

TP n° 8

Consignes les exercices ou questions marqués d'un ★ devront être rédigés sur papier (afin de se préparer aux épreuves écrites du partiel et de l'examen). En particulier, il est recommandé d'être dans les mêmes conditions qu'en examen : pas de document ni de calculatrice. Les questions marquées d'un ◇ sont des questions supplémentaires permettant d'aller plus loin (et ne seront pas forcément corrigées en TP). Tous les TPs se font sous Linux.

But du TP

Le but de cette feuille d'exercice est :

- familiarisation avec les concepts de base du réseau
- découverte des commandes de base
- utilisation de Python pour écrire un programme communiquer par une socket TCP

Exercices

Exercice 0

Ouvrir un terminal :

- créer un répertoire **IntroInfo** et un répertoire **TP8** à l'intérieur. Note : les fichiers des TPs précédents ne sont pas accessibles directement depuis cet environnement.
- se placer à l'intérieur du répertoire **TP8**

On suppose pour les autres exercices que le répertoire **TP8** est le répertoire courant.

Exercice 1

★ Pour chaque question indiquer les affirmations correctes (sans justifier). Il peut y avoir plusieurs ou aucune affirmation correcte.

1. Le protocole IP :
 - (a) se situe au même niveau que le protocole Ethernet.
 - (b) permet d'attribuer à chaque périphérique réseau une adresse.
 - (c) garantit la bonne distribution des données.
2. Étant donné le masque de sous réseau 255.255.192.0, on peut représenter
 - (a) 1024 adresses du même sous-réseau
 - (b) 8192 adresses du même sous réseau
 - (c) 16384 adresses du même sous-réseau
3. On considère la façon dont est programmée un navigateur Web. Lorsqu'un navigateur Web se connecte au site **www.universite-paris-saclay.fr** il doit :
 - (a) ouvrir une connexion TCP vers la machine **www.universite-paris-saclay.fr** sur un port donné
 - (b) effectuer une requête DNS pour obtenir l'adresse IP correspondante
 - (c) déterminer la route lui permettant d'atteindre la machine **www.universite-paris-saclay.fr**
4. Soit la table de routage suivante (simplifiée) d'un routeur, donnée ci-dessous :

Destination	Gateway	Genmask	... Iface
192.168.0.0	0.0.0.0	255.255.248.0	... eth0
129.175.20.0	0.0.0.0	255.255.255.0	... eth1
default	129.175.20.1	0.0.0.0	... eth1

- (a) le routeur possède trois cartes réseau configurées.
- (b) les paquets à destination de l'IP 8.8.8.8 passent par la carte **eth1**.
- (c) les paquets à destination de l'IP 192.168.1.42 passent par la carte **eth0**.
- (d) les paquets à destination de l'IP 192.168.42.1 passent par la carte **eth0**.

On rappelle la règle de calcul. Si R est une adresse réseau (ici les adresses de la colonne **Destination**) et M une adresse de masque (ici dans la colonne **Genmask**), alors pour une adresse IP quelconque I , I appartient au sous-réseau R si et seulement si : $R = I \& M$ où $\&$ représente l'opération « et » bit à bit. Il faut donc convertir chaque composante des adresses en base 2 et faire le « et » bit à bit.

5. Sur un réseau IP (comme Internet) :
- (a) les paquets IPs peuvent se perdre (ne pas arriver à destination)
 - (b) les paquets IPs peuvent arriver dans le désordre
 - (c) les paquets IPs prennent toujours la même route

Réponse:

1. Correct : (b). Ethernet est un protocole de la couche de liaison donc « en dessous » d'IP (plus bas niveau). IP ne garantit pas non plus la bonne distribution des données, c'est le rôle de TCP (au dessus).
2. Correct : (c). Rappel : le nombre de bits à 0 dans le masque indique le nombre de bits pouvant être utilisés pour identifier les machines du sous-réseau. Le masque 255.255.192.0 correspond (en binaire) à 11111111.11111111.11000000.00000000₂ soit 14 bits à zéro. On peut représenter $2^{14} = 16384$ adresses du même sous-réseau.
3. Correct : (a) et (b). Le navigateur Web effectuera d'ailleurs (b) en premier, puis connaissant l'adresse IP pourra initier une connexion vers cette adresse, en utilisant le protocole TCP (sur le port 80 pour HTTP ou 443 pour HTTPS). La route est calculée dans le cadre du protocole IP, les applications n'ont pas à la déterminer manuellement.
4. Correct : (b) et (c). Le routeur possède ici deux cartes réseaux configurées : **eth0** et **eth1**. L'IP 8.8.8.8 ne faisant partie d'aucun des deux sous-réseaux explicites configurés pour **eth0** et **eth1**, elle sera envoyée sur la carte **eth1** pour prendre la route par défaut. Les paquets seront tous envoyés à la passerelle 129.175.20.1 qui se chargera de les envoyer (de proche en proche) à la destination finale. L'IP 192.168.1.42 appartient bien au sous-réseau 192.168.0.0/255.255.248.0 (i.e. si on fait le « et » bits à bits entre l'IP et le masque et entre l'adresse réseau et le masque on obtient la même valeur : 192.168.0.0. En effet le masque correspond à 11111111.11111111.11111000.00000000,

