

<p>TP N° 4 de Réseaux Etude des protocoles de la couche transport d'Internet UDP et TCP</p>

Pascal Sicard

1 INTRODUCTION

L'objectif de ce TP est d'observer et de commencer à comprendre le fonctionnement des protocoles de la couche transport d'Internet **UDP** (User Datagram Protocol) et **TCP** (Transmission Control Protocol).

Avant d'entamer les manipulations, voici quelques rappels :

1.1 Architecture des réseaux, protocoles et services

Pour communiquer sur un réseau, les machines utilisent un ensemble de règles et de conventions appelées **PROTOCOLES**. Partant du principe de modularité et compte tenu de leurs complexités, les protocoles ont été structurés en couches dans le but de faciliter et de contrôler leurs implémentations.

L'un des avantages de cette structuration est d'isoler les différents protocoles pour que tout changement introduit sur l'un d'eux n'affecte pas les fonctionnalités des autres.

Ce modèle offre des interfaces entre les différentes couches afin de permettre aux protocoles d'une couche donnée d'interagir avec ceux des couches qui lui sont directement adjacents. En effet, chacune des couches s'appuie sur des **services** offerts par une couche inférieure et vise à offrir ses services à la couche qui lui est supérieure.

1.2 Présentation des protocoles étudiés

La famille de protocoles TCP/IP est devenu un standard de fait, de part son utilisation dès le début d'Internet.

La figure 1 montre les diverses interactions qui existent entre les principaux protocoles de la famille de protocoles d'Internet :

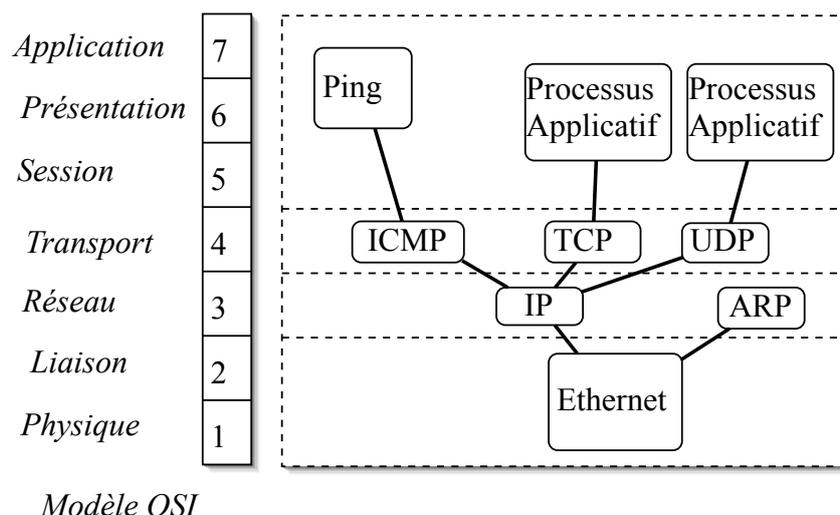


FIGURE 1 – Principaux protocoles

Dans la famille de protocoles d’Internet, la couche de niveau transport est la couche la plus haute avec laquelle les processus communiquent pour envoyer ou recevoir des données. Deux protocoles de niveau transport sont définis : TCP et UDP. Du point de vue de l’utilisateur, les qualités de services proposés par ces deux protocoles sont très différentes.

1.2.1 Le protocole TCP

Le service offert par TCP peut être comparé à celui offert par le téléphone : Quand vous téléphonez à quelqu’un, le dialogue ne peut s’instaurer qu’a partir du moment où votre interlocuteur décroche son combiné et dit ”ALLO” (ce qui correspond à l’établissement de la connexion) : vous pouvez alors dialoguer...

Un protocole offrant un tel service, en l’occurrence TCP, est dit un protocole **orienté connexion** : cela signifie que si deux processus veulent s’échanger des données par TCP, ils doivent préalablement établir une connexion (dite « virtuelle »). Une fois la connexion établie, TCP garantit que toutes les données envoyées par le premier processus seront reçues sans la moindre erreur par le deuxième : il n’y aura ni perte, ni modification des données. De plus, si les données sont envoyées dans un certain ordre, elles seront réceptionnées dans le même ordre.

