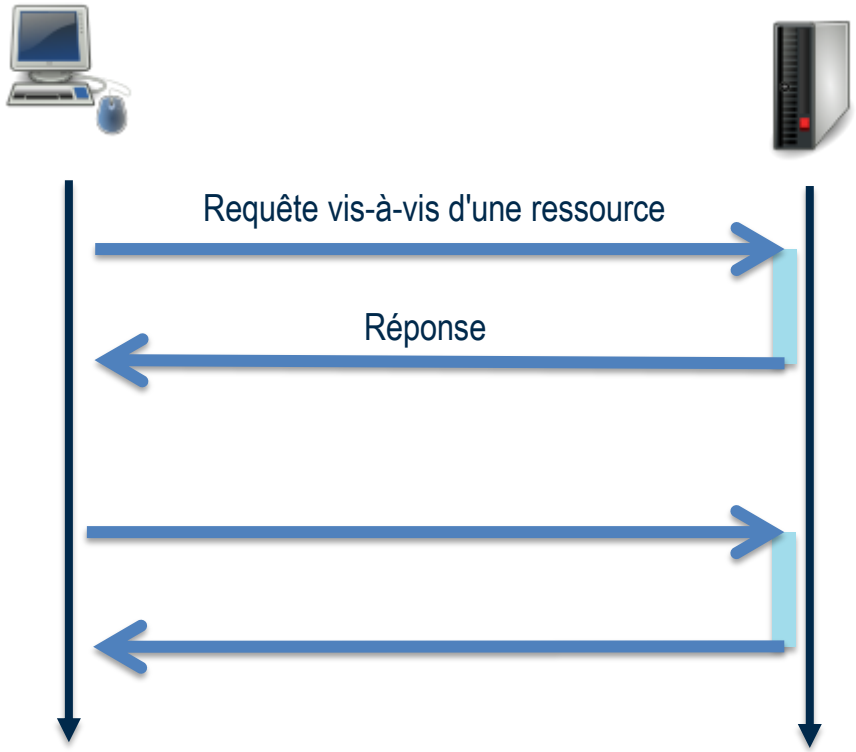
www.cnrs.fr

Protocole

Le P de HTTP

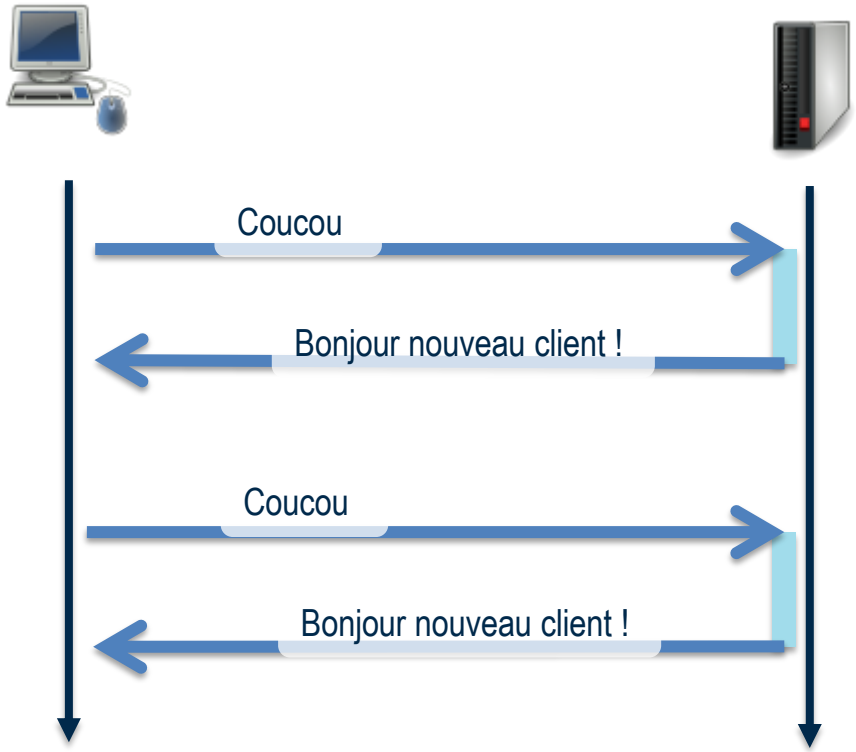
Requête, Réponse, Ressource

- Suite de **requête** / **réponse** vis-à-vis d'une **ressource**
- Texte intelligible et *human-friendly*



Stateless

- L'échange est sans état
- le serveur HTTP ne retient rien du client entre deux échanges



