Cp Sc 854 Homework #2:  
Creating and Testing a Poisson-Process Traffic Generator for ns2  
Jordan Jueckstock  
(with some embellishment by Dr Martin – these additional comments are in red)  

*Spring 2010*

**Overview**

The objective set forth in this homework assignment was the creation, testing, verification, and validation of an “ideal” Poisson-process traffic generator for the network simulator program ns2. Since it is generally considered common knowledge that Internet traffic does not follow a Poisson model (at least at the packet level), this assignment was not so much about accurately modeling a real-world network scenario as it was an exercise in modifying ns2 and comparing experimental results to a well-known analytical model.

The original homework description is added in Appendix 1.

**Procedure**

The first part of the project entailed creating a pure (“ideal”) Poisson-process traffic generator for ns2. Such a traffic generator emits fixed-size UDP datagrams with inter-packet delays following an exponential distribution.

The implementation proved fairly simple thanks to ns2’s modular, object-oriented design. The new PoissonProcess_Generator class is a subclass of the TrafficGenerator class, which handles all of the messy details of ns2’s inner operations for us. The new traffic generator publishes 2 parameters that can be set from OTcl scripts: average rate (in packets per second) and packet size (in bytes). Inter-packet delay is generated by an object of type ExponentialRandomVariable (conveniently built-in to ns2’s class library) that has its average set to

$$\frac{1}{\text{rate}}$$
The experiments used to verify and validate the new traffic generator involved a simple network model loosely reminiscent of a typical ADSL network. It consisted of a single router, a single “server” node, and a variable number of “client” nodes. The new traffic generator was used to generate UDP traffic from each client node to the server node. The topology looks like this:

![Figure 1](image.png)

The model can be manipulated via the following parameters:

- Depth of queue between router and server (unit: packets)
- Upstream bandwidth from client to router (unit: bps)
- Downstream bandwidth from router to client (unit: bps)
- Traffic generator parameters shared by all clients (rate and packet size)

The homework specifications proposed 3 experiments (henceforth referred to as A, B, and C) by which to verify that the traffic generator and the model were both logically correct. The following table lists the model parameter values used for each of these tests:

<table>
<thead>
<tr>
<th>Name</th>
<th>Queue Depth</th>
<th>Upstream Bandwidth</th>
<th>Downstream Bandwidth</th>
<th>Packets Per Second</th>
<th>Packet Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1000</td>
<td>12Mbps</td>
<td>2Mbps</td>
<td>20</td>
<td>1460 B</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>12Mbps</td>
<td>2Mbps</td>
<td>20</td>
<td>1460 B</td>
</tr>
<tr>
<td>C</td>
<td>200</td>
<td>20Mbps</td>
<td>10Mbps</td>
<td>75</td>
<td>1460 B</td>
</tr>
</tbody>
</table>

The objectives of these experiments are twofold. The first objective is to determine whether or not the model (and its underlying traffic generator) worked as designed. The second is to determine whether or not the design chosen resulted in a traffic model that was indeed Poisson.

To reach the first objective, the data gained from the above experiments must show reasonable, expected behavior. The per-flow throughput should match the average
bit-rate calculated from the traffic generator’s parameters. As the aggregate throughput begins to saturate the upstream link, queue levels and packet loss should rise as throughput begins to fall.

To attain the second, I must be able to detect appropriate patterns in queue levels and inter-packet-arrival intervals. As link utilization approaches saturation, the average queue level should rise as predicted by an M/D/1 process model. Inter-packet-arrival intervals should follow an exponential distribution.

**RESULTS**

The first step of model verification proved straightforward. The model works. The observed data behave as expected—i.e., the per-flow throughput closely matches the expected rate, and performance degrades predictably as aggregate throughput begins to saturate the upstream link.

The expected per-flow (and therefore, per-client) throughput for one of our experiments is given by the formula

\[ \text{packetsPerSecond} \times \text{packetSize} \times 8 \]

The expected throughput for each of the 3 experiments, therefore, is

- A: 233,600 bps
- B: 233,600 bps
- C: 876,000 bps

**Figures 2-4** include not only throughput, but also loss rate, upstream link utilization, and upstream queue level as well. It is clear that the observed data match the expected rates quite closely (so long as the upstream channel remains uncongested).

