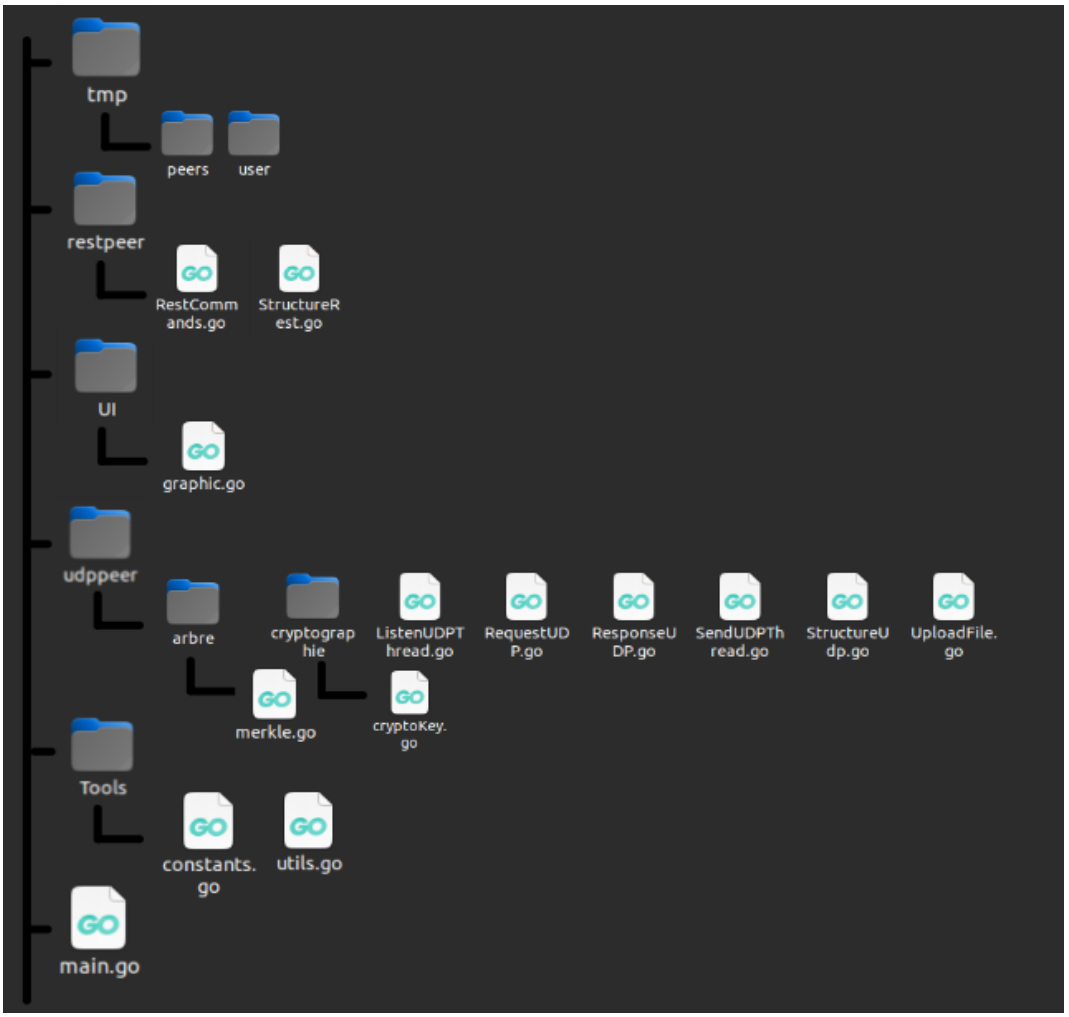
Rapport de Protocoles Internet

Benakli William 21960601
Charly Sonneville 71800649

1 - Architecture de fichiers:



Le projet est espacé en 5 dossiers distincts:

La fonction du dossier **tmp** de contenir dans le dossier **user** nos fichiers que l'on souhaite pouvoir partager et dans le dossier **peers** contiendra un dossier pour chaque pair dont ont aura décidé de télécharger leurs arborescence.
Le dossier **restpeer** quant à lui contient les fichiers go qui permettent l'envoie de commande restAPI c'est dans ce dossier que l'on gère l'envoie des commandes rest.

