Simulating DifServ

(Differentiated Services)

Eitan Altman[1],

[1]Projet MAESTRO,
INRIA, Sophia Antipolis,
France altman@sophia.inria.fr

Jan-Fev 2006

1 Background

- Traditional Internet: best effort. All connections get the same treatment in the network.
- In contrast with other networking concepts, such as the ATM. that can offer quality of service requirements.
- Price: much higher signalling and processing related to
  - acceptance of new connections
  - maintaining the guarantees of ongoing connections.
- Yet, it is important to differentiate between connection classes and to allocate resources to connections according to their class.

- **Why diffserv?** A subscriber that is willing to pay more should get better service.
- Diffserv is based on marking packets at the **edge** of the network according to the performance level that the network wishes to provide them.
- According to the marks, packets are treated differently at the **network's nodes**.
- A common way to differentiate packets is by using RED buffers using different parameters for different packets.
- The ns module that handles diffserv has been developed in Nortel Networks.

2 Description of assured forwarding Diffserv

- The Diffserv implemented in ns follows the “Assured forwarding” approach standardized in

J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, “Assured forwarding PHB group”, RFC 2597, June 1999.

- A packet **belonging to a flow** may receive three possible priority levels within the flow.
- These are called ”drop precedences”. This can be used to provide a lower loss probability to sync packets in a TCP connection, since losses of sync packets result in very long time-outs.

- In addition to differentiation within each flow, all flows are classified to (≤ 4) classes (at most four), and different treatment can be given to the different classes.
- Moreover, it is possible to differentiate between flows.
- Four classes of flows are defined, and packets of a given class are queued in a class-dependent queue.
- In order to differentiate between packets belonging to the same class, three **virtual queues** are implemented in each of the four queues.
- To each of the 12 combinations of the four flow class and the three internal priority levels within a flow correspond a code point that a packet is given when entering the network.
- In practice not all queues and all priority groups need to be implemented.

3 Three components of Diffserv architecture

- **Policy and resource manager:**
  - It creates policies and distributes them to diffserv routers.
  - A policy determines which level of services in the network is assigned to which packets.
  - This assignment may depend on the behavior of the source of the flow, e.g. its **average rate** and its **burstiness**
  - Hence, special network elements are therefore added at the edge of the network so as to **measure** the source behavior.
  - In ns simulation, the policy is fully determined in the tcl script.

- **Edge routers:**
  - are responsible to assign the code points to the packets according to the policy specified by the network administrator.
  - To do so they **measure parameters of the input traffic** of each flow.
- **Core routers:**
  - the basic approach of diffserv is to **keep the intelligence in the edge of the network**;
  - routers within the network assign the appropriate priority to packets according to their code mark.
  - The priority translates to parameters of the scheduling and of the dropping decisions in the core routers.

4 MRED routers

● **General description.** There are three virtual RED buffers (called MRED - Multi RED) in each physical queue allows to enhance its behavior and to create dependence between their operation.

● **RIO-C:**

- One way to do that is through the RIO C (Rio Coupled) version of MRED.
- The probability of dropping low priority packets (called "out-of-profile packets") is based on the weighted average lengths of **all virtual queues**, whereas
- the probability of dropping a high priority ("in-profile") packet is based only on the **weighted average length of its own virtual queue**.

- **RIO-D**
In contrast, in RIO-D (RIO De-coupled), the probability of dropping each packet is based on the **size of its virtual queue**.

- Another version is the **WRED** (Weighted Red) in which all probabilities are based on a single queue length.

- **Ref:** D. D. Clark and W. Fang. Explicit Allocation of Best-Effort Packet Delivery Service. *IEEE/ACM Trans on Networking*, 6(4), 362-373, August 1998.

- It is possible to use also the dropTail queue.

5 Configuration of MRED in ns

- To determine the number of physical queues, we use the command

```
$dsredq set numQueues_ $m
```

where m can take values between 1 and 4.

- Configuring queue 0 to be a RIO-C is done with the command

```
$dsredq setMREDMode RIO-C 0
```

- If the last argument is not given then all queues are set to be RIO-C.
- Similarly, types other than RIO-C can be defined.

