Projet C
Simulation d'un commutateur de niveau 3
Rapport

Philippe Tran Ba, Élie Bouttier, Jiajun Shi, Émilie Abia
Groupe 15

12 juin 2012

Résumé

Ce rapport porte sur un projet de programmation d'un commutateur de niveau 3 en langage C. Il y est abordé le fonctionnement global du programme, suivi des choix techniques effectués afin de palier aux problèmes rencontrés. Il termine par une conclusion sur l'état du projet et des améliorations qui pourraient y être apportées.

Table des matières

1 Introduction **2**
1.1 Préface 2
1.2 Rappel du sujet 2

2 Implémentation **2**
2.1 Architecture de l'application en module 2
2.2 Détail des types et sous programmes 4
2.3 Présentation des principaux algorithmes 4

3 Réalisation du projet **4**
3.1 Difficultés et solutions 4
3.1.1 Solutions envisageable 4
3.1.2 Difficultés rencontrées 5
3.1.3 Solution retenue 5

4 Conclusion **5**

1 Introduction

1.1 Préface

Dans le cadre de notre préparation au diplôme d'ingénieur en Télécommunications et Réseaux, nous avons été amenés à effectuer notre projet de première année sous l'encadrement de Jérôme Ermont. Le but de ce projet était surtout de nous faire manipuler le langage C et la gestion de la mémoire mais il nous a également permis de modéliser une solution réseau simple.

La commutation de paquets, technique utilisée dans le transfert de données dans les réseaux informatiques, permet de rediriger les packets sur un lien physique particulier suivant des critères précis, avec éventuellement une modification de ceux-ci. Au vue de notre formation, le choix de la réalisation d'un simulateur de commutateur se trouve être pertinent et intéressant.

Ce rapport est composé de deux parties. La première partie présente l'application dans sa globalité, son fonctionnement. La seconde partie est consacrée au détail de l'implémentation. Enfin nous conclurons sur les apports personnels obtenus à la fin de ce projet.

1.2 Rappel du sujet

Nous avons effectué notre projet langage C de première année sous l'encadrement de Jérôme Ermont. Il était question principalement de simuler le fonctionnement d'un commutateur de niveau 3.

Ainsi il s'agissait dans ce projet de simuler le travail d'un élément réseau dont le rôle serait de recevoir des trames, de les assembler pour reformer des paquets et d'effectuer par la suite la commutation de ces paquets. Les paquets routés sont à nouveau fragmentés et les trames obtenues sont stockées dans des queues selon leur priorité.

La première étape a été l'étude préalable de l'élément réseau impliqué, il s'agissait d'assimiler les fonctionnalités attendues de celui-ci et de, relativement à notre compréhension du sujet, réfléchir à un algorithme.

Dans un second temps, après nous être mis d'accord sur le modèle retenu nous nous sommes départagé le travail afin de gagner du temps.

Nous avons ensuite regroupé notre travail et avons modifié la structure générale afin de l'optimiser au maximum.

Enfin, il faut remarquer que notre application est vouée à évoluer : il serait par exemple souhaitable que le commutateur soit capable de mieux traiter les erreur (reprise sur erreur, renvoie...)

