

# Le Protocole HTTP/1.0

---

PeiP2 : Introduction au Web  
Dino López, Benjamin Miraglio et

## 1. Introduction au protocole HTTP

Le protocole HTTP est un protocole relativement simple utilisé pour « servir » des pages Web. Les versions les plus connues du protocole HTTP sont les versions 1.0 et 1.1. Nous n'allons pas voir ce protocole en détail ici et nous nous bornerons aux méthodes GET et POST qui sont les plus utilisées et les plus simples. Par ailleurs nous nous limiterons à la version 1.0 car c'est la plus simple à utiliser lorsque l'on fait des tests avec des outils pas très complexes ou bien à la main.

Une requête du protocole HTTP est formée d'une suite de lignes dont la dernière est une ligne vide. La première ligne contient la commande à exécuter (par exemple la requête/commande GET pour accéder à une ressource telle une page html, un fichier pdf, une image, stockés sur le serveur web). Ensuite, se trouvent des lignes (parfois optionnelles) précisant la requête (le type de fichier que le demandeur sait traiter, les langues connues, le nom de navigateur qui formule la requête, ...). Voici un exemple de requête pour visualiser la page html principale, par exemple, du site [www.i3s.unice.fr](http://www.i3s.unice.fr) ou sinon, du site <http://www.polytech.unice.fr/>

```
GET / HTTP/1.0
Host: www.i3s.unice.fr
User-agent: Fait a la main v0.0
<== ligne vide: on a fini de discuter (la requête est terminée.
Après cette ligne vide, on devra avoir une réponse du serveur)
```

Ici, on déclare que l'on veut accéder à la ressource qui est à la racine du site web du serveur [www.i3s.unice.fr](http://www.i3s.unice.fr) (elle s'appelle "/", ) et que l'on utilise la version 1.0 du protocole HTTP. La ligne Host: indique que l'on veut se connecter sur le serveur [www.i3s.unice.fr](http://www.i3s.unice.fr) (ce qui est déjà le cas si c'est l'adresse qu'on a indiquée dans le navigateur ! mais un serveur Web peut héberger/servir plusieurs sites), et que notre navigateur (browser) Web s'appelle "Fait a la main v0.0"). Ces deux dernières lignes ne sont, en principe, pas obligatoires, mais certains serveurs Web les exigent; il est donc plus prudent qu'elles soient indiquées dans la requête.

Essayons maintenant une connexion sur le serveur Web du site du laboratoire CNRS dont vos enseignants font partie ! Pour cela, et au lieu d'un navigateur web habituel, nous allons utiliser la commande netcat (aussi appelée nc dans le cas Windows) qui est un cat sur le réseau (d'où son nom ;-)), c'est-à-dire qui permet d'envoyer ou recevoir des lignes de caractères au sein d'une session qui s'établit entre un client et un serveur. Un exemple de session vue côté client est donné ci-dessous, où les lignes que vous tapez sur l'entrée standard sont celles émises par votre terminal agissant en tant que client, et où les lignes qui s'affichent sur la sortie standard mais que vous n'avez pas tapées (comme avec un cat) sont celles que le client reçoit en réponse de la part du serveur avec lequel il interagit au sein de la session et au travers du réseau.

```
$ netcat www.i3s.unice.fr 80
GET / HTTP/1.0
Host: www.i3s.unice.fr
User-agent: Fait a la main v0.0
```

<= ligne vide. On a fini notre requête coté client. Sur le même terminal, vont venir s'afficher les lignes correspondant à la réponse émise par le serveur

```
HTTP/1.1 200 OK <= À partir de là, on voit la réponse, constituée
d'une entête, séparée ensuite par une ligne vide, suivi du contenu
(cad. ce qu'attend en pratique le client suite à sa requête de type
GET)
```

```
Date: Wed, 23 Jul 2014 14:26:53 GMT
Server: Apache
Connection: close
Content-Type: text/html
```

```
<HTML>
<META                HTTP-EQUIV="refresh"                CONTENT="0;
URL=http://www.i3s.unice.fr/I3S/">
</HTML>
```

La réponse, comme on le voit, est formée de deux parties (séparées là encore par une ligne vide). La première partie (l'entête) est une suite de lignes qui indiquent comment les choses se sont passées côté serveur (2XX dit que tout va bien, 4XX que la ressource n'a pas été trouvée...) puis des informations diverses comme le nom du serveur, et le type MIME du document (ici, il est dit que le contenu de la réponse est du « text/html »). La seconde partie (le corps) est le document (au format html dans cet exemple) proprement dit. Quand le serveur a fini sa réponse, il ferme la connexion.