- To specify the number \$n of virtual queues, we use the command:

```
$dsredq setNumPrec $n
```

- Red parameters are then configured using the command

```
$dsredq configQ $queueNum $virtualQueueNum $minTh $maxTh $maxP
```

- It thus has 5 parameters:

1. the queue number,
2. virtual queue number,
3. min_{th},
4. max_{th} and
5. max_p.

- The parameter q_w can also be given (as the 6th parmaeter) and if it is not stated then it is taken to be 0.002 by default.
- The droptail queue can also be used with the command

```
$dsredq setMREDMode DROP
```

- The configuation then is given as before with only the first three parameters:

```
$dsredq configQ $queueNum $virtualQueueNum $minTh
```

All arriving packets are dropped when the min_{th} value is reached.

- As we saw in the chapter on RED, for computing the drop probability we need an estimate of the packet size.
- For a packet of size 1000 bytes this command is given by the command

```
$dsredq meanPktSize 1000
```

6 Scheduling

- Particular scheduling regimes can be defined. For example the weighted round robin with queue weights 5 and 1 respectively will be defined through

```
$dsredq setSchedularMode WRR
$dsredq addQueueWeights 1 5
```

- Other possible scheduling are Weighted Interleaved Round Robin (WIRR), Round Robin (RR) which is the default scheduling, and the strict priorities (PRI).

7 PHB table

- The set of four queues along with the virtual queues is supplemented with a PHB (Per Hop Behavior) table.
- Its entries are defined by (i) the code point (ii) the class (physical queue) and (iii) the "precedence" (virtual queue).
- An entry is assigned with the command of the form

```
$dsredq addPHBEntry 11 0  1
```

which means that code point 11 is mapped to the virtual queue 1 of the physical queue 0.

8 TCL querying

- The following three commands result in printing
  - the PHB table,
  - the number of physical and virtual queues and
  - the RED weighted average size of the specified physical queues (0 in our case):

```
$dsredq printPHBTable
$dsredq printStats
$dsredq getAverage 0
```

9 Defining policies

9.1 Description

- All flows having the same source and destination are subject to a common policy.
- A policy defines a policer type, a target rate, and other policy specific parameters.
- It specifies at least two code points.
- The choice between them depends on the comparison between the flow's target and its current sending rate, and possibly on the policy-dependent parmaeters (such as burstiness).
- The policy specifies meter types that are used for measuring the relevant input traffic parameters.
- A packet arriving at the edge device causes the meter to update the state variables corresponding to the flow, and the packet is then marked according to the policy.
- The packet has an initial code point corresponding to the required service level; the marking can result in downgrading the service level with respect to the initial required one.
- A policy table is used in ns to store the policy type of each flow. Not all entries are actually used.

- The entries are

1. Source node ID
2. Destination node ID
3. Policer type
4. Meter type
5. Initial code point
6. CIR (committed information rate)
7. CBS (committed burst size)
8. C bucket (current size of the committed bucket)
9. EBS (excess burst size)

10. E bucket (current size of the excess bucket)
11. PIR (peak information rate)
12. PBS (peak burst size)
13. P bucket (current size of the peak bucket)
14. Arrival time of last packet
15. Average sending rate
16. TSW window length (TSW is a policer based on average transmission rates and the averaging is performed over the window length, in seconds, of data). The default value is 1sec.

The following are the possible policer types:

1. **TSW2CM (TSW2CMPolicer):** uses a CIR and two drop precedences. The lower one is used probabilistically when the CIR is exceeded.
2. **TSW3CM (TSW3CMPolicer)** [**?**]: uses a CIR, a PIR and three drop precedences. The medium priority level is used probabilistically when the CIR is exceeded, and the lowest one is used probabilistically when the PIR is exceeded.
3. **Token Bucket (TokenBucketPolicer)**: uses CIR and a CBS, and two drop precedences.
4. **Single Rate Three Color Marker (srTCMPolicer)** [**?**]: uses CIR, CBS and EBS to choose from three drop precedences.
5. **Two Rate Three Color Marker (trTCMPolicer)** [**?**]: uses CIR, CBS, EBS and PBS to choose from three drop precedences.

