Simuler RED

(Random Early Discard)

Tania Jiménez[1] **et Eitan Altman**[2],

[1] LIA, Université d'Avignon, tania@lia.fr
[2]Projet MAESTRO, INRIA, Sophia Antipolis, France altman@sophia.inria.fr

Jan-Fev 2006

1 Rappel

•Conçu en 1993 par Floyd et Jacobson, voir RFC 2309.

•Objectifs de RED:

1. **Délai :** Permettre aux rafales courtes de passer sans que la taille de la file d'attente augmente beaucoup. On souhaite détecter et réagir sur de longues périodes de congestion.
2. **Equité :** Les routeurs ayant des files d'attente standard "drop tail" et "random drop" ont un biais contre de trafic en rafale.
3. **Synchronization :** Empêcher la synchronisation, qui apparaît dans des buffers de type "drop tail", car quand cette file est pleine, plusieurs connexions perdent des paquets en même temps. La synchronisation créée des oscillations.
4. Contrôler la taille moyenne de la file d'attente. Cela permet de réduire le délai moyen.

2 Description de RED

•RED calcul une valeur moyenne avg de la taille de la file d'attente et vérifie que cette moyenne se trouve entre un seuil min_{th} et un seuil maximal max_{th}.

•Si c'est le cas alors un paquet qui arrive est rejeté (ou marqué) avec une proba $p = p(avg)$. p est croissante.

•Tous les paquets arrivant quand $avg \geq max_{th}$ sont rejetés.

•Quand la file moyennée varie entre min_{th} et max_{th}, p_b varie linéairement entre 0 et une valeur max_p :

$$p_b(avg) = max_p \frac{avg - min_{th}}{max_{th} - min_{th}}.$$

•p_b est utilisée pour $p(avg)$ si à l'arrivée du paquet précédent, $avg \geq min_{th}$. Sinon, $p(avg)$ est donné par $p(avg)/(1 + p(avg))$.

•Pour calculer l'évolution de la file moyennée, on initialise le paramètre avg à zéro. Avec chaque paquet qui arrive, avg est modifié selon

$$(1 - w_q)avg + w_q q$$

où q est la valeur instantannée de la file d'attente et w_q est une constante petite.

•Si la file se vide, on utilise une autre formule pour mettre à jour la valeur moyenne. Elle prend en compte

1. le temps écoulé depuis que la file s'est vidée, et
2. une estimation sur le nombre des paquets qui auraient pu être transmis durant ce temps.
3. Pour calculer ce dernier, le simulateur a besoin d'une estimation de la taille moyenne d'un paquet.

•Exemples de paramètres de RED utilisés par Floyd et V. Jacobson:

- $w_q = 0.002$,
- $min_{th} = 5$paquets,
- $max_{th} = 15$paquets,
- $max_p = 1/50$ et
- la taille de la file d'attente est 100.
- On explore les valeurs de min_{th} allant de 3 à 50, tout en gardant $max_{th} = 3min_{th}$.

•L'implémentation de RED en ns se trouve dans
ns-allinone-2.XXX/ns-2.XXX/queue/red.cc (XXX est la version, e.g. 1b9a).

3 Initialisation de paramètres de RED dans ns

Les paramètres de RED en ns sont donnés dans les objets suivants:

1. `bytes_`: prend comme valeur "true" si on travail dans le "byte mode" ou "false" si on travail dans le mode paquet (la valeur par défaut). La taille d'un paquet qui arrive détermine la probabilité d'être marqué (ou perdu).
2. `queue-in-bytes_`: indique que la taille de la file moyennée sera mesurée en bytes si cette variable a la valeur "true". Dans ce cas, `thresh_` et `maxthres_` sont affectés par le paramètre de l'estimation la taille moyenne `mean_pktsize_` d'un paquet. Ce paramètre a par défaut la valeur "false".
3. `thres_`: est la valeur du seuil minimal min_{th}.
4. `maxthres_`: est la valeur du seuil maximal max_{th}.
5. `mean_pktsize_`: est l'estimation de la taille moyenne d'un paquet en bytes. Sa valeur par défaut est 500.

6. `q_weight_`: le poids W_q utilisé dans le calcul de la taille moyenne de la file d'attente.
7. `wait_`: C'est un paramètre qui permet de maintenir un intervalle entre des paquets rejetés quand il est donné la valeur "true" (qui est la valeur par défaut).
8. `linterm_`: C'est la valeur réciproque de max_p, qui vaut par défaut 10.
9. `setbit_`: est "false" dans le cas où RED est utilisé pour rejeter des paquets et vaut "true" si RED marque les paquets dans un bit indiquant la congestion. (La version ECN de TCP réagit à ces bits de congestion).
10. `drop-tail_`: C'est un paramètre qui permet, quand il a la valeur "true" (qui est aussi la valeur par défaut), d'utiliser le mécanisme "drop-tail" quand le buffer déborde. ou quand la taille moyenne de la file d'attente dépasse max_{th}.

4 Valeurs par défaut et autres variantes de RED

•La valeur par défaut de `q_weight_`, `maxthresh_` et `thres_` ont été 0.002, 15 et 5 resp. jusqu'à fin 2001.

•Dans les versions plus récentes, ces paramètres sont configurés automatiquement.

•RED a d'autres paramètres et variantes implementés en ns.

•S. Floyd conseille dans http://www.icir.org/floyd/red/gentle.html d'utiliser la version "gentle Red" pour une performance optimale.

•Pour l'utiliser, on doit mettre à "true" le paramètre `gentle_` (c'est aussi la valeur prise par défaut).

5 Gentle RED et RED Adaptatif

•Dans la modification `gentle_` de RED en ns, la probabilité de perte d'un paquet varie de max_p à 1 quand la taille moyenne de la file d'attente varie de `maxthresh_` à deux fois `maxthresh_`.

•Cette option rend RED bien plus robuste au choix de paramètres `maxthresh` et `max_p`.

•Une autre variante de RED s'appelle "adaptive RED" (ou ARED). Ici, RED s'adapte les choix de paramètres au trafic.

