Differentiated Services Network Simulation

Paulo Rogério Pereira, Bruno Afonso, Daniel Gomes

Instituto Superior Técnico, Universidade Técnica de Lisboa. INESC ID, Rua Alves Redol, 9. 1000-029 Lisboa, Portugal.
Phone: +351-213100345. Fax: +351-213145843. Emails: prbp@inesc.pt, baaa@rnl.ist.utl.pt, dhgo@rnl.ist.utl.pt

Abstract

This paper describes the Network Modeler application that helps differentiated services network design, planning and configuration testing. This application allows a user to draw a network, configure it, and evaluate its performance through simulation. A simple scenario is presented, showing the potentialities of the Network Modeler, the advantages of using differentiated services, and proving that this application is an excellent learning and testing tool.

I. INTRODUCTION

In the last few years, the growth of the Internet and the use of new services such as e-business, voice over IP (VoIP) and multimedia applications has risen the need to support Quality of Service (QoS) requirements and to accommodate different service levels. The differentiated services architecture (DiffServ) [1] allows to provide quality of service to users. However, its use makes network management and planning a hard task.

To help performing network design, planning and configuration testing, a differentiated services Network Modeler application was developed [2]. This paper describes this application, and how it allows creating a network topology, configuring it, and defining modifications for simulation time. The application produces the code to run the simulation, runs the simulation, retrieves the results, and presents them to the user.

The main simulation work is done by the VINT Project's *network simulator* (*ns*) [3] version 2.1b5a, with a modified version of the DiffServ patches [4]. The *ns* network simulator has an extensible simulation engine implemented in C++ that uses MIT' s Object Tool Command Language,OTcl (an object oriented version of Tcl, the Tool Command Language [5]), as the command and configuration interface. *ns* is an event driven simulator. Events are scheduled in *ns* allowing OTcl procedures to be invoked at arbitrary points in simulation time. These OTcl callbacks provide a flexible simulation mechanism, as they can be used to start or stop sources, dump statistics, define link failures, reconfigure the network topology, etc.

This paper starts by describing the Network Modeler application in the next section. The following section describes a simple scenario to show the potentialities of the application and the advantages of using differentiated services. The final section draws some conclusions and raises some further work topics.

II. NETWORK MODELER APPLICATION

Figure 1 shows the Network Modeler main screen. The palette on the left shows the three main elements used to create a topology: **Router** represents a network node with a router; **POP** represents a network Point of Presence, with traffic sources and sinks; and **Linha** represents a physical line connecting two network nodes.

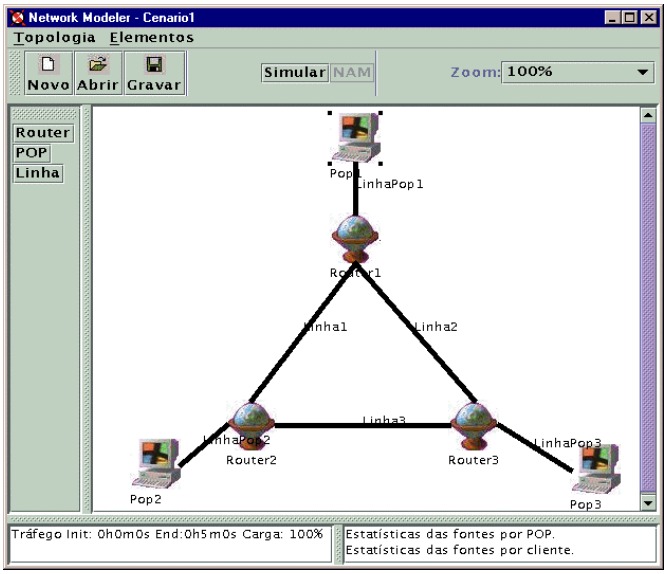

Fig. 1 Network Modeler main screen

Each element may allow defining some configuration parameters, events for simulation time, and statistics to be retrieved.

