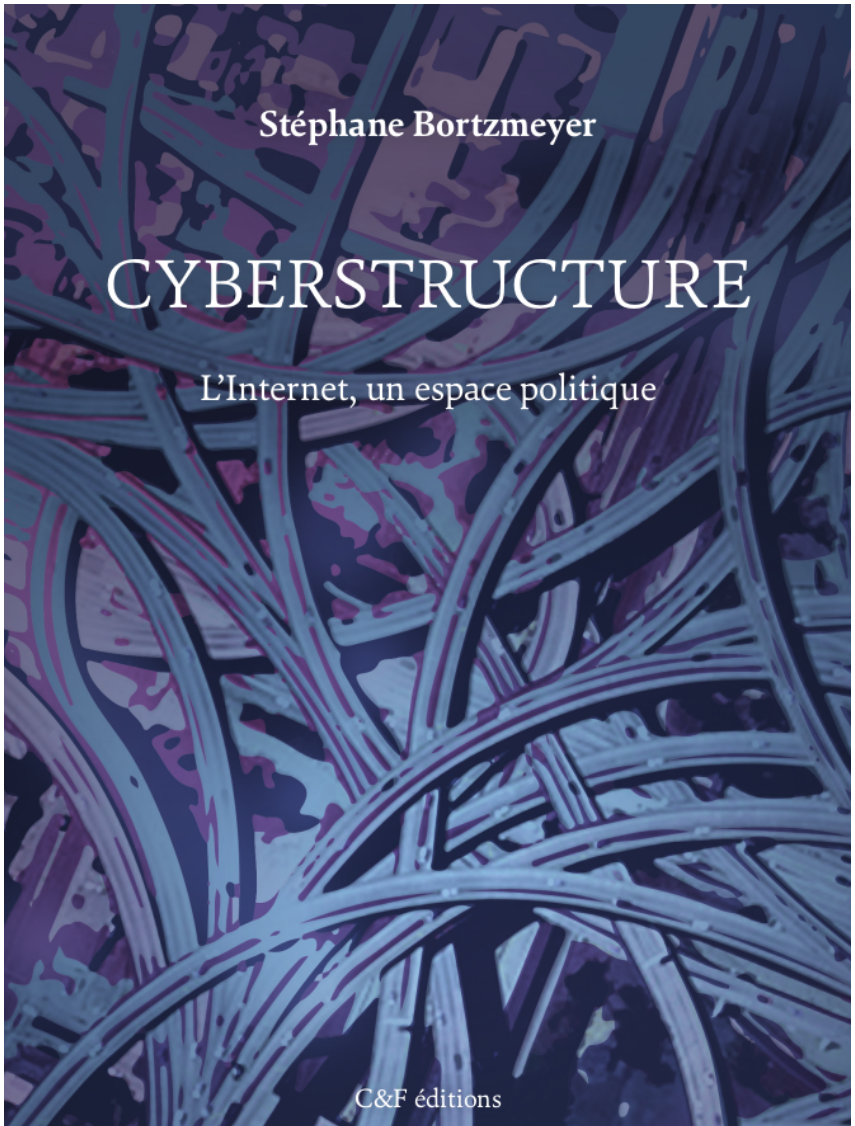
# Hypertext Transfer Protocol

Stéphane Bortzmeyer
stephane+cnam@bortzmeyer.org

CNAM, 17 avril 2019

# Auteur du livre « Cyberstructure »

Chez C&F Éditions. `https://cyberstructure.fr/`



# Plan du tutoriel

# Généralités

- On parlera bien de HTTP, pas du Web en général,

# Généralités

- On parlera bien de HTTP, pas du Web en général,
- Protocole de couche 7 (Application),

# Généralités

- On parlera bien de HTTP, pas du Web en général,
- Protocole de couche 7 (Application),
- Transfert de fichiers, simple,

# Généralités

- On parlera bien de HTTP, pas du Web en général,
- Protocole de couche 7 (Application),
- Transfert de fichiers, simple,
- Agnostique par rapport au contenu du fichier transféré,

# Généralités

- On parlera bien de HTTP, pas du Web en général,
- Protocole de couche 7 (Application),
- Transfert de fichiers, simple,
- Agnostique par rapport au contenu du fichier transféré,
- Créé pour le Web, mais peut s'utiliser ailleurs.

# Plan du tutoriel

# Un protocole client/serveur

# Un protocole client/serveur

- HTTP est requête/réponse,

# Un protocole client/serveur

- HTTP est requête/réponse,
- Asymétrique (client/serveur),

# Un protocole client/serveur

- HTTP est requête/réponse,
- Asymétrique (client/serveur),
- Le **client** ouvre une connexion TCP avec le **serveur**,

# Un protocole client/serveur

- HTTP est requête/réponse,
- Asymétrique (client/serveur),
- Le **client** ouvre une connexion TCP avec le **serveur**,
- Le client envoie une requête,

# Un protocole client/serveur

- HTTP est requête/réponse,
- Asymétrique (client/serveur),
- Le **client** ouvre une connexion TCP avec le **serveur**,
- Le client envoie une requête,
- Le serveur envoie une réponse (comportant parfois un corps),