Nous avons vu comment se connecter avec la commande netcat à un serveur, mais cette commande permet aussi de se faire passer pour un serveur (de "jouer le rôle de") grâce à son option -l (l pour *listen*). Par exemple, nous pouvons ouvrir un terminal et exécuter la commande :

```
$ netcat -l 1234
```

pour définir un serveur sur la machine locale (notez que c'est forcément la machine sur laquelle on a lancé le terminal dans lequel on a exécuté la commande) qui se met en attente des caractères qui pourraient arriver via le port 1234. Dans un autre terminal, nous pouvons ensuite exécuter la commande :

```
$ netcat localhost 1234
```

pour créer un client qui discute (envoie et reçoit des caractères) avec ce serveur de la machine locale au travers du port 1234.

Ici, nous avons utilisé une seule machine, mais bien-sûr, dans le cas général, nous pourrions utiliser deux machines distinctes (dans ce cas, localhost serait remplacé par une adresse IP). Une fois que le client et le serveur sont lancés, nous avons 2 exécutions de commande netcat qui sont connectées l'une avec l'autre à travers le réseau. Cela veut dire que si l'on tape des caractères dans un terminal, ils apparaissent dans le second, et vice-versa. En

quelque sorte, nous venons donc de nous construire un programme de chat très basique (et ne pouvant avoir seulement que deux interlocuteurs).

## 2. HTTP et les mobiles

L'utilisation des dispositifs mobiles pour l'accès Web est actuellement une pratique courante (et très probablement, vous êtes l'un de ceux qui utilisent le téléphone portable - smartphone- pour accéder au Web :) .

Pour augmenter le confort des utilisateurs des dispositifs mobiles, les développeurs Web adaptent les formats de pages web aux petits écrans grâce à l'utilisation des règles CSS, comme vous l'avez vu dans les exercices d'utilisation d'HTML et CSS.

Cependant, l'utilisation du CSS pour télécharger le bon format demande plusieurs étapes intermédiaires entre le client mobile et le serveur. En effet, dans un premier temps, la page web est téléchargée, ainsi qu'un premier fichier CSS pour déterminer la taille d'écran du client. Par défaut, ce CSS est adapté aux grands écrans (e.g. celui d'un ordinateur portable ou fixe). Si l'écran est inférieur à une certaine taille, une deuxième version CSS adapté pour le mobile est téléchargée (et dans le pire de cas, une autre page HTML aussi). Dans un environnement où l'utilisateur est proche du point d'accès (e.g. l'antenne 4G), les débit de téléchargement devrait être élevé, et le temps de téléchargement des fichiers précédemment décrit, proche de la seconde, et donc, avoir un impacte minimale sur la Qualité de Service (QoS) offert aux utilisateurs. Cependant, un point d'accès très chargé ou éloigné, avec cette méthode, le client voit tout de suite un ralentissement du téléchargement (bas niveau de QoS) et pourrait rapidement abandonner le site web en question.

Pour éviter ces problèmes de QoS, une solution consiste à laisser le serveur HTTP détecter les utilisateurs mobiles pour chaque requête reçue. Si un utilisateur mobile est détecté, le serveur redirige automatiquement l'utilisateur vers la version adaptée du site.

Comment fait donc un serveur Web pour détecter un utilisateur mobile ? Vous l'avez bien deviné : grâce au paramètre "User-agent" émis par le navigateur. Il y a des centaines de versions de navigateurs pour les dispositifs mobiles, cependant, en général, si dans l' User-agent le serveur web lit une ligne contenant le mot Android, iPhone, iPad, iPod, Windows Phone, BlackBerry ou webOS, il est probablement en présence d'un utilisateur mobile.

