Helsinki University of Technology
Department of Electrical and Communication Engineering

S-38.180 Quality of Service in Internet
Home assignment 1: Simulation of Rate Control mechanisms

Due date: 31.10.2001 at 1200 hours
Delivery: Paper report to course locker at G-wing 2nd floor

Contents

Chapter 1

Introduction

This is course work for course S-38.180 Quality of Service in Internet. This exercise makes you familiar with:

1. ns2. A network simulator which is one of the most used simulation tools in academic world.

2. Rate control mechanisms which could be used to mark, limit or shape the Internet traffic

Tool is installed into department workstation room into SUN workstations. You need to run it there. Program can be located from path */proj/bones/ns2/*. To be able to use it you must first run command */proj/bones/ns2/ns-allinone-2.1b8/use*, which tells you the **format** of environment specific use command. Typing the command outputted by generic use, you will set all environment variables required to run *ns2*.

Make working directory into your home directory and copy file */proj/bones/ns2/ns-allinone-2.1b8/work/example.tcl* to it. This file is the core which you must modify during the work.

Department has set quotas to workstations, so you have to run your simulations in */tmp* directory of workstations. Make temporary working directory under */tmp*. Copy processed results to your working directory and delete temporary working directory.

1.1 Network Simulator 2

Tool used in this exercise is *network simulator 2*. It is freely availabale and distributable object oriented simulator. In glance, it contains kernel code developed in **C++**. This kernel code contains class hierarchies for ready made building blocks. These building blocks are translated to **Otcl** to make construction of 'quick and dirty' simulations easier.

You may want to become familiar with the tool before starting your work. ns2 homepage is in http://www.isi.edu/nsnam/ns/index.html. Recomendend reading is the tutorial page in http://www.isi.edu/nsnam/ns/tutorial/index.html.

