

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Electrical and Communication Engineering

Quality of Service in Internet

Exercise 2: Simulation of access control and queue management methods

Due date: 25.10.2002 at 0900 hours
Delivery: Paper report to the course locker

Contents

Contents	i
1 Introduction	1
1.1 Network Simulator 2	2
2 Simulator code	3
2.1 Create topology	11
2.1.1 Nodes	11
2.1.2 Links	14
2.1.3 Rate control mechanisms	16
2.2 Probing the information	17
2.2.1 Starting event monitoring	18
2.2.2 Trace file format	18
2.3 Controlling simulation	19
3 Exercise	21

Chapter 1

Introduction

This is exercise for the Helsinki Summer School course 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 SUN workstations in Helsinki University of Technology department of Electrical and Communications Engineering workstation room B215. You can run it there or download it to your own computer from the world wide distribution site <http://www.isi.edu/nsnam/ns/index.html>. In HUT program can be located from the path `/p/gen/courses/S38/S38.180/Exercise2/`. To be able to use it, you must first run command `/p/gen/courses/S38/S38.180/Exercise2/usens2.sh` or `usens2.csh` depending on shell you use.

To start, make working directory into your home directory and copy files from the directory `/p/gen/courses/S38/S38.180/Exercise2/doc/HT2.tar` to it. These files are the to core which you must make modifications during the exercise.

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 available 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.

If you have any problems with *ns2*, I recommend you to read 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 testTime 25

set rate11 50000
set cir11 50000
set cbs11 2000

set rate21 50000
set cir21 50000
set cbs21 2000

set rate31 50000
set cir31 50000
set cbs31 2000

set rate41 50000
set cir41 50000
set cbs41 2000

set rate51 50000
set cir51 50000
set cbs51 2000


set cir12 350000
set cbs12 20000

set cir22 350000
set cbs22 20000

set cir32 350000
```

```

set cbs32      20000

set cir42      350000
set cbs42      20000

set cir52      350000
set cbs52      20000

set TCP_Packet_Size 1000
set UDP_Packet_Size 160

# 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 s11 [$ns node]
set s12 [$ns node]
set s21 [$ns node]
set s22 [$ns node]
set s31 [$ns node]
set s32 [$ns node]
set s41 [$ns node]
set s42 [$ns node]
set s51 [$ns node]
set s52 [$ns node]
set e1 [$ns node]
set e2 [$ns node]
set e3 [$ns node]
set e4 [$ns node]
set e5 [$ns node]
set c1 [$ns node]
set c2 [$ns node]
set dest [$ns node]

$ns duplex-link $s11 $e1 10Mb 5ms DropTail
$ns duplex-link $s12 $e1 10Mb 5ms DropTail
$ns duplex-link $s21 $e2 10Mb 5ms DropTail
$ns duplex-link $s22 $e2 10Mb 5ms DropTail
$ns duplex-link $s31 $e3 10Mb 5ms DropTail
$ns duplex-link $s32 $e3 10Mb 5ms DropTail
$ns duplex-link $s41 $e4 10Mb 5ms DropTail
$ns duplex-link $s42 $e4 10Mb 5ms DropTail
$ns duplex-link $s51 $e5 10Mb 5ms DropTail
$ns duplex-link $s52 $e5 10Mb 5ms DropTail

$ns simplex-link $e1 $c1 10Mb 5ms dsRED/edge
$ns simplex-link $e2 $c1 10Mb 5ms dsRED/edge
$ns simplex-link $e3 $c1 10Mb 5ms dsRED/edge

```

```

$ns simplex-link $e4 $c1 10Mb 5ms dsRED/edge
$ns simplex-link $e5 $c1 10Mb 5ms dsRED/edge

$ns simplex-link $c1 $e1 10Mb 5ms dsRED/core
$ns simplex-link $c1 $e2 10Mb 5ms dsRED/core
$ns simplex-link $c1 $e3 10Mb 5ms dsRED/core
$ns simplex-link $c1 $e4 10Mb 5ms dsRED/core
$ns simplex-link $c1 $e5 10Mb 5ms dsRED/core

$ns simplex-link $c1 $c2 2Mb 50ms dsRED/core
$ns simplex-link $c2 $c1 2Mb 50ms dsRED/core

$ns simplex-link $c2 $dest 10Mb 5ms dsRED/core
$ns simplex-link $dest $c2 10Mb 5ms dsRED/edge

