TD 12 : Protocole TCP

Introduction :

Durant ce TD, nous allons utiliser le programme « sock » de de Richard Stevens afin de montrer facilement quelques propriétés de TCP.

Etape 1 : télécharger l'archive contenant le programme :
http://www.icir.org/christian/downloads/sock-0.3.2.tar.gz

Etape 2 : Désarchiver le dossier obtenu

```
tar -xzvf  sock-0.3.2.tar.gz
```

Etape 3 : Installer le programme :

```
./configure

make

make install
```

Premiers tests pour se familiariser avec l'interface :

Dans deux fenêtres différentes, tapez les commandes suivantes :

Création d'un serveur :

```
sock  -u  -s  5555
```

et création d'un client :

```
sock  -u 127.0.0.1  5555
```

Ecrivez ensuite ce que vous voulez dans la fenêtre du client et observez ce qu'il se passe.

Question d'introduction :

En regardant les options utilisées et d'après la liste des options disponible sur http://www.icir.org/christian/sock.html , indiquez quel est le protocole utilisé ici ?

Exercice Flux de données :

Dans deux fenêtres séparées, lancez les commandes suivantes :

```
sock -i -s 7777
```

Avant de lancer la seconde commande, démarrez une capture sur Wireshark pour analyser les paquets.

```
sock -i -n8 127.0.0.1 7777
```

En observant votre capture, répondez aux questions suivantes :

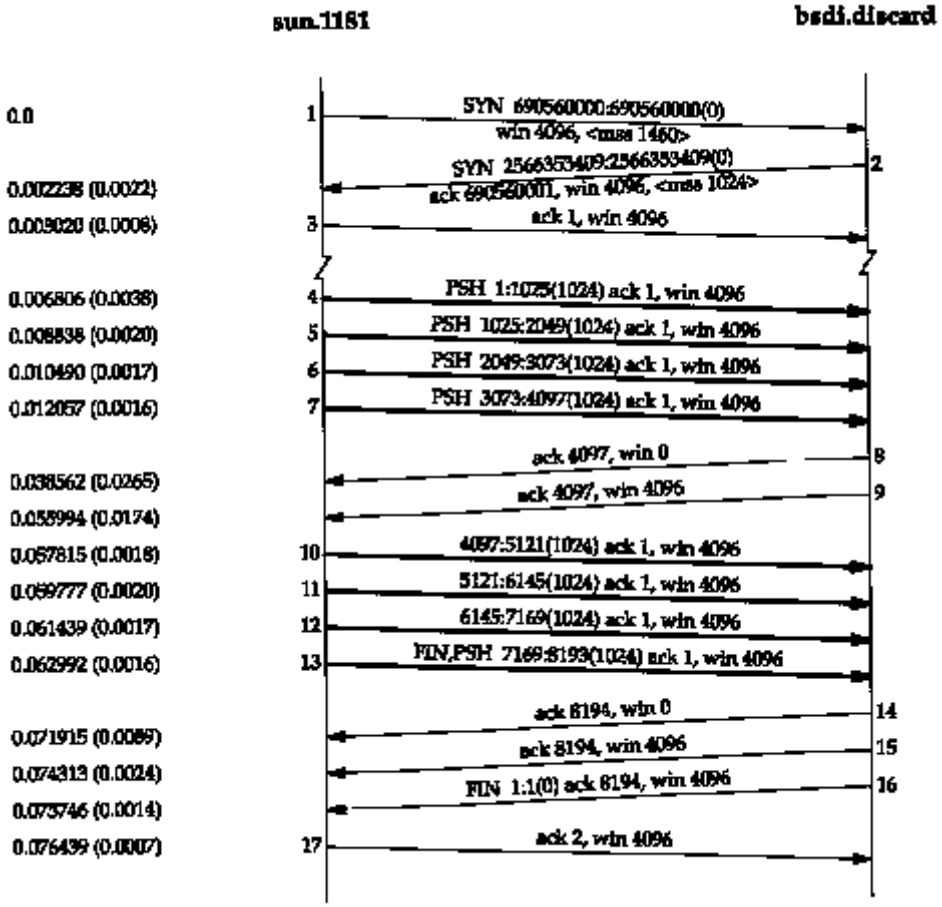
Question 1 :

Donnez les MSS des deux extrémités.

