[redacted]

“Priority Queueing to Alleviate Packet Re-ordering Problem”

[redacted]

[redacted]

Table of Contents

Table of Figures

Table of Tables

1.0 Problem Description

The goal of the project is to simulate an Internet router that uses priority queuing of out-of-order packets. The desired outcome is to demonstrate that the rate at which packets arrive at their destination out-of-order is reduced. This project utilizes Omnet++ to model the system in question and to try and collect enough statistics to prove or disprove this claim.

2.0 Priority Queuing Model

In the model there are three nodes: A sender, router, and receiver.
There are two channels: one between the sender and router, and the other between the router and the destination. Each of these channels has a rate of 1Mbps. This is purely in place to limit the rate at which the bits can be sent out of the router and the sender nodes. This should take roughly 8e-3 seconds (based on a message size of 1000 Bytes). The additional delay experienced by the packet is applied on top of this delay, and it is assumed that during this period another packet (or packets) can be sent in to the channel. The additional delay for the sender-router channel is based on a normal distribution with mean x, and standard deviation y. The second channel has a fixed 1 second delay.

The sender generates poisson traffic, which means that it generates a new message based on an exponential interarrival time. The exponential distribution is based on a mean of 1/40 seconds per packet. Because of the randomness introduced by the distribution, a network buffer must be in place to ensure that the channel is not overwhelmed with traffic. (Parameter for exponential distribution mean = 0.025 seconds).

The router has two queues which it uses to buffer network traffic, before it sends it on to the receiving node. The router inspects the incoming packets, and maintains a counter with the largest sequence number it is observed. The queues are filled by comparing the incoming packet's sequence number with the counter. If the packet's is less than the counter then it is out-of-order and should enter the fast queue. If the packet's is greater than the counter then it is in-order and should enter the slow queue. However, if both queues are empty then the packet is forwarded on immediately. Packets are dequeued and sent out one-by-one as the channel becomes available.

The receiver simply receives the messages, and does no further function. (In the simulation it is used to collect statistics, but this is auxiliary to the primary model).
The following diagram summarizes the network that will be simulated.

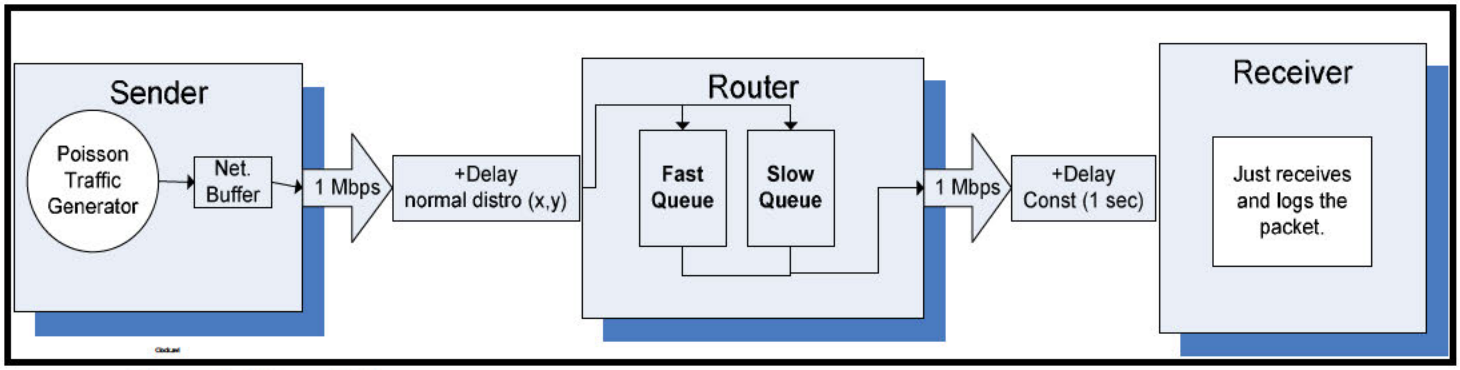

Figure 1 - Network Diagram

3.0 Simulation Goals & Parameters

Most of the parameters were statically set to the values defined in the previous section. However, there were three parameters which were changed to test the system.
The first two were the inputs to the normal distribution, used to add additional delay to the packets traveling from the source to the router. Parameter x is the mean of the distribution, and y is the standard deviation.
They were tested with the following suggested combinations (5 runs for each pairing):

x	y
1	0.05
1	0.5
1	1
2	0.05
2	0.5
2	1
3	0.05

The other parameter which was varied for the simulation was the maximum sequence number (ie. the number of packets used for each run). Two sets of tests were done: 1000 packets, and 100,000 packets. Due to time constraints, and the limitations of the host computer, larger runs could not be conducted.
Queue lengths were set to infinite for this simulation, though the resulting maximum queue length values were monitored, and never exceeded 8 for any of the runs. In theory this could be used as a baseline for future designs, though longer runs should probably be tested before settling on a hard limit.

There were two primary statistical goals to the simulation:
The first goal was to measure the out-of-order rate of each packet during a run, obtain the average, and display this information once the simulation has completed. An out-of-order packet is defined as one which has any lower sequence numbered packets arrive after it. The rate is therefore defined as the number counted out-of-order divided by the total number of packets sent during a run.
The second statistic is the average packet delay. This is defined as the simulation time when the packet arrives at the receiver minus the time it was created. Therefore the average is easily obtained by summing all these delays and then dividing by the total number of packets that were sent during the run.

