TD : Etude du protocole TCP avec le simulateur ns-2

1 Le simulateur ns-2

Le simulateur ns-2 (`http://www.isi.edu/nsnam/ns`) permet de modéliser des réseaux de transmission filaires, sans fil, cellulaires et satellitaires. Il permet d'étudier des protocoles réseaux de toutes les couches du modèle OSI. Il est écrit en C++ et en OTcl, une variante objet du langage de script Tcl. Sa documentation est disponible sous de multiples formats ici : `http://www.isi.edu/nsnam/ns/ns-documentation.html`.

2 Le script d'une simulation

Dans la suite du TD, les arguments entre `<...>` sont à remplacer par les valeurs que vous souhaitez. ns-2 s'exécute depuis un shell et s'utilise de façon interactive en tapant des commandes Tcl. Cependant, afin de faciliter son utilisation, on regroupe en général l'ensemble des instructions Tcl d'une simulation dans un fichier nommé `<mon-script>.tcl` pour utiliser ce dernier comme un script en tapant la ligne de commande : `ns <mon-script>.tcl`. Le début d'un script contient des instructions pour initialiser le simulateur. Ensuite généralement on crée des fichiers de trace dont un qui contient l'ensemble des évènements de la simulation et un qui contient l'animation. Le fichier de sortie `anim.nam` qui contient l'animation peut être visualisé graphiquement à l'aide du programme `nam`. Ci-dessous un début typique de script ns-2 :

```
set ns [new Simulator]
set namfile [open anim.nam w]
$ns namtrace-all $namfile
set tracefile [open trace.txt w]
$ns trace-all $tracefile
```

La fin d'un script contient une procédure `finish` qui clot les fichiers. Elle contient aussi les instructions liées à l'ordonnanceur, ainsi que celle qui exécute la simulation. Ci-dessous une fin de script ns-2 :

```
proc finish {} {
    global ns namfile tracefile
    $ns flush-trace
    close $tracefile
    close $namfile
    nam $namfile &
    exit 0
}

$ns at <temps-en-secondes> "<commande-tcl>"
$ns at <temps-final-en-secondes> "finish"
$ns run
```

Un tutoriel qui vous permettra de vous familiariser avec la syntaxe et l'utilisation de ns-2 se trouve ici : `http://www.isi.edu/nsnam/ns/tutorial/index.html`. Dans ce tutoriel, faites dès maintenant la section IV et éventuellement la section V afin de connaître les bases de ns-2.

3 Enregistrement et tracé des valeurs d'une variable

Pour suivre l'évolution d'une variable, ce qui sera utile par la suite, on utilise une procédure telle que celle présentée ci-dessous. Avec cette procédure nommée `plot_cwnd` par exemple, nous enregistrons la valeur de la variable `cwnd_` d'un agent TCP nommé `tcp0` toutes les 0.1 secondes dans un fichier nommé `plot.txt` sur le disque dur.

```
proc plot_cwnd {tcp file} {
    global ns
    set time_interval 0.1
    set now [$ns now]
    set cwnd [$tcp set cwnd_]
    puts $file "$now $cwnd"
    $ns at [expr $now + $time_interval] "plot_cwnd $tcp $file"
}
```

Nous appelons cette procédure dans ns-2 en plaçant la ligne :
`$ns at 0.1 "plot_cwnd $tcp0 $plot_file"`
au début des appels à l'ordonnanceur (`$ns at...`).
Pour afficher le graphe montrant l'évolution de la variable, on utilise le programme `gnuplot`. Il est déjà installé sur les machines. Sa documentation se trouve ici : `http://www.gnuplot.info/`. Si le fichier stockant la variable s'appelle `plot.txt`, taper simplement au prompt de `gnuplot` :
`gnuplot> plot 'plot.txt' w lines`.

4 Mécanismes du démarrage lent et d'évitement de la congestion

Dans un transfert TCP, l'émetteur utilise une fenêtre de retransmission dynamique. La taille de cette fenêtre est égale au minimum entre la valeur notée `window_` fournie par le récepteur (*advertised window*) et la valeur notée `cwnd_` (*congestion window*) qui est un entier qui évolue en fonction des pertes de paquets constatées. Dans cette section nous allons observer les mécanismes du démarrage lent et de l'évitement de la congestion. La taille d'un segment en octets est notée *Segment Size (SS)*. TCP utilise alternativement deux algorithmes. Pour augmenter le débit, il utilise l'algorithme du démarrage lent (*slow start*) :
– Au démarrage : `cwnd_ = 1 SS`.
– A chaque ACK reçu portant sur une valeur plus grande que les ACK précédemment reçus : `cwnd_ = cwnd_ + 1 SS`.
– On transmet la fenêtre de taille : `min(window_, cwnd_)`.
Notons qu'à chaque *Round Trip Time* (RTT), la taille de `cwnd_` double (RTT 1 : 1 paquet + 1 ACK ; RTT 2 : 2 paquets + 2 ACK ; RTT 3 : 4 paquets + 4 ACK ; etc). Cette croissance dite additive est donc exponentielle et s'arrête lorsque `cwnd_ = window_` ou lorsque des segments sont perdus. Dans ce dernier cas où des pertes de données sont détectées par l'expiration d'un timeout, la moitié de la valeur de `cwnd_` est sauvegardée dans une variable de seuil qui est un entier nommé *slow start threshold* (dans ns-2 c'est la variable `ssthresh_`) et l'algorithme *slow start* reprend avec sa fenêtre `cwnd_` initiale. Plus précisément, les variables sont fixées comme suit :

```
ssthresh_ = max(cwnd_ / 2, 2)
cwnd_ = 1
```

