A Case Study of Simulating DiffServ in NS-2

(with a modified version of NS-2)

Written by: **Timo Viipuri**

Date: **26.11.2002**

Table of Contents

1 Simulation structure

1.1 Topology

The network topology used in these simulations is presented in figure 1.

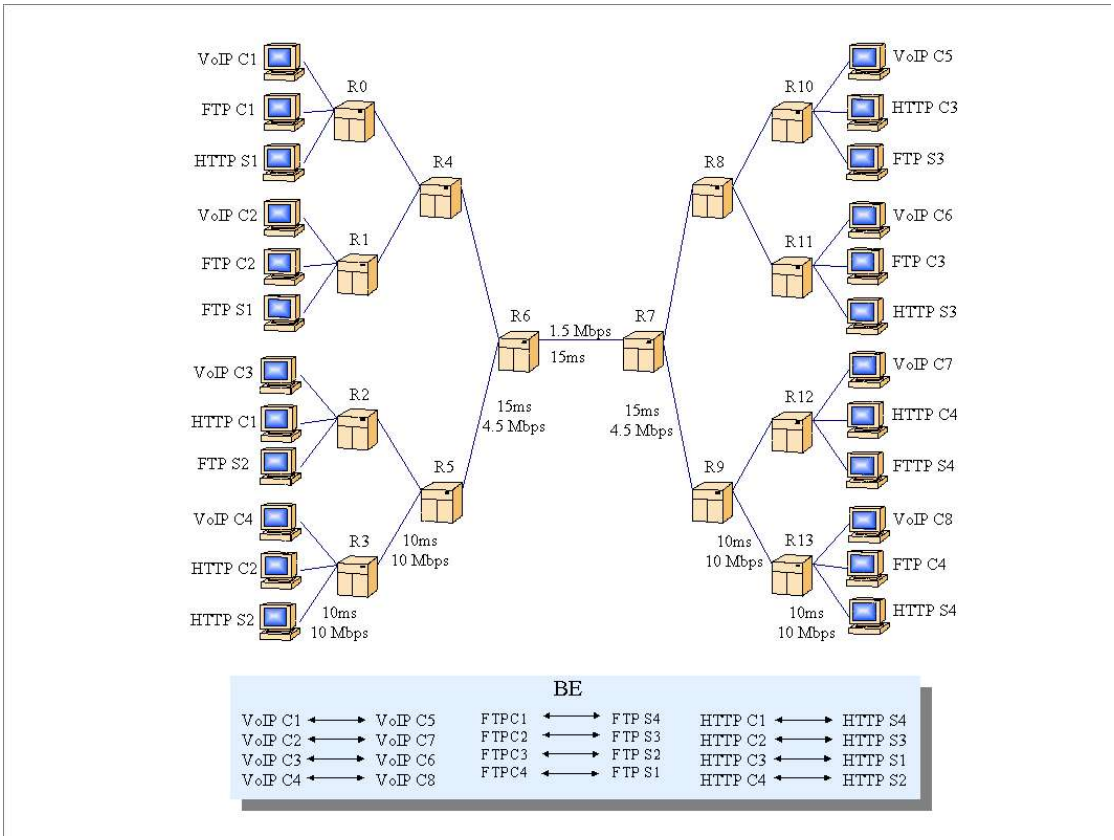

Illustration 1Network topology in this simulation

The bottle-neck link of the network is clearly the link between routers R6 and R7. All the connections traverse through the bottle-neck link.

1.2 Traffic generation

Three different types of traffic are generated in the network: VoIP, HTTP and FTP. They all have different characteristics and react differently to network congestion.

There are four connection pairs for each traffic type, as seen in figure 1. All connection pairs are located so that traffic must traverse through the bottle-neck link. Half of the servers are located on one side of the bottle-neck link and half on the other side. This produces on average even amount of traffic flowing from both sides of the bottle-neck link.

1.2.1 VoIP

VoIP-connections are based on UDP-agents. They offer a constant flow of packets and are not affected by loss of packets in terms of traffic generation, i.e. there are no retransmissions and no adjustment of packet send rate. Traffic generation is ON-OFF -type which means that packets are either sent at full rate or not at all.

Parameters describing VoIP-traffic are as follows:

- Burst time (seconds) - random variable with an exponential distribution
- Idle time (seconds) - random variable with an exponential distribution
- Burst send rate (bps) - fixed value
- Packet size (bytes) - fixed value

1.2.2 HTTP

HTTP-requests consist of three abstraction levels: sessions, pages and objects. A session contains one or more pages which all contain one or more objects.