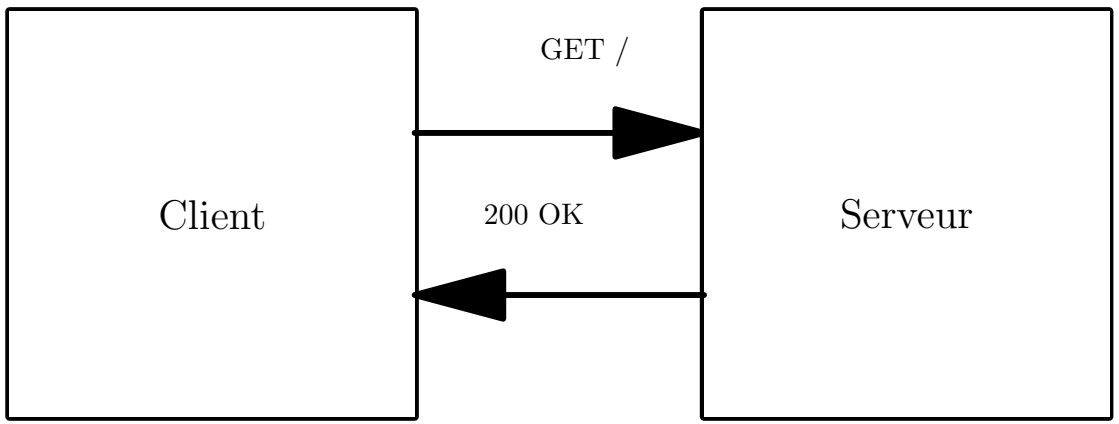
# Un protocole client/serveur

- HTTP est requête/réponse,
- Asymétrique (client/serveur),
- Le **client** ouvre une connexion TCP avec le **serveur**,
- Le client envoie une requête,
- Le serveur envoie une réponse (comportant parfois un corps),
- Au revoir (pas de notion de session).

# Un protocole client/serveur

- HTTP est requête/réponse,
- Asymétrique (client/serveur),
- Le **client** ouvre une connexion TCP avec le **serveur**,
- Le client envoie une requête,
- Le serveur envoie une réponse (comportant parfois un corps),
- Au revoir (pas de notion de session).
- Requêtes et réponses en texte, pas en binaire (pour HTTP/1).

# HTTP, c'est simple



# Vu avec tshark, www.internic.net

```
% tshark -r http.pcap
1 2001:67c:1348:7::86:133 → 2620:0:2d0:200::9 TCP 94 60900 → 80 [SYN] Seq=
2 2620:0:2d0:200::9 → 2001:67c:1348:7::86:133 TCP 94 80 → 60900 [SYN, ACK]
3 2001:67c:1348:7::86:133 → 2620:0:2d0:200::9 TCP 86 60900 → 80 [ACK] Seq=
4 2001:67c:1348:7::86:133 → 2620:0:2d0:200::9 HTTP 166 GET / HTTP/1.1
5 2620:0:2d0:200::9 → 2001:67c:1348:7::86:133 TCP 86 80 → 60900 [ACK] Seq=
6 2620:0:2d0:200::9 → 2001:67c:1348:7::86:133 TCP 1484 HTTP/1.1 200 OK  [TC
7 2001:67c:1348:7::86:133 → 2620:0:2d0:200::9 TCP 86 60900 → 80 [ACK] Seq=
8 2620:0:2d0:200::9 → 2001:67c:1348:7::86:133 TCP 1514 80 → 60900 [ACK] Se
9 2001:67c:1348:7::86:133 → 2620:0:2d0:200::9 TCP 86 60900 → 80 [ACK] Seq=
10 2620:0:2d0:200::9 → 2001:67c:1348:7::86:133 HTTP 6861 HTTP/1.1 200 OK  (
11 2001:67c:1348:7::86:133 → 2620:0:2d0:200::9 TCP 86 60900 → 80 [ACK] Seq
12 2001:67c:1348:7::86:133 → 2620:0:2d0:200::9 TCP 86 60900 → 80 [FIN, ACK
13 2620:0:2d0:200::9 → 2001:67c:1348:7::86:133 TCP 86 80 → 60900 [ACK] Seq
14 2620:0:2d0:200::9 → 2001:67c:1348:7::86:133 TCP 86 80 → 60900 [FIN, ACK
15 2001:67c:1348:7::86:133 → 2620:0:2d0:200::9 TCP 86 60900 → 80 [ACK] Seq
```

# La requête

# La requête

- HTTP agit sur des **ressources** (exemple : une page Web est une ressource),

# La requête

- HTTP agit sur des **ressources** (exemple : une page Web est une ressource),
- La requête comprend une **méthode** comme `GET` (récupérer), `PUT` (mettre), `POST` (modifier)...

# La requête

- HTTP agit sur des **ressources** (exemple : une page Web est une ressource),
- La requête comprend une **méthode** comme `GET` (récupérer), `PUT` (mettre), `POST` (modifier)...
- Et un chemin désignant la ressource,

# La requête

- HTTP agit sur des **ressources** (exemple : une page Web est une ressource),
- La requête comprend une **méthode** comme `GET` (récupérer), `PUT` (mettre), `POST` (modifier)...
- Et un chemin désignant la ressource,
- Et des **en-têtes** pour indiquer... un peu de tout.