6 Mesurer la taille du buffer

Afin de mesurer la file d'attente de RED, par exemple entre les nœuds `$n2` et `$n3`, on peut faire :

```
set redq [[$ns link $n2 $n3] queue]
set traceq [open red-queue.tr w]
$redq trace curq_
$redq trace ave_
$redq attach $traceq
```

Ici, `curq_` est la valeur courante de la file d'attente et `ave_` donne la valeur moyennée. Ça nous permet d'avoir un fichier en sortie (dans notre cas, le fichier "red-queue.tr") avec trois colonnes.

•La première indique s'il s'agit de la valeur instantanée (indiquée par le symbole "Q") ou la valeur moyennée (indiquée par le symbole a).

•La deuxième indique le temps courant et

•La troisième – la valeur mesurée.

7 Exemples de Simulation

Nous considérons le réseau suivant :

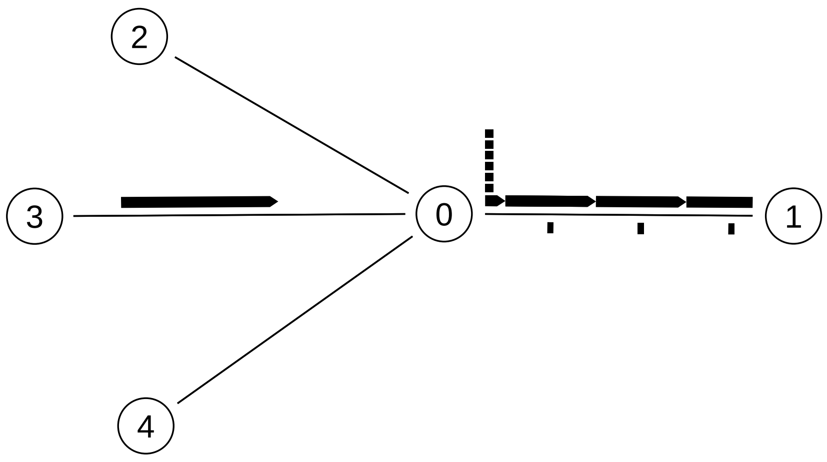

Figure 1: Le réseau utilisé pour l'étude de RED

On compare plusieurs mécanismes de gestion des files d'attente.

7.1 Le buffer "Drop tail"

•Le "drop tail" est le schéma le plus simple pour gérer des pertes.

•Nous considérons trois liens d'entrée avec des délais de 1 msec et une bande passante de 10 Mbps chacun,

•Un goulot d'étranglement commun a un délai de 20 msec et une bande passante de 700 kbps.

•Nous considérons trois connexions de FTP utilisant TCP ayant une fenêtre max de 8000.

•La taille du buffer au goulot d'étranglement est de 100.

•Les trois connexions commencent en des instants aléatoires entre 0 et 7 sec.

•Leurs délais jusqu'au goulot d'étranglement est de 1 msec.

•Nous choisissons la taille des paquets TCP à 552 bytes.

•A noter : dans la version 2.1b9a de ns, quand nous utilisons la commande

```
$tcp_src($j) set packetSize_ 552
```

alors la taille de paquet créé dans la simulation est de 592, car une entête supplémentaire de 40 bytes y est ajoutée.

•La durée de la simulation est de 50 sec.

•Utilisant l'option monitor-queue (déjà traitée dans le cours sur TCP), nous créons un fichier appelé queue.tr dont la première colonne correspond au temps, et la cinquième colonne est la taille instantanée de la file d'attente en paquets.

•Nous utiliserons aussi une procédure appelée plotWindow, pour suivre les tailles des fenêtres. On obtient alors un fichier où la première colonne est le temps, et les trois colonnes restantes présentent la taille de fenêtre de chacune des trois connexions.

```
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf

set tf [open out.tr w]
set windowVsTime [open win w]
set param [open parameters w]
$ns trace-all $tf

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

#Create bottleneck and dest nodes
set n2 [$ns node]
set n3 [$ns node]

#Create links between these nodes
$ns duplex-link $n2 $n3 0.7Mb 20ms DropTail

set NumbSrc 3
set Duration 50

#Source nodes
for {set j 1} {$j<=$NumbSrc} { incr j } {
set S($j) [$ns node]
}

# Create a random generator for starting the ftp and for bottleneck link delays
set rng [new RNG]
$rng seed 2

# parameters for random variables for begenning of ftp connections
set RVstart [new RandomVariable/Uniform]
$RVstart set min_ 0
$RVstart set max_ 7
$RVstart use-rng $rng

#We define random starting times for each connection
for {set i 1} {$i<=$NumbSrc} { incr i } {
set startT($i)  [expr [$RVstart value]]
set dly($i) 1
puts $param "startT($i)  $startT($i) sec"
}
```

```
#Links between source and bottleneck
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns duplex-link $S($j) $n2 10Mb $dly($j)ms DropTail
$ns queue-limit $S($j) $n2 20
}

#Set Queue Size of link (n2-n3) to 100
$ns queue-limit $n2 $n3 100

#TCP Sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
set tcp_src($j) [new Agent/TCP/Reno]
$tcp_src($j) set window_ 8000
}

#TCP Destinations
for {set j 1} {$j<=$NumbSrc} { incr j } {
set tcp_snk($j) [new Agent/TCPSink]
}
```

```
#Connections
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns attach-agent $S($j) $tcp_src($j)
$ns attach-agent $n3 $tcp_snk($j)
$ns connect $tcp_src($j) $tcp_snk($j)
}

#FTP sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
set ftp($j) [$tcp_src($j) attach-source FTP]
}

#Parametrisation of TCP sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
$tcp_src($j) set packetSize_ 552
}
```

```
#Schedule events for the FTP agents:
for {set i 1} {$i<=$NumbSrc} { incr i } {
$ns at $startT($i) "$ftp($i) start"
$ns at $Duration "$ftp($i) stop"
}

proc plotWindow {tcpSource file k} {
global ns NumbSrc
set time 0.03
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
if {$k == 1} {
   puts -nonewline $file "$now \t $cwnd \t"
  } else {
   if {$k < $NumbSrc } {
   puts -nonewline $file "$cwnd \t" }
}
```

```
if { $k == $NumbSrc } {
   puts -nonewline $file "$cwnd \n" }
$ns at [expr $now+$time] "plotWindow $tcpSource $file $k" }

# The procedure will now be called for all tcp sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns at 0.1 "plotWindow $tcp_src($j) $windowVsTime $j"
}

set qfile [$ns monitor-queue $n2 $n3  [open queue.tr w] 0.05]
[$ns link $n2 $n3] queue-sample-timeout;

$ns at [expr $Duration] "finish"
$ns run
```

