

CSIM for Java User Story: **An Experiment with Integrated and Differentiated Network Services: Queue Scheduling and Simulation with CSIM for Java**

By Thomas Hogarty, George Mason University



Introduction

IP-based networks are quickly becoming a preferred medium for services that demand high quality connectivity. These types of services include Voice Over IP (VOIP), streaming or on-demand video, video conferencing, and online gaming. Each of these demanding services has unique priorities, such as the low jitter and low delay needed for VOIP or video conferencing. Streaming and on-demand video requires high bandwidth and can tolerate intermittent delay. Online gaming demands low delay to keep players synchronized or “not lagging.”

In order to meet these demands, Internet Service Providers (ISPs) and network backbone carriers need solutions that allow them to guarantee varying levels of service to customers who pay for it. This project is an investigation into the quality of service (QoS) capabilities provided by selected queuing algorithms.

The goal of this simulation was to model the performance of integrated and differentiated services for QoS in a sample network, while varying the average rate of connections, number of concurrent connections, presence or absence of admission control for reserving network bandwidth, and connection types. I used CSIM for Java to simulate the network, with some of these factors varying within a run, and others changing between different runs.

Network Configuration, Queuing Algorithms, and Simulated Variables

This project investigated the quality of service (QoS) capabilities provided by two architectures in a single network: the Integrated and Differentiated Services architectures (Int-Serv and Diff-Serv respectively). Each architecture utilizes a specific queuing algorithm. In my Int-Serv architecture, I used the Weighted Fair Queuing (WFQ) algorithm for the Int-Serv routers. In the this algorithm, each connection entering the network is considered separately when making the decision of which packet to send next from a router.

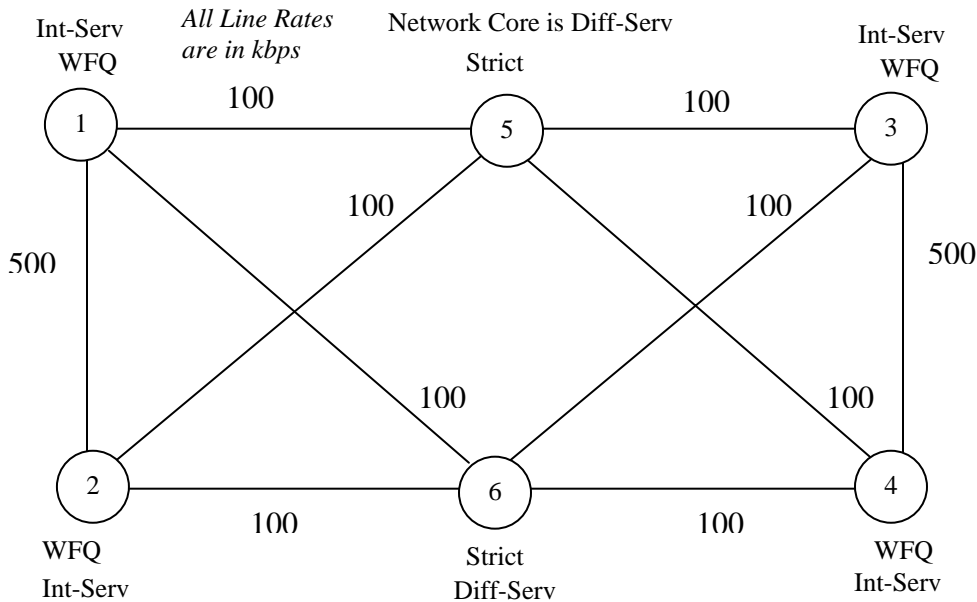
Application Area:
Networking

Platform:
Java

CSIM’s Challenge:
High-speed simulation of network traffic using Integrated and Differentiated network architectures, and different algorithms for packet-queuing



The Diff-Serv architecture uses a simple Strict priority scheduling algorithm, in which connections are aggregated into logical flows based on pre-determined priority levels. When traffic hits the network, packets with highest priority are served in first-in-first-out order. Medium priority packets are addressed when no high priority packets remain; and finally, if no high or medium priority packets exist, low priority packets are sent. My network configuration used a combination of both Int-Serv and Diff-Serv architectures, with the Diff-Serv structures at the core of the network, as shown in Figure 1.



[Figure 1: Physical Network Layout]

Regardless of queuing algorithms, quality of service in a network cannot be maintained if traffic sources are allowed to send data that exceeds the capacity of some link along the route to a destination. In order to avoid this problem, we used two complimentary methods. The first component is *token bucket conformance*, which forces each connection entering the network to conform to a prescribed average rate, maximum rate, and maximum burst size. The second component is *admission control*, which ensures that the network has enough capacity to support the average rate of each accepted connection. In this experiment, an admission-controlled network is called a *reserved* network; a network that accepts any connection is an *open* network.