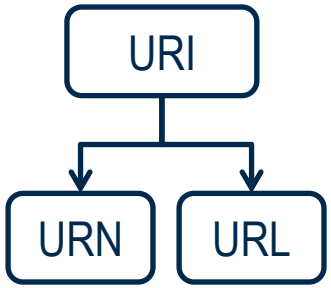
Ressource, URL, URI

www.cnrs.fr

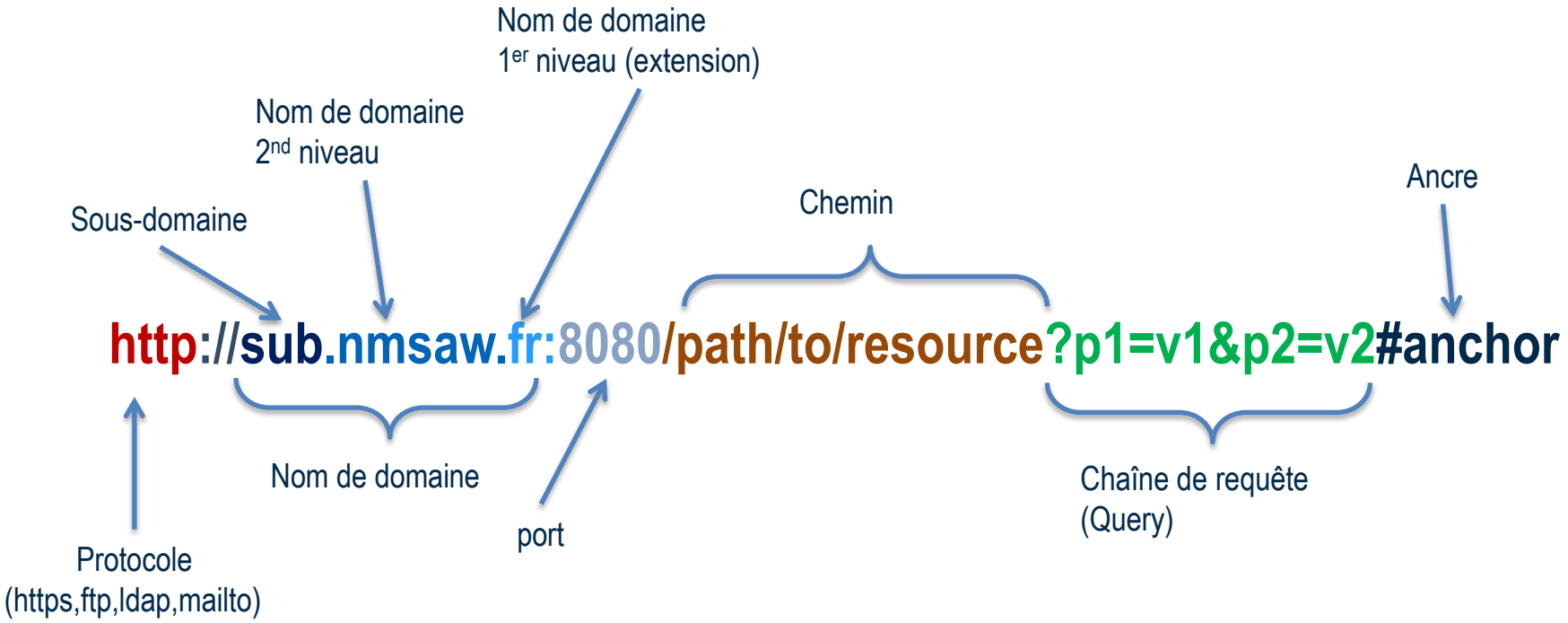
- ***Resource***
  - Un contenu, une entité
  - Représentée selon différents types ***(Multipurpose Internet Mail Extensions)***
    - Ex: même document au format text, json, xml, pdf, html… ➔ content/type
    - Dans une optique REST, on parle d'Etat (State) de la ressource

- La ressource est identifiée par une **URL : Uniform Resource Locator**

- L'URL est une **URI : Uniform Resource Identifier**
  - Mais une URI n'est pas forcément une URL…
  - Il existe même des URN (**name**), mais c'est un autre débat
    - http://www.bortzmeyer.org/3986.html