La détection des utilisateurs mobiles par le biais de l'User-Agent reste cependant peu utilisée.

## 3. POST vs GET

Les principales méthodes définies par le protocole HTTP pour demander au serveur web d'effectuer une action sont les méthodes nommées « GET » et « POST ». On peut trouver plus de détails pour les curieux [ici](#).

Les requêtes GET sont facilement repérables car c'est par défaut ce qui est demandé par le navigateur quand on tape une URL dans la barre d'adresse ! De plus, lorsqu'on a une URL qui

contient le symbole « ? », on sait que la suite est un paramètre d'une requête qui est GET. Plus précisément, la méthode GET est très utile car en plus de demander à récupérer la ressource indiquée dans l'URL, elle permet d'envoyer facilement des informations au serveur via cette URL et/ou ajouter des bookmarks sur une page dynamique. Ci-dessous l'exemple d'une requête (implicitement un GET, de la ressource de nom e1/fullnews.asp disponible sur la machine eftytimes.com) ayant un paramètre (ici de nom edid, et de valeur non typée 117061, ce paramètre servant au serveur à savoir plus précisément quelle section des nouvelles nous intéresse) : `http://efytimes.com/e1/fullnews.asp?edid=117061`

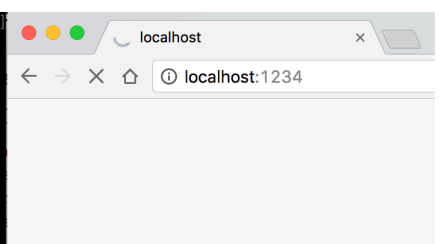
La force de la méthode GET fait partie aussi de ses faiblesses. Par exemple, si on utilise la méthode GET pour envoyer un mot de passe vers un serveur, notre mot de passe sera facilement repérable par un malin qui snifferait les données qui transitent sur le réseau et ainsi vulnérable (également, si l'URL a été tapée dans un navigateur d'une machine d'un lieu public et fait partie de l'historique que n'importe qui pourrait consulter après votre passage !).

La méthode POST pallie à ce problème car les informations envoyées au serveur se trouvent dans le corps de la requête (et non dans l'URL!). De ce fait, si on envoie un mot de passe par la méthode POST, notre mot de passe ne sera pas visible (car pas mis en cache) sur notre navigateur. Si de plus, le protocole HTTPS est utilisé (HTTPS=HTTP Sécurisé), le contenu de la requête envoyée par le navigateur, et donc notre mot de passe, sera chiffré et nos données bien protégées. Pour déclencher une requête POST depuis le navigateur Web, nous verrons comment c'est possible au travers d'un formulaire HTML (balise form)

## 4. Exercices

1. En utilisant netcat en mode serveur dans un terminal sur le port 1234, visualisez la requête HTTP qui est automatiquement envoyée par votre navigateur à ce serveur lorsque vous tapez la bonne URL dans la barre d'adresse. Notez bien qu'on n'a pas un vrai serveur Web, puisque il (le processus netcat que vous avez créé) ne fait que faire un cat de ce qu'il reçoit, sur sa sortie standard, et n'exécute aucune action après analyse des caractères reçus même si ils auraient pu correspondre à une vraie requête GET ou autre !).

```
macbook-pro-de-julien:~ Julien$ nc -l 1234
GET / HTTP/1.1
Host: localhost:1234
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
```



2. Avec la même configuration, répondez alors (mais) à la main (dans le terminal d'où vous avez lancé netcat) à la requête du navigateur et renvoyez-lui une ligne de texte de votre choix. Dans cette réponse vous donnerez comme première ligne la chaîne « HTTP/1.0 200 OK » (pour dire que tout va bien); et vous indiquerez que le mimetype pour le texte qui suit dans le corps est « text/plain ».

```
macbook-pro-de-julien:~ Julien$ nc -l 1234
GET / HTTP/1.1
Host: localhost:1234
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

HTTP/1.0 200 OK
Content-Type: text/plain

Hello world
```