A router allows configuring only simulation events to change the state (up/down) of the router.

A POP allows configuring the number of simultaneous users, the weight of the traffic in each DiffServ class (EF, AF, BE), the weight of each application protocol (HTTP, FTP, Telnet, OnOff, CBR), the weight of incoming and outgoing traffic, the traffic profile to the sources, the average duration of user traffic, and the rate of user arrivals. Internally, the POP can have different traffic aggregation configurations, ranging from all the sources in the same node, to all sources in different nodes with traffic shapers and conditioners for each one. The simulation events include the time for starting and ending traffic generation, and the initial load. The possible statistics to be retrieved are the throughput, delay, jitter, bytes, packets, and packet loss per user, or per POP.

As shown on figure 2, a line allows configuring the routing cost, line bandwidth, and delay. Additional parameters, which are common to all lines, allow configuring the differentiated services queue weights, sizes, and random early detection (RED) [6] parameters. The simulation events allow changing the state (up/down) of the line, or the routing

cost. The possible statistics to be retrieved are line load, bytes transmitted, packets, packet loss, and average queue delay.

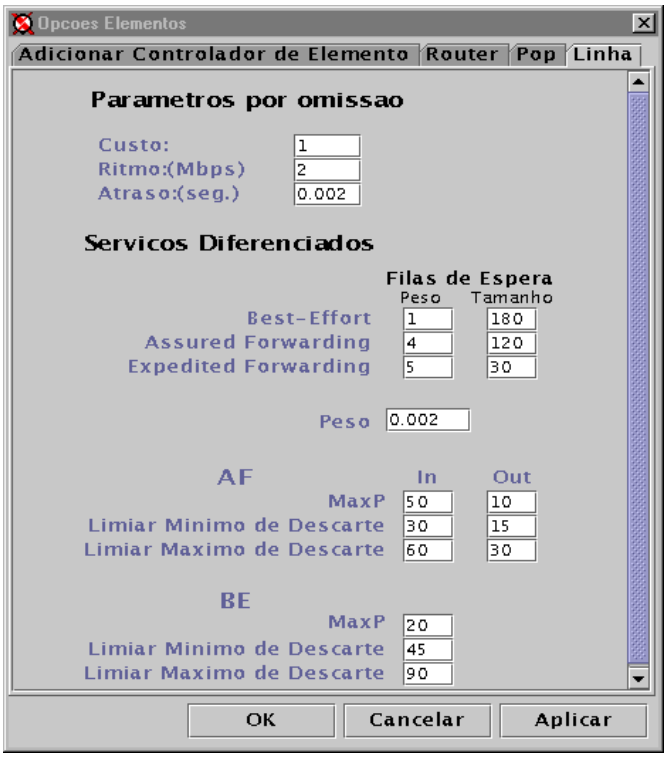

Fig. 2 Line configuration panel

The area on the right side of the screen of the Network Modeler application (figure 1) is the drawing area where the network topology is drawn by dropping the three mentioned components as needed. The bottom left window shows the events for the selected component (currently POP1), and the bottom right window shows the statistics requested for the selected component.

The toolbar in the top of the application includes the **Simular** button that produces the necessary OTcl code for *ns*, invokes *ns*, and retrieves the simulation statistics that can be presented by clicking the appropriate network components. These results are presented as graphics or text in separate windows. The **NAM** button invokes the network animator *nam* [7] that shows the packets flowing in the network.

III. EXAMPLE SCENARIO

In this section, a fictitious example scenario is analyzed with the Network Modeler application. A certain company needs to perform videoconferences between two of the company's buildings. However, the company's network manager noted that the quality being obtained is very poor, so he has to test alternative configurations, probably using differentiated services. The company has three buildings connected to each other by 2 Mbps lines. In each building there are local area networks that generate traffic to the networks of the other buildings. The network manager studied the problem and realized that the current configuration could be modeled in the Network Modeler as shown in figure 1. Each building is modeled by a POP that generates traffic, and a router that provides connections to the other buildings. First, the network manager analyzed the current situation, and then he studied alternatives to obtain good videoconference quality.