- Each of the above policer type defines the meter it uses.
- A policer table defines for each policy type the initial code point as well as one or two downgraded code points.
- The initial code point is often called "green code" and the lowest downgraded code is "red".
- If there is another code point inbetween, it is called "yellow".

10 Configuration

- To update the policy table, the "addPolicyEntry" command is used which contains

1. the edge queue variable denoting the edge queue,
2. the source and destination nodes of the flow,
3. the policer type,
4. its initial code point, and then
5. the values of the parameters that it uses;
6. these are some or all of CIR, CBS, PIR and PBS as stated above.
7. CIR and PIR are given in bps, and CBS, EBS and PBS in bytes.

- An example is:

```
$edgeQueue addPolicerEntry [$n1 id] [$n8 id] trTCM 10 200000 1000 300000 1000
```

- Here we added a policy for the flow that originates in $n1 and ends at $n8.
- If the TSW policers are used, one can add at the end the TSW window length.
- If not added, it is taken to be 1sec by default.
- Another "addPolicyEntry" command specific to the policy and to the initial code point (and not to a particular flow) defines the downgraded code points which are common to all flows that use the policy with the same initial code point.
- An example is:

```
$edgeQueue addPolicerEntry srTCM 10 11 12
```

11 TCL querying

The following three commands result in pringing respectively

1. the entire policy table,
2. the entire policer table and
3. the current size in bytes of the C buckets:

```
$edgeQueue printPolicyTable
$edgeQueue printPolicerTable
$edgeQueue getBucket
```

12 Simulating diffserv: protection of vulnerable packets

- In TCP connections, the loss of some segments has more impact than others on the performance of the connection.
- These segments are

1. the connection establishment segments,
2. the segments sent when the connection has a small window, and
3. the segments sent after a timeout or a fast retransmit. We call these "vulnerable" segments, or packets.

- By marking these segments with a higher priority and implementing the priority using a diffserv architecture, the performance of the TCP connection considerably improves.
- This marking requires, however, that network layer elements be aware of transport layer information, i.e. of the state of the TCP connection.
- The goal of our simulation is to see if one can achieve prioritization of sensitive segments without any use of transport layer information.
- This would simplify the implementation of diffserv marking of TCP packets.

12.1 The simulated scenario

Preliminaries on the service differentiation Two priority levels are defined. The higher "In packets" or "green packets" and the lower "Out packets" or "red packets". We focus on the simplest policer available in ns: the time-sliding window (TSW2CM). A CIR is defined for each edge router. As long as the connection's rate is below CIR, all packets are marked as high priority. When the rate exceeds CIR, packets are marked probabilistically such that at the average, the rate of packets marked with high priority corresponds to the CIR. The transmitted rate is computed as the rate averaged over the "TSW window"; in our simulation its duration is 20msec.

- In our experimentations we vary the CIR level at the source edge nodes and study its impact on performance.

12.2 The topology

- We consnider the simple network topology with a signle bottleneck, depicted in Fig. 1.