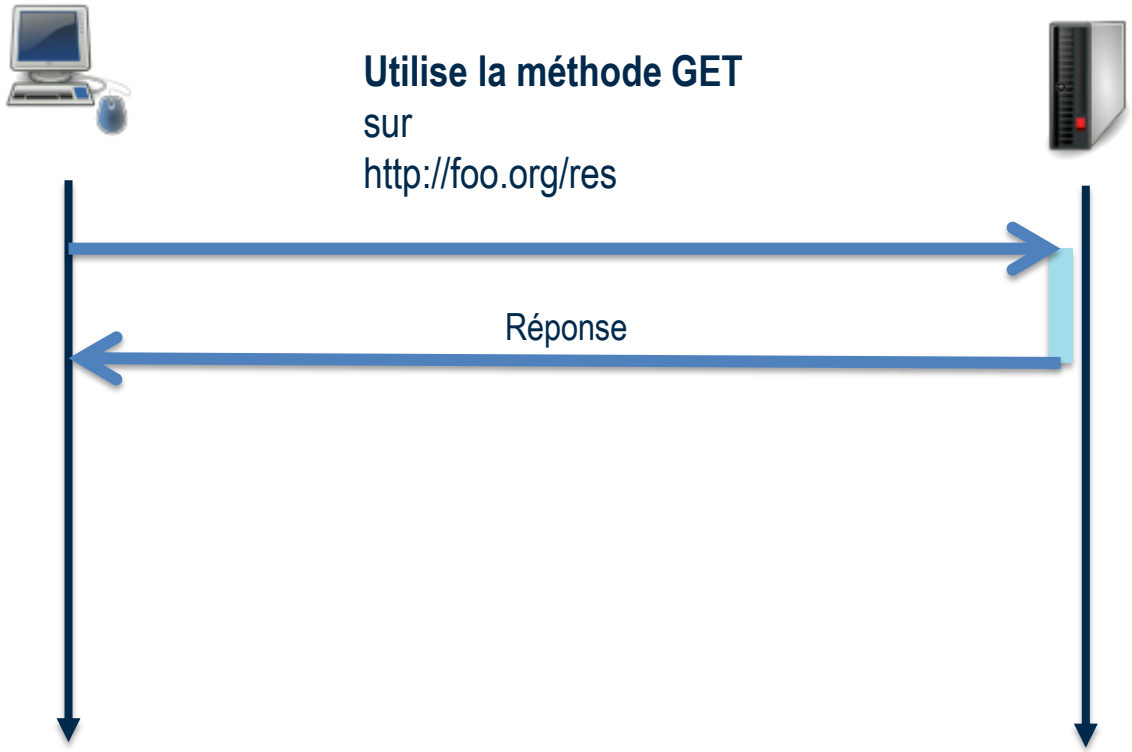


Uniform Resource Locator



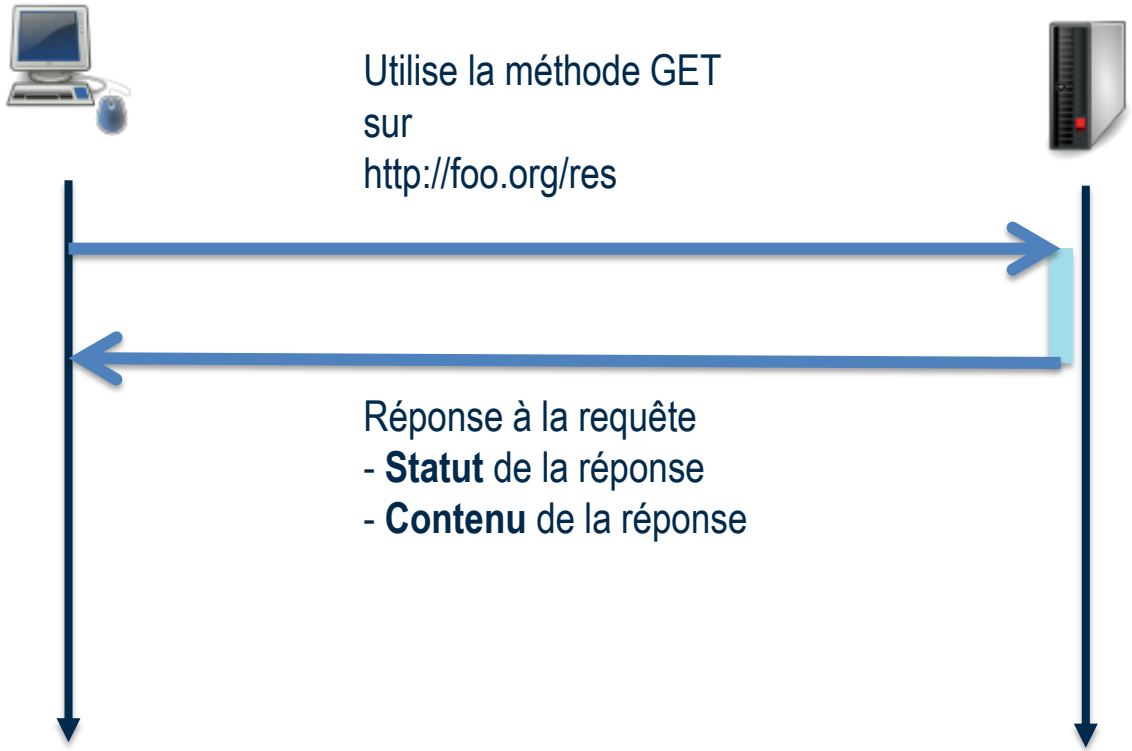
Requête, Méthode , Ressource

- Une requête s'exprime sous la forme d'une **méthode** à employer sur une *ressource*



Réponse, Statut, Contenu

- Le serveur répond à la requête avec un **statut** et un **contenu**



Anatomie d'une requête

- Patron de la requête

```
Ligne de commande (méthode, URL, Version de protocole )
En-têtes
<ligne vide>
Corps de la requête
```

- Exemple : obtenir la ressource index.html

```
GET /index.html HTTP/1.1
Host: nmsaw.fr
<ligne vide>
```

- Exemple: envoyer des informations

```
POST /inscription.php HTTP/1.1
Host: nmsaw.fr
<ligne vide>
prenom=jean&nom=gabin
```

Les Méthodes

Petit traité de manipulation des ressources à l'usage des honnêtes gens

- **GET** : pour obtenir une *représentation* fournie par la ressource
- **POST** : pour que la ressource procède à un *traitement* de la *représentation* envoyée

- **HEAD**: pour obtenir les méta-informations de GET, mais *sans la représentation*
- **PUT**: *modifier* ou *créer* une ressource avec la représentation envoyée
- **DELETE** : supprimer la ressource

- **CONNECT** : ouverture d'un tunnel pour une communication TLS de bout-en-bout
- **TRACE** : renvoyer un écho de la représentation (débogage)
- **OPTIONS** : obtenir des informations sur les options d'une ressource ou les capacités d'un serveur

Sureté et idempotence

- Méthode sure
  - L'invocation ne modifie pas la ressource sur le serveur: lecture seule

- Méthode idempotente
  - Toute invocation répétée donne un résultat identique

Méthode	Sure	Idempotente
GET	**Oui**	**Oui**
HEAD	**Oui**	**Oui**
POST	**NON**	**NON**
PUT	**NON**	**Oui**
DELETE	**NON**	**Oui**
OPTIONS	**Oui**	-
CONNECT	-	-
TRACE	**Oui**	

Anatomie d'une réponse

- Patron de la réponse

```
Ligne de statut(Version de protocole, Statut, Texte )
En-têtes
<ligne vide>
Corps de la réponse
```

- Exemple : réponse à la demande d'obtention de index.html

```
HTTP/1.1 200 OK
Date: Fri, 16 Jun 2016 23:59:59 UTC
Content-type: text/html
<ligne vide>
<html><body><…
```

- Exemple: Réponse à l'envoi d'informations
  - C'est la même chose ☺

Les statuts

Petit traité de manipulation des ressources à l'usage des honnêtes gens

- Les statuts sont organisés par classe de réponse
  - **1xx** *Informational*: la requête a été reçue, continuer
  - **2xx** *Success* : la requête a bien été reçue, comprise et acceptée
  - **3xx** *Redirection* : des actions sont à entreprendre pour compléter la requête
  - **4xx** *Client Error* : la requête est mal écrite et ne peut-être traitée
  - **5xx** *Server Error* : le server n'arrive pas à traiter la requête

- Attention: tout les codes au dessus de 400 sont considérés comme des erreurs de processus.

