Rapport de Projet : Système de Fichiers P2P

Jonas DOS SANTOS
Nathan FAUVELLE-AYMAR

25 janvier 2026

Table des matières

1 Fonctionnalités Principales

Ce projet implémente un nœud de réseau pair-à-pair (P2P) permettant le partage de fichiers en lecture seule de manière décentralisée. L'architecture repose sur un modèle hybride combinant un serveur central pour la découverte et des communications directes en UDP pour le transfert de données. L'intégrité des fichiers est garantie par une structure d'arbre de Merkle.

Sont implémentés l'intégralité du sujet de base, ainsi que deux extensions : `Hello` via `NatTraversalRequest` et chiffrement via échange de clef `Diffie-Hellman`.

1.1 Architecture Générale

Chaque pair est autonome et gère :
- Une identité cryptographique persistante (ECDSA P-256).
- Une base de données en mémoire associant des hashs SHA-256 à des blocs de données.
- Un serveur UDP asynchrone capable de traiter plusieurs requêtes simultanément.

1.2 Découverte et Enregistrement (`register`, `hello`)

La découverte des pairs se fait en deux temps :

1. **Register** (Enregistrement HTTP) : Le pair signale sa présence au serveur central via une requête signée, associant son nom à sa clé publique et son port d'écoute.
2. **Hello** (Handshake UDP) : Une fois l'adresse d'un pair obtenue, une session est établie via un échange de messages `Hello` et `HelloReply`, tous deux signés cryptographiquement pour prévenir l'usurpation d'identité.

1.3 Partage (`load`)

1. **Load :** Le système de fichiers local est converti en une structure partageable via la commande `load`. Les fichiers sont découpés en blocs (Chunks) de 1 Ko. Ces blocs sont organisés en un arbre de Merkle comportant quatre types de nœuds (Chunk, Directory, BigNode, BigDirectory). La racine de cet arbre (*RootHash*) suffit à identifier de manière unique l'état complet du dossier partagé.

1.4 Consultation et Téléchargement Fiable (`print`, `download`)

1. **Download :** Le protocole de téléchargement implémente un mécanisme de fiabilité sur UDP (avec timeouts exponentiels).
   - Le téléchargement commence par la récupération du *RootHash* du pair distant.
   - L'arbre est parcouru récursivement : pour chaque hash manquant, une requête `DatumRequest` est envoyée.
   - À la réception, l'intégrité de chaque bloc est vérifiée (`SHA256(Data) == Hash`).
   - Pour optimiser la vitesse, les requêtes sont parallélisées à l'aide de Goroutines, régulées par un sémaphore (Congestion Control) limitant le nombre de requêtes simultanées.
2. **Print :** Cette fonctionnalité permet d'afficher l'arborescence d'un pair.

— Le fonctionnement est assez similaire à celui de **download** sauf qu'aucun DatumRequest n'est envoyé sur les feuilles de l'arbre de Merkle. On affiche juste le contenu des dossiers.

1.5 Consultations d'informations (`info, active, peers, key, addr`)

De multiples commandes permettent de consulter des informations liées à notre session active, ainsi que celles accessibles depuis le serveur :

1. **info :** Renseigne les informations suivantes sur votre pair : son nom, son adresse auprès du serveur s'il a `Hello` le pair associé au serveur, ainsi que le `RootHash` associé au système de fichiers qu'il partage.
2. **active :** Renseigne l'adresse IP, le port et la clé publique de chaque pair ayant une session active auprès de votre pair.
3. **peers :** Renseigne les pairs s'étant récemment manifesté auprès du serveur central.
4. **key :** Renseigne la clé publique d'un pair connecté auprès du serveur central.
5. **addr :** Renseigne les adresses IP d'un pair connecté auprès du serveur central.

1.6 Gestion des keep-alives

`keepAlive.go` : Ce fichier implémente la logique de maintenance des sessions via la fonction `Start__maintenance__loop`. Exécutée dans une Goroutine indépendante, cette boucle se réveille à intervalle régulier (toutes les minutes) pour inspecter l'état de chaque session active dans la `map` des pairs. Elle applique une politique de gestion du temps stricte :

— **Keep-Alive (> 3 min) :** Si un pair n'a pas donné signe de vie depuis plus de 3 minutes, un message `Ping` lui est envoyé automatiquement. Cela sert un double objectif : vérifier que le pair est toujours en ligne et, surtout, générer du trafic sortant pour empêcher le routeur/NAT de fermer le port (NAT Timeout).