3. En utilisant netcat, téléchargez la page d'URL <http://www.i3s.unice.fr/~lopezpac/>. Il s'agit d'une page HTML simple, standard sans aucune image, et ne nécessitant pas d'envoi de paramètres particuliers au serveur web de l'I3S. Remarquez qu'elle utilise cependant quelques feuilles de style, pourtant celles-ci n'ont pas été téléchargées par votre commande netcat... (et cela aurait été de même pour les images incluses). Déduisez-en quelle est la valeur ajoutée d'un navigateur Web ! Analysez en détail l'entête renvoyé par le serveur web à I3S. Recommencez l'exercice en demandant une page inexistante (ex celle de l'utilisateur toto).

```
macbook-pro-de-julien:~ Julien$ nc www.i3s.unice.fr 80
GET /~lopezpac/ HTTP/1.0
Host: www.i3s.unice.fr
User-agent: Fait a la main v0.0

HTTP/1.1 200 OK
Date: Sun, 27 Nov 2016 14:01:33 GMT
Server: Apache
Connection: close
Content-Type: text/html

<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <meta name="description" content="Your description goes here" />
  <meta name="keywords" content="your,keywords,goes,here" />
  <meta name="author" content="Your Name" />
</head>
</html>
```

```
macbook-pro-de-julien:~ Julien$ nc www.i3s.unice.fr 80
GET /~toto/ HTTP/1.0
Host: www.i3s.unice.fr
User-agent: Fait a la main v0.0

HTTP/1.0 404 Not Found
```

4. En utilisant deux commandes netcat dans deux terminaux différents, copiez un fichier de la machine cliente sur la machine serveur (en supposant pour simplifier qu'il s'agit de la même machine). Cet exercice utilise seulement des redirections pour échanger le contenu du fichier.

```
macbook-pro-de-julien:~ Julien$ nc -l 1234 < /Users/Julien/Desktop/test.txt
macbook-pro-de-julien:~ Julien$
macbook-pro-de-julien:~ Julien$ nc localhost 1234 > /Users/Julien/Desktop/test2.txt
macbook-pro-de-julien:~ Julien$
```

- Exercice expérimental : répétez l'exercice précédent en utilisant 2 machines distantes. Ceci pourrait ne pas marcher car les administrateurs réseau bloquent parfois les flux entre deux machines connectées au même réseau sans-fil, ce par sécurité.

.... A tester sur le modèle de l'exemple précédent en remplaçant localhost par l'adresse de la machine serveur.

- Créez un serveur avec la commande netcat sur le port 1234. Ensuite, téléchargez et éditez le fichier form.html (disponible avec le ressources fournit pour ce TP). Indiquez la méthode GET pour l'envoi de données (où sont envoyées les données d'ailleurs ?) et testez la page avec votre navigateur en y chargeant le fichier html édité (pour tester, ouvrez l'URL file:///xxx/form.html).

```
<form method="POST" action="http://localhost:1234/">
```

The screenshot shows a netcat listener on the left and a web browser on the right. The netcat listener output is as follows:

```
macbook-pro-de-julien:~ Julien$ nc -l 1234
GET /?var=test HTTP/1.1
Host: localhost:1234
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
```

The web browser on the right shows a form with the text "test" in the input field and buttons "Envoyer" and "Annuler". The address bar shows "file:///Users/Julien/...".

- Puis, changez la méthode par POST, enregistrez, rechargez la page sur votre navigateur pour que la modification effectuée soit disponible, et testez à nouveau le formulaire de la page. Commentez : quelle est la différence que vous pouvez observer entre la méthode GET et POST grâce à ce formulaire. Profitez-en aussi pour commenter l'ensemble des lignes que vous avez vu apparaitre dans netcat.

```
<form method="POST" action="http://localhost:1234/">
```

The screenshot shows a netcat listener on the left and a web browser on the right. The netcat listener output is as follows:

```
macbook-pro-de-julien:~ Julien$ nc -l 1234
POST / HTTP/1.1
Host: localhost:1234
Connection: keep-alive
Content-Length: 8
Cache-Control: max-age=0
Origin: null
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

var=test
```

The web browser on the right shows the same form as before, but the netcat listener has received a POST request with the data "var=test".