Les codes les plus courants

- **200** : succès de la requête
- **201** : requête traitée avec succès et création d'un document
- **202** : requête traitée, mais sans garantie de résultat
- **204** : requête traitée avec succès mais pas d'information à renvoyer
- **301** : redirection permanente
- **302** : redirection temporaire
- 304 : non modifié
- **401** : utilisateur non authentifié
- **403** : accès refusé
- **404** : page non trouvée
- **405** : méthode de requête non autorisée
- **418** : I'm a teapot ☺
- **500 :** erreur interne du serveur.
- **503** : service temporairement indisponible ou en maintenance

- **Tips:** mieux vaut 204 que 404 : 404 ("tu t'es planté") vs 204 ("j'ai rien pour l'instant")

A mettre dans un coin de sa tête…

- Il est nécessaire de respecter l'esprit des RFC
  - Description des codes attendus en fonction des méthodes utilisées
- Ne jamais faire en sorte qu'une méthode **sûre** altère des ressources

```
GET /mailbox.php?action=delete&nb=1000
```

- Les méthodes idempotentes sont stockées dans les caches
- Si une méthode n'est pas idempotente, c'est qu'il y a une raison…
  - Ne pas mettre **POST** à toutes les sauces
- Pour les frontaux sous contrôle (Javascript…)
  - Ne pas utiliser **GET** quand un **HEAD** suffit

Les en-têtes

- Toute la communication d'information contextuel se fait avec des en-tête (*Header)*
- Syntaxe d'un header: pas d'espace dans le nom, en US-ASCII si possible

```
nom:valeur
```

- Il en est recensé **plus de 150 à l'IANA** (http://www.iana.org/assignments/message-headers/message-headers.xhtml)
- Les plus classiques
  - **Host** : site web vidé
  - **User-Agent** : identification de l'agent (navigateur)
  - **Content-Type**: la représentation de la ressource
  - **Content-Length**: taille en octets
  - **Expires** : date d'obsolescence de la ressource (gestion de cache)
  - **Last-Modified** : date de dernière modification de la ressource

La négociation de contenu

- Le client indique des préférences et le serveur tente d'y répondre avec des En-tête de négociation
  - **Accept** : types MIMES supportés par le client
  - **Accept-Charset** : encodage de caractère supporté
  - **Accept-Language**: langues acceptées

- Les valeurs sont *pondérées et* l'ordre importe

  Accept: text/html, application/xml;q=0.9,*/*;q=0.8
  - text/htmlsont acceptés à 100%
  - application/xml;q=0.9 avec une pondération de 90%
  - */*;q=0.8 tout le reste avec une pondération de 80%

- L'algorithme de décision est à la discrétion du serveur
- Le serveur détaille sa décision avec l'en-tête **vary.**
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation

www.cnrs.fr

Focus sur le cookie

Cookie

- Le cookie est un ensemble de clef/valeur envoyé par le serveur et renvoyé par le client pour maintenir un état.
- Il est échangé sous la forme de header

requête

```
GET /index.html HTTP/1.1
Host: nmsaw.fr
```

réponse

```
HTTP/1.1 200 Ok
Set-Cookie: onclose=destroyme
Set-Cookie: foo=bar;Expires=Wed, 20 Jan 2017 10:18:14 GMT
```

requête suivante

```
GET /index.html HTTP/1.1
Host: nmsaw.fr
Cookie: onclose=destroyme;foo=bar
```

Type de cookie

- **Cookie de session**: ne sert qu'à maintenir des informations de session
  - Détruit en fin de session ou à la fermeture du navigateur
- **Cookie persistant**: maintient d'information au-delà de la session
  - sur une durée limitée (date de péremption *Expires* ou durée limite *Max-Age)*
  - Permettent *aussi* de pister les utilisateurs: ***tracking cookie***
- **Cookie sécurisé*:*** ne peut être transmis que sur HTTPs
  - `Set-Cookie: onclose=destroyme; Secure`
- **httpOnly** : cookie ne pouvant être accédé par les API client (javascript)
  - `Set-Cookie: onclose=destroyme; HttpOnly`
- **Cookie tiers*:*** cookie d'un autre domaine fournit par une référence à une ressource tierce (régie publicitaire, pixel traqueur)

Les en-tête de gestion de cache
à la sauce REST

- Des en-têtes peuvent caractériser l'état d'une ressource par le **serveur**
  - **last-modified** : date de dernière modification

    ```
    Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT
    ```

  - **Etag**: identifiant de version

    ```
    ETag: "33a64df551425fcc55e4d42a148795d9f25f89d4"
    ```

- Le client peut indiquer des sélecteurs pour décrire l'état souhaité de la ressource
  - **if-modified-since** : demande une version plus récente que le dernier **last-modified** connu.

    ```
    If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT
    ```

  - **If-Not-Match** : demande une version qui ne matche pas le dernier Etag reçu.