Table 1: tcl script drprail.tcl

•La taille de la file (Fig. 2) et de la fenêtre (Fig. 3) :

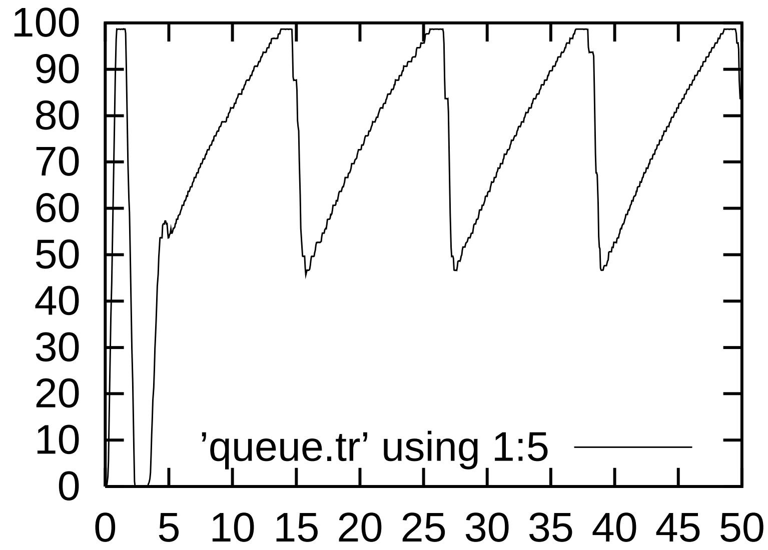

Figure 2: La taille de la file

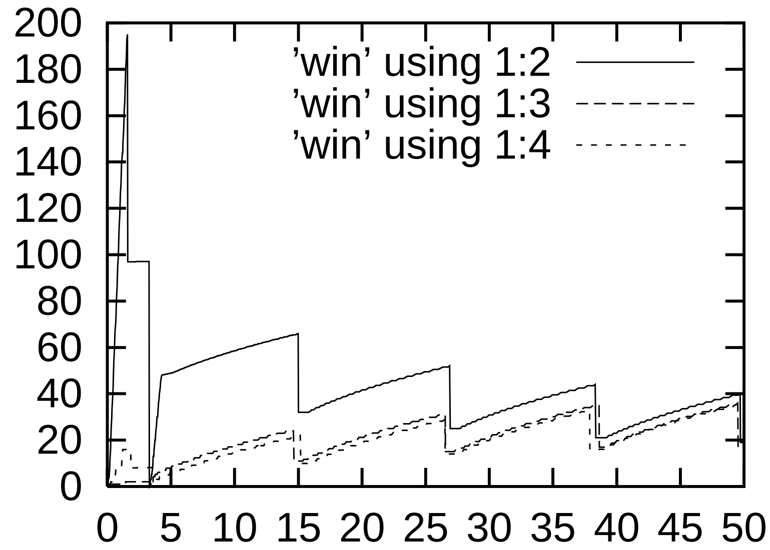

Figure 3: Taille de fenêtre des connexions TCP

•Les figures montrent bien un grand degré de synchronisation entre les fenêtres : les trois connexions perdent des paquets en même temps.

•On observe des grandes oscillations de la taille de la file d'attente correspondant aux variations de la taille des fenêtres.

•La taille moyenne de la file est environ 75 paquets.

•On a donc en moyenne un délai supplémentaire dans la file d'attente de

$$D_q = \frac{75 \times 592 \times 8}{700 \times 10^3} = 507.42msec$$

Remarque:

La gestion de "drop-tail" peut être simulée en

•utilisant une file RED, en choisissant la valeur $min_{th} = max_{th}$ égale à la valeur maximale de la file d'attente,

•et en choisissant la valeur de max_p proche de zéro.

•Cela nous permet d'utiliser des outils de mesure de la valeur instantanée et de la valeur moyenne qui existent dans RED, toute en utilisant le régime "drop tail".

•Bien sur, le paramètre `drop-tail_` doit être initialisé à "true".

7.2 Red avec une configuration automatique des paramètres

•Nous re-simulons avec les mêmes paramètres, seul le buffer change.

•Nous utilisons pour le délai aléatoire le noyau 2 dans toutes les simulations,

•Contrairement au noyau 0, les variables aléatoires générées dans toutes les simulations seront les mêmes.

•Durant 50 sec de temps simulé, la destination a reçu 7310 paquets TCP,

•C'est un peu plus que dans le cas "drop tail" qui a donné 7211 paquets.

•L'évolution de la file d'attente (Fig. 4 et 5) et de la taille de fenêtre (Fig.6).

