

# ELEC 2920 : Réseaux de communication

## Laboratoire sur le comportement ‘‘TCP friendly’’

Assistant : Cédric De Roover

Oscar Medina Duarte  
Aubry Springuel

### 1. Explication théorique du fonctionnement

#### **A. UDP : User Datagram Protocol (RFC 768)**

Ce protocole permet l'application de programmes pour envoyer des messages à d'autres programmes, assurant le service le plus coulant, ce qui veut dire que les datagrammes UDP peuvent être perdus et/ou délivrés de manière défectueuse. Pour cette raison, des applications requérant des flots ordonnés et fiables de données peuvent utiliser le protocole TCP (Transmission Control Protocol).

UDP fournit un service sans connexion, ce qui veut dire qu'il n'y a pas d'établissement de connexion, pas d'état de connexion entre l'émetteur et le récepteur et que chaque datagramme est traité indépendamment des autres. Ainsi, il n'y a pas de contrôle de flux, ni de contrôle de congestion et de temps ou la garantie d'une certaine bande passante.

D'un autre côté, les avantages de l'UDP sont qu'il n'y a pas de 'poignée de main' entre l'émetteur et le récepteur, le protocole est donc plus simple. De ce fait, la tête de l'UDP est plus petite et il n'y a pas de setup de connexion et donc moins de délai.

Le format d'un datagramme UDP est le suivant :

Source Port	Destination Port
Length	Checksum
Application Data (message)	

## **Structure d'un segment UDP**

Différents champs de l'en-tête :

- 1 Numéro d'accès d'origine : Cela indique le port du process d'émission, et on peut supposer que c'est le port vers lequel la réponse peut être adressée en l'absence d'autre information.
- 2 Numéro d'accès de destination : Cela indique le port lié à l'adresse Internet de destination.
- 3 Taille : La longueur en octets du datagramme incluant l'en-tête et les données.
- 4 Somme de contrôle : Cela permet de détecter des erreurs. UDP fait le complément à 1 de la somme de tous les mots de 16 bits du segment en éliminant tout dépassement de capacité et reporte le résultat dans le champ de la somme de contrôle.

Les plus communes des utilisations d'UDP sont le DNS et le SNMP mais il est aussi utilisé pour le streaming dans les applications multimédia, lesquelles tolèrent certaines pertes. UDP peut également être utilisé pour un transfert fiable, bien que la gestion de la fiabilité soit alors située au niveau de la couche d'application. Ces applications ont des error-recovery spécifiques, de cette manière, comme UDP n'a pas de contrôle de flux ou de congestion, elles peuvent ainsi utiliser toute la bande passante, et avoir un meilleur contrôle de leurs applications.

## **Les réactions d'UDP à la congestion**

Comme cela a été vu, UDP ne propose pas de contrôle de la congestion. Ainsi, quand elle apparaît, rien n'est fait. La transmission continue par la même voie, et il y aura probablement peu de pertes de paquets par suite de la congestion à la queue du routeur. S'il y a des flux UDP et TCP au même moment, et de la congestion, TCP réduira son débit de transfert pour éviter la congestion tandis qu'UDP ne le fera pas, comme nous pouvons le voir dans le labo.

## **B. TCP IP**

L'importance de disposer de communications robustes, fiables et faciles d'usage est essentielle pour la sûreté au sein des systèmes de partage présent dans de nombreuses applications. Le TCP (Transmission Control Protocol) prend soin de ces exigences, spécialement celle du maintien de la permanence de communication fiable, disponible et cela, également, quand il y a de la congestion de données.

TCP est orienté connexion, un protocole fiable désigné pour s'adapter dans une hiérarchie à couches de protocoles dans lequel se trouvent des applications multi-network. Ce protocole fournit une communication fiable entre les pairs de process d'ordinateurs hôtes à distinguer au sein d'un réseau de communication d'ordinateurs interconnectés. Le TCP a l'intention de fournir un service de communication de process à process fiable.

Au sein du TCP, il est possible d'y avoir :

- Transfert de données basique
- Fiabilité
- Contrôle du flux
- Multiplexage
- Connexions
- Préséance et sécurité

Le TCP est capable de transférer un stream continu d'octets dans chaque direction entre les utilisateurs par le paquetage d'un certain nombre d'octets dans des segments pour la transmission au travers du système internet. En général, le TCP détermine quand bloquer ou transférer les données en fonction de ses propres règles. La fiabilité du TCP est assurée avec la définition d'une fonction push. Un push fait que le TCP va transférer plus promptement et délivrer les données de ce point au destinataire. Le point exact de push ne peut être visible par l'utilisateur à la réception.