Each object is transmitted in their own TCP-connection. An HTTP-object is typically quite small (few kB) and many TCP-connections are created in relation to amount of data transmitted. This causes a large portion of time and packets to be spent on connection establishment and tear-down. Short-lived connections also generally don't have enough time for TCP's rate-adjustment algorithm to reach an optimum packet sending rate.

Parameters for HTTP-traffic are:

- Session size (number of pages in a session)
- Session inter-arrival time (time between session requests in seconds)
- Page size (number of objects in a session in bytes)
- Page read time (time between page requests in a session in seconds)
- Object size (kB)

Parameter values are averages of random variables with an exponential distribution.

1.2.3 FTP

FTP-requests are similar to HTTP but with one abstraction level (page) missing. Requests come in sessions which all contain one or more objects (files). Objects are typically much larger than the ones used in HTTP. Objects in the same session are transmitted sequentially so that one object-transfer starts after the previous has been sent in its whole.

FTP-traffic generally receives more bandwidth than HTTP-traffic because of a larger object size, hence longer TCP-connections. In relation to transferred data-bytes in a single TCP-connection, less time and packets are spent on handshake and tear-down.

Also the rate-adjusting algorithm of the TCP-agent has more time to adjust the send rate to an optimum level in the network.

Parameters for FTP-traffic are:

- Session size (number of objects in a session)
- Session inter-arrival time (time between session requests)
- Object size (kB)

Again, all the parameters are mean values of random variables with an exponential distribution.

1.3 Differentiated services

DiffServ uses codepoints (DSCP) attached to a packet's IP-header to distinguish traffic with different PHBs (Per Hop Behaviour). A PHBs defines a forwarding treatment of a single packet in a router. They do not offer any guarantees regarding gained bandwidth, packet delay or jitter. It is only a means of defining better treatment for some class of traffic than to another.

Within a router, traffic is divided in different queues based on DSCPs. Two things have to be considered:

1. How to manage packets inside a single queue (or buffer)?
2. How to control scheduling between multiple queues, i.e. how link access is shared between queues?

1.3.1 Buffer management

Buffer size - usually defined in number of packets - is the first thing that has to be considered. Since buffers have a habbit of filling up something has to be done to eccess packets arriving in a queue. Two most used buffer types are drop-tail and RED.

1.3.1.1 Drop-tail

This is the most simple type of buffer. Dropping occurs only to packets arriving in a full buffer. This is a simple and light implementation but usually causes problems with multiple constant packet flows and so called global syncronization of TCP-connections.

1.3.1.2 RED

RED (Random Early Detection) drops packets at an increasing probability as the buffer begins filling up. Minimum and maximum thresholds of buffer occupation are defined and between them the probability of dropping a packet is increased linearly. Below the minimum threshold packets are not dropped at all and above the maximum threshold all arriving packets are dropped.

An avarage number of packets in the buffer is maintained. This value, rather than the

instantaneous value of the buffer length, is compared to the dropping thresholds when a packet arrives.

1.3.2 Queue management

When dealing with multiple buffers a scheduler must be included for distributing link access rights. Two different schedulers are used in this simulation: PQ (Priority Queueing) and WRR (Weighted Round Robin).

1.3.2.1 PQ

Priority queueing sets different priorities to different buffers. Link access is always granted to the buffer with the highest priority if it has packets to send. This type of scheduling has the advantage of offering very small packet delay and jitter. On the other hand high priority traffic can easily exhaust traffic with smaller priorities unless bandwidth limits are configured properly.

1.3.2.2 WRR

Weighted round robin shares link access circularly, one buffer at a time. Each buffer has a weight which assigns a relative share of link access time compared to other buffers.

With proper configuration WRR can offer similar type of preferential treatment to high priority traffic than PQ with the advantage of not completely exhausting classes with lower forwarding treatments.

1.3.3 Core routers

Core routers are routers which are only connected to other routers - not hosts. They only have the functionality of performing PHB on incoming packets. For this, core routers maintain multiple physical queues which, in addition, each have virtual queues or precedences. An incoming packet is placed in a virtual queue withing a physical queue based on the DSCP-value in the packet's IP-header.

A physical queue is implemented as a RED-queue in NS2. Virtual queues within the same physical queue all use the same buffer to temporarily store packets. However, all virtual queues can be configured with their own RED-parameters (minimum and maximum thresholds and dropping probability).