# La requête

- HTTP agit sur des **ressources** (exemple : une page Web est une ressource),
- La requête comprend une **méthode** comme `GET` (récupérer), `PUT` (mettre), `POST` (modifier)...
- Et un chemin désignant la ressource,
- Et des **en-têtes** pour indiquer... un peu de tout.
- Syntaxe des en-têtes : nom, deux-points, valeur,

# La requête

- HTTP agit sur des **ressources** (exemple : une page Web est une ressource),
- La requête comprend une **méthode** comme `GET` (récupérer), `PUT` (mettre), `POST` (modifier)...
- Et un chemin désignant la ressource,
- Et des **en-têtes** pour indiquer... un peu de tout.
- Syntaxe des en-têtes : nom, deux-points, valeur,
- Exemples d'en-têtes : `User-Agent` (type de logiciel client), `Accept` (type de formats acceptés)...

# La requête

- HTTP agit sur des **ressources** (exemple : une page Web est une ressource),
- La requête comprend une **méthode** comme `GET` (récupérer), `PUT` (mettre), `POST` (modifier)...
- Et un chemin désignant la ressource,
- Et des **en-têtes** pour indiquer... un peu de tout.
- Syntaxe des en-têtes : nom, deux-points, valeur,
- Exemples d'en-têtes : `User-Agent` (type de logiciel client), `Accept` (type de formats acceptés)...
- Parfois un corps (charge utile, par exemple un fichier qu'on `PUT`).

# La réponse

# La réponse

- La réponse comprend un code de retour en trois chiffres,

# La réponse

- La réponse comprend un code de retour en trois chiffres,
- Des en-têtes,

# La réponse

- La réponse comprend un code de retour en trois chiffres,
- Des en-têtes,
- Exemples d'en-têtes : `Last-Modified` (dernière modification de la ressource), `Content-Length` (taille de la ressource), `Content-Type` (type de la ressource)...

# La réponse

- La réponse comprend un code de retour en trois chiffres,
- Des en-têtes,
- Exemples d'en-têtes : `Last-Modified` (dernière modification de la ressource), `Content-Length` (taille de la ressource), `Content-Type` (type de la ressource)...
- Souvent un corps.

# Les codes de retour

# Les codes de retour

- Trois chiffres, le premier indique la catégorie (2xx, ça va, 4xx, vous avez fait une erreur, 5xx, j'ai fait une erreur...)

# Les codes de retour

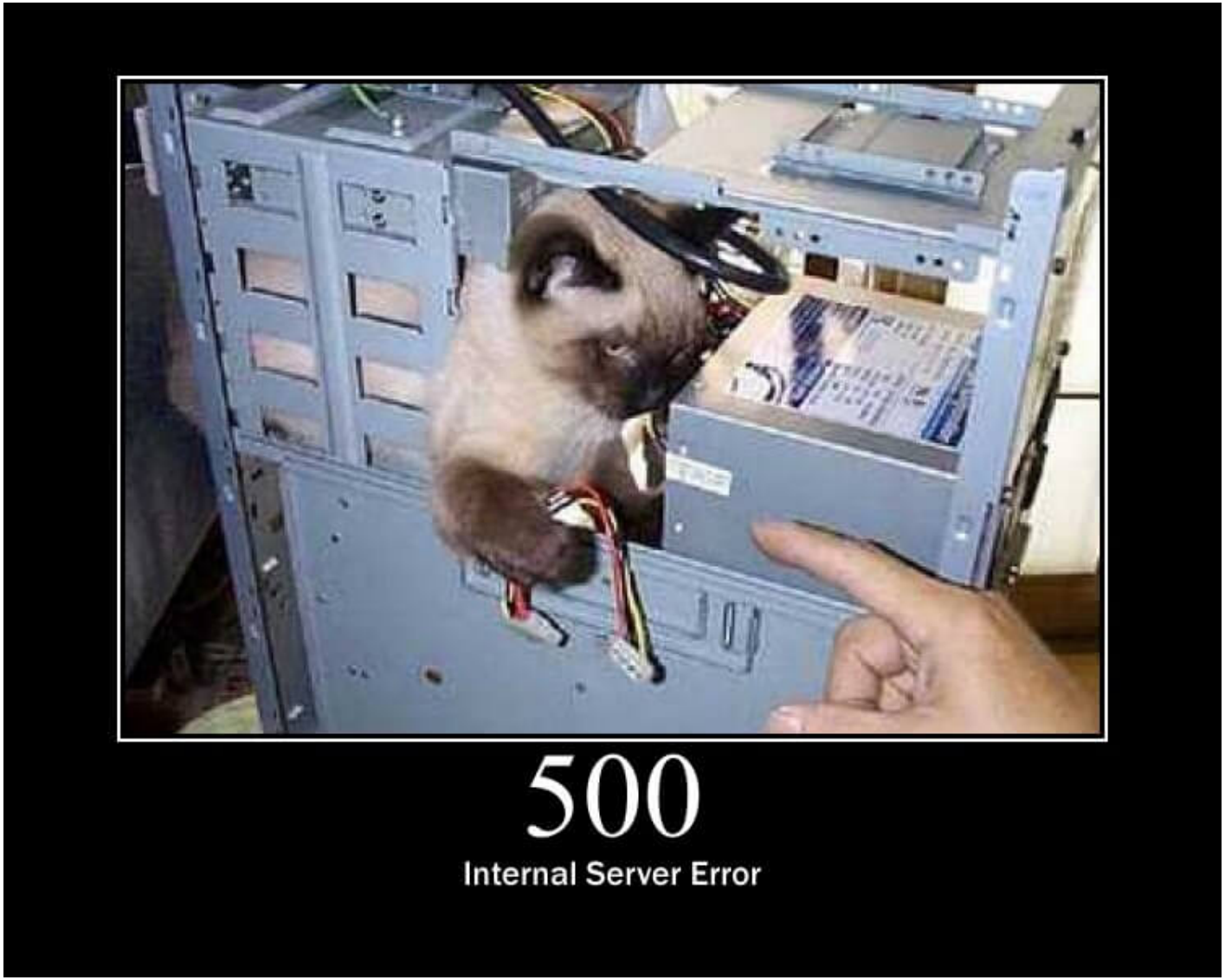
- Trois chiffres, le premier indique la catégorie (2xx, ça va, 4xx, vous avez fait une erreur, 5xx, j'ai fait une erreur...)
- Quelques codes fameux (et merci à `https://http.cat/`).

