MU4IN014 – Mini-Projet 1 : Rendu

Sommaire :

Introduction....1

Implémentation : Proxy TCP....1

Proxy TCP – Cache....2

Proxy TCP – Logger....3

Proxy TCP – Censor....3

Implémentations possibles....4

Proxy TCP – Cache....4

Proxy TCP – Logger....4

Configuration....5

Exemple....5

Conclusion....5

Introduction

Le mini-projet a été réalisé par DIEP Richard et comporte deux exercices. Le premier correspond à un proxy TCP avec un code pour le proxy. Le client étant le navigateur web et le serveur étant apache2. Il y a aussi la possibilité d'utiliser un code client et serveur. Ce premier exercice consiste à réaliser le proxy qui est chargé de rediriger les requêtes du client au serveur et les réponses du serveur au client. Le deuxième exercice correspond à plusieurs proxy TCP qui ont chacun leur tâche : un cache http, un loggeur http et un censeur http. Ils ont cependant le même but que dans l'exercice 1, qui est de retransmettre les données entre client et serveur. Nous parlerons dans un premier temps de l'implémentation du proxy TCP, ainsi que les différentes actions possibles dans un proxy. Puis nous verrons ce qui peut être rajouté. Pour finir, un exemple est d'exécution est présenté.

Implémentation : Proxy TCP

Le proxy TCP codé dans ce mini-projet est capable de rediriger les requêtes du client et les réponses du serveur. C'est un intermédiaire entre client-serveur qui contient des variables modifiables : le port, le nom (adresse IP) et la taille du buffer. Ce dernier correspond à la taille maximale du message que peut recevoir le proxy.

```
#Proxy config
proxyPort = 1234
proxyName = ''
proxySocket = socket(AF_INET, SOCK_STREAM)
proxySocket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
proxySocket.bind((proxyName,proxyPort))
proxySocket.listen(1)
bufferSize = 2048
```

D'autres variables sont disponibles et correspondent à la

```
#Server config
serverName = '127.0.0.1'
serverPort = 80 #80 pour le serveur Apache, 5678 pour le serveur local TCPServer.py
serverTimeout = 2 #Assure la déconnexion au serveur s'il y a un problème
```

connexion au serveur (ici Apache2).

Pour gérer une requête d'un client, une fonction handle_request(connSocket) a été faite et a pour but de recevoir la requête du client et d'envoyer la requête au serveur grâce à la fonction request_to_server(connSocket, req), tout en assurant que la requête ne soit pas vide.

La fonction request_to_server effectue dans un premier temps une connexion au serveur. Si la connexion est un échec, un message d'erreur est envoyé au client. Sinon, le proxy récupère toutes les données provenant du serveur et les envoie au client lorsque toutes les données ont été réceptionnées. Pour déterminer si le proxy a tout reçu, on vérifie la taille du message et compare avec la taille du buffer. La taille se trouve dans l'entête réponse du serveur avec Content-Length. Dans le cas où la taille du message moins la taille du buffer est inférieure ou égale à 0, alors toutes les données sont réceptionnées. Sinon il faut continuer à attendre les données.

Les proxy réalisés fonctionnent tous de la même manière ici. Nous allons à présent voir ce qui diffère entre-eux et comment les mécanismes spécifiques ont été implémentés.

Proxy TCP – Cache

Premièrement, nous allons parler du proxy contenant un cache qui possède des variables supplémentaires : le chemin des fichiers dans le serveur Apache, des listes contenant le chemin du fichier et du nom du fichier, et une variable indiquant le chemin où sont enregistrés les fichiers en cache.

```
#Server config
serverName = '127.0.0.1'
serverPort = 1235 #80 pour le serveur Apache, 5678 pour le serveur local TCPServer.py
serverFilePath = '/var/www/html' #Emplacement des fichiers du serveur, ici Apache
serverTimeout = 2 #Assure la déconnexion au serveur s'il y a un problème

#Proxy parameters
proxy_cachePath = [] #Contient le chemin du fichier à partir de serverFilePath.
proxy_cacheFile = [] #Contient le nom du fichier uniquement
proxy_cacheDir = 'caches/' #Les fichiers sont mis en caches dans ce répertoire
```

