

Rapport du PROJET de RSA

Programmation d'un serveur proxy

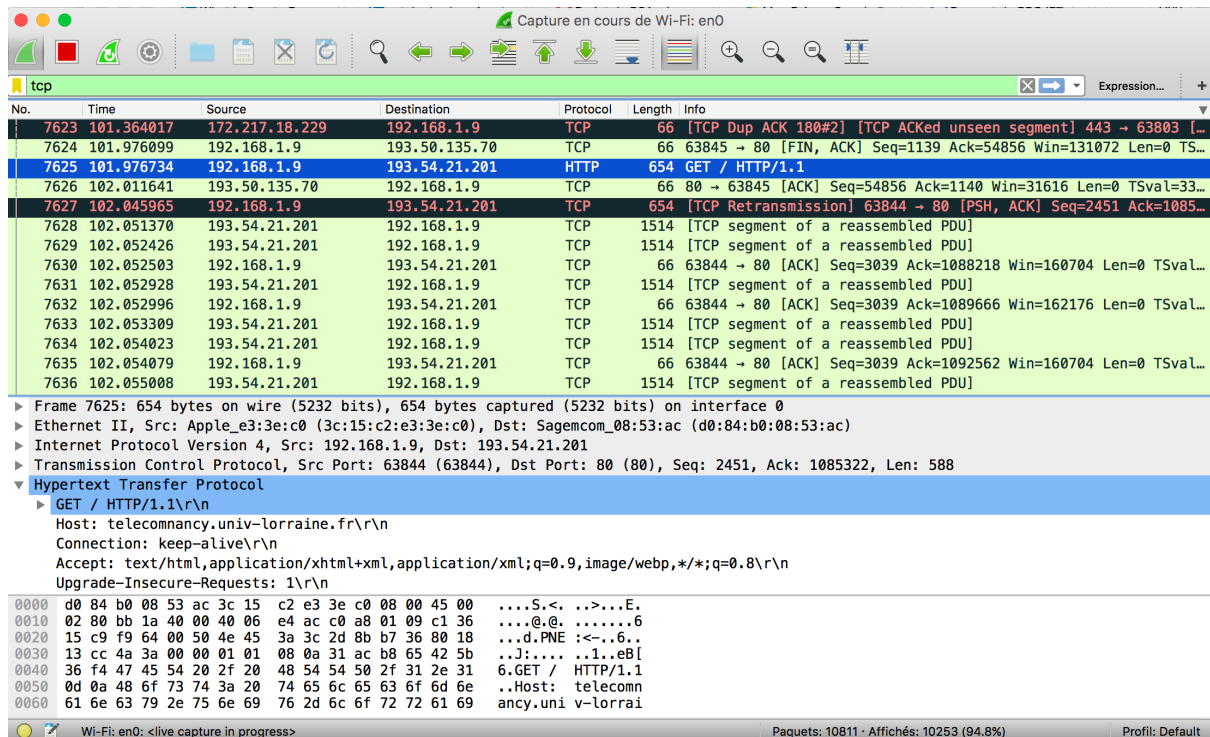
*Rédigé par
Jean-Baptiste DOMINGUEZ, Henry*

Table des matières

Question 1	3
Question 2	4
Question 3	5
Question 4	5
Question 5	6
Question 6	6

Question 1

A l'aide du logiciel Wireshark nous avons analysé les échanges entre notre machine et un serveur web. Voici la capture d'écran que nous obtenons.



C'est le paquet n°7625 qui nous intéresse. Nous observons que l'adresse IP de destination du serveur web à l'adresse `http.host == "telecomnancy.univ-lorraine.fr"` est 193.54.21.201 et que notre adresse IP est 192.168.1.9 Pour en être sûr nous utilisons la commande `ifconfig` dans le terminal

```
henryung — -bash — 80x24
bridge0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
options=63<RXCSUM,TXCSUM,TS04,TS06>
ether 3e:15:c2:3e:23:00
Configuration:
    id 0:0:0:0:0:0 priority 0 hellotime 0 fwddelay 0
    maxage 0 holdcnt 0 proto stp maxaddr 100 timeout 1200
    root id 0:0:0:0:0:0 priority 0 ifcost 0 port 0
    ipfilter disabled flags 0x2
member: en1 flags=3<LEARNING,DISCOVER>
    ifmaxaddr 0 port 5 priority 0 path cost 0
member: en2 flags=3<LEARNING,DISCOVER>
    ifmaxaddr 0 port 6 priority 0 path cost 0
nd6 options=1<PERFORMNUD>
media: <unknown type>
status: inactive
[MBP-de-Henry:~ henryung$ ifconfig en0
en0: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
ether 3c:15:c2:e3:3e:c0
inet6 fe80::3e15:c2ff:fe3:3ec0%en0 prefixlen 64 scopeid 0x4
inet 192.168.1.9 netmask 0xfffff00 broadcast 192.168.1.255
nd6 options=1<PERFORMNUD>
media: autoselect
status: active
MBP-de-Henry:~ henryung$ ]
```