2 Implémentation

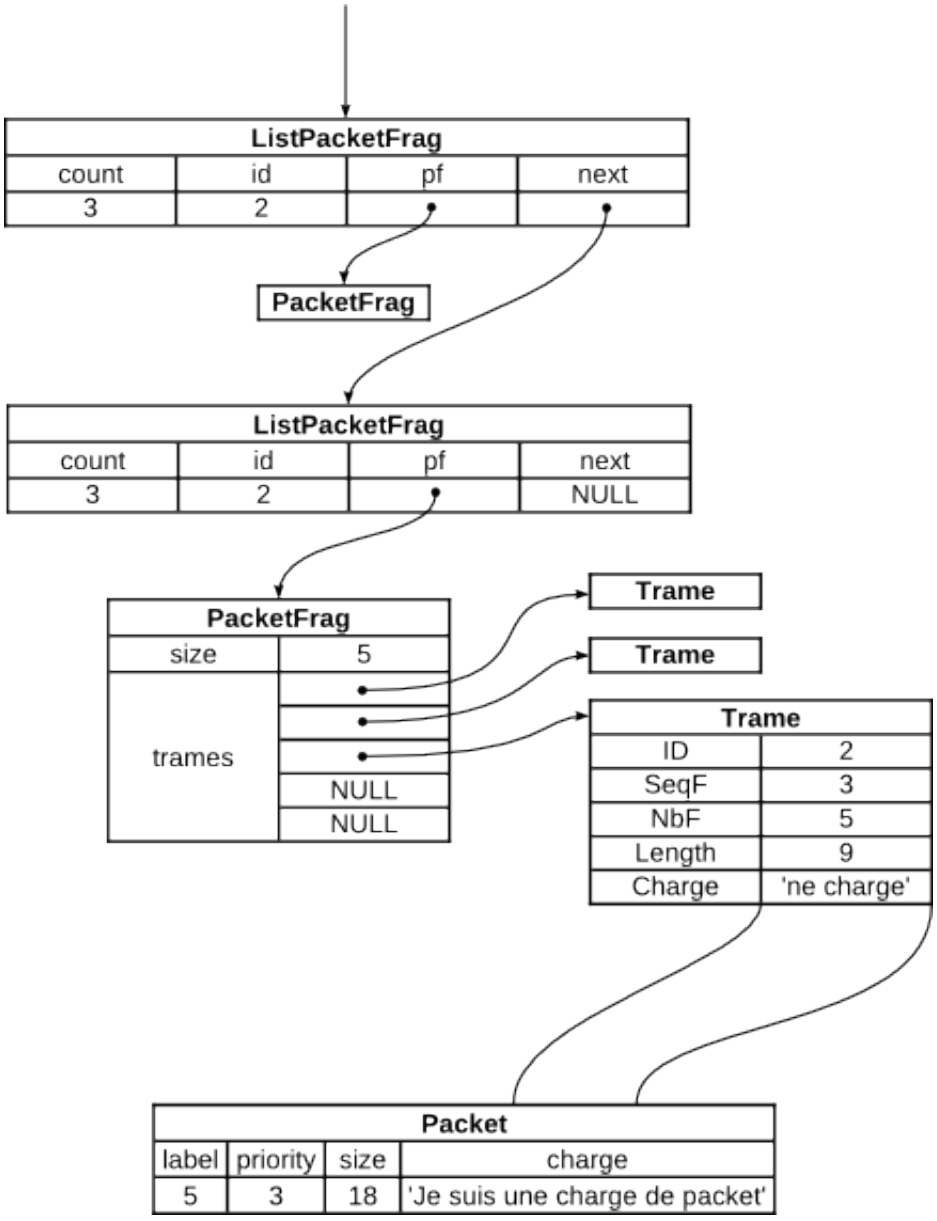
2.1 Architecture globale de l'application

A la vue des fonctionnalités énoncées ci-dessus, notre solution a été conçue pour être évolutive (car décomposée en modules distinct) et stable. L'application exécutable se décompose en modules fonctionnels explicités par le schéma ci-dessous :

Notre application ayant attrait au réseau, il est naturel de la décomposer en module indépendant à l'image des couches du modèle OSI.

Le simulateur est lancé par la méthode `simulate` du module `simulator`. Les paramètres de lancement sont obtenue par le module `main`, en charge d'analyser les arguments de la ligne de commande. La simulation est commandé par le fichier d'entrée, comportant des instructions `IN` et des instruction `out`, dictant l'entrée ou la sortie d'un packet.

Lors de la lecture d'une instruction `IN`, le module `trame` sera appelé afin de lire une trame depuis le fichier d'entrée. Celle-ci est alors rangée dans une structure de données nommé `ListPacketFrag`, en attandant d'obtenir toute les trames d'un même packet.


ListPacketFrag
count
id
pf
next
3
2
PacketFrag
ListPacketFrag
count
id
pf
next
3
2
NULL
PacketFrag
size
5
trames
NULL
NULL
Trame
Trame
Trame
ID
2
SeqF
3
NbF
5
Length
9
Charge
'ne charge'
Packet
label
priority
size
charge
5
3
18
'Je suis une charge de packet'

2.2 Détail des types et sous programmes

MODULE	TYPES	SOUS-PROGRAMMES
trame	Trame	trame_init trame_destroy trame_decode trame_encode trame_print
packet	Packet	packet_init packet_destroy packet_assemble packet_split packet_print
packetfrag	PacketFrag	lpf_init lpf_add lpf_del lpf_destroy lpf_print
ListPacketFrag	ListPacketFrag	A faire
queue	Queue WRRS	wrrs_init wrrs_push wrrs_pop wrrs _print
commut	ListRule	commut _init commut _load commut _commut commut _print
simulator	∅	simulate