Additionally, the overarching goal was to see how the system behaved when different delays were applied to the packets, and to compare the performance to a system which did not use the priority queuing router.

4.0 Methodology

Initial familiarization with the simulator was done by following the tic-toc example, as linked from the class website. After completing the first portion of the tutorial it became apparent that the requirements of the simulation could be met with only minor tweaks to the tic-toc template. It was unnecessary to use the additional (Queuenet & Queuelib) libraries mentioned in the project's helpful code section, as there would have been more overhead with learning their functionalities as well. Therefore, the approach was to code the cc,msg, ned, and other files directly within the eclipse text editor. This approach is outlined in the following subsections.

4.1 How OMNET++ is used to translate the model?

The simulation was comprised of six source files. Three defined the nodes in the network, one defined the message type used to represent the packets, one defined the network itself, and finally a simulation parameters file.
The files are listed below (all are included, in full, in Appendix B):

- Sender.cc
- Router.cc
- Receiver.cc
- MyMessage.msg
- all.ned
- omnetpp.ini

The Sender class extends the cSimpleModule class included by omnet. The sender is coded to get the following input parameters from the ini file:

- Maximum sequence number
- Interarrival time
  (distribution based, therefore sampled after every message creation to get time of next message creation)
- Packet size/length (in bytes)

Its' primary data object is a packet queue which buffers outgong packets.
The basic algorithm employed by the Sender is to keep sending two types of messages to itself, each triggering a different event. The first message is sent to trigger the creation of the next packet in the sequence. When the packet is created it is setup with the current sequence number (the sender then increments its' copy of the sequence number). Once it is created the packet is sent out immediately, if the channel is not in use. If the channel is busy then it is queued in the network buffer. The sender then polls the delay distribution to figure out when to schedule the creation of the next packet. This process repeats until the maximum number of packets has been generated.
The second message that is repeatedly sent to itself is the one for determining if the channel is free. When this message comes in it pops the next packet off of the network buffer queue and sends it out. It then checks the arrival time of that packet and schedules the next channel free message to trigger at that time.

The Router class also extends cSimpleModule. It uses the following input parameters from the ini file:

- Sender channel delay

  *This was specified as a normal distribution, but the simulator is forced to use the truncated normal distribution so that non-negative values are not obtained. This is a limitation of adding the delay on top of the channel transmission time.

- Queuing policy (used to turn priority queue off, for 2nd set of tests)

Its' primary data objects are two packet queues (labeled fast and slow), and a counter for tracking the largest sequence number it has seen so far.
When a packet arrives at the router it checks its' sequence number and compares it to the largest number. If the packet's number is less than the largest number it is out-of-order and is added to the fast queue. Otherwise it is added to the slow queue and the largest sequence number is updated to the new value.
To send from the two queues the router follows a pattern similar to the sender's network buffer: a message is sent to itself to trigger a channel free event whenever a message can be sent out. When the message comes in it first tries to pop a message from the fast queue, and if this is not possible it tries to pop one from the slow queue. If both are empty then the router waits for a new message to come in and become queued.

The Receiver class also extends cSimpleModule. Its' purpose is to receive messages from the router channel, and apply an additional constant 1-second delay, by sending the message to itself. (This is done using the scheduleAt() method). When the message comes in for a second time it is counted as actually being received, and the receiver collects the appropriate statistics (as discussed in section 4.3). The module's only other job is to free the packets once this process has completed, and to print out the statistics when the run has completed.

The MyMessage class is defined using the .msg format of omnet++. This means that it is written in a form of pseudo code, and the simulator compilation tools generate the corresponding .cc and .h files. In this case the MyMessage class extends the packet class and adds one additional variable. This variable is an unsigned integer which corresponds to the sequence number of the packet.

The all.ned file defines how the network is setup. It describes which modules are used in the network, which parameters they have, how the modules are interconnected, and what type of channels are used for those interconnections. The network topology was already shown in detail, in the queuing model section, and the all.ned file is written to implement that layout. Therefore if more detail is required the file can be reviewed in the appendix.

The omnetpp.ini file is where all the simulation parameters are defined. There are entries for turning on/off the priority queue, the interarrival distribution for the sender, the packet length, maximum sequence number, extra channel delay distributions, and the seed used to run the simulation. This file was designed to support the five different seeded runs required by the project. The seed is selected at run time from the ini file, as shown in the picture below.

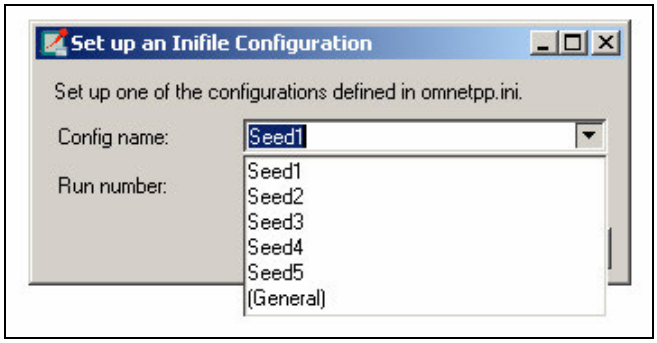

Figure 2 - Seed selection

This begins to lead in to how the simulations were run, so the full details will be covered in the following subsection.