Question 2 :

En étudiant la capture de paquets, donnez le nombre de bytes transmis du client vers le serveur. Observez maintenant la commande utilisée, ce résultat était-il attendu ?

Le schéma suivant montre un même type d'échange entre un émetteur rapide et un receveur lent.



Question 3 :

Analysez et expliquez les valeurs win des segments 8 et 9.

Exercice Congestion de données.

Question 1 :

En consultant la documentation, les options du programme et/ou en vous aidant du premier exercice, donnez la commande créant un serveur dont le but est simplement de recevoir des données sans les utiliser (« sink » server).

Question 2 :

Donnez la commande créant un client dont le but est simplement d'envoyer 32 messages de 1024 octets à notre serveur (« source » client). Indice, regardez l'option -n

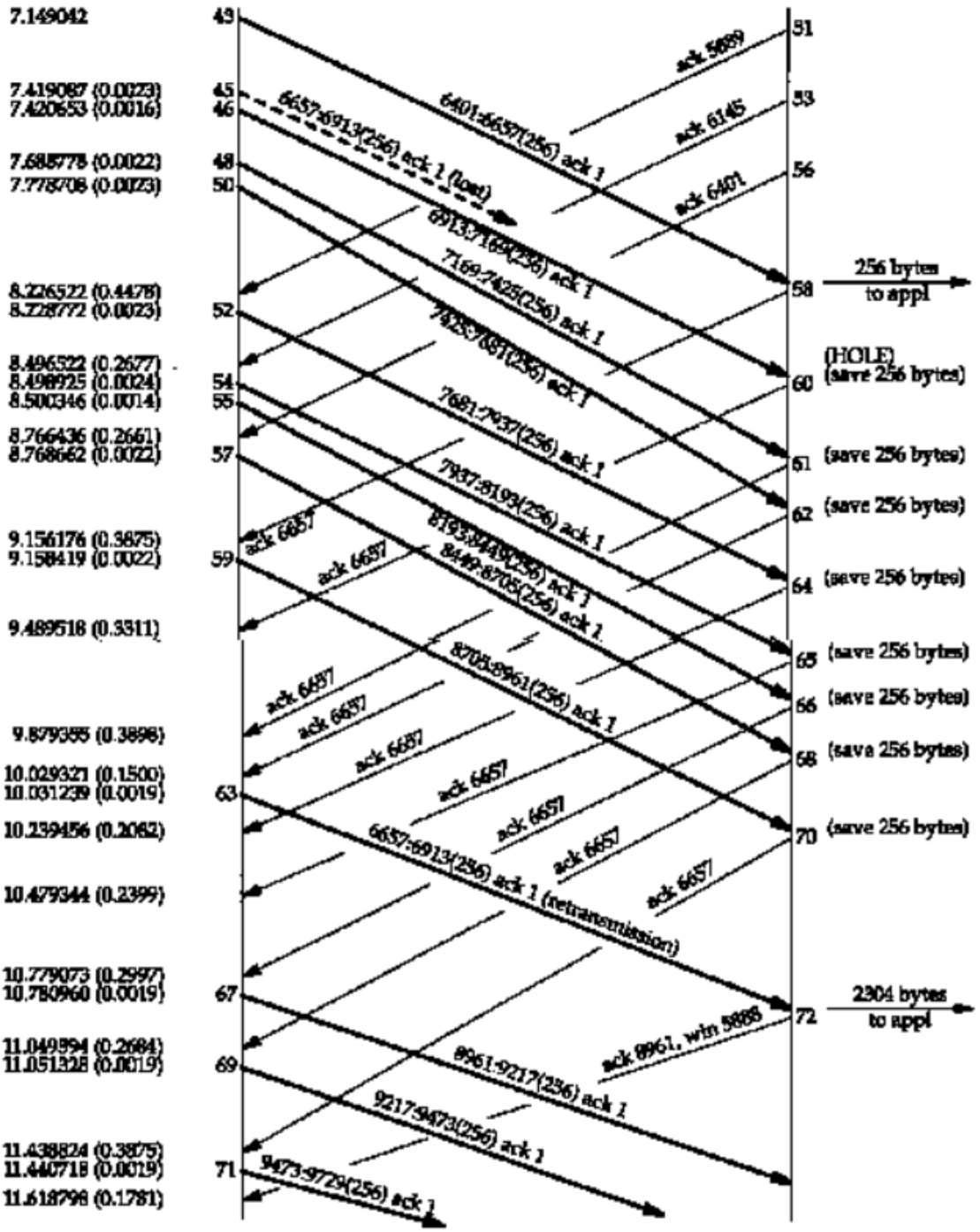
Question 3 :

