Equations for the STP Method and Details about its Setting in ns-2.

In the STP method the weight of each queue, of the system of queues, at every output interface of a node, is proportional to its relative average length. That is, comparing with the sum of the lengths of all the queues of the interface.

Contents.

1 The Updating of Weights.

The current weights of each queue are periodically updated as follows:

1- A at the output interface of a node there is a set of N queues, being attended by a scheduler using weights (like a WFQ scheduler), where the weights of the queues tend to be proportional to their relative queue lengths.

2- Time t_0 is the initial time. At this time the weights of the queues have all the same value (see Technical Report #1[1] of this group of technical reports for more information about the setting of initial weights).

3- After t_0 new weights are periodically calculated at intervals of size τ. The intervals are sufficiently big so as to be able to observe several packet arrivals and departures.

4- At time $(t_0 + (r + 1)\tau)^-$, where r is an integer and $r \geq 0$, a new calculation of weights is about to take place. The computing time is very small compared with the size of τ, thus it can be considered that the calculation is ended at time $t_0 + (r + 1)\tau$.

The symbol $\Delta\left(\phi_i^{Act}\left(t_0 + r\tau\right)\right)$ is the difference between the current values (valid from $t_0 + r\tau$ to $(t_0 + (r + 1)\tau)^-$) and the new ones, calculated at time $t_0 + (r + 1)\tau$, for $i = 1, ..., N$, as indicated by equation 1.

$$\Delta\left(\phi_i^{Act}\left(t_0 + r\tau\right)\right) = \phi_i^{New}\left(t_0 + \left(r+1\right)\tau\right) - \phi_i^{Act}\left(t_0 + r\tau\right), i = 1, ..., N. \quad \textbf{1}$$

5- A at the smallest time t' after $t_0 + (r + 1)\tau$, when the change of weights in use is feasible, in accordance with the operation of the scheduler [see Technical report #3[2] of this group of technical reports] the so-called "current weights" that were in use by the scheduler are updated with the recently computed ones as shown in equation 2.

$$\phi_i^{Act}\left(t'\right) \leftarrow \phi_i^{New}\left(t_0 + \left(r+1\right)\tau\right), t' \geq t_0 + \left(r+1\right)\tau, \quad i = 1, ..., N. \quad \textbf{2}$$

Since the difference between times $t_0 + (r + 1)\tau$ and t' should be very small compared to τ, in order to simplify the analysis of the method, this difference may be neglected, so equation 2 can be written as:

$$\phi_i^{Act}\left(t_0 + \left(r+1\right)\tau\right) \leftarrow \phi_i^{New}\left(t_0 + \left(r+1\right)\tau\right), i = 1, ..., N. \quad \textbf{3}$$

2 Computation of the New Weights.

At time $t_0 + (r + 1)\tau$ the method computes the indicator $I\Delta\left(\phi_i^{Act}\left(t_0 + r\tau\right)\right)$ for every queue i, to compare its current weight, $\phi_i^{Act}\left(t_0 + r\tau\right)$, (valid from time $t_0 + r\tau$ to time $(t_0 + (r + 1)\tau)^-$), with its current relative length, as indicated by equation 4.

$$I\Delta\left(\phi_i^{Act}\left(t_0 + r\tau\right)\right) = \frac{Q_i\left(t_0 + \left(r+1\right)\tau\right)}{\sum_{j=1}^{N} Q_j\left(t_0 + \left(r+1\right)\tau\right)} - \phi_i^{Act}\left(t_0 + r\tau\right) \quad \textbf{4}$$

[1] Rep-Tec-1-Delta-Functions-STP-Method.doc
[2] Rep-Tec-3-WFQ-Changing-Weights.doc.

Where $Q_i\left(t_0+\left(r+1\right)\tau\right)$ is the average length of the queue *i*, at $t_0 + (r + 1)\,\tau$. It is important to note that the sum of the weights must be equal to 1, that is $\sum_{i=1}^{N}\phi_i^{Act}\left(t_0+r\,\tau\right)=1$.

The method forms the sets *A* and *Z*, such that $Z \cap A = \varnothing$ and $Z \cup A = \{1, .., N\}$. If indicator $I\Delta > 0$ it designates queue *i* as a queue that has to gain weight, and so *i* is placed in set *A*; otherwise, *i* is placed in set *Z*, and queue *i* is designated as a queue that has to lose weight. The case where $I\Delta = 0$ has little probability, but for completeness it is assigned to set *A*.

The following paragraphs explain what the method does in every case.

Case when $I\Delta < 0$ for queue i (queue i is designated to lose weight).

With the above considerations, equation 1 can be rewritten as:

$$\Delta\left(\phi_i^{Act}\left(t_0+r\,\tau\right)\right)=\phi_i^{Act}\left(t_0+\left(r+1\right)\tau\right)-\phi_i^{Act}\left(t_0+r\,\tau\right) \qquad \mathbf{5}$$

The method indicates that the queue should lose a very small amount of weight at the ending of each interval of size τ. If queue *i* were designated to lose weight for T/τ consecutive times (where T/τ is a big integer like 100 or 1000), this loss should be such that queue *i* will have lost a (P·100) percentage of its weight. *P* is a positive constant parameter called the *loss factor*, which should be fairly small since the purpose of the method is to enforce a rather small total loss, in the order of 10% or 20%.

In accordance with this, for the purpose of convenience, *r* is to be interchanged with $n + l\ T/\tau$, where *l* and *n* are integers such that $l \geq 0$, and $0 \leq n < T/\tau$.

The negative increment (decrement) to be calculated at $t_0 + (r + 1)\,\tau = t_0 + (n + l\,K + 1)\,\tau$, employs the following equation:

$$\begin{aligned}\Delta\left(\phi_i^{Act}\left(t_0+r\,\tau\right)\right)&=-f\left(P\right)\frac{\tau}{T}\phi_i^{Act}\left(t_0+r\,\tau\right)<0\\ \Delta\left(\phi_i^{Act}\left(t_0+n\,\tau+l\,T\right)\right)&=-f\left(P\right)\frac{\tau}{T}\phi_i^{Act}\left(t_0+n\,\tau+l\,T\right)<0\end{aligned} \qquad \mathbf{6}$$

Note that as equations 5 (equivalent to 6) is calculated at time $t_0 + n\,\tau + lT + \tau$. All the calculations for a time interval of duration *T* are made from time ($t_0 + lT + \tau$), when $n = 0$, to time ($t_0 + (l + 1)T$), when $n = T/\tau - 1$ (Technical Report #1 of this group of technical reports gives an insight of why the form of equation 6 is chosen – see footnote 1 in page 2 in this document).

The function $f(P)$ is a straightforward function of *P*, as explained in the following paragraphs. Equation 6 suggests that the decrement of the selected queues is very small, since $\tau << T$.

Consider that queue *i* obtains T/τ consecutive results indicating that it has to lose weight, starting with calculation at time $t_0 + l\ T + \tau$ (when $n = 0$) and ending with calculation at $t_0 + (l + 1)T$ (when $n = T/\tau - 1$), then equation 7 holds.

$$\phi_i^{Act}\left(t_0+\left(l+1\right)T\right)-\phi_i^{Act}\left(t_0+l\,T\right)\approx\phi_i^{Act}\left(t_0+l\,T\right)\left(e^{-f(P)}-1\right) \qquad \mathbf{7}$$

Which is equivalent to:

$$\phi_i^{Act}\left(t_0+(l+1)T\right)/\phi_i^{Act}\left(t_0+lT\right)\approx e^{-f(P)} \quad \textbf{8}$$

For the proof of equation 7, see **Theorem 1**.