4.2 How did you setup the simulations?

The following sets of simulations were run:

- Initial testing was conducted on small runs of <100 packets, and the debug statements in the code were turned on so that the contents of the queues could be verified. Additional runs were done to check that the simulation operated correctly at the extremes (eg. x=0,y=0, etc.). Once the proper operation was verified as much as possible the following sets of runs were conducted.
- A total of 45 runs were done using 1000 packets per run. This was to test out the system and to collect some initial data. Also, at this point the system was simulating much slower and it didn't appear to be possible to faster runs. *It was decided that the variation for the statistics were low enough at this point that a warm-up period would have a minimal effect, and was therefore not used for the larger runs.
- A total of 45 runs were done using 100,000 packets per run. This was used for the calculations and graphs. These were done using the "Express Run" mode, as illustrated in figure 3 below.
- A total of 45 runs were done using a single-queue router, with 100,000 packets per run.
- Another 90 runs were conducted at the 100,000 level to get the additional distance squared statistic from the simulation. This approach will be discussed further in the analysis section.

Each simulation was setup using a combination of the ini file, and the simulator gui. The ini file specified which router queuing model to use, and which parameters the normal distribution would use for the delay. The gui was then used to select between the different seeds for a run (this was illustrated at the end of the previous section).

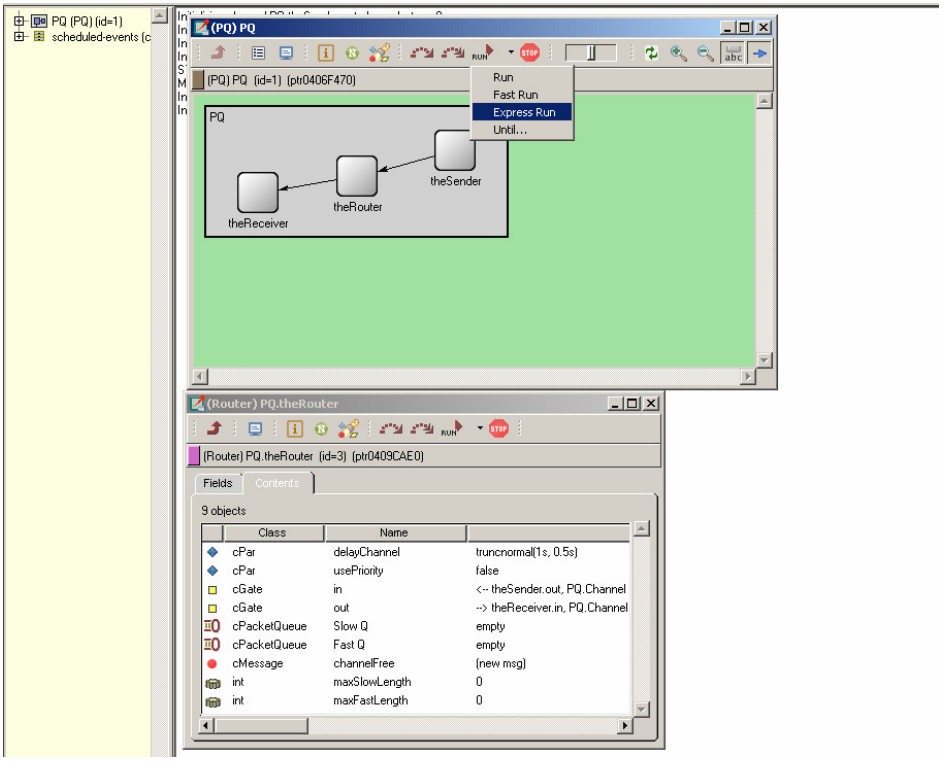

Figure 3 - Starting simulation

4.3 How did you collect the stats?

The statistics collection was done in two parts. The first part was done internal to the simulation, while it was running, and the second part was done using Microsoft Excel.

To collect the statistics during the simulation the receiver kept two arrays of values, each with enough room to record the information for every single packet. One array was indexed by the arrival position, with each position containing the packet sequence number that arrived at that time. This was used to calculate the out-of-order rate for each of the packets using the following algorithm:

1. Go to position i in the array and note the current sequence number,
2. Check that all subsequent positions contain only sequence numbers larger than the position's sequence number.
3. If this test fails, increment an out of order counter.
4. Increment i and go to step 1, until all positions have been checked.
5. Divide the counter by the total number of packets to get the rate.

The second array held the delay associated with each packet. This was used to calculate the mean packet delay. This operation involved summing up the delays for all the packets and dividing it by the total number of packets.

The two averages from the simulation run were then manually copied in to Excel so that they could be organized and analyzed further. Once in Excel the results from the runs were used to get the averages for each set, the sample variance, and the confidence interval. These numbers were then used to create the graphs required for the analysis. Formulas for calculations were straightforward, and based off the material covered in class and in the text book.

*The excel spreadsheet used is available upon request.

5.0 Results & Analysis

This section is broken in to three subsections. The first subsection contains all the data obtained and calculated from runs with both 1000 packets and 100,000 packets. This includes the run averages for the out-of-order rate and packet delay, as well as the mean, variance, and confidence across each set of runs. These tables are followed by several graphs which help to illustrate the behavior of the data, and are followed by the initial analysis.

