Rapport de projet programmation réseaux 2020 (Dazibao par inondation non-fiable)

Réalisé Par :

- Djelid Aymen (L3-Info4)
- Benamara Abdelkader Chihab (L3-Maths-Info)

Le 10/05/2020

2019/2020

Table des matières

Introduction :

Le projet de programmation réseaux pour l'année universitaire 2019-2020 réalisé par Djelid Aymen et Benamara Abdelkader sur le sujet de Dazibao et inondation non-fiable sous l'architecture de pair-pair (peer2peer) qui comme l'indique le sujet permet en bref d'avoir une synchronisation en termes de données publiées par chaque pair dans le protocole et pour implémenter les taches demandées dans le sujet on a divisé les différentes étapes du projet en premier temps des structures qui permettent le réutilisation du code et puis les fonctions qui sont auxiliaires et nécessaires pour l'exécution de notre implémentation du Dazibao.

Partie 01 (Structures utilisées) :

TLV & Tlv_chain:

```
// TLV data structure

struct TLV
{       int8_t type; // type
        char  * data; // pointer to data
        int16_t size; // size of data
};
```

```
// TLV chain data structure. Contains array of (50) tlv
// objects.
struct tlv_chain
{
    tlv  object[MAX_TLV_OBJECTS];
    uint8_t used; // keep track of tlv elements used
};
```

Cette structure comme indiqué dans le sujet représente un tlv (type ,longueur (size), value (data)).Pour la structure tlv_chain représente un tableau d'un maximum de (TLV_MAX_OBJECT=1000) de tlv.

Triplet & Data :

```
struct Triplet {
   unsigned char  id[8];
    uint16_t numDeSeq;
    char *data;
    int incr;
    struct Triplet *suivant;
};
```

```
struct Data{
    Triplet *tete;
    uint8_t used;
};
```

Dans le sujet cette structure représente la liste des données publiée sous forme de liste linéaire chainée allouée d'une manière dynamique ou chaque maillon est un triplet (node_id, num_seq, data).

Pour la structure Data elle contient la liste des triplets avec un indicateur de fin de liste (used)

Voisins :

```
struct Voisin {
    char *ip ;
    uint16_t port;
    struct timespec date;
    int permanent;
}Voisin;
```

```
struct Voisins  {
    Voisin *TableDevoisins[Max_voisin];
    uint8_t used;
};
```

C’est un tableau de maximum 15 voisin (pair) qui sont eux même des structures (ip, port ,last_seen,permanent) le champs date (last_seen) permet de vérifier si le pair en question nous a contacté les dernières 70 secondes et permanent est à 0 si le pair en question n’est pas permanent et 1 sinon.

Args : (dans la gestion des threads)

```
struct arg {
    Voisins *arg1;
    int sockfd;
};
```

```
struct arg2 {
    Voisins *arg1;
    Data *datalist;
    int sockfd;
};
```

Les deux structures en dessus sont utilisées comme des paramètres des deux threads qu’on va détailler dans la partie (boucle principale)

Partie 02 (Fonctions utilisées) :

Idée globale de notre implémentation :

Pendant l'envoi de données (en format conventionnel série de tlv dans un paquet) tout au début on insere les tlv un par un à la liste de tlv (tlv_chain) soit grâce à la fonction de caractère abstrait (add_tlv) et puis on transforme cette tlv_chain à un buffer avec la fonction (chain2Buff) et puis il sera temps à ajouter les entetes du paquet (magic=95 et 01) ceci implique l'utilisation de la fonction (chain2Paquet) pour les détails des fonctions citées au-dessus :

add_tlv : insere un tlv à une tlv_chain

chain2Buff : transforme une chaine de tlv à un buffer (u_char* [1])

chain2Paquet : cette fonction transforme en paquet une liste de donnée de tlv.

Rentrons un peu dans le détails du parsing des listes de tlv :

ParserV1 : cette fonction sert à désérialiser une (u_char*)[1] en une tlv_chain

parserTlv : cette fonction joue un rôle principal dans notre code puisque c'est ici ou on fait un switch des différents types de tlv reçu et donc on peut ici appliquer les étapes du protocole en envoyant à chaque fois le réponses dédiées tout en changeant les tables (Voisins & Data)

---

1 u_char* : unsigned char *

2

parserPaquet : cette fonction permet de parser les paquets reçu grâce à recvfrom et donc si le paquet en question respecte bien nos conventions en pourra le traiter sinon il est automatiquement ignoré.

parserIP : cette fonction permet de transformer les adresses ip reçu en format hexadécimale des paquets vers un format lisible par le lecteur (char*)

Partie 03 (manipulation de listes de données & voisins) :

Les fonctions de cette partie sont à propos de la partie de gestion de liste des voisins et inondation de données.

parcoursVoisins : cette fonction teste pour chaque pair dans notre table de voisin si ce dernier nous a contacté les derniers 70 secondes et si ce n'est pas le cas on peut le supprimer de notre table et ceci seulement si le pair n'est pas permanent.

moins5Voisins : cette fonction teste si notre table de voisins contient moins de 5 pairs et si c'est le cas on envoie au hasard un TLV_NGR[2] à un de notre table de voisins obtenu grâce à la fonction **hasardVoisin**.

rechercheEmetteur : cette fonction sert à savoir si le pair donné en argument existe dans notre table de voisins.

