

EXPERIMENT 7

Aim :

Simulating the effect of Queuing Disciplines on Network Performance – Random Early Detection (RED) / Weighted RED / Adaptive RED (This can be used as a lead up to DiffServ / IntServ later).

Theory :

Random Early Detection (RED) :

Random Early Detection (RED), also known as Random Early Discard or Random Early Drop is a Queuing Discipline for a Network Scheduler suited for Congestion Avoidance.

In the conventional tail drop algorithm, a router or other network component buffers as many packets as it can, and simply drops the ones it cannot buffer. If buffers are constantly full, the network is congested. Tail drop distributes buffer space unfairly among traffic flows. Tail drop can also lead to TCP global synchronization as all TCP connections "hold back" simultaneously, and then step forward simultaneously. Networks become under-utilized and flooded - alternately, in waves.

RED addresses these issues by pre-emptively dropping packets before the buffer becomes completely full. It uses predictive models to decide which packets to drop. It was invented in the early 1990s by Sally Floyd and Van Jacobson.

Operation :

RED monitors the average queue size and drops packets based on statistical probabilities. If the buffer is almost empty, then all incoming packets are accepted. As the queue grows, the probability for dropping an incoming packet grows too. When the buffer is full, the probability has reached 1 and all incoming packets are dropped.

RED is more fair than tail drop, in the sense that it does not possess a bias against burst traffic that uses only a small portion of the bandwidth. The more a host transmits, the more likely it is that its packets are dropped as the probability of a host's packet being dropped is proportional to the amount of data it has in a queue. Early detection helps avoid TCP global synchronization.

Problems with Classic RED :

According to Van Jacobson, "there are not one, but two bugs in classic RED". Improvements to the algorithm were developed, and a draft paper was prepared, but the paper was never published, and the improvements were not widely disseminated or implemented. There has been some work in trying to finish off the research and fix the bugs. Pure RED does not accommodate Quality of Service (QoS) Differentiation. Weighted RED (WRED) and RED with In and Out (RIO) provide early detection with QoS considerations.

Weighted RED – In Weighted RED you can have different probabilities for different priorities (IP precedence, DSCP) and/or queues.

Adaptive RED – The Adaptive RED or Active RED (ARED) algorithm infers whether to make RED more or less aggressive based on the observation of the average queue length. If the average queue length oscillates around *min* threshold then early detection is too aggressive. On the other hand, if the average queue length oscillates around *max* threshold then early detection is being too conservative. The algorithm changes the probability according to how aggressively it senses it has been discarding traffic.

Code :

red.tcl file

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

#Open the nam trace file
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 beginning of ftp connections
set RVstart [new RandomVariable/Uniform]
$RVstart set min_ 0
$RVstart set max_ 7
$RVstart use-rng $rng

#We define random starting time 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 $k" }
#Procedure will now be called for all tcp sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns at 0.1 "plotWindow $tcp_src($j) $windowVsTime $j" }
$ns at [expr $Duration] "finish"
puts "running nam"
$ns run
```

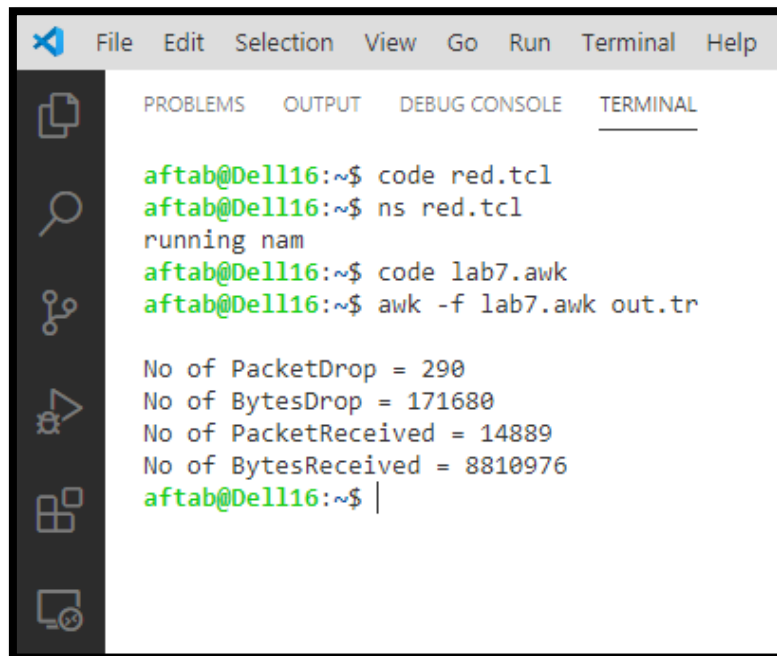
Code :

lab7.awk file

```
BEGIN {
pktDrop = 0;
byteDrop = 0;
pktRcd = 0;
byteRcd = 0;
}
{
if (($1 == "d") && ($5 == "tcp") || ($5 == "cbr"))
{
    pktDrop += 1;
    byteDrop += $6;
}
if (($1 == "r") && ($5 == "tcp") || ($5 == "cbr"))
{
    pktRcd += 1;
    byteRcd += $6;
}
}
END {
printf("\nNo of PacketDrop = %d \nNo of BytesDrop = %d \nNo of
PacketReceived = %d \nNo of BytesReceived = %d \n", pktDrop, byteDrop,
pktRcd, byteRcd);
}
```

Mohd. Aftab Alam
02013302717

Screen Shots :



```
aftab@Dell16:~$ code red.tcl
aftab@Dell16:~$ ns red.tcl
running nam
aftab@Dell16:~$ code lab7.awk
aftab@Dell16:~$ awk -f lab7.awk out.tr

No of PacketDrop = 290
No of BytesDrop = 171680
No of PacketReceived = 14889
No of BytesReceived = 8810976
aftab@Dell16:~$ |
```