Le service proposé par TCP possède aussi la caractéristique d’être de type ”**byte-stream**” (flux d’octets). Ainsi si un premier processus envoie 5 puis 15 caractères, ceux-ci peuvent être récupérés de différentes manières par le processus distant : en une lecture de 20 caractères, deux lectures de 10 caractères, deux lectures de 7 et une lecture de 6 caractères...

Autrement dit, il n’y a pas de découpage fixé au niveau applicatif dans le flux de données véhiculées par les paquets TCP.

1.3 Le protocole UDP

Le service offert par le protocole UDP peut être comparé à celui offert par la poste : quand vous postez une lettre, et si vous demandez le tarif habituel, il peut arriver que cette lettre se perde. Par ailleurs, il se peut qu'elle arrive à destination après une autre lettre qui avait pourtant été postée après elle...

De la même manière, deux processus peuvent aussi utiliser le protocole UDP pour s'envoyer des données. Avec UDP, **aucune connexion préalable n'est nécessaire**, mais à l'inverse de TCP, UDP ne donne aucune garantie quant à la qualité du service proposé : des données peuvent être perdues, arrivées dans le désordre, éventuellement modifiées... C'est à l'utilisateur d'effectuer ces contrôles, si cela est nécessaire.

Alors que TCP véhicule un flux d'octets entre les applications, UDP véhicule un flux de paquets de données applicatives : si un premier processus envoie 5 puis 15 caractères par UDP, le processus distant ne pourra pas les lire en une seule fois : il devra lire les deux paquets séparément.

Avec TCP, l'unité d'information est l'octet ; avec UDP c'est le message.

En résumé, deux applications peuvent communiquer en utilisant le protocole TCP ou UDP. Quels peuvent être les critères permettant de choisir l'un plutôt que l'autre ? Les critères les plus évidents sont directement liés aux qualités de services de TCP et UDP. Par exemple, si l'on veut absolument une communication fiable à 100% (par exemple transfert de fichier), on choisira TCP...

1.4 Problème de l'adressage au niveau transport

Dans le TP précédent, vous avez vu qu'une machine était identifiée par une adresse INTERNET ; C'est cette adresse que TCP indique à IP quand il veut envoyer un message, elle se retrouve ensuite dans l'entête IP. Mais au niveau *transport*, cette adresse de niveau *réseau* n'est pas suffisante, que vous vouliez utiliser TCP ou UDP.

A un instant donné, plusieurs applications utilisant TCP ou UDP peuvent tourner en même temps sur une machine, donc plusieurs connexions ont été ouvertes par plusieurs processus sur une même machine ; l'adresse INTERNET de la machine ne permet pas, à elle seule, de distinguer ces connexions ; la notion de **port** a été introduite pour faire ces distinctions. On peut parler d'adressage de niveau transport.

Un numéro de port est un entier de 16 bits qui sert à identifier un point d'accès aux protocoles de la couche 4. Ainsi une connexion TCP peut être identifiée de façon unique par deux couples [**adresse INTERNET, numéro de port**] source et destination. Quand un processus veut envoyer un paquet vers un autre processus d'une machine distante, il doit indiquer l'adresse INTERNET de cette machine, et un numéro de port particulier sur celle-ci. La figure 2 résume ces notions pour un petit exemple : elle représente deux connexions de deux clients sur la machine 2 vers un même serveur sur la machine 1.

