

AWADA Ahmad
BOSC Julie
EL MOUSSAOUI Habib
HALAWI Ali

TP Réseau N°2

Etude des protocoles

Décembre 2007

finsi

si événement est expiration du timer associé à une entrée
Effacement de l'entrée
finsi

si événement est réception requête (ARP Request)

La machine source émet une trame Broadcast de niveau 2 s'adressant à toutes les hôtes du réseau, comportant sa propre adresse physique et la question demandée.

finsi

si événement est réception réponse (ARP Reply)

L'hôte destinataire qui va se reconnaître va pouvoir d'un coté alimenter sa table de conversion et répondre à l'hôte source en envoyant une trame comportant son adresse physique.

finsi

fin tantque

Avant d'effectuer les manipulations, nous avons changé le nom des machines :

La machine 1 → m1

La machine 2 → m2

La machine 3 → m3

Manipulation 1 :

Nous avons tout d'abord ajouté dans la table ARP de la machine 1, une entrée pour la station de la machine 2, et inversement dans la table ARP de la machine 2.

Nous avons ensuite sur la machine 1 effectué un ping vers m2. Les paquets engendrés sont les suivants :

PING m2 (192.168.0.2): 56 data bytes

64 bytes from 192.168.0.2: icmp_seq=0 ttl=64 time=0.504 ms

64 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=0.369 ms

64 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=0.348 ms

64 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=0.343 ms

64 bytes from 192.168.0.2: icmp_seq=4 ttl=64 time=0.370 ms

64 bytes from 192.168.0.2: icmp_seq=5 ttl=64 time=0.362 ms

64 bytes from 192.168.0.2: icmp_seq=6 ttl=64 time=0.374 ms

64 bytes from 192.168.0.2: icmp_seq=7 ttl=64 time=0.382 ms

64 bytes from 192.168.0.2: icmp_seq=8 ttl=64 time=0.356 ms

64 bytes from 192.168.0.2: icmp_seq=9 ttl=64 time=0.360 ms

64 bytes from 192.168.0.2: icmp_seq=10 ttl=64 time=0.358 ms

64 bytes from 192.168.0.2: icmp_seq=11 ttl=64 time=0.354 ms

64 bytes from 192.168.0.2: icmp_seq=12 ttl=64 time=0.388 ms

64 bytes from 192.168.0.2: icmp_seq=13 ttl=64 time=0.358 ms

--- m2 ping statistics ---

14 packets transmitted, 14 packets received, 0% packet loss

round-trip min/avg/max/stddev = 0.343/0.373/0.504/0.038 ms

Nous avons ensuite supprimé l'entrée pour la station de la machine 2 dans la table ARP de la machine 1, après nous avons recommencé le ping dont voici les résultats :

```
PING m2 (192.168.0.2): 56 data bytes
64 bytes from 192.168.0.2: icmp_seq=0 ttl=64 time=0.781 ms
64 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=0.383 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=0.345 ms
64 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=0.745 ms
64 bytes from 192.168.0.2: icmp_seq=4 ttl=64 time=0.391 ms
64 bytes from 192.168.0.2: icmp_seq=5 ttl=64 time=0.359 ms
64 bytes from 192.168.0.2: icmp_seq=6 ttl=64 time=0.386 ms
64 bytes from 192.168.0.2: icmp_seq=7 ttl=64 time=0.376 ms
64 bytes from 192.168.0.2: icmp_seq=8 ttl=64 time=0.362 ms
64 bytes from 192.168.0.2: icmp_seq=9 ttl=64 time=0.371 ms
64 bytes from 192.168.0.2: icmp_seq=10 ttl=64 time=0.365 ms
64 bytes from 192.168.0.2: icmp_seq=11 ttl=64 time=0.364 ms
64 bytes from 192.168.0.2: icmp_seq=12 ttl=64 time=0.347 ms
64 bytes from 192.168.0.2: icmp_seq=13 ttl=64 time=0.365 ms
64 bytes from 192.168.0.2: icmp_seq=14 ttl=64 time=0.356 ms

--- m2 ping statistics ---
15 packets transmitted, 15 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.345/0.420/0.781/0.135 ms
```

Manipulation 2 :

Nous avons effacé les entrées puis exécuter les commandes demandées.

Manipulation 3 :

Après avoir débranché la station de la machine 3, nous avons effectué un ping, la durée du timer de rémission d'une requête ARP est d'une seconde.