adresse	192.168.1.42
masque	255.255.248.0
& bit à bit	192.168.0.0

d'où : Autrement dit, seront sur le même sous réseau, toutes les adresses entre 192.168.0.0 et 192.168.7.255. Ici le 7 est calculé car on peut utiliser 3 bits sur le troisième octet 11111000 et 255 car on peut utiliser tous les bits du 4ème octet : 00000000.

Pour l'IP 192.168.42.1 :

adresse	192.168.42.1
masque	255.255.248.0
& bit à bit	192.168.40.0

Sur le troisième chiffre : $42_{10} = 00101010_2$ et $248_{10} = 11111000_2$.
 $00101010_2 \& 11111000_2 = 00101000_2 = 40_{10}$. L'IP 192.168.40.0 est différente de 192.168.0.0. Les paquets vont donc aller sur **eth1** via la route par défaut (et probablement ne pas avoir de destination correcte une fois arrivés au routeur car ce sont des IPs réservées à un réseau local).

5. Correct : (a) et (b). Les paquets peuvent prendre des routes différentes pour arriver à la même destination (par exemple si un routeur devient indisponible entre la source et la destination).

Utilisation de Jupyterhub

Les exercices 2 (et 3) font appel à des commandes réseaux et ne fonctionnent que si certains paquets de tests peuvent circuler librement sur le réseau. Cependant, les salles machines du bâtiment 336 ont une configuration réseau relativement « verrouillée », qui rendent les résultats de certains exercices aléatoires.

Nous utilisons donc pour les exercices 2 et 3 le service `jupyterhub` qui est aussi utilisé dans l'UE *Intro à la programmation Impérative*.

Naviguer vers le site :

<https://jupyterhub.ijclab.in2p3.fr/>

- se connecter en utilisant son identifiant Paris-Saclay
- (si le site vous propose un bouton **Start my Server**, cliquer dessus, sinon votre serveur est déjà démarré)
- ouvrir un terminal avec **New** et **Terminal**. Vous pouvez utiliser ce terminal nouvellement créé pour entrer les commandes de l'exercice 3.

Attention, les fichiers créés sur le compte JupyterHub ne sont pas directement visible dans vos comptes des salles machines et inversement. Il convient donc de les récupérer et les déposer au bons endroits.

Exercice 2

Le but de cet exercice est de s'initier à quelques commandes réseau de base afin d'illustrer certains concepts vus en cours.

ifconfig cette commande affiche les interfaces réseau, leur adresse IP et leur adresse Ethernet (adresse MAC) si elles en ont une.

route cette commande affiche la table de routage de la machine. Dans le cas simple, une machine n'a qu'une seule interface configurée (en plus de l'interface fictive associée à 127.0.0.1). Cette interface configurée, **eth0**, permet accéder directement au réseau local ainsi qu'au reste du monde si une route par défaut est configurée. Dans ce cas, l'adresse du routeur permettant d'accéder au reste est donné dans la colonne Gateway (passerelle).

ping *m* cette commande permet de tester la connectivité au réseau. Elle utilise un protocole auxiliaire au protocole IP (le protocole ICMP) et envoie des petits paquets spéciaux à la machine *m*. *m* peut être une adresse IP ou un nom de machine. Dans ce protocole, la machine de destination répond à l'adresse source avec un paquet similaire d'accusé de réception. Si un accusé de réception ne revient pas, le paquet est considéré comme perdu.

traceroute *m* ce commande fonctionne comme **ping** mais envoie des paquets ayant une durée de vie de plus en plus longue. La durée de vie d'un paquet est le nombre de routeur qu'il a le droit de traverser. À chaque franchissement d'un routeur, la durée de vie est décrémentée. Lorsque un routeur rencontre un paquet avec une durée de vie à 0, il le détruit et envoie une notification à l'adresse source.

