Rapport
Travaux Pratiques Réseaux (Ethernet)

Quentin Tonneau - Adrien Lardenois

Table des matières

1 Introduction

Après avoir mis en réseau un parc de pc (cf rapport tp n° 1), nous nous intéressons maintenant à la communication entre ces derniers. En s'appuyant sur le cours de réseaux Ethernet et nos connaissances en langage C dans l'environnement Linux, nous allons concevoir une suite d'applications destinées à interagir avec les PC voisins. Une bibliothèque de création et d'envoi de trames, ainsi qu'un sniffer de réseau (affiche l'ensemble des trames circulant au voisinage de notre matériel) nous sont fournis afin de franchir les couches non étudiées en classe (4,5,6 et 7).

1.1 Écouter le réseau

Avant de communiquer sur un réseau, il nous faut un certain nombre d'informations sur le matériel avec lequel nous souhaitons "entrer en discussion", c'est à dire :

- Le nombre de personnes présentes sur le réseau
- Leurs adresses
- Le protocole de communication
- La nature des messages (émetteur, destinataire, type de message)

Pour cela, nous concevons un programme qui filtre les trames circulant sur le réseau, en ne conservant que les trames de type 9000 dont nous sommes le destinataire, puis affiche le message en question,tout en dressant une liste de toutes les machines (adresses MAC [1]) présentes sur le réseau. On pourrait associer ce programme à un module de conversation type IRC [2], qui n'affiche que les messages personnels ou à destination de l'ensemble des utilisateurs.

1.2 Envoyer un message

Après avoir pris connaissance du matériel voisin, nous écrivons une fonction d'envoi de trame simple, ainsi qu'un algorithme d'envoi en boucle d'un "bonjour" sur le réseau (broadcast). Le type choisi pour ces échanges est toujours 9000, afin de pouvoir être facilement reçu par les autres groupes du projet. Nous construisons la trame nécessaire, et l'envoyons à l'aide des bibliothèques fournies, et du programme write_eth_frame.

1.3 Accuser la réception d'un message

L'unification des deux parties précédentes nous permet de programmer un répondeur automatique, qui en recevant une trame de type 9000 sous la forme "Bonjour de XX" répond immédiatement à son émetteur "Bonjour de XX : reçu par Quentin et Adrien", accusant ainsi la réception du message. Nous ré-implémentons l'analyse de la première partie, ainsi que nos connaissances en matière de traitement de la langue pour détecter les messages qui nous sont destinés, et nous utilisons la fonction d'envoi de trame créée en deuxième partie pour répondre aux messages.

2 Structure du projet

2.1 Structure des trames

Avant de nous lancer dans l'écriture des algorithmes de ce projet, nous avons dû analyser les bibliothèques fournies, et définir la structure de notre projet. La structure d'une trame est imposée, sous la forme :

```
struct eth_frame {
char adr_dest[6];
char adr_send[6];
char type[2];
char data[1500 - 6 - 6 - 2];
```

1. Media access control
2. Internet Relay Tchat

};

On remarque donc que notre trame sera définie par deux adresses (destinataire et émetteur) MAC au format hexadécimal (6 octets → 12 caractères), d'un type (même encodage), et de données, n'excédant pas $1500 - 6 - 6 - 2 = 1486$ caractères. Une structure étant allouée de manière contiguë en mémoire, l'accès à l'adresse de l'émetteur pourra se faire sous la forme trame→adr_send ou bien (char*)trame[6] .

2.2 Arborescence du projet

Afin de faciliter la compréhension du projet, l'archive fournie est organisée de la sorte :
- Un fichier de compilation (script sh) se trouve à la racine
- Les fichiers sources (.c et .h) utiles au projet sont dans un répertoire **src**
- Une fois compilés, les 5 programmes se trouvent dans le répertoire **bin**