Manipulation 4 :

Après avoir supprimé les entrées demandées, nous effectuons sur la machine 1, un ping vers la machine 2.

Les paquets engendrés sont les suivants :

```
PING m2 (192.168.0.2): 56 data bytes
64 bytes from 192.168.0.2: icmp_seq=0 ttl=64 time=0.826 ms
64 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=0.468 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=0.379 ms
64 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=0.381 ms
64 bytes from 192.168.0.2: icmp_seq=4 ttl=64 time=0.375 ms

--- m2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.375/0.486/0.826/0.174 ms
```

Ensuite nous avons ajouté dans la station de la machine 2 une entrée pour la station de la machine 2. Le ping de la machine 1 vers la machine 2 est re-exécuté, le contenu des tables est inchangé.

Nous avons ensuite précisé l'option pub sur la commande ARP pour l'entrée de la machine 2 sur la table ARP de la machine 2.

Lors de l'ajout de l'entrée dans la table avec l'option PUB, une requête ARP est envoyée à destination de tout le monde.

Quand nous avons refait le ping de la machine 2 depuis la machine 1, il n'y a pas de requête ARP sur le réseau, les tables sont inchangées.

Après avoir ajouté dans la table de la machine 3 une entrée pour la station de la machine 2 avec l'option pub, et supprimé la machine 2 de la table de la machine 1, nous avons effectué un ping vers la machine 2 depuis la machine 1. Trois paquets ARP sont envoyés, un de la machine 1 vers tout le monde. Deux requêtes ARP de la machine 2 vers la machine 1.

2) Le protocole ICMP

Nous avons envoyé un fichier avec des valeurs quelconques, et nous n'avons obtenu aucune réponse, Ethereal nous répond : « **unknown ICMP (obsolete or malformed)** ».

Nous avons recommencé avec un fichier commençant par : x08 x00 avec l'identifiant : 0x5103 et la sequence number : 0x006b

Les champs identifiant et sequence number sont utilisés par l'émetteur et permettent le contrôle des réponses.

Lorsque l'on réalise une analyse hexadécimale, on remarque que l'émetteur initialise le champ Sequence Number à 0, puis à chaque paquet icmp envoyé, il passera à 256.

Le champ identifiant quand à lui sera initialiser avec une valeur qui restera fixe pendant les échanges.

L'algorithme de checksum se présente de la manière suivante :

```
/* Hypothèse : short = 16 bits  
               int = 32 bits  
               long = 64 bits */
```

```
unsigned short checksum (unsigned short *buf, int taille)
```

```
{  
    /* la variable somme contiendra la somme de tous les mots du tampon buf, on le déclare  
       unsigned long pour être sûr de pouvoir tout stocker */
```

```
    unsigned long somme;
```

```
    /* Calcul de la somme */
```

```

    for( somme = 0; taille > 0; taille--)
        somme += *buf++;

        /* somme & 0xffff = les 16 bits de poids faible */
        /* somme >> 16 = somme décalé de 16 bits, équivalent aux bits de poids
fort restants */
        somme = (somme >> 16) + (somme & 0xffff);

        /* au cas où les bits de poids fort restants dépassent 16 bits, il faut répéter
l'opération. Comme taille est codé sur 32 bits on sait qu'après, il n'y en a plus */

        somme += (somme >> 16);

        /* retour du complément à 1 */
        return ~(somme & 0xFFFF);
}

```

Ce type de vérification permet de détecter une erreur à la fois, puisqu'il est basé sur le principe de bit de parité. Cela ne permet pas de détecter les cas où il y aurait deux erreurs en même temps. Imaginons qu'il y ait deux erreurs avec un bit à 1 qui se met à 0 et un autre bit à 0 qui se met à 1, l'erreur ne sera pas détectée.

Les données du paquet sont additionnées avec la valeur du checksum, complémentée à 1. Ce qui permet de détecter les erreurs de parités.

II- Etude des protocoles de niveau 4

1) Le protocole UDP

La commande "sock udp" puis "sock" donne:

la machine m2: Socket UDP créée: Id=3, port=53392

la machine m3: Socket UDP créée: Id=3, port=57932

la machine m2 tape:

sendto 3 m3 57932

Message ?: salut

la machine m3 tape:

recvfrom 3 1024

Un message de 5 octets(s) a été reçu de la machine m2 (53392)

- L'entête UDP sert à déterminer le port de la socket source, de la socket destinataire, la longueur totale du packet UDP, et le checksum pour le contrôle d'erreur.
- Les infos passées à IP sont l'entête + les données

En permutant, on obtient le même résultat.

