Rendu TP5 Réseau

Bigard Florian, Sahuc Alexandre, Tournier Jonathan

2014-2015

14.5

2.1 Le protocole UDP 4,5/5

L'identificateur de socket sert à rattacher un numéro de port à sa socket. Par exemple, si on crée une socket numéro 4 rattachée au port 25000, et qu'on envoie un paquet UDP avec la socket 4 alors on l'enverra du port 25000. Mais du coup, on ne connaît pas le numéro de socket de la machine distante car elle ne nous sert pas.

et elle n'est jamais communiquée sur le réseau

- Source Port : Port d'émission du message UDP
- Destination Port : Port de destination du message UDP
- Length : Longueur du message UDP (nombre d'octets du message + 8 octets d'en-tête UDP (2 octets par champ))
- Checksum : Une somme hachée permettant de vérifier l'intégrité du paquet à la réception

infos données par UDP à IP ?

Si on demande la réception avant l'émission des données, alors le récepteur se met en attente.

Si on envoie plusieurs paquets, ils sont mis en attente dans le buffer de réception du récepteur. Lorsque le récepteur demande la réception, il récupère le premier paquet en attente.

Si on croise l'émission des paquets, alors les paquets sont envoyés et reçus dans le bon ordre.

Si on veut recevoir un certain nombre d'octets, et que l'émetteur en envoie moins ou autant alors tout va bien le récepteur reçoit bien tous les octets. En revanche si l'émetteur en envoie plus alors le récepteur ne reçoit que le maximum des octets demandés. Ceux en trop sont perdus.

Si on déconnecte la machine du réseau alors udp renvoie un "no route to host". Aucun paquet UDP n'est envoyé.

Si on essaye de saturer le buffer du récepteur, alors ce dernier ignore les paquets en trop.

Si on envoie un paquet UDP à un port inexistant alors le récepteur renvoie un paquet ICMP à l'émetteur lui disant que la destination n'est pas atteignable car port inexistant. En effet, lorsque le message est envoyé, il est ensuite récupéré par UDP et c'est ici qu'UDP sait si le port existe, ou non. Si le port n'existe pas, alors UDP fait envoyer un paquet ICMP pour prévenir l'émetteur que le port sur lequel il souhaite envoyer des messages n'existe pas. Sinon tout est ok ! Ici c'est bien UDP qui fait envoyer le paquet ICMP car le protocol IP ne sait pas si le port existe ou non (aucune information sur le port dans l'entête des paquets IP)

Lors de la reception d'un paquet UDP : Si la taille du message > taille buffer reception
le message est ignoré
Sinon le message est reçu
Si on demande à lire une taille inférieure à la taille du message
Seulement le nombre d'octets demandés est lu, le reste du message est ignoré
Sinon
Le message est lu en intégralité
Si le port de destination n'est pas atteignable Envoyer un paquet ICMP à l'expéditeur pour l'en informer

2.2 Le protocole TCP 2,5/3

2.2.1 Établissement de la connexion

On dit que la première socket est passive car elle attend une connexion. L'autre socket est active car c'est elle qui va aller demander une connexion à l'autre socket passive. La socket passive accepte ensuite la connexion. Cette dernière est donc établie.

On observe 3 paquets TCP :
- SYN : permet de demander une connexion TCP
- La réponse est un SYN + ACK : Acceptation de la connexion en disant qu'il a bien reçu la demande de connexion
- ACK : Le demandeur a bien reçu l'acceptation de sa demande de connexion

Le flag SYN permet de faire une demande de connexion.

Le numéro de séquence correspond au numéro du paquet envoyé, et le numéro d'acquittement permet de donner un accusé de réception à l'émetteur et a pour valeur le numéro de séquence reçu + 1.

Les différentes options sont : → options TCP → il y a aussi dans les paquets SYN des options pour se mettre d'accord sur la taille de window, la taille max des fragments, etc
- Reserved : Réservé pour un potentiel usage futur
- Nonce : Protection permettant de signaler la congestion de paquets
- CWR : Mis par l'émetteur pour indiquer qu'il a reçu un fragment TCP avec le flag ECE mis
- ECN-echo : Dépend du flag SYN :
  - Si SYN = 1 alors alors la pair TCP est ECN capable
  - Si SYN vaut 0 alors c'est simplement un paquet avec un flag ECN dans une transmission normale
- URG : Signale que le paquet contient des données urgentes
- ACK : Signale que le paquet est un accusé réception
- PSH : Ce sont des données à envoyer tout de suite
- RST : Rupture anormale de la connexion
- SYN : Demande de synchronisation ou établissement de la connexion
- FIN : Demande la fin de la connexion

L'accept permet de pouvoir faire communiquer les deux machines (émettre et recevoir des paquets). Dans le cas contraire, la connexion se fait bien mais il est impossible aux deux machines de pouvoir communiquer.