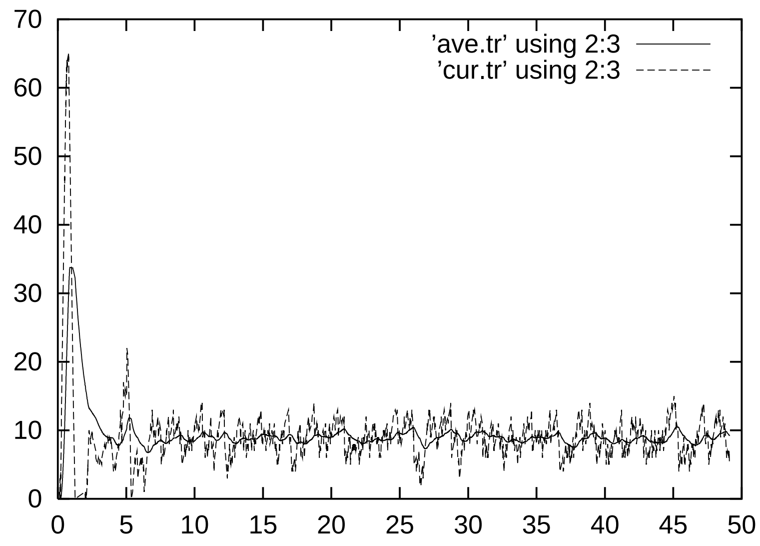

Figure 4: Valeur instantanée et moyenne de la file d'attente

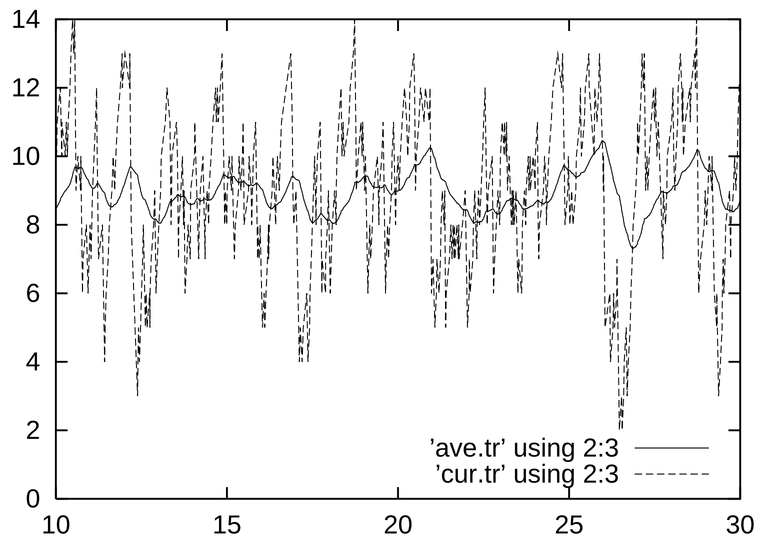

Figure 5: Valeur instantanée et moyenne de la file d'attente: zoom

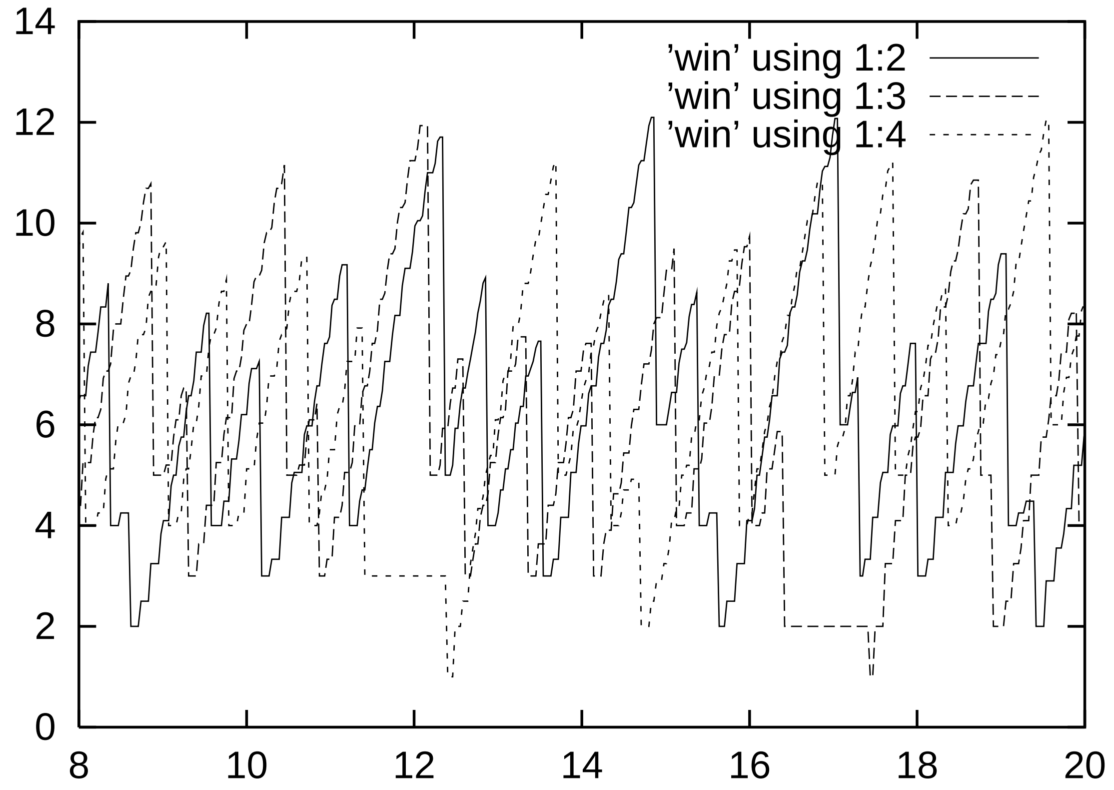

Figure 6: Evolution des fenêtres des connexions TCP pour RED avec configuration automatique

•Conclusion 1 : pas de synchronisation.

•Conclusion 2 : taille moyenne de la file d'attente env. 10 (pour drop-tail c'était 75).

•La valeur moyenne du délai d'une connexion est petite :

$$D_q = \frac{10 \times 592 \times 8}{700 \times 10^3} = 67.66msec.$$

•Les grandes oscillations de la file d'attente et des fenêtres sont remplacées par des variations petites et plus rapides de la taille de la file et des fenêtres.

•Durant la simu, pas de débordement de la file, contrairement au "drop-tail". RED a quand même permis à la file de croître beaucoup durant des rafales transiantes (voir début de la connexion).

