TP MPEV N°1
Introduction à Network Simulator

M. Haddad

Résumé

Network Simulator ou plus communément NS est un logiciel libre de simulation par événements discrets très largement utilisé dans la recherche académique et dans l'industrie (*cf. Wikipedia*). Il est considéré par beaucoup de spécialistes des télécommunications comme le meilleur logiciel de simulation par événements discrets, en raison de son modèle libre, permettant l'ajout très rapide de modèles correspondant à des technologies émergentes. Dans ce TP, nous aborderons différents aspects liés à la simulation dans NS2, en particulier, la création de topologies, les flux constants et FTP, la communication par échanges de messages ainsi que quelques débuts dans les environnements ad hoc (wireless).

1 Installation

1.1 Plan A

Téléchargez l'archive *ns-allinone-2.34.tar.gz* à partir de l'adresse :

`http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.34/`

Décompressez la (avec : tar -xvf ns-allinone-2.34.tar.gz).
Entrez dans le répertoire ns-allinone-2.34 et exécutez la commande ./install
Remarque : La compilation du module *otcl* peut poser problème sur la version du système installée (code incompatible avec la gcc-4.4). Pour y remédier, accédez au répertoire *otcl-1.13* et éditer le fichier Makefile.in puis changez la ligne CC = @CC@ par CC = gcc-4.3
Relancer l'installation.
Un résumé de variables d'environnement à déclarer ou à mettre à jour vous est alors donné. Faites le nécessaire en éditant ou en créant le fichier *.profile* de votre session.
Déconnectez-vous puis reconnectez-vous.

1.2 Plan B

Dans le cas où vous n'avez pas assez d'espace pour faire l'installation. Éditez ou créez le fichier *.profile* de votre session et ajoutez-y les lignes suivantes :

```
PATH=$PATH:/home/pers/mohammed.haddad/ns-allinone-2.34/bin:/home/pers/mohammed.haddad/
ns-allinone-2.34/tcl8.4.18/unix:/home/pers/mohammed.haddad/ns-allinone-2.34/tk8.4.18/unix
export PATH

LD_LIBRARY_PATH=/home/pers/mohammed.haddad/ns-allinone-2.34/otcl-1.13:/home/pers/
mohammed.haddad/ns-allinone-2.34/lib
export LD_LIBRARY_PATH

TCL_LIBRARY=/home/pers/mohammed.haddad/ns-allinone-2.34/tcl8.4.18/library
export TCL_LIBRARY
```

Déconnectez-vous puis reconnectez-vous.

2 Premiers pas

Tout au long du TP, construisez votre compte-rendu en prenant note et en commentant les différentes étapes que vous réalisez. Nous commencerons par l'étude su script TCL suivant. L'exécution d'un fichier se fait via la commande : *ns fichier.tcl*
Concidérons le code suivant :

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

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

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

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
```

```
#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

1. A quel moment est appelée la procédure finish ? que fait-elle ?
2. Quelle est la différence entre TCP et UDP ? et dans leurs implémentations NS ?
3. Sur quelle couche de transport circule un flux FTP ?
4. Pourquoi avant de communiquer, les agents doivent être attachés ?
5. Quelles est la taille par défaut d'un paquet CBR ?

3 Topologie du réseau et routage dynamique

Dans cette section, nous allons considérer les topologies suivantes : Anneau, Grille et Grille torique. Afin de gérer les reprise après pannes, nous utiliserons un protocole de routage de type DV (Distance vector). Une entrée de la table de routage DV est constituée (au moins) : d'un vecteur (i.e. une direction à suivre) et d'une distance (généralement en nombre de sauts). Documentez-vous sur ce type d'algorithmes de routage.

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

```
#Tell the simulator to use dynamic routing
$ns rtproto DV

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

#Define a 'finish' procedure
proc finish {} {
      global ns nf
     $ns flush-trace
    #Close the trace file
     close $nf
    #Execute nam on the trace file
     exec nam out.nam &
     exit 0
}

#RING TOPOLOGY (Topologie Anneau)
# IMPORTANT : 1- Remarquez que les noeuds sont définis sur un tableau
#             2- ESSAYEZ DE REFAIRE LA MEME BOUCLE AVEC UN WHILE
#Create eight nodes
for {set i 0} {$i < 8} {incr i} {
      set n($i) [$ns node]
}
#Create links between the nodes
for {set i 0} {$i < 8} {incr i} {
     $ns duplex-link $n($i) $n([expr ($i+1)%8]) 1Mb 10ms DropTail
}


#Create a UDP agent and attach it to node n(0)
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Create a Null agent (a traffic sink) and attach it to node n(3)
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0


#Schedule events for the CBR agent and the network dynamics
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

1. A quoi correspondent les petits paquêts qu'on peut voir sur nam. A quels moments sont-ils

échangés ? Pourquoi ?

2. Etude d'une propriété de k-connexité : Définir une Topologie en Grille de 9 noeuds (de n1 à n9).
   (a) Combien y a-il de routes différentes entre n1 et n9 ? Quel est le chemin le plus court ?
   (b) Même question pour n3 et n8.
   (c) Lancez un flux constant entre n3 et n8.
   (d) Donnez Le pire scenario que peut rencontrer l'algorithme DV pour la route n3 - n8
3. Refaire la question précédente pour une grille torique (exemple :
   http ://upload.wikimedia.org/wikipedia/commons/d/db/Toroidal_coord.png
   tous les sommets du graphe auront un degré égale à 4)

4 Échanges de messages

Dans cette section, nous aborderons les échanges de messages PERSONNALISES via le protocole UDP. Il est très recommandé de vous documenter sur les expressions régulières en TCL.