Experiment A’s network features a large enough queue that performance does not drop until the moment the upstream pipe is saturated (at approximately 428 connected clients). Even though Experiment C’s traffic is flowing faster, it too has a large enough queue to maintain performance until the upstream link is saturated (at about 113 clients—although in C’s case, the cutoff point is not so clearly defined as in A’s). Experiment B introduces a constricted upstream queue (just 20 packets) with the same level of traffic as Experiment A. As a result, B has less tolerance for congestion and shows signs of degraded performance much sooner and over a longer period of time compared to A. **Figures 5-7** plot the results of a second batch of experiments done to examine the number of connected clients that cause a backlog to develop and performance to fall (notice in B’s case that there is no obvious turning point).
Figure 4

Experiment C

Figure 5

Experiment A Close-Up
Figure 6

Experiment C Close-Up

Figure 7
The second objective (proving that the model is truly Poisson) was less obvious but not difficult to reach. As shown in Figure 8, the relationship between link utilization and average queue level in each experiment is reasonably similar to the analytical relationship predicted by an ideal M/D/1 process model. The observed data show higher than ideal queue levels across the board. While an in-depth diagnosis is beyond the scope of this assignment, I suspect the root cause to be packet fragmentation. All the experiments used 1460 bytes as the fixed packet size, but analysis of the packet arrival traces (see next paragraph) indicates that the simulated network links’ MTU was only 1000 bytes. The resulting traffic fragmentation would certainly have skewed the results in some direction—it seems reasonable to conjecture that it might have bumped up queue levels in this case.

![Figure 8](image)

Finally, as can be seen in Figures 9-12, the inter-packet-arrival intervals display a roughly exponential distribution (indicating that the arrival distribution is itself Poisson). The data used for Figures 9-12 was generated by tracing the arrival times of all packets entering the queue from the router to the server during an experiment of type A and by then plotting the PDF and CDF of the observed intervals. (Experiment A was chosen since its parameters provided the least constricted network and the best chance of observing an unimpeded flow of traffic.)

The curves in the resulting plots generally match those of an exponential distribution. The match is quite good in the case of a single flow of traffic (Figure 9), but it degrades noticeably when more connections are added (Figures 10-12). Two possible sources for this anomaly are the packet fragmentation mentioned above and unintended synchronization between the random variables used to
schedule inter-packet delays in the traffic generator. The latter looks especially intriguing given that in Figure 12 the curve appears to represent 2 exponential distributions on top of each other (one high-volume and the other low-volume).

Figure 9

Interarrival Intervals (50 Connections)

Figure 10
Figure 11

Interarrival Intervals (100 Connections)

Figure 12

Interarrival Intervals (200 Connections)
CONCLUSIONS

The nature of this assignment does not leave a lot of room for resounding conclusions. Suffice it to say that a Poisson-process ns2 traffic generator is very easy to create and fairly easy to test. In my case, I believe the experiment evidence strongly suggests that my implementation is both functionally correct and theoretically valid.

Appendix 1 Original Assignment

Q5 (60) ns2 simulation problem.

Build the following network model. The basic idea is to model a service provider’s DSL access network. You will be developing a new traffic generator such that the aggregate traffic arrival process arriving at ROUTER 1 (from the DSL Nodes) is Poisson.

```
#                DOWNLINKSPEED,
#                       10000 Mbps   UPLINKSPEED   Upstream Buffer Capacity
#Server ------ ROUTER 1 ------------------------- DSLNode 1 ... UDP1
#     Tprop                   |_______________DSLNode 2 --- UDP2
#  (n Number of UDP Sinks)    ......    
#                        |_______________DSLNode N --- UDPn
```

The model must support the following:

- A configurable number of nodes that connect to ROUTER 1. Each node must have its own simplex link configured with a separately configurable upstream and downstream link speed. The propagation delay for each DSL access link is 4 milliseconds. The upstream buffer capacity is an experimental parameter
- Each Node will have a UDP Agent with either a NEW traffic generator attached...call this generator the PoissonProcess. Traffic will always flow
upstream (from the DSL nodes to the server). You should follow the design of
the expoo exponential traffic generator. Create a new source C++ file called
PPGenerator.cc and.h. The tcl object name should be
Application/Traffic/PoissonProcess.