Token bucket conformance and admission control, combined with high-quality queue scheduling algorithms, should allow a network provider to serve each customer with sufficient quality based on the amount paid and the service level agreed upon.

“With CSIM for Java, I was able to a model a very complex, multi-variable, and high-traffic network.”

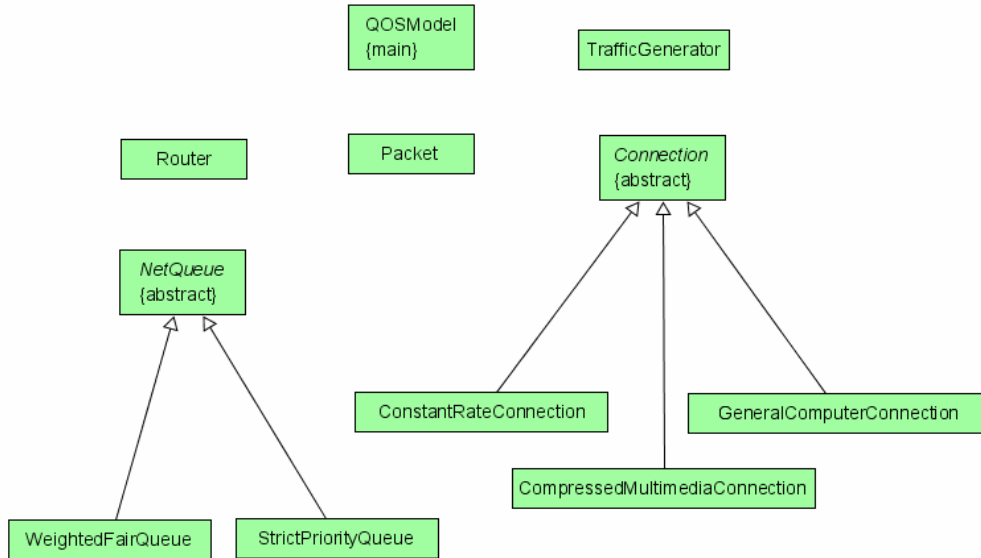
With CSIM for Java, I was able to model a very complex, multi-variable, and high-traffic network. I modeled network performance over a series of three runs. Each run simulated multiple values for several different variables, including number of concurrent connections, average connection rate, and allowed variance (range of values) of the number of connections. The inputs for each run were chosen to produce Goal Average Queue Sizes of Low, Medium, and High, respectively.



I also varied the *connection type*, which describes the type of data being sent. Possible types were *Constant*, *Multimedia*, or *General*. *Constant* data has a fixed packet size and even transmission intervals; *Multimedia* data streams at a constant, high rate; and *General* data simulates web browsing and email traffic, composed of random bursts of data. During each run, simulated traffic was inserted into two identical networks: one with admission control (reserved), and one without (open).

Simulation and Design Methodology

The network design was separated into components to better fit with Java's object-oriented structure and to keep the code modular. The important components and class hierarchy are pictured here in Figure 2.



[Figure 2: UML Class Diagram]

Each Router is capable of having any number of abstract NetQueues (WFQ or Strict in this project); one for each out-going link. When a packet arrives at a router, the router determines the next hop and passes the packet to the queue serving the next hop link. Each queue services packets according to its implementation algorithm; in this project the queues are work-conserving, so

whenever a packet is in the queue, there will be a packet transmitting on the outgoing link.

Static routing is used to keep route paths consistent for admission control bandwidth reservations. Static routing also made the experiment code easier to implement.

There is a TrafficGenerator associated with each source router (all four Int-Serv routers in Figure 1). The generator uses the mean arrival time parameter from a model input file provided when a user starts the simulation to select when a new connection arrives. The generator also selects what type the connection is {Constant, Multimedia, General} and to which destination the packets will be sent. The selection of connection type is uniform with equal probability. The selection of destination is uniform with the following probabilities: 2/3 of connections from a source go to the local destination (across high-speed link), 1/3 of connections go to the remote destinations (1/6 probability to each remote Int-Serv router as pictured in Figure 1).

When a connection is initiated by the TrafficGenerator, the connection selects its own rate and packet parameters randomly, based on values from the input file. Each parameter may have a different distribution. A separate random number stream is used for each parameter selection at a source to make sure the sequence will correctly match the requested distributions (uniform, exponential, etc.). Connections are the processes that generate packets and deliver them to the source router. The source router then gives the packet to the queue for next hop and the queue delivers the packet to the next router as scheduled by the queuing algorithm. The process continues until the packet reaches its destination.

Statistics are collected throughout the simulation using the CSIM Table, QTable, and Facility features.

Token bucket compliance is built into each connection type. The connections generate packets that conform to the randomly-selected mean rate, maximum rate, and maximum burst size.