With regard to $e^{-f(P)}$ in equation 8, it is known that:

$$e^{-f(P)}=\sum_{j=0}^{\infty}\frac{\left(-f(P)\right)^j}{j!}=1+\frac{\left(-f(P)\right)^1}{1!}+\frac{\left(-f(P)\right)^2}{2!}+\frac{\left(-f(P)\right)^3}{3!}+\ldots \quad \textbf{9}$$

If $f(P)$ were relatively small, then $e^{-f(P)}$ in equation 9 could be approximated with $1-f(P)$ and equation 8 would become:

$$\begin{gathered}\phi_i^{Act}\left(t_0+(l+1)T\right)/\phi_i^{Act}\left(t_0+lT\right)\approx 1-f(P)\\ \phi_i^{Act}\left(t_0+(l+1)T\right)-\phi_i^{Act}\left(t_0+lT\right)\approx -f(P)\phi_i^{Act}\left(t_0+lT\right)\end{gathered} \quad \textbf{10}$$

Equation 10 shows that $f(P)$ is the loss percentage of the weight value of queue i. For example, if $f(P) = 0.2$, then $\phi_i^{Act}\left(t_0+(l+1)T\right)/\phi_i^{Act}\left(t_0+lT\right)\approx 1-f(P)=0.8$.

This means that ϕ_i^{Act} would have lost 20% of its value from time lT to time $(l+1)T$.

Since $f(P)$ would not always be enough small, another form to build $f(P)$ is forcing $e^{-f(P)}$ to be equal to $1-P$, that is, $f(P) = -\ln(1-P)$. Substituting into equation 7:

$$\phi_i^{Act}\left(t_0+(l+1)T\right)-\phi_i^{Act}\left(t_0+lT\right)=\phi_i^{Act}\left(t_0+lT\right)\left(e^{\ln(1-P)}-1\right)=-P\phi_i^{Act}\left(t_0+lT\right)$$

The manipulation of the above equation results in equation 11.

$$\phi_i^{Act}\left(t_0+(l+1)T\right)/\phi_i^{Act}\left(t_0+lT\right)=1-P \quad \textbf{11}$$

As $f(P) = -\ln(1-P)$, then when P is small, then $f(P)$ is also small, and it holds that $f(P) = -\ln(1-P) \approx P$, making equations 10 and 11 to give similar results.

Wrapping up, when $I\Delta < 0$ for queue i, the STP method states that equation 6 is to be used to calculate $\Delta\left(\phi_i^{Act}\left(t_0+n\tau+lT\right)\right)$ with P being the loss factor, and $f(P) = -\ln(1-P)$. If P is sufficiently small it can be used directly, instead of $f(P)$. These calculations are made at the ending of every interval of duration τ.

Even if τ were fairly large, it is wise to avoid computing logarithms, so the method proposes the following linear approximations:

$$f(P)=-\ln(1-P)\approx m\cdot P+b \quad \textbf{12}$$

Where m is the slope and b is the ordinate to origin of the line, with the chosen following parameters:

$m = 1.277064059$, $b = 0.0$, for $0 \le P < 0.4$
$m = 2.310490602$, $b = -0.41337062$, for $0.4 \le P < 0.7$
$m = 5.493061443$, $b = -2.64117021$, for $0.7 \le P \le 0.9$

$m = 6.931471806$, $b = -3.935739532$, for $0.9 < P$

Case when when $I\Delta \geq 0$ for queue *i (queue i is designated to gain weight).*

Remember that the method forms the sets A and Z so that if $I\Delta \geq 0$ for queue i then i belongs to A (the queue has to gain weight); otherwise i belongs to Z (and the queue has to lose weight).

With regard to the weights and weight-increments of the queues, the following expressions must hold.

$$\begin{aligned}
&\sum_{j=1}^{N}\Delta\left(\phi_j^{Act}\left(t_0+n\tau+lT\right)\right)=0\\
&\sum_{j\in A}\Delta\left(\phi_j^{Act}\left(t_0+n\tau+lT\right)\right)=-\sum_{j\in Z}\Delta\left(\phi_j^{Act}\left(t_0+n\tau+lT\right)\right)\\
&0\leq\sum_{j\in A}\Delta\left(\phi_j^{Act}\left(t_0+n\tau+lT\right)\right)<1\\
&-1<\sum_{j\in Z}\Delta\left(\phi_j^{Act}\left(t_0+n\tau+lT\right)\right)\leq 0\\
&\sum_{j=1}^{N}\phi_j^{Act}\left(t_0+n\tau+lT\right)=1
\end{aligned} \qquad \textbf{13}$$

Then, when $I\Delta \geq 0$, for queue i, then $i \in A$, and $\Delta\left(\phi_i^{Act}\left(t_0+n\tau+lT\right)\right)$ is calculated as:

$$\Delta\left(\phi_i^{Act}\left(t_0+n\tau+lT\right)\right)=\frac{I\Delta\left(\phi_i^{Act}\left(t_0+n\tau+lT\right)\right)}{\sum_{j\in A}I\Delta\left(\phi_j^{Act}\left(t_0+n\tau+lT\right)\right)}\left[-\sum_{j\in Z}\phi_j^{Act}\left(t_0+n\tau+lT\right)\right] \qquad \textbf{14}$$

Summing 14, $\forall i \in A$, yields,

$$\sum_{j\in A}\Delta\left(\phi_j^{Act}\left(t_0+n\tau+lT\right)\right)=-\sum_{j\in Z}\Delta\left(\phi_j^{Act}\left(t_0+n\tau+lT\right)\right)$$

This last expression fulfils equation 13. In the special case where there were only two queues, while one increases its weight, the other decreases its own, in the same amount.

Following the method proposed in [1], the average length Q_i of queue i is computed as $Q_i=\left(1-w_q\right)Q_i+w_q q_i$, where w_q is the averaging parameter and q_i is the instantaneous queue-length. This equation acts as a low-pass filter. The smallest the value of w_q the smoother the output will be. The value for w_q is set to 0.002, to cope with the burst behavior of the instantaneous queue-length. This average length can be calculated in different ways – in a simulator it can be calculated every time a package arrives of leaves the queue.

Theorem 1. The Maximum Expected Lose of Weight.

Consider that queue i obtains T/τ consecutive results indicating that it has to lose weight. The results are obtained within a big time interval of duration T, at times $t_0 + n\tau + lT + \tau$, that is, starting at time $t_0 + l\,T + \tau$ (when $n = 0$) and ending with calculation at $t_0 + (l + 1)T$ (when $n = T/\tau - 1$). Then equation 7 holds (this equation is repeated here for purposes of convenience).

$$\phi_i^{Act}\left(t_0+\left(l+1\right)T\right)-\phi_i^{Act}\left(t_0+lT\right)\approx\phi_i^{Act}\left(t_0+lT\right)\left(e^{-f(P)}-1\right) \qquad \textbf{7 (Rept).}$$

Proof

From equation 5, changing r by $n + l\ T/\tau$ and remembering that l and n are integers such that $l \geq 0$ and $0 \leq n < T/\tau$, and that T/τ is an integer much greater than 1, the next equations are obtained:

$$\Delta\left(\phi_i^{Act}\left(t_0+\left(n+l\frac{T}{\tau}\right)\tau\right)\right)=\phi_i^{Act}\left(t_0+\left(n+l\frac{T}{\tau}+1\right)\tau\right)-\phi_i^{Act}\left(t_0+\left(n+l\frac{T}{\tau}\right)\tau\right)$$

Calculating with equation 5, using time $(t_0 + lT + \tau)$, when $n = 0$, and rearranging:

$$\phi_i^{Act}\left(t_0+lT+\tau\right)=\phi_i^{Act}\left(t_0+lT\right)+\Delta\left(\phi_i^{Act}\left(t_0+lT\right)\right)=\phi_i^{Act}\left(t_0+lT\right)-f\left(P\right)\frac{\tau}{T}\phi_i^{Act}\left(t_0+lT\right)=\phi_i^{Act}\left(t_0+lT\right)\left(1-\frac{f\left(P\right)}{T/\tau}\right)$$

Calculating again with equation 5, using time $(t_0 + lT + 2\tau)$, when $n = 1$, and rearranging:

$$\phi_i^{Act}\left(t_0+lT+2\tau\right)=\phi_i^{Act}\left(t_0+lT+\tau\right)+\Delta\left(\phi_i^{Act}\left(t_0+lT+\tau\right)\right)=\phi_i^{Act}\left(t_0+lT+\tau\right)-f\left(P\right)\frac{\tau}{T}\phi_i^{Act}\left(t_0+lT+\tau\right)=$$

$$\phi_i^{Act}\left(t_0+lT+\tau\right)\left(1-\frac{f\left(P\right)}{T/\tau}\right)=\phi_i^{Act}\left(t_0+lT\right)\left(1-\frac{f\left(P\right)}{T/\tau}\right)^2$$

The last calculation, using time $(t_0 + (l + 1)T)$, when $n = T/\tau - 1$, and rearranging:

$$\phi_i^{Act}\left(t_0+\left(l+1\right)T\right)=\phi_i^{Act}\left(t_0+lT\right)\left(1-\frac{f\left(P\right)}{T/\tau}\right)^{T/\tau}$$

As $T/\tau >> 1$, the above result is approximated as $\phi_i^{Act}\left(t_0+\left(l+1\right)T\right)=\phi_i^{Act}\left(t_0+lT\right)e^{-f(P)}$

From here, it follows that $\phi_i^{Act}\left(t_0+\left(l+1\right)T\right)-\phi_i^{Act}\left(t_0+lT\right)\approx\phi_i^{Act}\left(t_0+lT\right)\left(e^{-f(P)}-1\right)$

□

Corollary.