Lorsqu'on ouvre plusieurs connexion sur un même port d'une machine distante et après avoir fait un *status*, on observe que ce qui différencie une connexion est l'id et donc par extension le port de l'émetteur.

Si on fait une demande de connexion vers un port inexistant alors on nous dit que la connexion est refusée. En effet, le récepteur renvoie un paquet RESET-ACK disant qu'il a bien reçu la demande de connexion mais que la connexion ne peut pas se faire (car positionne l'option RST de l'header).

2.2.2 Étude du séquencement et de la récupération d'erreur 3.5/4

Les paquets générés sont entièrement des paquets TCP. Il y a toutefois un échange de paquets entre les deux machines. Régulièrement, la machine réceptrice envoie un paquet à l'émetteur pour lui indiquer le dernier octets qu'elle a reçu.
- SEQUENCE NUMBER permet de préciser le numéro de la séquence des données qu'on envoie. Le SEQUENCE NUMBER est propre à chaque machine.
- ACK NUMBER permet de donner un accusé réception du numéro de séquence qu'on a reçu auquel on ajoute le nombre d'octets reçus. Par exemple si on reçoit 20 octets d'une séquence de 10, on renverra 30 comme ACK NUMBER. les 20 premiers

Non, il y a un acquittement par séquence de données que l'on reçoit. On est obligé au cas ou l'on perdrait une des séquences de données lorsqu'on reçoit un paquet. NON → si tu envoies 2K octets (coupés en 2 TCP donc) et tu reçois la 2e partie avant la 1re, tu enverras un seul ACK

Si on envoie un paquet TCP alors que la machine est débranchée du réseau alors rien ne se passe (aucun message, rien sur Wireshark). En revanche, si on rebranche la machine alors les paquets sont bien transmis. En effet, TCP attend un moment avant de renvoyer le paquet. C'est la différence avec UDP où les paquets ne sont pas retransmis mais perdus.

Après avoir mis un buffer de 2000 sur la machine émettrice, on observe que les données sont divisées avant d'être envoyées en plusieurs paquets TCP de taille 2000. Plus le buffer est petit du coup plus le débit sera faible car on devra subdiviser les données en plusieurs paquets car on devra attendre pour chaque paquet l'acquittement associé.

Le buffer d'émission de TCP permet de stocker les données en attendant l'acquittement de ces mêmes données qu'on aura précédemment envoyé.
important pour pouvoir renvoyer en cas de perte

Comme dit précédemment avec un buffer trop petit on sera obligé de subdiviser les données qu'on veut envoyer en beaucoup de petits paquets et d'attendre impérativement chaque acquittement de chacun d'eux. Le débit en sera donc grandement réduit.

On réalise $\frac{\text{taille des données}}{\text{taille du buffer d'émission}}$ envois pour envoyer toutes les données. Pour chaque envoi, on doit attendre de recevoir le paquet d'acquittement pour cet envoi. On subit donc 2 fois la latence du réseau, une fois lors de l'envoi des données et une seconde fois lors de la réception du paquet d'acquittement.
On obtient la formule : (latence * 2)*($\frac{\text{Taille données}}{\text{taille buffer émission}}$)

Avec un buffer de 1000 octets : $(10 * 2) * \frac{10000}{1000} = 200$ms.

Avec un buffer de 2000 octets : $(10 * 2) * \frac{10000}{2000} = 100$ms

Avec un buffer de 10 000 octets : $(10 * 2) * \frac{10000}{10000} = 20$ms

En émission : Envoyer un paquet de sequence_number S, d'acquittement_number A et de longueur L Si le timer correspondant à ce paquet arrive à 0, renvoyer le paquet. Si un paquet d'acquittement_number S+L est reçu, fin.
→ préciser avec le buffer

En réception : Quand un paquet de sequence_number S, d'acquittement_number A et de longueur L est reçu : Envoyer un paquet de sequence_number A, d'acquittement_number S+L et de longueur 0 ;

2.2.3 Contrôle de flux 1/3

On remarque que l'émetteur envoie toutes les séquences du paquet. Sauf qu'en retour, le récepteur renvoie des ACK avec comme valeur pour Window 0. → cela indique à l'émetteur de ralentir le rythme
→ en faisant un read, ACK avec Window ≠ 0 ⇒ l'émetteur renvoie

2.2.4 Libération d'une connexion 2/3

La machine demandant la fin de la connexion envoie un paquet avec le flag FIN positionné expliquant qu'il veut la fin de la connexion. La machine distante répond avec un paquet ACK lui disant qu'il a bien reçu sa demande de fin de connexion.