```
#
#  This is a simple demonstration of how to send data in UDP datagrams
#

set ns [new Simulator]

$ns color 0 blue
$ns color 1 red


# open trace files
set f [open out.tr w]
$ns trace-all $f
set nf [open out.nam w]
$ns namtrace-all $nf


# create topology.  three nodes in line: (0)--(2)--(1)
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$ns duplex-link $n0 $n2 2Mb 5ms DropTail
$ns duplex-link $n2 $n1 1.5Mb 10ms DropTail

$ns duplex-link-op $n0 $n2 orient right
$ns duplex-link-op $n2 $n1 orient right


# create UDP agents
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

$ns connect $udp0 $udp1

# The "process_data" instance procedure is what will process received data
# if no application is attached to the agent.
# In this case, we respond to messages of the form "ping(###)".
# We also print received messages to the trace file.
```

```
Agent/UDP instproc process_data {size data} {
global ns
$self instvar node_

# note in the trace file that the packet was received
$ns trace-annotate "[$node_ node-addr] received {$data}"

# if the message was of the form "ping(###)" then send a response of
# the form "pong(###)"
if {[regexp {ping *\(([0-9]+)\)} $data entirematch number]} {
$self send 100 "pong($number)"
} elseif {[regexp {countdown *\(([0-9]+)\)} $data entirematch number]
  && $number > 0 } {
incr number -1
$self send 100 "countdown($number)"
}
}


# Setting the class allows us to color the packets in nam.
$udp0 set class_ 0
$udp1 set class_ 1


$ns at 0.1 "$udp0 send 724 ping(42)"
$ns at 0.2 "$udp1 send 100 countdown(5)"
$ns at 0.3 "$udp0 send 500 {ignore this message please}"
$ns at 0.4 "$udp1 send 828 {ping  (12345678)}"

$ns at 1.0 "finish"

proc finish {} {
global ns f nf
$ns flush-trace
close $f
close $nf

puts "running nam..."
exec nam out.nam &
exit 0
}

$ns run
```

Donnez une description détaillée de tout ce qui se passe dans cette cette simulation.

5 Réseaux sans fil

Considérons le script TCL suivant :

```
# A 3-node example for ad-hoc simulation with AODV

# Define options
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)           Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)          Phy/WirelessPhy            ;# network interface type
set val(mac)            Mac/802_11                 ;# MAC type
```

```
set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)             LL                         ;# link layer type
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                         ;# max packet in ifq
set val(nn)             3                          ;# number of mobilenodes
set val(rp)             AODV                       ;# routing protocol
set val(x)              500       ;# X dimension of topography
set val(y)              400       ;# Y dimension of topography
set val(stop) 150    ;# time of simulation end

set ns   [new Simulator]
set tracefd       [open simple.tr w]
set windowVsTime2 [open win.tr w]
set namtrace      [open simwrls.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo       [new Topography]

$topo load_flatgrid $val(x) $val(y)

create-god $val(nn)

#
#  Create nn mobilenodes [$val(nn)] and attach them to the channel.
#

# configure the nodes

$ns node-config -adhocRouting $val(rp) \
 -llType $val(ll) \
 -macType $val(mac) \
 -ifqType $val(ifq) \
 -ifqLen $val(ifqlen) \
 -antType $val(ant) \
 -propType $val(prop) \
 -phyType $val(netif) \
 -channelType $val(chan) \
 -topoInstance $topo \
 -agentTrace ON \
 -routerTrace ON \
 -macTrace OFF \
 -movementTrace ON

for {set i 0} {$i < $val(nn) } { incr i } {
set node_($i) [$ns node]
}

# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 150.0
$node_(2) set Y_ 240.0
```

```
$node_(2) set Z_ 0.0

# Generation of movements
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 110.0 "$node_(0) setdest 480.0 300.0 5.0"

# Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(1) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"

# Printing the window size
proc plotWindow {tcpSource file} {
global ns
set time 0.01
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 10.1 "plotWindow $tcp $windowVsTime2"

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150.01 "puts \"end simulation\" ; $ns halt"
proc stop {} {
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
}

$ns run
```

Dans cet exemple, la procédure de finalisation s'appelle *stop*. Elle ne fait pas appel à nam. Faites le manuellement à partir de votre console avec la commande : nam fichier.nam. Référez vous au script pour avoir le nom du fichier nam en question.

Dans le langage que vous préférez (le C/C++ par exemple), concevez un générateur de mouvements aléatoires pour ce type de réseaux. Appelez votre générateur setdest.exe. Il prendra en argument le nombre de nœuds, les dimensions de l'arène, les bornes de vitesse et temps de simulation en seconde. Il générera une séquence de commandes (dans un fichier) de la forme :

```
$ns at T "$node_(i) setdest destX DestY V"
```

où T est en secondes, i est un numéro de nœud, V est la vitesse et destX et destY sont les coordonnées de la destination (à l'intérieur de l'arène).