Pour garantir que les données arrivent sans dommage, perte ou duplication, ou bien dans un mauvais ordre à cause du système de communication, il est nécessaire qu'il soit capable de rétablir ce qui a été envoyé. Cela est réalisé par l'assignation d'une séquence de nombres à chaque octet transmis et la nécessité d'une approbation (ACK) de la réception TCP. Si l'ACK n'est pas reçu dans un intervalle de temps, les données sont retransmises. Au niveau du récepteur, des séquences de nombres sont utilisées pour ordonner correctement les segments, qui ont peut-être été mélangés, et pour éliminer les duplicatas. Les détériorations sont traitées par l'ajout d'un checksum pour chaque segment transmis, la vérification de celui-ci par le destinataire et l'écartement des segments endommagés. Tant que les TCP continuent à fonctionner convenablement et que le système internet ne devient pas complètement partitionné, les erreurs de transmission n'affecteront les transferts de données.

Le flux de données est très important pour assurer que toutes les données auront une partition de la bande passante adéquate pour atteindre leur cible. Pour faire cela, il est possible que TCP fournisse un moyen pour que le récepteur décide de la quantité de données à envoyer par l'émetteur. Cela est réalisé par le renvoi d'une 'fenêtre' avec chaque ACK indiquant le niveau de séquence de nombres acceptable ainsi que le dernier segment reçu avec succès. La fenêtre indique un nombre alloué d'octets que l'émetteur peut transmettre jusqu'à une prochaine permission du récepteur.

Un autre problème qui a été résolu grâce à une communication TCP est de permettre d'utiliser simultanément de plusieurs process au sein d'un hôte particulier sans difficulté. Le TCP fournit une série d'adresses ou de ports pour chaque hôte. Ils sont joints avec le réseau et les adresses hôtes d'internet, ils forment un 'socket'. Une paire de sockets identifie de manière unique chaque connexion. Cela étant, un socket peut être simultanément utilisé dans de multiples connexions.

La sécurité et la préséance des communications de l'utilisateur TCP nécessite d'être indiquée, une valeur défaut est spécifiée quand ces caractéristiques ne ont pas été indispensables.

Pour garantir que le contrôle du flux possède la fiabilité et les autres mécanismes décrits ci-

dessus, il est nécessaire d'initialiser et de maintenir une information sur le statut de chaque stream de données. Quand deux process souhaite communiquer, leur TCP doit d'abord établir une connexion. Quand leur communication est finie, la connexion se termine ou se ferme pour libérer les ressources pour les autres utilisateurs. Lorsque des connexions doivent être établies entre deux hôtes non fiables et cela dans le système de communication non fiable qu'est internet, on procède à un mécanisme de 'poignée de main' pour établir la communication.

Les segments TCP sont envoyés comme des datagrammes internet. Une en-tête TCP comprend une en-tête internet, qui fournit l'information spécifique au protocole TCP. Ce mécanisme assure l'existence d'un niveau hôte pour les protocoles autre que TCP.