Chapter 2

Simulator code

This chapter present simulator code which is used in this exercise. Following listing shows whole code as one unit. To become familiar the structure of *ns*, we dissect code into peaces and explain what they do in detail.

```
set ns [new Simulator]

set rate1 2000000
set cir1  1500000

set rate2 2000000
set cir2  1500000

set cir3  1500000

set cir4  1500000

set cir5  1000000

set cir6  1000000

set cir7  1000000

set cir8  1000000

set cir9  0

set cir10  0

set testTime 50
set packetSize 1000

# Open a file for writing the trace data
set trace_all [open out.all w]

# Trace all events for post processing with animator (nam) or with any
# other software
$ns trace-all $trace_all
```

```
# Set up the network topology shown at the top of this file:
set s1 [$ns node]
set s2 [$ns node]
set s3 [$ns node]
set s4 [$ns node]
set s5 [$ns node]
set s6 [$ns node]
set s7 [$ns node]
set s8 [$ns node]
set s9 [$ns node]
set s10 [$ns node]
set e1 [$ns node]
set e2 [$ns node]
set dest [$ns node]

$ns duplex-link $s1 $e1 10Mb 5ms DropTail
$ns duplex-link $s2 $e1 10Mb 5ms DropTail
$ns duplex-link $s3 $e1 10Mb 5ms DropTail
$ns duplex-link $s4 $e1 10Mb 5ms DropTail
$ns duplex-link $s5 $e1 10Mb 5ms DropTail
$ns duplex-link $s6 $e1 10Mb 5ms DropTail
$ns duplex-link $s7 $e1 10Mb 5ms DropTail
$ns duplex-link $s8 $e1 10Mb 5ms DropTail
$ns duplex-link $s9 $e1 10Mb 5ms DropTail
$ns duplex-link $s10 $e1 10Mb 5ms DropTail

$ns simplex-link $e1 $e2 10Mb 5ms dsRED/edge
$ns simplex-link $e2 $e1 10Mb 5ms dsRED/edge

$ns duplex-link $e2 $dest 10Mb 5ms DropTail


$ns duplex-link-op $e1 $e2 orient right
$ns duplex-link-op $e2 $dest orient right


set qE1E2 [[$ns link $e1 $e2] queue]
set qE2E1 [[$ns link $e2 $e1] queue]


# Set DS RED parameters from Edge1 to Core:
$qE1E2 meanPktSize $packetSize
$qE1E2 set numQueues_ 2
$qE1E2 setNumPrec 2
$qE1E2 addPolicyEntry [$s1 id] [$dest id] TSW2CM 20 $cir1
$qE1E2 addPolicyEntry [$s2 id] [$dest id] TSW2CM 20 $cir2
$qE1E2 addPolicyEntry [$s3 id] [$dest id] TSW2CM 10 $cir3
$qE1E2 addPolicyEntry [$s4 id] [$dest id] TSW2CM 10 $cir4
$qE1E2 addPolicyEntry [$s5 id] [$dest id] TSW2CM 10 $cir5
$qE1E2 addPolicyEntry [$s6 id] [$dest id] TSW2CM 10 $cir6
$qE1E2 addPolicyEntry [$s7 id] [$dest id] TSW2CM 10 $cir7
$qE1E2 addPolicyEntry [$s8 id] [$dest id] TSW2CM 10 $cir8
```

```
$qE1E2 addPolicyEntry [$s9 id] [$dest id] TSW2CM 10 $cir9
$qE1E2 addPolicyEntry [$s10 id] [$dest id] TSW2CM 10 $cir10
$qE1E2 addPolicerEntry TSW2CM 10 11
$qE1E2 addPolicerEntry TSW2CM 20 21
$qE1E2 addPHBEntry 10 0 0
$qE1E2 addPHBEntry 11 0 1
$qE1E2 addPHBEntry 20 0 0
$qE1E2 addPHBEntry 21 1 1
$qE1E2 configQ 0 0 10 40 0.02
$qE1E2 configQ 0 1 10 40 0.10
$qE1E2 configQ 1 1 0 0 1

# Set DS RED parameters from Edge2 to Core:
$qE2E1 meanPktSize $packetSize
$qE2E1 set numQueues_ 2
$qE2E1 setNumPrec 2
$qE2E1 addPolicyEntry [$dest id] [$s1 id] TSW2CM 20 $cir1
$qE2E1 addPolicyEntry [$dest id] [$s2 id] TSW2CM 20 $cir2
$qE2E1 addPolicyEntry [$dest id] [$s3 id] TSW2CM 10 $cir3
$qE2E1 addPolicyEntry [$dest id] [$s4 id] TSW2CM 10 $cir4
$qE2E1 addPolicyEntry [$dest id] [$s5 id] TSW2CM 10 $cir5
$qE2E1 addPolicyEntry [$dest id] [$s6 id] TSW2CM 10 $cir6
$qE2E1 addPolicyEntry [$dest id] [$s7 id] TSW2CM 10 $cir7
$qE2E1 addPolicyEntry [$dest id] [$s8 id] TSW2CM 10 $cir8
$qE2E1 addPolicyEntry [$dest id] [$s9 id] TSW2CM 10 $cir9
$qE2E1 addPolicyEntry [$dest id] [$s10 id] TSW2CM 10 $cir10
$qE2E1 addPolicerEntry TSW2CM 10 11
$qE2E1 addPolicerEntry TSW2CM 20 21
$qE2E1 addPHBEntry 10 0 0
$qE2E1 addPHBEntry 11 0 1
$qE2E1 addPHBEntry 20 0 0