Lors de l'appel à la fonction request_to_server, la fonction copy_check(connSocket, req) est appelée et permet de vérifier si le fichier demandé est déjà en cache et que le contenu peut être lu par le client. Si le fichier contient des erreurs, le fichier est supprimé des listes et la requête du client est envoyée au serveur.

Dans le cas où le fichier est trouvé, aucune requête au serveur n'est envoyée, et le proxy envoie le fichier au client. La vérification se fait en comparant le nom du fichier et le chemin du fichier pour assurer que le bon fichier est disponible. La seconde vérification étant la plus importante puisque plusieurs dossiers peuvent contenir le même nom de fichier. Une fonction file_infos(reqData) est disponible pour récupérer le chemin et le nom du fichier dans la requête du client.

Dans le cas où le fichier n'est pas trouvé en cache, la requête est envoyée au serveur, la réponse est envoyée au client et une fonction copy_file(connSocket, req, content) est appelée si le serveur a trouvé le fichier dans son répertoire de fichiers. Cette fonction est chargée de créer un fichier s'il n'existe pas et d'écrire les données du serveur dans le fichier. Le chemin et le nom du fichier sont aussi ajoutés dans les listes cache correspondants. Des vérifications sont nécessaires avant de copier : le nom du fichier ne doit pas être vide et doit être un fichier.

Le dossier contenant les fichiers en cache doit être crée avant le lancement du proxy. La création du fichier ne pourra pas se faire si le dossier n'est pas créé.

Proxy TCP – Logger

Deuxièmement, nous allons observer du proxy permettant d'enregistrer dans un fichier log, toutes les requêtes du client et réponses du serveur. La fonction logger_file(reqData, respData) permet de réaliser un fichier log. Deux variables sont utilisées pour la fonction : le nom du fichier contenant le log, et un délimiteur qui permet de séparer chaque requête et réponse.

```
#Proxy parameters
loggerName = "Logger.txt"
loggerDelimiter = '\n\t\t\n'
```

La fonction possède trois étapes :

- Ecriture de la requête du client dans la variable data :
  L'écriture se fait si la requête n'est pas vide. Une date et heure sont écrits avant l'écriture de la requête.
- Ecriture de la réponse du serveur dans la variable data :
  L'écriture se fait si la réponse n'est pas vide. La taille du fichier est écrite avant l'écriture de la réponse.
- Ecriture du contenu de data dans le fichier log :
  L'écriture se fait si data n'est pas vide.

Entre chaque requêtes et réponses, il y a un délimiteur qui est écrit dans la variable data.Ainsi, le fichier log contient toutes les requêtes et réponses passant par le proxy.

Le fichier logger peut contenir des réponses du serveur extrêmement long et le fichier peut devenir rapidement difficile à lire. Pour remédier à cela, il est possible de n'afficher que les requêtes du client et l'entête http des réponses du serveur sans les données. La chaine de caractère \r\n\r\n est le délimiteur entre entête et données, content.split('\r\n\r\n')[0] permet donc de récupérer l'entête uniquement.

Proxy TCP – Censor

Dernièrement, nous allons étudier la façon dont le proxy censor interdit l'accès à certains fichiers. Une variable et une liste de fichiers interdits sont nécessaires. La variable contient le nom du fichier par défaut à retourner au client. La variable peut aussi être un chemin au fichier. A noter que le fichier par défaut doit être crée avant, le proxy ne le crée pas.

```
#Proxy parameters
censorDefaultFile = "CensorDefaultFile.txt" #Fichier par d
censor_ForbiddenFiles = ["adminl.txt", "modl.txt", "file"]
```

Lorsque la fonction request_to_server est appelée, la fonction check_file(reqData) est appelée. Elle permet de vérifier si le fichier demandé est accessible. Si le fichier n'est pas dans la liste des fichiers interdits, alors la requête est envoyée au serveur et la réponse du serveur est envoyée au client. Dans le cas contraire, le proxy envoie le fichier par défaut et indique au client que c'est un fichier inaccessible pour lui.