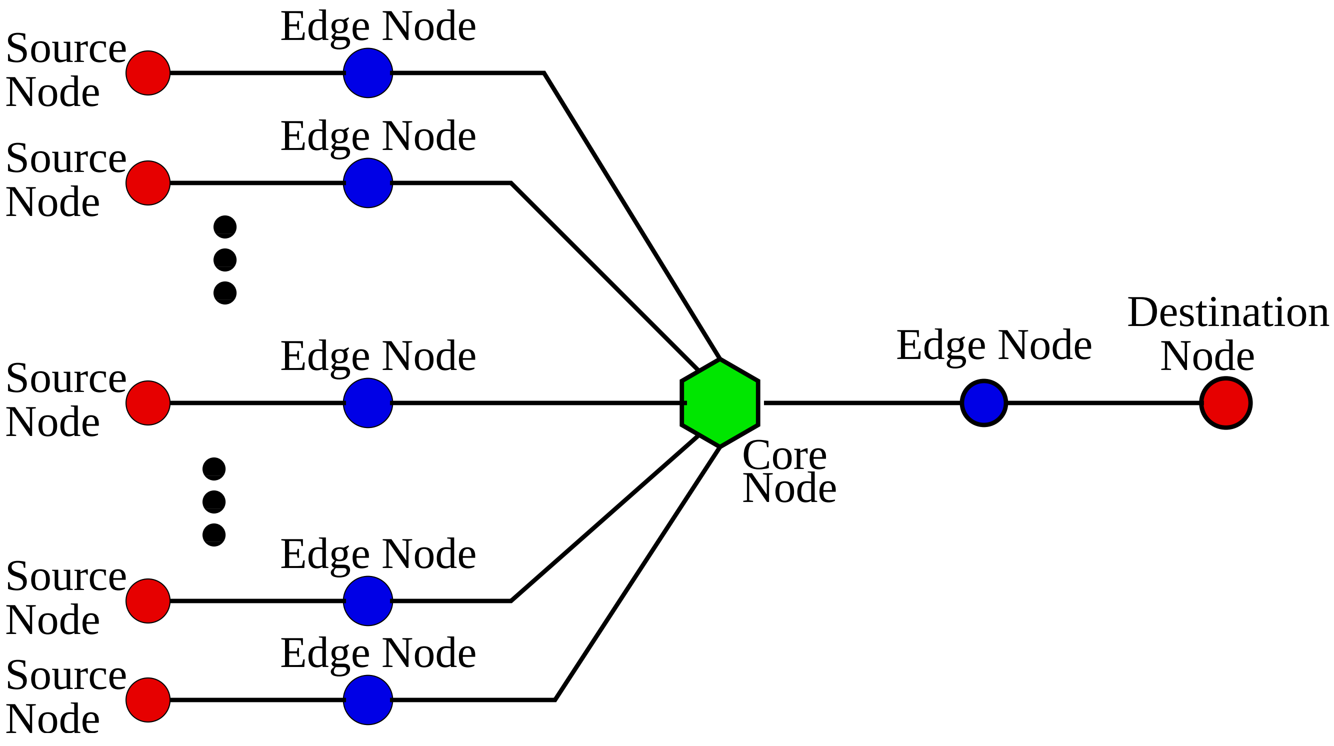

Figure 1: Network topology

- Each source node is connected to a corresponding edge node where the traffic is marked according to parameters that will be specified.
- The edge routers are connected to a bottleneck corerouter, and then through another edge router, to a destination node.
- There are 20 source nodes, and each one of them generates TCP connections.
- We experiment with a high-speed local area type network (short propagation delays) with completely symetric links:
  - Links between the edge nodes and the corresponding source nodes have delays of 10 μsec and 6Mbps bandwidth.
  - Links between the edge node and the corresponding destination node has 10 μsec delay and 10Mbps bandwidth.
  - Links between the core node and edge nodes that are attached to the sources have 0.1msec delay and 6Mbps bandwidth.
  - The link between the core node and edge node attached to the destination has 1msec delay and 10Mbps bandwidth.

12.3 The traffic model

A transfered file has a Pareto distribution with shape parameter 1.25 and an average size of 10kbytes. See:

- T. Bonald and J. Roberts, "Performance modeling of elastic trafic in overload", ACM Sigmetrics, pp. 342-343, 2001.
- B. Sikdar, S. Kalyanaraman and K. S. Vastola, "An Integrated Model for the Latency and Steady-State Throughput of TCP Connections", *Performance Evaluation*, v.46, no.2-3, pp.139-154, September 2001.

●Files to be transmitted arrive at each source node according to a Poisson process with an average rate of 5 files per second.

●Several sessions from the same source node can be active simultaneously.

12.4 Queueing management parameters

- Queue can build up only at the bottleneck router, i.e. at the link between the core node and the edge node that connects to the destination.
- We chose its size to be of 100 packets. Thus the qeueue management parameters at other nodes did not have an influence on the results.
- In the bottleneck queue at the core node, a multi-RED queue management is used with a RIO-D version; we choose the same parameters for both priority levels.
- Our aim in choosing the parameters was not to obtain an optimal performance but rather

  1. to create conditions that allow us to study the effect of diffserv on **diminishing the loss probabilities of vulnerable segments**, and
  2. the impact of this action on TCP performance (delay, throughput).