# 200 OK

# 403 Forbidden

# 404 Not Found

# 500 Internal Server Error



# Vu avec curl

```
% curl -v http://www.cnam.fr/
* Connected to www.cnam.fr (163.173.128.40) port 80 (#0)
> GET / HTTP/1.1
> Host: www.cnam.fr
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 08 Apr 2019 15:14:12 GMT
< Server: Apache/2.4.10 (Debian)
< Set-Cookie: JSESSIONID=F70B19A7BFFFD3FB25C139FFA7527B6D.abeliap; Path=/; H
< Vary: Accept-Encoding
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html;charset=UTF-8
< Set-Cookie: SERVERID=abeliap; path=/
< Cache-control: private
<
<!DOCTYPE html>
```

# Exemple de requête

```
Internet Protocol Version 6, Src: 2001:67c:1348:7::86:133, Dst: 2620:0:2d0:2
    Next Header: TCP (6)
    Source: 2001:67c:1348:7::86:133
    Destination: 2620:0:2d0:200::9
Transmission Control Protocol, Src Port: 60900, Dst Port: 80, Seq: 1, Ack: 1
    Source Port: 60900
    Destination Port: 80
    TCP payload (80 bytes)
Hypertext Transfer Protocol
    GET / HTTP/1.1\r\n
        [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
            [GET / HTTP/1.1\r\n]
            [Severity level: Chat]
            [Group: Sequence]
        Request Method: GET
        Request URI: /
        Request Version: HTTP/1.1
    Host: www.internic.net\r\n
    User-Agent: curl/7.52.1\r\n
```

# Exemple de réponse

```
Internet Protocol Version 6, Src: 2620:0:2d0:200::9, Dst: 2001:67c:1348:7::8
    Next Header: TCP (6)
    Source: 2620:0:2d0:200::9
    Destination: 2001:67c:1348:7::86:133
Transmission Control Protocol, Src Port: 80, Dst Port: 60900, Seq: 2827, Ack
    Source Port: 80
    Destination Port: 60900
    TCP payload (6775 bytes)
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Mon, 08 Apr 2019 15:29:00 GMT\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    Last-Modified: Sun, 11 Jun 2017 00:01:00 GMT\r\n
    Content-Length: 9133\r\n
    \r\n
    <!DOCTYPE html>\n
    <html>\n
    <head>\n
    <title>InterNIC | The Internet's Network Information Center</title>\n
```

# Plan du tutoriel

# Le Web classique

# Le Web classique

- Un navigateur Web sert de client,

# Le Web classique

- Un navigateur Web sert de client,
- Il récupère de l'HTML et l'affiche,

# Le Web classique

- Un navigateur Web sert de client,
- Il récupère de l'HTML et l'affiche,
- HTTP ne fait « que » le transfert de fichiers.

# Transfert de fichiers

# Transfert de fichiers

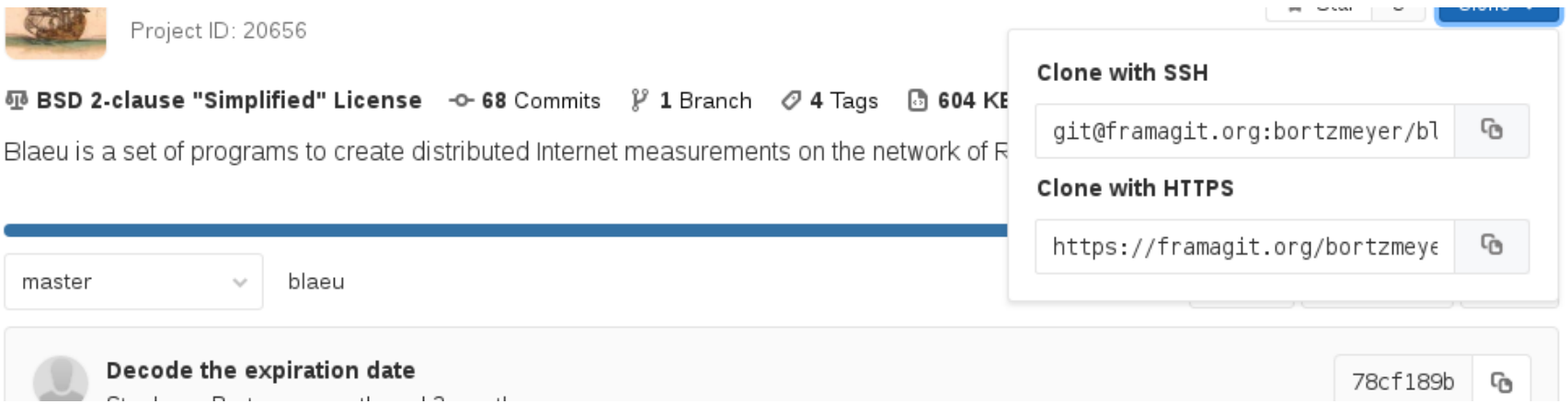
- HTTP peut transférer du contenu, même en dehors de tout navigateur Web,

