Étude détaillée du protocole TCP TP N° 3

Introduction

L'objectif de ce TP est de comprendre les fonctionnalités du protocole TCP (Transfert Control Protocol) utilisé par la plupart des applications utilisant le réseau Internet.

Ce protocole est assez complexe vu les nombreux services qu'il rend :

- Contrôle et récupération des erreurs : TCP garantit que toutes les données envoyées seront reçues sans la moindre erreur par l'entité distante : il n'y aura ni perte, ni modification des données pendant leur transport d'une machine à l'autre.
- Ré-ordonnancement : si les données sont envoyées dans un certain ordre, elles seront rendues par TCP dans le même ordre.
- Contrôle de flux : Pour ne pas saturer le récepteur et risquer d'avoir des pertes chez le récepteur, TCP régule le flux des données en fonction de la cadence « d'absorption » par l'entité réceptrice.
- Contrôle de congestion : pour palier au problème d'embouteillage dans les routeurs intermédiaires d'Internet une régulation de l'émission est mise en place dans TCP.
- Multiplexage des accès : une ou plusieurs entités peuvent gérer simultanément plusieurs connexions TCP.

Pour réaliser ces différents services TCP est un protocole orienté connexion : cela signifie que si deux entités veulent s'échanger des données à travers TCP, elles doivent préalablement établir une connexion virtuelle. Au départ les deux entités possèdent deux rôles distincts :

- l'une est à l'écoute de demandes éventuelles, elle est dit « passive », c'est ce que l'on appelle le serveur ;
- l'autre est active, elle fait la demande de connexion. C'est le client.

Une fois la connexion ouverte, elle est symétrique et bi-directionnelle.

Le service proposé par TCP possède aussi la caractéristique d'être de type *byte-stream* (flux d'octets), il n'y a pas de découpage fixe par TCP dans le flux de données véhiculées.

La notion de port a été introduite pour permettre le multi-accés à TCP. Un port est un entier de 16 bits qui sert à identifier un point d'accès aux protocoles de la couche 4. Ainsi

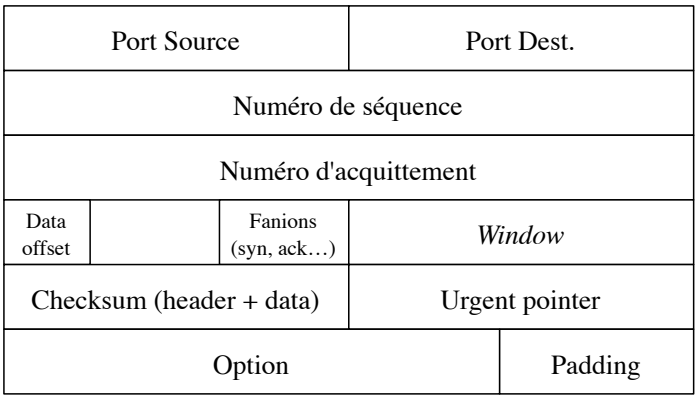

FIG. 1 – Entête TCP

une connexion TCP est identifiée de façon unique par *deux* couples [adresse Internet, numéro de port].

L'entête TCP est représenté en figure **??**. Le champ de bits *fanions* est composé des 6 fanions suivants : URG, ACK, PSH, RST, SYN, FIN.

Signification des différents champs :

- SOURCE PORT ET DESTINATION PORT ont le même sens que dans les paquets UDP.
- SEQUENCE NUMBER et ACK NUMBER sont utilisés pour le séquencement et la récupération d'erreurs de données (le flag ACK indique si le champ ACK NUMBER contient une valeur valide).
- *Data offset* indique la taille de l'entête TCP en mots de 4 octets (la taille de l'entête TCP est variable : elle peut être complétée par une ou plusieurs options de 4 octets chacune).
- *Checksum* a le même sens que dans les paquets UDP.
- *Urgent pointer* est utilisé pour le transport de « données urgentes » (le flag URG indique si le champ URGENT POINTER contient une valeur valide). Il indique les octets à traiter en priorité.
- Les flags SYN et FIN sont utilisés pour l'établissement et la fermeture des connexions virtuelles.
- Le flag RST est utilisé pour fermer les connexions virtuelles qui sont dans un état incertain (SYN dupliqués ou panne).
- Le flag PSH permet de signaler au récepteur qu'il faut délivrer immédiatement à l'application les données.
- Le champ *Window* est utilisé pour le contrôle de flux. Il contient le nombre d'octet que le récepteur peut encore stocker à partir du numéro d'acquittement contenu dans le même paquet.