- For that reason, we choose the same parameters for the two priority levels (this will be explained below).
- For each color of packets (red, green), the averaged queue sized is monitored.
- this is done using the standard exponential averaging with parameter $w_q = 0.01$.
- Packets of a given color start to be dropped when the averaged number of queued packets of this color exceeds min_w;
- we choose $min_w = 15$;
- this drop probability increases linearly with the averaged queue size until it reaches the value $max_w = 45$, where the drop probability is taken to be $max_p = 0.5$.
- When this value is exceeded, the drop probability is 1.

- Often the differentiation between the priorities is done using different sets of parameters:
drops are performed at a larger queue size for green packets. see e.g.

S. Sahu, P. Nain, C. Diot, V. Firoiu and D. F. Towsley", "On achievable service differentiation with token bucket marking for TCP", Proc. ACM SIGMETRICS'00, Santa Clara, CA, June 2000.

- We prefer not to use this approach, since with rejection at a larger window size we also get larger delays, which in some experimented parameters result in a **lower throughput for green packets** than for red packets and in global degradation of performance!
- By giving the same parameters to both priorities, we can learn about the direct effect of protecting vulnerable packets on the TCP performance.
- The differentiation is then done by using the RIO-D approach, in which the rejection probability of each type of color depends on the average number of packets of that type.

- Thus to have green packets dropped less than red ones, we simply choose their throughput (and consequently also the corresponding averge queue size) to be lower;
- this is done by the proper choice of the CIR value which determines the fraction of packets that will be marked green.
- Simulations are 80sec long. The rate of arrival of bits to the bottleneck is