VARIANTES :

Si on fait la réception avant l'émission des données il ne se passe rien.

Si on envoie quatre paquets, il faut refaire quatre fois la réception pour tous les recevoir un par un

Quand on croise l'émission des paquets, les messages arrivent quand même à destination.

Quand on envoie un paquet de 10 et qu'on reçoit 5 on ne reçoit pas la suite des données : elles sont perdues.

Il y a juste une requête ARP mais le paquet UDP n'est pas réellement envoyé.

Lorsqu'on envoie des paquets de 5000 octets six fois, les paquets sont fractionnés par le protocole IP. De plus, il ne sont pas tous réceptionnés (4/6)

Le paquet UDP est envoyé car la socket existe mais comme le numéro de port n'existe pas, un message ICMP revient à la station source avec l'info "*port unreachable*"

Fonctionnement UDP : voir net

Le paquet UDP est encapsulé dans un paquet IP. Il comporte un en-tête suivi des données proprement dites à transporter.

En-tête IP En-tête UDP Données

L'en-tête (*header* en anglais) d'un datagramme UDP est bien plus simple que celui de TCP :

Port Source (16 bits)	Port Destination (16 bits)
Longueur (16 bits)	Somme de contrôle (16 bits)
Données (longueur variable)	

2) Le protocole TCP

a- Etablissement d'une connexion

Création de la socket :

```
socklab-TCP> passive
Socket TCP creee: id=4, port=55718
```

Une socket passive en écoute (syn) et une active qui envoie le message et qui fait une réception (syn, ack).

Packet generé : syn, syn + ack, ack, psh + ack, ack

Au moment de l'« accept », il ne se passe rien. Au lieu de faire le « connect », il y a création d'une nouvelle socket sur le poste avec la socket passive

Si on fait le « accept » apres le « connect », la connexion est quand même établie :

Un appel de la machine m3 (61769) a été intercepte.

La connexion est établie sous l'identificateur 13.

Id	Proto	Adresse	Connexion	RWX ?
3	TCP	*(63961)	-	...
4	TCP	*(55718)	-	R..
5	TCP	*(61204)	-	...
6	TCP	*(60042)	-	...
7	TCP	*(55542)	-	...
8	TCP	*(60013)	-	...
9	TCP	m2(60013)	m3(49761)	.W.
10	TCP	*(61122)	-	...
11	TCP	m2(61122)	m3(55526)	.W.
12	TCP	*(60625)	-	...
>13	TCP	m2(60625)	m3(61769)	.W.

Flag SYN:

L'établissement d'une connexion TCP suit un protocole strict :

- Une requête de synchronisation [SYN] de la part de l'initiateur du dialogue (le client),
- une réponse d'accusé réception de la synchronisation [SYN,ACK] de la part du serveur,
- un accusé réception du client [ACK]

Si on fait plusieurs connexions sur un même port, les deux machines qui créent les connexions ont le message « connexion réussie ».

En revanche, si on fait un « status » sur la passive, on voit que une seule socket s'est créer avec le poste qui a fait le socket en premier.

Si on fait un « connect » vers un port inexistant : une requête tcp est envoyée avec un SYN mais on à le message « connection refuse ».

b- Libération d'une connexion

Le flag [FIN] sert à interrompre la connexion.

Le champ spécifique à cette phase de la communication est le champ 'flags'.
Selon la nature des messages envoyés, il peut prendre les valeurs suivantes :

[FIN, ACK] : 0001 0001
[ACK] : 0001 0000

Le client met fin à une connexion avec le flag [FIN] une fois seulement que la seconde connexion est établie.

La commande shutdown ne coupe pas la connexion, mais elle empêche la réception ou l'émission des données depuis la machine sur laquelle elle a été exécutée (mais les paquets émis par une machine distante circulent quand même sur le réseau).

La commande « close » coupe brutalement et définitivement la connexion, plus aucun transfert de données n'est possible entre les deux machines via ces sockets.