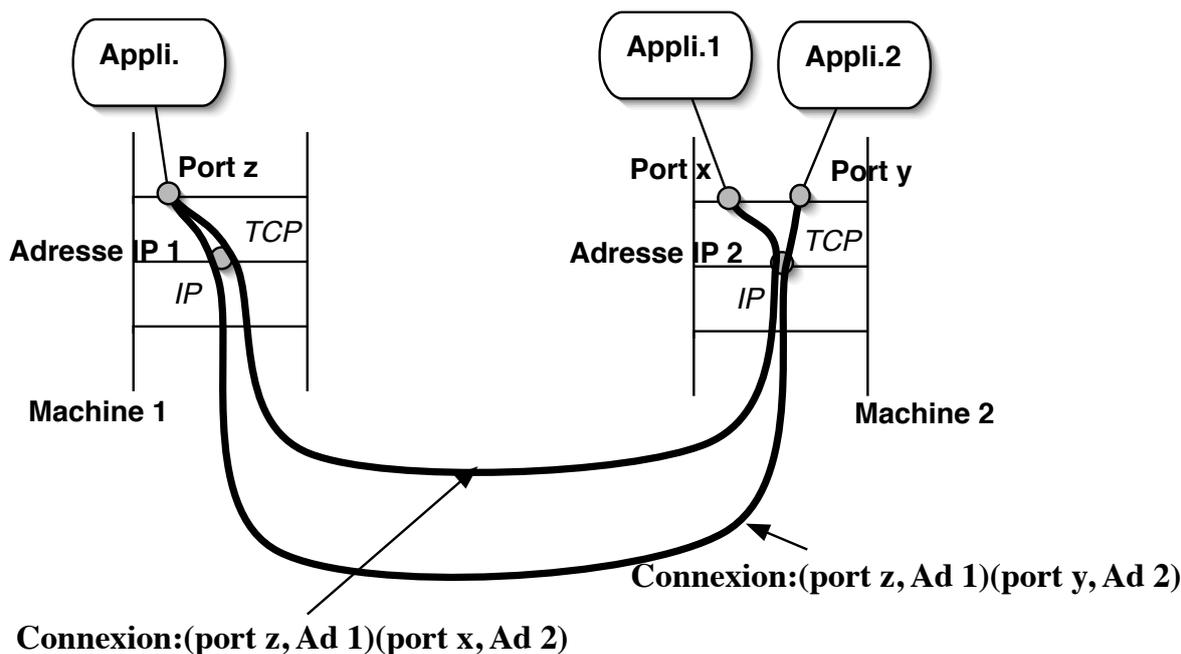


FIGURE 2 – Connexions TCP

1.5 Interface logicielle sur les protocoles TCP et UDP

Jusqu'ici, nous vous avons présenté les aspects théoriques de TCP et UDP. Comme ces deux protocoles sont directement accessibles par n'importe quel processus utilisateur (ou applicatif), cela signifie que le système met à la disposition de l'utilisateur un certain nombre de primitives permettant d'envoyer et recevoir des données par TCP ou UDP. Toutes ces primitives constituent une interface (API : Application Program Interface). Il existe plusieurs interfaces possibles, selon le type de système d'exploitation et le langage de programmation utilisé. Sous les systèmes UNIX, les deux interfaces les plus utilisées sont :

- les **sockets** BSD (Berkeley Software Distribution).
- Systeme V TLI (Transport Layer Interface).

Dans ce TP, nous utilisons les **sockets**. Le langage de programmation à utiliser est à l'origine le C, mais dans un premier temps, vous ne serez pas amenés à développer en C. Pour étudier les protocoles TCP et UDP, 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** définit aussi un "**point d'accès**" que vous devez créer pour envoyer ou recevoir des données par le protocole UDP ou TCP.

2 DEROULEMENT DU TP :

 : Cette icône indique par la suite les expérimentations à effectuer et à résumer.

 – 1 : Celle ci indique les questions auxquelles il faut donner une réponse précise et détaillée dans votre compte rendu.

 Dans ce TP, au moins 2 stations doivent être connectées sur un même réseau.