$$\frac{20 \times 1.04 \times 10^4 \times 8}{0.22} = 7.563 Mbps$$

- This is obtained as follows:
  - An average packet size is 1040 bytes of which 1000 are data and 40 bytes are an extra header.
  - An average ftp file is assumed to contain 10^4 10^4 bytes of data, which means that its total average size (including the extra headers) is approximately $1.04 \times 10^4 \times 8$ bits.
  - The result is obtained by multiplying by the number of source nodes and dividing by the average time between arrivals of files at a node.
- The ns script is given in Table 1.

```
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) ----- E(1) ----
#     .                   |
#     .     ---- E(i) ---Core---- Ed -------- D
#     .                   |
#     S(NodeNb)- E(NodeNb)-

set cir0       100000; # policing parameter
set cir1       100000; # policing parameter
set pktSize    1000
set NodeNb       20; # Number of source nodes
set NumberFlows 360 ; # Number of flows per source node
set sduration   60  ; # Duration of simulation
```

```
#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green
$ns color 4 Brown
$ns color 5 Yellow
$ns color 6 Black

set Out [open Out.ns w];   # file containing transfer
                           # times of different connections
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

#Open the NAM trace file
set file2 [open out.nam w]
# $ns namtrace-all $file2
```

```
# defining the topology
set D    [$ns node]
set Ed   [$ns node]
set Core [$ns node]

set flink [$ns simplex-link $Core $Ed 10Mb 1ms dsRED/core]
$ns queue-limit  $Core $Ed  100
$ns simplex-link $Ed $Core 10Mb 1ms dsRED/edge
$ns duplex-link  $Ed   $D  10Mb   0.01ms DropTail

for {set j 1} {$j<=$NodeNb} { incr j } {
 set S($j) [$ns node]
 set E($j) [$ns node]
 $ns duplex-link  $S($j) $E($j)  6Mb   0.01ms DropTail
 $ns simplex-link $E($j) $Core   6Mb   0.1ms dsRED/edge
 $ns simplex-link $Core  $E($j)  6Mb   0.1ms dsRED/core
 $ns queue-limit $S($j) $E($j) 100
}
```

```
#Config Diffserv
set qEdC    [[$ns link $Ed $Core] queue]
$qEdC       meanPktSize 40
$qEdC   set numQueues_   1
$qEdC    setNumPrec      2
for {set j 1} {$j<=$NodeNb} { incr j } {
 $qEdC addPolicyEntry [$D id] [$S($j) id] TSW2CM 10 $cir0 0.02
}
$qEdC addPolicerEntry TSW2CM 10 11
$qEdC addPHBEntry  10 0 0
$qEdC addPHBEntry  11 0 1
$qEdC configQ 0 0 10 30 0.1
$qEdC configQ 0 1 10 30 0.1

$qEdC printPolicyTable
$qEdC printPolicerTable
```

```
set qCEd    [[$ns link $Core $Ed] queue]
# set qCEd    [$flink queue]
$qCEd     meanPktSize $pktSize
$qCEd set numQueues_   1
$qCEd set NumPrec       2
$qCEd addPHBEntry  10 0 0
$qCEd addPHBEntry  11 0 1
$qCEd setMREDMode RIO-D
$qCEd configQ 0 0 15 45  0.5 0.01
$qCEd configQ 0 1 15 45  0.5 0.01
```

```
for {set j 1} {$j<=$NodeNb} { incr j } {
 set qEC($j) [[$ns link $E($j) $Core] queue]
 $qEC($j) meanPktSize $pktSize
 $qEC($j) set numQueues_   1
 $qEC($j) setNumPrec      2
 $qEC($j) addPolicyEntry [$S($j) id] [$D id] TSW2CM 10 $cir1 0.02
 $qEC($j) addPolicerEntry TSW2CM 10 11
 $qEC($j) addPHBEntry  10 0 0
 $qEC($j) addPHBEntry  11 0 1
# $qEC($j) configQ 0 0 20 40 0.02
 $qEC($j) configQ 0 0 10 20 0.1
 $qEC($j) configQ 0 1 10 20 0.1

$qEC($j) printPolicyTable
$qEC($j) printPolicerTable
```

```
 set qCE($j) [[$ns link $Core $E($j)] queue]
 $qCE($j) meanPktSize      40
 $qCE($j) set numQueues_   1
 $qCE($j) setNumPrec      2
 $qCE($j) addPHBEntry  10 0 0
 $qCE($j) addPHBEntry  11 0 1
# $qCE($j) configQ 0 0 20 40 0.02
 $qCE($j) configQ 0 0 10 20 0.1
 $qCE($j) configQ 0 1 10 20 0.1
}

# 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 22

# Random inter-arrival times of TCP transfer at each source i
set RV [new RandomVariable/Exponential]
$RV set avg_ 0.22
$RV use-rng $rng1

# Random size of files to transmit
set RVSize [new RandomVariable/Pareto]
$RVSize set avg_ 10000
$RVSize set shape_ 1.25
$RVSize use-rng $rng1

# dummy command
set t [$RVSize value]
```

```
# 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
 $tcpsrc($i,$j) set sess $j
 $tcpsrc($i,$j) set node $i
 set t [expr $t + [$RV value]]
 $tcpsrc($i,$j) set starts $t
         $tcpsrc($i,$j) set size [expr [$RVSize value]]
  $ns at [$tcpsrc($i,$j) set starts] "$ftp($i,$j) send [$tcpsrc($i,$j) set size]"
  $ns at [$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] - [$self set starts] ]
  set i [$self set node]
  set j [$self set sess]
  set time [$ns now]
```

```
  puts $Out "$i \t $j \t $time \t\
      $time \t $duration \t [$self set ndatapack_] \t\
      [$self set ndatabytes_] \t [$self set  nrexmitbytes_] \t\
      [expr [$self set ndatabytes_]/$duration ]"
  # update the number of flows
      countFlows [$self set node] 0
}

# The following recursive procedure updates the number of connections
# as a function of time. Each 0.2 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 it 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"
} }
```

```
#Define a 'finish' procedure
proc finish {} {
        global ns tf qsize qbw qlost file2
        $ns flush-trace
        close $file2
        exit 0
}

$ns at 0.5 "countFlows 1 3"
$ns at [expr $sduration - 0.01] "$fmon dump"
$ns at [expr $sduration - 0.001] "$qCEd printStats"
$ns at $sduration "finish"
$ns run
```

Table 1: diffs.tcl Script

13 Simulation results

13.1 Losses

●We check the influence of the CIR marking rate on the loss probabilities of the SYN packets and of the first data packet in a connection, as before.

