Rapport de Projet : Protocoles Internet

(Client Peer-to-Peer de partage d'arbres de Merkle)

I. Introduction / Contexte

Le projet consiste en l'implémentation d'un client interagissant avec un serveur API REST qui nous partage des informations sur des peers disponibles pour des échanges d'informations via UDP. Pour ce projet, nous avons décidé de coder le client en Rust en utilisant la bibliothèque de runtime *tokio* ainsi que *egui* pour l'interface graphique.

II. Fonctionnalité implémentées

Le protocole et sa version minimale décrite dans le sujet sont implémentés par notre client:
- s'enregistrer avec le serveur et maintenir la connexion indéfiniment
- rendre une arborescence disponible
- télécharger les fichiers d'un peer

ensuite concernant l'efficacité du programme / les autres fonctionnalités :
- Nous pouvons télécharger l'arbre de Merkle partagé par les peers enregistrés.
- il est possible d'envoyer plusieurs paquets en même temps (asynchrone)
- il est possible de choisir un chunk ou big à télécharger
- nous avons implémenté Nat Traversal
- nous avons implémenté une GUI

III. Spécifications techniques de l'implémentation

Notre projet est structuré de la forme model / view. Le contenu du model se trouve dans la crate client_network et la view dans client_gui. La communication entre les threads se fait via des canaux "crossbeam channels" (c.f. Documentation Rust). L'interface graphique envoie des commandes via l'envoi de "NetworkCommands" (FetchPeerList(), Ping(user), etc.) et la logique rapporte les informations via des "NetworksEvents" (PeerListUpdated(list), DataReceived, etc.).

Détail d'implémentation - Logique de récupération d'une adresse socket d'un peer :

Lorsque notre client souhaite récupérer une adresse socket UDP, il va procéder de la sorte :
Un thread dédié va d'abord envoyer une requête HTTP pour récupérer la liste d'adresse du peer enregistré au serveur. Une fois cette liste récupérée, nous récupérons uniquement les adresses IPv4 de la liste (IPv6 non implémenté). Le thread va ensuite effectuer cette procédure pour chaque adresse de la liste jusqu'à recevoir une réponse d'une adresse :

- Envoi d'un ping à l'adresse puis attente de 1s.
- Si l'adresse a répondu, retourne cette adresse comme moyen de communication fiable.
- Sinon, envoi d'une requête Nat Traversal au serveur avec l'adresse du peer puis attente de 1s.
- Si réception de ping de l'adresse, on la retourne.
- Sinon, envoi d'un ping à l'adresse puis attente de 3s.
- Si l'adresse ne répond pas, on passe à la suivante.

Détail d'implémentation - Système de réémission de message (exponential backoff):

Nous utilisons un algorithme de backoff exponentiel pour remettre les paquets si ceux ci n'ont pas reçu de réponse ou sont arrivés cassés ou incomplets. A chaque envoi de message, nous avons une RetryMessage contenant le message à remettre, son nombre de tentatives d'émissions et l'heure unix à laquelle il faut le renvoyer au prochain essai. Nous l'ajoutons dans une VecDequeue : un thread va se charger de récupérer le premier élément de la queue, vérifier si l'heure de réémission est dépassée : si oui, le message est réémis et le champ retrymessage associé voit son nombre de tentatives augmenter, sinon le message est remis en fin de queue. Ce système ne garantit pas que les messages seront envoyés exactement à l'heure de leur réémission prévue mais le délai est censé rester acceptable.

IV. Extensions implémentées

Notre client implémente les extensions suivantes :

- Messages NatTraversalRequest2 implementés
- Algorithme d'exponential backoff
- Interface graphique

V. Conclusion / Retour

Ce projet nous a permis d'utiliser en situation réelle des notions que l'on a apprises en cours comme la communication en UDP, la gestion de congestion, réémission de message, communication asynchrone... Le fait d'avoir effectué ce projet en Rust nous a également permis de découvrir ce langage moderne, ses points positifs ainsi que négatifs.

VI. Sources / Bibliographie

https://doc.rust-lang.org/stable/rust-by-example/
https://docs.rs/http/latest/http/index.htm
https://docs.rs/p256/latest/p256/
https://doc.rust-lang.org/std/net/struct.UdpSocket.html
https://docs.rs/sha2/latest/sha2/
https://rutracker.org/forum/index.php
https://docs.rs/egui/latest/egui/
https://docs.rs/crossbeam/latest/crossbeam/