$qE2E1 addPHBEntry 21 1 1
$qE2E1 configQ 0 0 10 40 0.02
$qE2E1 configQ 0 1 10 40 0.10
$qE2E1 configQ 1 1 0 0 1

# Set up one CBR connection between each source and the destination:
set udp1 [new Agent/UDP]
$ns attach-agent $s1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packet_size_ $packetSize
$udp1 set packetSize_ $packetSize
$cbr1 set rate_ $rate1
set null1 [new Agent/Null]
$ns attach-agent $dest $null1
$ns connect $udp1 $null1

set udp2 [new Agent/UDP]
```

```
$ns attach-agent $s2 $udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
$cbr2 set packet_size_ $packetSize
$udp2 set packetSize_ $packetSize
$cbr2 set rate_ $rate2
set null2 [new Agent/Null]
$ns attach-agent $dest $null2
$ns connect $udp2 $null2

set tcp3 [new Agent/TCP]
$ns attach-agent $s3 $tcp3
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$tcp3 set packetSize_ $packetSize
set sink3 [new Agent/TCPSink]
$ns attach-agent $dest $sink3
$ns connect $tcp3 $sink3


set tcp4 [new Agent/TCP]
$ns attach-agent $s4 $tcp4
set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp4
$tcp4 set packetSize_ $packetSize
set sink4 [new Agent/TCPSink]
$ns attach-agent $dest $sink4
$ns connect $tcp4 $sink4


set tcp5 [new Agent/TCP]
$ns attach-agent $s5 $tcp5
set ftp5 [new Application/FTP]
$ftp5 attach-agent $tcp5
$tcp5 set packetSize_ $packetSize
set sink5 [new Agent/TCPSink]
$ns attach-agent $dest $sink5
$ns connect $tcp5 $sink5


set tcp6 [new Agent/TCP]
$ns attach-agent $s6 $tcp6
set ftp6 [new Application/FTP]
$ftp6 attach-agent $tcp6
$tcp6 set packetSize_ $packetSize
set sink6 [new Agent/TCPSink]
$ns attach-agent $dest $sink6
$ns connect $tcp6 $sink6


set tcp7 [new Agent/TCP]
$ns attach-agent $s7 $tcp7
```

```
set ftp7 [new Application/FTP]
$ftp7 attach-agent $tcp7
$tcp7 set packetSize_ $packetSize
set sink7 [new Agent/TCPSink]
$ns attach-agent $dest $sink7
$ns connect $tcp7 $sink7


set tcp8 [new Agent/TCP]
$ns attach-agent $s8 $tcp8
set ftp8 [new Application/FTP]
$ftp8 attach-agent $tcp8
$tcp8 set packetSize_ $packetSize
set sink8 [new Agent/TCPSink]
$ns attach-agent $dest $sink8
$ns connect $tcp8 $sink8


set tcp9 [new Agent/TCP]
$ns attach-agent $s9 $tcp9
set ftp9 [new Application/FTP]
$ftp9 attach-agent $tcp9
$tcp9 set packetSize_ $packetSize
set sink9 [new Agent/TCPSink]
$ns attach-agent $dest $sink9
$ns connect $tcp9 $sink9


set tcp10 [new Agent/TCP]
$ns attach-agent $s10 $tcp10
set ftp10 [new Application/FTP]
$ftp10 attach-agent $tcp10
$tcp10 set packetSize_ $packetSize
set sink10 [new Agent/TCPSink]
$ns attach-agent $dest $sink10
$ns connect $tcp10 $sink10



proc finish {} {
    global ns trace_all
    close $trace_all

    #Following are awk scripts to process trace of all events.

    exec awk {
{
    if (($1 == "r") && ($4 == "12"))

    print $2, $9, $6
}
    } out.all > out.rec
```

```
    exit 0
}

$qE1E2 printPolicyTable
$qE1E2 printPolicerTable

$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"
$ns at 0.0 "$ftp6 start"
$ns at 0.0 "$ftp7 start"
$ns at 0.0 "$ftp8 start"
$ns at 0.0 "$ftp9 start"
$ns at 0.0 "$ftp10 start"
$ns at 10.0 "$qE1E2 printStats"
$ns at 20.0 "$qE1E2 printStats"
$ns at 30.0 "$qE1E2 printStats"
$ns at 40.0 "$qE1E2 printStats"
$ns at $testTime "$cbr1 stop"
$ns at $testTime "$cbr2 stop"
$ns at $testTime "$ftp3 stop"
$ns at $testTime "$ftp4 stop"
$ns at $testTime "$ftp5 stop"
$ns at $testTime "$ftp6 stop"
$ns at $testTime "$ftp7 stop"
$ns at $testTime "$ftp8 stop"
$ns at $testTime "$ftp9 stop"
$ns at $testTime "$ftp10 stop"
$ns at [expr $testTime + 1.0] "finish"

$ns run
```