CIR	10kbps	30kbps	100kbps	200kbps	300kbps	1Mbps	10Mbps
lost SYN packets	120	95	53	45	17	78	114
First packets lost	125	119	90	56	37	73	115
Total losses	1699	1612	1476	1286	1088	1290	1577

Table 2: Protection of vulnerable packets as a function of CIR

●We manage to decrease the losses of SYN packets by a factor of seven, and the losses of the first data packet of a connection by a factor of around 3.4, both obtained at CIR of 300kbps.

13.2 Throughput and Goodput

- The number of data packets that were successfully transmitted during the simulations was quite independent of the CIR:
- it was in the average 58285, with a standard deviation of 395 packets.
- This is due to the fact that arrival rate of sessions does not depend on the CIR.
- In view of the low loss probabilities, the throughput too, is almost constant as a function of CIR.

13.3 The number of sessions

- The total number of sessions as a function of time is given in Fig. 2 for the CIR of 200kbps (the optimal) and for the case of no prioritization (CIR of 10Mbps).

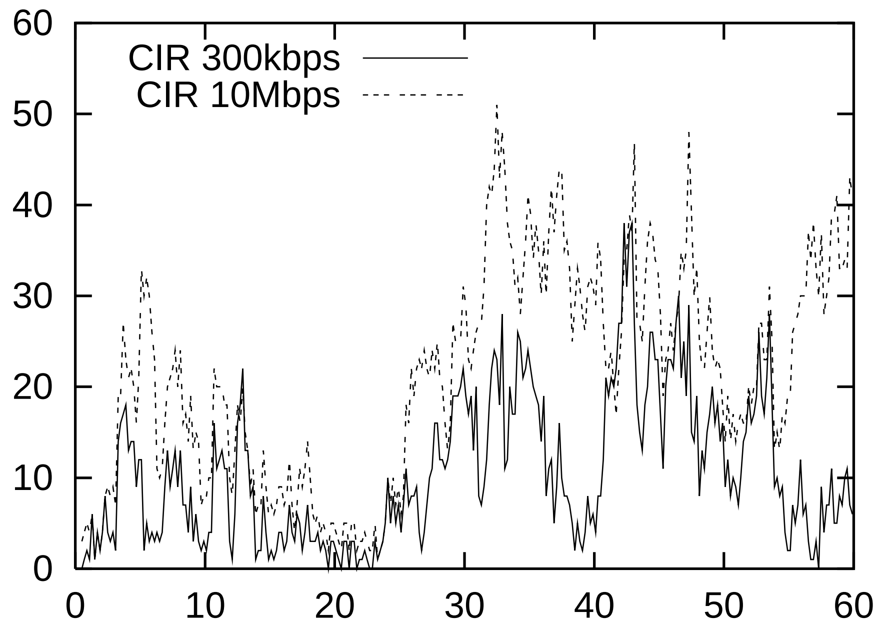

Figure 2: Evolution of total number of sessions

- We see that our marking scheme with CIR of 300kbps gives a better performance: less number of active sessions are present under this marking.

- This is related to the fact that the session duration in our marking is shorter, as we show in the next section.
- Indeed, since the arrival rate of sessions is the same independently of CIR, the average number of session should be proportional to the average duration of a session (the proportionality factor being the arrival rate of sessions).

13.4 Session duration

- In Table 3 we present the average duration of a session as a function the CIR.

CIR	10kbps	30kbps	100kbps	200kbps	300kbps	1Mbps	10Mbps
sess. duration	0.252616	0.231077	0.167948	0.149478	0.119509	0.203202	0.225075

Table 3: Average duration of a session as a function of CIR

- We see that in the range of 100kbps till 300kbps the average duration decreases by a factor between 1/3 (for 100kbps) and 1/2 (for 300kbps) with respect to the case of no prioritization.

14 Excercise

Our simulations have restricted to FTP type traffic. In this exercise we shall consider **HTTP type traffic:** The time between the end of transmission of a file till the beginning of the next transmission is exponentially distributed with a mean of 0.1 sec.

This is called a "thinking time".

Thus from each source node there can be only one file transmitted at the same time (at most one active session).

Write a tcl program for this traffic model and check its performance as a function of the CIR. Compare with FTP type traffic