The following parameters must be defined for a core router:

- Number of physical and virtual queues
- All DSCPs needed
- Mapping of each DSCP to a physical and a virtual queue
- RED-parameters (min and max thresholds and dropping probability) for each virtual queue

1.3.4 Edge routers

Edge routers, or access routers, are used between the hosts and the core network. They perform the same functionality as core routers (PHB), and in addition manage the following functions:

- policing
- metering
- packet marking

When configuring edge routers the same parameters must be defined as for the core routers. In addition, the criteria of marking the packet with a DSCP must be included. The decision is two-fold:

1. Distinguish the packet of belongin to a certain traffic aggregate. The decision is based on the following IP-header values: source address, destination address and traffic type (or any combination of these).
2. Measure transmission rates of the given traffic aggregate and the conformance of it to pre-defined values

1.4 Statistics and monitoring

For monitoring and gathering of statistics, a number of different choices is mainteined. The approach is to trace events and monitor parts of the topology as the simulation progresses and write them to files. The actual simulation scale statistics is collected by post-processing the trace- and monitor-files with awk-scripts.

1.4.1 Session monitoring

Session monitoring is the most useful part of gathering information from these simulations. Three session monitoring files are maintained - one for each traffic type: HTTP, FTP and VoIP. The trace files are named as 'smon_<tType>.mon', where 'tType' is either 'http', 'ftp' or 'exp' (=VoIP).

1.4.1.1 HTTP and FTP

HTTP and FTP traffic both use the same format in session monitoring files. Files are named as 'smon_http.mon' and 'smon_ftp.mon' Two types of events are traced:

1. Information on object transfers
2. Times when a connection pair starts or stops communicating; used for determining the total transmission time for a connection pair

An object is a data element which is transferred in its own TCP-connection, for example a file in FTP or a picture in an HTTP-page. An object entry looks something like this:

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

```
36->31 obj 13 begin 6.8236 end 7.9719 dur 1.1483 objSize 4887 ..
  sentpack 7 ndatapack 4 ndatabytes 4887 nrexmitbytes 0 nrexmitpack 0 recBytes 4887
recPacks 7 avgDelay 0.110511
--->&lt;---
```

On the first line, the first column indicates the direction of the flow in the form: 'server'->'client'. Second is the object's identification number. Each object in a given traffic type (FTP, HTTP) has its own unique ID. Next there are the start and end times and the duration of the transfer. Connection handshake and tear-down are included in these times. The next one is the object size in bytes.

On the second line, the first column is the number of packets sent in total (in the direction of server->client). Next one is the number of sent packets which contain object data and the number of data bytes included in these packets. 'nrexmitbytes' and 'nrexmitpack' tell the number of retransmitted bytes and packets. Next there are the number of bytes and packets received by the client. Number of bytes received should be the same as the objSize - unless the connection was closed prematurely. A general approximation is that recPack ≈ sentPack-nrexmitpack. Finally there is the avarage packet delay.

1.4.1.2 VoIP

Session related information for VoIP-clients is written on its own file (smon_exp.mon). The events monitored are the beginning and the end of a VoIP-connection. Start-event is of the form:

```
--->&lt;---
56.761185 VOIP 3 starting, size 3990 pkts -> estTxTime 91.200000
--->&lt;---
```

where the first column is the time of the event. The next one is the identification number for the VoIP-peers. The ID remains static throughout the simulation for the given VoIP-peers. Next there are the planned size and duration of the given session.

The end-event is of the form:

```
--->&lt;---
147.961185 VOIP 3 stopped, sent 3990 pkts, txTime 91.200000
--->&lt;---
```

Sent packets and transmission time are the realized values in the simulation. They are the same as the ones scheduled in the corresponding start-event if the simulation time hasn't expired during the transmission.

1.4.2 Tracing

Tracing of packet arrivals is the most basic method of collecting simulation information. Tracing can be set on individual links or for all links in the network. The problem - especially in tracing all links - is a considerable performance penalty and large amounts of output data. Storing and post-processing of the data is often problematic on large-scale simulations.

The trace-file format is presented in the NS2-manual
(http://www.isi.edu/nsnam/ns/doc/node243.html)

1.4.3 Queue monitoring

Queue monitors take samples of a queue between two adjacent nodes at given times. In this simulation-script queue monitors are mainly used for measuring throughput for VoIP-connections by monitoring the access links. The format of the queue monitor output can be configured in the simulation scripts. Currently a sample is of the form:

```
--><---
50 6->7 size 12020  pkts 14 parrivals 9330 barrivals 7477369 pdrops 22 bdrops 16520
--><---
```

where the first column indicates the sample time in simulated seconds. Next there are the source and target nodes for the monitored queue. The size of the queue at the sample time is stated next both in bytes and in packets. 'parrivals' and 'barrivals' tell the total number of packets and bytes that have traversed through this queue from the beginning of the simulation. The last two figures are the total number of dropped packets and bytes.