2.1 Create topology

Topology in ns is based on collection of nodes and links. Topology is basic requirement for the succesful simulation of particular scenario. Therefore topology is so called base requirement and components od topology are defined in base level of ns. This is easily detectable with the declarations which start by `$ns`.

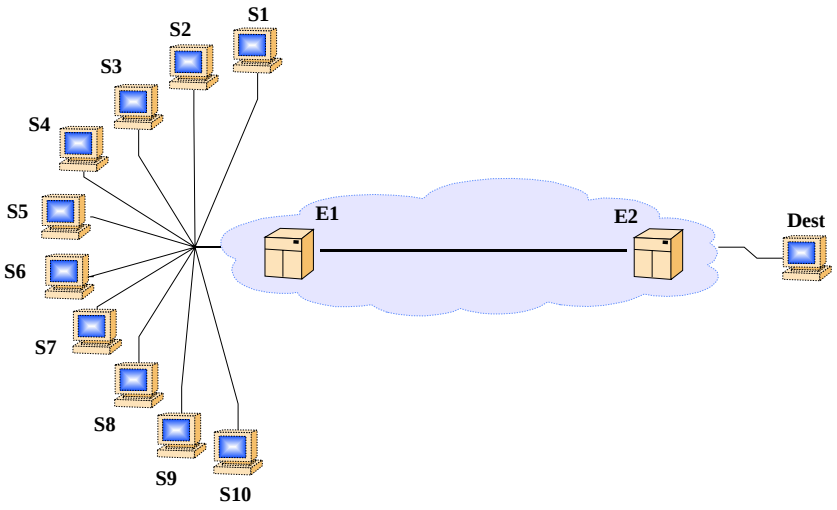

Figure 2.1: Topology of simulation

2.1.1 Nodes

There is no differentiation between end systems and routers in creation of topology, i.e. each node can be end system and/or router.

Node is created with command `set s1 [$ns node]`

`s1` is the name for the node which is later on used as pointer to the node. `[$ns node]` tells to the ns that properties of *class node* should be assigned to new object named `s1`. When node is an end system, we must add protocols and traffic generation models to it. In our exercise we have two types of clients.

1. Traffic sources

   Traffic sources use UDP and TCP as their transport protocol. Protocol for the end system is created with commands

```
--><--
set udp1 [new Agent/UDP]
$ns attach-agent $s1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packet_size_ $packetSize
$udp1 set packetSize_ $packetSize
$cbr1 set rate_ $rate1
--><--

--><--
set tcp10 [new Agent/TCP]
```

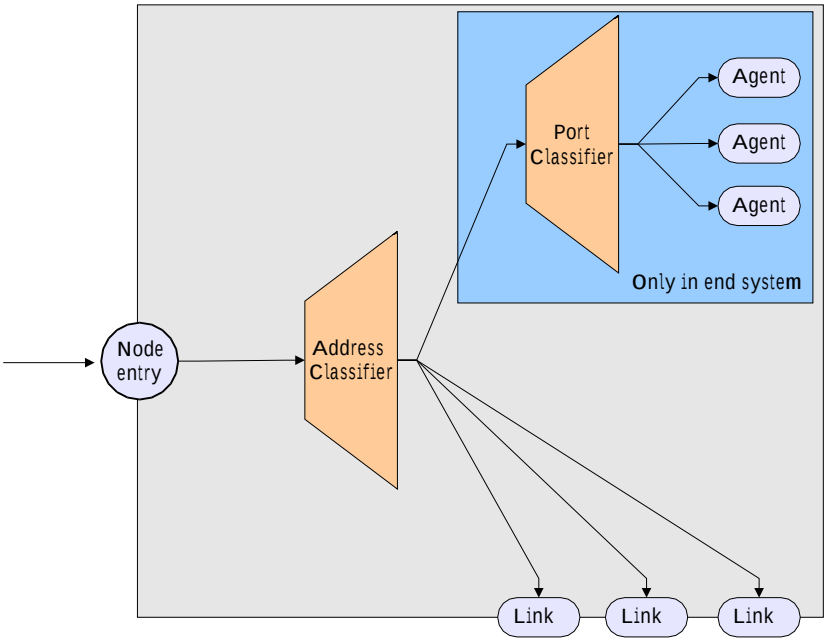

Figure 2.2: Construction of node in ns2

```
$ns attach-agent $s10 $tcp10
set ftp10 [new Application/FTP]
$ftp10 attach-agent $tcp10
$tcp10 set packetSize_ $packetSize
--><--
```

Where `udp1` and `tcp10` are the names for the protocol stacks. These names are used for combining right node and source model to these protocol stacks. `[new Agent/UDP]` and `[new Agent/TCP]` are used to tell for ns to combine properties of *class UDP* and *class TCP* to new objects named `udp1` and `tcp10`. These classes inherit all features of their parents, i.e. Agent.