Liste et description des fichiers :
- **compilation.sh** : Fichier de Compilation (voir partie suivante)
- src
  + **envoyer_trame .c/.h** : Fichier source de l'envoi de trame (bonjour)
  + **eth_lib .c/.h** : Bibliothèque d'utilisation du réseau fournie
  + **fonctions .c/.h** : Bibliothèque de fonctions communes aux trois principaux programmes
  + **inet_str.h** : Structure des trames, nécessaire à la compilation de eth_lib
  + **recevoir_trame .c/.h** : Programme d'affichage des trames circulant sur le réseau
  + **reponse_auto .c/.h** : Programme de réponse automatique aux messages reçus (accusé de réception)
  + **sniff.c** : Programme de sniff du réseau (fourni)
  + **write_eth_frame.c** : Programme permettant l'envoi de la trame donnée en premier paramètre (fourni)
- bin
  + **envois** : programme d'envoi des "bonjour"
  + **recevoir** : programme d'affichage des trames
  + **reponse** : programme d'accusé de réception
  + **sniff** : programme utilisé pour l'exécution de **recevoir** et **reponse**
  + **write_eth_frame** : programme appelé par **envois** et **reponse**

2.3 Compilation et mode d'emploi

2.3.1 Définition des variables avant compilation

Afin de compiler les différents algorithmes selon nos besoins, quelques variables (DEFINE) sont à préciser au début de chaque fichier source. En voici la liste :
- envoyer_trame.c
  + NOMBRE_ENVOIS 200 *Nombre de messages à envoyer*
  + TEMPS_ATTENTE 0 *Temps entre deux envois (seconde)*
  + ADRESSE "e0 :cb :4e :2f :fc :98" *Adresse par défaut de la machine*
  + MESSAGE"Bonjour de Quentin et Adrien" *Message à envoyer*
- recevoir_trame.c
  + TEMPS_AFFICHAGE 50 *Nombre de trames avant affichage de la liste des adresses*
- reponse_auto.c
  + ADRESSE "e0 :cb :4e :2f :fc :98" *Adresse par défaut de la machine*

2.3.2 Compilation

Pour compiler l'ensemble du projet, il suffit d'exécuter le fichier **compilation.sh**
Si la compilation échoue, ou pour une architecture spécifique, voici les étapes de la compilation :

```
#Cree le repertoire des programmes compiles
```

```
mkdir bin
#Compile la bibliotheque fournie (necessite inet_str.h)
gcc src/eth_lib.c -c
#Compile le logiciel d'envoi de trame
gcc src/write_eth_frame.c eth_lib.o -o bin/write_eth_frame
#Compile le sniffer
gcc src/sniff.c eth_lib.o -o bin/sniff
#Compile le programme d'envoi
gcc src/envoyer_trame.c eth_lib.o src/fonctions.c -o bin/envois
#Compile le programme d'affichage
gcc src/recevoir_trame.c eth_lib.o src/fonctions.c -o bin/recevoir
#Compile le programme de reponse auto
gcc src/reponse_auto.c eth_lib.o src/fonctions.c -o bin/reponse
#Supprime le fichier de bibliotheque genere
rm eth_lib.o
```

2.3.3 Notes de compilation

Les éventuelles erreurs dues à la compilation proviennent des bibliothèques et structures fournies par l'enseignant, mais n'impactent aucunement sur le fonctionnement des différents programmes générés. On remarque également que la compilation du sniffer sur des machines en environnement "Ubuntu 11.04 et ultérieur version 32bits" provoque des erreurs lors de l'exécution (segmentation fault). Il est donc conseillé en cas d'erreur de compilation de reprendre les exécutables générés dans l'archive du projet, ou de contacter les auteurs des sources.

2.4 Manuel d'instruction

ATTENTION : L'ensemble des exécutions doivent se faire à l'aide de droits administrateur. L'utilisation du compte root ou d'une session admin (sudo su) est obligatoire !