1.4.4 Analysis

A convenient way of gathering statistics from the various monitoring-files is needed before analysing the results. Two awk-scripts (tp_ftp_http.awk & tp_exp.awk) and a command line script (parse_stats.scr) for controlling them are implemented. Scripts are launched with a shell command:

```
source stats_ext.scr   OR
source stats.scr   (for a more compact list of statistics)
```

The result is a list of statistics for every connection pair in the simulation. The output format is similar to the output for single objects explained in part 1.4.1.1. Sample statistics for each traffic type is listed below:

```
--><---
ftpS1 -> ftpC4
  duration 67.3349 sentDataPack 2894 sentPack 2912 recPack 2844 rxmitPack 85
packetLoss 0.0233516  avgDelay 0.200511..
  .. sentBytes 4454742 recBytes 4324522 rxmitbytes 130900 throughPut 513793
connections 6 pageTOs 0
--><---

--><---
httpS1 -> httpC3
  duration 53.9694 sentDataPack 281 sentPack 595 recPack 589 rxmitPack 8 packetLoss
0.010084  avgDelay 0.105093 ..
  .. sentBytes 361500 recBytes 357627 rxmitbytes 3951 throughPut 53011.8 connections
103 pageTOs 0
--><---
```

```
--><---
voip: 1 -> 5
txTime 10.8705 pSent 476 pTxmitted 465 pLoss 0.0231092 avgDelay 0.091206 sendRate
70061.3 throughPut 68442.2
--><---
```

1.4.5 Additional information

Optional information can also be collected from the simulation. Here is a glance at these commands. Note! This chapter is only meant as a reference guide to some extra information and is not needed for normal operation.

1.4.5.1 Performance analysis:

There is a comparison between running time and simulated time in a file named 'performance.mon'. It tells the runtime of the simulation every 100 simulated seconds. In addition, the total number of open connections is printed here. Graphical presentation can be seen with commands:

```
xgraph performance.mon
awk ' {print $1,$2-prev; prev = $2} ' performance.mon |xgraph
```

1.4.5.2 Queue length in the bottleneck link

Both instantaneous and average queue lengths are monitored in the bottleneck link. Current sample interval is set to 0.2s but it can be changed by modifying the parameter 'q_interval' in the simulation scripts (monitoring.tcl). Trace-file has the following format:

```
--><---
22.60 queue number 2: length 8.000000
--><---
```

where the first column is the sample time. Queue number tells which physical queue is being sampled. Length is the number of packets (sample or average) in the queue.

Graphical representation is seen, for example by using the following awk-command:

```
awk ' {if ($4 == "2:") {print $1,$6;}} ' q_r6r7_avg.mon | xgraph,
```

where the parameter in the if-statement is the physical queue number.

2 Code walk-through

This chapter covers most the code line by line explaining its format and purpose. However, redundancy is avoided by removing lines which are similar and only have different parameters. Also, basic knowledge of TCL-language is expected. A TCL-tutorial can be found in http://hegel.ittc.ukans.edu/topics/tcltk/tutorial-noplugin

2.1 General structure

The code is divided in five files on the basis of what functions they perform:

- Diffnet.tcl - The main-script which controls the simulation.
- Topology.tcl - Creates the nodes and links for the simulated network.
- Peer_setup.tcl - Contains procedures for initiating the HTTP-, FTP and VoIP-connections.
- 2q2p.tcl - Procedures for setting up queues for core and edge routers.
- Monitoring.tcl - All tracing and monitoring procedures

2.2 Diffnet.tcl

2.2.1 Random number generator's seed value

```
$defaultRNG seed 1
```

Seed value is needed for initializing the (pseudo-)random number generator. A same seed value produces exactly the same sequence of random numbers. When measuring the variance between multiple simulation runs the seed number must be different for each run. Seed value 0 produces a new seed for every simulation run.

2.2.2 Packet tracing

```
$ns trace-all [open tr_diffnet.out w]
```

This activates tracing on all links of the network. Trace output is dumped in file 'tr_diffnet.out'

```
set tf [open "tr_bottleneck.out" w]
$ns trace-queue $n(r6) $n(r7) $tf
$ns trace-queue $n(r7) $n(r6) $tf
```

Activates tracing on a link between nodes n(r6) and n(r7) (=bottleneck link). Traces are taken on both directions of the link and printed on file 'tr_bottleneck.out'.

Note! See naming convention of nodes in chapter "2.3.1 Setting up nodes"

2.2.3 Including another script

```
source <filename.tcl>
```

where <filename.tcl> is the file to be included. Control is returned to the calling script after the file has been read.

2.2.4 Assigning traffic generation

2.2.4.1 HTTP

```
launchHttp httpC1 httpS4
```

A procedure call for 'launchHttp' which generates HTTP-traffic between nodes n(httpC1) and n(httpS4). Traffic is generated for the duration of the simulation. Parameters for the traffic are defined in the procedure itself.