Déroulement du TP

Pour étudier le protocole TCP, vous utiliserez un utilitaire vous proposant une interface souple et simplifiée sur les sockets : `socklab`. Ce « laboratoire » à *socket* vous permet en fait d'appeler de façon interactive les primitives de base de manipulation des sockets. Une socket est un « point d'accès » que vous devez créer pour envoyer ou recevoir des données par le protocole TCP. Pour chacune des expériences qui suivent, résumez les échanges de paquets sur un croquis temporel en faisant apparaître les champs pertinents.

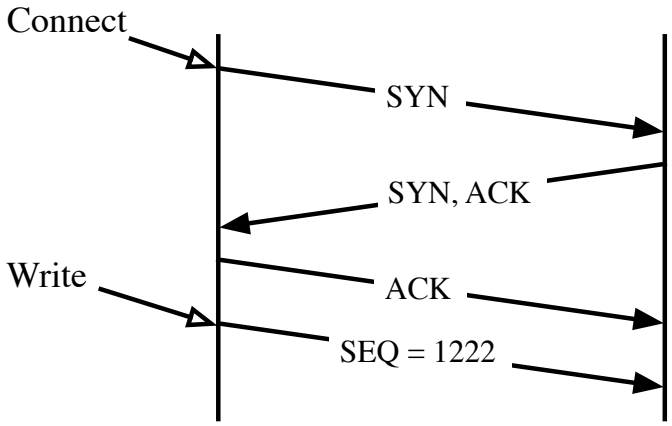

FIG. 2 – Ébauche de croquis temporel

1 Établissement et libération d'une connexion

1.1 Ouverture/fermeture « normales »

Sur deux machines, lancez `socklab`.

- Analysez les paquets générés lors de l'établissement d'une connexion TCP. Décomposez les étapes de cette connexion : enchaînement dans le temps des demandes de services à TCP et des messages échangés.
- TCP utilise des options, expliquez à quoi sert chacune d'elle ?
- Ouvrez plusieurs connexions d'une machine vers un même port destinataire. Qu'est-ce qui identifie réellement une connexion, c'est-à-dire, comment TCP associe-t-il les messages reçus aux différentes connexions en cours ?
- **Analysez les paquets générés lors de la fermeture d'une connexion.** Attention une connexion n'est complètement fermée que lorsque la commande `close` a été faite de chaque côté de la connexion.
- Après fermeture d'un seul des deux côtés par un `close`, essayez de continuer à émettre de l'autre côté (par un `write`). Que se passe t-il (observez les paquets échangés). Si vous refaites un write que se passe t-il ? pourquoi ?

- Essayer différents scénarios de fermeture à l'aide de la commande `shutdown` (voir documentation Socklab).
  Par exemple, réaliser une fermeture en sortie (`shutdown out`) sur une machine, puis une émission de données depuis l'autre. Peut-on continuer à lire les données envoyées ? Peut-on continuer à faire un `write` après un `shutdown out` ? Faites de même pour le `shutdown in` et `both`.

  Résumez les échanges de messages dans chaque cas (faites en sorte que des données non lues soient présentes au moment de la fermeture) en spécifiant la valeur des champs spécifiques à cette phase de la communication.
- Expliquer les avantages et inconvénients de ces différents types de fermeture (`close` et `shutdown`). Quel intérêt de considérer la connexion comme deux demi-connexions unidirectionnelles ?