The second subsection contains all the data and graphs for the 100,000 packet runs done on the single queue (non-priority) router. This section also includes the corresponding analysis, which compares this design to the priority queuing router.

The third and final subsection includes an additional statistic, the distance to proper order, squared. It was collected for all 90 of the 100,000 packet runs, and was added (and the simulations were re-run) after collecting the original data, because the out-of-order rate did not appear to reflect any improvements with the priority queuing system. The results and approach are elaborated further in the third subsection.

5.1 Priority Queue Data

Table 1 - Priority Queue Data

X=1,y=0.05	**1000 packets**		**100k packets**	
Seed	out-of-order rate	average packet delay	out-of-order rate	average packet delay
1	0.534	2.01688	0.52453	2.01973
2	0.519	2.01868	0.5241	2.01996
3	0.543	2.01593	0.525	2.02006
4	0.526	2.02238	0.5238	2.02012
5	0.513	2.02032	0.52306	2.02
Mean:	0.527	2.018838	0.524098	2.019974
Var:	0.0001415	6.76432E-06	5.4162E-07	2.228E-08
Confidence +/-	0.01042656	0.002279685	0.000645075	0.000130834

1,0.5	**1000 packets**		**100k packets**	
Seed	out-of-order rate	average packet delay	out-of-order rate	average packet delay
1	0.9	2.05681	0.90382	2.04908
2	0.898	2.05976	0.90246	2.04741
3	0.885	2.03736	0.90438	2.0512
4	0.901	2.0468	0.90432	2.04906
5	0.886	2.04548	0.90333	2.04738
Mean:	0.894	2.049242	0.903662	2.048826
Var:	6.15E-05	8.23002E-05	6.3212E-07	2.46278E-06
Confidence +/-	0.006873855	0.007951761	0.000696887	0.001375548

1,1	**1000 packets**		**100k packets**	
Seed	out-of-order rate	average packet delay	out-of-order rate	average packet delay
1	0.947	2.31591	0.94477	2.30826
2	0.94	2.29821	0.94426	2.30961
3	0.944	2.30817	0.94488	2.31093

4	0.939	2.34014	0.9444	2.31201
5	0.928	2.33166	0.94373	2.3069
Mean:	0.9396	2.318818	0.944408	2.309542
Var:	5.23E-05	0.000291518	2.0887E-07	4.16147E-06
Confidence +/-	0.006338897	0.014965639	0.000400591	0.001788077

2,0.05	**1000 packets**		**100k packets**	
Seed	out-of-order rate	average packet delay	out-of-order rate	average packet delay
1	0.534	3.01688	0.52453	3.01973
2	0.519	3.01868	0.5241	3.01996
3	0.543	3.01593	0.525	3.02006
4	0.526	3.02238	0.5238	3.02012
5	0.513	3.02032	0.52306	3.02
Mean:	0.527	3.018838	0.524098	3.019974
Var:	0.0001415	6.76432E-06	5.4162E-07	2.228E-08
Confidence +/-	0.01042656	0.002279685	0.000645075	0.000130834

2,0.5	**1000 packets**		**100k packets**	
Seed	out-of-order rate	average packet delay	out-of-order rate	average packet delay
1	0.902	2.99441	0.90419	3.01993
2	0.9	3.00895	0.90261	3.01912
3	0.898	2.98134	0.90419	3.02278
4	0.908	3.04164	0.90547	3.02054
5	0.892	3.027	0.90391	3.02083
Mean:	0.9	3.010668	0.904074	3.02064
Var:	3.4E-05	0.000588351	1.03648E-06	1.86005E-06
Confidence +/-	0.005110958	0.021260875	0.000892367	0.001195433

2,1	**1000 packets**		**100k packets**	
Seed	out-of-order rate	average packet delay	out-of-order rate	average packet delay
1	0.951	3.09377	0.94537	3.07843
2	0.947	3.10007	0.94443	3.07507
3	0.935	3.05415	0.94654	3.08265
4	0.942	3.07297	0.94548	3.07837
5	0.927	3.07168	0.94549	3.07491
Mean:	0.9404	3.078528	0.945462	3.077886
Var:	9.18E-05	0.000342112	5.5917E-07	1.00031E-05
Confidence +/-	0.00839816	0.016212398	0.000655443	0.002772233

3,0.05	**1000 packets**		**100k packets**	
Seed	out-of-order rate	average packet delay	out-of-order rate	average packet delay
1	0.534	4.01688	0.52453	4.01973
2	0.519	4.01868	0.5241	4.01996
3	0.543	4.01593	0.525	4.02006
4	0.526	4.02238	0.5238	4.02012
5	0.513	4.02032	0.52306	4.02

Mean:	0.527	4.018838	0.524098	4.019974
Var:	0.0001415	6.76432E-06	5.4162E-07	2.228E-08
Confidence +/-	0.01042656	0.002279685	0.000645075	0.000130834

3,0.5	**1000 packets**		**100k packets**	
Seed	out-of-order rate	average packet delay	out-of-order rate	average packet delay
1	0.902	3.99441	0.90353	4.01965
2	0.9	4.00895	0.90388	4.02136
3	0.898	3.98134	0.90506	4.02251
4	0.908	4.04164	0.90483	4.02251
5	0.892	4.027	0.90281	4.02236
Mean:	0.9	4.010668	0.904022	4.021678
Var:	3.4E-05	0.000588351	8.6537E-07	1.51587E-06
Confidence +/-	0.005110958	0.021260875	0.000815387	0.00107918