# Set DS RED parameters from Edge1 to Core1:
set qE1C1 [[$ns link $e1 $c1] queue]
$qE1C1 meanPktSize 1000
$qE1C1 set numQueues_ 1
$qE1C1 setNumPrec 2
$qE1C1 addPolicyEntry [$s11 id] [$dest id] TokenBucket 10 $cir11 $cbs11
$qE1C1 addPolicyEntry [$s12 id] [$dest id] TokenBucket 10 $cir12 $cbs12
$qE1C1 addPolicerEntry TokenBucket 10 11
$qE1C1 addPHBEntry 10 0 0
$qE1C1 addPHBEntry 11 0 1
$qE1C1 configQ 0 0 10 40 0.02
$qE1C1 configQ 0 1 0 0 1

# Set DS RED parameters from Core1 to Edge1:
set qC1E1 [[$ns link $c1 $e1] queue]
$qC1E1 meanPktSize 1000
$qC1E1 set numQueues_ 1
$qC1E1 setNumPrec 2
$qC1E1 addPHBEntry 10 0 0
$qC1E1 addPHBEntry 11 0 1
$qC1E1 configQ 0 0 10 40 0.02
$qC1E1 configQ 0 1 0 0 1

# Set DS RED parameters from Edge2 to Core1:
set qE2C1 [[$ns link $e2 $c1] queue]
$qE2C1 meanPktSize 1000
$qE2C1 set numQueues_ 1
$qE2C1 setNumPrec 2
$qE2C1 addPolicyEntry [$s21 id] [$dest id] TokenBucket 10 $cir21 $cbs21
$qE2C1 addPolicyEntry [$s22 id] [$dest id] TokenBucket 10 $cir22 $cbs22
$qE2C1 addPolicerEntry TokenBucket 10 11
$qE2C1 addPHBEntry 10 0 0
$qE2C1 addPHBEntry 11 0 1
$qE2C1 configQ 0 0 10 40 0.02
$qE2C1 configQ 0 1 0 0 1

```

```

# Set DS RED parameters from Core1 to Edge2:
set qC1E2 [[${ns} link $c1 $e2] queue]
$qC1E2 meanPktSize 1000
$qC1E2 set numQueues_ 1
$qC1E2 setNumPrec 2
$qC1E2 addPHBEntry 10 0 0
$qC1E2 addPHBEntry 11 0 1
$qC1E2 configQ 0 0 10 40 0.02
$qC1E2 configQ 0 1 0 0 1

# Set DS RED parameters from Edge3 to Core1:
set qE3C1 [[${ns} link $e3 $c1] queue]
$qE3C1 meanPktSize 1000
$qE3C1 set numQueues_ 1
$qE3C1 setNumPrec 2
$qE3C1 addPolicyEntry [$s31 id] [$dest id] TokenBucket 10 $cir31 $cbs31
$qE3C1 addPolicyEntry [$s32 id] [$dest id] TokenBucket 10 $cir32 $cbs32
$qE3C1 addPolicerEntry TokenBucket 10 11
$qE3C1 addPHBEntry 10 0 0
$qE3C1 addPHBEntry 11 0 1
$qE3C1 configQ 0 0 10 40 0.02
$qE3C1 configQ 0 1 0 0 1

# Set DS RED parameters from Core1 to Edge3:
set qC1E3 [[${ns} link $c1 $e3] queue]
$qC1E3 meanPktSize 1000
$qC1E3 set numQueues_ 1
$qC1E3 setNumPrec 2
$qC1E3 addPHBEntry 10 0 0
$qC1E3 addPHBEntry 11 0 1
$qC1E3 configQ 0 0 10 40 0.02
$qC1E3 configQ 0 1 0 0 1

# Set DS RED parameters from Edge4 to Core1:
set qE4C1 [[${ns} link $e4 $c1] queue]
$qE4C1 meanPktSize 1000
$qE4C1 set numQueues_ 1
$qE4C1 setNumPrec 2
$qE4C1 addPolicyEntry [$s41 id] [$dest id] TokenBucket 10 $cir41 $cbs41
$qE4C1 addPolicyEntry [$s42 id] [$dest id] TokenBucket 10 $cir42 $cbs42
$qE4C1 addPolicerEntry TokenBucket 10 11
$qE4C1 addPHBEntry 10 0 0
$qE4C1 addPHBEntry 11 0 1
$qE4C1 configQ 0 0 10 40 0.02
$qE4C1 configQ 0 1 0 0 1

