

Projet de Programmation Réseaux :

**« Groupe de discussion par inondation fiable »**

Membres :

**Alim Yanis**

**Douah Ilies**

Sous la direction de :

**Mr Juliusz Chroboczek**

*Mai 2019*

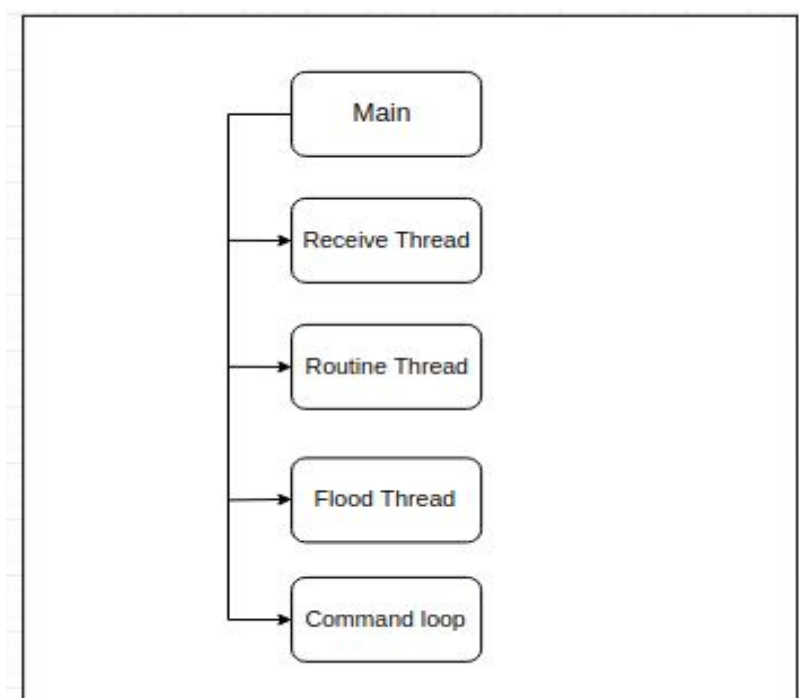
# 1) Introduction :

Le pair à pair est un système de communication entre plusieurs ordinateurs. Il s'agit d'un réseau informatique grâce auquel des internautes vont pouvoir s'échanger des fichiers divers (vidéos, images, fichiers textuels, etc). Sa particularité est que les différents éléments ne sont pas stockés sur un serveur central, mais sur les différents ordinateurs composant ce réseau. Chaque client est donc un serveur.

Ce projet nous a aidé à mettre en pratique ce qu'on a appris pendant le cours et les TP à propos de la programmation réseau et surtout la programmation UDP.

Avant d'entamer l'implémentation, on a pris notre temps à étudier le contenu du projet et ses principes fonctionnalités, vous pouvez trouver notre résumé dans notre répertoire sur [moodle](#).

Voici l'architecture de notre solution qu'on a appelé Diderot Group Communication Protocol (DGCP) :



Architecture de DGCP.

Le détail de cet implémentation va être expliquer dans ce qui suit.

## 2) Manuel :

### 2.1) Compilation :

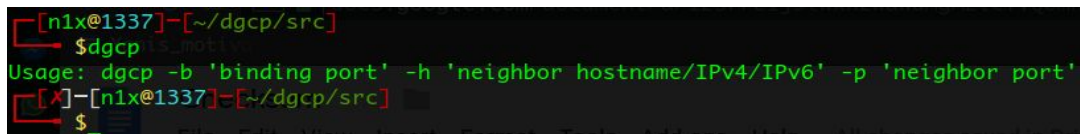
Pour compiler il faut exécuter la commande :

```
make dgcp
```

Pour installer il faut exécuter la commande :

```
chmod +x install.sh  
./install.sh
```

### 2.2) Utilisation :



```
[n1x@1337]~[~/dgcp/src]  
$dgcp  
Usage: dgcp -b 'binding port' -h 'neighbor hostname/IPv4/IPv6' -p 'neighbor port'  
[n1x@1337]~[~/dgcp/src]  
$
```

Pour utiliser dgcp, il le faut 3 arguments :

- -b “binding port” : c’est le numéro de port à lequel bind la socket
- -h “neighbor hostname” : soit le nom de la hôte du pair distant, soit son IPv4 ou IPv6
- -p “neighbor port” : le numéro de port du pair distant

Exemple :



```
[n1x@1337]~[~/dgcp/src]  
$dgcp -b 1337 -h "jch.irif.fr" -p 1212  
$
```

## 3) Réalisation :

### 3.1) Sujet minimal :

DGCP assure le bon fonctionnement du projet minimal.