Le Tableau 2 contient le code tcl utilisé.

```
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf

set tf [open out.tr w]
set windowVsTime [open win w]
set param [open parameters w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        close $nf
        close $tf
        exec nam out.nam &
```

```
exec grep  "a" red-queue.tr > ave.tr
exec grep  "Q" red-queue.tr > cur.tr
        exit 0
}

#Create bottleneck and dest nodes
set n2 [$ns node]
set n3 [$ns node]

#Create links between these nodes
$ns duplex-link $n2 $n3 0.7Mb 20ms RED

set NumbSrc 3
set Duration 50

#Source nodes
for {set j 1} {$j<=$NumbSrc} { incr j } {
set S($j) [$ns node]
}
```

```
# Create a random generator for starting the ftp
# and for bottleneck link delays
set rng [new RNG]
$rng seed 2

# parameters for random variables for begenning of ftp connections
set RVstart [new RandomVariable/Uniform]
$RVstart set min_ 0
$RVstart set max_ 7
$RVstart use-rng $rng
```

```
#We define random starting times for each connection
for {set i 1} {$i<=$NumbSrc} { incr i } {
set startT($i)  [expr [$RVstart value]]
set dly($i) 1
puts $param "startT($i)  $startT($i) sec"
}

#Links between source and bottleneck
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns duplex-link $S($j) $n2 10Mb $dly($j)ms DropTail
$ns queue-limit $S($j) $n2 20
}
```

```
#Set Queue Size of link (n2-n3) to 100
$ns queue-limit $n2 $n3 100

set redq [[$ns link $n2 $n3] queue]
set traceq [open red-queue.tr w]
$redq trace curq_
$redq trace ave_
$redq attach $traceq

#TCP Sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
set tcp_src($j) [new Agent/TCP/Reno]
$tcp_src($j) set window_ 8000
}
```

```
#TCP Destinations
for {set j 1} {$j<=$NumbSrc} { incr j } {
set tcp_snk($j) [new Agent/TCPSink]
}

#Connections
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns attach-agent $S($j) $tcp_src($j)
$ns attach-agent $n3 $tcp_snk($j)
$ns connect $tcp_src($j) $tcp_snk($j)
}

#FTP sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
set ftp($j) [$tcp_src($j) attach-source FTP]
}
```

```
#Parametrisation of TCP sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
$tcp_src($j) set packetSize_ 552
}

#Schedule events for the FTP agents:
for {set i 1} {$i<=$NumbSrc} { incr i } {
$ns at $startT($i) "$ftp($i) start"
$ns at $Duration "$ftp($i) stop"
}

proc plotWindow {tcpSource file k} {
global ns NumbSrc
set time 0.03
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
```

```
if {$k == 1} {
   puts -nonewline $file "$now \t $cwnd \t"
  } else {
   if {$k < $NumbSrc } {
   puts -nonewline $file "$cwnd \t" }
}
if { $k == $NumbSrc } {
   puts -nonewline $file "$cwnd \n" }
$ns at [expr $now+$time] "plotWindow $tcpSource $file $k" }

# The procedure will now be called for all tcp sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns at 0.1 "plotWindow $tcp_src($j) $windowVsTime $j"
}

$ns at [expr $Duration] "finish"
$ns run
```

Table 2: tcl script red.tcl

7.3 RED avec d'autres paramètres

•Supposons qu'on souhaite utiliser des paramètres autres que ceux par défaut.

•Par exemple, supposons qu'on souhaite avoir dans l'exemple précédent `max_{th}`=60, `min_{th}`=40 et `q_weight_=0.02`.

•Il faut alors ajouter les commandes

```
Queue/RED set thresh_ 60
Queue/RED set maxthresh_ 80
Queue/RED set q_weight_ 0.002
```

Important : note: Il faut mettre ces commandes au début, avant de définir les liens !

•On obtient alors Figures 8 et 7 :

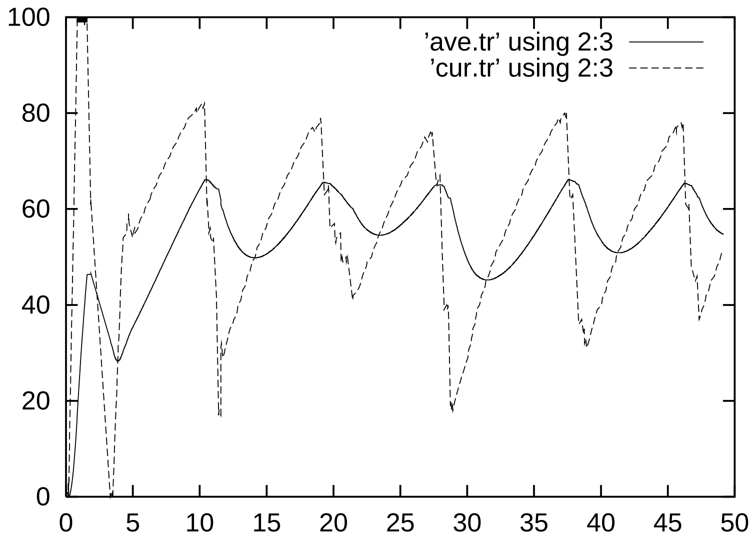

Figure 7: Valeur instantanée et moyenne de la file d'attente

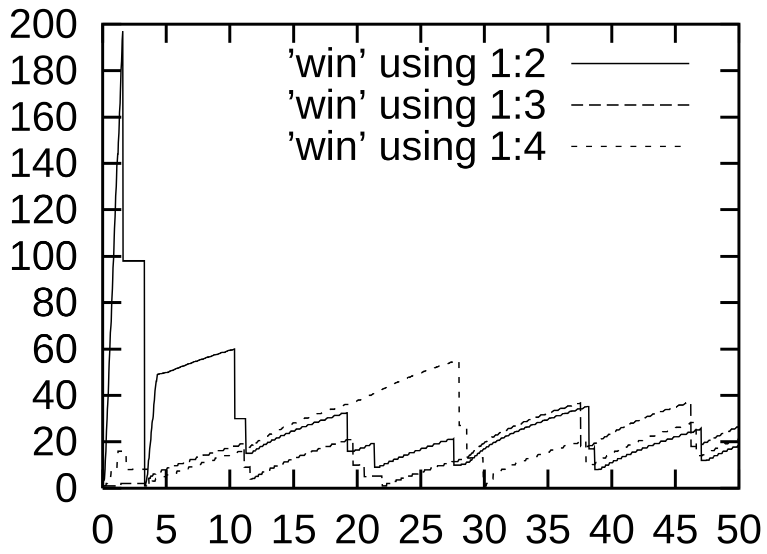

Figure 8: Evolution des fenêtres des connexions TCP pour RED

•Avec ce choix de paramètres, la file d'attente est autour de 50.