Protocol stack is attached to particular node with the command `$ns attach-agent $s1 $udp1`. Which combines node `$s1` to protocol stack `$udp1` and node `$s10` to protocol stack `$tcp10` respectively.

Source type, i.e. traffic generation pattern, is set by defining the client.

UDP protocol is attached to application which generates constant bit rate traffic. This is done first by defining the application `set cbr1 [new Application/Traffic/CBR]` and then attaching the application to the transport protocol `$cbr1 attach-agent $udp1`. `$cbr1` inherits all features of Application, Traffic and CBR. One of the features is that

CBR traffic is defined by rate and packet size. These are defined by `$cbr1 set rate_` and `$cbr1 set packet_size_`. They are connected to locally modifiable variables located at the beginning of tcl script.

TCP protocol is attached to FTP application. Client name is `ftp10` and its operation is defined in class *class Application/FTP*. FTP without any additional parametrisation is a greedy traffic source, i.e. it tries to fill the pipe full of its traffic.

2. Traffic sinks

   Traffic sinks are points where the flow of information are terminated. There are several different types of sinks for different stacks and protocol levels. We are not interested in the operation of TCP and UDP, so we will terminate information flows without any processing. We create a TCP sink which is defined by class *Agent/TCPSink* and name it as `sink10`. This sink then attached to the node `dest`. Sameway we create UDP sink which is defined by class *Agent/Null* and attach that to same destination node `dest`.

Clients are connected together (after the links are declared) with the command

```
--><--
$ns connect $udp1 $null1
$ns connect $tcp10 $sink10
--><--
```

This command connects clients at the level of transmission protocol together, i.e. state is established on the TCP.

2.1.2 Links

Links are the other part of topology. Because nodes are universal, i.e. end systems and routers share same construction, additional mechanisms are implemented into the links. Links contain queues which in real world would have been implemented in routers.

Links are created with command

```
--><--
$ns duplex-link $s1 $e1 10Mb 5ms DropTail
$ns simplex-link $e1 $e2 10Mb 5ms dsRED/edge
--><--
```

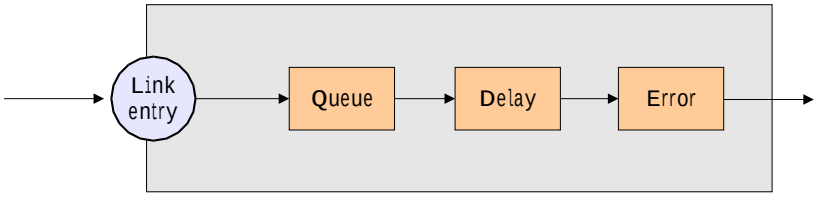

Figure 2.3: Construction of link in ns2

First parameter, `duplex-link` in command expresses the type of link. This type may vary depending on what you are doing from simplex, duplex or some special link, like CBQLink or IntServLink. We use in our exercise duplex link to carry the data from source to access point and simplex links in core network.

Second and third parameters refer from where to where link goes. End points are nodes which are attached to the link, i.e. example shows link going from `s1` to `e1`. Ordering of end points does not account when we are talking about duplex links. However, simplex links are directed based on the order of end points.

Fourth parameter is the bandwidth (data rate) of the link. You may use qualifiers k (kilo) and M (mega) as b (bit) and B (byte).

Fifth parameter is the delay of the link this has same idea as bandwidth you may use m (milli) and u (mikro) as qualifiers.

Last parameter stands for queue management algorithm used in the queue. Valid arguments are DropTail (FIFO), RED, dsRED/edge, dsREDcore, CBQ, FQ, SFQ and DRR. We use DropTail everywhere else but at the core network to have most straight forward simulation.

For visualisation in nam ns contains possibility to adjust topology on a way you want. This is done with command

```
$ns duplex-link-op $s1 $e1 orient right-down
```

which makes possible to give orientational directions of links. Orientation is in direction of *from* to *to*. Distance between two nodes comes from the delay. All distances are scaled based on the shortest distance.