### **Les datagrammes TCP sont constitués des éléments suivant:**

- 1 Numéro d'accès d'origine : Cela indique le numéro de port du process d'émission.
- 2 Numéro d'accès de destination : Cela indique le numéro de port de destination.
- 3 Champ de numéro de séquence et un champ d'accusé de réception : Ils sont utilisés par l'expéditeur et le destinataire TCP pour la mise en œuvre du service de transfert fiable.
- 4 La fenêtre de réception ou 'window' : Elle comporte 16 bits servant au contrôle de flux. Celle-ci sert à indiquer le nombre d'octets qu'un destinataire peut prendre en charge.
- 5 Champ de taille d'en-tête : Il a 4 bits qui spécifient la longueur de l'en-tête TCP en termes de mots de 32 bits. Le champ d'options provoque une certaine variabilité dans ce champ. Mais, ce champ d'options étant généralement vide, la taille d'en-tête TCP est souvent de 20 bits.
- 6 Le champ d'options : Il a une longueur variable et intervient au moment de la négociation de la taille du MSS entre expéditeur et destinataire. Il existe également une option d'horodatage.
- 7 Champ de fanion : Il comprend 6 bits. Le bit ACK sert à indiquer que la valeur contenue dans le champ d'accusé de réception est bonne. Les bits RST, SYN et FIN sont utilisés pour l'établissement et la rupture de la connexion TCP. Il y a un bit PSH réglé à 1 qui signifie que le destinataire doit immédiatement remettre les données à la couche supérieure. Le bit URG sert, pour sa part, à indiquer que le segment contient des données urgentes selon la couche supérieure du pôle expéditeur. L'emplacement du dernier octet de ce groupe de données urgentes est indiqué par le champ de données urgentes de 16 bits. TCP doit informer cette entité de l'existence de données urgentes et lui remettre un tel pointeur à la suite de ces données.
- 8 Somme de contrôle ou 'checksum' : Cela permet de détecter des erreurs. UDP fait le complément à 1 de la somme de tous les mots de 16 bits du segment en éliminant tout dépassement de capacité et reporte le résultat dans le champ de la somme de contrôle.

Après cette brève explication à propos du TCP et de l'UDP, il sera plus évident d'observer ce qui se passe dans la simulation et d'expliquer comment certains phénomènes se déroulent.

## 2. Résultats de la simulation

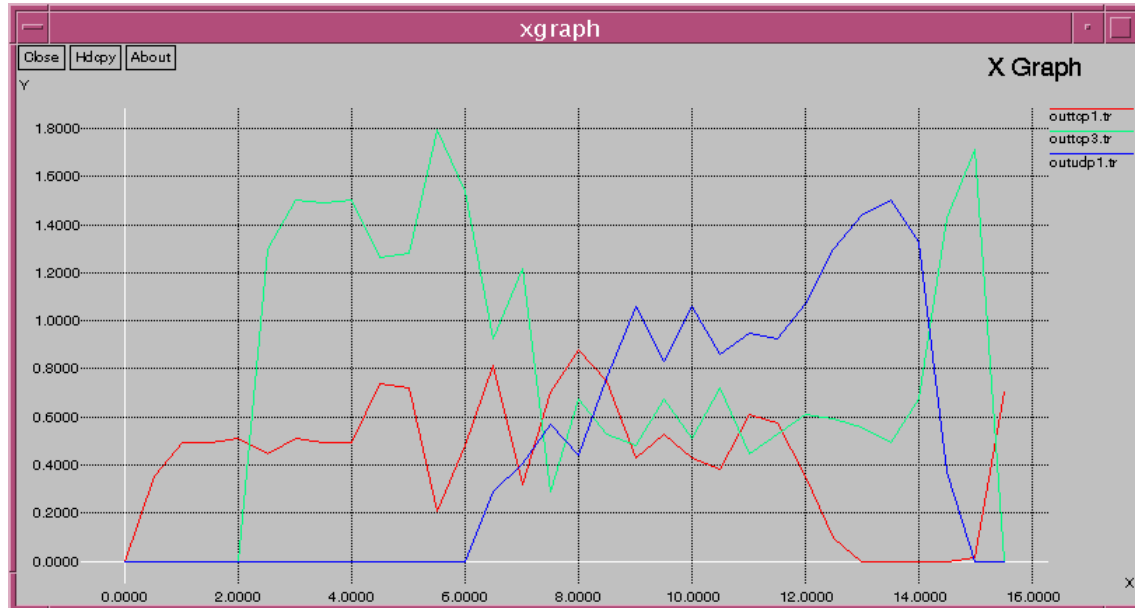


Figure : Graphe de l'évolution de la bande passante au cours du temps

Le but de la simulation était de montrer certaines des caractéristiques du TCP concernant la fiabilité en utilisant ns (Network Simulator) et de montrer comment l'usage de la bande passante est affecté par quelques événements générés dans le réseau suggéré pour le labo parmi lesquels les principaux éléments sont:

A1-Début d'une émission de paquets TCP après 0,1s, avec un débit de 500 kb/s, et après 4s débit de 1 Mb/s

A2-Début d'une émission de paquets TCP après 2s, avec un débit de 1,5 Mb/s et arrêt après 10s.