A. Scenario without DiffServ

The company's current situation does not use differentiated services. As the Network Modeler application uses differentiated services, the network manager configured all the traffic sources to use the same DiffServ traffic class, to simulate the current situation. The class that best serves this purpose is the EF class, as it does not use a random early detection mechanism. To simulate the videoconference, the network manager configured a constant bit rate (CBR) source at POP1 with 128 Kbps both ways to POP2. Then he configured background traffic at POP2 and POP3 using all possible application protocols: HTTP, FTP, Telnet, OnOff, and also CBR.

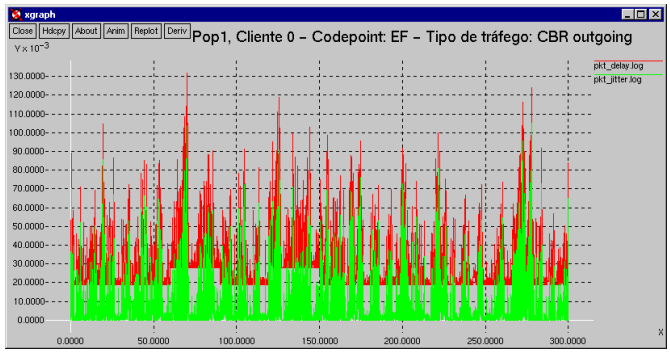

Fig. 3 Delay and jitter for traffic generated by client 0, Pop1

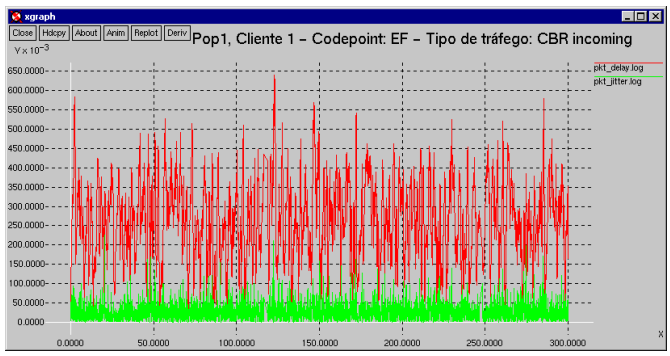

Fig. 4 Delay and jitter for traffic received by client 1, Pop1

The network manager performed a 5-minute simulation, requesting the statistics of the videoconference application at POP1. Additional simulation events were added to put line Linha1 down from time 60 to 90 seconds, and to increase its routing cost from 1 to 4 for times 120 to 150 seconds. Figure 3 shows the packet delay and jitter for the outgoing videoconference traffic, and figure 4 for the corresponding incoming traffic. The statistics for the incoming traffic are shown on figure 5. These results show large values for the delay (average 255 ms) and jitter (average 26 ms), that are

unacceptable to a good quality videoconference. Worse, there was some packet loss for this traffic, with about 3% packet loss ratio, which again is unacceptable. By analyzing the statistics from the other sources, the network manager concluded that the HTTP and FTP traffic was using a large fraction of the line bandwidth. Figure 6 shows the load on the line connecting POP1 and Router1 for both directions. The load for the incoming direction is almost always 1, meaning that the line is almost always transmitting data.