From the results of **Theorem 1**, the maximum expected loss of weight for queue i would be $e^{-f(P)}-1$, the reason for this comes from observing equation 8.

Appendix 1. Details about the Implementation of the STP Method in ns-2.

The simulations used[3] ns-2, version 2.33, also using the DS tools included in the simulator [2]. In the STP method this tools of ns-2 were modified to add the WFQ packet-scheduling operation, with the implementation proposed by researchers of the Swiss Federal Institute of Technology, Zurich [3], who followed the algorithm of WFQ made by Parekh and Gallager [4][4].

Additional modifications[5] were made to ns-2 to incorporate the possibility to change the weights of the WFQ scheduler in a dynamic form, as a function of the length of the queues, in accordance with the STP method ([see Technical report #3 of this group of technical reports –see footnote 2 on page 2 of this document].

To implement the STP method the DS functions of ns-2 are used in the following form. With regard to **Fig. 1**, for the *q2* and *q2f* cases, the output interface of *c1* (the central node) has two queues (this is the only interface in the experiment which follows the STP method). The packets coming from *s1* to *e1* (packets of *Route1*) are placed in class #10 when entering the DS Domain. The packets of flows coming from *s3* to *e3* (packets of *Route3*) are placed in class #18 when entering the DS Domain.

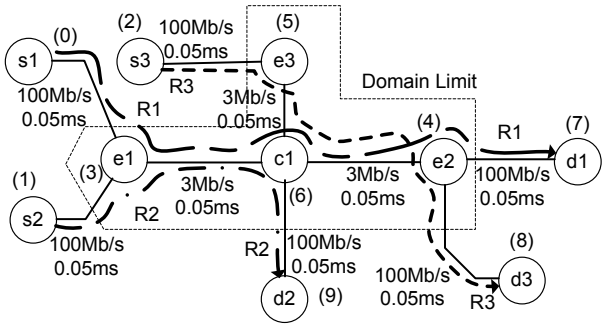

Fig. 1. The topology for the experiments has two routes: *Route1* traversing nodes *s1–e1–c1–e2–d1*, and *Route3* traversing nodes *s3–e3–c1–e2–d3*. The routes intersect at the output interface of node *c1* and separate at node *e2*. *Route1* and *Route2* intersect at the output interface of node *e1*.

In this way, when packets get to the output interface of *c1* (going to *c2*), these packets are recognized by their class, so that packets arriving from *e1* are placed in one queue and packets arriving from *e3* are placed in another queue. It is not the interest of this paper to use the DS behaviors in other way but just as a tool to obtain the desired operation for the STP method.

The policer at *e1* could place packets of class #10 in class #11, and the policer at *e3* could place packets of class #18 in class #19. This had practically no impact in the results of the experiments as only a very small fraction of packets were penalized by policers (see **Fig. 2**).

[3] This version of ns2 was used with the Fedora 12 distribution of Linux.
[4] For this just 3 C++ ns-2 files were modified: dsred.h, dsred.cc and wfq-list.h.
[5] Just to the 3 files mentioned at footnote 4.

```
Packets Statistics
=======================================
 CP  TotPkts   TxPkts   ldrops   edrops
 --  -------   ------   ------   ------
All  2182194  2182194        0        0
 10  1056710  1056710        0        0
 11       35       35        0        0
 12  1125230  1125230        0        0
 13      219      219        0        0

 qc1e2-estadísticas.

Packets Statistics
=======================================
 CP  TotPkts   TxPkts   ldrops   edrops
 --  -------   ------   ------   ------
All  2284295  2284295        0        0
 10  1056709  1056709        0        0
 11       35       35        0        0
 18  1225487  1225487        0        0
 19     2064     2064        0        0

 qe3c1-estadísticas.

Packets Statistics
=======================================
 CP  TotPkts   TxPkts   ldrops   edrops
 --  -------   ------   ------   ------
All  1227552  1227552        0        0
 18  1225488  1225488        0        0
 19     2064     2064        0        0
1200.000182
```

Fig. 2. One of the tables of statistics obtained from the runs of the simulations.

The links connecting to *c1* are of type "dsRED". These links have DS functions. The nodes connecting towards *c1* (*e1*, *e2* and *e3*) have the edge-node functions of policing and of incorporation of packets to classes.

The policer type is TokenBucket with committed interface rate (CIR) and committed burst size indicated in Appendix 2.

The queues used in the dsRED nodes (like *c1*) were designed [2] to employ the RED [1] congestion avoidance strategies. These strategies are not part of the STP method, and were practically not used in the experiments, as there were practically no discards in the two queues of link *c1–e2*, as the limits of the amount of packets held by these queues were especially big: 20′000 packets. Anyway, these RED parameters were required to be set for the experiments.

For these RED parameters, the minimum threshold was 4000 packets; the maximum threshold was 20000 packets. The probability of throwing a packet if it arrived to the queue when its average length was between the minimum and maximum thresholds were: 0.02 for virtual queues of non-penalized packets (of classes 10 or 18), and 0.1 for virtual queues of penalized-packets (of classes 11 and 19).

The queue weighting parameter used to calculate the average length of a queue is equal to 0.002. This parameter is used in the equation $Q = (1 - w_q)\, Q + w_q \cdot q$, where w_q is queue weighting parameter, Q is the average length of the queue and q is the instantaneous length of the queue.

For the delay time, a distance of 4[Km] between any two nodes, and approximately 1.5 x $10^8 m/s$ of propagation speed, to obtain 0.027*ms* of propagation delay (to this propagation the simulator adds other delays like queuing delay and transmission delay). With all this, a 0.05*ms* was used as a propagation delay.

All events of interest throughout the experiment are saved, to be analyzed later, in trace files.