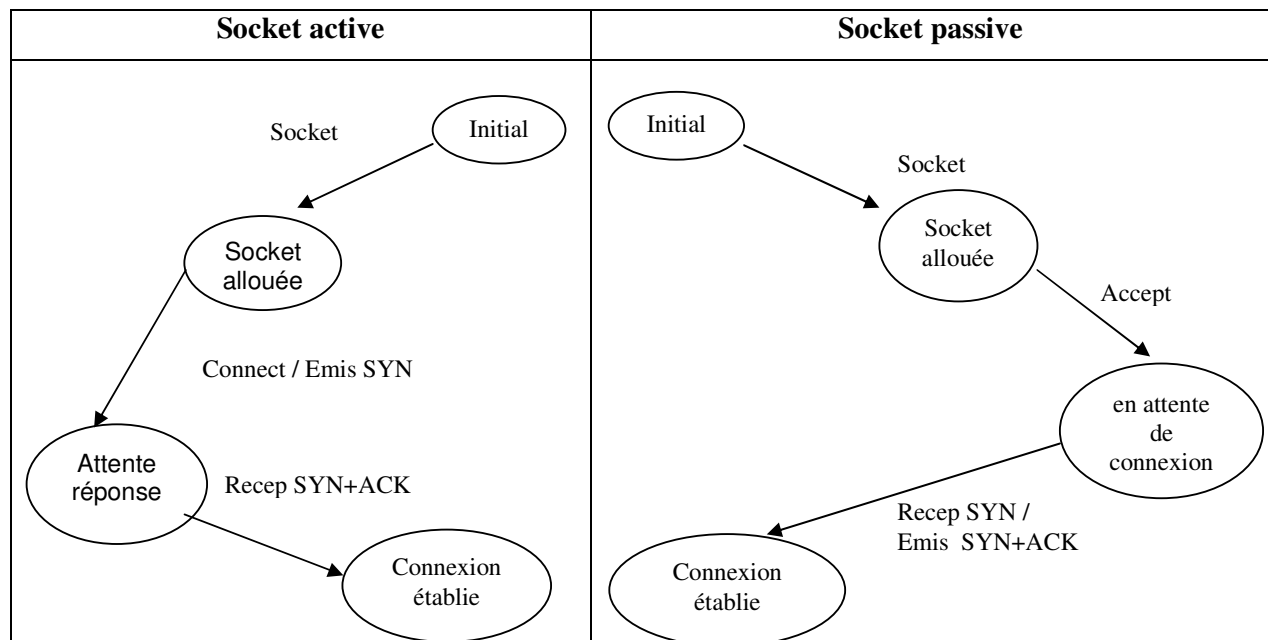
L'avantage de la commande shutdown est qu'elle permet d'empêcher le transfert de toute donnée dans seulement l'un des deux sens choisi. Par exemple, lorsque l'on veut terminer une connexion, mais que la machine est encore en train de recevoir des données depuis une machine distante, on peut empêcher notre machine d'émettre et continuer à recevoir jusqu'à ce que la machine distante ait fini d'émettre avant de fermer complètement la connexion.

c- Etude du séquençement et du contrôle d'erreur

Paquet engendré :

un tcp avec PSH et ACK voir net pour psh
un tcp ACK est renvoyé

Etablissement de la connexion :



On obtient le message « tcp retransmission » lorsqu'on envoie vers une machine débranché.

Dans un segment TCP, il y a un champ FLAG sur 6 bits. Il y a différentes valeurs possibles pour le FLAG :

URG: si ce drapeau est à 1 le paquet doit être traité de façon urgente.

ACK: si ce drapeau est à 1 le paquet est un accusé de réception.

PSH (PUSH): si ce drapeau est à 1, le paquet fonctionne suivant la méthode PUSH.

RST: si ce drapeau est à 1, la connexion est réinitialisée.

SYN: Le Flag TCP SYN indique une demande d'établissement de connexion.

FIN: si ce drapeau est à 1 la connexion s'interrompt.

Dans le cas de l'envoi de données urgentes, le flag URG est à 1. Cela signifie que le paquet doit être traité comme une donnée urgente.

De plus, il y a aussi un champ URGENT POINTER, qui indique le numéro de séquence à partir duquel commence les données deviennent urgentes.