Additional parameters for the link can be given with same command `simplex-link-op`. In command

```
$ns simplex-link-op $e1 $e2 queuePos 0.5
```

queue on link between `e1` and `e2` is monitored in nam visualisation.

2.1.3 Building Differentiated Services

Differentiated Services support in *ns2* is based on the modified RED queue (one physical queue contains three virtual queues). Different RED parameters are used for different virtual queues, causing some virtual queue to have lenient operation while other are more strained.

Base of the DiffServ operation is the policy which is set for the simulation. Policy declares how particular connection should be metered, marked or policed during the simulation.

```
--><--
$qE2E1 addPolicyEntry [$dest id] [$s1 id] TSW2CM 20 $cir1
$qE2E1 addPolicyEntry [$dest id] [$s10 id] TSW2CM 10 $cir10
--><--
```

These policies define connection which is under the control i.e. from `$dest` to `$s1` and `$a10`. Also the type of metering and marking is defined `TSW2CM` which stands for Time Sliding Window Metering with Two Color Marking. Two Color marking means simply in or out of profile marking. Last two colums in policy entry define the codepoint in case if packet falls within policy and parameter used to set up metering and marking.

ns2 has build in support for following metering and marking combinations:

1. TSW2CM: Time Sliding Window Meter with Two Color Marker. TSW2CM is controlled with single parameter Committed Information Rate (CIR). CIR is the rate for which ISP offers QoS level commitment. Traffic falling out of profile is probabilistally marked to lower precedence.
2. TSW3CM: Time Sliding Window Meter with Three Color Marker. TSW3CM is controlled with two parameters Committed Information Rate (CIR) and Peak Information Rate (PIR). CIR is the rate for which ISP offers QoS level commitment. Traffic falling out of CIR is probabilistally marked to medium precedence. Traffic falling out of PIR is probabilistally marked to lowest precedence.

3. tokenBucket: Token Bucket uses CIR and committed burst size (CBS) with two drop precedences. Packet is marked to lower precedence if it falls out of profile defined by CIR (token generation rate * token size) and CBS (size of the token bucket).

4. srTCM: Single Rate Three Color Marker uses combination of two token buckets in cascade. Traffic is metered by CIR, CBS and excess burst size (EBS). Excess burst size is addtional burst level defined by second token bucket. Traffic falling in first bucket is marked to highest precedence. Traffic falling within second but not first token bucket is marked for medium drop precedence. Traffic falling out from both of the buckets is marked to lowest precedence.

5. trTCM: Two Rate Three Color Marker uses a combination of two token bucket in cascade. Traffic is metered by CIR and CBS in first token bucket, and PIR and peak burst size (PBS) in second token bucket. Traffic falling in first bucket is marked to highest precedence. Traffic falling within second but not first token bucket is marked for medium drop precedence. Traffic falling out from both of the buckets is marked to lowest precedence.

Ones the policy is defined. Policers which are attached to policy need to be parametrized. Parameters which they require are marks which are associated to their conditioning actions.

```
--><--
$qE1E2 addPolicerEntry TSW2CM 10 11
$qE1E2 addPolicerEntry TSW2CM 20 21
--><--
```

After the policy is completely established one must take care about forwarding of the marked packets. Each codepoint (mark) needs to have associated forwarding treatment. Forwarding treatments are in Differentiated Services called per hop behaviors (PHB). In *ns2* PHB is determined only on the level of queue and precendence which some codepoint reflects `$qE1E2 addPHBEntry 10 0 0`. RED algorithms operating on the queues need also to be paramtrised through `$qE1E2 configQ 0 1 10 40 0.10`

2.2 Probing the information

Event monitoring is special class of Tcl operation, where a Trace class object is wrapped around monitored system.