Ensuite, lorsque `cwnd_` atteint `ssthresh_`, TCP bascule en mode d'évitement de congestion où chaque ACK augmente `cwnd_` de $SS \times SS/$`cwnd_` octets. Pour diminuer le débit, TCP utilise donc l'algorithme d'évitement de congestion (*congestion avoidance*) :
– A chaque ACK reçu portant sur une valeur plus grande que les ACK précédemment reçus : `cwnd_ = cwnd_ + (SS * SS)/cwnd_`.
– On transmet la fenêtre de taille : `min(window_, cwnd_)`.

– Notons qu'à chaque RTT, la taille de `cwnd_` augmente au maximum de 1 segment.
Le passage d'un algorithme à l'autre est donc guidé par l'algorithme suivant :

```
if (cwnd_ < ssthresh_)
    cwnd_ = cwnd_ + 1 SS
else
    cwnd_ = cwnd_ + (SS * SS) / cwnd_
```

Le principe utilisé par le démarrage lent et l'évitement de le congestion est donc une croissance additive et une décroissance multiplicative. Récupérer le fichier `demar-lent.tcl`. Le compléter en écrivant les instructions manquantes à la place des commentaires `#VOTRE CODE ICI`. Créez dans ce script la topologie ci-dessous :

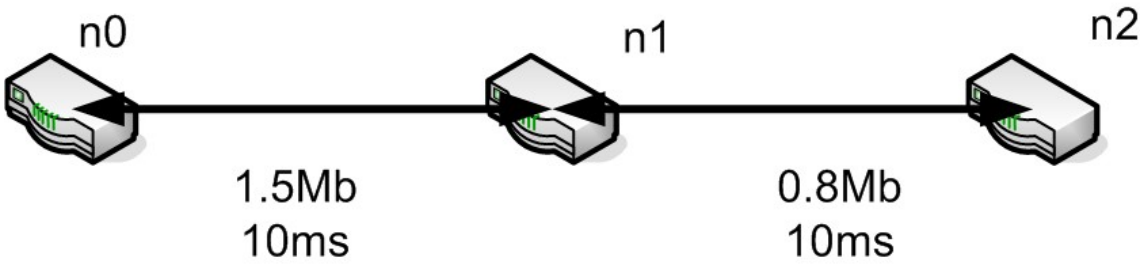

FIGURE 1 – Topologie réseau pour le démarrage lent.

La taille de la file d'attente (*queue*) du lien sera fixée à 10 et son type sera fixé à `Droptail`. Lancez la simulation en tapant : `ns demar-lent.tcl`. Visualisez le résultat en tapant : `nam out.nam` si cette ligne n'est pas déjà dans votre script. Enregistrez l'évolution des variables `cwnd_` et `ssthresh_` en utilisant le code donné ci-dessus puis visualisez les avec `gnuplot`. Observez et commentez les résultats.

5 Mécanismes de retransmission rapide et de récupération rapide

Dans cette section nous allons observer les mécanismes de retransmission et de récupération rapide. Avec la retransmission rapide, si 4 ACK identiques sont reçus, TCP n'attend pas l'expiration du *timer* mais réémet immédiatement les trames (mode *fast retransmission*). Avec le mode de récupération rapide (*fast recovery*), l'émetteur réémet le 1er segment non acquitté puis attend un ACK. En cas de non réception de cet ACK, et selon les versions de TCP, le comportement diffère :
– TCP Tahoe : redémarrage à `cwnd_ = 1 SS`.
– TCP Reno : si en mode *congestion avoidance*, redémarrage à `cwnd_ = ssthresh_`.
– TCP New Reno : il modifie le comportement de TCP Reno lorsqu'on reçoit un ACK partiel c'est-à-dire un ACK qui acquitte au moins le segment perdu (celui qui nous a fait entrer en récupération rapide) mais pas tous les segments envoyés. Lors de la réception d'un tel ACK, TCP Reno quitte le mode *fast recovery* ce qui pose un problème de performance lorsque plusieurs segments d'une même fenêtre sont perdus (les trames sont souvent perdues en rafale). TCP New Reno ne quitte le mode *fast recovery* que si l'ACK reçu acquitte tous les segments envoyés. A la réception d'un ACK partiel, TCP New Reno retransmet immédiatement le paquet suivant le dernier paquet acquitté dans cet ACK, diminue la taille de la fenêtre du nombre de paquets acquittés par cet ACK partiel et retransmet un paquet si possible par la taille de `cwnd_`.