1.2 Ouverture vers un port non utilisé

Faites une demande de connexion vers un port non utilisé.

Expliquez ce qui se passe en observant le réseau.

1.3 Fermeture anormale

- Après ouverture d'une connexion, coupez le réseau et faites une demande de déconnexion depuis une des deux machines (soit A). Comment TCP résout-il ce problème ? Pourquoi des paquets ARP peuvent-ils apparaître au bout d'un certain temps ? Le timer de ré-émission utilisé dans ce cas par TCP, a-t-il une durée fixe ?
- Combien de temps TCP insiste-il avant de considérer la connexion fermée ? Attention cela peut être long.
- Que se passe-il, si la demande de déconnexion depuis l'autre machine (B) arrive après que la première machine (A) ait considérée la connexion comme fermée ? Pour observer cela, faites un `close` depuis B après avoir rebranché le réseau

1.4 Gestion des numéros de séquences initiaux

Regardez les champs *sequence number* initiaux lors de l'ouverture de deux connexions successives. Attention, `ethereal` représente **par défaut** les numéros de séquences des flux TCP *relativement au premier paquet*. Il est possible de changer cette présentation dans `ethereal` sous `Edit/Preferences/protocols`.
Essayez de trouver une relation entre le temps écoulé (donné dans ethereal) entre les émissions des demandes d'ouverture de connexion et ces numéros. Pour obtenir chiffres significatifs on attendra quelques minutes entre les ouvertures de connexions.

Rappel : Ces numéros servent d'identificateurs de connexions et sont calculés à partir d'une horloge.

Risque-t-on d'avoir des mêmes numéros de séquence initiaux pour des demandes d'ouverture de connexion « originales » et des doublons « traînant » sur le réseau ?

1.5 Résumé du fonctionnement de TCP

En ce qui concerne l'ouverture et la fermeture de connexion, donnez un automate dont les entrées sont :
- les commandes des sockets (*connect*, *write*...)
- l'arrivée de messages particuliers (DATA, SYN, ACK...)

et les sorties sont les envois de messages particuliers.

Un automate de Mealy semble plus approprié.

Vous pouvez faire un automate pour le côté serveur et le côté client, ou réunir les deux.

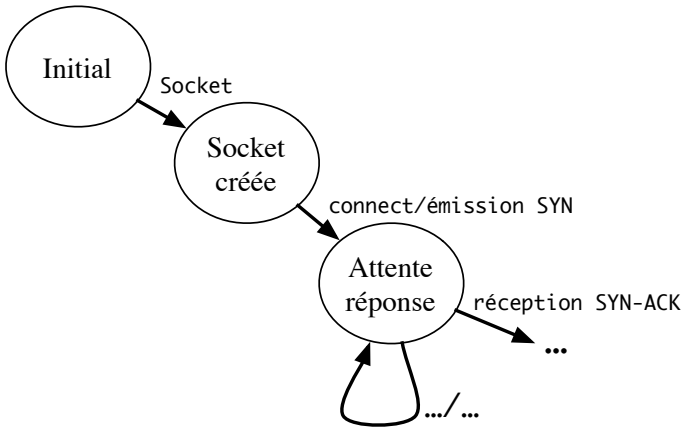

FIG. 3 –

2 Etude du séquencement

Etablissez une connexion TCP entre deux machines.

Envoyez un paquets de 5000 octets de données entre les deux machines grâce à la commande `write` (à la place d'un message normal, vous pouvez utiliser la notation `#nnn` pour envoyer un message de *nnn* octets).

Analysez les paquets engendrés par le transport des données.

Résumez sur un croquis temporel l'expérience en faisant apparaître les champs *sequence number* et *ack. number* de l'entête TCP.

Y a-t-il un acquittement par paquet de donnée ?

`Attention` : l'ordre temporel des paquets donné par Ethereal n'est as forcément exactement celui de la circulation réelle des paquets sur le réseau.