The procedure call can also take lists of clients and servers as an argument. This doesn't affect the total amount of bytes to be transmitted - only the way traffic is distributed among hosts. The procedure call takes the form:

```
launchHttp "httpC1 httpC2" "httpS2 httpS3 httpS4"
```

Traffic is shared randomly among these servers and clients following two principles:

1. For every launched page, a server is picked randomly from the server list
2. For every launched object, a client is picked randomly from the client list

The procedure 'launchHttp' is implemented in file: 'peer_setup.tcl'.

2.2.4.2 FTP

```
launchFtp ftpC1 ftpS4
```

Exactly the same rules apply for determining the FTP-traffic generation as for HTTP described above.

2.2.4.3 VoIP

```
launchVoip voip1 voip5 0
```

Creates VoIP-traffic for the duration of the simulation. Parameters are defined in the launchVoip-procedure. Difference to 'launchHttp' and 'launchFtp' is that a third parameter - ID - has to be included. ID is a unique identification number for each VoIP-connection pair.

LaunchHttp, launchFtp and launchVoip are described more thoroughly in chapters 2.4.1 and 2.4.2.

2.2.5 Setting up DiffServ-queues

There are three types of queues defined in this topology:

- Core queues: A DiffServ-capable queue type, which will handle traffic congestion in the core network. Congestion will only happen in the bottle-neck link in this topology so only it has to be configured as a core queue.
- Edge queues: A DiffServ-capable queue type, which will handle policing, packet marking, etc. All queues between a host and its access router in the direction of host->router are edge queues.
- Irrelevant queues: All other queues. They are never congested and do not have to perform differentiated PHB. Queue type is set as drop-tail because of its simple and light implementation.

2.2.5.1 Core queues

```
confDSCore r6 r7
```

A procedure call for 'confDSCore' which configures a queue between nodes n(r6) and n(r7). It configures the queue only in the direction of n(r6)->n(r7), so a separate call is needed for configuring the queue in direction of n(r7)->n(r6).

2.2.5.2 Edge queues

```
confDSEdges voip1 voip5 $cir_exp 29_app BE
```

Configures two one-way queues, namely the queue between host n(voip1) and its access router and also the queue between host n(voip5) and its access router. The direction of the configured queue is from host to the router.

The queues in the other direction (router->host) are simple drop-tail queues in this topology.

The procedures confDSCore and confDSEdge are explained in more detail in chapters 2.5.2 and 2.5.3.

2.2.6 Launching and shutting down the simulation

```
$ns at 100 "timeStats 100"
```

The procedure 'at' schedules the command in quotes to be executed after 100 seconds of simulated time. The procedure 'timeStats' is implemented in file 'monitoring.tcl' and prints the simulated time and runtime on the screen every 100 seconds. It also prints the number of open connections in each HTTP and FTP pagepool.

```
$ns at $testTime "close-connections"
$ns at [expr $testTime + 5] "kill-em-all"
$ns at [expr $testTime + 11] "finish"
```

The simulation shut-down sequence is three-fold.

1. At 'testTime' all open connections are sent the close-signal which activates the normal TCP-connection tear-down.
2. After 'testTime + 5' seconds any remaining connections are forced to close. This removes any hung connections from the system.
3. After 'testTime + 11' seconds all trace files are flushed and the simulation is closed.

```
$ns run
```

This is the last command in the startup script. It is executed after the setup of the simulation and tells the simulation time to begin running.

2.3 Topology.tcl

2.3.1 Setting up nodes

```
for {set i 0} {$i < $num_r} {incr i} {
   set n(r$i) [$ns node]
   set r_list "[set r_list] [list r$i]"
}
```

This sets up router nodes. A table of nodes named 'n' is created here which contains a number of 'num_r' nodes. TCL accepts strings as table indexes and here the table entries are: {n(r0), n(r1), ..., n(r$num_r)}. The variable 'r_list' is a list of strings and it contains the node indexes.

All other nodes are created the same way and added to the table 'n'. The nodes created are:

- Routers: {n(r0), n(r1), ..., n(r13)}
- VoIP-hosts: {n(voip1), n(voip2), ..., n(voip8)}
- FTP-clients: {n(ftpC1), n(ftpC2), n(ftpC3), n(ftpC4)}
- FTP-servers: {n(ftpS1), n(ftpS2), n(ftpS3), n(ftpS4)}
- HTTP-clients: {n(httpC1), n(httpC2), n(httpC3), n(httpC4)}
- HTTP-servers: {n(httpS1), n(httpS2), n(httpS3), n(httpS4)}

All the nodes are the same type of generic node-objects. The naming simply states what kind of traffic-agent will be placed in that node.