•On reçoit durant la simulation 7212 paquets.

8 Monitorer les flux

•On va voir comment monitorer les flux afin de mesurer des quantités par flux tel que les pertes et la quantité de trafic écoulé.

•Nous montrons comment modifier le code shortTcp2.tcl (Tableau **??**) pour avoir un buffer RED buffer avec ces mesures.

•On doit définir le lien à monitorer :

```
Le lien flink [$ns simplex-link $N $D 2Mb 1ms RED]
```

Puis on définit le moniteur de flux par rapport à ce lien :

```
set monfile [open mon.tr w]
set fmon [$ns makeflowmon Fid]
$ns attach-fmon $flink $fmon
$fmon attach $monfile
```

•Quand on active les monitoring, nous obtenons des statistiques souhaitées jusqu'au temps souhaité dans un fichier :

```
$ns at $time "$fmon dump"
```

•Dans le Tableau 3 nous proposons le code entier shortRed.tcl qui nous permet d'étudier l'interaction de TCP avec RED dans le cas de sessions **non-persistantes**.

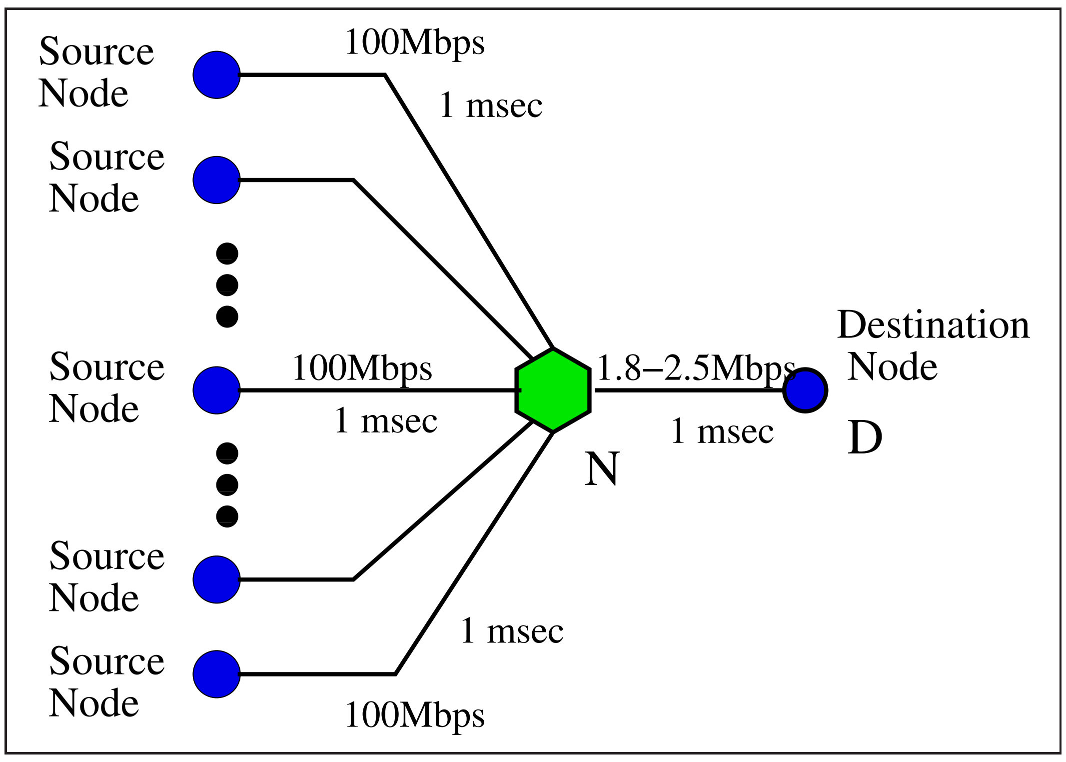

Figure 9: Network topology

```
set ns [new Simulator]

# There are several sources each generating many TCP sessions sharing
# a bottleneck link and a single destination. Their number
# is given by the paramter NodeNb

#      S(1)           ----
#        .                |
#        .            ----N --------------- D
#        .                |
#     S(NodeNb)       ----

set Out [open Out.ns w];   # file containing transfer times
set Conn [open Conn.tr w]; # file containing the number of connections
set tf [open out.tr w];    #Open the Trace file
$ns trace-all $tf

set NodeNb      6;   # Number od source nodes
```

```
set NumberFlows 253; # Number of flows per source node
set sduration    50;  # Duration of simulation

# When the following parameters are commented, the RED is
# configured automatically.
# Queue/RED set thresh_ 5
# Queue/RED set maxthresh_ 15
# Queue/RED set q_weight_ 0.002

# defining the topology
set N [$ns node]
set D [$ns node]
set flink [$ns simplex-link $N $D 2Mb 1ms RED]
$ns simplex-link $D $N 1Mb 1ms DropTail
$ns queue-limit $N $D 50
```

```
# queue monitoring, RED
set redq [[$ns link $N $D] queue]
set traceq [open red-queue.tr w]
$redq trace curq_
$redq trace ave_
$redq attach $traceq

#Nodes and links
for {set j 1} {$j<=$NodeNb} { incr j } {
set S($j) [$ns node]
$ns duplex-link $S($j) $N 100Mb 1ms DropTail
$ns queue-limit $S($j) $N 100
}
```

```
# set flow monitor
set monfile [open mon.tr w]
set fmon [$ns makeflowmon Fid]
$ns attach-fmon $flink $fmon
$fmon attach $monfile

#TCP Sources, destinations, connections
for {set i 1} {$i<=$NodeNb} { incr i } {
for {set j 1} {$j<=$NumberFlows} { incr j } {
set tcpsrc($i,$j) [new Agent/TCP/Newreno]
set tcp_snk($i,$j) [new Agent/TCPSink]
set k [expr $i*1000 +$j];
$tcpsrc($i,$j) set fid_ $k
$tcpsrc($i,$j) set window_ 2000
$ns attach-agent $S($i) $tcpsrc($i,$j)
$ns attach-agent $D $tcp_snk($i,$j)
$ns connect $tcpsrc($i,$j) $tcp_snk($i,$j)
set ftp($i,$j) [$tcpsrc($i,$j) attach-source FTP]
} }
```

```
# Generators for random size of files.
set rng1 [new RNG]
$rng1 seed 0
set rng2 [new RNG]
$rng2 seed 0

# Random inter-arrival times of TCP transfer at each source i
set RV [new RandomVariable/Exponential]
$RV set avg_ 0.3
$RV use-rng $rng1

# Random size of files to transmit
set RVSize [new RandomVariable/Pareto]
$RVSize set avg_ 10000
$RVSize set shape_ 1.5
$RVSize use-rng $rng2

# We now define the beginning times of transfers and the transfer sizes
# Arrivals of sessions follow a Poisson process.
```

```
for {set i 1} {$i<=$NodeNb} { incr i } {
     set t [$ns now]
     for {set j 1} {$j<=$NumberFlows} { incr j } {
 # set the beginning time of next transfer from source and attributes
 set t [expr $t + [$RV value]]
 $tcpsrc($i,$j) set starts $t
 $tcpsrc($i,$j) set sess $j
 $tcpsrc($i,$j) set node $i
         $tcpsrc($i,$j) set size [expr [$RVSize value]]
  $ns at [$tcpsrc($i,$j) set starts] "$ftp($i,$j) send [$tcpsrc($i,$j) set size]"
  # update the number of flows
   $ns at [$tcpsrc($i,$j) set starts] "countFlows $i 1"
}}
```

```
for {set j 1} {$j<=$NodeNb} { incr j } {
set Cnts($j) 0
}

# The following procedure is called whenever a connection ends
Agent/TCP instproc done {} {
