

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

Réalisé Par :

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

Table des matières

Rapport de projet programmation réseaux 2020 (Dazibao par inondation non- fiable).....	1
Réalisé Par :	1
Introduction :	3
Partie 01 (Structures utilisées) :	3
TLV & Tlv_chain:	3
Triplet & Data :	3
Voisins :	3
Partie 02 (Fonctions utilisées) :	4
add_tlv :	4
chain2Buff :	4
chain2Paquet :	4
ParserV1 :	4
parserTlv :	4
parserPaquet :	4
Partie 03 (manipulation de listes de données & voisins) :	5
parcoursVoisins :	5
moins5Voisins :	5
rechercheEmetteur :	5
concatTriplet :	5
networkHash :	5
Boucle principale du projet :	6
Partie 04 (Gestion d'erreurs) :	6
Partie 05 (extensions) :	6

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 tâches demandées dans le sujet on a divisé les différentes étapes du projet en premier temps des structures qui permettent la 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:

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 :

Dans le sujet cette structure représente la liste des données publiée sous forme de liste linéaire chaînée 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 :

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.

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](#))

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.

¹ u_char* : unsigned 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_NGR² 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.

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

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 !

² TLV_NGR :tlv neighbour request

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 threads³ 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 (**miseAJour20s** 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_S⁴) 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 dès la réception du paquet il fait appel à **parserPaquet** et comme déjà cité on peut exécuter notre protocole sans aucun souci.

Partie 04 (Gestion d'erreurs) :

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

Un paquet excédant la longueur maximale (1024octets au lieu d'ignorer directement le paquet qui dépasse la taille maximale ce serait une bonne idée d'envoyer un warning avec le message adéquat au pair correspondant (mais tout en s'assurant que le magic est bien à 95 et donc c'est bien un paquet qui doit être traité).

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 !

Partie 05 (extensions) :

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)