2.3.2 Setting up the links

```
$ns simplex-link $n(r6) $n(r7) 1.5Mb 15ms dsRED/core
```

This creates a simplex-link (=one-way) from node n(r6) to n(r7). The bandwidth of the link is 1.5 Mbps and the packet processing delay is 15 ms. The queue used in this link is a differentiated services type core queue using RED.

```
$ns simplex-link $n(voip1) $n(r0) 10.0Mb 10ms dsRED/edge
```

Similarly creates a DiffServ-capable edge link between a host and an access router.

```
$ns duplex-link $n(r4) $n(r0) 10.0Mb 10ms DropTail
```

Creates a duplex-link using a drop-tail queue. Duplex-link is simply two simplex-links in opposite directions.

2.4 Peer_setup.tcl

2.4.1 Defining HTTP and FTP-traffic

Traffic generation mechanisms are very similar with HTTP and FTP. The differing parameters are defined in procedures 'launchHttp' and 'launchFtp'. They both call the procedure 'launchSession' to perform the common operations.

2.4.1.1 Proc launchHttp

```
set sessionSize_  5      ;# Number of pages in a session
set interSession_ 5      ;# Time between session arrivals (seconds)
set pageSize_     3      ;# Number of objects in a page
set interPage_    12     ;# Time between page requests (seconds)
set objSize_      3300   ;# Object size (bytes)
set interObj_     0.1    ;# Time between object requests (seconds)
```

Define parameters describing HTTP-traffic generation. All parameters are mean values of an exponential distribution.

```
set sessionSize [new RandomVariable/Exponential]
$sessionSize set avg_ $sessionSize_
```

This creates an exponential random number generator named 'sessionSize'. The mean value of the random numbers distribution is set as 'sessionSize_'.

```
set pool [new PagePool/WebTraf]
```

Creates the pagepool-object that generates HTTP-traffic.

```
$ns set httpMonFile [open "smon_http.mon" w]
```

Opens a file (smon_http.mon) for outputing session monitoring events in HTTP-traffic.

```
launchSession $pool $clnt $svr $sessionSize $interSession_ $pageSize_ $interPage_
$objSize_ $interObj_
```

A procedure call for 'launchSession' which configures the rest of the traffic parameters which are common with FTP-traffic.

2.4.1.2 Proc launchFtp

Performs the same operations as 'launchHttp' only with differing parameters.

2.4.1.3 Proc launchSession

```
set numSession_   [expr round(1.4*$testTime/$interSession_)]
```

Sets the number of sessions created by this pagepool. The formula used is

$$\text{Number of sessions} = \frac{1.4 * \text{Total simulation time}}{\text{Time between session requests}}$$

The coefficient 1.4 is for creating eccess number of sessions to ensure that they do not run out. Any sessions launched after the simulation time has expired are simply ignored.

```
$pool set-num-client [llength $clnt]
$pool set-num-server [llength $svr]

set i 0
foreach s $clnt {
        $pool set-client $i $n($s)
        incr i
}

set i 0
foreach s $svr {
        $pool set-server $i $n($s)
        incr i
}
```

Add the given servers and clients to the pagepool's server and client pool.

```
$pool set-num-session $numSession
set launchTime [$interSession value]
for {set i 0} {$i < $numSession} {incr i} {
        ## Number of objects per page
        if {[$pool set application_] == 27 } {
           # FTP always has a page size of 1 -> a page represents a file
           set pageSize [new RandomVariable/Constant]
           $pageSize set val_ $pageSize_
        } else {
           set pageSize [new RandomVariable/Exponential]
           $pageSize set avg_ $pageSize_
        }

        set interPage [new RandomVariable/Exponential]
        $interPage set avg_ $interPage_

        set interObj [new RandomVariable/Exponential]
        $interObj set avg_ $interObj_
```

```
        set objSize [new RandomVariable/Exponential]
        $objSize set avg_ $objSize_

        if { $launchTime < $testTime } {
                $pool create-session $i $numPage $launchTime \
                        $interPage $pageSize $interObj $objSize
        }
        set launchTime [expr $launchTime + [$interSession value]]
}
```

This launches the sessions of the pagepool. New random number objects are created for each launched session.