- Remplissez les fichiers `/etc/hosts` des machines pour associer des noms aux adresses, cela vous fera gagner du temps et évitera que des résolutions de noms DNS soient lancées inutilement par certains outils.
- Les captures et les analyses des paquets générés par les manipulations se feront à l'aide de l'utilitaire **Wireshark** (cf. documentation).
- Ces captures pourront se faire sur l'une quelconques des 2 machines. Toutefois si l'outil de capture vous indique des paquets d'une taille plus grande que le MTU habituel (1500 pour Ethernet), et/ou des paquets TCP non acquittés, il faut :
 - soit faire la capture sur le récepteur de données TCP,
 - soit rajouter l'option **-tso4** à la commande `ifconfig` de l'interface utilisée (exemple : `ifconfig em0 -tso4`).
- Vous pouvez sauvegarder vos captures en format « Wireshark » ou format ASCII (voir le mode d'emploi dans la documentation « Outils »).
- Une option intéressante de Wireshark pour vous aider se trouve dans le menu **statistics/ flow graph**.
- On résumera les expérimentation à l'aide de croquis temporel faisant apparaître les paquets capturés et les champs intéressants des entêtes.
- Vous utiliserez le logiciel **socklab** (voir la documentation fournie), dont le but est de vous proposer une interface sur les primitives de manipulation des sockets.

2.1 Le protocole UDP

Rappel : La figure 3 donne le format de l'entête UDP.

 Socket UDP

- Sur deux stations différentes, lancez **socklab** :
`# socklab`
- Sur chacune des deux stations, créez une socket UDP. Ces deux sockets seront

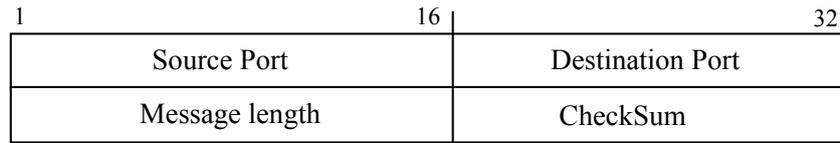


FIGURE 3 – Format de l'entête UDP

utilisées pour échanger des messages entre les deux stations :

socklab> socket

Précisez le protocole utilisé :

Protocol ? [tcp/udp] : udp

- Associez une adresse IP et un numéro de port à la socket précédemment créée à l'aide de la commande **bind**.

Par exemple :

socklab> bind

Id. socket (4) ? :

Host ? : 192.168.0.1

Port ? : 2000

Remarque : On peut aussi taper directement **socklab> bind 4 192.168.0.1 2000**. A noter que ces deux formes d'utilisation sont possibles pour toutes les commandes de socklab.

Il faut bien entendu faire cette association sur les deux stations.

Notez bien le numéro de port que vous avez utilisé : il identifie de façon unique la socket sur la machine.



– **2** Le numéro de port doit il être le même sur les deux machines ? Pourquoi ?

Peux t-on utiliser deux fois le même numéro de port sur une machine ? Pourquoi ?

Remarque : On peut utiliser le numéro de port **0** si on veut laisser le libre choix au système d'un numéro de port "libre" sur la machine (souvent le cas sur le client). On peut utiliser le caractère ***** au lieu de spécifier une adresse IP de la machine. Ce caractère ***** sous entend "toutes les adresses de la machine", elle peut en effet en avoir plusieurs (intéressant pour un serveur).

Essayez ces possibilités.

On peut voir à tout moment la liste des sockets créées sur la machine à l'aide de la commande **status**.

- Sur une des deux machines, demandez à émettre un paquet de données vers l'autre machine, en précisant son nom (ou adresse IP) et le numéro de port de la socket

précédemment créée :

```
socklab> sendto <Id de Socket> <nom de machine distante> <numéro de port>
```

- Tapez et validez la chaîne de caractères qui doit être transmise vers la machine distante.
- Sur la deuxième machine, demandez à recevoir un paquet de données sur la socket précédemment créée, en précisant le nombre maximum d'octets à lire :
socklab> recvfrom <Id de Socket> <nb d'octets>

 – **3** Capturez et analysez le (ou les) paquets engendrés par l'émission d'un message.

 – **4** A quoi sert l'identificateur de **socket** ? Connait-on le numéro de la socket distante ?

 – **5** Précisez le rôle de chaque champ de l'entête UDP. Quelles sont les informations passées à IP par UDP à l'émission d'un paquet (données + paramètres de service) ?