The model parameters are:

- Tprop: the propagation delay for each direction of the duplex link between
  ROUTER 1 and the Server.
- DOWNLINKSPEED, UPLINKSPEED: the upstream and downstream link
  capacity in bps.
- USBufferSize: The upstream buffer capacity (i.e., the maximum size of the
  upstream buffer). This should be in units of packets and is likely to range
  from 4 to 1024.
- NumberNodes: the number of DSL nodes that are configured.
- Traffic Generator parameters: Average rate (packets per second) and the
  packet size (it should be constant).

From a given simulation you will obtain the following results:

- Average UDP results from UDPstats.out (throughput, loss rate, jitter, latency)
- Link utilization
- Average queue level
- Average queue delay (estimate this from the UDP end-to-end results)

You must design and conduct an analysis that addresses the two following issues:

1. Demonstrate that your model is correct. Design a set of simple baseline
   experiments that provide both me and you with confidence that your model is
   correct. Make sure to exercise the model parameters to show the effects of changing
   them over a reasonable range.

2. Related to #1, I want you to validate that the traffic generator is indeed creating a
   Poisson arrival process at ROUTER 1 (again, just in the upstream direction). Your
   model should have added a 'packet sniffer' into the network (via the set qmon [$ns
   monitor-queue $n1 $n0 ""] tcl line). You need to edit the file ~/tools/queue-
   monitor.cc

#define TRACEARRIVALS 1
#define TRACEARRIVALS
FILE* fp;
fp = fopen("arrivals.dat", "a+");
fprintf(fp,"%g %d\n",now,pktsz);
fclose(fp);
#endif

This will log all packets that arrive at the link. You can plot the distribution using the plotArrivalDistribution.m matlab program (some modifications might be necessary).

Model the network using an M/D/1 queue model and compare the theoretical results (queue delay and queue levels versus utilization).

Your set of experiments should include at least the following:

- **EXPERIMENT 1:** (vary number nodes)
  - Tprop: 30 ms
  - DOWNLINKSPEED, UPLINKSPEED: 12 Mbps/2Mbps
  - USBufferSize: 200 packets
  - NumberNodes: vary 1 – something large to give you a nice range of utilizations and average queue levels. Try to avoid packet loss. Do 6 simulations all together, each with a different number of nodes.
  - Traffic Generator parameters: Set packet size to 1460 and rate of 20 packets per second.

- **EXPERIMENT 2:** (Same as EXP1, but reduce US Buffer size)
  - Tprop: 30 ms
  - DOWNLINKSPEED, UPLINKSPEED: 12 Mbps/2Mbps
  - USBufferSize: 20 packets
  - NumberNodes: vary 1 – something large to give you a nice range of utilizations (Do 6 simulations)
  - Traffic Generator parameters: Set packet size to 1460 and rate of 20 packets per second.

- **EXPERIMENT 3:**
  - Tprop: 30 ms
  - DOWNLINKSPEED, UPLINKSPEED: 20 Mbps/10Mbps
  - USBufferSize: 200 packets
  - NumberNodes: vary 1 – something large to give you a nice range of utilizations (do 6 simulations)
Traffic Generator parameters: Set packet size to 1460 and rate of 75 packets per second.

Present your results in a 4-6 page writeup. Please include the following:

- Document all aspects of the experiments, someone reading this should be able to duplicate your results.
- Include plots of average queue levels/queue delays versus link utilization. Also plot (in the same graph) the theoretical results using an M/D/1 queue model. Comment on the results, try to explain any observed errors or unexpected results.
- Comment on the sensitivity of the ability to generate a Poisson traffic arrival process to the various parameters of the model.

Submission: email to me your 4-6 page writeup in Microsoft Word format, all tcl scripts, all shell and awk scripts, and any matlab programs you developed. Put all this together into a file called cpsc854-HW2.tar.gz. Also include a readme.txt that identifies/describes each file in that package. As in HW1, please include in the readme.txt file instructions I would follow to run any of your scripts and matlab programs to reproduce your results.

Grading:

- (15) Verify the implementation of the model by exercising the four main model parameters?
- (15) Validate the model by comparing simulation results with M/D/1 results?
- (15) Overall design methodology effective and complete?
- (15) Documentation complete? If I gave the document to another student, could he/she know exactly what you wanted to do, how you did, and what you accomplished? Is there sufficient documentation allowing another student to recreate your results?