Année Universitaire 2024-2025 session 1 d'automne

Parcours : TAH Licence pro ADSILLH **UE :** 4TAH506U
Épreuve : Examen réseaux
Date : 22 novembre **Heure :** 14h00 **Durée :** 1h30
Modalités : Épreuve sans document, calculatrice autorisée
Épreuve de : Mr Samuel Thibault 2 **pages**

université de BORDEAUX

Collège Sciences et technologie

Exercice 1. Questions / discussions de cours

Q1.1 127.0.0.1, c'est l'adresse locale de ma machine, ce n'est pas là que l'on va trouver le site web :) Effectivement, sur le serveur web lui-même, 127.0.0.1 va mener au site web, mais c'est seulement sur cette machine-là

Même si 127.0.0.1 pointait effectivement vers un serveur web qui sert le site web, sans fournir le nom de domaine visé, le serveur web aura du mal à savoir quel site web servir s'il héberge plusieurs sites webs (comme c'est essentiellement toujours le cas de nos jours)

Q1.2 Dans le cadre du protocole HTTP lui-même, on se connecte au serveur web visé et récupère la page web. Il n'y a pas de "centre" du web, on peut se connecter directement à n'importe quel serveur web sans avoir à consulter une instance centrale ou décentralisée.

Dans le cadre du protocole DNS, il y a effectivement un centre, c'est la racine des noms de domaines, hébergée par les root servers DNS bien connus. C'est tout de même décentralisé, au sens où les root servers ne connaissent qu'une partie de la réponse : les TLD (top-level domains). Le détails des noms de domaines existant, c'est hébergé par les différents serveur DNS faisant autorité pour chaque TLD et en-dessous.

Q1.3 Quand la machine 10.0.1.2 envoie son ping, 10.0.2.2 le reçoit peut-être bien, mais quand 10.0.2.2 veut renvoyer sa réponse à 10.0.1.2, puisqu'il est configuré avec le réseau 10.0.2.2/16, il croit à tort que 10.0.1.2 est dans le même réseau que lui ! Et il envoie donc un ARP pour demander quelle machine possède l'IP 10.0.1.2. Mais puisqu'ils sont dans des vlans différents, 10.0.1.2 ne peut pas recevoir cet ARP et y répondre. 10.0.2.2 reste donc sans réponse pour savoir vers quelle adresse MAC envoyer sa réponse au ping.

Q1.4 Une socket UDP est sans connexion. La socket d'"écoute" est aussi celle de réception des données, il n'y a pas de sens d'appeler accept pour attendre la connexion d'un client : on appelle simplement recvfrom() pour recevoir une requête (et savoir d'où elle vient), et on appelle sendto() pour y répondre.

Q1.5 connect() est cependant utile en UDP : cela configure simplement l'adresse distante de la socket. On pourra alors utiliser simplement send() au lieu de sendto(), l'adresse destination étant déjà spécifiée par l'appel connect().

Exercice 2. Adresses

Q2.1 Il y a 80.67.2.1 à 80.67.2.255 et 80.67.3.0 à 80.67.3.254 Il y en a 510.

Q2.2 $1500 > 512$, mais $1500 < 2048 = 2^{11}$, donc 80.67.0.0/21 c'est assez large (de 80.67.0.0 à 80.67.7.255)

On aurait pu essayer de répondre 80.67.0.1/21. Ce n'est pas correct puisque le .1 n'est pas inclut dans la partie réseau de /21. Mais c'est déjà bien d'avoir correctement calculé /21.

Q2.3 On peut simplement numéroter 2001 :910 :wxyz : :/48

Q2.4 Il y a 128-48 = 80 bits pour la partie machine, donc 2^{80} adresses IPv6 (i.e. un million de milliards de milliards :))

Exercice 3. Calculs (on pourra faire des arrondis de calculs grossiers si l'on n'a pas de calculatrice). Précisez le déroulement de votre calcul.

Q3.1 3Go = 24Gb, à 1Gb/s il faut donc 24s

Q3.2 S'il y a plusieurs postes qui s'allument en même temps, le lien 1Gbps avec le serveur va être partagé entre les différents transferts, cela divise d'autant la bande passante disponible et multiplie donc d'autant le temps

Q3.3 3Go, cela fait 2 millions de paquets de 1500o. Avec une latence A/R de 100µs pour chaque paquet, on se retrouve à attendre 200s !

Exercice 4.

Q4.1 10.0.4.7

Q4.2 0x28 = 40, il n'y a pas de données applicatives !

Q4.3 0x17 = 23, 0x405 = 1029

Q4.4 C'est l'émetteur qui a un numéro de port inférieur à 1024, donc c'est très probablement lui le serveur.

Exercice 5.

Q5.1

```
client ---    SYN   --> serveur
       <--  SYN+ACK ---
       -->    ACK   -->
       -->  GET+GET -->
       <-- HTTP/1.1 200 + fichier1 ---
       <-- fichier1 ---
       <-- fichier1 ---
       <-- fichier1 + HTTP/1.1 200 + fichier2 ---
       <-- fichier2 ---
       <-- fichier2 ---  (et des ACK du client)
       <--    FIN   ---
       ---  FIN+ACK -->
       <--    ACK   ---
```

Q5.2 On gagne le temps du three-way handshake de TCP, donc quelques A/R de latence, on peut faire un seul paquet avec les différents GET. Avec https, on économise en plus le temps de négociation TLS, donc non seulement encore plus de temps avec des A/R de latence, mais aussi du temps CPU serveur pour répondre au challenge, générer une clé de session, etc.

Q5.3 Le client a besoin de savoir où se situe la coupure entre les différents fichiers. Il faut donc que le serveur indique dans sa réponse HTTP la taille des fichiers pour savoir où découper.

Q5.4 si le client HTTP/1.1 commence par demander le gros fond de page au serveur avant de demander les petites images, il va devoir attendre que le serveur ait fini d'envoyer le fond de page (pas vraiment utile) avant de pouvoir recevoir les petites images (utiles). Avec plusieurs connexions HTTP/1.0 en parallèle, celle pour le fond de page peut prendre du temps, ce n'est pas gênant, les connexions pour les petites images peuvent progresser et fournir les petites images rapidement. Pour avoir les petites images rapidement en HTTP/1.1 avec une seule connexion, il faudrait bricoler le html pour mettre en premier les références aux petites images...

Q5.5 Cela permet de le déployer en espace utilisateur dans les navigateurs web, plutôt qu'avoir à modifier les systèmes d'exploitation de tous les ordinateurs de la terre. Et surtout, cela évite d'avoir à apprendre aux firewalls etc. comment faire le masquerading de ce nouveau protocole. Bref, cela permet d'éviter l'"ossification des protocoles"