 Effectuez les variantes suivantes lors de l'échange de données à l'aide d'UDP :

- Demandez la réception avant l'émission des données ;
- Envoyez plusieurs paquets avant de demander leur réception ;
- Croisez l'émission des paquets : sur chacune des stations, demandez d'envoyer un paquet, puis demandez de lire le paquet envoyé par la station distante.
Notez ce qu'il se passe quand vous demandez à recevoir plus ou moins d'octets que la station distante en envoi.
- Envoyez un paquet vers une machine que vous aurez au préalable, débranchée du réseau. Observez ce qui se passe sur le réseau à l'aide d'une troisième machine (ou sur la machine émettrice).

 – **6** Expliquez ce qu'il s'est passé.

- Vous pouvez connaître et changer la taille du buffer de réception à l'aide de la commande **options** de Socklab.

- Fixer la taille du buffer de réception d'une socket à 10 000 octets.
- Lancez une capture sur *wireshark*.
- Envoyez plusieurs paquets de taille importante vers cette socket de façon à "saturer" le récepteur (remplir son buffer de réception). Par exemple envoyer 3 paquets de 4000 octets (Sous socklab taper # 4000 à la place du message à envoyer), puis un paquet de petite taille sous forme de texte puis essayez de les réceptionner .

 – 7 Expliquez ce qu'il s'est passé.

- Envoyez un paquet UDP vers un port inexistant.
Observez les paquets qui sont alors échangés entre les machines.

 – 8 Quel protocole est utilisé alors ? Expliquez ce qui se passe entre les différents protocoles sur la machine réceptrice lors de la réception du paquet UDP.

 – 9 Ecrivez un résumé sur le fonctionnement du protocole UDP au vu de l'ensemble de ces manipulations

2.2 Le protocole TCP

Rappel : La figure 4 donne le format de l'entête TCP.

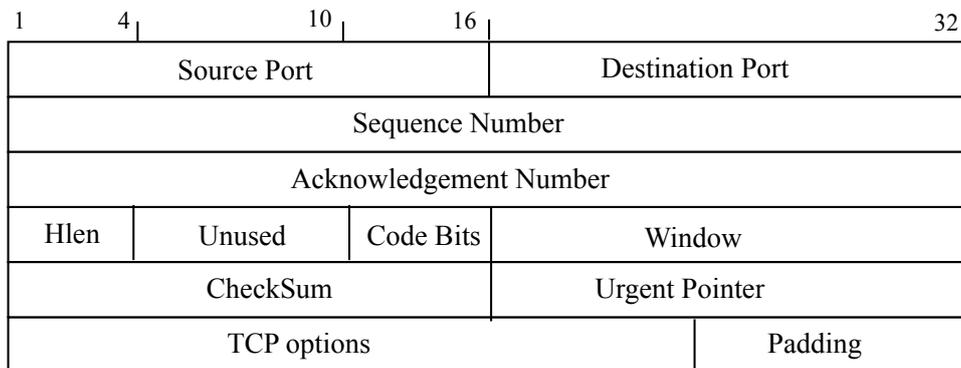


FIGURE 4 – Format de l'entête TCP

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 séquençement et la récupération d'erreurs des données (le flag ACK indique si le champ ACK NUMBER contient une valeur valide).

- **HLEN** 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).
- **Le champ Code bits** est composé des six flags suivants : **URG, ACK, PSH, RST, SYN, FIN**.
 - Les flags **SYN** et **FIN** sont utilisés pour l'établissement et la fermeture des connexions virtuelles.
 - Le flag **RST** est utilisé pour signaler au destinataire une demande de re-initialisation des connexions qui sont dans un état incertain (**SYN** dupliqués, fermeture anormale, panne ...).
 - Le flag **PSH** ne sera pas étudié.
- Le champ **WINDOW** est utilisé pour le contrôle de flux.

2.2.1 Etablissement d'une connexion

 Sur deux machines, lancez **socklab**.