1. Donner l'adresse IP de votre machine ainsi que la passerelle associée à la route par défaut. Quel est le masque du sous-réseau et combien d'adresses se trouvent sur ce sous-réseau ?

Réponse:

2. les commandes **ifconfig** et **route** donnent les réponses. L'adresse IP doit être en 172.17.*x.y*. La passerelle est 172.17.0.1. Le masque est 255.255.0.0 (visible dans **ifconfig** ou **route**). On a 2^{16} ou 65536 adresses.

3. Utiliser la commande **traceroute** pour rechercher la route (*i.e.* les adresses des routeurs) pour atteindre la machine **www.google.com**¹. Combien de routeurs sont traversés ?

Réponse: **traceroute www.google.com** indique le nombre de routeur. Le premier est sans surprise 172.17.0.1. Lors de mon test, il y a 10 routeurs traversés (la 11^{ème} IP est l'IP de destination).

4. Les paquets prennent-ils toujours la même route ? Comment pouvez vous vérifier cette hypothèse ?

Réponse: En appelant **traceroute www.google.com** plusieurs fois de suite, on peut se rendre compte que certains routeurs intermédiaires ne sont pas les mêmes tout le temps.

1. Le but du TP n'est pas de faire de la publicité pour cette entreprise, mais la manière dont le réseau est configuré chez Google en particulier permet de faire des observations intéressantes, que l'on ne peut pas faire avec un site trop local (hébergé en France par exemple) ou ne disposant pas d'une grande infrastructure réseau.

5. On peut passer à la commande `ping` des options, parmi lesquelles `-t n` qui définit à n la durée de vie du paquet émis et `-c n` qui envoie exactement n paquets avant de s'arrêter (par défaut, la commande envoie un paquet toutes les secondes jusqu'à ce qu'on l'interrompte avec CTRL-C).
Proposer une méthode pour simuler le comportement de `traceroute` avec la commande `ping`. Vérifier votre méthode.

Réponse: Il suffit de faire `ping -c 1 -t i www.google.com` en commençant à $i = 1$ et en s'arrêtant lorsque la machine qui répond est la machine que l'on cherche à atteindre.

Remarque si lors de test, la commande `traceroute` affiche `*` pour un routeur, c'est que la machine se trouvant à cette distance choisi de ne pas répondre à ces paquets de tests. La connaissance précise d'une route pouvant être un pré-requis à certaines attaques informatiques, il est courant que les routeur « interne » bloquent les paquets de test envoyé par les commandes telles que `ping` et `traceroute`. Attention le fait que la route soit connue ne constitue pas une faiblesse a priori, mais c'est une information qui peut être utile à un attaquant.

6. Il existe des bases de données référençant la localisation de certaines adresses IPs (les IPs de machines fixes des grands fournisseurs d'accès). Le site

<https://www.maxmind.com/en/geoiip-demo>

permet de saisir dans une boîte de dialogue des adresses IPs et d'obtenir leur localisation. Récupérer les adresses IP renvoyées par `traceroute -n www.google.com` (l'option `-n` affiche uniquement les IPs et pas les noms de domaine) et retrouver leur localisation. Il faudra exclure l'adresse 172.17.0.1, cette dernière étant celle d'un réseau local (comme 192.168.x.y) elle n'est pas une adresse IP publique valide.

Réponse: La plupart des IPs se trouvent en France, puis aux USA.

Exercice 3 ◇

Dans cet exercice, nous allons manipuler quelques primitives réseau en Python. Encore une fois, le but est d'illustrer le fonctionnement de certains concepts vu en cours. Il est conseillé de faire cet exercice dans JupyterHub plutôt que dans `geany`. Pour cela :