A3-Début d'une émission de datagrammes UDP après 6s au débit de 500 kb/s, passant à 1Mb/s à 8s, et s'élevant ensuite à 2,5 Mb/s après 12s, puis stoppant à 14s.

Après l'exécution de la simulation (voir le code) appliquant ces paramètres, nous obtenons un graphe de l'évolution des bandes passantes occupées par chacun des agents (A1, A2, A3) à partir duquel il est appréciable qu'il soit interprété selon les 'fourchettes' de temps suivantes:

de 0 à 2 secondes

Seul A1 envoie des paquets au débit de 500kb/s sans problème.

de 2 à 4 secondes

A2 commence l'envoi de paquets au débit de 1,5 Mb/s. C'est un temps pendant lequel A1 réduit légèrement son débit d'émission pour qu'ensuite les deux stabilisent leur débit.

de 6 à 8 secondes

A3 commence l'envoi de datagrammes UDP à 500 kb/s, ce qui cause la perte de paquets envoyé par A1, A2 et A3. Puis A1 et A2 réajustent leur débit à la baisse.

de 8 à 10 secondes

A3 élève son débit d'émission à 1 Mb/s. Cela cause un peu plus de perte de paquets. Ce qui mène A1 et A2 à un débit plus faible, pour petit à petit partager leur ressource à un niveau de 400 kb/s.

de 10 à 12 seconds

A2 arrête l'envoi de paquets, mais occupe encore de la bande passante parce qu'il doit renvoyer la queue des paquets. Mais cette queue occupe moins de bande passante. Si bien que le temps pour lequel l' agent A2 a complètement fini est la seconde 13.

Comme A2 libère de la bande passante, A3 et A1 peuvent occuper un débit plus élevé.

de 12 à 14 secondes

A3 élève à nouveau son taux d'émission à 2,5 Mb/s mais atteint seulement environ 1,5 Mb/s parce qu'il y a encore un partage de la bande passante disponible avec A1 qui émet toujours.

de 14 à 16 secondes

A3 arrête d'émettre des datagrammes, ce qui donne l'espace à A2 d'émettre au niveau voulu.

Dans le graphique, nous pouvons observer que cela donne une émission plus élevée qu'à l'origine parce qu'il y a encore émission

de paquets pour qui le temps dans la queue d'admission s'est écoulé.

SI nous observons le graphe de la distance, nous pouvons apprécier le fait que quand l'utilisation de la bande passante est plus grande, le débit de l'émetteur TCP devient bloqué et change de nombreuses fois pour obtenir la meilleure utilisation qu'il peut de la bande passante.

## Annexes : code du script

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the output file
set ftp1 [open outtcp1.tr w]
set ftp3 [open outtcp3.tr w]
set fudp1 [open outudp1.tr w]

#Define a 'finish' procedure
proc finish {} {
    global ns nf
    global ftp1 ftp3 fudp1
    $ns flush-trace
    #Close the NAM trace file
    close $nf
    #Close output files
    close $ftp1
    close $ftp3
    close $fudp1
    #Execute NAM on the trace file
    exec nam out.nam &
    exec xgraph outtcp1.tr outtcp3.tr outudp1.tr -geometry 800x400 &
    exit 0
}

#Define a 'record' procedure
proc record {} {
    #global tcp1 tcp3 tcp2 tcp4 udp2 ftp1 ftp3 fudp
    global tcp1 tcp2 ftp1
    global tcp3 tcp4 ftp3
    global null fudp1
    #Get an instance of the simulator
    set ns [Simulator instance]
    #Set time after which the procedure should be called again
    set time 0.5
    #How many bytes have been received by the traffic sink ?
    set packetreceived [$tcp1 set nackpack_]
    set bwtcp1 [expr $packetreceived*1000]
    set packetreceived [$tcp3 set nackpack_]
```

```

set bwtcp3 [expr $packetreceived*1000]

set bwudp1 [$null set bytes_]

#Get the current time
set now [$ns now]

#Calculate the bandwidth (in MBit/s) and write it to the file
puts $ftcp1 "$now [expr $bwtcp1/$time*8/1000000]"
puts $ftcp3 "$now [expr $bwtcp3/$time*8/1000000]"
puts $fudp1 "$now [expr $bwudp1/$time*8/1000000]"