Appendix 2. ns Script File.

```
# Este programa topologia5e es igual al de topologia5 pero con sus argumentos de entrada
ordenados.
# ns topologia5e.ns
# En la topologia5 cambian las fuentes con relación a la topologia3
# Creator. Alfredo Mateos. Feb - Apr 2007.
# Modificacion. Dec. 2008. Alfredo Mateos.
# MODIFICATION DEC 2008. The filter to the trace-all command.
# Modified Jun 2009- Nov 2010.

# OTcl Test Procedure.
# This Script takes as a model the Script included in the Publication "A Network
#  Simulator Differentiated Services Implementation. Open IP, Nortel Networks",
#  by P. Pieda, J. Ethridge, M. Baines, F. Shallwani.
# The WFQ scheduler is also implemented. This funciontality in ns-2 was taken from
#  the report from Aleksander Mrkaic 2001. Alfredo Mateos modified the functionality
#  to include the possibility to change the weights of this ns-2 WFQ scheduler
#  according to the sizes of the queues and according to an algorithm also designed
#  by Alfredo Mateos.
#-----------------------------------------------------------------
#                                        ______
#     (0)             -(2)          /  -(5)\
#    ----             ----          | ----  |
#    |s1|---------\   |s3|----------+-|e3|  |_DOMINION
#    ---- 100Mb/s  \  ---- 100Mb/s  | --+-          |_____
#        0.05ms     \       0.05ms   |   |3Mb/s            \
#                    _\_______ _____/    |0.05ms       -(4)|
#                  /  \---            --+-            ----|         ----
#                 |   |e1|------------|c1|------------|e2|+---------|d1|(7)
#                  \ /----    3Mb/s   ---+     3Mb/s  --+-| 100Mb/s ----
#     (1)           X (3)    0.05ms  (6) |    0.05ms    | | 0.05ms
#    ----          / \_____________________|_____________|_|
#    |s2|---------/                       |             |
#    ---- 100[Mb/s]              100Mb/s |       100Mb/s\     ----
#        0.05[ms]                0.05ms -+--       0.05ms\----|d3|(8)
#                                       |d2|(9)               ----
#-----------------------------------------------------------------

# A flow in this ds environment simulation is identified by its pair: [source node - \
   destination node] All flows matching the same pair are treated as a \
   single traffic aggregate (behavior aggregate) This information is stored in \
   the a Policy Table. There is a Policy Table for each interface pointing towards \
   the inward of the Dominium, of each edge node. This means that there must be \
   an accordance regarding the information contained in the different Policy \
   Tables in operation.

# The routes s1-d1 and s3-d3 are predefined. Through the simulator it is easy to define \
   routes. In practice, some method like MPLS should be used. Route s1-d1 is the main \
   route (of interest) and route s3-d3 is the crossed route. The only interface that has \
   two queues is c1-e2 (a point where the two routes cross)


# **************************
# Set main variables.
# **************************

# Lo siguiente lo quitamos porque queremos iguales los tráficos para los experimentos con el
mismo número de fuentes.
set semilla [exec date +%N]
# set semilla 34
 $defaultRNG seed $semilla
 set rand [new RNG]

# # This program is called with several parameters (see below)
```

```
# To call this program we must write.

#  ns topologia5e.ns
#            PRM_GEN testTime T3time packetSize_tcl
#            PARAM_q2 one_two_q chgWght time_chg_wght_tcl T_tcl P_tcl weight_0_c1e2
weight_1_c1e2
#            NUM_FTS CBR1 PAR1 CBR2 PAR2 CBR3 PAR3 PAR3A
#            IMPRESN sal_trace weight_out_tcl impr_parc
#            ARCHIVS file_trace

# PRM_GEN;  # Etiqueta de ordenamiento $argv 0.
set testTime [lindex $argv 1]; # Tiempo de simulación (en segundos).
set T3time [lindex $argv 2]; # Tiempo desde donde operan el número de fuente PAR3A.
set packetSize_tcl [lindex $argv 3]; # El tamaño del paquete que manejarán las fuentes Pareto
en Bytes.
set packetSize_CBR 1500; # El tamaño del paquete de fuentes CBR en Bytes.
# PARAM_q2; # Etiqueta de ordenamiento $argv 4.
set one_two_q [lindex $argv 5]; # Se indica si habrá 1 ó 2 colas en c1 - e2 (s1 - e1 y s3 -
e3).
set chgWght [lindex $argv 6]; # Habilita cambio dinámico de pesos en c1 - e2 (si hay más de 1
cola ahí).
set time_chg_wght_tcl [lindex $argv 7]; # Intervalo. Tiempo entre cambio de pesos.
set T_tcl [lindex $argv 8]; # Valor de T.
set P_tcl [lindex $argv 9]; # Valor de P.
set weight_0_c1e2 [lindex $argv 10]; # Peso #0 c1 - e2 (para s1 - d1 y s2 - d1) Entero.
set weight_1_c1e2 [lindex $argv 11]; # Peso #1 c1 - e2 (para s3 - d3) Entero.
# NUM_FTS; # Etiqueta de ordenamiento $argv 12.
set CBR1 [lindex $argv 13]; # Num Fuentes CBR s1.
set PAR1 [lindex $argv 14]; # Número de fuentes en s1, desde el inicio al final.
set CBR2 [lindex $argv 15]; # Num Fuentes CBR s2.
set PAR2 [lindex $argv 16]; # Num Fuentes PAR s2.
set CBR3 [lindex $argv 17]; # Num Fuentes CBR s3.
set PAR3 [lindex $argv 18]; # Número de fuentes en s3.
set PAR3A [lindex $argv 19]; # Número de fuentes en s3 que inician desde T3time.
# IMPRESN; # Etiqueta de ordenamiento argv 20.
set sal_trace [lindex $argv 21]; # Habilita (=1) la salida de registro de sucesos a archivo.
set weight_out_tcl [lindex $argv 22]; # Si =1 crea archivo con pesos_cola y long_prom_cola.
Cola c1 - e2.
set impr_parc [lindex $argv 23]; # Pone en pantalla núm. paquetes encolados y tirados.
# ARCHIVS; # Etiqueta de ordenamiento $argv 24.
set file_trace [lindex $argv 25]; # Nombre archivo que recibe reg de sucesos (se pasa a
rutinas en C++)
#--PARAMETROS NO_USADOS O SIEMPRE FIJOS.
set typSch WFQ; # WFQ - WRR Tipo de scheduler en c1 - e2 (cuando hay más de 1 cola)
set wl1on_tcl 0; # Modificación de algoritmo (no usado por ahora).

# Creacón de nombre de archivo de pesos (para caso q2)
# set archivo [string range $file_trace [expr {[string last / $file_trace] + 1}] [string
length $file_trace]]
set archivo $file_trace
if {[string last . $archivo] > 0} {set archivo [string range $archivo 0 [expr {[string last .
$archivo] -1}]]}
set file_wght "$archivo.wgt"
# Concluye creación de nombre de archivo de pesos.
# El archivo se pone en el directorio papá.


# ***************************
# Set reference to the unique OTcl object of class Simulator.
set ns [new Simulator]
# ns is the reference to the unique OTcl object of class Simulator.
# ***************************

# ***************************
# Trace Command
# ***************************
if {$sal_trace == 1} {
```

```
 set n_nod_fte_s1d1 0;
 set n_nod_egr 4;
 set n_nod_rec_s1d1 7;
 set n_nod_fte_s3d3 2;
 set n_nod_rec_s3d3 8;
 set n_nod_cen 6;
 # Remember:
 # $1 = "+" means the packet has arrived completely to the queue of the transmiting node.
 # $1 = "-" means that the packet starts to leave the queue of the transmiting node.
 # $1 = "r" means tha packet has arrived completely to the queue of the receiving node.
 # $substr($9,1,1) = source node of the path.
 # $substr($10,1,1) = destination node of the path.
 # We are interested in the delay time of a packet since ti has been completely received
 #  inside the domain (completely received by e3 or by e1)

# LAS SIGUIENTES LINEAS HACEN MAS PEQUEÑA LA SALIDA
 set awkcode1