# Transfert de fichiers

- HTTP peut transférer du contenu, même en dehors de tout navigateur Web,
- Exemple : mise à jour de logiciel (sur Debian, sources HTTP dans `/etc/apt/sources.list`),

# Transfert de fichiers

- HTTP peut transférer du contenu, même en dehors de tout navigateur Web,
- Exemple : mise à jour de logiciel,
- Exemple : le logiciel de gestion de versions Git récupérant un dépôt en HTTP :



# REST

# REST

- REST (*Representational state transfer*) est une architecture d'accès à des services au dessus de HTTP,

# REST

- REST est une architecture d'accès à des services au dessus de HTTP,
- Les méthodes HTTP sont utilisées par exemple pour mettre en œuvre les opérations CRUD (*Create*, *Read*, *Update*, *Delete*).

# REST

- REST est une architecture d'accès à des services au dessus de HTTP,
- Les méthodes HTTP sont utilisées par exemple pour mettre en œuvre les opérations CRUD (*Create*, *Read*, *Update*, *Delete*).
- API (*Application programming interface*) au-dessus de HTTP, pour accéder à des services distants,

# Exemple d'API

Le cours du Bitcoin, chez CoinDesk :

```
% curl -s https://api.coindesk.com/v1/bpi/currentprice.json
...
    "EUR": {
      "code": "EUR",
      "symbol": "&euro;",
      "rate": "4,508.2361",
      "description": "Euro",
    }
  }
```

# Exemple d'API

Le protocole RDAP :

```
% curl  https://rdap.db.ripe.net/ip/2001:660:330f:2::7f
...
  "name" : "FR-ENST-PARIS-RAP-1",
...
    "remarks" : [ {
    "description" : [ "Ecole Nationale Superieure des Telecommunications - T
  } ],
...
```

# Exemple d'API

Un exemple CRUD `https://petstore.swagger.io/`

```
%  curl https://petstore.swagger.io/v2/pet/21
{"code":1,"type":"error","message":"Pet not found"}

% curl -H "Content-type: application/json" --data "$(cat potamochere.json)"
   -X PUT https://petstore.swagger.io/v2/pet/

%  curl https://petstore.swagger.io/v2/pet/21
{"id":21,"category":{"id":0,"name":"Dom"},"name":"Pork",
   "photoUrls":["https://www.potamochère.fr/potamoch%C3%A8re.jpg"],...
```

# Plan du tutoriel

# Clients

# Clients

- Navigateurs Web, bien sûr (Firefox, Chrome, Edge, lynx),

# Clients

- Navigateurs Web, bien sûr (Firefox, Chrome, Edge, lynx),
- Ligne de commande (curl, wget),

# Clients

- Navigateurs Web, bien sûr (Firefox, Chrome, Edge, lynx),
- Ligne de commande (curl, wget),
- *Crawlers* (*bots*),

# Clients

- Navigateurs Web, bien sûr (Firefox, Chrome, Edge, lynx),
- Ligne de commande (curl, wget),
- *Crawlers* (*bots*),
- Bibliothèques pour tous les langages.

# Serveurs

# Serveurs

- Serveurs autonomes (Apache, Nginx),

# Serveurs

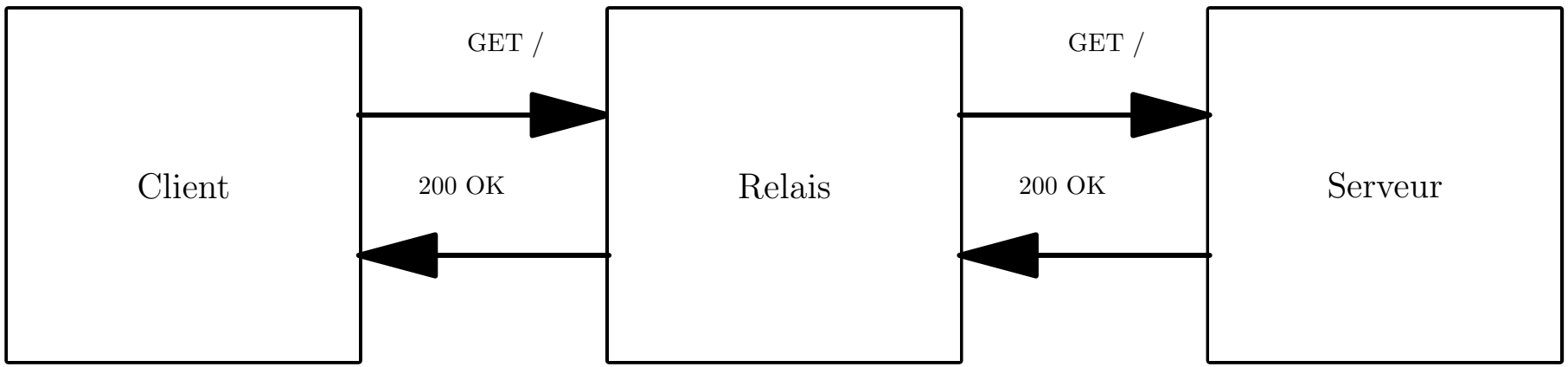
- Serveurs autonomes (Apache, Nginx),
- Bibliothèques pour tous les langages.