global tcpsrc NodeNb NumberFlows ns RV ftp Out tcp_snk RVSize
# print in $Out: node, session, start time,  end time, duration,
# trans-pkts, transm-bytes, retrans-bytes, throughput
  set duration [expr [$ns now] - [$self set starts] ]
  puts $Out "[$self set node] \t [$self set sess] \t [$self set starts] \t\
      [$ns now] \t $duration \t [$self set ndatapack_] \t\
      [$self set ndatabytes_] \t [$self set  nrexmitbytes_] \t\
      [expr [$self set ndatabytes_]/$duration ]"
  countFlows [$self set node] 0
}
```

```
# The following recursive procedure updates the no. of connections
# as a function of time. Each 0.2 sec it prints them into $Conn. This
# is done by calling the procedure with the "sign" parameter equal
# 3 (in which case the "ind" parameter does not play a role). The
# procedure is also called by the "done" procedure whenever a
# connection from source i ends by assigning the "sign" parameter 0,
# or when it begins, by assigning 1 (i is passed through the "ind"
# variable).
proc countFlows { ind sign } {
global Cnts Conn NodeNb
set ns [Simulator instance]
      if { $sign==0 } { set Cnts($ind) [expr $Cnts($ind) - 1]
} elseif { $sign==1 } { set Cnts($ind) [expr $Cnts($ind) + 1]
} else {
  puts -nonewline $Conn "[$ns now] \t"
  set sum 0
  for {set j 1} {$j<=$NodeNb} { incr j } {
    puts -nonewline $Conn "$Cnts($j) \t"
    set sum [expr $sum + $Cnts($j)]
  }
```

```
  puts $Conn "$sum"
  $ns at [expr [$ns now] + 0.2] "countFlows 1 3"
} }

proc finish {} {
        global ns tf
        $ns flush-trace
close $tf
        exec grep  "a" red-queue.tr > ave.tr
        exec grep  "Q" red-queue.tr > cur.tr
        exit 0
}
$ns at 0.5 "countFlows 1 3"
$ns at [expr $sduration - 0.01] "$fmon dump"
$ns at $sduration "finish"
$ns run
```

Table 3: tcl script shortRed.tcl

•Le monitoring nous fournit un fichier avec des informations supplémentaires sur le type de perte de paquets.

•Il nous permet de distinguer entre des "Early drops" (ED) causées par RED et les pertes de débordement.

•Le format du fichier :

1. Colonne 1 : le temps auquel le "dump" a été effectué
2. Colonnes 2 et 5 : les deux donnent l'id du flux
3. Colonne 3 : null (zeros).
4. Colonne 4 : type de flux.
5. Colonnes 6 et 7 : source et destination du flux.
6. Colonnes 8 et 9 : le nombre total d'arrivées du flux en paquets et en bytes, resp.

7. Colonnes 10 et 11 : quantité de ED (early drops) du flux en packets et en bytes, respectivement.
8. Colonnes 12 et 13 : nombre total d'arrivées de tous les flots en paquets et en bytes, respectivement.
9. Colonnes 14 et 15 : nombre de ED de tous les flux en paquets et en bytes, respectivement.
10. Colonnes 16 et 17 : nombre de pertes de tous les flux en paquets et en bytes, respectivement.
11. Colonnes 18 et 19 : total nombre de pertes dans le flux spécifié en paquets et en bytes, respectivement.

•Pour utiliser le moniteur de flux, chaque connexion TCP que nous souhaitons monitorer doit avoir un numéro id.

•Dans notre cas, nous avons initialement identifié un flux par son numéro et sa source (par exemple, la troisième session qui commence au nœud 4).

•Puis nous transformons cela à un vecteur uni-dimentionnel :

```
set k [expr $i*1000 +$j];
$tcpsrc($i,$j) set fid_ $k
```

•La simu a permis d'obtenir les fichiers suivants :

1. cur.tr et ave.tr qui décrivent l'évolution de la taille instantanée et moyenne de la file d'attente,
2. Conn.tr permet de monitorer le nombre de connexions actives de chacune des six sources (six est donné dans le code tcl par le variable NumberFlows) ainsi que le nombre des sessions actives en fonction du temps,
3. Out.ns pour monitorer chacune des sessions identifiée par le nœud source et le numéro de session dont cette source est l'origine, l'instant du début et de la fin ainsi que la durée de la connexion, le nombre de paquets et de bytes transmis, le nombre de bytes retransmis, et le débit de la session.
4. mon.tr est la trace génerée par le moniteur de flux, contenant le nombre de paquets et de bytes perdus par connexion.
5. out.tr est la trace globale de tous les événements.