- dans l'explorateur de fichiers de JupyterHub choisir **New** puis **Text file** (ne pas choisir **Python Notebook**)
 - changer le nom du fichier (en haut de la page) en `client_messages.py`
 - pour tester le programme, basculer dans un terminal (par exemple celui créé à l'exo 2) et faire `python3 client_message.py`
1. Lire attentivement le programme `serveur_messages.py` de la figure 1. Ce programme est celui d'un serveur qui attend des connexions sur le port 4455. Ce serveur maintient en mémoire un dictionnaire dont les clés sont des adresses IP des clients qui se sont connectés à lui, et les valeurs des messages envoyés par ces clients (les messages sont de simple chaînes de caractères). Lorsqu'un client se connecte, le serveur récupère son adresse IP, puis lit le message envoyé et l'ajoute dans le dictionnaire. Une fois ajouté, le serveur envoie comme réponse au client l'historique de tous les messages de tous les clients. On voit bien que ce serveur simule de façon très simple un service de messagerie centralisé (de type discord par exemple). Lorsqu'un participant se connecte, il peut envoyer des messages et recevoir les messages envoyés par tous les autres clients connectés au serveur.
- Un vrai serveur est bien plus sophistiqué : il doit pouvoir envoyer et recevoir des messages à plusieurs clients « en parallèle », gérer des notions de salons ou canaux, authentifier les utilisateurs par mot de passe, *etc.* Le principe de fonctionnement reste cependant le même : le serveur attend les connexions et répond aux clients qui peuvent se connecter à tout moment.
2. Écrire un programme `client_messages.py` qui :
- crée une connexion `cnx`
 - utilise `cnx.connect((ip, port))` où `ip` est l'adresse IP du serveur et `port` le port sur lequel le serveur écoute.
 - demande à l'utilisateur de saisir une chaîne de caractères au clavier avec `input()`

- convertit cette chaîne de caractères en chaîne d'octets
 - envoie cette chaîne au serveur au moyen de `cnx.send(...)`
 - lit des réponses du serveur, les convertit en chaîne de caractères et les affiche, tant que les messages envoyés par le serveur ne sont pas vides.
3. Tester le programme et vérifier que vous pouvez envoyer des messages. Vous pouvez demander à votre chargé de TP d'exécuter le son serveur et de vous donner son adresse IP. Les différentes machines virtuelles JupyterHub se voient entre elle (ce qui n'est pas le cas des machines du 336 qui sont isolées les unes des autres et ne voient que les passerelles et les machines extérieures).

Remarque : vous pourrez tester le corrigé seul en lançant deux terminaux et en lançant dans l'un le serveur et dans l'autre le client. Il faudra modifier le client pour utiliser l'adresse IP `'127.0.0.1'`.

```

1  from socket import socket
2  #On crée une connexion TCP en mode "serveur".
3  #Cette dernière se place en écoute sur l'adresse
4  #fictive '0.0.0.0', sur le port 4455.
5  # ('0.0.0.0' est une adresse spéciale
6  #indiquant qu'on écoute sur toutes les cartes réseau
7  #configurées).
8
9  adresse = ('0.0.0.0', 4455)
10 cnx = socket()
11 cnx.bind(adresse)
12 cnx.listen()
13
14 #Dictionnaire dont les clés sont les IPs
15 #et les valeurs les messages envoyés par le client.
16 messages_clients = {}
17
18 #Le serveur s'exécute "à l'infini" jusqu'à ce
19 #qu'on l'interrompe dans la console avec CTRL-C
20 try:
21     while True:
22
23         #La fonction accept() bloque jusqu'à ce
24         #qu'un client se connecte.
25         #cnx_client est une connexion vers le client
26         #adr_client est un couple (ip_client, num port)
27         cnx_client, adr_client = cnx.accept()
28
29
30         #on reçoit 1000 octets du client, au plus.
31         msg = cnx_client.recv(1000)
32
33         #on ajoute le message dans le dictionnaire en utilisant
34         #comme clé l'ip.
35         ip_client = adr_client[0]
36         if ip_client in messages_clients:
37             mess_client = messages_clients[ip_client]
38         else:
39             mess_client = ""
40
41         messages_clients[ip_client] = mess_client + '\n' + msg.decode()
42
43         #on initialise une chaîne de caractères vide.
44         res = ""
45
46         #on crée une grande chaîne contenant pour chaque adresse
47         #le message stocké
48         for a in messages_clients:
49             res = res + a + ' : ' + messages_clients[a] + '\n——\n'
50
51         #on envoie cette chaîne au client et on se déconnecte.
52         #la chaîne de caractères doit être convertie en chaîne d'octets.
53         cnx_client.send(res.encode())
54         cnx_client.close()
55
56 #Si on presse ctrl-C
57 except KeyboardInterrupt:
58     cnx.close()

```