a) Donnez la taille des entêtes TCP/IP.

b) Si la MTU (Maximum Transmission Unit) entre notre serveur et notre client et de 296 octets, de quelle taille sera la MSS (Maximum Segment Size) ?

c) Quelle sera donc le nombre de paquets de données que notre client enverra au total ?

Voici un schéma montrant un extrait des échanges de paquets issus de cette commande.



Question 4 :

Que représente la barre verticale de gauche ? client ou serveur ?

Le schéma montre que le segment 45 est perdu.

Question 5 :

Qu'arrive-t 'il aux segments 46,48,50,52,54,55,57 et 59 ?

Question 6 :

Expliquez les numéros d'acquittement du segment 58 d'une part et des segments 60,61,62,64,65,66,68,70 d'autre part.

On dit que ces segments sont des ACK dupliqués (DUP ACK)

Question 7 :

Combien le serveur reçoit-t 'il d'acquittements dupliqués avant de réémettre le paquet perdu ?

Quel est le nom de la méthode de TCP permettant ce phénomène ?

Si le serveur n'avait pas reçu ces acquittements dupliqués, il aurait réémis le paquet perdu après un certain temps qui est le délai de retransmission (RTO, Retransmission Timeout) ce délai est calculé en fonction du RTT (Round-Trip Time) c'est-à-dire du temps nécessaire pour envoyer un message et recevoir son acquittement. Il est toujours supérieur à 3 fois le RTT (rfc 2988).

A gauche du diagramme figurent les temps à partir du premier envoi.

Question 8 :

Calculez le RTT pour le segment 49 par exemple et calculez le temps qui a été nécessaire pour retransmettre le paquet perdu. Conclure sur l'efficacité de la méthode utilisée ici.

Question 9 :

Justifiez le numéro d'acquittement du segment 72.

Exercice TCP Keepalive :

Nous allons vous présenter l'option keepalive de TCP. Pour se faire, nous avons au préalable lancé un serveur avec l'option keepalive comme ci-dessous :

```
sock -s -K 5555
```

Puis nous avons connecté un client en telnet sur serveur. Le client est resté inactif durant plusieurs heures pendant lesquelles nous avons capturé les paquets sur la carte réseau local lo avec la commande

```
dumpcap -i lo keepalive_cap
```

Vous trouverez le fichier keepalive_cap que vous pourrez analyser sous Wireshark.

Question 1 :

Durant plusieurs heures, le client est resté inactif. Comment a réagi le serveur ?

Au bout de combien de temps le serveur envoie un paquet TCP ? (Convertir en heures)

Question 2 :

A votre avis, si le client ne répond pas, qu'est que cela peut signifier ?

Comment le serveur peut-il réagir ?

Question 3 :

Quelles sont les avantages et les inconvénients d'une telle option ?