# Set DS RED parameters from Core4 to Edge1:
set qC1E4 [[${ns} link $c1 $e4] queue]

```

```

$qC1E4 meanPktSize 1000
$qC1E4 set numQueues_ 1
$qC1E4 setNumPrec 2
$qC1E4 addPHBEntry 10 0 0
$qC1E4 addPHBEntry 11 0 1
$qC1E4 configQ 0 0 10 40 0.02
$qC1E4 configQ 0 1 0 0 1

# Set DS RED parameters from Edge5 to Core:
set qE5C1 [[$ns link $e5 $c1] queue]
$qE5C1 meanPktSize 1000
$qE5C1 set numQueues_ 1
$qE5C1 setNumPrec 2
$qE5C1 addPolicyEntry [$s51 id] [$dest id] TokenBucket 10 $cir51 $cbs51
$qE5C1 addPolicyEntry [$s52 id] [$dest id] TokenBucket 10 $cir52 $cbs52
$qE5C1 addPolicerEntry TokenBucket 10 11
$qE5C1 addPHBEntry 10 0 0
$qE5C1 addPHBEntry 11 0 1
$qE5C1 configQ 0 0 10 40 0.02
$qE5C1 configQ 0 1 0 0 1

# Set DS RED parameters from Core1 to Edge5:
set qC1E5 [[$ns link $c1 $e5] queue]
$qC1E5 meanPktSize 1000
$qC1E5 set numQueues_ 1
$qC1E5 setNumPrec 2
$qC1E5 addPHBEntry 10 0 0
$qC1E5 addPHBEntry 11 0 1
$qC1E5 configQ 0 0 10 40 0.02
$qC1E5 configQ 0 1 0 0 1

# Set DS RED parameters from Core1 to Core2:
set qC1C2 [[$ns link $c1 $c2] queue]
$qC1C2 meanPktSize 1000
$qC1C2 set numQueues_ 1
$qC1C2 setNumPrec 2
$qC1C2 addPHBEntry 10 0 0
$qC1C2 addPHBEntry 11 0 1
$qC1C2 configQ 0 0 10 40 0.02
$qC1C2 configQ 0 1 0 0 1

# Set DS RED parameters from Core2 to Core1:
set qC2C1 [[$ns link $c2 $c1] queue]
$qC2C1 meanPktSize 1000
$qC2C1 set numQueues_ 1
$qC2C1 setNumPrec 2
$qC2C1 addPHBEntry 10 0 0
$qC2C1 addPHBEntry 11 0 1
$qC2C1 configQ 0 0 10 40 0.02

```