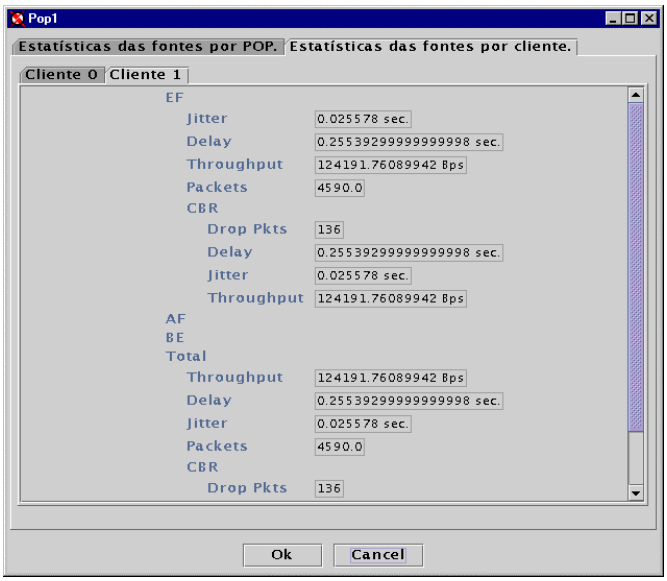

Fig. 5 Statistics for the sources at Pop1

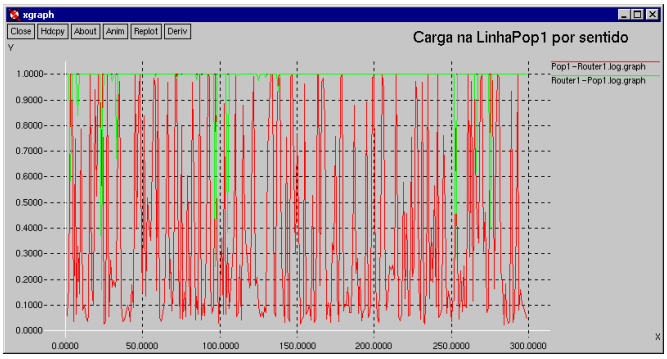

Fig. 6 Load for line LinhaPop1

As the TCP traffic of HTTP and FTP applications is mainly limited by the TCP slow-start mechanism, it is useless to increase the bandwidth of the lines connecting the company's buildings. The network manager concluded that the solution to having good quality videoconference might be using differentiated services technology in the company's network.

B. Scenario with DiffServ

The Differentiated Services architecture [1] aggregates traffic with similar QoS requirement in traffic classes that share the same per-hop-behavior (PHB) throughout the network. The border nodes implement packet classification and traffic conditioning functions, including metering, marking, shaping, and policing.

The EF (Expedited Forwarding) PHB [8] offers rigid QoS guaranties and may be used to implement services that require bounded delay and guaranteed bandwidth. Examples of these include circuit emulation, voice and video services. The AF (Assured Forwarding) PHB [9] on the other hand, offers limited QoS guarantees and may be used for applications such as Web access. The best-effort (BE) packet forwarding of current Internet is maintained for low priority and background traffic.

Policies are used to classify the user traffic into the available PHBs according to the QoS requirements for each user application. This is achieved by marking the traffic with the correct differentiated services code point (DSCP) for each application (port number and transport protocol), user (source or destination address or network), and time of day. Mechanisms such as COPS (Common Open Policy Service) [10] may be used to distribute such policies.

The network manager configured the videoconference traffic to use the EF traffic class, and the remaining applications to use the BE traffic class. The videoconference source had its traffic conditioners configured with a 128 Kbps rate profile, which is the required transmission rate.

The packet schedulers were configured as shown on figure 2. The EF queue weight is 5, and the total weights 10, assuring that 5 tenths of the bandwidth are available for EF traffic. The scheduling mechanisms used by the DiffServ *ns* patches [4] is the Weighted Deficit Round Robin. In this mechanism, the unused bandwidth of a traffic class is divided among the other traffic classes proportionally to their weights.