Le dossier **UI** est celui qui contient le code pour l'affichage graphique (ont utilise la bibliothèque Fyne pour l'affichage)
Le dossier **udppeer** est le dossier principal du projet, il contient les fichiers qui permettent l'écoute et l'envoie en UDP ainsi que toutes les structures qui s'en rapportent. Ont a également décidé d'ajouter les sous dossier **arbre** qui contient la structure de l'arbre de merkle, et le dossier **cryptographie** qui s'occupe des fonctions utiles à la cryptographie.
Le dossier Tools permet juste de définir des constantes pour notamment une meilleure lisibilité du code. bien sur pour finir nous avons le fichier **main** que nous utilisons pour lancer le programme

2 - Structure de donnée du projet :

```
type RequestUDPExtension struct {
    Id     int32
    Type   uint8 // care changement
    Length int16
    Body   []byte
}
```

Tout d'abord nous avons un type qui correspond à un paquet. Nous conversion les octects reçu par cette structure et vice verca avec les fonction BytesToStruct() et StructToBytes()

Nous avons fait une version étendue de la même structure avec du temps pour les rémissions.

```
type RequestTime struct {  2 usages
    TIME    int64
    REQUEST RequestUDPExtension //  t
}

var RequestTimes sync.Map  5 usages
```

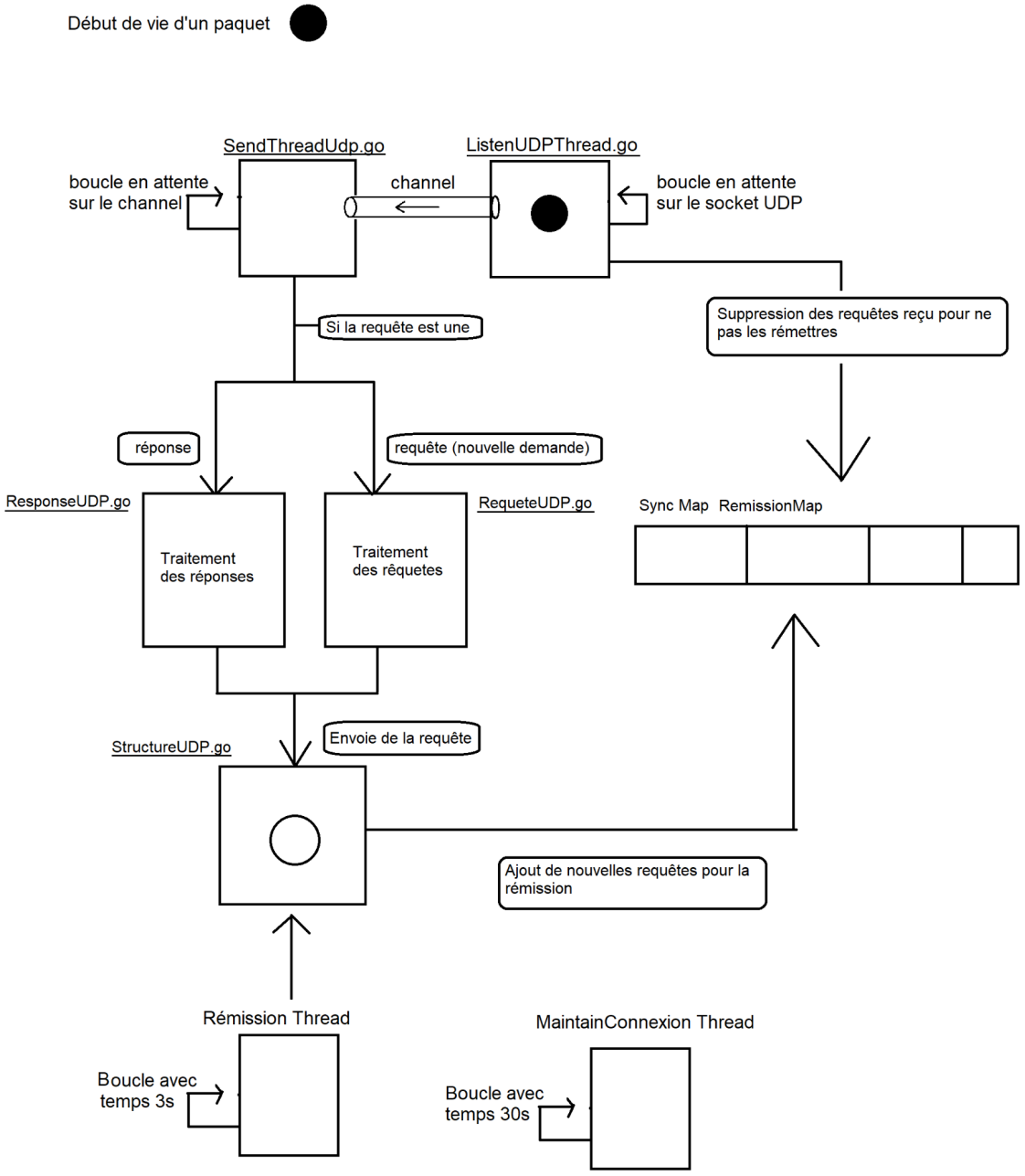
```
type Noeud struct {  30 usages
    //  HashCalculate []byte
    ID          int
    Type        int8
    HashReceive []byte
    NAME        string
    Data        []byte
    Fils        []*Noeud
}
```