"(((\$3==$n_nod_fte_s1d1)||(\$3==$n_nod_egr))&&(\$1==\"r\")&&(substr(\$9,1,1)==$n_nod_fte_s1d1
)&&(substr(\$10,1,1)==$n_nod_rec_s1d1))";
 set awkcode3
"(((\$3==$n_nod_fte_s3d3)||(\$3==$n_nod_egr))&&(\$1==\"r\")&&(substr(\$9,1,1)==$n_nod_fte_s3d3
)&&(substr(\$10,1,1)==$n_nod_rec_s3d3))";
 append awkcode1 "||$awkcode3";

# La operacón del programa tasa.awk requiere que este archivo genere los registros de las
llegadas y salidas
#  de paquetes al nodo 6 (c2), lo que se hace descomentando (activando) las siguientes DOS
líneas.
# set awkcode6 "(\$3==$n_nod_cen)&&((\$1==\"+\")||(\$1==\"r\"))";
# append awkcode1 "||$awkcode6";

 # puts "\n$awkcode1;"
$ns trace-all [open "|gawk $awkcode1 > $file_trace" w];
# $ns trace-all [open $file_trace w];
}; # End if $sal_trace==1

# End if.
# ***************************

# ***************************
# Set variables for the policers.
# ***************************
set cir_par 1500000; # [bits / s] Committed Interface Rate.
# Para todas las fuentes PARETO de s1 (nodo 0 en ns) a d1 (nodo 7 en ns)
#  o de s3 (nodo 2 en ns) a d3 (nodo 8 en ns)

set cbs_par 10000; # [bytes] Committed Burst Syze.
# Relacionado con cir_par.

set cir_cbr 800000; # [bits / s] Committed Interface Rate.
# Para las fuentes de tasa constante de s2 (nodo 1 en ns) a d1 (nodo 7 en ns)

set cbs_cbr 2000; # [bytes] Committed Burst Syze.
# Relacionado con cir_cbr

# In the manual "A Network Simulator Differentiated Services Implementation" of \
   Pieda et al, Pg. 29 we can see the units for CIR and CBS.
# ***************************

# ***************************
# Set variables for Source #1, Source #2 and Source #3.
# ***************************
set rate0 68000; # [bits / sec] Modificado para topologia3.
set rate1 256000; # [bits / sec] Modificado para topologia3.
# In order to see forms of entering values using variables see. Tcl Basics. \
   of B. Welch. Pg. 16.
# ***************************
```

```
# ***************************
# Set up the nodes.
# ***************************
set s1 [$ns node]; # Source node #1. Node 0 in ns.
set s2 [$ns node]; # Source node #2. Node 1 in ns.
set s3 [$ns node]; # Source node #3. Node 2 in ns.
set e1 [$ns node]; # Edge node #1. Node 3 in ns.
set e2 [$ns node]; # Edge node #2. Node 4 in ns.
set e3 [$ns node]; # Edge node #3. Node 5 in ns.
set c1 [$ns node]; # Core node. Node 6 in ns.
set d1 [$ns node]; # Destination node #1. Node 7 in ns.
set d3 [$ns node]; # Destination node #3. Node 8 in ns.
set d2 [$ns node]; # Destination node #2. Node 9 in ns. Nuevo para topologia3.
# puts "Nodes made"
# ***************************

# ***************************
# Set up links and main characteristics.
# ***************************
$ns duplex-link $s1 $e1 100Mb 0.05ms DropTail
$ns duplex-link $s2 $e1 100Mb 0.05ms DropTail
$ns duplex-link $s3 $e3 100Mb 0.05ms DropTail
$ns duplex-link $d1 $e2 100Mb 0.05ms DropTail
$ns duplex-link $d3 $e2 100Mb 0.05ms DropTail
$ns duplex-link $c1 $d2 100Mb 0.05ms DropTail; # Nuevo para topologia3.
# The DS queues are:
$ns simplex-link $e1 $c1 3.0Mb 0.05ms dsRED/edge
$ns simplex-link $c1 $e1 3.0Mb 0.05ms dsRED/core
$ns simplex-link $c1 $e2 3.0Mb 0.05ms dsRED/core
$ns simplex-link $e2 $c1 3.0Mb 0.05ms dsRED/edge
$ns simplex-link $c1 $e3 3.0Mb 0.05ms dsRED/core
$ns simplex-link $e3 $c1 3.0Mb 0.05ms dsRED/edge
# ***************************

# Cambios para topologia3
$ns queue-limit $c1 $e2 25000
$ns queue-limit $e3 $c1 4000
$ns queue-limit $e1 $c1 4000; # Modificado para topologia3.

# ***************************
# Create handles to each of six DS queues.
# ***************************
set qe1c1 [[$ns link $e1 $c1] queue]
set qe2c1 [[$ns link $e2 $c1] queue]
set qe3c1 [[$ns link $e3 $c1] queue]
set qc1e1 [[$ns link $c1 $e1] queue]
set qc1e2 [[$ns link $c1 $e2] queue]
set qc1e3 [[$ns link $c1 $e3] queue]
# ***************************

# ***************************
# Set DS RED parameters at the exit interface of the node e1 to go to node c1.
# ***************************
# $qe1c1 setChgWghtOpt 1; We do not want e1 to have more than one queue.
# This makes that the weight-change process takes effect when using the WFQ \
   or the WRR scheduling forms (the C++ variable set_chw_op is set to 1) \
   This is not necessary here as the dsREDQueue has only one queue.
# ***************************

$qe1c1 set numQueues_ 1
# The variable numQueues_ is the number of physical queues that will be used at
#  the exit interface of the node e1 to go to node c1. numQueues_ (must be less
#  than MAX_QUEUES = 8 --see dsred.h--)
#  numQueues_ is bound to numQueues_ C++ variable --see dsred.cc--)
#  numQueues_ will be used in many of the routines instead of MAX_QUEUES.
```

```
$qe1c1 setMREDMode RIO-C
# Sets up the average queue accounting mode (to calculate queue size)
#  of physical queues from 0 to MAX_QUEUES - 1 (where MAX_QUEUES = 8)
# $qe1c1 setMREDMode RI0-C i sets up the average queue accounting mode of
#  physical queue i of ds group of queues $qe1c1.
# The modes available are RIO-C, RIO-D, WRED, DROP (see dsred.cc and the manual
#  A Network Simulator Differentiated Services Implementation, of Pieda et al
#  Pg. 18) By default the MRED mode is set to RIO-C.

$qe1c1 setNumPrec 2
# Sets the number of precedences (number of virtual queues) for each of the
#  8 physical queues of the ds group of queues (numbered from 0 to 7)
#  Each one of the physical queues will have the same number of virtual queues.
# Explanation regarding C++ code:
# This line runs the setNumPrec routine of the ds object of class dsREDQueue,
#  with one argument equal to 2. See dsred.cc. This setNumPrec routine
#  sets the number of precedences for all the physical queues of the mentioned
#  object (the numbering of the physical queues go from 0 to MAX_QUEUES - 1
#  (not to numQueues_ - 1) where MAX_QUEUES = 8.

$qe1c1 meanPktSize $packetSize_tcl
# Sets the mean packet size for each of the virtual queues (set in the above line)
#  of each of the 8 physical queues of the ds group. All queues receive the same
#  value.
# Explanation regarding C++ code:
# This line runs the setMPS routine for each of the MAX_QUEUES (=8) physical
#  queues (objects redq_[i] for i from 0 to MAX_QUEUES each of redQueue class)
#  of the ds object (of dsREDClass class) (see dsredq.cc) (see red.h variable
#  edp.mean_pktsize given in bytes)
#  The setMPS routine sets the mean packet size of each of the virtual queues
#  (all with the same value) of red queue (of redQueue class)

$qe1c1 addPolicyEntry [$s1 id] [$d1 id] TokenBucket 10 $cir_par $cbs_par