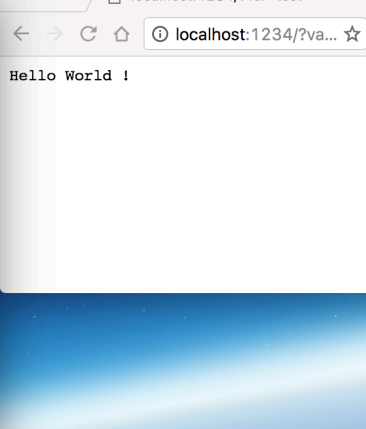
8. Pour aller plus loin dans cet exercice expérimental rigolo, essayez de faire en sorte que le serveur réponde quelque chose au navigateur dans les 2 cas expérimentés (pensez à envoyer un Ctrl-C depuis netcat pour "pousser" la réponse sur le réseau, mais, en même temps, regardez bien votre navigateur avant que l'onglet où était visualisé le formulaire ne se ferme).

```
var=testmacbook-pro-de-julien:~ Julien$ nc -l 1234
GET /?var=test HTTP/1.1
Host: localhost:1234
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

HTTP/1.1 200 OK
Content-Type: text/plain

Hello World !

^C
macbook-pro-de-julien:~ Julien$
```



9. Exécutez 2 fois la commande netcat en mode serveur : l'un pour écouter le port 1234 et l'autre, le port 1235. Depuis votre navigateur firefox, exécutez une requête vers <http://localhost:1234/>. Logiquement, vous devriez avoir une requête sur le premier serveur netcat.

```
macbook-pro-de-julien:~ Julien$ nc -l 1235
Last login: Sun Nov 27 15:18:16 on ttys003
macbook-pro-de-julien:~ Julien$ nc -l 1234
GET / HTTP/1.1
Host: localhost:1234
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
```

Répondez à la requête avec

```
HTTP/1.0 200 OK
```

```
<html>
<head></head>
<body></body>
</html>
```

Vérifiez maintenant le 2ème serveur netcat et expliquez ce comportement.

```
macbook-pro-de-julien:~ Julien$ nc -l 1235
GET /test.jpg HTTP/1.1
Host: localhost:1235
Connection: keep-alive
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36
Accept: image/webp,image/*,*/*;q=0.8
Referer: http://localhost:1234/
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

macbook-pro-de-julien:~ Julien$ nc -l 1234
Last login: Sun Nov 27 15:18:16 on ttys003
macbook-pro-de-julien:~ Julien$ nc -l 1234
GET / HTTP/1.1
Host: localhost:1234
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

HTTP/1.0 200 OK
<html>
<head></head>
<body></body>
</html>
^C
```



En envoyant à la page web un code avec un lien vers une image, le navigateur va automatiquement essayé de la retrouver en envoyant lui même une requete à l'adresse de l'image, ici situé sur le port 1235 de localhost, ainsi on voit cette requête apparaître sur le serveur qui écoute le port 1235.

10. Dernier exercice : testez la présence d'un mécanisme de détection User-agent chez tripadvisor. Pour cela, exécutez une fois la commande

```
$ wget www.tripadvisor.com
```

Ceci créera un fichier index.html. Ensuite, exécutez la commande

```
$ wget www.tripadvisor.com --user-agent="Mozilla/5.0 (Linux; U; Android 4.0.3; ko-kr; LG-L160L Build/IML74K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30"
```

afin de faire croire au serveur que nous utilisons un dispositif mobile. Ceci créera un fichier index.html.1. Commentez les différences entre ces 2 fichiers HTML téléchargées (i.e. index.html et index.html.1).

On constate que le premier fichier correspond à une page trip advisor ordinateur classique. Le deuxième à une version mobile. On a une apparition du javascript plus tôt dans le premier fichier et des propriétés css différentes. La page mobile est plus légère également.

Dans la page normale on a

```
<link      rel='stylesheet'      type='text/css'      media='screen,      print'
href='https://static.tacdn.com/css2/home_2015-v23113832244a.css'      data-
rup='home_2015'/>
```

Alors que le fichier css de l'autre est :

```
<link      rel='stylesheet'      type='text/css'      media='screen,      handheld'
href='//static.tacdn.com/css2/mobile2012-rollup-android-v21282158048a.css' data-
rup='mobile2012-rollup-android'/>
```