```
$qC2C1 configQ 0 1 0 0 1
```

```
# Set DS RED parameters from Dest to Core2:
```

```
set qDC2 [[ $ns link $dest $c2] queue]
```

```
$qDC2 meanPktSize 1000
```

```
$qDC2 set numQueues_ 1
```

```
$qDC2 setNumPrec 2
```

```
$qDC2 addPolicyEntry [$dest id] [$s11 id] TokenBucket 10 $cir11 $cbs11
```

```
$qDC2 addPolicyEntry [$dest id] [$s12 id] TokenBucket 10 $cir12 $cbs12
```

```
$qDC2 addPolicyEntry [$dest id] [$s21 id] TokenBucket 10 $cir21 $cbs21
```

```
$qDC2 addPolicyEntry [$dest id] [$s22 id] TokenBucket 10 $cir22 $cbs22
```

```
$qDC2 addPolicyEntry [$dest id] [$s31 id] TokenBucket 10 $cir31 $cbs31
```

```
$qDC2 addPolicyEntry [$dest id] [$s32 id] TokenBucket 10 $cir32 $cbs32
```

```
$qDC2 addPolicyEntry [$dest id] [$s41 id] TokenBucket 10 $cir41 $cbs41
```

```
$qDC2 addPolicyEntry [$dest id] [$s42 id] TokenBucket 10 $cir42 $cbs42
```

```
$qDC2 addPolicyEntry [$dest id] [$s51 id] TokenBucket 10 $cir51 $cbs51
```

```
$qDC2 addPolicyEntry [$dest id] [$s52 id] TokenBucket 10 $cir52 $cbs52
```

```
$qDC2 addPolicerEntry TokenBucket 10 11
```

```
$qDC2 addPHBEntry 10 0 0
```

```
$qDC2 addPHBEntry 11 0 1
```

```
$qDC2 configQ 0 0 10 40 0.02
```

```
$qDC2 configQ 0 1 0 0 1
```

```
# Set DS RED parameters from Core2 to Dest:
```

```
set qC2D [[ $ns link $c2 $dest] queue]
```

```
$qC2D meanPktSize 1000
```

```
$qC2D set numQueues_ 1
```

```
$qC2D setNumPrec 2
```

```
$qC2D addPHBEntry 10 0 0
```

```
$qC2D addPHBEntry 11 0 1
```

```
$qC2D configQ 0 0 10 40 0.02
```

```
$qC2D configQ 0 1 0 0 1
```

```
# Set up one FTP connection between each source and the destination:
```

```
set udp11 [new Agent/UDP]
```

```
$ns attach-agent $s11 $udp11
```

```
set cbr11 [new Application/Traffic/CBR]
```

```
$cbr11 attach-agent $udp11
```

```
$cbr11 set packet_size_ $UDP_Packet_Size
```

```
$udp11 set packetSize_ $UDP_Packet_Size
```

```
$cbr11 set rate_ $rate11
```

```
set null11 [new Agent/Null]
```

```
$ns attach-agent $dest $null11
```

```
$ns connect $udp11 $null11
```

```
set udp21 [new Agent/UDP]
```

```
$ns attach-agent $s21 $udp21
```

```
set cbr21 [new Application/Traffic/CBR]
```

```
$cbr21 attach-agent $udp21
$cbr21 set packet_size_ $UDP_Packet_Size
$udp21 set packetSize_ $UDP_Packet_Size
$cbr21 set rate_ $rate21
set null21 [new Agent/Null]
$ns attach-agent $dest $null21
$ns connect $udp21 $null21

set udp31 [new Agent/UDP]
$ns attach-agent $s31 $udp31
set cbr31 [new Application/Traffic/CBR]
$cbr31 attach-agent $udp31
$cbr31 set packet_size_ $UDP_Packet_Size
$udp31 set packetSize_ $UDP_Packet_Size
$cbr31 set rate_ $rate31
set null31 [new Agent/Null]
$ns attach-agent $dest $null31
$ns connect $udp31 $null31

set udp41 [new Agent/UDP]
$ns attach-agent $s41 $udp41
set cbr41 [new Application/Traffic/CBR]
$cbr41 attach-agent $udp41
$cbr41 set packet_size_ $UDP_Packet_Size
$udp41 set packetSize_ $UDP_Packet_Size
$cbr41 set rate_ $rate41
set null41 [new Agent/Null]
$ns attach-agent $dest $null41
$ns connect $udp41 $null41

set udp51 [new Agent/UDP]
$ns attach-agent $s51 $udp51
set cbr51 [new Application/Traffic/CBR]
$cbr51 attach-agent $udp51
$cbr51 set packet_size_ $UDP_Packet_Size
$udp51 set packetSize_ $UDP_Packet_Size
$cbr51 set rate_ $rate51
set null51 [new Agent/Null]
$ns attach-agent $dest $null51
$ns connect $udp51 $null51

set tcp12 [new Agent/TCP]
$ns attach-agent $s12 $tcp12
set ftp12 [new Application/FTP]
$ftp12 attach-agent $tcp12
$tcp12 set packetSize_ $TCP_Packet_Size
set sink12 [new Agent/TCPSink]
$ns attach-agent $dest $sink12
$ns connect $tcp12 $sink12

set tcp22 [new Agent/TCP]
$ns attach-agent $s22 $tcp22
```

```
set ftp22 [new Application/FTP]
$ftp22 attach-agent $tcp22
$tcp22 set packetSize_ $TCP_Packet_Size
set sink22 [new Agent/TCPSink]
$ns attach-agent $dest $sink22
$ns connect $tcp22 $sink22

set tcp32 [new Agent/TCP]
$ns attach-agent $s32 $tcp32
set ftp32 [new Application/FTP]
$ftp32 attach-agent $tcp32
$tcp32 set packetSize_ $TCP_Packet_Size
set sink32 [new Agent/TCPSink]
$ns attach-agent $dest $sink32
$ns connect $tcp32 $sink32