# Establishes the first policy entry to the policy table of the exit interface of
#  node e1 to go to node c1.
# In the manual "A Network Simulator Differentiated Services Implementation" of
#  Pieda et al, Pg. 26, says: "A policy is established between a source and
#  destination node. All flows matching that source-destination pair are treated
#  as a single traffic aggregate."
# This policy table assigns the codepoint 10 to each packet of the aggregate that
#  goes from node s1 to node d1. This codepoint is assigned before the packet
#  passes through the policer of the inferface of node e1 to go to node c1.
# Source node = s1
# Destination node = d1
# Policer type: TokenBucket
# Initial code point = 10.
# CIR: ($cir_par) Committed information rate [bits / sec]
# CBS: ($cbs_par) Committed burst size [bytes]
# A policy table consists of policy entries. Each entry has four main elements
#  that distinguishes one entry from another: 1- source node, 2- destination node,
#  3- policer type, 4- initial code point.

# Modificado para topologia3
$qe1c1 addPolicyEntry [$s2 id] [$d2 id] TokenBucket 12 $cir_par $cbs_par
# Establishes the second policy entry to the policy table of the exit interface of
#  node e1 to go to node c1.

$qe1c1 addPolicerEntry TokenBucket 10 11
# Establishes a policer entry to the policer table of the exit interface of node
#  e1 to go to node c1.
# Each entry of the Policer Table is distinguished by two elements: 1- A policer
#  type, and 2- An initial code point. The policer entry specifies one or two target
#  code points to which a downgrading process could lead, from an initial code point,
#  when using a specific policer. For some policer types there is just one code point
#  towards which a flow is to be downgraded, if it does not fulfill the policer
#  objectives; for some other policer types there are two code points a flow can be
#  downgraded to.
```

```
$qe1c1 addPolicerEntry TokenBucket 12 13
# Establishes a policer entry to the policer table of the exit interface of node
#  e1 to go to node c1.

$qe1c1 addPHBEntry 10 0 0
# Adds an entry to the PHB table of the exit interface of node e1 to go to node c1.
# After a packet is revised by the policer in this interface, its code point will
#  be left unchanged, or it will be changed in accordance to the policer results, and
#  to the Policer Table.
# Any packet with code point 10 will be forwarded to physical queue 0 and virtual
#  queue 0 of the exit interface of node e1 to go to node c1.

$qe1c1 addPHBEntry 11 0 1
# Any packet with code point 11 will be forwarded to physical queue 0 and virtual
#  queue 1 of the exit interface of node e1 to go to node c1.

$qe1c1 addPHBEntry 12 0 0
# Any packet with code point 12 will be forwarded to physical queue 0 and virtual
#  queue 0 of the exit interface of node e1 to go to node c1.

$qe1c1 addPHBEntry 13 0 1
# Any packet with code point 13 will be forwarded to physical queue 0 and virtual
#  queue 1 of the exit interface of node e1 to go to node c1.

$qe1c1 configQ 0 0 200 400 0.02 0.002
# Configures the virtual queue #0 of the physical queue #0 of the exit interface of
#  node e1 to go to node c1.
# Parameters (see "Random Early Detection Gateways for Congestion Avoidance" paper
#  of S. Floyd and V. Jacobson), in order:
# 0 = Number of physical queue.
# 0 = Number of virtual queue of physical queue indicated.
# 20 = Minimum threshold (minth) in packets.
# 40 = Maximum threshold (maxth) in packets.
# maxth is recommended to be 3 times minth.
# 0.02 = 1 / 50 = Maxp = max probability of throwing a packet, if the average queue length
#  when the packet arrives, is between minth and maxth. This value is used in the article:
# "Random Early Detection Gateways for Congestion Avoidance" of Floyd and Jacobson
#   (see pg. 9 and 11)
# 0.002 = wq (0.002 = default value) Queue weight to measure the average length of
#  the queue. avg = (1 - wq)avg + wq * (instanteneous queuelength)
#  Value also used in Floyd and Jacobson article.
# C++ Explanation.
#  It runs the config routine of the object of class redQueue (the physical queue)
#  with number 0 (first argument after configQ) with the pointer name redqp_[0], and it
#  passes to it the arguments.

$qe1c1 configQ 0 1 200 400 0.10 0.002
# Configures the virtual queue #1 of the physical queue #0 of the exit interface of
#  node e1 to go to node c1

# $qe1c1 setSchedularMode WFQ; # Not necessary here as there is only one queue.
# See dsred.cc.
# The Scheduling modes are: WRR, RR, WIRR, PRI and WFQ (This last form added by
#  A MrKaic)

# $qe1c1 addQueueWeights 0 5; # "Initial" queue weight 5 to physical queue 0 of ds group \
   group of queues of the interface of node e1 that goes from e1 to c1 (object name qe1c1 \
   of dsREDQueue class)
# QueueWeigths not necessary here as there is only one queue.
# In WRR, RR and WIRR the weights should be integers.
# In WFQ the weight must be integer as the C++ program is only ready to accept it as \
   integer, but otherwise it could work all right with weights not being integers.
# **************************

# **************************
# Print tables qe1c1
```

```
# ***************************
# $qe1c1 printPHBTable; # Prints the entire PHB Table of $qe1c1
# $qe1c1 printPolicyTable
# $qe1c1 printPolicerTable
# ***************************

# ***************************
# Set DS RED parameters at the exit interface of the node e2 to go to node c1.
# Not necessary as there is not traffic in this direction.
# ***************************
$qe2c1 set numQueues_ 1
$qe2c1 setMREDMode RIO-C
$qe2c1 setNumPrec 2
$qe2c1 meanPktSize $packetSize_tcl
# From d1 to s1
$qe2c1 addPolicyEntry [$d1 id] [$s1 id] TokenBucket 14 $cir_par $cbs_par
$qe2c1 addPolicerEntry TokenBucket 14 15
$qe2c1 addPHBEntry 14 0 0
$qe2c1 addPHBEntry 15 0 1
# From d3 to s3
$qe2c1 addPolicyEntry [$d3 id] [$s3 id] TokenBucket 20 $cir_par $cbs_par
$qe2c1 addPolicerEntry TokenBucket 20 21
$qe2c1 addPHBEntry 20 0 0
$qe2c1 addPHBEntry 21 0 1
# Config Virtual Queues
$qe2c1 configQ 0 0 200 400 0.02 0.002
$qe2c1 configQ 0 1 200 400 0.10 0.002
# $qe2c1 setSchedularMode WFQ; # Not used as there is only one queue.
# $qe2c1 addQueueWeights 0 5; # Not used as there is only one queue.
# ***************************

# ***************************
# Set DS RED parameters at the exit interface of the node c1 to go to node e1.
# There are no policers in core nodes.
# Not necessary as there is not traffic in this direction.
# ***************************
$qc1e1 set numQueues_ 1
$qc1e1 setMREDMode RIO-C
$qc1e1 setNumPrec 2
$qc1e1 meanPktSize $packetSize_tcl
# There is just one physical queue at exit interface from node c1 to node e1.
$qc1e1 addPHBEntry 14 0 0
$qc1e1 addPHBEntry 15 0 1
$qc1e1 configQ 0 0 4000 20000 0.02 0.002
$qc1e1 configQ 0 1 4000 20000 0.10 0.002
# $qc1e1 setSchedularMode WRR; # Not used as there is only one queue.
# $qc1e1 addQueueWeights 0 5; # Not used as there is only one queue.
# ***************************

# ***************************
# Set DS RED parameters at the exit interface of the node c1 to go to node e2.
# There are no policers in core nodes.
# ***************************
if {$one_two_q == 1} {
 # One queue in node c1 to go to node e2.
 $qc1e2 setChgWghtOpt 0
  # Even with one queue here, we can send impression of weights and lengths.
  # $qc1e2 weight_out $weight_out_tcl $file_wght; # although it makes no sense.
 $qc1e2 set numQueues_ 1
 $qc1e2 setMREDMode RIO-C
 $qc1e2 setNumPrec 2
 $qc1e2 meanPktSize $packetSize_tcl
 $qc1e2 addPHBEntry 10 0 0
 $qc1e2 addPHBEntry 11 0 1
 $qc1e2 addPHBEntry 18 0 0
 $qc1e2 addPHBEntry 19 0 1
 $qc1e2 configQ 0 0 4000 20000 0.02 0.002
```