Losqu'une machine demande la fermeture de la connexion, elle envoie un paquet avce un ACK_number A et le flags FIN positionné à 1. La machine distante renvoie alors simplement un paquet ACK de valeur A pour lui signifier qu'elle a bien reçu sa demande.

Lorsqu'on essaye d'écrire après avoir fermé la connexion, on nous demande sur quel id de connexion écrire ; or on en a pas car on l'a fermé précédemment. En revanche, de l'autre côté si on essaye d'écrire dans une connexion fermée alors le récepteur renvoie un paquet avec le flag RST positionné. Si on refait un write on a un "broken pipe" car la connexion a été fermée aussi de ce côté après que l'on ai réceptionné le paquet avec le flag RST.

L'envoi de données depuis la machine ayant fermé la connexion en écriture est impossible et la lecture sur la machine distante n'est plus bloquante.
La lecture de données sur la machine ayant fermé la connexion en écriture est toutefois possible, tout comme l'écriture sur la machine distante.

L'envoi de données depuis la machine ayant fermé la connexion en lecture est possible de même que la lecture sur la machine distante.
La lecture de données depuis la machine ayant fermé la connexion en lecture n'est cependant pas possible même si l'envoi de données depuis la machine distante reste possible.

shutdown both ferme les deux modes. La connexion existe toujours mais il n'est pas possible de lire ni d'écrire.

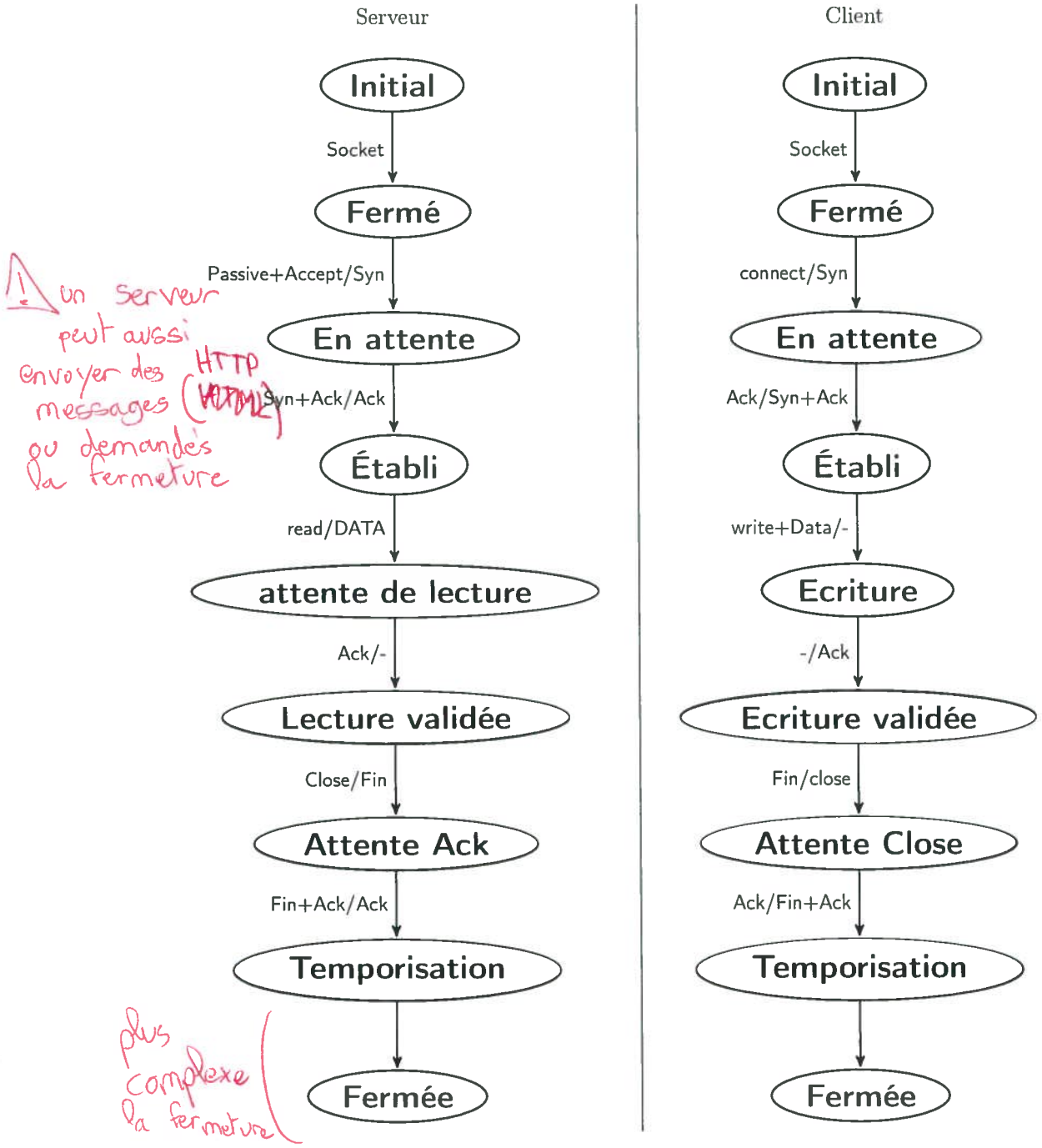
La fermeture d'une connexion en écriture, entraîne l'envoi d'un paquet TCP FIN pour en informer la machine distante.

