

Projet de programmation réseau

BOZMAN Rodi-Can & ALDAKAR Omar

Table des matières

| | | |
|----------|---|----------|
| 1 | Manuel d'utilisation | 2 |
| 1.1 | Installation des paquets pour l'interface graphique | 2 |
| 1.2 | Compilation | 2 |
| 1.3 | Exécution | 2 |
| 1.4 | Exemple d'utilisation | 3 |
| 2 | Fonctionnalités implémentées | 6 |
| 3 | Détails de l'implémentation | 6 |
| 3.1 | Strucutre général du projet | 6 |
| 3.2 | Inondation | 7 |
| 3.2.1 | Structures de données utilisées | 7 |
| 3.2.2 | Trame | 7 |
| 3.3 | Envoie des requêtes | 7 |
| 3.4 | Réception des requêtes | 7 |
| 3.5 | Maintenance de la liste des voisins directs | 8 |
| 3.5.1 | Structures utilisées | 8 |
| 3.5.2 | Trame | 8 |
| 3.6 | Maintenance de la liste des voisins potentiels | 8 |
| 3.6.1 | Structures utilisées | 8 |
| 3.6.2 | Trame | 8 |
| 3.7 | Envoie des "gros" messages | 8 |
| 4 | Conclusion | 9 |

1 Manuel d'utilisation

1.1 Installation des paquets pour l'interface graphique

Avant toute chose, il est nécessaire, afin de faire fonctionner notre programme de télécharger quelques bibliothèques :

- \$ sudo apt-get install libgtk-3-dev
- \$ sudo apt-get install fonts-noto-color-emoji

1.2 Compilation

Pour compiler notre programme, on se place dans le dossier `BOZMAN_ALDAKAR_RESEAU/` puis on lance la commande

- \$ make

puis pour supprimer les binaires, on peut utiliser la commande :

- \$ make fclean

1.3 Exécution

Il existe deux versions de notre chat : une version avec un affichage graphique et une version avec un affichage terminal, pour lancer la version terminal il suffit d'exécuter la commande :

- \$./main 'mon-numéro-de-port' 'mon-pseudo'

où 'mon-numéro-de-port' est remplacé par le numéro de port sur lequel on veut que notre serveur écoute et 'mon-pseudo' est remplacé par le pseudo de notre choix, on peut également décider de ne pas choisir de pseudo et on aura alors par défaut le pseudo "Snow".

Par exemple :

- \$./main 1215 MonPseudo
- \$./main 1215

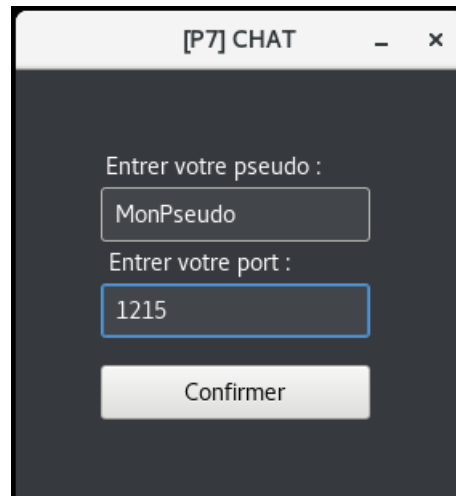
Pour l'affichage avec une interface graphique il suffit de faire :

- \$./main

sans mettre d'arguments supplémentaires.