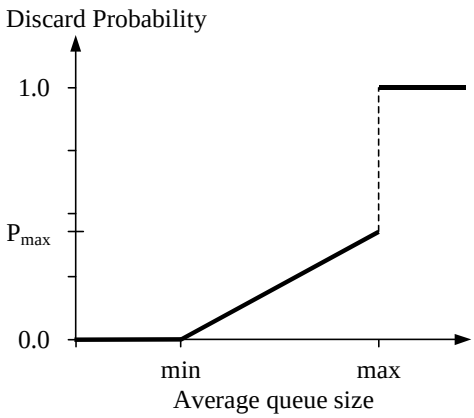

Fig. 7 RED packet discard ratio

The AF and BE queues have a random early detection (RED) [6] mechanism to detect and avoid congestion. This mechanism adjusts the packet discard probability as shown on figure 7. Its purpose is to start discarding some packets before the queue overflows, so that the TCP protocol detects the packet loss and reduces the transmission rate, thus reducing the probability of queue overflow. The average queue size is calculated by an exponential weighted moving

average to allow some traffic bursts. To allow a burst of size L without packet discard, the moving average weight wq should be calculated [6] so that the equation is verified:

$$L + 1 + \frac{(1 - w_q)^{L+1} - 1}{w_q} < min \qquad (1)$$

Additionally, the maximum threshold should be set to about twice the minimum threshold [6].

The AF queue has two sets of parameters, one to control the inside profile packet discard, and a more aggressive one to control outside profile packet discard. This is a RED mechanism with in/out bit (RIO). The parameters used in the simulation are shown on figure 2. These parameters allow 183-packet bursts in the AF class, and 227-packet bursts in the BE class without packet discard.

The simulation of the previous subsection was repeated with this DiffServ configuration. Again, simulation events were added to put line Linha1 down from time 60 to 90 seconds, and to increase its routing cost from 1 to 4 from time 120 to 150 seconds. The new results are shown on figures 8-11. The code produced by the Network Modeler application and fed to *ns* is shown in appendix. This code uses a separate library with most of the support functions. In this way, the code produced includes mostly parameter definitions and object instantiation.

The statistics obtained show that this configuration gives about 28 ms delay and 2 ms jitter with no packet loss for the videoconference traffic. These results are excellent for the videoconference application, even though line Linha1 remains heavily loaded.