2.2.1 Starting event monitoring

Event monitoring means that all events surrounding some object are recorded into the trace file. For that we need to open the trace file and connect a handle to that file I/O.

```
--><--
set trace_all [open out.all w]
--><--
```

After the file is opened events can be recorded to that. Simulator contains a special commands to do that. Commands specify the object which is traced and the handle to the file where events are recorded. Following command

```
--><--
$ns trace-all $trace_all
--><--
```

records all events from the simulation to the file pointed by handle `$trace_all`. After the simulation trace buffer must be cleared and the file closed

```
--><--
$ns flush-trace
close $trace_all
--><--
```

2.2.2 Trace file format

Trace mechanism generates a trace file where events are stored for post processing. Trace file contains information in a form of white space separated table.

```
--><--
r 6.6938 4 5 tcp 1000 ------- 1 0.0 2.0 657 1301
+ 6.6938 5 2 tcp 1000 ------- 1 0.0 2.0 657 1301
- 6.6938 5 2 tcp 1000 ------- 1 0.0 2.0 657 1301
r 6.694155 1 4 tcp 1000 ------- 2 1.0 3.0 10 1356
+ 6.694155 4 5 tcp 1000 ------- 2 1.0 3.0 10 1356
d 6.694155 4 5 tcp 1000 ------- 2 1.0 3.0 10 1356
r 6.6946 0 4 tcp 1000 ------- 1 0.0 2.0 684 1357
--><--
```

First column of table represents the type of event. Event may be receive ('r'), drop ('d'), engue ('+') or deque ('-').

Second column is the simulated time when the event occured. Time is given in seconds from the beginning of simulation.

Third and fourth columns are source and destination nodes of event, i.e. they show where the tracing of event took place and on which direction.

Fifth column expresses the type of packet that was traced. Type may be protocol or action, depending on agent that generated the packet.

Sixth column indicates the packet size as encoded in IP header.

Seveth column (-------) represent flags which may be coded in packets.

Eigth column is flow id which is used to separate connections in aggregated links and end points.

Nineth and tenth columns are source and destinations addresses. If special addressing is used, this shows it. Otherwise it is the node numbers in decimal format.

Eleventh column is the sequence number field. This field is used for protocols and agents which make use of sequence numbers.

Twelvth column is the uniqueue id of packet. Each packet has uniqueue id throughou the simulation. This makes easy to trace events which have happened to a single packet on a path through the network.

2.3 Controlling simulation

Simulation is started and stopped with commands

```
--><--
$ns run
$ns at [expr $testTime + 1.0] "finish"
--><--
```

Finish command schedules the subroutine finish at the time set by parameter `$testTime`. Subroutine finish is used to close trace files, to do post processing of information (not neccessary) and to plot trace information (not neccesary). Last command in subroutine is `exit` which is used to terminate the program.

```
--><--
```

```
proc finish {} {
    global ns  trace_all
    $ns flush-trace
    close $trace_all
...
    exit 0
}
--><--
```

Other events which must be scheduled are client start and stop times. These events control traffic generation within the network.

```
--><--
$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$ftp3 start"
$ns at 0.0 "$ftp4 start"
$ns at 0.0 "$ftp5 start"
$ns at 0.0 "$ftp6 start"
$ns at 0.0 "$ftp7 start"
$ns at 0.0 "$ftp8 start"
$ns at 0.0 "$ftp9 start"
$ns at 0.0 "$ftp10 start"
$ns at $testTime "$cbr1 stop"
$ns at $testTime "$cbr2 stop"
$ns at $testTime "$ftp3 stop"
$ns at $testTime "$ftp4 stop"
$ns at $testTime "$ftp5 stop"
$ns at $testTime "$ftp6 stop"
$ns at $testTime "$ftp7 stop"
$ns at $testTime "$ftp8 stop"
$ns at $testTime "$ftp9 stop"
$ns at $testTime "$ftp10 stop"
--><--
```

Chapter 3

Exercise

This exercise is about comparison of different rate control methods in Internet. Methods which are used here are token bucket, time sliding window and single rate two color marker.

Invetigate operation of different metering and marking algorithms in network which is used for transmission of pure TCP, pure UDP and mixed (2-4 UDP clients and 6-8 TCP clients) traffic. Use different combinations of rates (all clients equal capacity, some clients substantially higher capacity and some client zero capacity). See what is the outcome of the service expressed as recovered capacities. See also how rate control mechanisms work when you load them with different levels of traffic load (limit to substantial overload).

Due date for the exercise is 31.10.2001 at 1200 hours. Reports can only be delivered to course locker in G-wing 2nd floor. Other means used in delivery are not considered.