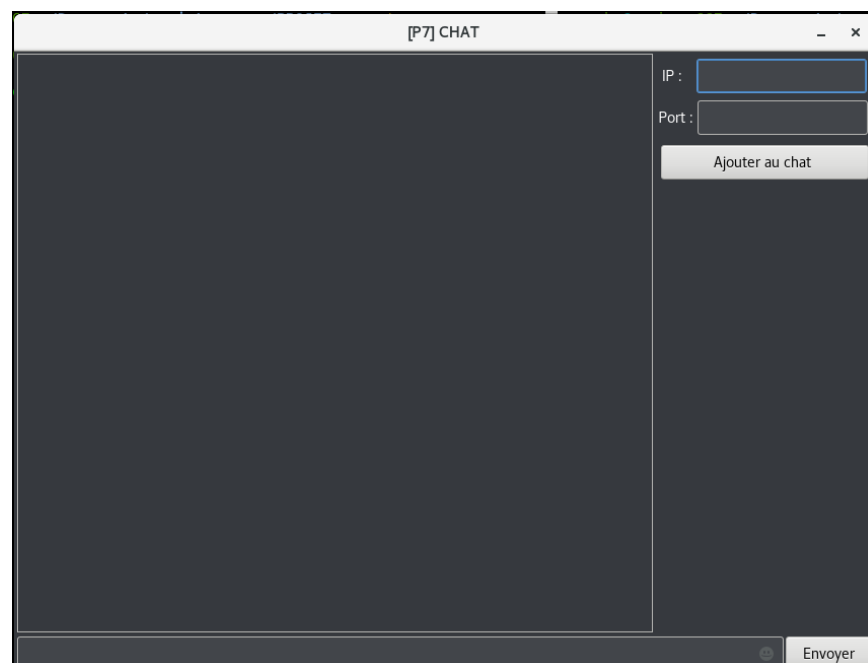
1.4 Exemple d'utilisation

Voici un exemple d'utilisation lorsqu'on lance le programme avec l'affichage graphique :



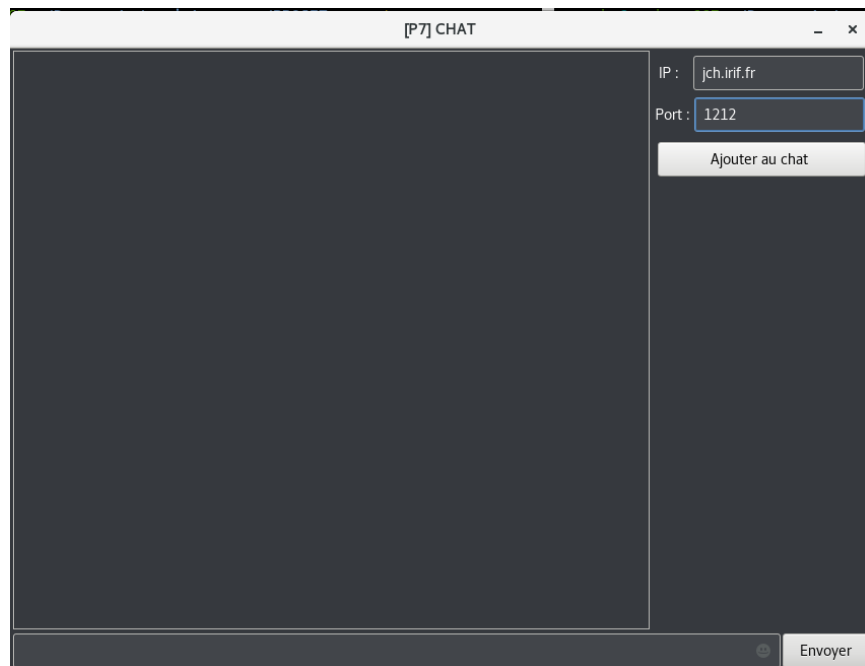
The screenshot shows a window titled "[P7] CHAT" with a dark background. It contains two text input fields. The first is labeled "Entrez votre pseudo :" and contains the text "MonPseudo". The second is labeled "Entrez votre port :" and contains the text "1215". Below these fields is a button labeled "Confirmer".

Entrez votre pseudo et le port de votre choix.