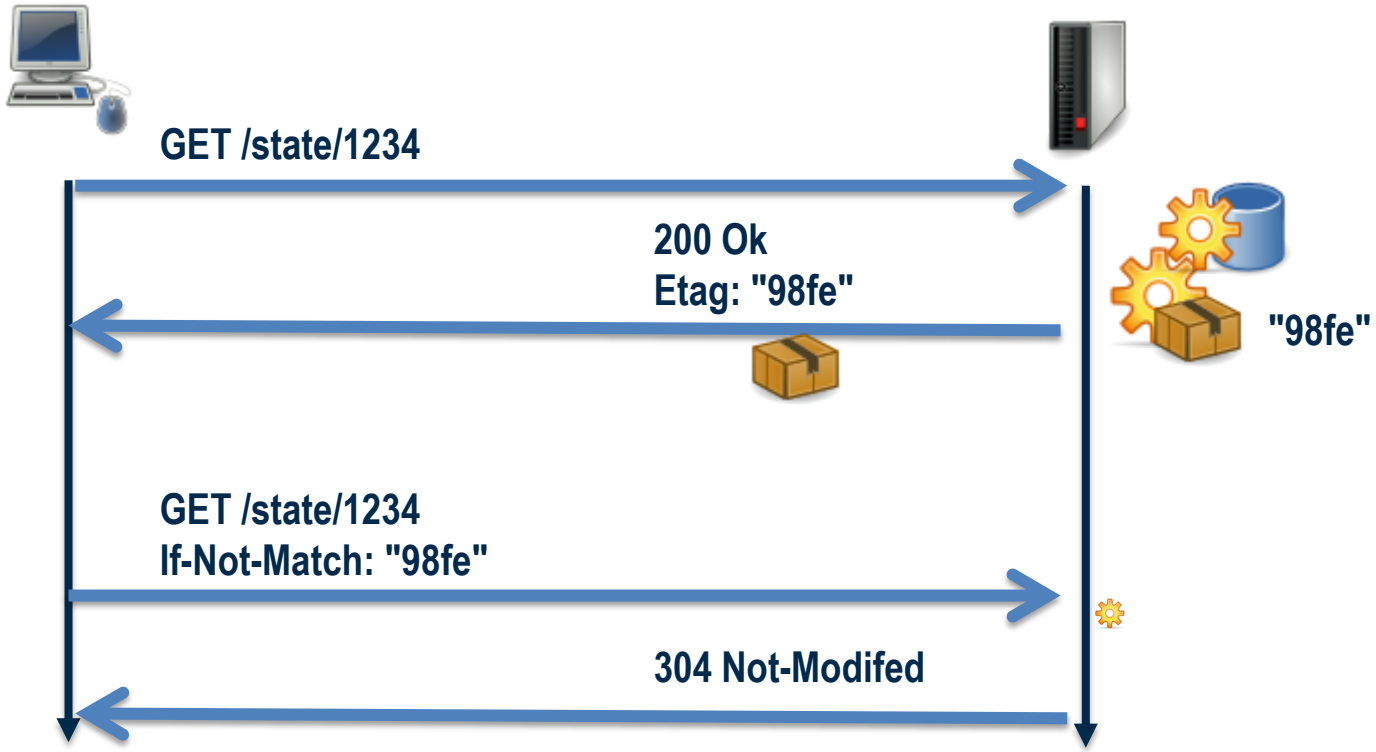
    ```
    If-Not-Match: "33a64df551425fcc55e4d42a148795d9f25f89d4"
    ```

  - Si la sélection est infructueuse pour GET et HEAD, le serveur renvoie un 304 Not Modified (entre autre)

- Il existe les opposés: **if-unmodified-since** et **If-Match**

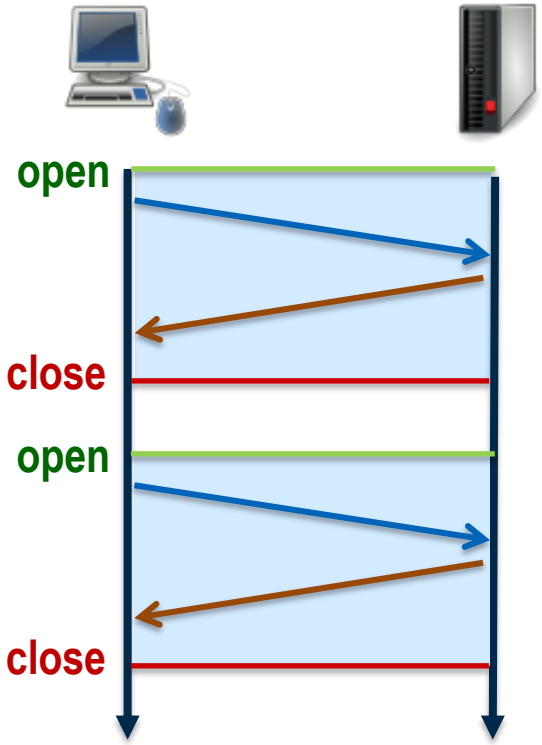
Les en-tête de gestion de cache à la sauce REST

- Dans la philosophie REST (**RE**presentational **S**tate **T**ransfer) l'usage de **Last-Modified** et **If-Not-Match** permet d'économiser des ressources.