•Nous présentons la taille instantanée et moyenne de la file d'attente dans les figures 10-11.

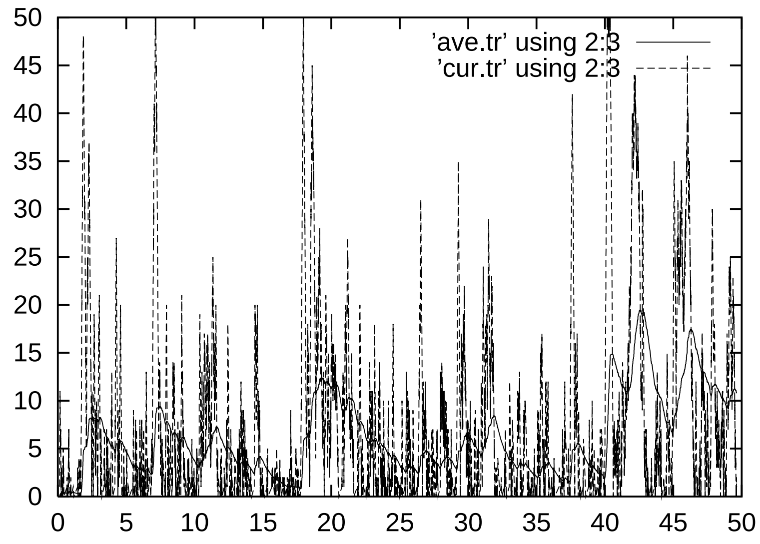

Figure 10: Evolution of the queue size and of its average

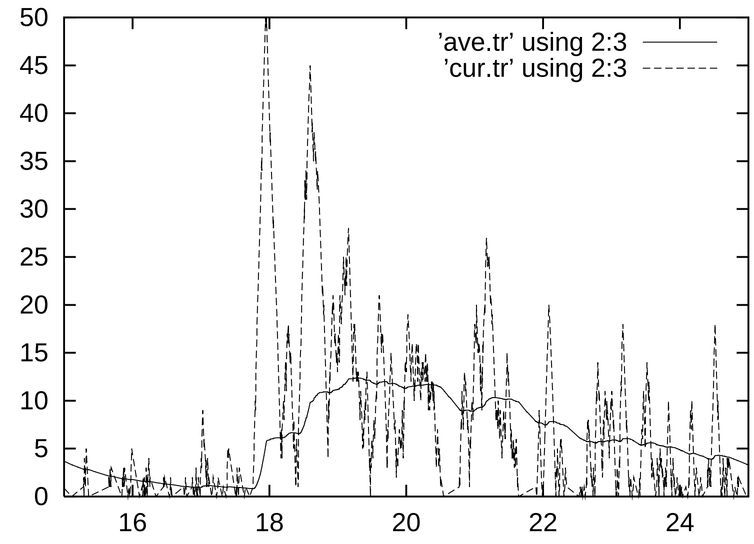

Figure 11: Queue size and averaged size evolution: a zoom

•On voit que la taille de la file d'attente est beaucoup plus variable que dans le cas des connexions persistantes (les 4 et 5).

•Le nombre de connexions actives est présenté dans la Fig. 12.

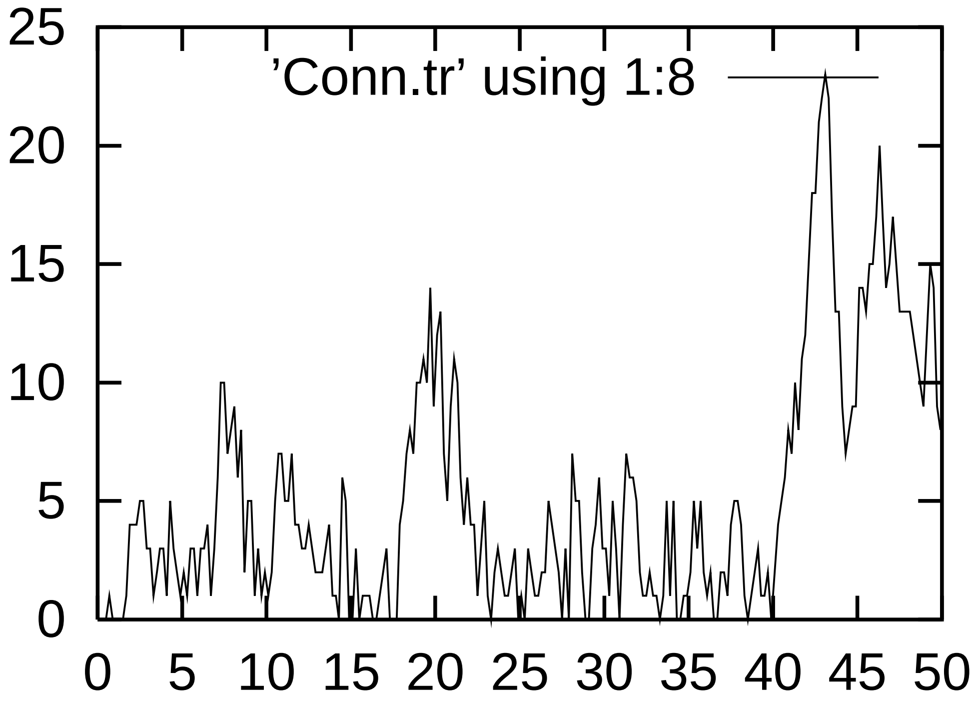

Figure 12: Number of active connections as a function of time

Nous avons integré dans le code plusieurs façons de monitorer.

•La manière directe de monitorer le nombre de retransmissions et d'arrivées des paquets se fait par la procédure "done". elle contient toute l'information (globale) concernant la connexion.

•Le moniteur de flux contient l'information locale de pertes dans un lien particulier. Cette méthode génère plus d'informations permettant de comprendre la contribution de chaque élément du réseau sur les pertes et les délais des paquets d'une session.