- Sur la première machine, créez une socket. Précisez le protocole utilisé :
Protocol? [tcp/udp] : tcp
- Associez une adresse IP et un numéro de port à l'aide la primitive **bind**. Notez le numéro de port de la socket ainsi créée.
- Mettez cette socket en attente de demande de connexion à l'aide de la primitive **listen**.
Précisez lors de cette commande le nombre de demande de connexion que TCP peut mémoriser : **Nb of requests [5] : 5**
Vous venez de créer une socket passive d'un **serveur** en attente de demande de connexion de **clients**.
- Puis acceptez sur cette socket une nouvelle connexion à l'aide de la primitive **accept**
- Sur la deuxième machine, créez une socket et associez la à un numéro de port à l'aide de **bind**.
- Lancez une demande de connexion sur le serveur précédemment crée à l'aide la primitive **connect** :

socklab> connect <identificateur> <nom de machine> <numero de port>

Observez la connexion sous *socklab* à l'aide de la commande **status** sur chacune

des deux stations.

 – **10** Pourquoi la première socket créée est dite ”**passive**”, et la seconde ”**active**” ? Quels sont les rôles respectifs de ces deux sockets dans l’établissement de la connexion ?

 – **11** Du côté de la socket passive pourquoi deux sockets existent une fois que vous avez fait **accept** ?

 – **12** Analysez les paquets générés lors de l’établissement de la connexion. Décomposez les étapes de cette connexion : enchaînement dans le temps des demandes de services à TCP et des messages échangés.

 – **13** Quel est le rôle du flag SYN ? Quelles sont les informations échangées durant ces étapes ? A quoi servent les numéros de séquence et d’acquittement ? (attention les numéros donnés par *Wireshark* sont « relatifs », voir les vrais dans l’hexadécimal) A quoi servent chacune des options utilisées lors de l’ouverture de connexion par TCP (voir à la fin de l’entête TCP) ?

 – **14** Expliquez ce qu’il se passe au moment de la primitive des sockets ”*accept*”. Essayez de faire ”*accept*” avant et après le ”*connect*”, que se passe-t-il ?

Ouvrez plusieurs connexions d’une machine vers un même port destinataire. Observez la liste des sockets (commande *status* sous *socklab*) sur le serveur.

 – **15** Dessinez le graphe de l’automate représentant les étapes lors de l’ouverture pour une socket active ainsi que pour une socket dite passive. Les entrées de ces automates sont les commandes des sockets (SOCKET, CONNECT, WRITE...) ou l’arrivée de messages particuliers (DATA, SYN, ACK...), et les sorties sont les envois de messages. Un automate de Mealy semble plus approprié (voir l’ébauche sur la figure 5).

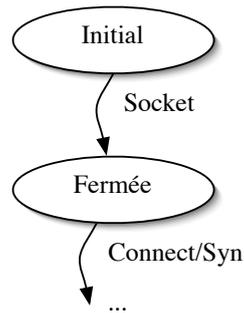


FIGURE 5 – Automate ouverture de connexion TCP pour une socket active

 – **16** Qu'est ce qui identifie réellement une connexion, c'est-à-dire, comment TCP associe les messages reçus aux différentes connexions en cours ?

 **OPTIONNEL** : Vous pouvez observer l'état des connexions TCP sur un ordinateur à l'aide de la commande système ***netstat -a -p tcp***.

Observez l'état d'une connexion pendant les différentes phases de l'ouverture côté serveur et côté client.

 – **17** **OPTIONNEL** : Retrouver la signification de l'état de la connexion (*state*) dans le résultat de la commande *netstat*. Notez sur les automates précédemment dessinés les noms des états apparaissant dans le résultat de la commande *netstat* : LISTEN...

 – **18** Faites une demande de connexion (**Connect**) vers un port inexistant. Expliquez ce qu'il se passe.

2.2.2 Etude du séquençement et de la récupération d'erreur

 Etablissez une connexion TCP entre deux machines, grâce aux commandes vues précédemment.

Echangez des données entre les deux machines grâce aux deux commandes **read** et **write** (à la place d'un message normal, vous pouvez utiliser la notation **#nnn** pour envoyer un message de *nnn* octets). Essayez en particulier avec une taille de donnée importante (10000 octets par exemple).

 – **19** Analysez les paquets engendrés par le transport des données. Expliquez le rôle des champs SEQUENCE NUMBER et ACK NUMBER dans l'entête des paquets TCP.

 – **20** Il y a-t-il toujours un acquittement par paquet de donnée ? Pourquoi ?

