Rapport du projet de reseau

Akinotcho
Méjane

May 2019

Table des matières

1 Introduction

1.1 Présentation générale

Ce document explique nos choix d'implémentations et pistes d'améliorations de l'application de chat par inondation.

1.2 Comment compiler et exécuter le chat

Effectuer dans le terminal :

— cd lib_dir/build

— cmake ..

— make

— sudo make install

— ldconfig

Dans le cas, où cela ne fonctionnerait pas, commencer par exécuter les commandes :

— mkdir lib

— cd lib

— git clone https ://github.com/warmcat/libwebsockets.git

— sudo apt-get install libssl-dev

— mkdir build

Ensuite dans le répertoire parent du projet, lancer make. L'exécutable généré est de nom main.

Pour utiliser le chat, après l'avoir compilé, il faut l'exécuter avec 0,1,2 ou 3 arguments, qui représente le port auquel on va lié la socket, et l'adresse de la personne à qui envoyer le premier hello ainsi que le port. Par défaut la socket se lie à un port aléatoire et le programme envoie un hello au port 1212 de l'adresse "jch.irif.fr". Avec un argument on choisit le de liaison de la socket. Avec deux, on choisit l'adresse et le port avec lequel on veut communiquer. Et enfin avec trois, on choisit les le port de liaison, l'adresse et le port de récepteur.

2 Structure interne

Lors de la création du chat, nous avons eu à faire un choix de représentation pour nos structures de données. Nous avons décidé d'en utiliser deux différentes.

2.1 De l'arbre binaire de recherche pour les voisins ...

Afin de représenter tous nos voisins, nous avons décidé d'implémenter un arbre binaire de recherche. Cette structure de données nous permets ainsi de pouvoir accéder à tous les voisins de manière efficace mais aussi de d'avoir accès à un voisin précis en temps logarithmique. Nous avons deux arbres, un pour

représenter les voisins et un et un pour les voisins potentiels, qui à partir de l'ip et du port nous donne l'existence, la date du dernier hello et la date du dernier hello long.

2.2 A la liste chainée pour les données

Ensuite, nous avons une liste chainée. Cette liste contient les données à envoyer chacune associée à une liste des voisins à qui il faut envoyer la données. Le choix de la liste à été fait afin de pouvoir trier à la fois les données dans l'ordre de création (de la plus vielle à la plus récente) mais aussi les voisins (nous expliquerons cette liste plus en détails dans la partie 3.2).

3 Le projet minimal...

3.1 Envoie et réception des messages

Lors de la réception d'un message par la socket, nous regardons tout d'abord le magic, la version et la taille du message. Si le magic ou la version de correspondent pas alors on ignore le message, de plus si le longueur du corps du message plus les 4 octets d'entête sont supérieurs à la taille des données reçu par la socket on ignore le message. Ensuite on traite les tlv un par un en vérifiant toujours que les données lu ne dépassent pas la taille du message annoncé. Si le tlv est de type 4 (data), alors on envoie un acquittement.

Pour l'envoie, toutes les 30 secondes on envoie des hello longs, tous les voisins symétriques, des messages goaway aux voisins qui ne sont plus symétriques et un hello cours à tous les voisins potentiels si on a moins de 8 voisins symétriques (on en profite aussi pour supprimer les voisins potentiels qui n'ont pas répondu à un hello ou dont on a pas reçu d'information dans les deux dernières minutes). En parallèle, quand l'utilisateur veut envoyer un message, on ajoute ce messages a la liste d'inondation.

3.2 Inondation

Comme dit dans la partie 2.2, notre liste d'inondation est une liste de message + liste de voisins. Cette liste est maximum de taille 100 et lorsqu'on ajoute un message on l'ajoute a la fin (pour avoir un temps constant) et on enlève les messages les plus anciens lorsque la liste à atteint sa taille maximale (le premier élément). Ensuite, à chaque message la liste des voisins on associe un temps avant le prochain envoie a chaque voisin, la liste est trié dans par rapport à ce temps. A chaque inondation, on parcours la liste les messages et la liste des voisins et on envoie le message à qui (d'après le temps tiré au hasard selon les règles du sujet) il est temps de les envoyer. Si on à déja envoyé le message 5 fois, on supprime de voisin de toute nos liste et de notre arbre de voisins.

4 ... et plus encore

4.1 Interface graphique web

Nous nous sommes servis de la librairie C libwebsocket qui implémente le protocole websocket afin d'obtenir une interface web réalisée en html, jquery et css.

4.2 Agrégation des TLV

Pour agréger les tlv, nous avons tout d'abord associé un message à chaque voisin. Ensuite à chaque tour de boucle, tout les massages devant être envoyé à un voisin x seront tout d'abord stocké dans le message associé à x et si il est plein (ajouter un tlv ferait dépasser le PMTU) l'envoie est immédiat, sinon on envoie régulièrement les messages associés aux voisins (chaque tour de boucle).

5 Comment avoir un projet plus complet ?

Nous pourrions améliorer notre projet en le rendant plus efficace (en calculant le PMTU pour chaque voisin), plus pratique pour l'utilisateur (pouvoir se connecter ajouter un voisin à partir de son ip et de son port). Nous aurions aussi aimé pouvoir implémenter la découverte de voisins en multicast. De plus, l'envoi de fichiers est une fonctionnalité qui a été implémentée mais n'a malheureusement pas été rendue fonctionnelle, faute de temps et de maitrise de la librairie libwebsockets.

6 Spéciales dédicaces

Nous tenons tout d'abord à remercier Juliusz Chroboczek sans qui rien n'aurait été possible, mais aussi Maxime Flin, Pierre Gimalac, Rodi-can Bozman, Omar Aldakar avec qui nous avons partager des idées et des connaissances, et enfin nos mères qui ont toujours été présentes pour nous.