# Un serveur HTTP en trois lignes de Python

```
import http.server

server_address = ('', 8080)
httpd = http.server.HTTPServer(server_address,
                               http.server.SimpleHTTPRequestHandler)
httpd.serve_forever()
```

# Avec intermédiaire



# Relais, caches et autres intermédiaires

# Relais, caches et autres intermédiaires

- Relais (*proxies*), pour divers contournements,

# Relais, caches et autres intermédiaires

- Relais (*proxies*), pour divers contournements,
- Caches (mémorisation des ressources) pour économiser la capacité réseau (moins important maintenant, avec TLS),

# Relais, caches et autres intermédiaires

- Relais (*proxies*), pour divers contournements,
- Caches (mémorisation des ressources) pour économiser la capacité réseau,
- Répartiteurs de charge,

# Relais, caches et autres intermédiaires

- Relais (*proxies*), pour divers contournements,
- Caches (mémorisation des ressources) pour économiser la capacité réseau,
- Répartiteurs de charge,
- Pare-feux, pour protéger des applications Web écrites avec les pieds.

# Plan du tutoriel

# L'Internet n'est plus ce qu'il était

# L'Internet n'est plus ce qu'il était

- De plus en plus d'intermédiaires sur le trajet (pare-feux, par exemple),

# L'Internet n'est plus ce qu'il était

- De plus en plus d'intermédiaires sur le trajet (pare-feux, par exemple),
- Souvent bogués et/ou mal configurés,

# L'Internet n'est plus ce qu'il était

- De plus en plus d'intermédiaires sur le trajet (pare-feux, par exemple),
- Souvent bogués et/ou mal configurés,
- Par défaut, bloquent tout ce qu'ils ne connaissent pas,

# L'Internet n'est plus ce qu'il était

- De plus en plus d'intermédiaires sur le trajet (pare-feux, par exemple),
- Souvent bogués et/ou mal configurés,
- Par défaut, bloquent tout ce qu'ils ne connaissent pas,
- Cela mène à une **ossification** de l'Internet. Il est de plus en plus dur de déployer quelque chose de nouveau.

# L'exemple de DoH

# L'exemple de DoH

- Pour contourner la censure, DNS devient chiffré,

# L'exemple de DoH

- Pour contourner la censure, DNS devient chiffré,
- DNS sur TLS utilise le port 853, qui peut être bloqué,

# L'exemple de DoH

- Pour contourner la censure, DNS devient chiffré,
- DNS sur TLS utilise le port 853, qui peut être bloqué,
- D'où DoH (DNS sur HTTPS, RFC 8484).

# L'exemple de DoH

- Pour contourner la censure, DNS devient chiffré,
- DNS sur TLS utilise le port 853, qui peut être bloqué,
- D'où DoH (DNS sur HTTPS, RFC 8484).
- Tout sur le port 443 ?

# Plan du tutoriel

# Simplicité d'abord

# Simplicité d'abord

- Pas spécialement de sécurité dans HTTP au début,

# Simplicité d'abord

- Pas spécialement de sécurité dans HTTP au début,
- Pas d'authentification,

# Simplicité d'abord

- Pas spécialement de sécurité dans HTTP au début,
- Pas d'authentification,
- Pas de session,

# Simplicité d'abord

- Pas spécialement de sécurité dans HTTP au début,
- Pas d'authentification,
- Pas de session,
- Tout en clair.

# Simplicité d'abord

- Pas spécialement de sécurité dans HTTP au début,
- Pas d'authentification,
- Pas de session,
- Tout en clair.
- Cette simplicité a permis à HTTP de s'imposer.

# Biscuits

# Biscuits

- But : créer une **session** à partir d'un ensemble de requêtes/réponses HTTP.

# Biscuits

- But : créer une **session** à partir d'un ensemble de requêtes/réponses HTTP.
- Solution : le biscuit (*cookie*).

# Biscuits

- But : créer une **session** à partir d'un ensemble de requêtes/réponses HTTP.
- Solution : le biscuit (*cookie*).
- Un petit groupe de bits, que le serveur fabrique et envoie au client au début de la session,

# Biscuits

- But : créer une **session** à partir d'un ensemble de requêtes/réponses HTTP.
- Solution : le biscuit (*cookie*).
- Un petit groupe de bits, que le serveur fabrique et envoie au client au début de la session,
- et que le client renvoie à chaque requête, permettant de voir que c'est la même session,

# Biscuits

- But : créer une **session** à partir d'un ensemble de requêtes/réponses HTTP.
- Solution : le biscuit (*cookie*).
- Un petit groupe de bits, que le serveur fabrique et envoie au client au début de la session,
- et que le client renvoie à chaque requête, permettant de voir que c'est la même session,
- Normalisés dans le RFC 6265.