### 3.2) Les extensions :

- *Gestion d’erreur* : dgcp traite tout sorte d’erreur en se basant sur la définition du protocole tel que tout sorte de paquet non reconnue soit à cause du l’entête ou du corp va être ignoré avec l’envoi d’un tlv warning où d’un tlv goaway selon le cas d’erreur.
- *Concurrence* : dgcp peut recevoir plusieurs donnée à la fois et faire de l’inondation simultanément grâce au 2 threads séparés ( Receive Thread & Flood Data Thread ).
- *Agrégation* : dgcp peut agréger plusieurs TLV dans un seul message soit à la réception soit à l’envoi ( par exemple, le partage de voisins ) et cela se fait d’une façon optimale grâce à la structure ( header.h ) qui définit un paquet dgcp tel qu’elle élimine la phase de parser un paquet pour identifier ses champs.

- *Adresses multiples* : dgcp traite les pairs qui ont des adresses multiple en les identifiant grâce à leur ID puis ajouter une entrée (IP,PORT) dans la liste des adresses. On pensait à ajouter un flag qui indique quelle paire est la plus activé pour but d'optimiser et d'éviter d'inonder une même donnée à la même personne qui a plusieurs interfaces.
- *Sécurité de l'implémentation* : dgcp a simulé le comportement du pair (jch.irif.fr,1212) quand on lui envoie des paquets non attendus : taille supérieure au datagramme, débordement de tlv, communication avec des pairs inconnus...
- *Sécurité cryptographie* : dgcp pense à implémenter des communication chiffrer dans ce qui suit, d'une façon à utiliser les deux clefs ssh existant. En première partie, on fera un partage de la clef publique avec les voisins. Puis, chiffrer un message à l'aide de la clef publique et le mettre dans tlv data de type 100 par exemple. Enfin, à la réception d'un tlv data de type 100, on utilise la clef privée pour déchiffrer le message.
- *Les autres extensions* : dgcp pourra dans le future proche implémenter les autres extensions.

## 4) Implementation :

DGCP se compose de plusieurs sous modules qui interagissent entre eux pour assurer le bon fonctionnement du protocole, on décrit dans ce qui suit le contenu et le rôle de chaque module dans le protocole :

### Header :

ce module définit les structures du paquets dgcp, leur fonction de création, les structures des voisins et la structure de donnée :

```
typedef struct
{
    uint8_t magic;        // (magic != 93) -> ignore
    uint8_t version;      // (version != 2) -> ignore
    uint16_t body_length; // (body_length > PACKET_SIZE-4) -> ignore out of bounds data
} __attribute__((packed, scalar_storage_order("big-endian"))) msg_hd;

typedef struct
{
    type_t type;          // if (type != [0-7]) -> ignore
    uint8_t length;       // if (length > body_length-2) -> ignore out of bounds
    unsigned char ip[16];
    uint16_t port;
} __attribute__((packed, scalar_storage_order("big-endian"))) tlv3;

typedef union _msg_body // union to make only one of the tlvs possible
{
    tlv0 pad1;
    tlv1 padn;
    tlv21 s_hello;
    tlv22 l_hello;
    tlv3 neighbor;
    tlv4 data;
    tlv5 ack;
    tlv6 goaway;
    tlv7 warning;
} msg_body;

typedef struct
{
    msg_hd header;
    msg_body tlv;
} dgc_packet;
```

\_\_attribute\_\_((packed, scalar\_storage\_order("big-endian"))): pour ne pas laisser du padding entre les champs du paquet et les arranger en big endian.

```
typedef struct recent_neighbors // the list that contains recent neighbors, symmetric f
{
    uint64_t id;
    ip_port* key;
    int symmetric;
    time_t hello_t;
    time_t long_hello_t;
    struct recent_neighbors* next;
} __attribute__((packed, scalar_storage_order("big-endian"))) recent_neighbors;

typedef struct potential_neighbors // the list of potential neighbors, the id exists
{
    uint64_t id;
    ip_port* key;
    struct potential_neighbors* next;
} __attribute__((packed, scalar_storage_order("big-endian"))) potential_neighbors;
```

On a deux type de voisin : récent et potentiel, le récent c'est lui qui communique avec nous actuellement, le potentiel peut être récent dans le future ( l'ami de mon ami ).

```
typedef struct data_array          // the data array structure
{
    data_key key;
    uint8_t type;
    char* data;
    recent_neighbors* data_neighbors;
} data_array;
```

Chaque donnée à une clef (ID,NONCE) et un type et la liste de voisin à inonder.