3 Récupération des erreurs, temporisateur de ré-émission et acquittements

3.1 Temporisateur de retransmission

Après l'ouverture d'une connexion TCP, coupez le réseau, et envoyez des données vers une machine que vous aurez au préalable, débranchée du réseau.

Observez ce qu'il se passe en capturant les paquets côté émetteur.

Comment évolue le Temporisateur de retransmission ? Quel intérêt ? Pourquoi observe-t-on des séries de paquets ARP au bout d'un certain temps ?

Donnez la fonction d'évolution du temporisateur de réémission utilisé dans TCP.

3.2 Temporisateur d'émission des acquittements

Emettez quelques paquets de 1000 octets.

Donnez les temps d'arrivée des données et de retour des acquittements correspondants. Pourquoi TCP retarde-t-il ses acquittements ?

4 Contrôle de flux

On peut choisir la taille du buffer de réception et d'émission d'une socket à l'aide de la commande `option` dans socklab.

Ouvrez une connexion TCP, entre deux machines éloignées sur la plate-forme (traversées de plusieurs routeurs) en vous servant de sockets dont la taille du buffer de réception et d'émission est fixée à 1000 octets. Attention la taille des buffers doit être fixée avant l'ouverture de la connexion.

Emettez sur cette connexion deux messages de 2000 octets.

Résumez sur un croquis temporel l'expérience en faisant apparaître les champs *sequence number*, *ack. number* et *window* de l'entête TCP.

Pourquoi l'émetteur continue-t-il à envoyer des paquets d'un octet de donnée ? Donnez l'évolution du temporisateur associé à ces émissions.

Pourquoi le deuxième `write` est-il bloqué ?

Faites ensuite des Read successifs coté récepteur de 500 octets. Pourquoi le récepteur ne débloque-t-il pas tout de suite la situation ? A partir de quand le fait-il ? Quel intérêt ?

Rappelez à quoi servent les buffers d'émission et de réception dans le contrôle d'erreur et de flux. Est que l'utilisation d'un buffer d'émission de taille supérieure à celui de réception est pertinente ? Et le contraire ?

5 Contrôle de congestion

Dans l'expérience précédente, analysez la succession des paquets de données et des acquittements.

Notez en particulier la taille des données que TCP se permet d'envoyer sans acquittement. Pourquoi n'est-elle pas égale à la taille donnée par le récepteur dans le champ WINDOW ? Refaites si nécessaire des expériences pour mettre en évidence la gestion de la fenêtre de congestion par TCP (algorithme du « *slow-start* », seuil, *timer* de ré-émission qui modifie la taille de la fenêtre).

5.0.1 Flux UDP contre flux TCP

Générer un flux UDP et un flux TCP sur deux couples de machines de votre réseau (utilitaires `tcpmt`, `udpmt`, `tcptarget` et `udptarget`).

`tcpmt -p #port -s taille_paquet @IP_dest`. Le numéro de port est donné par la cible au moment du démarrage. Il peut également être fixé à son lancement.

Observez le réseau et expliquez ce qui se passe. Il est préférable de démarrer la source TCP d'abord. Pourquoi ?

6 Algorithme de Naggle

Cette expérience permet de mettre en évidence l'algorithme développé par Naggle pour TCP. Il consiste à regrouper l'émission de paquets de petites tailles. TCP n'envoie pas plus d'un paquet de petite taille sans avoir reçu d'acquittement. Pour observer ce phénomène, il faut donc faire varier le délai de transmission.

Lancez un `telnet -y`[1] entre deux machines « proches » sur la plate-forme. Pour simuler la frappe très rapide de caractère, faites un copier/coller d'une quarantaine de caractères. Analysez les paquets échangés. (On peut également utiliser la répétition automatique de la frappe.)

Refaites la manipulation avec des machines éloignées (plusieurs routeurs intermédiaires). Quel est l'intérêt de cet algorithme ?

[1] Le `-y` désactive le chiffrement du trafic.