2.4.2 Defining VoIP-traffic

```
set load 0.6
```

Load of the VoIP-client is determined by the formula:

$$load = \frac{\text{Mean burst time}}{\text{Mean inter-arrival time}} = \frac{\text{Mean burst time}}{(\text{Mean burst time}) + (\text{Mean idle time})} \quad ,$$

where burst time is the ON-time and idle time is the OFF-time of the ON-OFF-traffic generator.

```
set udp [new Agent/UDP]
set null [new Agent/Null]
$ns attach-agent $n($src) $udp
$ns attach-agent $n($dst) $null
$ns connect $udp $null
```

This sets up an UDP-connection between two nodes. In NS2, a UDP-agent is the server and a NULL-agent is the client of the connection. The procedure 'attach-agent' binds an agent to a node. 'connect' creates a logigal connection between the UDP- and the NULL-agent.

```
set expoo [new Application/Traffic/Exponential]
$expoo attach-agent $udp
$expoo set packetSize_  [expr 160+40]; # data + header
$expoo set burst_time_  180
$expoo set idle_time_   [format %.1f [expr [$expoo set burst_time_]*(1/$load - 1)]]
$expoo set rate_    70000
$expoo set id_          $id
```

This sets up an exponential ON-OFF-traffic generator (burst and idle times have exponential distributions) and assigns their parameters. The first two lines create the traffic-generating object and attach it to the UDP-agent created before. The parameter 'packetSize_' indicates the packet size, including the packet header and data. 'burst_time_' is the mean holding time of a single VoIP-call. 'idle_time_' is the mean duration of a non-active period between two consecutive calls. Usually only load and mean holding time are defined and idle time is derived from the above equation. Rate

is the total send rate (bps) during a call. 'id_' is a unique identification number for the VoIP connection pair.

2.5 2q2p.tcl

2.5.1 Defining PHBs (BE, AF, EF)

```
set EF(cp) 10
set AF(cp) 20
set BE(cp) 30
```

Sets the initial DSCP-values for PHBs. This value is given to a packet when the traffic conforms to the bandwidth values defined in the policy. A degraded value is given to a packet when the traffic is non-conformant. The degraded value is the initial codepoint of the class +1.

```
set EF(qw) 6
set AF(qw) 3
set BE(qw) 1
```

Assigns queue-weights for the PHBs. This is the ratio in which packets are forwarded. The scheduler is work-conserving which means that if some class has no packets to send its turn is handed over to the next class.

```
 # EF
set EF(in_min) 4.5
set EF(in_max) 8.0
set EF(in_prob) 0.05
set EF(out_min) 0.0   ;# Drop every packet that is out of profile
set EF(out_max) 0.0
set EF(out_prob) 1.00
set EF(qlimit) 10

 # AF
set AF(in_min) 30     ;# ~30% of the queue limit
set AF(in_max) 60     ;# ~60%
set AF(in_prob) 0.05
set AF(out_min) 15    ;# ~15%
set AF(out_max) 60    ;# ~60%
set AF(out_prob) 0.10
set AF(qlimit) 100

 # BE
set BE(in_min) 15     ;# ~15% of the queue limit
set BE(in_max) 60     ;# ~60%
set BE(in_prob) 0.05
set BE(out_min) 15     ;# No effect - BE uses only IN-precedence
set BE(out_max) 60
```

```
set BE(out_prob) 0.05
set BE(qlimit) 100
```

This sets up the RED-parameters for the PHBs. Separate parameters are given to IN- and OUT-precedences. Also the size of the RED-buffer is set here. A router has each of these PHBs defined, so the total amount of buffer space for a router is 100+100+10=210. However, if traffic sources use only one class, for example BE, then effectively a buffer space of only a 100 packets is used.

Note! BE-class has both IN- and OUT-precedences defined but the edge-router policers are configured so that only one precedence (IN) is used.

EF-class drops all packets that are marked out of profile. This way the rate of the EF-traffic can be controlled precisely.

2.5.2 Configuring a core link

```
$q($a$b) set numQueues_ 3
$q($a$b) setNumPrec 2
$q($a$b) meanPktSize 300
```

Sets the number of physical queues and precedences in the link. Mean packet size is an estimated average size of a packet traversing through the link.

```
$q($a$b) setSchedularMode WRR      ;# Options: RR, WRR, WIRR, PRI
$q($a$b) addQueueWeights 0 $EF(qw)
$q($a$b) addQueueWeights 1 $AF(qw)
$q($a$b) addQueueWeights 2 $BE(qw)
```

Selects a scheduler mode; options: RR (Round Robin), WRR (Weighted RR), WIRR (Weighted Interleaved RR), PRI (Priority queueing). Queue weights have to be configured for every physical queue when using WRR or WIRR. Queue weights defined in parameter tables EF, AF and BE are used here.

```
$q($a$b) addPHBEntry $EF(cp) 0 0
$q($a$b) addPHBEntry [expr $EF(cp)+1] 0 1
$q($a$b) configQ 0 0 $EF(in_min) $EF(in_max) $EF(in_prob)
$q($a$b) configQ 0 1 $EF(out_min) $EF(out_max) $EF(out_prob)
$q($a$b) setPhysQueueSize 0 $EF(qlimit)
```

Configures a single physical queue within the link. For example this sets the parameters for traffic in EF-class. The first two lines add two PBH-entries to the physical queue 0: codepoint in variable EF(cp) is mapped to precedence 0 and codepoint EF(cp)+1 is mapped to precedence 1.