# Authentification

# Authentification

- Peut se faire par des en-têtes HTTP `Authorization:` envoyés à chaque requête,

# Authentification

- Peut se faire par des en-têtes HTTP `Authorization:` envoyés à chaque requête,
- Exemple, mot de passe mémorisé par le client HTTP,

# Authentification

- Peut se faire par des en-têtes HTTP `Authorization:` envoyés à chaque requête,
- Exemple, mot de passe mémorisé par le client HTTP,
- Ne permet pas de se déconnecter,

# Authentification

- Peut se faire par des en-têtes HTTP `Authorization:` envoyés à chaque requête,
- Exemple, mot de passe mémorisé par le client HTTP,
- Ne permet pas de se déconnecter,
- En pratique, l'authentification est presque toujours faite par une solution non-HTTP (exemple : mot de passe dans un formulaire et deuxième facteur par exemple sur une application externe),

# Authentification

- Peut se faire par des en-têtes HTTP `Authorization:` envoyés à chaque requête,
- Exemple, mot de passe mémorisé par le client HTTP,
- Ne permet pas de se déconnecter,
- En pratique, l'authentification est presque toujours faite par une solution non-HTTP,
- puis les *cookies* maintiennent la session (« c'est bon, on a déjà authentifié le possesseur de ce biscuit »).

# Authentification

- Peut se faire par des en-têtes HTTP `Authorization:` envoyés à chaque requête,
- Exemple, mot de passe mémorisé par le client HTTP,
- Ne permet pas de se déconnecter,
- En pratique, l'authentification est presque toujours faite par une solution non-HTTP,
- puis les *cookies* maintiennent la session (« c'est bon, on a déjà authentifié le possesseur de ce biscuit »).
- (Il y a aussi les certificats clients mais c'est très rare.)

# Authentification

- Peut se faire par des en-têtes HTTP `Authorization:` envoyés à chaque requête,
- Exemple, mot de passe mémorisé par le client HTTP,
- Ne permet pas de se déconnecter,
- En pratique, l'authentification est presque toujours faite par une solution non-HTTP,
- puis les *cookies* maintiennent la session (« c'est bon, on a déjà authentifié le possesseur de ce biscuit »).
- (Il y a aussi les certificats clients mais c'est très rare.)
- Les API utilisent d'autres solutions : biscuit géré manuellement et mis dans l'URL ou dans un en-tête plus ou moins spécifique.

# Tout en clair

# Tout en clair

- Le trafic numérique est trop facilement écoutable, et traitable,

# Tout en clair

- Le trafic numérique est trop facilement écoutable, et traitable,
- Communiquer **en clair** en 2019 n'est plus acceptable,

# Tout en clair

- Le trafic numérique est trop facilement écoutable, et traitable,
- Communiquer **en clair** en 2019 n'est plus acceptable,
- La surveillance est massivement pratiquée, par les opérateurs, les États, les délinquants...

# Tout en clair

- Le trafic numérique est trop facilement écoutable, et traitable,
- Communiquer **en clair** en 2019 n'est plus acceptable,
- La surveillance est massivement pratiquée, par les opérateurs, les États, les délinquants...
- Révélations Snowden en 2013 : plus moyen de faire l'autruche.

# TLS

# TLS

- *Transport Layer Security*, une couche de chiffrement pour les applications,

# TLS

- *Transport Layer Security*, une couche de chiffrement pour les applications,
- Normalisé dans le RFC 8446,