3,1	**1000 packets**		**100k packets**	
Seed	out-of-order rate	average packet delay	out-of-order rate	average packet delay
1	0.959	4.01279	0.94501	4.02553
2	0.945	4.02081	0.94573	4.02242
3	0.945	3.96576	0.94596	4.0291
4	0.947	4.0632	0.94625	4.02789
5	0.931	4.05514	0.94578	4.02767
Mean:	0.9454	4.02354	0.945746	4.026522
Var:	9.88E-05	0.001508255	2.1073E-07	6.91147E-06
Confidence +/-	0.00871247	0.034040834	0.00040237	0.002304348

There are a few key results that jump out of this data:

1. A) The variance is greatly improved (lower) in the 100k runs, and therefore there is greater confidence in the longer runs.
1. B) The confidence intervals are relatively small for both statistics, but the average packet delay has almost no variation.

2. A) The standard deviation (or y value) of the normal distribution has the biggest effect on the out-of order rate.
2. B) The average packet delay is consistent between x=1,2, or 3. In other words, the averages jump by the expected average value of the distribution.

3. The best performance, in terms of out-of-order rate, is for y=0.5 runs

4. Some results are nearly identical between runs, which seems odd at first inspection, but can be justified and explained.

These points are illustrated below, and further comments are made if necessary.

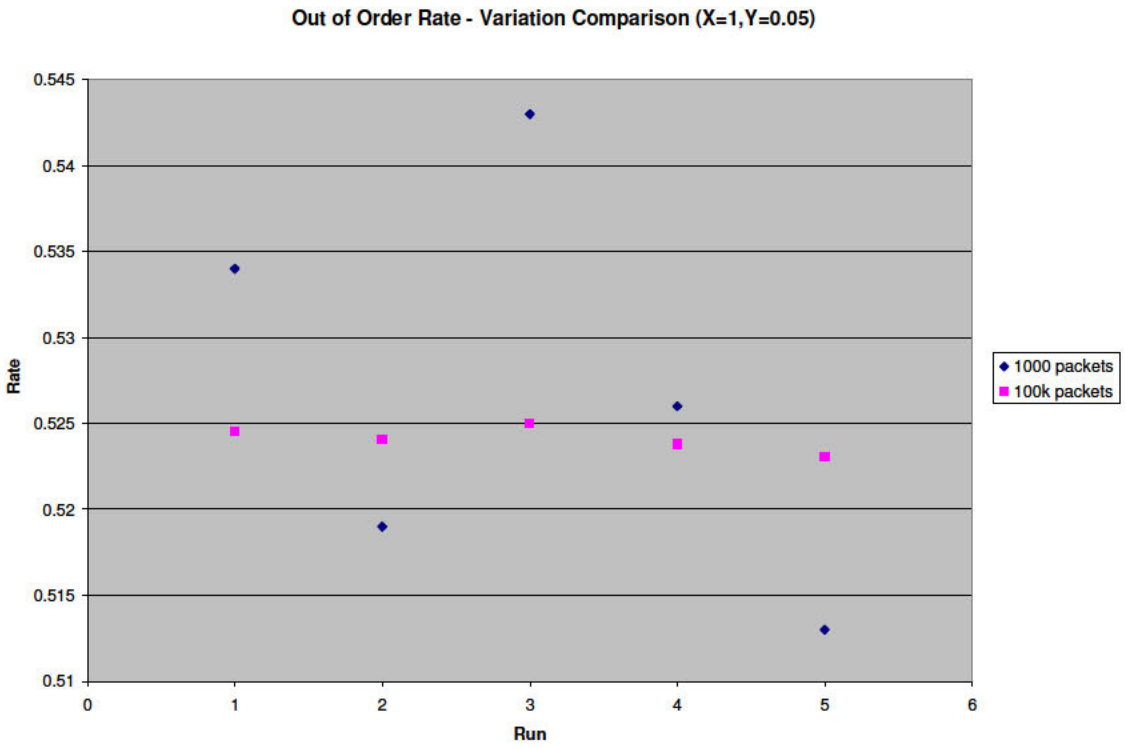

Figure 4 - Rate, 1000 vs 100k packets eg.

1-A)

This figure clearly illustrates how doing long runs decrease the variation in the average out-of-order rate. Examining the data, shows us this holds true for all the runs, as the confidence interval is smaller for the 100k sets (because it is proportional to the variance). This is encouraging, and conforms to the expected behavior.

1-B)

As can be seen from Figure 5 and 6, below, the confidence intervals are quite small. In fact, in Figure 6 they are so small that they aren't even visible on the chart! This shows that the variation between runs is low, and that the long-run average is stable.

