TP 1 : Introduction au Réseau — HTTP

Guillaume Didier

05/11/2024, à rendre le 08/11/2024

1 Théorie

1.1 Concepts d'internet

Question 1

Quelle différence faites-vous entre un client et un serveur ? *Bonus : Un même processus peut-il agir en tant que client et serveur simultanément ?*

Question 2

Quelle différence y a-t-il en principe entre les nœuds extrémités d'un traffic réseau et les nœuds intermédiaires ?

1.2 Latence et Bande-passante

Question 3

Rappeler la définition de la latence et de la bande passante.

Question 4

Dans le cadre de l'Event Horizon Telescope, qui a permis de produire la première image astronomique d'un trou noir, les chercheurs ont dû centraliser près 5 pétaoctets de données (5×10^{15} octets) depuis 9 télescopes différents.

On estime que les chercheurs ont transféré près de 700 téraoctets depuis Hawaï vers le MIT.

Il faut 14h à un chercheur pour rejoindre le MIT (avec un colis de 100 kg de disques durs). Quelle est la latence et la bande passante de ce transfert ?

Question 5

En supposant que l'observatoire dispose d'un accès fibre à 10 Gigabits par seconde, et qu'il faut 75 ms pour transmettre un paquet du télescope au laboratoire du MIT, combien de temps au total faut-il pour transmettre les 700 To ?

Question 6

Ma connection internet permet un débit montant d'au plus $2\,\text{Gbit}\,\text{s}^{-1}$.

Comparer ce débit à celui obtenu avec un camion $9\,\text{m}^3$, d'une charge utile maximale limitée à $1600\,\text{kg}$, rempli de cartes SD, pour transporter des données de Beaulieu au centre de recherche Eurecom à Sophia-Antipolis. *On pourra effectuer les calculs avec un seul chiffre significatif.*

À partir de quel volume de données est-il préférable d'employer cette méthode ?

2 Pratique

Pour cette partie, vous devrez récupérer le code source fourni via `https://autolab.guillaumedidier.fr/`, et soumettre vos solutions dessus, afin d'effectuer une relecture de votre code. Remarque, ce TP n'est pas noté, mais familiarisez-vous avec la plateforme qui sera utilisée pour le TP noté par la suite. Un Makefile est déjà fourni pour compiler les fichiers sources des trois exercices (dont un optionnel), et les lier avec la bibliothèque CS:APP.

2.1 Récupérer une page web en HTTP

Le premier exercice est d'effectuer une requête HTTP 1.0 afin de récupérer une page web. Le programme `http-client` prend comme unique argument une URL http, comme décrit dans le premier amphi. Vous pouvez aussi vous référer à la RFC 1945[1]. Celle-ci se compose du protocole (`http`), suivi de `://`, puis d'un domaine (on supportera les IPv4, mais pas les IPv6, non décrite en HTTP 1.0), de façon optionnelle, un numéro de port, précédé d'un `:`, puis d'un chemin, commençant par `/`, et un fragment, précédé de `#`.

Il vous faut tout d'abord extraire le domaine, et le port éventuel, et ouvrir une connection TCP vers l'hôte ainsi précisé. Ensuite, vous devez envoyer une requête HTTP 1.0, composé d'une ligne de requête `GET <url complète> HTTP/1.0`, de zéro ou plusieurs headers, puis d'une ligne vide. Les lignes doivent être terminé par un `CRLF` (`\r\n`).

Enfin, vous devez lire la réponse, composé d'une ligne de status, zéro ou plusieurs Headers, d'une ligne vide et de la réponse. Vous afficherez le statut et les Headers sur stderr et la réponse elle-même sur stdout.

Le Header `Connection: close` pourrait vous être utile.

Le code de retour de votre programme doit refléter le statut HTTP. Un code signalant un succès doit être renvoyé pour les statuts 200, un code signalant une erreur pour les codes 400 et 500.