Procedure 'configQ' sets RED-parameters for the precedences. For example the first one sets packets in precedence 0 of physical queue 0. The parameters are: minimum and maximum dropping thresholds and the maximum dropping probability.

The last line tells how many packets fit in the physical queue 0.

2.5.3 Configuring an edge link

The same parameters as for a core link also need to be configured for an edge link. In addition, a traffic limit for marking a packet IN or OUT of a profile has to be set.

Notice that in this topology there is only one host at the end of each edge link. This means that only one physical queue has to be configured per edge link to handle traffic of that host.

```
$q($qe) addPolicyEntry -1 -1 TSW2CM $cp_ $cir $app
$q($qe) addPolicerEntry TSW2CM $cp_ [expr $cp_ + 1]
```

The first line adds a policy entry which uses a TSW2CM (Time Sliding Window with 2-Color Marking) policer. The initial codepoint for the policy is $cp_ and the traffic limit for determining if a packet is IN or OUT of the profile is $cir. The link's queue manager uses three-level granularity to conclude which policy entry a packet belongs to: source and destination nodes and the application type. Here a value of -1 indicates that destination and target nodes are irrelevant and only the application type ($app) should be checked.

The second line adds a policer entry which indicates which codepoints should be used. If the traffic conforms to the levels agreed in $cir, packets are marked with codepoint $cp_. Excess traffic is marked with codepoint $cp_+1.

2.6 Monitoring.tcl

2.6.1 proc timeStats

Prints out the following information on the screen and on a file at given intervals:

- Simulation time and runtime
- The number of open connections in each WebTraf and FTPTraf pagepool

2.6.2 proc printDSQueueLength

This is a helper function for the next procedure (diffServStats). At a given interval prints out the sample and avarage lengths of a queue. The following is given as a parameter:

- A handle for the DiffServ queue or set of physical queues
- Physical queue number to be monitored from the set of queues
- Sample interval
- Output files where the sample and average queue lengths are printed on.

2.6.3 proc diffservStats

Print out various DiffServ-related information. The following operations are performed:

- PHB-table of the bottle-neck link as well as all the policy-tables for the applications are printed in the beginning of the simulation.
- Queue length monitors are initiated and the procedure 'printDSQueueLength' is called.
- Packet statistics for the bottle-neck link is printed on the screen at 50 second intervals.

2.6.4 proc crQmon

```
set qf [open "qmon_[set svr]_to_[set clnt].mon" w]
```

Opens an output file for the monitoring information. A handle for the file is 'qf' and the file name is for example 'qmon_voip1_to_voip2.mon'.

```
set qmon [$ns monitor-queue $n($svr) [$n($svr) neighbors] $qf]
set qmon2 [$ns monitor-queue [$n($clnt) neighbors] $n($clnt) $qf]
```

Creates two queue-monitors: one for the server-side and one for the client-side access link. This way the monitor can be used to check how many packets have been sent by the server and how many packets have been received by the client.

```
record $qmon $n($svr) [$n($svr) neighbors] $interval $qf
record $qmon2 [$n($clnt) neighbors] $n($clnt) $interval $qf
```

The procedure 'record' is used as a helper function to print out queue monitor information to a file every 'interval' seconds.

2.6.5 proc record

```
set time  [$ns now]           ;# current simulated time
set size [$qmon set size_]     ;# size of the queue in bytes
set pkts [$qmon set pkts_]     ;# size of the queue in packets
set parrivals [$qmon set parrivals_] ;# number of packets arrived in the queue
set barrivals [$qmon set barrivals_] ;# number of bytes arrived in the queue
set pdrops [$qmon set pdrops_]  ;# number of packets dropped
set bdrops [$qmon set bdrops_]  ;# number of bytes dropped
puts $file "$time [$src id]->[$dst id] size $size  pkts $pkts parrivals $parrivals barrivals
$barrivals pdrops $pdrops bdrops $bdrops"
```

Queue monitors contain a number of variables related to the state of the queue. Here, some of them are picked out and written on a file.

```
$ns at [expr $time+$interval] "record $qmon $src $dst $interval $file"
```

Recursively call 'record' every 'interval' seconds.