 Envoyez un message vers une machine que vous aurez au préalable débranchée du réseau.

Observez ce qui se passe en capturant les paquets sur la machine émettrice. Attention, pour que TCP envoie des réémissions il faut impérativement que seule la machine réceptrice soit débranchée, il faut donc utiliser un HUB ou un switch entre les deux machines.

 – **21** Expliquez en détail les mécanismes utilisés dans ce genre de cas (perte d'un message), comparez avec le protocole UDP.

2.2.3 Buffer d'émission utilisé pour la récupération d'erreur

Avant les manipulations qui suivent, tapez cette commande :

```
sysctl net.inet.tcp.sendbuf_auto=0
```

Elle désactive une option permettant à TCP d'ajuster la taille du buffer d'émission lors d'une connexion.



- Ouvrez une connexion TCP entre les deux machines.
- Sur la socket client modifiez la taille du buffer d'émission à 3000 octets (commande *option* dans *socklab*).
- Envoyez 10 000 octets depuis la machine sur laquelle le buffer d'émission a été réduit. Observez les paquets circulant à ce moment-là.

 – **22** Expliquez le comportement de TCP lors de l'envoi des paquets que vous avez capturés.

Pour rappel,

- Le buffer d'émission permet à TCP de mémoriser les paquets en attente d'acquittement.

- Un émetteur TCP ne peut pas envoyer un paquet s'il ne rentre pas dans le buffer d'émission.
- Si un acquittement arrive le ou les paquets acquittés sont supprimés du buffer.

 – **23** Quelle influence la taille du buffer d'émission a-t-elle sur le débit applicatif ?

 – **24** Dans le cas où le temps d'aller-retour (RTT) sur le réseau est de 20 ms, calculez le débit applicatif que l'on obtiendrait avec un buffer de 1000 octets, 2000 octets, 10000 octets ?

 – **25** Donnez dans ce cas la taille optimale du buffer d'émission.

 – **26** Quel inconvénient peut-on avoir si l'on utilise un buffer d'émission de taille non adéquate à la latence du réseau ? Expliquez pourquoi.

 – **27** Donnez de façon informelle l'algorithme de mise à jour des deux champs ACK et SEQ. Faites attention à distinguer le cas de l'émetteur et du récepteur de données. Pensez aux cas de pertes.

2.2.4 Contrôle de flux

Avant les manipulations qui suivent, tapez cette commande :

```
sysctl net.inet.tcp.recvbuf_auto=0.
```

Elle désactive une option permettant à TCP d'ajuster la taille du buffer de réception lors d'une connexion.

 **Dépassement du buffer de réception**

- sur une socket passive réglez le buffer de réception à 10 000 octets (faire **option** sous Socklab).

Remarque : On peut changer la taille du buffer de réception mais seulement avant l'ouverture de la connexion.

- Lancez *wireshark*.
- Ouvrez une connexion TCP depuis un client sur une autre machine.
- Vérifiez que la taille du buffer de réception annoncé par le serveur (champ *WIN*) est bien celle que vous avez réglé précédemment.

- Emettez depuis le client un message de taille supérieure à celle du buffer de réception (par exemple 12000 octets).

Faites attention à capturer l'ouverture de la connexion TCP (l'option *Winscale* étant utilisée ici), sinon Wireshark affiche des valeurs du champs WIN incohérentes.

 – **28** Analysez les paquets échangés ? Regardez en particulier le champ **Window** de l'entête TCP (au moment de l'ouverture de la connexion et lors des échanges de données/acquittements). L'émetteur peut-il émettre plus d'octets que la taille du buffer de réception ? Pourquoi ?

 Faites ensuite des *read* successifs coté récepteur de 5000 octets et observez les paquets engendrés à ce moment là.

 – **29** Analysez les derniers paquets échangés. Regardez en particulier le champ **Window** de l'entête TCP. Expliquez.

 Refaîtes l'expérience en libérant le buffer du récepteur par des *reads* de peu d'octets.

 – **30** A partir de quel moment l'émetteur est-il « débloqué » ? Pourquoi ?