#Reset the bytes_ values on the traffic sink
$tcp1 set nackpack_ 0
$tcp3 set nackpack_ 0
$null set bytes_ 0
#Re-schedule the procedure
$ns at [expr $now+$time] "record"
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 12Mb 1.25ms DropTail
$ns duplex-link $n1 $n2 12Mb 1.25ms DropTail

$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail

$ns duplex-link $n3 $n4 12Mb 1.25ms DropTail
$ns duplex-link $n3 $n5 2Mb 10ms DropTail

$ns duplex-link $n5 $n6 12Mb 1.25ms DropTail
$ns duplex-link $n5 $n7 12Mb 1.25ms DropTail
$ns duplex-link $n5 $n8 12Mb 1.25ms DropTail

#Set Queue Size of link (n1-n2) to 50

$ns queue-limit $n2 $n3 50
$ns queue-limit $n3 $n5 50

```

#Give node position (for NAM)

\$ns duplex-link-op \$n0 \$n2 orient right-up  
\$ns duplex-link-op \$n1 \$n2 orient right-down

\$ns duplex-link-op \$n2 \$n3 orient right

\$ns duplex-link-op \$n3 \$n4 orient up  
\$ns duplex-link-op \$n3 \$n5 orient right

\$ns duplex-link-op \$n5 \$n6 orient right-up  
\$ns duplex-link-op \$n5 \$n7 orient right  
\$ns duplex-link-op \$n5 \$n8 orient right-down

#Monitor the queue for link (n1-n2). (for NAM)

\$ns duplex-link-op \$n2 \$n3 queuePos 0.5  
\$ns duplex-link-op \$n3 \$n5 queuePos 0.5

#Setup a TCP connection

set tcp1 [new Agent/TCP]  
\$tcp1 set class\_ 2  
\$ns attach-agent \$n0 \$tcp1  
set tcp2 [new Agent/TCPSink]  
\$ns attach-agent \$n6 \$tcp2  
\$ns connect \$tcp1 \$tcp2  
\$tcp1 set fid\_ 1  
\$tcp1 set window\_ 64

#Setup a CBR over TCP connection

set cbrtcp1 [new Application/Traffic/CBR]  
\$cbrtcp1 attach-agent \$tcp1  
\$cbrtcp1 set type\_ CBR  
\$cbrtcp1 set packet\_size\_ 1000  
\$cbrtcp1 set rate\_ 500kb

#Setup a TCP connection

set tcp3 [new Agent/TCP]  
\$tcp3 set class\_ 2  
\$ns attach-agent \$n4 \$tcp3  
set tcp4 [new Agent/TCPSink]  
\$ns attach-agent \$n7 \$tcp4  
\$ns connect \$tcp3 \$tcp4  
\$tcp3 set fid\_ 1

```
$tcp3 set window_ 64
```

```
#Setup a CBR over TCP connection  
set cbrtcp3 [new Application/Traffic/CBR]  
$cbrtcp3 attach-agent $tcp3  
$cbrtcp3 set type_ CBR  
$cbrtcp3 set packet_size_ 1000  
$cbrtcp3 set rate_ 1500kb
```

```
#Setup a UDP connection  
set udp [new Agent/UDP]  
$ns attach-agent $n1 $udp  
set null [new Agent/LossMonitor]  
$ns attach-agent $n8 $null  
$ns connect $udp $null  
$udp set fid_ 2
```

```
#Setup a CBR over UDP connection  
set cbrudp [new Application/Traffic/CBR]  
$cbrudp attach-agent $udp  
$cbrudp set type_ CBR  
$cbrudp set rate_ 500kb
```

```
#Schedule events for the CBR and FTP agents  
$ns at 0.0 "record"  
$ns at 0.1 "$cbrtcp1 start"  
$ns at 2.0 "$cbrtcp3 start"  
$ns at 4.0 "$cbrtcp1 set rate_ 1000kb"  
$ns at 6.0 "$cbrudp start"  
$ns at 8.0 "$cbrudp set rate_ 1000kb"  
$ns at 10.0 "$cbrtcp3 stop"  
$ns at 12.0 "$cbrudp set rate_ 2500kb"  
$ns at 14.0 "$cbrudp stop"  
$ns at 16.0 "$cbrtcp1 stop"
```

```
#Call the finish procedure after 16 seconds of simulation time  
$ns at 16.0 "finish"
```

```
#Run the simulation  
$ns run
```