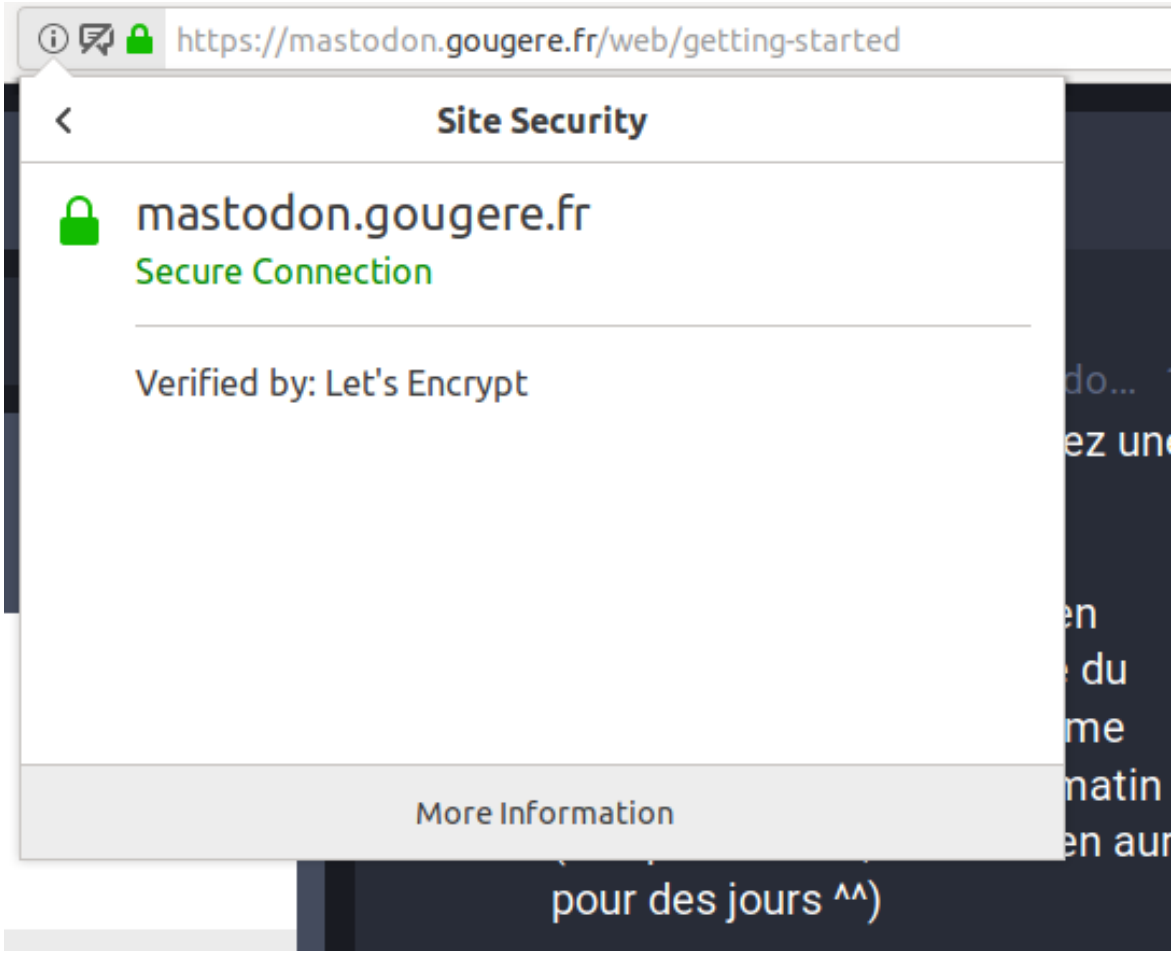
# TLS

- *Transport Layer Security*, une couche de chiffrement pour les applications,
- Normalisé dans le RFC 8446,
- Principe : on chiffre au départ et on déchiffre à l'arrivée. Protège contre un tiers mais pas contre les extrêmités.

# TLS

- *Transport Layer Security*, une couche de chiffrement pour les applications,
- Normalisé dans le RFC 8446,
- Principe : on chiffre au départ et on déchiffre à l'arrivée. Protège contre un tiers mais pas contre les extrêmités.
- Risque d'attaque de l'Homme du Milieu : il faut **authentifier** l'extrêmité.

# Exemple de certificat



# Certificats

# Certificats

- But : authentifier le serveur TLS,

# Certificats

- But : authentifier le serveur TLS,
- Moyen : la cryptographie asymétrique (une clé ayant une partie publique et une partie privée),

# Certificats

- But : authentifier le serveur TLS,
- Moyen : la cryptographie asymétrique,
- La partie publique est signée par une AC (Autorité de Certification), RFC 5280,

# Certificats

- But : authentifier le serveur TLS,
- Moyen : la cryptographie asymétrique,
- La partie publique est signée par une AC (Autorité de Certification), RFC 5280,
- En pratique, une source d'ennuis, de pannes, de frustrations et parfois de dépenses.

# HTTPS

# HTTPS

- Intercaler une couche TLS (chiffrement) entre TCP et HTTP,

# HTTPS

- Intercaler une couche TLS (chiffrement) entre TCP et HTTP,
- RFC 2818

# HTTPS

- Intercaler une couche TLS (chiffrement) entre TCP et HTTP,
- RFC 2818
- Signes de chifrement (cadenas vert),

# HTTPS

- Intercaler une couche TLS (chiffrement) entre TCP et HTTP,
- RFC 2818
- Signes de chifrement (cadenas vert),
- Attention, HTTPS garantit confidentialité et intégrité, pas confiance.

# Plan du tutoriel

# Évolution

# Évolution

- Le premier HTTP, 0.9, n'avait même pas d'en-têtes,

# Évolution

- Le premier HTTP, 0.9, n'avait même pas d'en-têtes,
- HTTP/1 a été le premier normalisé,

# Évolution

- Le premier HTTP, 0.9, n'avait même pas d'en-têtes,
- HTTP/1 a été le premier normalisé,
- HTTP/1.1 a introduit les connexions persistentes,

# Évolution

- Le premier HTTP, 0.9, n'avait même pas d'en-têtes,
- HTTP/1 a été le premier normalisé,
- HTTP/1.1 a introduit les connexions persistentes,
- HTTP/2 passe en encodage binaire,

# Évolution

- Le premier HTTP, 0.9, n'avait même pas d'en-têtes,
- HTTP/1 a été le premier normalisé,
- HTTP/1.1 a introduit les connexions persistentes,
- HTTP/2 passe en encodage binaire,
- HTTP/3 **futur** HTTP au dessus du protocole de transport QUIC (qui intègre TLS)...

# La norme aujourd'hui

- RFC 7230, Message Syntax and Routing
- RFC 7231, Semantics and Content
- RFC 7232, Conditional Requests
- RFC 7233, Range Requests
- RFC 7234, Caching
- RFC 7235, Authentication
- RFC 7540, Hypertext Transfer Protocol version 2

# Conclusion

# Conclusion

- Un succès fou,

# Conclusion

- Un succès fou,
- En partie dû à la simplicité du protocole.