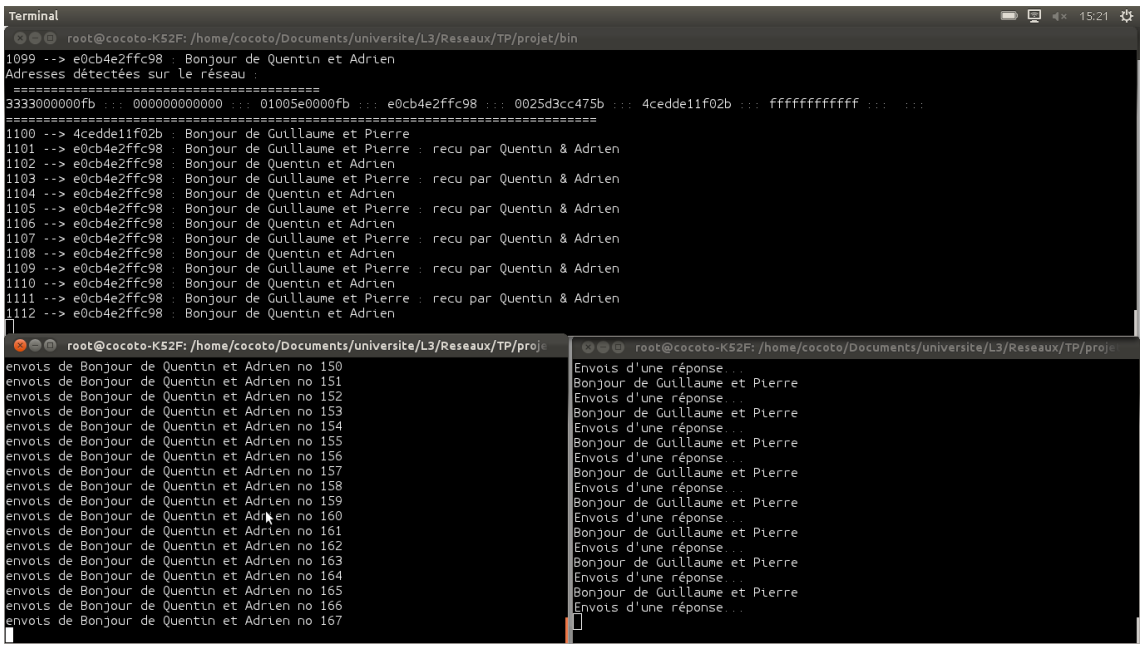
Les programmes d'envoi de trame (reponse et envois) prennent en paramètre l'adresse MAC [3] de la machine. Par défaut, l'adresse prise sera l'adresse indiquée dans les sources avant compilation (cf partie ci-dessus). Les programmes de réception et d'analyse de trame (reponse et recevoir) doivent posséder en entrée la sortie standard de sniff au moyen d'un "pipe" (|).
Selon les besoins, voici la liste des commandes d'exécution :

- **bin/sniff | bin/recevoir** : Affichage des trames (9000) qui circulent sur le réseau
- **bin/sniff | bin/reponse ADRESSEMAC** : Affiche les messages qui nous sont destinés et répond automatiquement
- **bin/envois ADRESSEMAC** : Envoie une série de trames dans un intervalle de temps (cf partie 2)

2.5 Exemple de disposition

Pour utiliser pleinement les fonctions de ce projet, il faut donc ouvrir et disposer trois terminaux. Voici un exemple obtenu en plaçant l'afficheur en haut de l'écran, et les autres programmes en bas :

3. Les lettres doivent y figurer en minuscule



3 Réception des trames

La réception de trames se fait par redirection de la sortie du sniffer vers notre programme défini par **recevoir_trame.c**, au moyen d'un pipeline. Les informations reçues sont traitées une à une par la fonction *lire_trame()*. On vérifie alors si l'adresse de destination et l'adresse source nous sont déjà connues puis, si l'un et/ou l'autre est "nouveau", on l'ajoute à notre liste des matériels connectés.

Nous avons choisi de représenter la liste des adresses rencontrées par une liste chaînée. Cela a pour avantage de permettre l'ajout d'un maillon sans connaître la taille actuelle de la liste, au contraire d'un tableau qu'il aurait fallu redimensionner au fur et à mesure.

Dans un second temps, on vérifie si la trame appartient au type défini spécialement pour le TP (9000). Si c'est le cas, on affiche à l'écran les informations suivantes :

- Numéro de la trame (au moyen d'un compteur)
- Adresse source
- Contenu de la trame

Enfin, chaque fois que Y trames ont été affichées[4], on affiche un récapitulatif des adresses qui circulent sur le réseau. Suivant le choix de Y (50 dans nos exemples) et en fonction du trafic sur le réseau, on bénéficie alors d'un affichage clair des trames qui circulent avec un bilan régulier des utilisateurs présents.

3.1 la fonction lire_trame

La fonction *lire_trame* est définie dans **fonctions.c**. Elle retire les 3 premières chaines de la sortie du sniffer, qui ne nous intéressent pas, puis récupère la longueur de la trame. Après quoi elle capte toute la trame grâce à la fonction *get_buf* et la retourne en sortie.