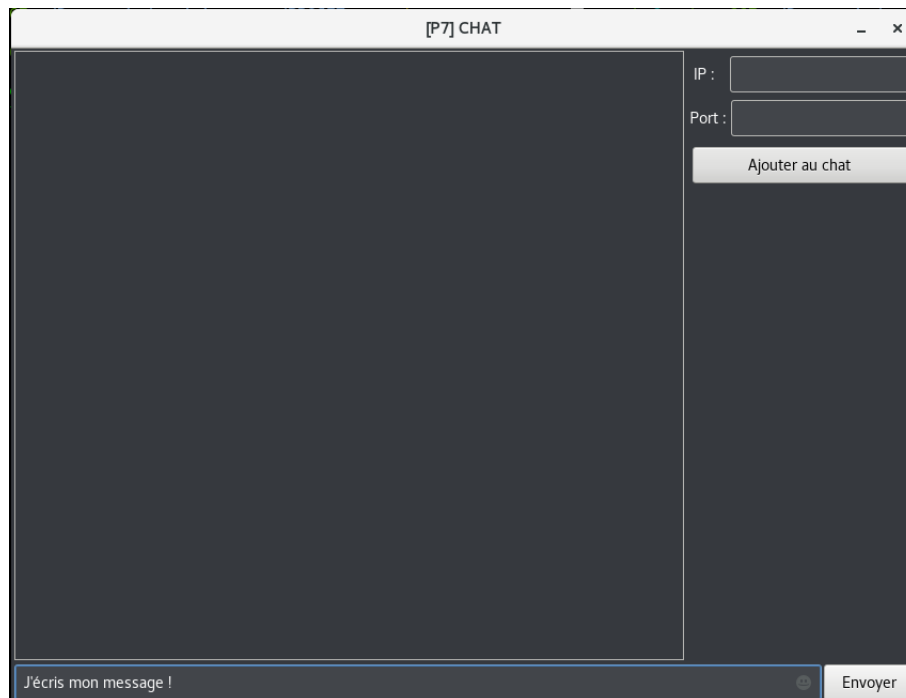


The screenshot shows the same window titled "[P7] CHAT", but now it displays a chat interface. On the right side, there are two input fields labeled "IP :" and "Port :", followed by a button labeled "Ajouter au chat". At the bottom right, there is a button labeled "Envoyer". The main area of the window is a large, empty dark rectangle, presumably for chat messages.

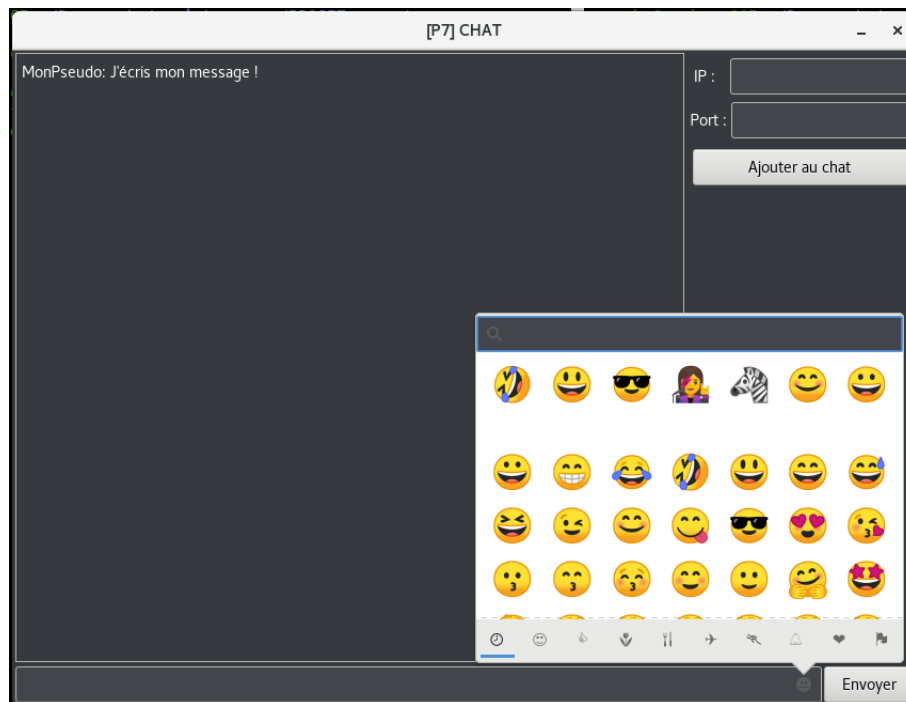
Fenêtre de chat.



Entrez l'IP et le port auquel vous souhaitez vous connecter.



Vous pouvez maintenant discuter !



Vous pouvez également envoyer des émoticônes !

2 Fonctionnalités implémentées

Nous avons implémenté les fonctionnalités du sujet minimal, c'est-à-dire :

- La réception des requêtes
- L'envoi des requêtes
- La maintenance de la liste des voisins directs
- La maintenance de la liste des voisins potentiels
- L'inondation des TLV Data
- Un affichage terminal du chat.

Puis nous avons implémenté quelques fonctionnalités optionnelles qui sont :

- L'agrégation de plusieurs TLV dans un seul message
- La sécurité face aux pairs malicieux
- Une interface graphique
- L'envoi de "gros" messages textuels.

3 Détails de l'implémentation

3.1 Strucutre général du projet

Nous avons décidé de séparer les fichiers de notre projet en plusieurs sections :

- interface : Dossier où se situe tout ce qui est relatif à l'interface graphique
- library : Dossier où se situent toutes les structures de données utiles au bon déroulement du programme
- network : Dossier où se situent toutes les fonctions relatives au réseau
- threads : Dossier où se situent les fonctions permettant le lancement des différents threads.