— **Expiration (> 5 min) :** Si aucune activité n'est détectée après 5 minutes (malgré les pings), le pair est considéré comme déconnecté. La session est alors supprimée de la mémoire pour libérer les ressources.

2 Extensions Implémentées

Conformément aux suggestions du sujet, nous avons implémenté deux extensions majeures pour renforcer la sécurité et la connectivité du réseau.

2.1 Extension de Confidentialité (Diffie-Hellman)

Le protocole de base garantit l'authenticité (via signatures) mais pas la confidentialité. Nous avons ajouté une couche de chiffrement de bout en bout respectant la propriété de **Forward Secrecy** (Confidentialité Persistante).

- **Activation lors du handshake :** Lors du Handshake, si les deux pairs supportent l'extension (indiqué via le bitmap dans le message `Hello`), ils initient un échange de clés Diffie-Hellman sur la courbe elliptique **X25519** (recommandation NIST RFC 7748).
- **Echange de clés :** Nous avons choisi de créer un nouveau type de message pour supporter cet échange de clés, afin de ne pas s'encombrer d'une génération et d'un partage de clé à chaque `Hello`) avant de savoir si le pair avec lequel on communique supporte un tel chiffrement (TypeKeyExchange = 20). Cet échange se lance automatiquement.
- **Clés Éphémères :** Les clés privées utilisées pour cet échange sont générées en mémoire vive et détruites immédiatement après le calcul du secret partagé. Ainsi, une compromission future de la machine ne permet pas de déchiffrer les communications passées.
- **Transport (AES-GCM) :** Une fois la session sécurisée, les données (`Datum`) sont chiffrées et authentifiées avec l'algorithme **AES-256-GCM** (grâce aux fonctions de la librairie crypto de Go). Les informations enregistrées concernant la session permet de savoir si un Datum reçu doit être déchiffré ou non.

2.1.1 Commentaire sur ce protocole :

Nous avons échangé avec un autre groupe concernant les normes à adopter pour l'implémentation de ce protocole. Nous n'avons pas adapté notre protocole au leur, qui diffère principalement par le choix de faire l'échange de clef lors du Handshake et non via des messages indépendants. Ceci peut donc causer des erreurs lors des Hello/HelloReply entre les pairs implémentés par notre groupe et le leur.

2.2 Traversée de NAT (NAT Traversal)

Pour permettre la communication entre pairs situés derrière des pare-feux stricts ou des NAT, nous avons implémenté la technique de "Hole Punching" assistée par un tiers (le serveur ou un autre pair).

- Le pair A envoie un `Hello` au pair cible B. Si cela échoue, il faut `NatTraversalRequest`, mais cette étape est nécessaire pour "créer un trou" du côté de A.
- Le pair A (bloqué) envoie une requête. `NatTraversalRequest` à un intermédiaire R.
- L'intermédiaire R relaie cette demande au pair cible B via `NatTraversalRequest2`.
- Le pair B initie alors une connexion sortante vers A (un Ping). Ce paquet sortant "perce" le NAT de B et crée une entrée dans sa table de routage, autorisant ainsi les futurs paquets venant de A à entrer.

2.2.1 Commentaire sur ce protocole :

Il est spécifié dans le sujet qu'à la réception d'un NatTraversalRequest2, il faut envoyer un Ping à l'initiateur de la traversée, cependant le peer "jch.irif.fr" est implémenté tel qu'un ping reçu d'un peer inconnu soit rejeté avec l'erreur "please hello first".
De notre côté, si on implémente ces deux fonctionnements on rencontre un problème, ils sont incompatibles, ou du moins, il y a un problème de logique : pourquoi envoyer un message dont on est sûr que la réponse sera une erreur ?
Nous avons néanmmoins décidé d'implémenter ces deux fonctionnements à la différence de certains autres groupes avec qui nous avons pu échanger.

3 Analyse Structurelle du Code

L'architecture du projet est modulaire, séparant la logique réseau, la cryptographie et la gestion des fichiers. Voici le rôle détaillé de chaque fichier source.