L'utilisation de close permet de mettre fin à la connexion simplement et des deux côtés (la machine qui a demandé la fermeture met immédiatement fin à la connexion et la machine distante mettra fin à la connexion lorsqu'elle tentera d'envoyer un message à la première machine ; cette dernière enverra alors un paquet RST indiquant à la machine distante qu'elle doit fermer sa connexion).

L'utilisation de shutdown permet une fermeture moins "brutale" de la connexion. En particulier, elle permet de libérer les tampons d'émission/réception avant de fermer la connexion. Shutdown permet aussi de fermer partiellement une connexion ; par exemple si l'on souhaite recevoir des données d'une machine distante mais que l'on a pas besoin d'envoyer en retour (on peut alors fermer la connexion en écriture). Inconvénient, shutdown ne permet pas de mettre fin à la connexion (shutdown both ferme la connexion en écriture et lecture mais ne met pas fin à la connexion pour autant).

Automate deux automates qui représentent la connexion entre un client et un serveur, puis l'exécution d'une opération write par le client, et d'une opération read par le serveur. Enfin la fermeture de la connexion entre le client et le serveur

Pour chaque liaison
- **A gauche de** / : Action réaliser
- **A droite de** / : Ce qu'il attend



2.3 Exercice de synthèse

Les paquets reçus sont :
- Pour l'ouverture de la connexion : Échange des logins utilisés, puis demande de synchronisation. Puis ouverture d'une connexion TCP.

— Pour les messages : Pour chaque lettre on envoie un paquet, puis il y a un accusé réception
— Pour la fermeture : Celui qui veut fermer envoie un paquet avec le flag FIN, l'autre renvoie l'accusé de réception puis un paquet ACK avec le flag FIN positionné. Enfin, celui qui voulait fermer la connexion en premier renvoie un paquet accusé réception (ACK)

0.5/1

Tous les paquets sont TCP. Trois premiers paquets : établissement de la connexion. La première machine essaye de s'identifier, la seconde lui répond et la première acquitte. Ensuite pour chaque lettre qu'on tape on envoie un paquet et le récepteur nous renvoie la lettre tapée. Pour la fin de la connexion, la machine serveur ferme la connexion après avoir reçu un exit en envoyant un premier paquet de fermeture (flag FIN). Le client acquitte et ferme à son tour. Enfin le serveur acquitte. C'est quasiment la même chose que la fin d'une connexion TCP, sauf que dans notre cas c'est le serveur qui ferme la connexion.

0.5/1

Les commandes sont envoyées lettre à lettre à la destination.

Les caractères sont renvoyés à la source afin que la machine source puisse afficher les lettres sur son terminal. Ceci permet d'être sûr que chaque lettre écrite sur le terminal est aussi enregistrée sur la machine réceptrice.

Les réponses aux commandes sont par contre envoyés en un seul bloc