3.2 la fonction recherche

La fonction *recherche* parcourt la liste chaînée à partir du premier maillon et s'arrête si en passant au suivant elle arrive à NULL. Il est possible de faire le test à la fin car l'initialisation de la liste garantie l'existence d'au moins un maillon différent de NULL.

Pour chaque maillon, on compare le champ adresse à celle que l'on cherche : s'il y a correspondance on sort directement en renvoyant 1, sinon on passe au suivant. Si la recherche a eu lieu jusqu'au bout sans succès, on retourne 0.

4. Y étant défini au début du programme

3.3 la fonction ajout

La fonction d'ajout ajoute un élément en tête de liste. On crée simplement un nouvel élément que l'on défini comme l'adresse à ajouter et dont le suivant est l'ancienne tête de liste.

3.4 la fonction affiche

La fonction *affiche* parcourt la liste de la même manière que *recherche* . Pour chaque étape de la liste, elle affiche l'adresse et sépare les adresses par : : :

4 Envoi des trames (type 9000)

4.1 Fonction d'envoi

Maintenant que nous recevons les trames de type 9000, nous devons à notre tour envoyer, afin de :
- Nous identifier sur le réseau
- Échanger des informations
- Accuser la réception d'informations externes

Pour cela, une fonction nommée "envois_trame" dans le fichier fonctions.c construit la trame en fonction des paramètres :

```
//Envoie une trame sur le reseau, de type 9000
void envois_trame(char *source, char *dest, char *message)
```

où source et dest sont les adresses MAC des machines concernées au format XX :XX :XX :XX :XX :XX. Pour cela, nous avons créé une procédure convaddr qui convertit les adresses en ajoutant le caractère « : ». Ensuite, nous créons la trame à l'aide de la fonction

```
int make_ping_request(char* eth_addr_dest, char* eth_addr_send,
 char* type_frame, char* data, struct eth_frame* frame);
```

dont la valeur de retour nous indique la taille de la trame générée. Nous convertissons ensuite cette trame au format hexadécimal par la fonction char_to_charhexa et nous envoyons la trame en passant cette dernière en paramètre du programme write_eth_frame fourni dans le projet.

4.2 Programme de présentation

Afin de nous présenter une fois connectés sur le réseau, nous avons créé le programme *recevoir_trame.c* dont les paramètres suivants sont à définir avant la compilation :

```
#define NOMBRE_ENVOIS 10   //Nombre de message a envoyer
#define TEMPS_ATTENTE 1    //Temps d'attente entre deux envois (secondes)
#define ADRESSE "e0:cb:4e:2f:fc:98" //Adresse par defaut
#define MESSAGE "Bonjour de Quentin et Adrien" //Message
```

Ce programme envoie un certain nombre de fois un message d'identification sur le réseau, à l'adresse de broadcast *ff :ff :ff :ff :ff :ff*.

5 Synthèse et automatisation

La troisième et dernière partie de notre projet consiste à filtrer les messages qui nous sont adressés et à y répondre par un accusé de réception. Cela est fait par **reponse_auto.c**. Par défaut, on définit l'adresse de notre poste mais il est possible de la passer en argument.

5.1 fonctionnement global

En premier lieu, on vérifie si l'adresse est passée en paramètre, sinon on définit l'adresse à partir de l'adresse par défaut. Puis on commence à écouter les trames.

Quand on reçoit une trame, on vérifie qu'elle est de type 9000, si oui :

- On convertit les adresses source et de destination. (car les octets ne sont pas séparés par des " :" à la réception.)
- On vérifie :
  + que l'adresse de destination est la nôtre, ou celle de broadcast
  + que l'on n'est pas l'émetteur, afin d'ignorer un de nos messages qui serait broadcasté
- Si oui :
  + on vérifie que le message n'est pas déjà une réponse (présence d'un " :")
  + Si ce n'est pas déjà une réponse, on renvoie la trame vers la source en concaténant "reçu par Quentin et Adrien" à la fin de la trame.
- Sinon on passe à la suivante

5.2 fonctionnement de est_dans

On parcourt la chaîne et on retourne 1 dès que l'on trouve le caractère cherché. Si on ne trouve pas on renvoie 0.