Hachage de données !

Cette partie est constitué de trois grandes fonctions pour la gestion de tout ce qui est hash de données :

Hash : cette fonction retourne tout simplement les 16 premiers octets d'un hash SHA256 de notre donnée passée en paramètre.

---

[2] TLV_NGR :tlv neighbour request

concatTriplet : cette fonction concatène une triplet (id, seq , data) et donc elle pourra donner un résultat utile pour la fonction **networHash()**

networkHash : cette fonction permet de calculer le hash du réseau en effectuant un tri rapide selon les id et puis on concatène chaque triplet et on le hash.

En plus de ces fonctions essentielles au déroulement du protocole il y a aussi les fonctions un peu auxiliaires qui permettront l'affichage de nos structures afin de voir notre implémentation en concret !

Boucle principale du projet :

Afin de mettre en marche toutes les fonctions citées un pair s'exécute en boucle infinie et on a choisi d'utiliser deux threads[3] et un select !

Tout d'abord le premier **thread** permet de parcourir et vérifier si la table de voisins contient moins de cinq voisin chaque 20 secondes il lance donc la fonction (miseAjour**20s** qui fait exactement ce boulot)

Et puis le deuxième **thread** qui lance en arrière-plan sendNet20s et donc permet d'envoyer un network hash et puis unifier les données publiées (grâce à les tlv N_S [4]) chaque 20 secondes

Le select permet enfin de faire en sorte que lors de la réception d'un paquet d'un autre pair de traiter ce paquet et donc dés la réception du paquet il fait appel à **parserPaquet** et comme déjà cité on peut exécuter notre protocole sans aucun souci.

---

[3] Threads grâce à la bibliothèque pthreads et ses fonctionnalités

[4] N_S : tlv de type node state (permettant de donner une idée de l'état du pair)

Partie 04 (Gestion d'erreurs) :

Dans cette partie on essaye de rendre le projet robuste et donc en gérant les différentes erreurs qui peuvent apparaitre dans notre implémentation on propose deux types d'erreurs et donc deux solutions !

- Un paquet annonce une longueur supérieure à la taille du datagramme qui le contient moins quatre octets (cette condition est vérifiée dans la fonction principale parserPaquet

)

- le cas d'un TLV qui annonce une longueur qui le ferait déborder du paquet (cette condition est vérifiée dans la fonction parserV1

).

Partie 05 (extensions) :

Vérification de la cohérence des Node State :

- Un paquet avec un hash erroné avec celui qu'on calcule pour vérifier dans ce cas on envoie également un warning avec un message contenant le hash qu'on doit trouver !

Calcul des hashes :

- dans cette partie au lieu de recalculer le hash à chaque fois on vérifier le numéro de séquence du voisin si il a été incrémenté, et pour cela il suffit de vérifier l'attribut incr (dans la structure Triplet), cet attribut est en fait un toggler en terme d'utilisation c'est-à-dire si le numéro de séquence a été modifié(incrémenté) on le met à 1 et sinon il reste à 0. Et donc au cas de réception d'un TLV de type 6 il suffit seulement de

vérifier ce champ s'il est à 1 on le remet à 0 et dans ce cas un TLV de type 7 sera envoyer. Dans le cas échéant on n'a rien à faire

Agrégation :

Dans le traitement de nos paquets émis et/ou reçus on gère bien ceux d'entre eux qui contiennent plusieurs TLV de différents types et d'une manière qu'on juge être optimale car à chaque fois on traite un TLV d'une liste la réponse qui doit être retourner est directement faite sans aucune attente de traitement des autres TLV d'une même liste et cela donc implique un temps de traitement un peu large mais en termes d'efficacité cela reste un point fort.

Quelques Captures de notre projet !

Table de voisins en plein exécution :

```
Affichage VOISINS :

port =1212 || IP = ::ffff:81.194.27.155
port =49153 || IP = ::ffff:88.124.234.159
--------------------------------------------------------------------------------
```

Table de données partagées :

```
--------------------------------------------------------------------------------
 Affichage DATA :
0000000000000000  :
0000000000000001  : testtt
0000000000000002  : 17:24 - This is another pamplemousse :D
0000000000000003  : Node state test
0000000000000004  :
0000000000000005  : Node state test
0000000000000006  : ❤️✖❤️🔻 ❁✖🔥 🔥 🔥 ✖🌹 🌹 🌹 🐦✖🐦 🪶 🪶 ⚽ 🔔 🍖✖

0000000000000007  : test tmp
0000000000000008  : 17:24 - This is another pamplemousse :D
0000000000000009  : test tmp
000000000000000a  : n
000000000000000b  : Node state test
000000000000000c  : J'ai passé une excellente soirée mais ce n'était pas celle-ci.
000000000000000f  : oo
0000000000000010  : test tmp
0000000000000011  : dattebayo
0000000000000012  : !!!
0000000000000013  : `��U
0000000000000014  : dattebayo
0000000000000015  : Node state test
0000000000000018  :
0000000000000019  :
000000000000001a  : !!!
```