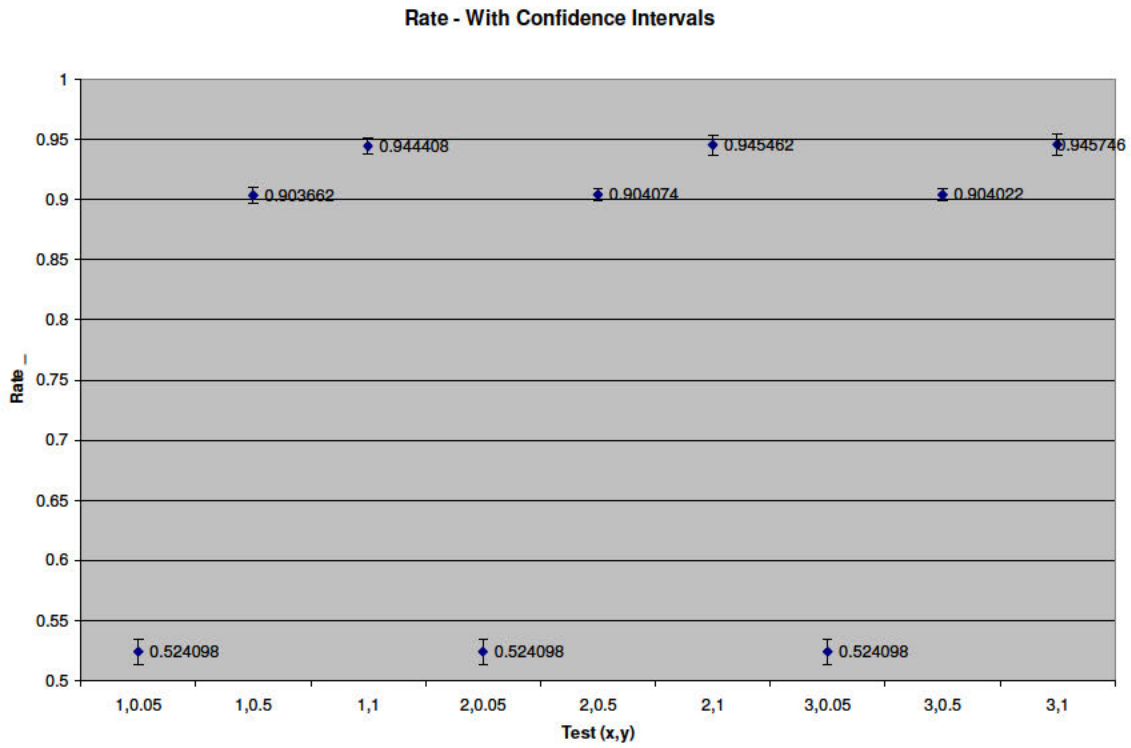

Figure 5 - Rate, with confidence intervals

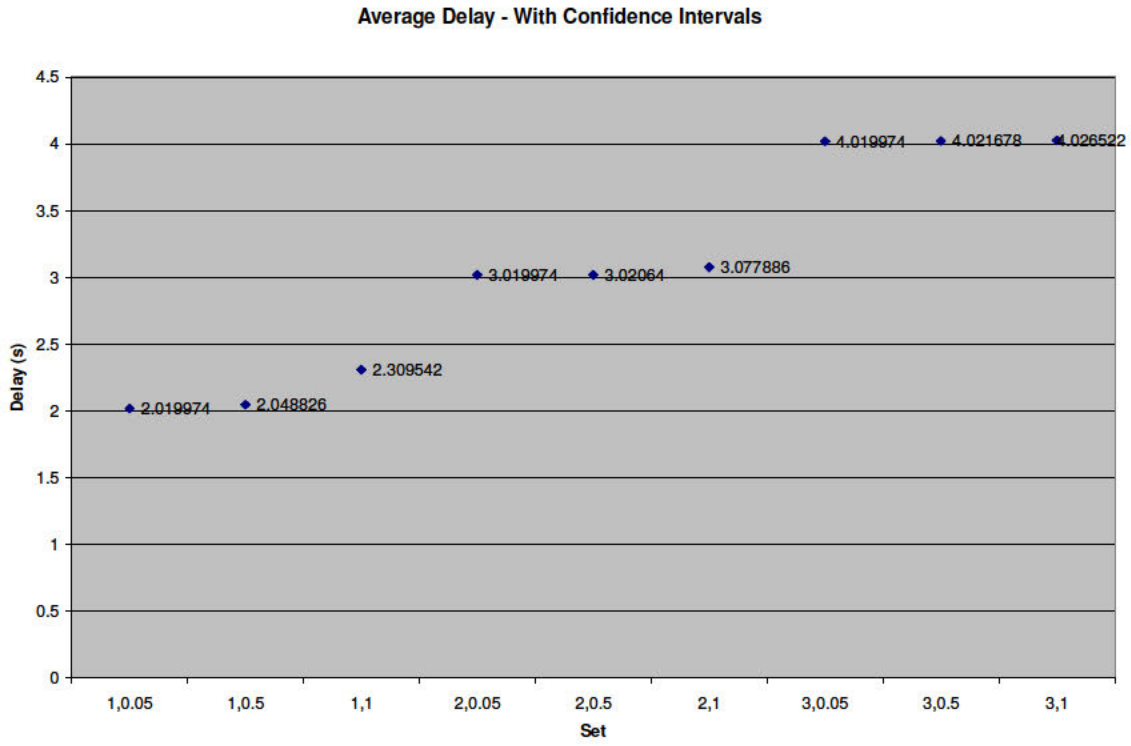

Figure 6 - Delay, with confidence intervals

2-A) Figure 5 shows that the higher the y-value of the normal distribution, the higher the out-of-order rate. Additionally, the performance is quite poor as the rate is never less than 50%, which shows that the priority queue may not be doing that much to improve things. Further tests, in the following section will have to verify this.