```
 $qc1e2 configQ 0 1 4000 20000 0.10 0.002

 #SUSTITICION DE LA SIGUIENTE LINEA POR LA SUBSIGUIENTE - A MATEOS - ABR 2010.
 $qc1e2 setSchedularMode $typSch; # although the scheduler is not used with just one queue.
 # $qc1e2 setSchedularMode RR

 $qc1e2 addQueueWeights 0 1; # Queue weight to physical queue 0.
  # Queue from c1 to e2 is handled with object named qc1e2 of class type dsREDQueue in C++.
} else {
 # Two queues in node c1 to go to node e2. There could be dynamic change of weight for the
 #  queues.
 $qc1e2 setChgWghtOpt $chgWght
 $qc1e2 T $T_tcl
 $qc1e2 P $P_tcl
 $qc1e2 queue_length_decr $wl1on_tcl
 if {$weight_out_tcl == 1} {$qc1e2 weight_out $weight_out_tcl $file_wght}; # Archivo de salida
de pesos.
 $qc1e2 time_chg_wght $time_chg_wght_tcl
 $qc1e2 set numQueues_ 2
 $qc1e2 setMREDMode RIO-C
 $qc1e2 setNumPrec 2
 $qc1e2 meanPktSize $packetSize_tcl
 $qc1e2 addPHBEntry 10 0 0
 $qc1e2 addPHBEntry 11 0 1
 $qc1e2 addPHBEntry 18 1 0
 $qc1e2 addPHBEntry 19 1 1
 # The sizes of theses queues might be to big. We want to avoid losses.
 $qc1e2 configQ 0 0 4000 20000 0.02 0.002
 $qc1e2 configQ 0 1 4000 20000 0.10 0.002
 $qc1e2 configQ 1 0 4000 20000 0.02 0.002
 $qc1e2 configQ 1 1 4000 20000 0.10 0.002
 $qc1e2 setSchedularMode $typSch
 $qc1e2 addQueueWeights 0 $weight_0_c1e2; # Queue weight to physical queue 0.
  # Queue from c1 to e2 is handled with object named qc1e2 of class type dsREDQueue in C++.
 $qc1e2 addQueueWeights 1 $weight_1_c1e2; # Queue weight to physical queue 1.
} ; # end if.
# ***************************

# ***************************
# Set DS RED parameters at the exit interface of the node e3 to go to node c1.
# ***************************
$qe3c1 set numQueues_ 1
$qe3c1 setMREDMode RIO-C
$qe3c1 setNumPrec 2
$qe3c1 meanPktSize $packetSize_tcl
$qe3c1 addPolicyEntry [$s3 id] [$d3 id] TokenBucket 18 $cir_par $cbs_par
$qe3c1 addPolicerEntry TokenBucket 18 19
$qe3c1 addPHBEntry 18 0 0
$qe3c1 addPHBEntry 19 0 1
$qe3c1 configQ 0 0 200 400 0.02 0.002
$qe3c1 configQ 0 1 200 400 0.10 0.002
# $qe3c1 setSchedularMode WRR; # Not used as there is only one queue.
# $qe3c1 addQueueWeights 0 5; # Not used as there is only one queue.
# ***************************

# ***************************
# Set DS RED parameters at the exit interface of the node c1 to go to node e3.
# There are no policers in core nodes.
# Not necessary as there is not traffic in this direction.
# ***************************
$qc1e3 set numQueues_ 1
$qc1e3 setMREDMode RIO-C
$qc1e3 setNumPrec 2
$qc1e3 meanPktSize $packetSize_tcl
# There is just one physical queue at exit interface from node c1 to node e3.
$qc1e3 addPHBEntry 20 0 0
$qc1e3 addPHBEntry 21 0 1
```

```
$qc1e3 configQ 0 0 4000 2000 0.02 0.002
$qc1e3 configQ 0 1 4000 2000 0.10 0.002
# $qc1e3 setSchedularMode WRR; # Not used as there is only one queue.
# $qc1e3 addQueueWeights 0 5; # Not used as there is only one queue.
# **************************

# ************************************
# **** Set up the traffic generation. ***
# ************************************
# ******************************************************
# **FUENTE CBR PARA s1 a d1***
# ******************************************************
  # An agent is a place where the network packets (IP packets) are built or
  #  terminated.
  set start1cbr 0.15; # seconds.
  set u1cbr [new RandomVariable/Uniform]

  for {set i 0} {$i < $CBR1} {incr i} {
    set src_1cbr($i) [new Application/Traffic/CBR]
    $src_1cbr($i) set packetSize_ $packetSize_CBR; #[bytes]
    $src_1cbr($i) set interval_ [expr {1.0 / [expr {$rate1 / [expr {8.0 *
$packetSize_CBR}]}]}]
    # $interval_ (should be given in: [sec / packet])
    # $packetSize_CBR (given in: [bytes / packet])
    # 8.0 * $packetSize_CBR (given in: [bits / packet])
    # $rate1 (should be given in: [bits / sec])

    set udp_1cbr($i) [new Agent/UDP]
    set nul_1cbr($i) [new Agent/Null]
    $ns attach-agent $s1 $udp_1cbr($i)
    $ns attach-agent $d1 $nul_1cbr($i)
    $src_1cbr($i) attach-agent $udp_1cbr($i)
    $ns connect $udp_1cbr($i) $nul_1cbr($i)

    # *** Place the first events of traffic generation in the Scheduler.
    set w($i) [expr {[$u1cbr value] * $start1cbr}]; # A start time between 0 and $start1cbr.
    $ns at $w($i) "$src_1cbr($i) start"
    $ns at [expr {$testTime + $w($i)}] "$src_1cbr($i) stop"
  }; # End for.

# ************************************
# Set up Pareto Sources connected to UDP agents. ***From node s1 to node d1***
# ************************************
  # An agent is a place where the network packets (IP packets) are built or
  #  terminated.
  set start1par 0.15; # seconds.
  set u1par [new RandomVariable/Uniform]
  for {set i 0} {$i < $PAR1} {incr i} {
    set src_1par($i) [new Application/Traffic/Pareto]
    $src_1par($i) set burst_time_ 250ms
    $src_1par($i) set idle_time_ 250ms
    $src_1par($i) set packetSize_ $packetSize_tcl; #[bytes]
    $src_1par($i) set rate_ $rate0; # [bits / s]
    # See Tcl Basics - B. Welch. Pg. 16 to introduce a value made out of several \
      variables.
    $src_1par($i) set shape_ 1.7
     # If shape_ < n the nth moment is infinite. In this case the first moment is \
        finite and the second moment is infinite.
    set udp_1par($i) [new Agent/UDP]
    set nul_1par($i) [new Agent/Null]
    $ns attach-agent $s1 $udp_1par($i)
    $ns attach-agent $d1 $nul_1par($i)
    $src_1par($i) attach-agent $udp_1par($i)
    $ns connect $udp_1par($i) $nul_1par($i)
    # *** Place the first events of traffic generation in the Scheduler.
    set w($i) [expr {[$u1par value] * $start1par}]; # A start time between 0 and $start1par.
    $ns at $w($i) "$src_1par($i) start"
```

```
    $ns at [expr {$testTime + $w($i)}] "$src_1par($i) stop"
  }; # End for.

# ******************************************************
# **FUENTE CBR PARA s2 a d2***
# ******************************************************
  # An agent is a place where the network packets (IP packets) are built or
  #  terminated.
  set start2cbr 0.15; # seconds.
  set u2cbr [new RandomVariable/Uniform]

  for {set i 0} {$i < $CBR2} {incr i} {
    set src_2cbr($i) [new Application/Traffic/CBR]
    $src_2cbr($i) set packetSize_ $packetSize_CBR; #[bytes]
    $src_2cbr($i) set interval_ [expr {1.0 / [expr {$rate1 / [expr {8.0 *
$packetSize_CBR}]}]}]
    # $interval_ (should be given in: [sec / packet])
    # $packetSize_CBR (given in: [bytes / packet])
    # 8.0 * $packetSize_CBR (given in: [bits / packet])
    # $rate1 (should be given in: [bits / sec])

    set udp_2cbr($i) [new Agent/UDP]
    set nul_2cbr($i) [new Agent/Null]
    $ns attach-agent $s2 $udp_2cbr($i)
    $ns attach-agent $d2 $nul_2cbr($i)
    $src_2cbr($i) attach-agent $udp_2cbr($i)
    $ns connect $udp_2cbr($i) $nul_2cbr($i)

    # *** Place the first events of traffic generation in the Scheduler.
    set w($i) [expr {[$u2cbr value] * $start2cbr}]; # A start time between 0 and $start2cbr.
    $ns at $w($i) "$src_2cbr($i) start"
    $ns at [expr {$testTime + $w($i)}] "$src_2cbr($i) stop"
  }; # End for.