Enfin, nous avons notre Arbre de merkel. Cette structure nous permet de convertir stocker les paquets reçu sous forme d'arbre. Elle permet également lors de l'upload de renvoyé le bon hash.

Avec l'ID on assure que les paquets sont reçu dans l'odre et construit dans le bonne ordre.

3 - Architecture du système:

Schéma détaillé de l'architecture principale



Explication du schémas :

ListenUDPThread : Ce thread permet l'écoute sur le port UDP afin de pouvoir réceptionner les réponses, de les traiter, conversion en RequestUDPExtension et de les renvoyer par un channel vers SendThreadUDP.

SendThreadUDP: Ce thread analyse le type reçu par le paquet RequestUDPExtension. Ajoute le paquet à une liste de paquet qui ont déjà été vu (pour ignorer les rémissions de l'émetteur) et renvoie vers les fonctions ReponseUDP et RequestUDP.

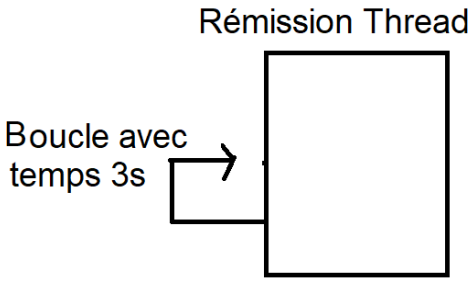
ReponseUDP : Cette fonction analyse le type de la requête et renvoie à StructureUDP la rêquete à envoyer. L'id de la requête reste identique.

RequestUDP : Cette fonction analyse le type de la requête et renvoie à StructureUDP la rêquete à envoyer. L'id de la requête est changé.

StructureUDP : Cette fonction ajoute à la rémission le paquet et envoyer sur le socket UDP la rêquete.

RémissionThread:

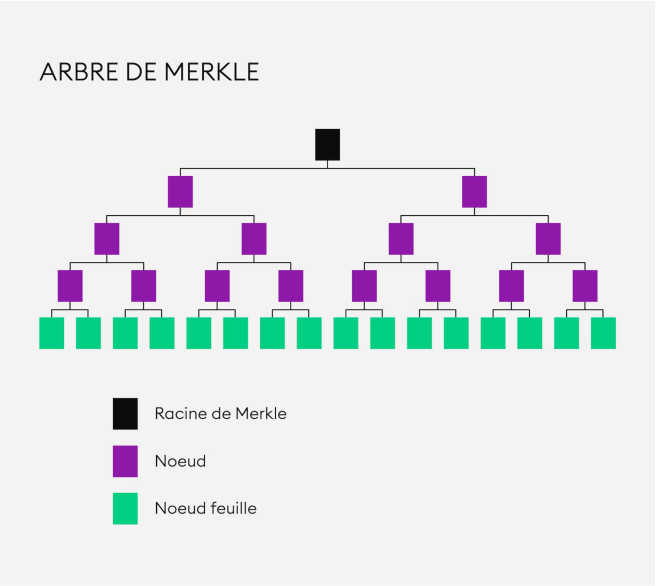
Pour la rémission de paquets, nous rémeton des paquets toutes les 3 secondes à un temps aléatoire compris entre 20ms et 500ms. (expliquer plus en detaille dans la partie 5)
.