set tcp42 [new Agent/TCP]
$ns attach-agent $s42 $tcp42
set ftp42 [new Application/FTP]
$ftp42 attach-agent $tcp42
$tcp42 set packetSize_ $TCP_Packet_Size
set sink42 [new Agent/TCPSink]
$ns attach-agent $dest $sink42
$ns connect $tcp42 $sink42

set tcp52 [new Agent/TCP]
$ns attach-agent $s52 $tcp52
set ftp52 [new Application/FTP]
$ftp52 attach-agent $tcp52
$tcp52 set packetSize_ $TCP_Packet_Size
set sink52 [new Agent/TCPSink]
$ns attach-agent $dest $sink52
$ns connect $tcp52 $sink52

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 == "17"))        print $2, $9, $6
        out.all > out.rec

    exit 0
```

```
$qE1C1 printPolicyTable
$qE1C1 printPolicerTable
$qE1C1 printPHBTable

$ns at 0.0 "$cbr11 start"
$ns at 0.0 "$cbr21 start"
$ns at 0.0 "$cbr31 start"
$ns at 0.0 "$cbr41 start"
$ns at 0.0 "$cbr51 start"
$ns at 0.0 "$ftp12 start"
$ns at 0.0 "$ftp22 start"
$ns at 0.0 "$ftp32 start"
$ns at 0.0 "$ftp42 start"
$ns at 0.0 "$ftp52 start"
$ns at 10.0 "$qE1C1 printStats"
$ns at 20.0 "$qE1C1 printStats"
$ns at $testTime "$cbr11 stop"
$ns at $testTime "$cbr21 stop"
$ns at $testTime "$cbr31 stop"
$ns at $testTime "$cbr41 stop"
$ns at $testTime "$cbr51 stop"
$ns at $testTime "$ftp12 stop"
$ns at $testTime "$ftp22 stop"
$ns at $testTime "$ftp32 stop"
$ns at $testTime "$ftp42 stop"
$ns at $testTime "$ftp52 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 successful simulation of particular scenario. Therefore topology is so called base requirement and components of topology are defined in base level of *ns*. This is easily detectable with the declarations which start by *\$ns*.

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.