The QOSModel component creates the simulation and tracks global state. It handles admission control and maintains a sum of all accepted connection rates on each link. If a new connection arrives that needs more bandwidth than is available on the path from source to destination, the connection is rejected. If a connection is accepted, its rate is added to the sum for each link on its path by the QOSModel. When the last packet in the connection arrives at the destination, the connection frees up resources in the network by asking the QOSModel to remove its



“This project was my first program using discrete time simulation. I was pleased that I could come up to speed quickly. The documentation was very helpful, and it only took a week or two to get familiar with most of the features I wanted use.”

reservation. After the reservation is removed, the bandwidth is then available for newly arriving connections to use.

In order to experimentally view the need for and benefits of admission control, the simulation contains two identical networks. One uses admission control to prevent congestion, and the other admits all arriving connections. The same set of traffic generators and connections are used for both networks so that the results are based on the same exact traffic data set. The QOSModel is responsible for “injecting” packets generated by connections to the appropriate network(s) based on whether the connection has been accepted or rejected in each network. In the case of our experiment, every packet is delivered to the *open* network (no admission control), and packets are only delivered to the *reserved* network (with admission control) if the connection has been admitted.

Results with CSIM for Java

The main purpose of this simulation was to observe the behavior of multiple variables. The resulting log files are detailed and not included here, though I show a snapshot below.

The simulations in this experiment were conducted with the time unit of *seconds* and the data size unit of *bits*. The following table shows only a small portion of the detailed statistics generated in the simulation output file of the program, including confidence intervals and/or histograms for many parameters. The table lists overall queue size and connection reservation statistics for three sample runs with Goal Average Queue Size of Low, Medium, and High respectively. The medium congestion run has the worst queue size and reservation results because the input parameters for that run allowed for a much wider range of rates and thus allowed more congestion to occur during busy times in the open network.

Goal Avg Queue Size	Open: Avg Queue Size	Reserved: Avg Queue Size	Reserved Connections
Low	0.1581	0.1576	0.9999
Medium	287.7	0.8848	0.9045
High	182.3	1.8769	0.9472

[Table 1: Overview Statistics]

In this table, “Reserved Connections” denotes the fraction of connection requests that were admitted to the network. In an open network, all connections are admitted, for a value of 1.0.

The most obvious result was the effect of admission control on the network. If admission control is not employed (open network), too much traffic gets into network, exceeding the available bandwidth on bottleneck points. The queue becomes extremely large, as shown in Table 1, and the network becomes overly congested, causing intolerable end-to-end delay. The results with admission



“CSIM has many useful features, and I found several to be particularly helpful, such as the statistics tools and reporting features (QTable, Table, Facility, Histogram, Confidence Interval, Random streams), and the discrete time and event simulation features (hold, use, wait, set, etc.).”

control (reserved network), show a much shorter average queue for medium and high-traffic situations. While I expected this behavior, the simulation confirmed it with clear evidence.



The following table lists end-to-end queuing delay as observed from the same sample runs used to generate Table 1.

Destination Type	Goal Avg Queue Size	Open	Reserved	Open	Reserved	Open	Reserved
		Constant		Multimedia		General	
Local	Low	.00119	.00119	.00118	.001183	.00095	.00095
Remote		.01120	.01118	.01121	.011197	.02195	.02188
Local	Med	.01197	.00624	.01174	.006145	.00880	.00352
Remote		14.566	.03261	16.164	.031311	12.598	.07340
Local	High	.00695	.00695	.00690	.006904	.00371	.00371
Remote		7.9792	.06524	9.0769	.063121	8.2806	.12250

[Table 2: End-to-End Queuing Delay]

Useful Features in CSIM for Java

This project was my first program using discrete time simulation. I was pleased that I could come up to speed so quickly. The documentation was helpful, and it only took a week or two to get familiar with most of the features I wanted use.

CSIM has many useful features, and I found several to be particularly helpful, such as the statistics tools and reporting features (QTable, Table, Facility, Histogram, Confidence Interval, Random streams), and the discrete time and event simulation features (hold, use, wait, set, etc.).

CSIM for Java is an excellent tool for discrete time simulation and statistics-gathering, and I'd definitely recommend it for Java developers.

Thomas Hogarty used this project to help secure his Masters in Computer Science from George Mason University. He is now a Systems Administrator and Network Engineer at Integrity Applications Inc., in the Washington D.C. area.

To learn more about the project and findings, or to see the code used, please contact Mr. Hogarty directly at thogarty@gmualumni.org.

For more information about CSIM for Java or to try it out (free), please contact Mesquite Software:

Mesquite Software
 8500 N. MOPAC Expwy, Suite 825
 Austin, TX 78759 USA
 Phone: +1 (512) 338-9153 or 1 (800) 538-9158
 Email: info@mesquite.com
 WWW: <http://www.mesquite.com>

“CSIM for Java is an excellent tool for discrete time simulation and statistics-gathering, and I'd definitely recommend it for Java developers.”