6 Bilan

6.1 Jeux de test et exemples d'utilisation

L'ensemble des tests ci-dessous a été réalisé en collaboration avec le groupe "Pierre Roquin et Guillaume Coutable", sur des ordinateurs personnels reliés par une connexion WIFI ad-hoc.

6.1.1 Envoi de trames

Le fichier **autres/envois.txt** décrit les sorties de la console après un échange dans le sens PC_QUENTIN_ADRIEN → PC_PIERRE_GUILLAUME.

La première partie montre la sortie du programme "recevoir", on y retrouve donc nos trames envoyées, ainsi que les réponses du pc distant. Arrivé à la trame n° 50, le programme affiche la liste des adresses présentes sur le réseau, conformément aux variables de compilation :

```
00114304cb38 ::: ffffffffffff ::: 01005e000016
::: 333300000016 ::: 333300000002 ::: 01005e0000fb :::
00266c8a6869 ::: e0cb4e2ffc98 ::: 3333000000fb ::: 000000000000 ::: :::
```

On remarque la présence de nos deux adresses MAC respectives (e0cb4e2ffc98 et 00114304cb38), des adresses de broadcast et réseau, ainsi qu'une série d'adresses "parasites", que l'on justifie par la présence du réseau WIFI (autres pc souhaitant se connecter, protocoles divers).

La seconde partie nous montre la sortie du programme envoyer.

La troisième partie nous montre la sortie du programme reponse, dans lequel on peut observer uniquement les trames de réponse du PC distant.

6.1.2 Réception de trames

Le fichier **autres/reception.txt** décrit les sorties console après un échange dans le sens PC_PIERRE_GUILLAUME → PC_QUENTIN_ADRIEN.

La première partie montre la sortie du programme "recevoir", on y retrouve cette fois-ci les messages envoyés par l'autre PC, et nos réponses. Les remarques complémentaires sont identiques à la section ci-dessus

La seconde partie nous montre la sortie du programme reponse, dans lequel on voit les trames envoyées par le PC distant, et un message indiquant que le répondeur a envoyé une trame de réponse automatiquement.

6.1.3 Multi-échanges

Le fichier **autres/multi_echange.txt** décrit les sorties console après un échange dans les deux sens. Les parties sont similaires aux deux premières sections. On remarque lors de l'exécution que l'affichage apparaît subitement après une trentaine de trames.

6.2 Conclusion

L'ensemble du projet nous permet donc de lire les trames circulant sur le réseau, de répondre aux trames qui nous sont destinées, et de nous présenter aux autres machines. L'ensemble des exécutables sont utilisables sous environnement UNIX, en Wifi ou Ethernet, selon la définition des variables avant compilation.

Ce projet nous a permis d'exploiter les notions de réseau et trames Ethernet étudiées en cours, d'utiliser le support mis en place lors du tp précédent (mise en réseau des ordinateurs de la salle). De nombreuses versions de code et structure nous ont permis d'exploiter de façon optimale les chaînes de caractères en C, ainsi que l'utilisation des structures et listes-chaînées.

Nous avons rencontré des problèmes lors de la lecture de la bibliothèque fournie, car les fonctions ne sont pas toutes décrites de façon explicite, et les erreurs à la compilation nous ont amenés à revoir de nombreuses fois notre code sans comprendre leurs origines. Nous avons également essayé d'utiliser l'interception de signal (exemple Ctrl+C) afin d'afficher la liste des adresses MAC du réseau lors de la fin d'exécution. Mais devant l'instabilité de la solution, nous avons décidé d'abandonner cette option.

Comme indiqué en première partie de ce rapport, l'assemblage des trois exécutables et la présentation proposée en 2.5 nous rappellent les solutions de communication de type tchat (Exemple IRC), où chacun peut envoyer un message à destination de tous les utilisateurs, ou ne répondre qu'à la personne choisie. Il aurait été intéressant dans le cadre de ce projet de ré-exploiter les fonctions créées afin d'écrire un programme similaire, avec des trames de type 9000, et même un encodage de nos messages par diverses techniques (RSA).Par exemple, un envoi de trame sous le type XXX permet de nous identifier sur le réseau, et les trames de type 9000 concernent les échanges à proprement parlé. Une lecture plus approfondie du sniffer nous aurait également permis d'inclure ses fonctionnalités sous la forme de fonctions, et d'ainsi rendre l'exécution plus simple.