Le mécanisme des données urgentes permet d'envoyer des données qui échappent au contrôle de flux. L'intérêt est de pouvoir réaliser une communication rapide entre deux applications, en effet, le contrôle de flux représente une perte de temps de quelques ms à chaque fois. Si il n'y a pas de contrôle de flux, les données peuvent circuler plus « librement », rapidement.

Ce mécanisme peut être utile dans le cas des applications qui nécessitent une faible latence, applications en temps réel, téléphonie, vidéoconférence...

d- Contrôle de flux

Taille buffer réception : 66608 octets

Taille buffer émission : 33304 octets

Le champ « window » correspond au nombre d'octet à partir de la position marquée dans l'acquittement que le récepteur est capable de recevoir. Le destinataire ne doit donc pas envoyer les paquets après numéro de séquence+window

III- Etude de la fragmentation des paquets IP

TCP ne fragmente pas avec IP contrairement à UDP

La taille d'un datagramme maximal est de 65536 octets. Toutefois cette valeur n'est jamais atteinte car les réseaux n'ont pas une capacité suffisante pour envoyer de si gros paquets. De plus, les réseaux sur Internet utilisent différentes technologies, si bien que la taille maximale d'un datagramme varie suivant le type de réseau. C'est pourquoi dans certains cas, la fragmentation des datagrammes IP est utile.

Pour tenir compte de la fragmentation, chaque datagramme possède plusieurs champs permettant leur réassemblage:

champ identification, FRAGMENT OFFSET (16 bits) : numéro attribué à chaque fragment afin de permettre leur réassemblage.

champ longueur totale (16 bits), **TOTAL LENGTH** : il est recalculé pour chaque fragment.

champ drapeau (3 bits) : il est composé de trois bits:

Le premier n'est pas utilisé.

Le second (appelé **DF : Don't Fragment**) indique si le datagramme peut être fragmenté ou non. Si jamais un datagramme a ce bit positionné à un et que le routeur ne peut pas l'acheminer sans le fragmenter, alors le datagramme est rejeté avec un message d'erreur.

Le dernier (appelé **MF : More Fragments**, en français Fragments à suivre) indique si le datagramme est un fragment de donnée (1). Si l'indicateur est à zéro, cela indique que le fragment est le dernier (donc que le routeur devrait être en possession de tous les fragments précédents) ou bien que le datagramme n'a pas fait l'objet d'une fragmentation

IV- Exercice de synthèse

1) Observation de la commande « talk »

Cette commande effectue la connexion avec le protocole UDP
Ensuite les messages sont envoyés par le protocole TCP.
Les messages sont fragmentés et envoyés octet par octet.
Les caractères blancs occupent des octets.

2) Observation de la commande « rlogin »

Cette commande permet d'établir des connexions à distance sur d'autres machines à partir de sa machine locale.

Sans argument autre que le nom de la machine distante, rlogin permet d'établir une connexion vers cette machine en utilisant le nom de l'utilisateur local comme nom de login.
Ainsi l'utilisateur Julie travaillant sur la machine m1 pourra se connecter sous son nom sur la machine distante m2, à partir de la machine m1, avec rlogin à la seule condition qu'il soit enregistré sur la machine m2. S'il n'a pas de compte sur la machine m2 la requête est refusée.

Par défaut l'utilisateur devra donner son mot de passe sur la machine distante pour être authentifié.

Cependant ce n'est pas toujours obligatoire comme expliqué ci-après :

Si l'une des conditions suivante est vérifiée, l'utilisateur n'a pas à donner son mot de passe :

- la machine distante possède un fichier /etc/hosts.equiv qui contient le nom de la machine locale
- l'utilisateur possède un fichier personnel .rhosts dans son répertoire d'accueil distant et ce fichier contient le nom de la machine locale.

Ces fichiers sont très insécuritaires puisqu'ils peuvent permettre des connexions sans mot de passe. Le fichier /etc/hosts.equiv est presque toujours inhibé. Chaque utilisateur peut néanmoins posséder son fichier .rhosts pouvant même autoriser la connexion à partir de noms d'utilisateurs différents.

Si l'utilisateur Julie possède un fichier .rhosts sur la machine m2, et que ce fichier contienne la ligne m1 Ahmad, alors l'utilisateur Ahmad de la machine m1 peut se connecter sans mot de passe sur la machine m2 sous le nom Julie avec la commande:

rlogin m2 -l Julie (tapée par Ahmad sur la machine m1)