*Les fonctions :*

- *uint64\_t generate\_id()* : génère un ID pour la pair en utilisant l'adresse MAC pour but de garder le même ID dans différent session ( mieux à nous identifier )
- *uint32\_t generate\_nonce()* : génère un nonce en se basant sur le temps actuelle
- *int check\_header(dgc\_packet\* m)* : vérifier si l'entête est bonne
- *void create\_X(dgc\_packet\* m, ..... )* : crée un tlv X

## neighborController :

Ce module sert à contrôler les listes des voisins et la liste de donnée, les fonctions nécessaire sont ( notez que X représente soit potentielle soit symétrique ) :

- *search\_id\_X(...)* : cherche un voisin en utilisant son ID
- *search\_key\_X (...)* : cherche un voisin en utilisant son (IP,PORT)
- *create\_X\_neighbor (...)* : ajoute un voisin
- *delete\_id\_X (...)* : supprime un voisin en utilisant son ID
- *delete\_key\_X (...)* : supprime un voisin en utilisant son (IP,PORT)
- *nb\_recent\_neighbors (...)* : calcule le nombre de voisin récent
- *search\_data (...)* : cherche une donnée dans la table de données
- *add\_data (...)* : ajoute une donnée à la table de données

## dgcp\_handler :

Ce module est se résume 3 actions nécessaire : écouter, traiter, réagir. Expliquant ce raisonnement en détail :

- *int get\_peer\_info(...)* : remplit les information de la socket de la paire destinataire.
- *void \*routine(...)* : un thread qui automatise 3 tâches : rafraîchissement des hello long avec les voisins symétriques chaque 30sec, partage de voisins chaque 2mn, découverte de voisins chaque 1mn30sec si le nombre de voisin récent est inférieur à 8.
- *void \*dgcp\_rcv(...)* : un thread qui écoute pour des connections multiples non bloquantes et fait appel à la fonction qui traite le paquet reçu.
- *void \*data\_flood(...)* : un thread qui inonde les tlv data
- *void dgcp\_send(...)* : sert à envoyer un paquet dgcp
- *void header\_handler(...)* : il vérifie l'entête reçu et map chaque tlv avec sa fonction de traitement.
- *X\_handler(...)* : X est le nom d'un tlv, ces fonctions vérifier le contenu et la source du tlv et réagissent de bonne façon à ces tlv.
- *void flood\_clean(...)* :supprime les voisins qui n'ont pas envoyé un ack pour un tlv data envoyé.
- *void share\_neighbors(...)* : crée et envoie un tlv neighbor qui contient tout les voisins symétriques.
- *void check\_neighbors(...)* : vérifie le temps du dernier tlv hello envoyé et faire la bonne action selon ce temps.
- *void discover\_neighbors(...)* : envoie un hello short pour tous les voisins potentiels pour agrandir son réseau.
- *void move\_to\_potential(...)* : déplace un voisin de la liste potentielle à la récente.

## Commandline :

Ce module traite les commandes entrées par l'utilisateur :

- *nick(...)* : définir le nom de l'utilisateur.
- *add(...)* : ajouter un voisin (IP,PORT) à la liste des voisins.
- *rm(...)* : supprimer un voisin (IP,PORT) à la liste des voisins.
- *send(...)* : envoyer une donnée aux voisins.
- *print(...)* : afficher la liste des voisins
- *verbose(...)* : activer le mode de débogage pour voir ce qui se passe en arrière plan.
- *leave(...)* : quitter le groupe.

## Main :

Comme le montre l'architecture de l'introduction, le main commence d'abord par parser les arguments de la ligne de commande. Une fois les arguments sont bons, il lance 3 threads consécutives ( Receive Thread, Routine Thread, Flood Thread ) puis il écoute aux commandes entrées par l'utilisateur est fait le bon appel selon la commande entrée.

## 4) Conclusion :

Personnellement, on a bien adoré faire ce projet et on souhaitait si seulement si on a avait plus de temps pour vous rendre un travail parfait. On va continuer dans ce projet pendant notre temps libre pour réaliser une solution plus utilisable au niveau des réseaux WAN.

Deux critiques à propos de ce projet est au niveau de fragmentation, si un voisin nous envoie des paquets fragmentés pourquoi doit on acquitter les paquets et les contrôler, ce qui apparaît comme recreation d'une roue qui existe déjà (TCP). Une autre critique à propos du système d'identifier un voisin à l'aide de son ID ce qui mal sécurisé vu qu'un attaquant pourrait avoir tout les ID des voisins et aura la possibilité de vider les listes de voisins en prenant les ID des autres et envoyer des tlv goaway à tout le monde et ainsi de suite.