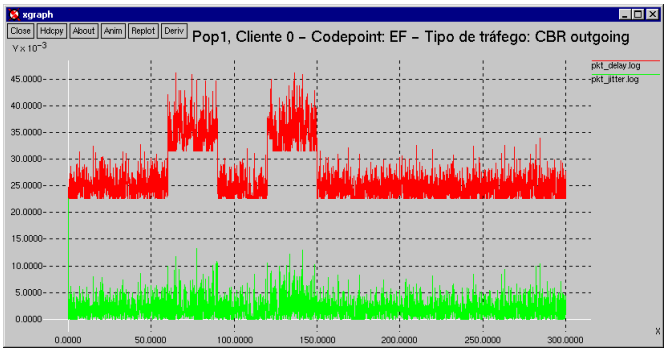

Fig. 8 Delay and jitter for traffic generated by client 0, Pop1

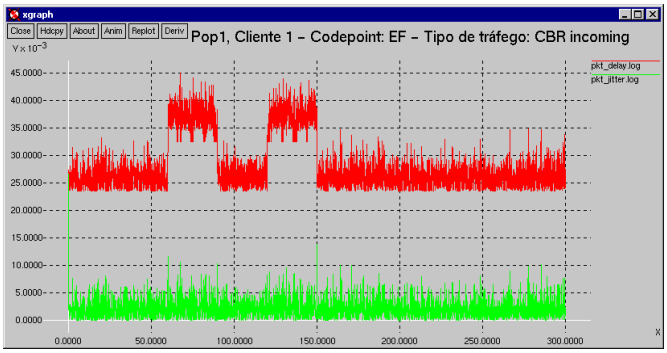

Fig. 9 Delay and jitter for traffic received by client 1, Pop1

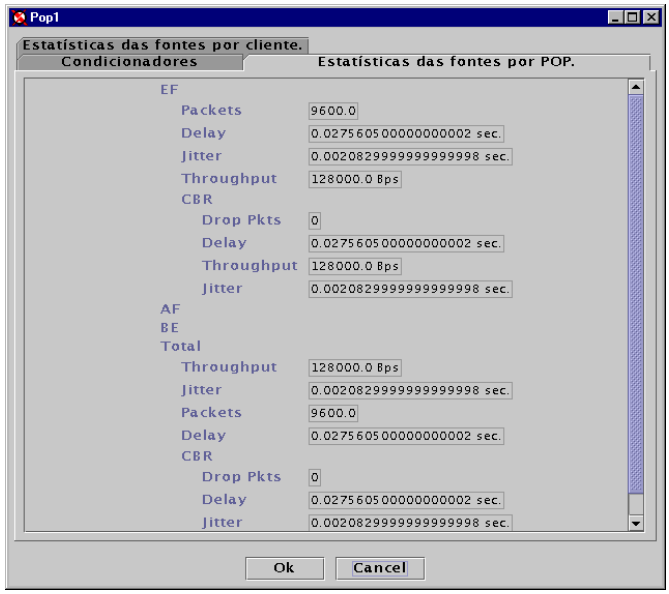

Fig. 10 Statistics at Pop1

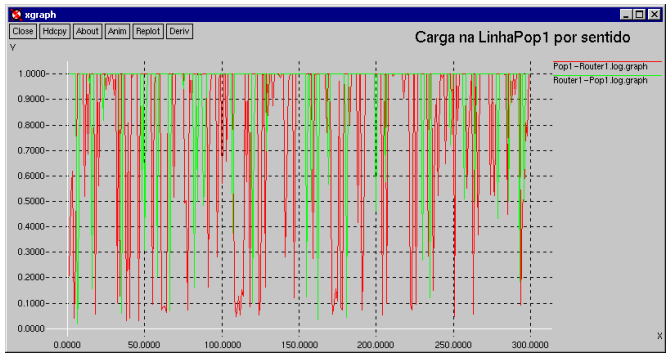

Fig. 11 Load for line LinhaPop1

Even when line Linha1 goes down, forcing the videoconference traffic to go through the other two lines, the results remain excellent. The network manager concluded that by using differentiated services in his network, he could have enough quality of service to support the new multimedia applications.

IV. CONCLUSION

The Network Modeler application, described in this paper, allows drawing, configuring and simulating differentiated services networks, permitting a user to assess the quality of service being obtained by network applications and also the network performance.

The Network Modeler is an easy to use network design and testing tool, especially useful for learning and testing different network configurations.

Some topics were left for further work. The POP needs some enhancements to allow specifying individual destinations for each traffic source. A bandwidth broker to perform connection admission control should be developed. Additionally, an enhanced policy mechanism should be studied. A possibility is [11] that could be integrated into the Network Modeler application.

V. REFERENCES

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, “An Architecture for Differentiated Services”, IETF RFC 2475, December 1998.

[2] Bruno Afonso, Daniel Gomes, “Simulação e Gestão de Redes”, Instituto Superior Técnico, Lisboa, Portugal, Relatório Final, Manual de Utilizador, Manual de Desenvolvimento, Trabalho Final de Curso, 2000.

[3] UCB/LBNL/VINT Network Simulator (version 2). http://www.isi.edu/nsnam/ns/

[4] DiffServ additions to NS. http://www.teltec.dcu.ie/~murphys/ns-work/diffserv/index.html

[5] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994. ISBN: 0-201-63337-X.

[6] Sally Floyd, Van Jacobson, “Random Early Detection Gateways for Congestion Avoidance”, *IEEE/ACM Transactions on Networking*, **1**(4):397-413, August 1993. ftp://ftp.ee.lbl.gov/papers/early.pdf

[7] UCB/LBNL/VINT Network Animator. http://www.isi.edu/nsnam/nam/

[8] J. Heinamen, F. Baker, W. Weiss, J. Wroclawski, “Assured Forwarding PHB Group”, IETF RFC 2597, June 1999.

[9] V. Jacobson, K. Nichols, K. Poduri, “An Expedited Forwarding PHB”, IETF RFC 2598, June 1999.

[10] J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry, “The COPS (Common Open Policy Service) Protocol”, IETF RFC 2748, January 2000.

