

TRAVAIL DE SESSION

Par

Ivan Kenfack & Noviwoto Kokou

Travail présenté à

M. Yacine Benahmed (PhD)

Dans le cadre du cours de Téléinformatique

INF26207

Date

08/04/2024

# **Introduction**

## **Contexte et justification**

Le travail pratique a pour objectif d’explorer l’utilisation des sockets UDP pour la conception d’un service de transfert de fichiers fiable.Contrairement au protocole TCP, les sockets UDP ne garantissent pas la fiabilité ni l’ordre des messages, ce qui nous amène à implémenter manuellement certaines fonctionnalités du protocole TCP comme le fenêtrage,les quittements et la retransmission.

## **Contexte et justification**

Comment compenser l’absence de mécanismes de fiabilité intégrés à UDP pour assurer un transfert sécurisé et sans perte de données ?

## **Objectifs du travail**

Concevoir et implémenter un service de transfert de fichiers basé sur UDP tout en assurant un transfert de fichier fiable sur un réseau simulé non fiable. Implémenter un protocole d’application personnalisé avec en-têtes. Ajouter des mécanismes de fiabilité (accusés de réception, numérotation des paquets, retransmission).

## **Méthodologie**

Approche théorique sur UDP, conception du protocole de fiabilité, implémentation en Python 3.13.2, et tests

# **Fonctionnement de nos applications**

## **Fonctionnement/Logique général**

Le client initie la connexion avec le serveur en utilisant **open <adresse IP>**. Une poignée de main est ensuite déclenchée pour établir la connexion entre les deux sockets toute en négociant quelques paramètres d’envoi de fichiers, notamment : la **taille du segment** et la **taille de la fenêtre**. Après quoi les commandes suivantes peuvent ensuite être utilisées par le client pour interagir avec le serveur, **ls, get nom\_fichier, bye**. Durant la transmission, le fichier est divisé en morceaux, lesquels sont envoyé selon le fenêtrage du client avec des mécanismes de contrôle d’intégrité (retransmission en cas d’échec d’envoi, vérification de données reçues et séquençage) et le serveur attend les ACK.

## **Fonctionnement détaillé de chaque code (serveur et client)**

### **Code Client**

Le code coté client est principalement une boucle while qui itère sur les entrées des commandes de l’utilisateur et dépendamment de la commande entrée, un certain traitement est declenché.

Nous avons utilisé les bibliothèques tels que : **socket** pour établir la connexion réseau, **struct** pour structurer les données pour l’envoi, le **shad1** de **hashlib** pour vérifier l’intégrité des données et le **random** pour générer les valeurs aléatoires. Nous avons définis l’adresse(**localhost**) et le port (**2213**) du client ensuite crée un **socket** **UDP** (SOCK**\_DGRAM**) et associé le socket à l’adresse. Le protocole utilise plusieurs champs pour structurer les segments échangés entre le client et le serveur. Les numéros de séquence (numero\_seq) et d’acquittement (numero\_ack) permettent de suivre l’ordre des messages et de garantir la fiabilité des transmissions. Le champ drapeau est utilisé pour identifier le type de message, comme SYN ou ACK, facilitant ainsi le processus de connexion et de contrôle des échanges. La taille du segment (tailleMorceau) est déterminée de manière aléatoire entre 274 et 280 octet tandis que la taille de la fenêtre du client (fenêtrage) est également générée aléatoirement entre 65486 et 65536 octet

### **Code Serveur**

Du côté serveur, nous avons également utilisé les bibliothèques **socket, struct , hashlib** (SHA1) et **random** pour mettre en place un protocole de communication fiable via UDP. Le serveur est configuré pour écouter en permanence sur une adresse (**localhost**) et un port spécifié **(2212**). À la reception d’un segment SYN du client ,il génère aléatoirement sa propre taille de fenêtre et sa MSS (taille maximale de segment ), puis répond avec un segment SYN-ACK contenant ces paramètres ainsi qu’un checksum pour garantir l’intégrité des données. Une fois le segment ACK reçu du client et vérifié, la conexion est considérée comme établie. Le serveur peut alors traiter les différentes commande envoyées ( **ls , get , bye** ) en fonction du contexte. Les segments reçus sont dépaquétés à l’aide du format structuré, permettant d’extraire correctement les champs comme le numéro de séquence ,le drapeau , le nom de fichier ou les données , et de répondre de manière appropriée à chaque demande du client. Ce mécanisme assure une communication structurée et controlée, malgré l’utilisation d’UDP qui est un protocole non orienté connexion

# **détails de réalisation**

## **Choix de conception**

### **Explication des choix**

Le choix d’utiliser le protocole UDP plutôt que TCP à été motivé par l’objectif de comprendre et de concevoir un protocole de transport fiable en partant d’un protocole non fiable .UDP ne garantit ni l’ordre ni la livraison des paquets , ce qui nous a permis d’implémenter manuellement des mécanismes de fiabilité inspirés de TPC , comme le Three-way Handshake , les accusés de reception, et le contrôle de flux via le fenêtrage. Nous avons opté pour une structration claire des segments affin de faciliter leur traitement et leur interprétation par les deux parties (client et serveur). Le protocole repose sur des échanges simples mais efficaces , où chaque segment contient les informations nécessaires pour assurer l’intégrité, l’identification ,et la synchronisation des messages. L’utilisation de SHAD1 pour le champ checksum garantit que les données reçues n’ont pas été altérées . Par ailleurs , la géneration aléatoire des tailles de fenêtre et des segments permet de simuler une négociation dynamique des capacités réseau.