Pour vérifier que la fichier n'est pas interdit, la fonction check_file récupère le chemin et le nom du fichier avec la fonction file_infos, et vérifie si le nom du fichier se trouve dans la liste de fichiers interdits.

Implémentations possibles

Proxy TCP – Cache

Le fichier copié dans le cache contient la réponse du serveur donc l'entête est inclus et donc certains fichiers ne peuvent pas être ouvert directement dans le dossier de cacheDir, par exemple les images PNG, JPEG, JPG, etc. Si on veut obtenir exactement le même fichier que le serveur, il faut enregistrer uniquement le contenu provenant de la réponse http sans http. \r\n\r\n est le délimiteur entre entête et données, donc un appel à content.split('\r\n\r\n')[1] permettrait d'obtenir les données (content contient toutes les réponses du serveur pour la requête). Mais dans ce cas, il va falloir envoyer un entête http si un client demande le fichier en cache puisque ce n'est pas une réponse du serveur mais un fichier.

Un autre problème en fonction des cas est que le fichier se trouvant dans le répertoire du serveur Apache peut avoir des fichiers qui sont modifiés après que le proxy ait mis en cache le fichier et donc le fichier n'est pas à jour. Pour remédier à ce problème, il y a plusieurs possibilités :

- Il est possible de comparer le fichier en cache et le fichier qui se trouve dans le répertoire du serveur si le proxy a accès au répertoire.
- Créer une liste TTL (Time To Live) qui contient des entiers et qui décrémente à chaque fois qu'une requête est faite. La position de l'entier correspond à la position du fichier dans la liste de fichier en cache. Lorsque le fichier vient d'être mis en cache, on modifie la valeur TTL correspondant au fichier. Cette solution ne règle pas le problème complètement, mais permet de mettre à jours les fichiers en cache.
- Indiquer au serveur d'envoyer une requête spécialement dédiée au proxy qu'un fichier a été modifié.
- Le proxy peut envoyer une requête au serveur vérifiant si le fichier a été modifié après une date ou en comparant la taille des deux fichiers à chaque fois qu'il y a une requête provenant d'un client.
- Etc.

Un autre potentiel problème est la création obligatoire du dossier avant de mettre en cache les fichiers. Pour cela, en important la bibliothèque os, et de vérifier si le dossier existe (avec exists()). S'il n'existe pas, il faut appeler la fonction mkdir(path) provenant d'os.

Proxy TCP – Logger

Le logger ne permet pas de voir les données qu'envoie la réponse du serveur. Elles n'ont pas été ajoutées dans le logger puisque le fichier serait illisible au vu de la taille de certains fichiers. De plus, des fichiers de type image (.jpg, .png, etc.) ne peuvent pas être affiché, seul les octets seront visibles. Mais il est possible d'afficher les textes provenant de fichier .txt par exemple. content.split('\r\n\r\n')[1] contient les données que le serveur a envoyé.

Configuration

Pour pouvoir tester un proxy, il faut tout d'abord ajouter le dossier « files » dans le répertoire d'Apache2. C'est-à-dire dans « /var/www/html/ » (sudo cp –r files /var/www/html/). Attention, il faut prendre le dossier files du dossier Mini-Projet1 et non de l'Exercice1.

Il est possible de modifier le nom et le port du proxy à sa guise, mais aussi le port du serveur (80 étant le serveur Apache par défaut).

De même que la variable serverFilePath être modifié si le chemin « /var/www/html/ » par défaut d'Apache a été modifié.