3.1 Point d'entrée

`main.go` : C'est le fichier principal pour le lancement de l'application. Il initialise la structure principale `Me`, charge l'identité cryptographique, lance la boucle d'écoute réseau dans une Goroutine, puis entre dans une boucle interactive (CLI). C'est ici que les commandes utilisateur (`register`, `download`, `nattraversal`) sont parsées et exécutées.

3.2 Package `pkg/p2p` (Protocole UDP)

`peer.go` : Définit la structure `Me` qui représente l'état global du pair (connexion UDP, base de données, clés, sessions actives). Il contient la fonction `Listen_loop`, qui reçoit les paquets bruts, les désérialise et les dirige vers les bons handlers.

`messages.go` : Définit le format binaire des messages (Type-Length-Value). Il contient les fonctions `Serialize` et `Deserialize` qui transforment les structures Go en tableaux d'octets prêts à être envoyés sur le réseau, gérant l'écriture des entêtes (ID, Type) et des signatures conformément au sujet.

`handlers.go` : Contient la logique décisionnelle pour chaque type de message reçu. Par exemple, `Handle_hello` vérifie la signature et crée une session ; `Handle_DatumRequest` cherche la donnée en mémoire et répond ; `Handle_KeyExchange` effectue les calculs mathématiques pour établir le secret partagé.

`senders.go` : Centralise la logique d'envoi. Il implémente notamment la fonction critique `Send_with_timeout`, qui gère la fiabilité sur UDP via un mécanisme de réessais exponentiels et l'utilisation de canaux (Channels) pour attendre les réponses asynchrones.

`download.go` : Implémente l'algorithme de téléchargement récursif. Il utilise des primitives de synchronisation avancées (`sync.WaitGroup`) pour attendre la fin des téléchargements multiples, et un canal "sémaphore" pour limiter le nombre de requêtes simultanées (Congestion Control). Il contient aussi la logique de reconstruction des fichiers sur le disque.

`keepAlive.go` : Gère la maintenance du réseau. Une boucle nettoie périodiquement les sessions inactives et envoie des `Ping` automatiques pour maintenir les connexions NAT ouvertes (Keep-Alive).

3.3 Package `pkg/identity` (Cryptographie)

`crypto.go` : Fichier gérant toute la logique de chifrement à partir de la bibliothèque crypto/ de Go. Il fournit des fonctions simples pour signer/vérifier (ECDSA), chiffrer/déchiffrer (AES-GCM) et générer les clés éphémères (X25519) nécessaires à l'extension de confidentialité.

`key_storage.go` : Gère la persistance de l'identité. Il permet de sauvegarder et de charger la clé privée depuis un fichier `identity.pem`, assurant que le pair conserve la même identité entre les redémarrages.

3.4 Package `pkg/filesystem` (Arbre de Merkle)

`file.go` : Responsable de la transformation des fichiers locaux en structure de Merkle. Il lit les fichiers, les découpe en blocs de 1024 octets, calcule les hashs SHA-256, et assemble les nœuds parents (Directory, BigNode) pour former l'arbre complet.

3.5 Package `pkg/client` (HTTP)

`client.go` : Gère toutes les communications avec le serveur central. Il encapsule les requêtes HTTP `GET` et `PUT` nécessaires pour récupérer la liste des pairs, obtenir leurs clés publiques ou leurs adresses IP.

4 Code non original & sources

Globalement, de nombreuses fonctionnalités de notre codes sont basés sur des morceaux de codes trouvés dans la documentation Go (https://pkg.go.dev/) retravaillés.

4.1 key_storage.go

Conçernant le choix du format PEM pour le stockage de clef : nous avons remarqué que ce format semblait courant :
https://stackoverflow.com/questions/21322182/how-to-store-ecdsa-private-key-in-go

4.2 main.go

Pour la gestion du 'scanner' dans le main, nous avons cherché comment faire sur des forums et sur la doc du package bufio.
https://stackoverflow.com/questions/20895552/how-can-i-read-from-standard-input-in-the-console

4.3 crypto.go

La première partie du fichier crypto.go concernant la signature est une copie du code proposée dans le sujet du projet.

La seconde partie qui concerne l'échange de clés Diffie-Hellman et le chiffrement AES utilise largement ce que propose la documentation des bibliothèques crypto/cipher et crypto/aes.

AES : https://pkg.go.dev/crypto/aes
cipher : https://pkg.go.dev/crypto/cipher