Enfin, un autre code doit refléter un usage incorrect ou une erreur interne.

Le support des redirections (statut 30x) est optionnel. Si vous l'implémentez, prenez gardes aux boucles de redirections, et aux redirections en HTTPS.

Autrement, considérez un code 300 comme un succès.

Il est possible d'utiliser les commandes `nc -c`*, ou* `telnet` *pour ouvrir une connection TCP et faire des requêtes à la main*

Le site du cours est disponible en HTTP sur le port 8192 (le port 80 vous redirigera en HTTPS) : `http://sys1.guillaumedidier.fr:8192/`.

2.2 Un serveur qui dit bonjour — Hello Server

Dans cet exercice, vous allez écrire votre premier serveur. Celui-ci doit écouter sur le port 18519[2]. Lorsqu'un client se connecte, il envoie une séquence de noms de personne à saluer, une personne par ligne terminée par CRLF (`"\r\n"`). Le serveur répond à chaque ligne par une ligne de salutation de votre choix (par exemple `"Bonjour Martin Quinson =)\r\n"`), en réponse à `"Martin Quinson\r\n"`. L'échange se termine lorsque le client coupe la connection, ou qu'une ligne vide est rencontrée.

Toute salutation de votre serveur doit se terminer par une fin de ligne CRLF.

Pour tester votre serveur vous pouvez utiliser netcat (`nc -c`), ou telnet.

Votre serveur doit pouvoir gérer plusieurs clients les uns à la suite des autres, mais il n'est pas nécessaire d'implémenter un serveur employant du multiplexage ou de la concurrence pour gérer plusieurs clients simultanément.

2.3 Bonus : Un client pour votre serveur — Hello Client

Vous pouvez implémenter un programme C lisant des listes de noms depuis la console ou un fichier, et contactant le serveur de votre choix.

1. `https://datatracker.ietf.org/doc/html/rfc1945`
2. `0x4857`, ce qui correspond à l'encodage ascii de `HW`

2.4 Rendu

Vous devez rendre vos trois fichier source C, (et uniquement eux). Pour ce faire, exécutez `make submit` pour générer le fichier `handin.tar` que vous soumettrez sur Autolab. Le système de notation automatique doit compiler automatiquement vos fichiers et vous donnera un score de 2 points par fichier compilant avec succès. (Ce score ne rentre pas dans votre note de module, c'est juste une façon de vérifier le bon fonctionnement et de vous familiariser avec ce système) Après la date de rendu, l'équipe du cours effectuera une relecture de votre code, dont vous pourrez consulter les commentaires sur Autolab.

Une note de Style sur 10, tout aussi indicative vous sera donnée. Celle-ci ne rentre toujours pas en compte dans votre note de module, mais le style représentera 50% de votre note au TP noté, soit 20% de votre note pour la partie réseau.

2.5 La bibliothèque rio

Outre les fonctions `open_clientfd` et `open_listenfd`, la bibliothèque CS:APP comprend une bibliothèque d'entrées-sorties fiables, appelée `rio`. Cette bibliothèque permet de s'épargner la gestion des *short-count* qui peuvent se produire avec les appels systèmes unix `read` et `write`, lorsque ces fonctions traitent un nombre d'octets non-maximal. La bibliothèque `rio` répète ces appels lorsque c'est nécessaire, mais cela ne vous dispense pas de gérer les erreurs.

Elle comprend une fonction d'écriture, `rio_writen`, une fonction de lecture directe `rio_readn`, et des fonctions de lecture bufferisée, qui manipulent un type `rio_t`, et dont le nom se termine par `b`. Pour lire une ligne entière, la fonction `rio_readlineb` vous sera utile. La fonction `rio_readinitb` permet d'initialiser la structure `rio_t` nécessaire, et `rio_readnb` permet de lire un nombre fixé d'octet.

Il ne faut pas mélanger `rio_readn`*, non-bufferisée avec les fonctions bufferisées.*