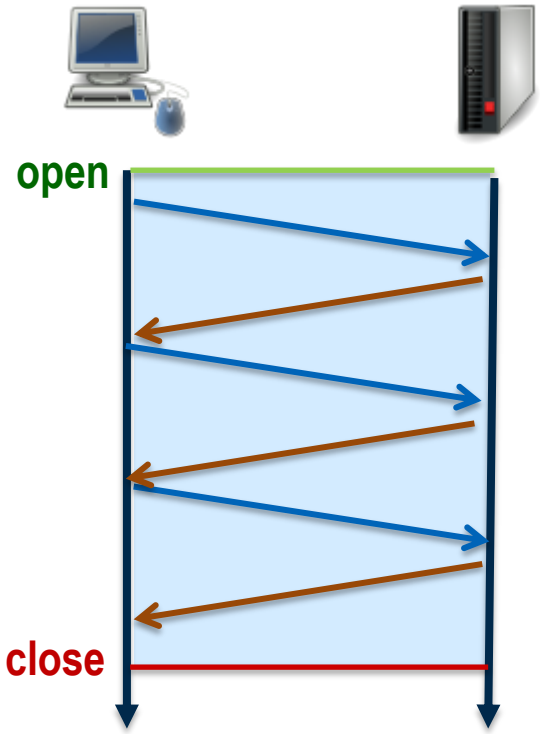


L'amélioration de la performance (1)

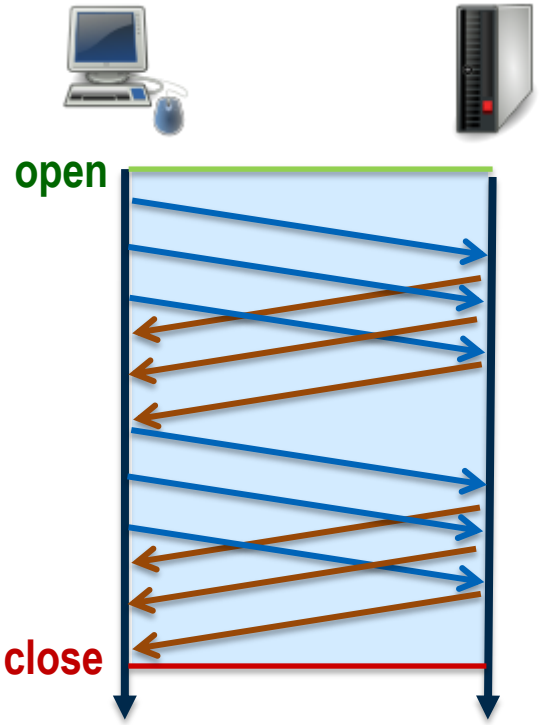
Connexions multiples



Connexion persistante



Pipelining



L'amélioration de la performance (2)

- Mode classique: 1 requête HTTP par connexion TCP
  - Lent
  - Préserve le serveur
- Connexion persistante
  - HTTP/1.0, il fallait le spécifier avec

    ```
    Connection: KeepAlive
    ```
  - HTTP/1.1, par défaut, la fermeture est indiquée par

    ```
    Connection: close
    ```
  - Time out de quelque secondes côté serveur
- Pipelining
  - Pas de pilotage par des en-têtes, uniquement par la gestion des connexions
  - Limité au méthodes idempotentes (GET, HEAD, PUT, DELETE)
  - Désactivé par défaut dans les navigateurs

Un mot sur HTTP/2

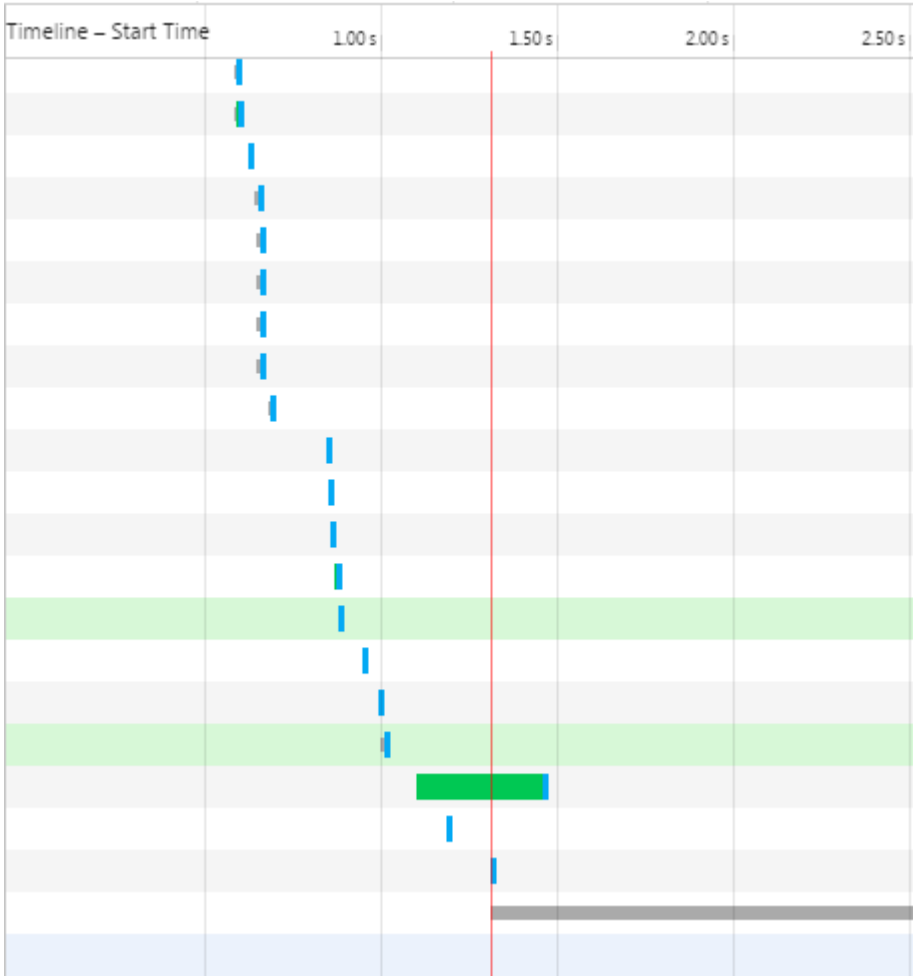
- Protocole issu de SPDY développé par Google.
- Ne remplace par HTTP/1.1
  - Lourd : 150 pages de norme contre 305 pour toutes les RFC HTTP/1.1
  - Encodage binaire (fin du *human-friendly*)
  - Utilisation d'un dictionnaire de compression des en-têtes
  - Utilisation de canaux virtuels au sein de TLS
- En cours de normalisation par l'IETF mais surtout poussé par le Géants du Web
  - Pas encore stabilisé côté serveur et clients
  - Analyse sécurité assez (Misc n°88)
- Il est urgent d'attendre…