Concernant les threads nous lançons l'exécution de 3 threads :

- Un pour la réception et la gestion des requêtes.
- Un pour l'inondation et le nettoyage des structures de données (liste de voisins directs et potentiels)
- Un pour écouter l'entrée standard (afin d'envoyer des messages).

Ce choix est motivé par l'envie d'exécuter en parallèle la réception des requêtes et l'inondation qui sont deux tâches bien distinctes. Il a donc fallu mettre des "cadenas" sur les structures partagées entre les threads pour éviter les problèmes.

3.2 Inondation

3.2.1 Structures de données utilisées

Nous stockons les messages reçus dans un tableau de 600 cases (car un gros message peut être fragmenter en 282 TLV), lorsqu'il est rempli nous le re-remplissons à partir de la première case.

Ce choix permet à notre serveur de ne pas être débordé et d'automatiquement libérer la mémoire des messages "anciens".

Un élément de ce tableau contient

- les informations du TLV Data à inonder (ID,nonce,type,taille,contenu)
- une hashmap des voisins symétriques n'ayant pas encore acquitté,

nous retrouvons aussi dans la hashmap la date du dernier envoi et le nombre d'envois du TLV Data en question.

3.2.2 Trame

L'inondation d'un TLV Data à un voisin se fait toute les $\frac{2^n + 2^{n-1}}{2}$ secondes, avec n le nombre d'envoi déjà effectuer. (On à fait le choix d'un temps non aléatoire afin de ne pas désynchroniser les messages envoyés en même temps ce qui facilité l'agrégation).

Nous enlevons le voisin de la hashmap associée à un message lors d'un acquittement ou un 5-ème envoi.

Le choix d'une hashmap nous permet de vérifier si un voisin identifier par un couple (IP,port) doit être inondé en tant constant ce qui est plus rapide lors de l'envois d'une requête.

3.3 Envoie des requêtes

Pour l'envoi des requêtes, on concatène au maximum (dans la mesure du possible) les TLV que l'on doit envoyer à un voisin, pour ce faire on a choisi d'inonder en même temps les messages envoyés au même moment (cf. inondation) pour éviter de désynchroniser les messages envoyés au même moment.

3.4 Réception des requêtes

Lors de la réception d'une requête, on réalise une analyse de la requête c'est-à-dire que l'on vérifie tout d'abord si la taille déclarée dans la requête dépasse 4093 octets auquel cas on rejette la requête.

Ensuite pour éviter une faille dans le programme en donnant une mauvaise taille de TLV, on s'assure que la position du dernier octet du TLV est inférieure à la taille déclarée. Enfin selon le type du TLV, on réalise l'action en conséquence.

3.5 Maintenance de la liste des voisins directs

3.5.1 Structures utilisées

Nous maintenons une hashmap de voisins symétrique dans laquelle un voisin contient les informations suivantes :

- le couple (IP,port) qui identifie ce dernier
- la date de réception du dernier TLV Hello Long
- la date du dernier TLV Neighbour transmit
- la date du dernier TLV Hello Long transmit

3.5.2 Trame

Nous envoyons à un voisin un TLV Hello Long et un TLV Neighbours toutes les 10 secondes.

Un voisin symétrique qui n'a pas envoyé de TLV Hello long depuis trop longtemps sera déplacé dans la liste des voisins potentiels.

3.6 Maintenance de la liste des voisins potentiels

3.6.1 Structures utilisées

Nous maintenons une liste de voisins potentiels dans laquelle un voisin contient les informations suivantes :

- (IP,port) identifiant ce voisin
- la date du premier TLV Hello short transmit
- la date du dernier TLV Hello Short transmit

3.6.2 Trame

Nous envoyons à un voisin potentiel un TLV Hello short toutes les 10 secondes et s'il ne répond pas pendant trop longtemps nous le supprimons de la liste des voisins potentiels

3.7 Envoie des "gros" messages

On appelle message global un "gros" message que l'on souhaite envoyer et sous-message un fragment du message global.

Pour envoyer un gros message, on fragmente le message global en sous-message, ces sous-messages seront envoyés dans des TLV Data de type 220 (il s'agit du type prévu à cet effet pour plus de détails voir la mailing list).

Pour la réception on stocke les TLV Data de type 220 dans un tableau circulaire dont une case est une liste de sous-messages lorsque l'on a reçu tous les fragments on affiche le message.

4 Conclusion

Nous avons pour but de créer un jeu en réseau, et ce sujet a été pour nous un excellent moyen de nous donner des idées pour mettre en place notre propre protocole de discussion (chat) que l'on va intégrer à notre jeu.