Exemple

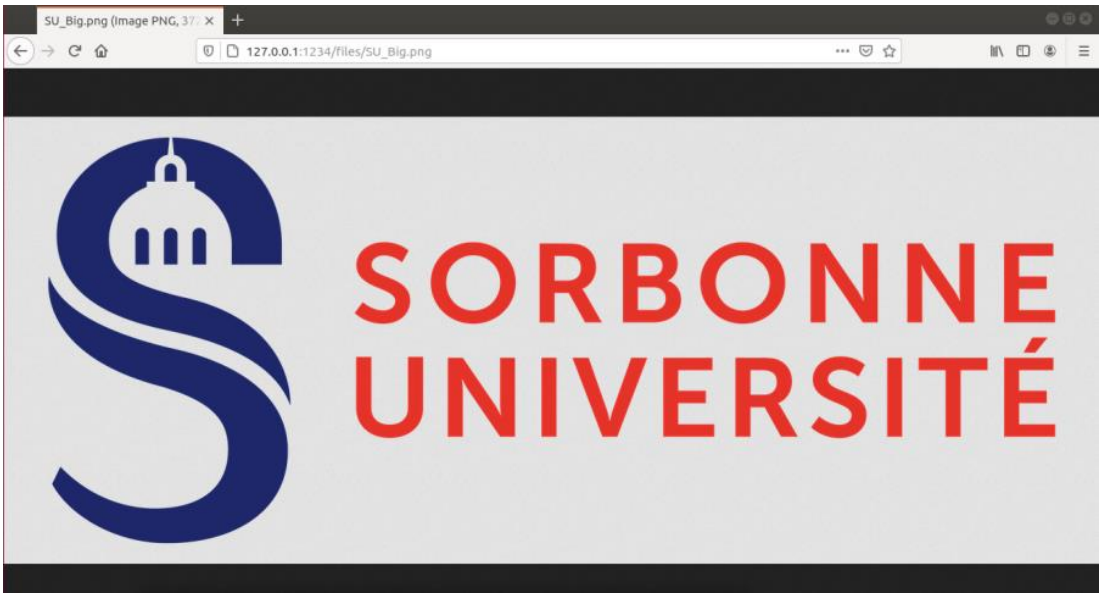
Exécution pour TCPProxy.py de l'exercice 1 :

- Démarrer TCPProxy.py
- Entrer sur le navigateur: 127.0.0.1:1234/files

```
rich@ubuntur:~/Documents/MU4IN014/Mini-Projet1/Exercice1$ python3 TCPProxy.py
Proxy is running

Client request:
 b'GET /files/SU_Big.png HTTP/1.1\r\nHost: 127.0.0.1:1234\r\nUser-Agent: Mozilla
/5.0 (X11; Ubuntu; Linux x86_64; rv:82.0) Gecko/20100101 Firefox/82.0\r\nAccept:
 text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\nA
ccept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3\r\nAccept-Encoding: gzip, de
flate\r\nDNT: 1\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\
n'

All data sent
```



Conclusion

Pour conclure, le proxy permet d'envoyer les requêtes du client au serveur et d'envoyer les réponses du serveur au client. Il permet de réaliser des actions entre le client et le serveur. Par exemple, il est possible d'éviter de surcharger le serveur en enregistrant en cache le fichier demandé au serveur. De plus, s'il y a besoin d'avoir un suivi des requêtes des clients et des réponses du serveur pour surveiller le trafic, le cas d'un proxy logger est le plus adapté. De même que s'il est nécessaire d'interdire l'accès à certains fichiers, le proxy censor correspond le plus à ce cas. Mais ce qui est puissant dans ce cas, c'est la possibilité de « fusionner » les proxy, c'est-à-dire qu'en connectant les proxy entre eux, une requête d'un client au serveur va réaliser toutes les actions de tous les proxy connectés.