La limitation du nombre de connexions

www.cnrs.fr

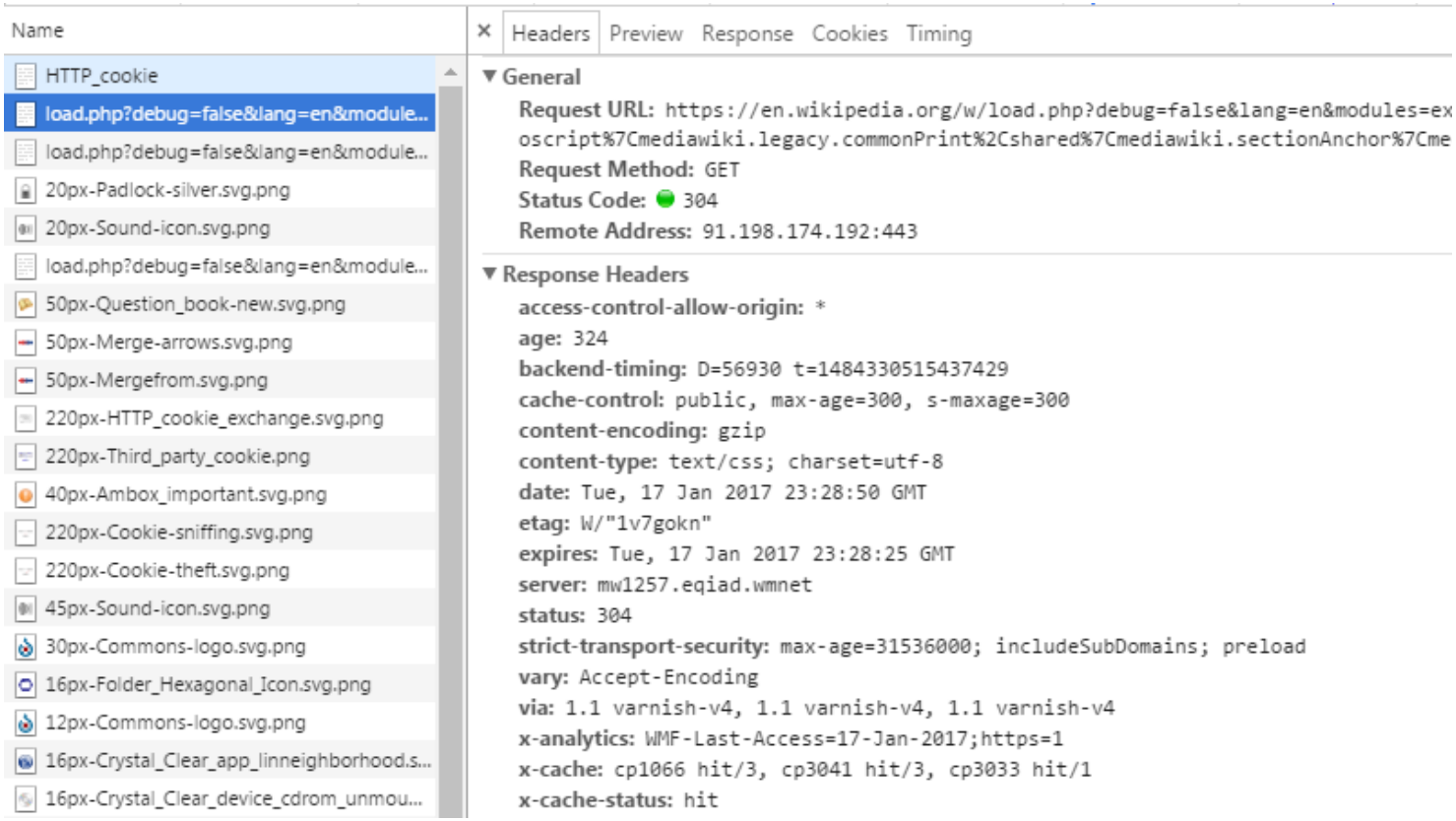
- HTTP/1.1 limitait les connexion simultanées à 2 par serveur
- La RFC mentionne désormais un nombre "raisonnable" par serveur
- Problème pour les applications mobiles (avant la 4G ?)
  - A la source de multiples stratégies de chargement
  - Ne pas comptabiliser sur le même domaine
    - CDN: Content Delivery Network, pour les librairies javascripts, Domain Sharding: découper son domaine en petit bout
  - La *concaténation+compression+sprit* pour limiter le nombre de dépendance à télécharger
  - L'ordonnancement des dépendances (les scripts JS à la fin)

BROWSER	MAX PARALLEL CONNECTIONS PER HOST
IE 6 and 7	2
IE 8	6
IE 9	6
IE 10	8
Firefox 2	2
Firefox 3	6
Firefox 4 to 17	6
Opera 9.63	4
Opera 10	8
Opera 11 and 12	6
Chrome 1 and 2	6
Chrome 3	4
Chrome 4 to 23	6
Safari 3 and 4	4