2-B) For each set of x-values, the average delays are roughly the same. Then, when the x-value increases by 1, the result is that the packet delay is also increased by 1. This is to be expected, as the priority queue should not have an effect on the average delay. This is because it is based on the rates of the channels, and packet ordering does not effect it.

3) The best rate performance is for the lowest y values. This follows from point 2-A, and shows that for higher variability in delay, the packets are more likely to end up out-of-order. In fact, for the best performance there is approximately 50/50 chance that the packet will arrive in the correct order. This is logical, as for y=0 we would expect a constant delay, and that the packets would always enter the router in the order they were produced. Therefore, as the standard deviation increases so does the chance of their changing places.

4) The results for the rates of (1,0.05), (2,0.05), (3,0.05) are identical across all runs. This seems odd at first glance, but can be explained by the fact that out-of-order rate is dependent on the y-value. What really matters is the difference in delay between subsequent packet. For them to get mixed up the difference must be greater than the channel transmission time. For low values of y, the difference should be the same, as the normal distribution shape is the same for most of the values
By this logic, the question is why aren't all the rates the same for each y-value set: (1, 0.5), (2,0.5), & (3,0.5), as well as (1,1),(2,1),&(3,1)? The answer is that they are very close, but the difference can be attributed to the outliers that come in to effect for higher x-values. Remember, the delays cannot be negative, therefore we were forced to use the truncated normal distribution to obtain the delay. This means the shape of the curves will vary based on how large the average is. So for larger means the curve will better approximate the actual normal distribution, which gives a greater variation above AND below the mean value (whereas the truncated version is lob-sided).

*note: We would expect to see more differences if we did larger runs, because more values would appear from the distribution's tails. These effects would of course be rare, should not impact the statistics.

5.2 Single Queue Data & Analysis

Table 2 - Single Queue Data

1,0.05	**100k packets**	
Seed	out-of-order rate	average packet delay
1	0.52841	2.01973
2	0.52804	2.01996
3	0.52898	2.02006
4	0.52779	2.02012
5	0.5267	2.02
Mean:	0.527984	2.019974
Var:	7.1573E-07	2.228E-08
Confidence +/-	0.000741545	0.000130834

1,0.5	**100k packets**	
Seed	out-of-order rate	average packet delay
1	0.90382	2.04908
2	0.90246	2.04741
3	0.90439	2.0512
4	0.90432	2.04906
5	0.90333	2.04738
Mean:	0.903664	2.048826
Var:	6.3573E-07	2.46278E-06
Confidence +/-	0.000698874	0.001375548

1,1	**100k packets**	
Seed	out-of-order rate	average packet delay
1	0.94477	2.30826
2	0.94426	2.30961
3	0.94488	2.31093
4	0.9444	2.31201
5	0.94373	2.3069
Mean:	0.944408	2.309542
Var:	2.0887E-07	4.16147E-06
Confidence +/-	0.000400591	0.001788077

2,0.05	**100k packets**	
Seed	out-of-order rate	average packet delay
1	0.52841	3.01973
2	0.52804	3.01996
3	0.52898	3.02006
4	0.52779	3.02012
5	0.5267	3.02
Mean:	0.527984	3.019974
Var:	7.1573E-07	2.228E-08

Confidence		
+/-	0.000741545	0.000130834

2,0.5	**100k packets**	
Seed	out-of-order rate	average packet delay
1	0.90419	3.01993
2	0.90261	3.01912
3	0.90419	3.02278
4	0.90547	3.02054
5	0.90391	3.02083
Mean:	0.904074	3.02064
Var:	1.03648E-06	1.86005E-06
Confidence		
+/-	0.000892367	0.001195433

2,1	**100k packets**	
Seed	out-of-order rate	average packet delay
1	0.94537	3.07843
2	0.94443	3.07507
3	0.94654	3.08265
4	0.94548	3.07837
5	0.94549	3.07491
Mean:	0.945462	3.077886
Var:	5.5917E-07	1.00031E-05
Confidence		
+/-	0.000655443	0.002772233

3,0.05	**100k packets**	
Seed	out-of-order rate	average packet delay
1	0.52841	4.01973
2	0.52804	4.01996
3	0.52898	4.02006
4	0.52779	4.02012
5	0.5267	4.02
Mean:	0.527984	4.019974
Var:	7.1573E-07	2.228E-08
Confidence		
+/-	0.000741545	0.000130834

3,0.5	**100k packets**	
Seed	out-of-order rate	average packet delay
1	0.90353	4.01965
2	0.90388	4.02136
3	0.90506	4.02251
4	0.90483	4.02251
5	0.90281	4.02236
Mean:	0.904022	4.021678
Var:	8.6537E-07	1.51587E-06
Confidence		
+/-	0.000815387	0.00107918

3,1	100k packets	
Seed	out-of-order rate	average packet delay
1	0.94501	4.02553
2	0.94573	4.02242
3	0.94596	4.0291
4	0.94625	4.02789
5	0.94578	4.02767
Mean:	0.945746	4.026522
Var:	2.1073E-07	6.91147E-06
Confidence +/-	0.00040237	0.002304348

Put simply, the results from the single queue runs are identical to those obtained from the priority queue. This is very discouraging at first glance, and really makes further scrutiny of the simulator warranted. This was done, repeatedly, and the simulator was verified to be operating correctly.