Récupérer le fichier `retransmit-rapide.tcl`. Le compléter en écrivant les instructions manquantes à la place des commentaires `#VOTRE CODE ICI`. Créez dans ce script la topologie ci-dessous :

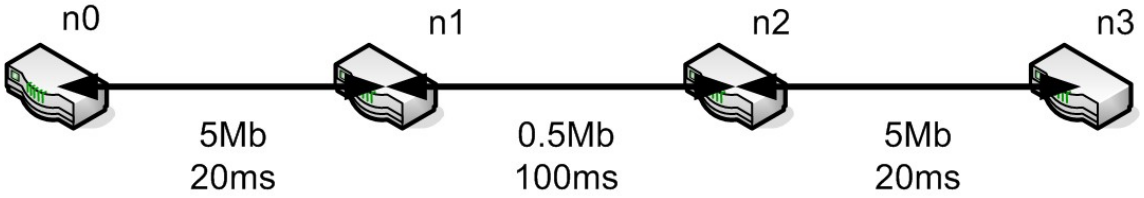

FIGURE 2 – Topologie réseau pour la retransmission rapide.

La taille de la file d'attente (*queue*) du lien n1-n2 sera fixée à 5 et son type sera fixé à `Droptail`. Lancez la simulation en tapant : `ns retransmit-rapide.tcl`. Visualisez le résultat en tapant : `nam out.nam` si cette ligne n'est pas déjà dans votre script. Enregistrez l'évolution des variables `cwnd_` et `ssthresh_` en utilisant le code donné ci-dessus puis visualisez les avec `gnuplot`. Observez et commentez les résultats. Modifiez votre script pour utiliser l'agent `TCP/Newreno`. Refaites la simulation et visualisez les résultats en les superposant aux résultats précédents. Observez et commentez les nouveaux résultats. Que constatez vous par rapport à la version `TCP/Reno` ?

6 Mécanisme des ACK retardés

Dans cette section nous allons observer le mécanisme des acquittements retardés. Dans ce cas, un seul acquittement est envoyé pour d paquets reçus. Par défaut d est fixé à 2 (cf. RFC1122). Cependant pour éviter tout blocage (e.g., fenêtre de retransmission valant 1), lorsqu'un premier paquet arrive, un *timer* est enclenché et si celui-ci arrive à expiration avant que les d paquets soient reçus, un acquittement est immédiatement envoyé. Ce timer est fixé en général à une valeur de 100ms (`interval_`). Récupérer le fichier `ack-retardes.tcl`. Le compléter en écrivant les instructions manquantes à la place des commentaires `#VOTRE CODE ICI`. Créez dans ce script la topologie ci-dessous :

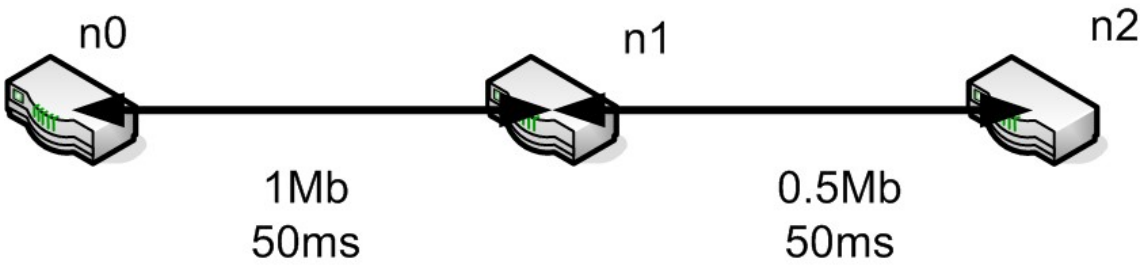

FIGURE 3 – Topologie réseau pour les ACK retardés.

Les trois variables du début du script permettent de faire varier la taille de la file d'attente du lien n1-n2, l'intervalle des ACK et la durée de la simulation respectivement. La taille de la file d'attente (*queue*) du lien n1-n2 sera fixée à 10 et son type sera fixé à `DropTail`. Fixez l'intervalle des acquittements à 100ms et la durée de simulation à 10s. Lancez la simulation en tapant : `ns ack-retardes.tcl`. Visualisez le résultat en tapant : `nam out.nam` si cette ligne n'est pas déjà dans votre script. Enregistrez puis visualisez avec `gnuplot` l'évolution des variables `seqno_` et `ack_`. Observez et commentez les résultats. Que constatez vous ? Fixez maintenant l'intervalle des acquittements à 200ms puis 500ms. Refaites la simulation et décrivez ce qui se passe.