 – **31** Résumez le principe du contrôle de flux de TCP.

 – **32** Est-il intéressant d'avoir un buffer d'émission plus grand que le buffer de réception ? Expliquez pourquoi.

2.2.5 Libération d'une connexion

 Après ouverture d'une connexion entre deux machines, fermez la connexion : **close**

 – **33** Analysez les paquets générés lors de la fermeture de la connexion de chaque côté. Décomposez les étapes de cette fermeture. Quel est le rôle du flag FIN ?

 – **34** Résumez les échanges de messages en spécifiant la valeur des champs spécifiques à cette phase de la communication.

 Après fermeture d'un seul des deux côtés, essayez de continuer à émettre de l'autre côté (par un *write*).

 – **35** Que se passe t-il (observez les paquets échangés). Si vous refaites un *write* que se passe t-il ? Pourquoi ?

 – **36** Résumez le fonctionnement de TCP en ce qui concerne la fermeture de connexion grâce à un automate dont les entrées sont les commandes des sockets (*close...*) ou l'arrivée de messages particuliers (DATA, FIN, ACK...), et les sorties sont les envois de messages.

Commentez cet automate en rappelant les expérimentations effectuées.

OPTIONNEL : Notez sur cet automate les noms des états apparaissant dans le résultat de la commande *netstat -a -p tcp* : CLOSE, CLOSE_WAIT ...

2.3 Exercices de synthèse (A faire attentivement)

- Observation de l'application de dialogue interactif *talk*

 Faites un échange de caractères entre deux machines à l'aide de la commande **talk** (`talk <nom utilisateur>@<nom de machine officiel>`).

Exemple : `talk root@machine1`

Attention : Dans la commande **talk**, il faut utiliser impérativement un nom que vous aurez déclaré dans le fichier `/etc/hosts` .

Il est souvent nécessaire d'exécuter au préalable la commande *mesg y* afin d'autoriser les messages d'autres utilisateurs.

Pour des raisons obscures, il faut aussi sortir de **xinit** pour pouvoir recevoir une demande de **talk** (côté serveur).

Capturez sur une des machines les messages échangés lors de ce dialogue.

 – **37** Analysez les paquets capturés. Expliquez ce qu'il se passe au niveau du réseau (protocole transport utilisé, numéro de port utilisés, ouverture de connexion, échange des caractères, fermeture de connexion, ...).

- Observation d'une application de commandes à distance

 Aujourd'hui on utilise l'application *ssh* pour lancer des commandes à distance sur un ordinateur. Nous allons ici observer un des deux ancêtres de *ssh* (*telnet* ou *rlogin*) qui ont l'avantage de ne pas chiffrer les données échangées. Comme précédemment, sur une des machines, capturez les paquets circulant sur le réseau.

- Lancez l'application *rlogin* ou à défaut *telnet* d'un ordinateur sur une autre par la commande `rlogin <adresseIP>` (ou `telnet -y <adresseIP>`), il faut ensuite indiquer le compte (login) *guest*, mot de passe *guest./*).
- Puis procédez à une ou deux commandes Unix à travers cette application (*pwd*, *ls* ...).

 – **38** Enumérez les messages échangés lors de cette manipulation et expliquez ce qui se passe au niveau du réseau (protocole transport utilisé, numéro de port utilisés, ouverture de connexion, échange des caractères, fermeture de connexion, ...).
Comment les commandes sont elles envoyées à destination ?
Pourquoi les caractères des commandes sont ils renvoyés à la source ?

Regardez le contenu du fichier `/etc/services` et retrouvez les numéros de port des serveurs `talk`, `rlogin` et `telnet`.

Références Bibliographiques :

- Réseaux locaux et Internet (des protocoles à l'interconnexion) Laurent Toutain – 2ème Edition - HERMES
- Analyse structurée des Réseaux Des Applications de l'Internet aux infrastructures de télécommunication
James Kurose et Keith Ross
2^e Edition - Pearson Education
- https://fr.wikipedia.org/wiki/User_Datagram_Protocol
- https://fr.wikipedia.org/wiki/Transmission_Control_Protocol