Outillage de base

- Sur tous les navigateurs, la touche magique: **F12**

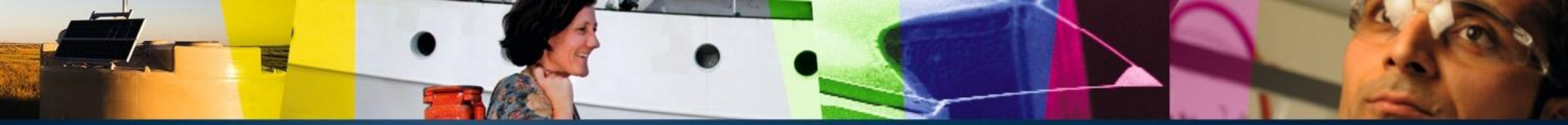


Pour les (très) curieux : Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
30	3.161347	104.25.219.21	192.168.1.21	TLSv1.2	92	Application Data
31	3.360119	192.168.1.21	104.25.219.21	TCP	54	54864 → 443 [ACK] Seq=133 Ack=4251 Win=16415 Len=0
32	3.412512	192.168.1.21	69.89.31.88	TCP	66	54899 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
33	3.589006	69.89.31.88	192.168.1.21	TCP	66	80 → 54899 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS
34	3.589069	192.168.1.21	69.89.31.88	TCP	54	54899 → 80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
35	3.590677	192.168.1.21	69.89.31.88	HTTP	409	GET /_images/Three_way_handshake.png HTTP/1.1
36	3.767257	69.89.31.88	192.168.1.21	TCP	54	80 → 54899 [ACK] Seq=1 Ack=356 Win=30336 Len=0
39	4.851488	69.89.31.88	192.168.1.21	TCP	1514	[TCP segment of a reassembled PDU]
40	4.852224	69.89.31.88	192.168.1.21	TCP	1514	[TCP segment of a reassembled PDU]

```
▷ Frame 35: 409 bytes on wire (3272 bits), 409 bytes captured (3272 bits) on interface 0
▷ Ethernet II, Src: IntelCor_2a:16:0b (80:00:0b:2a:16:0b), Dst: Sagemcom_8d:5b:f0 (18:62:2c:8d:5b:f0)
▷ Internet Protocol Version 4, Src: 192.168.1.21, Dst: 69.89.31.88
▷ Transmission Control Protocol, Src Port: 54899 (54899), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 355
◢ Hypertext Transfer Protocol
  ▷ GET /_images/Three_way_handshake.png HTTP/1.1\r\n
    Host: www.netpro360.com\r\n
    Connection: keep-alive\r\n
    User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36\r\n
    Accept: image/webp,image/*,*/*;q=0.8\r\n
    Accept-Encoding: gzip, deflate, sdch\r\n
    Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4\r\n
    \r\n
    [Full request URI: http://www.netpro360.com/_images/Three_way_handshake.png]
    [HTTP request 1/1]
```

```
0030  40 29 fa bc 00 00 47 45  54 20 2f 5f 69 6d 61 67   @)....GE T /_imag
0040  65 73 2f 54 68 72 65 65  5f 77 61 79 5f 68 61 6e   es/Three _way_han
0050  64 73 68 61 6b 65 2e 70  6e 67 20 48 54 54 50 2f   dshake.p ng HTTP/
0060  31 2e 31 0d 0a 48 6f 73  74 3a 20 77 77 77 2e 6e   1.1..Hos t: www.n
0070  65 74 70 72 6f 33 36 30  2e 63 6f 6d 0d 0a 43 6f   etpro360 .com..Co
0080  6e 6e 65 63 74 69 6f 6e  3a 20 6b 65 65 70 2d 61   nnection : keep-a
0090  6c 69 76 65 0d 0a 55 73  65 72 2d 41 67 65 6e 74   live..Us er-Agent
00a0  3a 20 4d 6f 7a 69 6c 6c  61 2f 35 2e 30 20 28 57   : Mozill a/5.0 (W
00b0  69 6e 64 6f 77 73 20 4e  54 20 36 2e 31 3b 20 57   indows N T 6.1; W
00c0  69 6e 36 34 3b 20 78 36  34 29 20 41 70 70 6c 65   in64; x6 4) Apple
```



Et encore tant à dire

www.cnrs.fr

- Les authentifications Basic et Digest
- Le chunking
- TLS en détails
- Des références
  - https://developer.mozilla.org/en-US/docs/Web/HTTP
  - http://wiki.linuxwall.info/doku.php/fr:ressources:dossiers:ssl_pki:1_les_bases
  - https://en.wikipedia.org/wiki/HTTP_cookie
  - https://www.ssllabs.com/ (pour se faire peur)

RFC de référence de HTTP/1.1

http://www.bortzmeyer.org/http-11-reecrit.html

www.cnrs.fr

- RFC 7230, qui décrit l'architecture, les URI, et la syntaxe des messages,
- RFC 7231, qui décrit la sémantique des messages, les codes de retour à trois chiffres, les en-têtes des requêtes et des réponses,
- RFC 7232, sur les requêtes conditionnelles,
- RFC 7233, normalise les requêtes demandant une portion d'un contenu, en spécifiant un intervalle,
- RFC 7234, décrit le fonctionnement des caches Web,
- RFC 7235, spécifie les mécanismes d'authentification de HTTP,
- RFC 7236, enregistre les anciens mécanismes d'authentification, qui avaient été spécifiés avant le RFC 7235,
- RFC 7237, enregistre les anciennes méthodes HTTP, pour initialiser le registre.

www.cnrs.fr

Fin

questions ?