A la ligne inet on observe que notre adresse IP est bien 192.168.1.9 nous utilisons le netmask 0xffffffff00 et notre adresse de broadcast est 192.168.1.255

Afin de mieux voir ce qui nous intéresse nous utilisons le filtre “ip.addr == 193.54.21.201” qui nous permet de n’afficher que la “discussion” entre notre client et le serveur web de telecomnancy.eu

Pour accéder au site on fait une requête HTTP d’une longueur de 654 bits : “GET / HTTP/1.1” on utilise l’Internet Protocol Version 4 (IPv4) et la Transmission Control Protocol (TCP) afin d’établir une connexion et garantir la bonne transmission des données.

On effectue une demande de connexion ainsi le serveur web et le client s’échange des données lorsque la requête GET est satisfaite on reçoit : “HTTP/1.1 200 OK (text/html)”.

Wireshark capture showing a TCP connection to 193.54.21.201. The packet list shows a three-way handshake (7641 SYN, 7642 ACK, 7643 FIN) and subsequent data exchange. Packet 7649 is a GET request, and packet 7650 is the corresponding 200 OK response. The packet details for packet 7650 show the HTTP response structure.

No.	Time	Source	Destination	Protocol	Length	Info
7640	102.056188	193.54.21.201	192.168.1.9	HTTP	71	HTTP/1.1 200 OK (text/html)
7641	102.056233	192.168.1.9	193.54.21.201	TCP	66	63844 → 80 [ACK] Seq=3039 Ack=1095354 Win=162144 Len=0 TSval=...
7642	102.085942	193.54.21.201	192.168.1.9	TCP	78	[TCP Dup ACK 7628#1] 80 → 63844 [ACK] Seq=1095354 Ack=3039 W...
7643	102.092621	104.244.43.140	192.168.1.9	TCP	66	443 → 63846 [FIN, ACK] Seq=13651 Ack=768 Win=16896 Len=0 TSv...
7644	102.092690	192.168.1.9	104.244.43.140	TCP	66	63846 → 443 [ACK] Seq=768 Ack=13652 Win=131072 Len=0 TSval=8...
7645	102.109161	192.168.1.9	193.54.21.201	TCP	78	63849 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=...
7646	102.109740	192.168.1.9	193.54.21.201	TCP	78	63850 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=...
7647	102.133814	192.168.1.9	193.54.21.201	HTTP	623	GET /sites/all/modules/lightbox2/js/lightbox.js?1463380529 H...
7648	102.141957	192.168.1.9	193.54.21.201	HTTP	630	GET /%E2%00%9Dhttp://code.jquery.com/jquery-1.5.2.min.js%E2%
7649	102.144716	193.54.21.201	192.168.1.9	TCP	74	80 → 63849 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 S...
7650	102.144796	192.168.1.9	193.54.21.201	TCP	66	63849 → 80 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval=83340314...
7651	102.145291	193.54.21.201	192.168.1.9	TCP	74	80 → 63850 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 S...
7652	102.145342	192.168.1.9	193.54.21.201	TCP	66	63850 → 80 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval=83340314...

Packet details for packet 7650 (HTTP response):

- Next sequence number: 1095354 (relative sequence number)
- Acknowledgment number: 3039 (relative ack number)
- Header Length: 32 bytes
- Flags: 0x018 (PSH, ACK)
- Window size value: 164
- [Calculated window size: 20992]
- [Window size scaling factor: 128]
- Checksum: 0x8e84 [validation disabled]
- Urgent pointer: 0

Frame (71 bytes) | Reassembled TCP (10032 bytes) | De-chunked entity body (9462 bytes) | Uncompressed entity body (46727 bytes)

