Rapport de projet – L3 MI

15 MAI 2020

Projet de programmation réseau

Dazibao par inondation non-fiable

PROJET RÉALISÉ PAR

GRÉGOIRE REMY

THIBAULT TRAN

université PARIS DIDEROT PARIS 7

Université de Paris

Sommaire

Notre projet

Nous avons implémenté la partie obligatoire de ce projet sans extensions.

C'est à dire un **dazibao** utilisant deux protocoles :

- Maintenance d'une liste de voisins

- Inondation

Ce qui ne fonctionne pas :

Parfois le *malloc* nécessaire à l'envoi d'un Hash du réseau peut échouer.

Cela est dû au fait que lorsque nous demandons d'allouer une zone de mémoire trop importante, le *malloc* peut échouer.

Structure du programme et techniques employées

A. Structures employées

- **Voisin** : il contient 4 attributs : une adresse IP, un port (un short) allant avec cette adresse, un INT permanent et une structure *Timeval* pour connaitre l'heure du dernier paquet reçu (pour la maintenance de la liste des voisins). Dans le *main*, nous utilisons un tableau de 15 Voisin (tableau de structure).
- **Data** : Elle contient :
    - un uint64. Ici, nous nous intéressons seulement aux octets (un tableau de charactère de longueur 8 est équivalent) pour représenter l'ID d'une donnée ;
    - un INT len_chaine qui contient la longueur de la chaine.(On ne peut utiliser strlen car les messages sont sans '\0').
    - un short qui représente le numéro de séquence ;
    - un tableau de 192 charactères qui représente le message ;
    - un INT *myself* qui sert à identifier la donnée propre à l'utilisateur.

Dans le *main*, nous utilisons un tableau de Data d'initialement 5000 données et qui est agrandit de 50% lorsque la capacité maximale est dépassée.

B. Techniques employées

- **Boucle à événements** : Nous utilisons une boucle *for* qui est infinie avec 3 événements principaux.
  1. Le premier évènement est le suivant : dès réception d'un paquet, nous le traitons. Pour nous tenir informés de sa réception, nous utilisons *select* sur la *socket*.
  2. Le second événement est lorsque l'utilisateur envoie un message via l'entrée standard qui est sur écoute : nous utilisons également *select*.
  3. Le troisième et dernier évènement a lieu toutes les 20 secondes : nous envoyons un TLV4 , trions la liste des voisins, etc.

  Ici, nous utilisons *select* car l'appel *recvfrom* est bloquant. De plus, cela nous permet de moins utiliser le processeur.

- **Parsage des TLV** : Lorsque nous recevons un paquet, nous utiliserons une fonction qui traite les TLV de ce paquet en les parcourant un par un. En fonction du type du TLV, nous appellerons une des fonctions "AnswerTLV" pour y répondre. De plus, nous nous déplaçons de TLV en TLV en utilisant l'attribut length du TLV (le deuxième octet de celui-ci). Si le TLV est d'un type non désiré, nous passons au suivant.

- **Ajout des Données dans le tableau** : Ici, nous ajoutons les données au fur et à mesure, alors nous les ajoutons dans un tableau déjà trié. Il suffit de trouver la place de la nouvelle donnée en fonction de son ID, puis de décaler les autres données d'un cran vers la droite.

- **Préparation d'un Paquet :** Pour préparer un paquet, nous commençons par ajouter dans le *buffer* les TLV que nous souhaitons envoyer avec les fonctions "MakeTLV".

Puis, nous appelons “create_req” pour rendre la requête correcte (bon header et bonne taille).

- **Fonction de Hashage :** Pour les fonctions de ‘*hashage*’, nous utilisons le git et l’exemple que vous nous aviez communiqué sur la liste. Ainsi, nous créons simplement un tableau ou un pointeur contenant la taille des données à ‘*hasher*’.

- **Envoyer un message :** Il suffit que l’utilisateur entre un message dans l’entrée standard et appuie sur entrée. Avec *select*, nous détectons ce message, nous mettons à jour la table des données et envoyons un TLV8 à nos voisins.

Différence avec le sujet

Dans l’ensemble, nous avons respecté les consignes et les protocoles à appliquer. Nous avons simplement ajouté une fonction “actuMyData” qui à chaque boucle d’événements envoie un TLV7 avec l’ID du nœud propre à l’utilisateur. Cela permet d’avoir le numéro de séquence de notre donnée plus rapidement pour pouvoir envoyer des messages plus vite.

Nous avons préféré faire une interface utilisateur réduite pour ne pas mélanger entrée standard et sortie standard. De plus “jch.irif.fr” étant extrêmement efficace pour cela, nous ne voulions pas voler la vedette.

Remarques

Afin d’avoir un protocole plus fiable, nous pensons qu’il faudrait limiter le nombre de TLV à 1 par paquet ; ce qui est le cas la plupart du temps sauf pour les TLV6.

Pourquoi ?

Il n’est pas bien sûr que le TLV que nous allons recevoir soit bien construit, notamment avec le deuxième octet qui correspond à la length. En effet, si celui-ci est faux, nous ne pouvons plus *parser* correctement le reste du paquet et nous ignorons donc des TLV.

Avec un TLV par paquet, si celui est mal formé, nous ne perdons que celui-ci et non pas les autres.