# ************************************
# Set up Pareto Sources connected to UDP agents. ***From node s2 to node d2***
# ************************************
  # An agent is a place where the network packets (IP packets) are built or
  #  terminated.
  set start2par 0.15; # seconds.
  set u2par [new RandomVariable/Uniform]
  for {set i 0} {$i < $PAR2} {incr i} {
    set src_2par($i) [new Application/Traffic/Pareto]
    $src_2par($i) set burst_time_ 250ms
    $src_2par($i) set idle_time_ 250ms
    $src_2par($i) set packetSize_ $packetSize_tcl; #[bytes]
    $src_2par($i) set rate_ $rate0; # [bits / s]
    # See Tcl Basics - B. Welch. Pg. 16 to introduce a value made out of several \
      variables.
    $src_2par($i) set shape_ 1.7
     # If shape_ < n the nth moment is infinite. In this case the first moment is \
        finite and the second moment is infinite.
    set udp_2par($i) [new Agent/UDP]
    set nul_2par($i) [new Agent/Null]
    $ns attach-agent $s2 $udp_2par($i)
    $ns attach-agent $d2 $nul_2par($i)
    $src_2par($i) attach-agent $udp_2par($i)
    $ns connect $udp_2par($i) $nul_2par($i)
    # *** Place the first events of traffic generation in the Scheduler.
    set w($i) [expr {[$u2par value] * $start2par}]; # A start time between 0 and $start2par.
    $ns at $w($i) "$src_2par($i) start"
    $ns at [expr {$testTime + $w($i)}] "$src_2par($i) stop"
  }; # End for.

# ******************************************************

# ******************************************************
```

```
# **FUENTE CBR PARA s3 a d3***
# ******************************************************
  # An agent is a place where the network packets (IP packets) are built or
  #  terminated.
  set start3cbr 0.15; # seconds.
  set u3cbr [new RandomVariable/Uniform]

  for {set i 0} {$i < $CBR3} {incr i} {
    set src_3cbr($i) [new Application/Traffic/CBR]
    $src_3cbr($i) set packetSize_ $packetSize_CBR; #[bytes]
    $src_3cbr($i) set interval_ [expr {1.0 / [expr {$rate1 / [expr {8.0 *
$packetSize_CBR}]}]}]
    # $interval_ (should be given in: [sec / packet])
    # $packetSize_CBR (given in: [bytes / packet])
    # 8.0 * $packetSize_CBR (given in: [bits / packet])
    # $rate1 (should be given in: [bits / sec])

    set udp_3cbr($i) [new Agent/UDP]
    set nul_3cbr($i) [new Agent/Null]
    $ns attach-agent $s3 $udp_3cbr($i)
    $ns attach-agent $d3 $nul_3cbr($i)
    $src_3cbr($i) attach-agent $udp_3cbr($i)
    $ns connect $udp_3cbr($i) $nul_3cbr($i)

    # *** Place the first events of traffic generation in the Scheduler.
    set w($i) [expr {[$u3cbr value] * $start3cbr}]; # A start time between 0 and $start3cbr.
    $ns at $w($i) "$src_3cbr($i) start"
    $ns at [expr {$testTime + $w($i)}] "$src_3cbr($i) stop"
  }; # End for.

# ******************************************************
# Set up Pareto Sources connected to UDP agents. From node s3 to node d3.
# ******************************************************
  set start3par 0.2; # seconds.
  set u3par [new RandomVariable/Uniform]
  for {set i 0} {$i < [expr {$PAR3 + $PAR3A}]} {incr i} {
    set src_3par($i) [new Application/Traffic/Pareto]
    $src_3par($i) set burst_time_ 250ms
    $src_3par($i) set idle_time_ 250ms
    $src_3par($i) set packetSize_ $packetSize_tcl; #[bytes]
    $src_3par($i) set rate_ $rate0; # [bits / s]
    $src_3par($i) set shape_ 1.7
     # If shape_ < n the nth moment is infinite. In this case the first moment is
     #  finite and the second moment is infinite.
   set udp_3par($i) [new Agent/UDP]
   set nul_3par($i) [new Agent/Null]
   $ns attach-agent $s3 $udp_3par($i)
   $ns attach-agent $d3 $nul_3par($i)
   $src_3par($i) attach-agent $udp_3par($i)
   $ns connect $udp_3par($i) $nul_3par($i)
   # *** Place the first events of traffic generation in the Scheduler.
   set w($i) [expr {[$u3par value] * $start3par}]; # A start time between 0 and $start3par.
   if {$i < $PAR3} {
     $ns at $w($i) "$src_3par($i) start"
   } else {
     $ns at [expr {$T3time + $w($i)}] "$src_3par($i) start"
   } ; # end if
   $ns at [expr {$testTime + $w($i)}] "$src_3par($i) stop"
  }; # End for.
#
# Note. The paths are not indicated but as there is only a possible path between source node
and
#  target node, there is no necessity of such indication.
#
# ******************************************************

# ******************************************************
```

```
# Printing Queue from e1 to c1 and from c1 to e2
# ******************************************************
if {$impr_parc == 1} {
 $ns at $testTime "puts {\n qe1c1-estadísticas.}"
 $ns at $testTime "$qe1c1 printStats"
 $ns at $testTime "puts {\n qc1e2-estadísticas.}"
 $ns at $testTime "$qc1e2 printStats"
 $ns at $testTime "puts {\n qe3c1-estadísticas.}"
 $ns at $testTime "$qe3c1 printStats"
} ; # End if.
# ******************************************************

# ******************************************************
# Ending Lines
# ******************************************************
if {$sal_trace == 1} {
 $ns at [expr {$testTime + 1.0}] "$ns flush-trace"; # Terminating instruction.
} ; # End if.
$ns at [expr {$testTime + 1.0001}] "exit 0"; # Terminating instruction.

$ns run; #Start the revision and execution of events of Scheduler.
# ******************************************************
# ****Tables****
#
# **Policy Table e1-c1**
# s1 d1 Token Bucket 10 $cir_par $cbs_par (commited information rate / commited burst size)
Pareto
# s2 d1 Token Bucket 12 $cir_cbr $cbs_cbr (commited information rate / commited burst size)
Pareto
# ** Policer Table e1-c1**
# Token Bucket 10 11
# Token Bucket 12 13
# **PHB Table e1-c1** (Physical queue / Virtual queue)
# 10  0  0
# 11  0  1
# 12  0  0
# 13  0  1

# **Policy Table e3-c1**
# s3 d3 Token Bucket 18 $cir_par $cbs_par
# **Policer Table e3-c1**
# Token Bucket 18 19
# **PHB Table e3-c1**
# 18  0  0
# 19  0  1

# ** Policy Table e2-c1**
# d1 s1 Token Bucket 14 $cir_par $cbs_par
# d1 s3 Token Bucket 20 $cir_par $cbs_par
# **Policer Table e2-c1**
# Token Bucket 14 15
# Token Bucket 20 21
# **PHB Table e2-c1**
# 14  0  0
# 15  0  1
# 20  0  0
# 21  0  1

# **PHB Table c1-e1**
# 14  0  0
# 15  0  1

# **PHB Table c1-e2** (for one queue)
# 10  0  0
# 11  0  1
# 18  0  0
# 19  0  1
```

```
# **PHB Table c1-e2** (for two queues)
# 10  0  0
# 11  0  1
# 18  1  0
# 19  1  1

# **PHB TABLE c1-e3**
# 20  0  1
# 21  0  1
```

Appendix 3. REFERENCES.

[1] Floyd, S., Jacobson, V.: Random Early Detection Gateways for Congestions Avoidance. IEEE/ACM Transactions on Networking, vol. 1(4), pp. 397-413. (1993).

[2] P. Pieda, J. Ethridge, M. Baines, F. Shallwani, "A Network Simulator Differentiated services Implementation", Open IP, Nortel Networks, (2000), Available: http://www-sop.inria.fr/members/Eitan.Altman/COURS-NS/DOC/DSnortel.pdf, Jun 2010.

[3] A. Mrkaic, (tutor U. Fiedler, supervisor. Prof. Dr. B. Plattner), "Porting a WFQ Scheduler into ns-2's Diffserv Environment", Computer Engineering and Networks Laboratory (Insitut für Technische Informatik und Kommunikationsnetze) Swiss Federal Institute of Technology (Eidgenössische Technische Hochschule Zürich), (2001).

[4] A. Parekh, R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case", IEEE / ACM Transactions on Networking, 1(3) (1993) 344-357.