Paquets: 31449 - Affichés: 30218 (96.1%) - Markés: 1 (0.0%) - Profil: Default

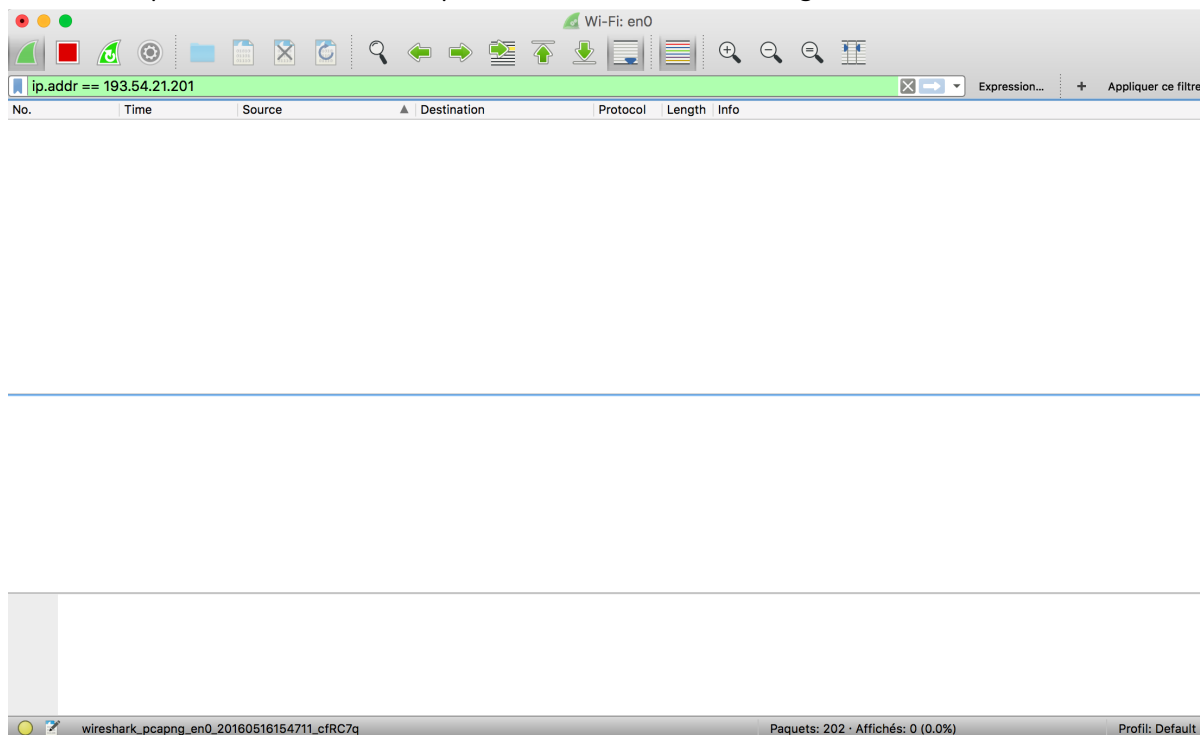
On cherche ensuite le three-way handshake qui caractérise TCP et qui garantie que la connexion a bien été effectué. On trouve les réponses respectives des paquets [SYN] numérotés 7645 et 7646 au paquet n°7649 [SYN, ACK] et n°7650 [ACK] ainsi qu’au paquet n° 7651 [SYN, ACK] et n° 7652 [ACK]. Avant on observe que deux requêtes HTTP “GET” ont été effectuées pour récupérer des données. Globalement, plus d’un milliers de paquets on été échangés entre le client et le serveur web. Le dernier paquet portant le numéro 8181.

Question 2

Pour configurer notre machine afin d’utiliser le serveur proxy que nous avons programmé nous allons dans les paramètres réseaux. Nous entrons l’adresse et le numéro de port adéquat. Soit 127.0.0.1 pour l’adresse IP et 1330 port le numéro de port.

Question 3

Comme le serveur proxy n'est pas configuré nous nous attendons à voir aucun échange entre notre machine et le serveur web. Ainsi l'essai de connexion entre le client et le serveur est voué à échouer. En effet une capture d'écran confirme qu'aucune donnée n'est échangée.



Nous observons alors une page blanche. Cela paraît banal mais montre qu'une "simple" configuration de proxy HTTP permet de contrôler la connexion aux différents sites web présent sur Internet. Nous allons maintenant concevoir et implanter un serveur proxy afin de gérer d'abord un client et ensuite plusieurs client.

Question 4

La communication entre un client et un serveur en passant par un proxy est représentable de la manière suivante.

Client >>>> Proxy >>>> Serveur web >>>> Proxy >>>>> Client

L'algorithme général du proxy est le suivant : on attend qu'un client fasse une requête HTTP. Lorsque le serveur proxy reçoit une requête il envoie cette requête au serveur web. Ensuite, il attend de recevoir la réponse du serveur web qu'il transmet par la suite au client. On répète cette procédure jusqu'à ce que le client n'ait plus de requêtes et met un terme à la communication.

Question 5

Nous avons implémenté un proxy qui permet de gérer les requêtes d'un client. Puis nous avons amélioré notre programme afin qu'il gère simultanément plusieurs clients.

Question 6

Pour la gestion simultanée de plusieurs clients nous avons utilisé plusieurs threads. Nous avons fait ce choix en nous interrogeant sur plusieurs critères. Nous connaissons en tout trois manières de gérer des requêtes simultanées. La première que nous avons étudié en TD consiste à créer une liste et de servir les clients au fur et à mesure avec les "raw sockets". Cela ne respecte pas ce qui est demandé donc nous avons écarté cette possibilité. En cours, nous avons étudié les processus dupliqués grâce à l'instruction `fork()`. L'utilisation de cette instruction présente un avantage principal, il est facile à implanter par rapport à la troisième alternative que nous connaissons : le multi-threading. Malheureusement, l'utilisation de l'instruction `fork()` est gourmande en ressource car elle duplique intégralement le programme. C'est donc par souci d'économie de ressources que nous avons choisi d'implanter un serveur proxy qui gère plusieurs clients à la fois en utilisant le multi-threading.