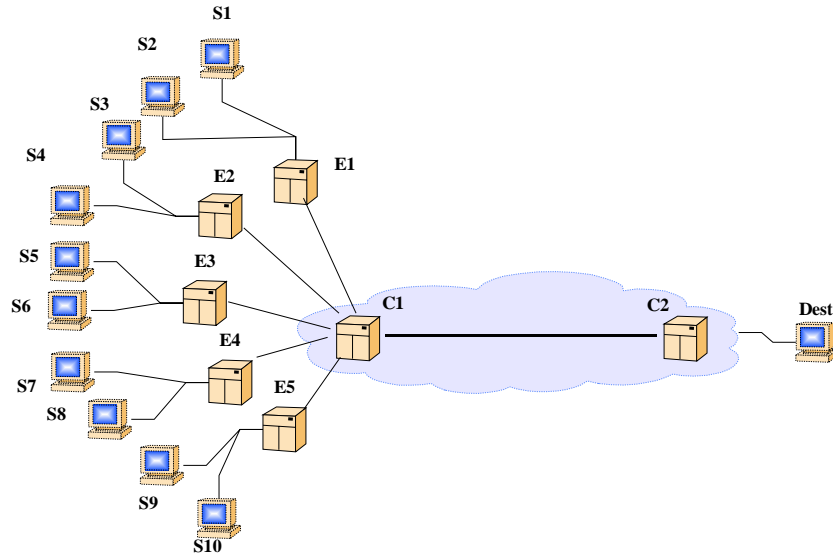


Figure 2.1: Topology of simulation

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

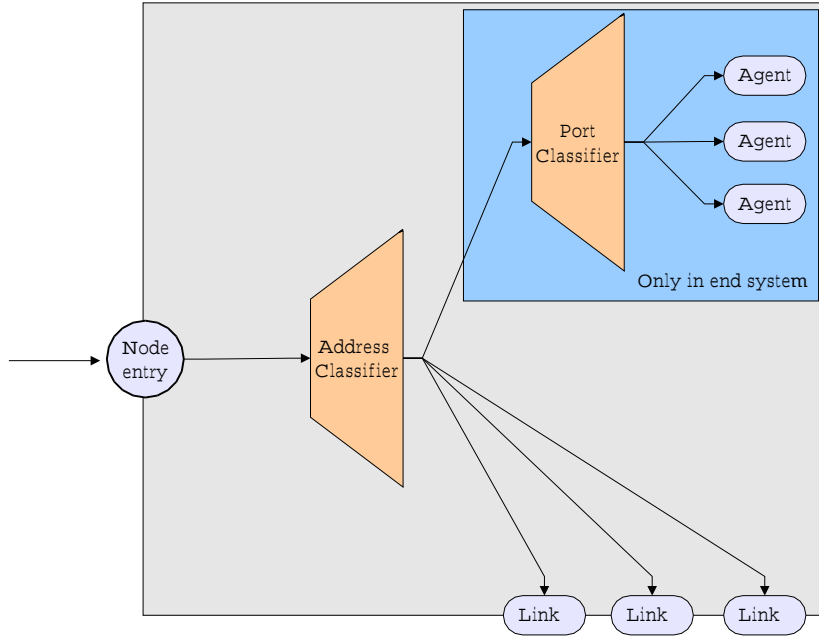


Figure 2.2: Construction of node in ns2

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.

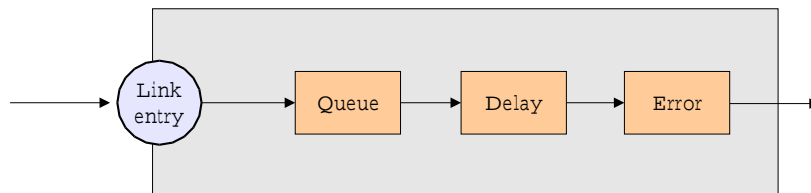


Figure 2.3: Construction of link in ns2

Links are created with command


```
--><--  
$ns duplex-link $s1 $e1 10Mb 5ms DropTail  
$ns simplex-link $e1 $e2 2Mb 50ms dsRED/edge  
--><--
```

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 Rate control mechanisms

Rate control mechanisms are implemented in Differentiated Services packet. Forwarding 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 `$s10`. 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. After the type of policer a initial codepoint (20) for the packet is given. This codepoint is degraded based on the metered result. Depending on the type of policer there may be several parameters controlling the rate and/or burst size. In TSW2CM's case there is only committed information rate (cir).

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 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 additional 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.

Once the policy is defined. Policers which are attached to policy need to be parametrized. Parameters which they require are degraded 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 precedence which some codepoint reflects `$qE1E2 addPHBEntry 10 0 0`. RED algorithms operating on the queues need also to be parametrised through `$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'), enqueue ('+') or deque ('-').

Second column is the simulated time when the event occurred. 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.

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

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

Ninth 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.

Twelfth column is the unique id of packet. Each packet has unique id throughout 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 necessary) and to plot trace information (not necessary). 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.

```
--><--  
$ns at 0.0 "$cbr11 start"  
$ns at 0.0 "$cbr21 start"  
$ns at 0.0 "$ftp12 start"  
$ns at 0.0 "$ftp22 start"  
$ns at $testTime "$cbr11 stop"  
$ns at $testTime "$cbr21 stop"  
$ns at $testTime "$ftp12 stop"  
$ns at $testTime "$ftp22 stop"  
--><--
```

Chapter 3

Exercise

This exercise is about comparison of different rate control methods in Internet. Methods which are used here are token bucket and time sliding window two color marker. Investigate the operation of different metering and marking algorithms in a network which is used for transmission of mixed (5 UDP clients and 5 TCP clients) traffic.

Calculate how these rate control mechanisms distribute capacity to individual connections in otherwise same network conditions. See how excess capacity is distributed to the clients (proportionally to cir / rate / something else). Use loose and strict policing (i.e. measured excess traffic is admitted to lower priority or measured excess traffic is discarded). See how same parameters effect the overall operation of different client types. With TCP-traffic check the effect of different round-trip times to actual capacity. What causes these effects. Run overloaded and underloaded cases to see what happens if you overutilize your infrastructure.

Write report (3-5 pages) explaining your findings of these rate control algorithms. You can explain the operation of these algorithms and compare then based on their structure. Also explain the sensitivity of these algorithms to the case when type of client changes behind the access port. Is any of these mechanisms superior to others and if is on what you base your argument.