Therefore this led to investigating why no difference was noted between the two router designs. The average packet delay can be discounted in the same way as in the previous section. The router design has no effect on the average packet delay, and it is only effected by the transmission delay, and the means of the additional delays. This is because it is more of a matter of throughput than packet ordering.

To account for the out-of-order rate being the "same" it is important to look at how the statistic is defined. The router only has an effect on the statistic if it is able to completely fix the order of a packet. It turns out, that is not likely to happen using this method.

In fact, if we look closer at all the runs and get the difference between the priority and single queue we find the following:

The difference is either 0, or is very very small (so with too few decimals the results appear identical). Additionally, the priority queue always outperforms the single queue, but by an extremely small margin (and well within the confidence interval, so is basically worthless for trying to say that it is statistically better).

Looking at the differences across all 45 runs gives the following values:

Table 3 - Router Rate, Difference Summary

Mean:	0.001093133
Var:	3.10462E-06

This leads to the next section which tries to determine a more helpful statistic. Specifically, one that can show more clearly how well the priority queue design is performing.

5.3 Additional Statistic –Distance Squared

The idea for this section is that the out-of-order rate only reflects the effectiveness of the router if it is completely successful in reordering a packet. In theory the router could be doing a half-decent job, but only partially re-ordering the packets. If all the packets are examined after a run we can find the distance from the position they arrived in compared to their sequence number. This number can then be squared, summed up, and then the average can be found across all packets. The squaring of distance or error is a fairly standard approach. In this case the justification is that any packet that is early is equally out-of-order as a late packet, and that the further out of order the worse it is.

The results have been collected, but because this is a bonus section, have only been included in Appendix A (to keep the main report length down). To summarize the results the distance squared shows that the priority queue is very slightly better as far as the means that are reported. The amount of improvement is also slightly more significant than when comparing the out-of-order rate. However, the margin of improvement is still so small that it borders on insignificant as well. In fact, the confidence levels overlap with the means and so nothing should be said about improvement (again). In other words this new statistic reinforces that the improved router design has no effect, when compared to a standard single queue router.

6.0 Conclusion

6.1 Experience in learning OMNET++

In general the OMNET++ environment is robust and its' usefulness goes well beyond simple networks, like what was simulated for the project. Initial download, setup, configuration, tutorials were all well documented and easy to follow. The host system used for this project was running windows xp 32-bit, and no problems were encountered. The usefulness of the tic-toc tutorial cannot be emphasized enough. The template provided all the basic knowledge required for getting the simulation setup. Additional details were found easily enough through the api available on their website.
One detail that caused some problems initially was the fast vs express setting for the simulator. First of all, the simulation runs quite slow by default and switching to this mode meant that runs of 100,000 could be done instead of 1000. However, the hitch is that to get prints from inside the simulation they must be moved in to a classes finalize method. The alternative is to add a watch() flag on a variable, but this was more cumbersome because the output format is restricted. These are just some of the key parts of the experience, and looking at all the features this simulator has it there is definitely much more that could be learned.

6.2 Results and Analysis Summary

The priority queue provides no statistically verifiable benefit, in comparison to the regular single queue approach. The difference in performance of the out-of-order rate was on the order of 10^-6 or less, and well within the confidence intervals for the statistics. The average packet delay was not effected by the packet ordering, as it is dependent on the delay averages and transmission times, not the variation of the delay (aka, packet order). Finally, the additional statistic measuring the square of the distance showed more measurable improvement. Unfortunately the means of the two approaches were within each others' confidence intervals and therefore no conclusions could be made. These results are disappointing but proved to be a good exercise in thinking about how the system actually worked, and the effect input parameters had on the output statistics.

6.3 Recommendations

In terms of the project, it was relatively clear what was expected, though section 2 was a bit ambiguous as to what type of model was required (took a logical guess which is hopefully satisfactory). So, a more thorough report template might be desirable if this project is redone next year. The amount of work required for this project was a little on the heavy side, and mostly related to collecting the statistics (this is of course discounting the additional work done to get the extra stats). It also might be improved with more experience using the simulator, for example knowing how to do batch runs.

In terms of the router design there are two approaches that might yield more statistically significant results. The first would be to have the fast queue (or both queues) be converted from fifo queues to heaps. This would allow packets to correct their position even further, though this would only have an effect if the queues regularly had more than 1 or 2 packets in them. Another idea is to chain together many of these routers in series. In theory there is the potential that having many of these routers along the route could have an aggregate performance benefit.

Authors' note:
Thanks for a great semester.
Very enjoyable course, and definitely some challenging questions to consider.
:)

References

Banks, et al. “Discrete Event System Simulation”, Fifth edition. Prentice Hall 2010.

Sekercioglu, Ahmet “TicToc Tutorial for OMNeT++”, http://www.omnetpp.org/doc/omnetpp41/tictoc-tutorial/ (visited December 2010)

“Omnet++ API Reference” http://www.omnetpp.org/doc/omnetpp41/api/index.html (visited December 2010)

“Data rate units” http://en.wikipedia.org/wiki/Data_rate_units (visited December 2010)

APPENDIX – Code Used.
…see accompanying pdf document with source in it.
….also see accompanying .zip file for actual simulation directory.