2.3 Présentation des principaux algorithmes

3 Réalisation du projet

3.1 Difficultés et solutions

Au départ, le stockage dans l'ordre des trames semblait bien difficile. Un tableau dynamique n'était pas simple à manipuler mais nous ne pouvions pas mettre de tableau fixe. Nous avons réalisé que lorsque l'on recevait une trame, on savait directement le nombre de trame que l'on recevrait grâce à l'attribut nbf. Ainsi il semblait bien plus simple d'allouer directement un tableau de la bonne taille et ranger les trames dans le bon ordre.

3.1.1 Solutions envisageable

Plusieurs solutions ont été proposées lors de l'étude du cahier des charges. Dans la mesure où il nous était demandé de ne pas nous focaliser sur les performances mais plutôt sur le choix des structures de données et l'efficacité des algorithmes.

Au départ nous comptions gérer les trames dans une partie décodage, gérer les paquets dans une partie assemblage et gérer la queue dans une partie éponyme.

Dans la partie décodage nous avions défini une structure de données Trame, une liste chaînée de trames utilisée pour stocker les trames d'un même paquet qui n'est pas encore complet et une liste de « Paquets Fragmentés » ListePacketFrag utilisée pour stocker les différentes listes de trame.

Nous avions alors envisagé que les opérations disponibles sur une trame pourraient être : Créer une trame à partir d'un numéro de paquet, d'un numéro de séquence, d'un nombre de séquence, de la longueur d'une charge, d'une somme de contrôle et d'un contenu.

1. Initialiser une trame

2. Transformer une séquence de bits en trame
3. Transformer une trame en une séquence de bits
4. Vérifier si un ID existe
5. Ajouter une trame dans la ListeTrame
6. Ajouter une ListeTrame dans la ListePacketFrag
7. Vérifier si la ListeTrame est complète
8. Vérifier si la Trame est correcte en utilisant CRC
9. Calculer CRC

C'est dans la partie assemblage que nous avions défini la structure Packet et les opérations disponibles sur un paquet, la plus importante étant bien sûr le décodage de la trame dans le but de l'intégrer à un paquet.

Puis dans la partie queue nous avions envisagé que la queue serait un tableau de taille 256, soit le nombre maximal de priorité.

3.1.2 Difficultés rencontrées

3.1.3 Solution retenue

Ainsi, partant de la volonté de développer une application efficace, nous avons effectué grand nombre de modifications. La plus significative concernant la structure de donnée qui stocke les trames lorsqu'un paquet n'est pas encore complet. En effet, nous sommes passés d'une liste de liste à une liste de tableau.

Un second regard nous a permis de réaliser que lorsque l'on recevait une trame, on savait directement le nombre de trame que l'on recevrait grâce à l'attribut nbf. Ainsi il semblait bien plus simple d'allouer directement un tableau de la bonne taille et ranger les trames dans le bon ordre.

4 Conclusion

Pour mener à bien ce projet de première année, nous avons dû approfondir nos connaissances en terme d'analyse de trame. Nous nous sommes aussi documenté sur le fonctionnement des protocoles TCP/IP afin de s'en inspirer.

De plus, ce projet nous a permis de nous familiariser avec la démarche de création d'un programme complexe ainsi que des notions vues en réseaux. En effet, nous avons développé cette application par un travail de groupe, ce qui nous attend en tant que futurs ingénieurs. Plusieurs personnes travaillant sur un même programme ont besoin d'énormément de coordination et de synchronisation. Cela s'est révélé bien plus difficile que prévu, outre la standardisation de nos variables et la répartition des modules.

La partie que nous avons développée correspond aux objectifs de départ. Les résultats de simulation sont tout à fait satisfaisants tant au niveau routage qu'au niveau gestion de la mémoire. Savoir faire des simulation et le faire tout le long du projet est un outil précieux afin de corriger les erreurs le plus vite possible avant que celles-ci ne se répercutent sur la suite.

Le projet que nous avons réalisé cette année peut encore évoluer. En effet, il subsiste un problème lorsqu'une trame arrive puis une des trames du même paquets n'arrive pas. La trame est gardée. Nous pourrions ajouter un compte à rebours qui détruit la trame si une autre trame du même paquet n'arrive pas.