### **Conception des en-têtes**

Les en-têtes ont été conçus de manière à intégrer tous les champs essentiels au bon fonctionnement du protocole. La structure est définie à l’aide de la bibliothèque **struct** , avec le format suivant :

**!I I 3s I I 40s 15s 200s** Cela corresponds aux champs suivants :

* I (4 octets) : Numéro de séquence (numero\_seq)
* I (4 octets) : Numéro d’acquittement (numero\_ack)
* 3s (3 octets) : Drapeau (SYN, ACK ...)
* I (4 octets) : Taille de la fenêtre ( fenêtrage)
* I (4 octets) : Taille maximale du segment ( tailleMorceau)
* 40s (40 octets) : Checksum SHA1
* 15s (15 octets) : Nom du fichier
* 200s ( 200 octets) : Données

Chaque champ joue un rôle précis dans le contrôle de la connexion, le suivi des paquets , la validation de leur contenu et l’identification des fichiers. En utilisant le format binaire uniforme , letraitement reste rapide et cohérent , tout en facilitant le dépaquétage lors de la réception.

# **détails d’implémentation**

## **Justification des choix techniques**

Pour cette implémentation , nous avons choisi Python en raison de sa simplicité et de ses bibliothèques robustes pour la programmation réseau. La bibliothèque **socket** permet de gérer facilement les connexions UDP, tandis que **struct** nous aide à formater les données à envoyer de manière binaire, essentielle pour un échange réseau performant et structuré.

Nous avons utilisé **hashlib** et son algorithme **SHA1** pour générer un **checksum** assurant l’intégrité des données échangées. Ce choix garantit que toute altération des données lors de la transmission est détectée. L'utilisation de **random** permet de simuler la variabilité des environnements réseau en générant des tailles de fenêtres et de segments différentes, rendant l’échange plus dynamique et proche d’un comportement réel.Enfin, la separation des rôles client et serveur dans deux scripts distincts permet une meilleure organisation du code, une compréhension claire des responsabilité de chaque partie , et facilite les tests et débogages.

## **Détails techniques d’implémentation**

L’implémentation repose sur une communication en UDP, ou le clien initialise une connexion par un segment SYN, et le serveur répond par un SYN-ACK suivi d’un ACK du client, complétant ainsi un Three-way Handshake similaire à TCP. Chaque segment est construit selon un format fixe grâce à la méthode **struct.pack()**avec le format **! I I 3s I I 40s 15s 200s ,**où chaque champ remplit une fonction bien précise :

* Les numéros de séquence et d’acquitement permettet de suivre l’ordre des messages,
* Le drapeau (SYN ,SYN-ACK, ACK) identifie l’etape de la communication
* Le checksum SHA1 valide l’intégrité
* Les données utiles sont stockées dans un champ de 200 octets

Le serveur dépaquette les segments avec **struct.unpack()** et vérifie le checksum recu avec une nouvelle génération locale. En cas de correspondance , il poursuit la communication.Ce mécanisme offre une validation de bout en bout des message. Le client gère différente commandes comme **open , ls , get** et **bye** , envoyées au serveur pour déclencher des actions spécifiques

# **conclusion**

## **Difficultés rencontrées**

Lors de la réalisation de ce TP, plusieurs défis techniques ont été rencontrés. La gestion de la perte de paquets et des retransmissions a nécessité la mise en place d’un contrôle rigoureux des ACK pour assurer la fiabilité des échanges. Il a également fallu implémenter une détection des doublons de segments afin d’éviter des traitements redondants côté serveur. La synchronisation entre les acquittements (ACK) et les retransmissions a posé des difficultés, notamment pour éviter les envois excessifs de segments. Enfin, lors du transfert de fichiers, la gestion de l’encodage et du décodage binaire sans corruption des données a exigé une attention particulière pour conserver l'intégrité des fichiers reçus.

## **Solutions**

Pour faire face à ces difficultés, plusieurs solutions ont été mises en place. Pour la gestion des pertes de paquets, un mécanisme de retransmission basé sur l’absence d’ACK dans un certain délai a été implémenté. La détection des doublons a été assurée par l’utilisation rigoureuse des numéros de séquence, permettant d’ignorer les segments déjà reçus. Afin de garantir une bonne synchronisation entre les ACK et les retransmissions, une logique de suivi des états de chaque segment a été intégrée au protocole. Enfin, pour éviter toute corruption des fichiers transférés, les données binaires ont été soigneusement encodées et décodées en respectant la taille des segments et en validant l'intégrité à l’aide de la signature SHA1.