MaintainConnexion : c'est le thread qui toutes les 30 secondes communique avec le serveur pour ne pas qu'on nous oublies pour ce faire il envoie un message de type HelloRequest

5 - Principaux algorithme:

Arbre de merkle



Notre algorithme recursif permet la construction d'un arbre de merkel lors de la réception de paquet. Les Noeud Feuilles sont des chuncks de donnée inferieur à 1024. Les Noeud sont des BigFile ou des Directory.

Nous parcourons donc recursivement sur les fils et nous ajoutons à l'abre au hash du père.

(photo: https://www.bitpanda.com/academy/fr/lecons/tout-ce-que-vous-devez-savoir-sur-les-arbres-de-merkle/)

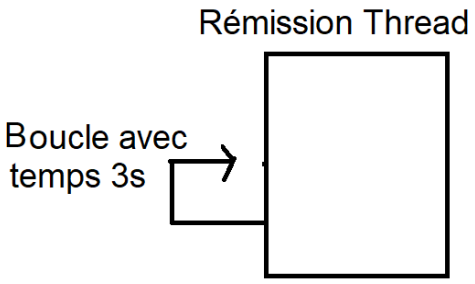
Avec cette algorithme nous obtenons cette arbre. Et nous l'utilisons également pour l'upload de fichier.

Nous obtenu cette arbre :

```
Lancement du programme
Connexion REST API terminée
Noeud : Type 2 Fils: 1 ID: 0 Hash: ce4ac, Name: user, Data: 0
       -Noeud : Type 2 Fils: 2 ID: 0 Hash: 66469, Name: okayyyyyy, Data: 0
       -       -Noeud : Type 2 Fils: 3 ID: 0 Hash: f4efb, Name: on, Data: 0
       -       -       -Noeud : Type 0 Fils: 0 ID: 0 Hash: 6e340, Name: empty, Data: 0
       -       -       -Noeud : Type 0 Fils: 0 ID: 0 Hash: 6e340, Name: low, Data: 7
       -       -       -Noeud : Type 1 Fils: 20 ID: 0 Hash: , Name: oui, Data: 0
       -       -       -       -Noeud : Type 1 Fils: 33 ID: 0 Hash: , Name: , Data: 0
       -       -       -       -       -Noeud : Type 0 Fils: 0 ID: 0 Hash: , Name: , Data: 1024
       -       -       -       -       -Noeud : Type 0 Fils: 0 ID: 1 Hash: , Name: , Data: 1024
       -       -       -       -       -Noeud : Type 0 Fils: 0 ID: 2 Hash: , Name: , Data: 1024
       -       -       -       -       -Noeud : Type 0 Fils: 0 ID: 3 Hash: , Name: , Data: 1024
       -       -       -       -       -Noeud : Type 0 Fils: 0 ID: 4 Hash: , Name: , Data: 1024
       -       -       -       -       -Noeud : Type 0 Fils: 0 ID: 5 Hash: , Name: , Data: 1024
       -       -       -       -       -Noeud : Type 0 Fils: 0 ID: 6 Hash: , Name: , Data: 1024
       -       -       -       -       -Noeud : Type 0 Fils: 0 ID: 7 Hash: , Name: , Data: 1024
```

Rémission Thread:

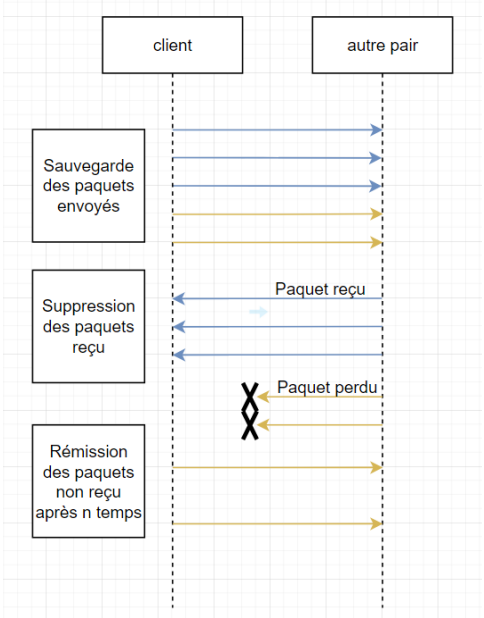
Pour la rémission de paquets nous sauvegardons un nouveau Type RequestTime dans une sync map (une map qui gère le parallèlisme en go).




```
type RequestTime struct {  2 usages
    TIME    int64
    REQUEST RequestUDPExtension //  t
}

var RequestTimes sync.Map  5 usages
```

Cette structure contient le paquet avec un temps donnée. Nous conservons dans la map RequestTimes le temps d'envoie et le paquet avec la clé ID.



Ce qui nous permet par la suite à la reception de paquet de supprimer les paquets reçu pour ne réémettre que les paquets perdu.

Afin d'éviter une surcharge du réseau, l'intervalle entre les réémissions des paquets perdus est défini de manière aléatoire. Cette approche prévient l'envoi simultané de blocs de paquets, ce qui pourrait s'avérer contre-productif.

Upload de fichier:

Lorsque nous recevons une demande de type 'getDatum' pour téléverser un fichier vers un autre pair, notre processus consiste à parcourir l'arbre de merkel. Nous transmettons les hash des nœuds progressivement jusqu'à atteindre chunk, où nous envoyons alors les données. Cette méthode est rendue possible grâce à l'implémentation d'un arbre de Merkle dans notre système.

6 - Cryptographie:

Pour la parti cryptographie on a regroupé les fonctions que vous nous avez donné dans le fichier **cryptoKey.go** la plupart des fonctions parle d'elle même les choses importante à retenir sont que la classe contient, notre privateKey initialisé au début du programme, la clef publique également initialisé au début et on enregistre la clef publique de notre interlocuteur dans la variable otherPublickey.
Lorsque l'on nous demande notre publique key on la renvoie en la formatant dans la requête publicKeyReply et nous signons les messages publicKey, publicKeyReply,Root et RootReply.

7 - Problèmes rencontrées :

Le principal problème rencontré, à été la vérification de Hash lorsque l'on reçoit un message et la création des hash.
La création des hash n'est pas très dure lorsque nous avons un dossier ou un chunk. En revanche lorsque nous avons un bigFile c'est plus compliqué car nous devons parcourir récursivement l'arbre pour calculer le bon Hash.
C'est une partie relativement compliquée car les hash ne sont pas lisibles et nous devons donc afficher le body et essayer de comprendre ce qu'il se passe. La fonction qui calcul le hash que l'on envoie et la fonction requestDatumReply() présente dans le fichier **RequestUdp**

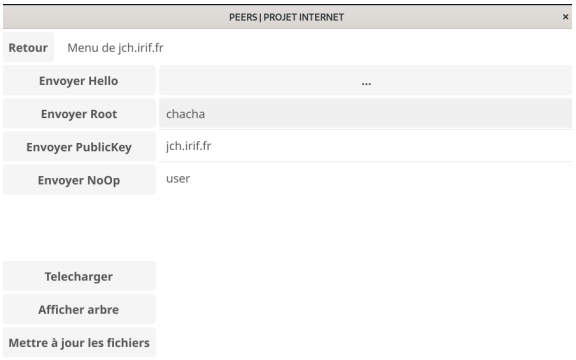
Un problème qui est d'ailleurs corrélé a été le téléchargement de BigFile, il fallait également un algorithme récursif avec un débogage compliqué.

Et enfin un problème toujours pas résolu est le fait que l'on perd des paquet et donc qu'on les réémet le soucis étant qu'on les remet trop donc on a des pertes conséquente ce qui fait que le téléchargement et l'envoie d'arborescence peuvent devenir assez lent notamment pour une arborescence qui contient de grosse vidéo par exemple.

8 - Extensions:

- Gestion des pertes de paquets: Avec notre système de rémission.

- Interface graphique



- Mise à jour des fichiers depuis l'interface graphique