[11] Paulo Pereira, Djamel Sadok, Paulo Pinto, “Service Level Management of Differentiated Services Networks with Active Policies”, in *ConfTele’2001 proceedings*, Figueira da Foz, Portugal, April 2001.

VI. APPENDIX

This appendix shows the *ns* code produced by the Network Modeler application for the scenario described in section III-B.

```
set fichNAM /home/dhgo/tfc2000/Cenario1/TFC-
  Cenario1.nam
set directoria /home/dhgo/tfc2000/Cenario1
source /home/dhgo/tfc2000/networkmodeler/main.tcl
set end 300
source /home/dhgo/tfc2000/elementos/Router.tcl
$ns rtproto Session
source  /home/dhgo/tfc2000/elementos/pop.tcl
POP set pop_nodes 3
POP set packetsize 1000
POP set telnetPacketsize 1000
source /home/dhgo/tfc2000/elementos/Linha.tcl
Linha set ef_weight 5
Linha set af_weight 4
Linha set be_weight 1
Linha set ef_length 30
Linha set af_length 120
Linha set be_length 180
Linha set qweight 0.0020
Linha set maxPAfIn 50
Linha set maxPAfOut 10
Linha set minAfIn 30
Linha set minAfOut 15
Linha set maxAfIn 60
Linha set maxAfOut 30
Linha set maxPBe 20
Linha set minBe 45
Linha set maxBe 90
set Router1 [newNode "Router1"]
set Router2 [newNode "Router2"]
set Router3 [newNode "Router3"]
set Linha1 [new Linha "Linha1" Router1 Router2 2.0Mb
  0.0050 1.0 1 ]
set Linha2 [new Linha "Linha2" Router1 Router3 2.0Mb
  0.0050 1.0 1 ]
set Linha3 [new Linha "Linha3" Router2 Router3 2.0Mb
  0.0050 1.0 1 ]
newPOPnode popNode(0) "Pop1"
set pop(0) [new POP "Pop1" $ns 0 128kb 1 300 2 0 1.0 1.0
  0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0 10Mb 64kb 64kb 1 ]
newPOPnode popNode(1) "Pop2"
set pop(1) [new POP "Pop2" $ns 1 128kb 1 300 12 0 1.0 3.0
  1.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 10Mb 64kb 64kb 1 ]
newPOPnode popNode(2) "Pop3"
set pop(2) [new POP "Pop3" $ns 2 128kb 1 300 12 0 1.0 3.0
  1.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 10Mb 64kb 64kb 1 ]
set LinhaPop1 [new Linha "LinhaPop1" popNode(0) Router1
  2.0Mb 0.0010 1.0 1 ]
set LinhaPop2 [new Linha "LinhaPop2" popNode(1) Router2
  2.0Mb 0.0010 1.0 0 ]
set LinhaPop3 [new Linha "LinhaPop3" popNode(2) Router3
  2.0Mb 0.0010 1.0 0 ]

$Linha1 downLink 60 90
$ns at 120 "$Linha1 setCost 4.0"
$ns at 150 "$Linha1 setCost 1.0"
$ns at 0 "$pop(0) start 1.0 300"
$ns at 300 "$pop(0) stop"
$ns at 0 "$pop(1) start 1.0 300"
$ns at 300 "$pop(1) stop"
$ns at 0 "$pop(2) start 1.0 300"
$ns at 300 "$pop(2) stop"
$ns at $end "$Linha1 mostraStat 1"
$ns at $end "$Linha1 mostraStat 2"
$ns at $end "$pop(0) mostraStat 1"
$ns at $end "$pop(0) mostraStat 2"
$ns at $end "$pop(1) mostraStat 1"
$ns at $end "$pop(2) mostraStat 1"
$ns at $end "$LinhaPop1 mostraStat 1"
$ns at $end "$LinhaPop1 mostraStat 2"
$ns at [expr 1 + $end] "end"
puts "SIMULATION STARTED"
$ns run
```