Title: *EE509 Assignement 1*

Name: *Ross Andreucetti*

Email: *ross.andreucetti4@mail.dcu.ie*

Student ID: 12211742

Programme: *MTCC*

Table of Contents

Specification of Single-Queue Design Simulator Design

Section 1

The class ExpRandGenerator is used to generate our random numbers needed for this assignment. It takes advantage of Javas' Math.random function. The function genRandom() in the ExpRandGenerator class is used to calculate both the inter-arrival time and the transmission time for packets. It takes in the following variables

- μ – This is the mean service rate in packets per second and when passed as an argument genRandom() returns the time in nanoseconds it will take to service a packet.
- λ – This is the mean transmission rate in packets per second and when passed as an argument genRandom() returns the time until the next packet in nanoseconds.

genRandom() uses the formula that was given in class for generating samples for a negative exponential distribution:

$$-\frac{1}{a}log(r)$$

The value is returned in nanoseconds as it is easier to understand then dealing with tiny fractions of seconds.

Section 2

The departure time is calculated by the Queue class. The method addPacket(Event e) adds Event e to the linked list and when the packet reaches the top of the queue from processing, the current time – the join time tells us the amount of time the packets was in the Queue. We can refer to this as waiting time, if we add the packets waiting time to the time it took to transmit the packet we get the total time for a packet joining the Queue until it leaves the Queue again. GenRandom(λ) is used to calculated the transmission of an individual packets.

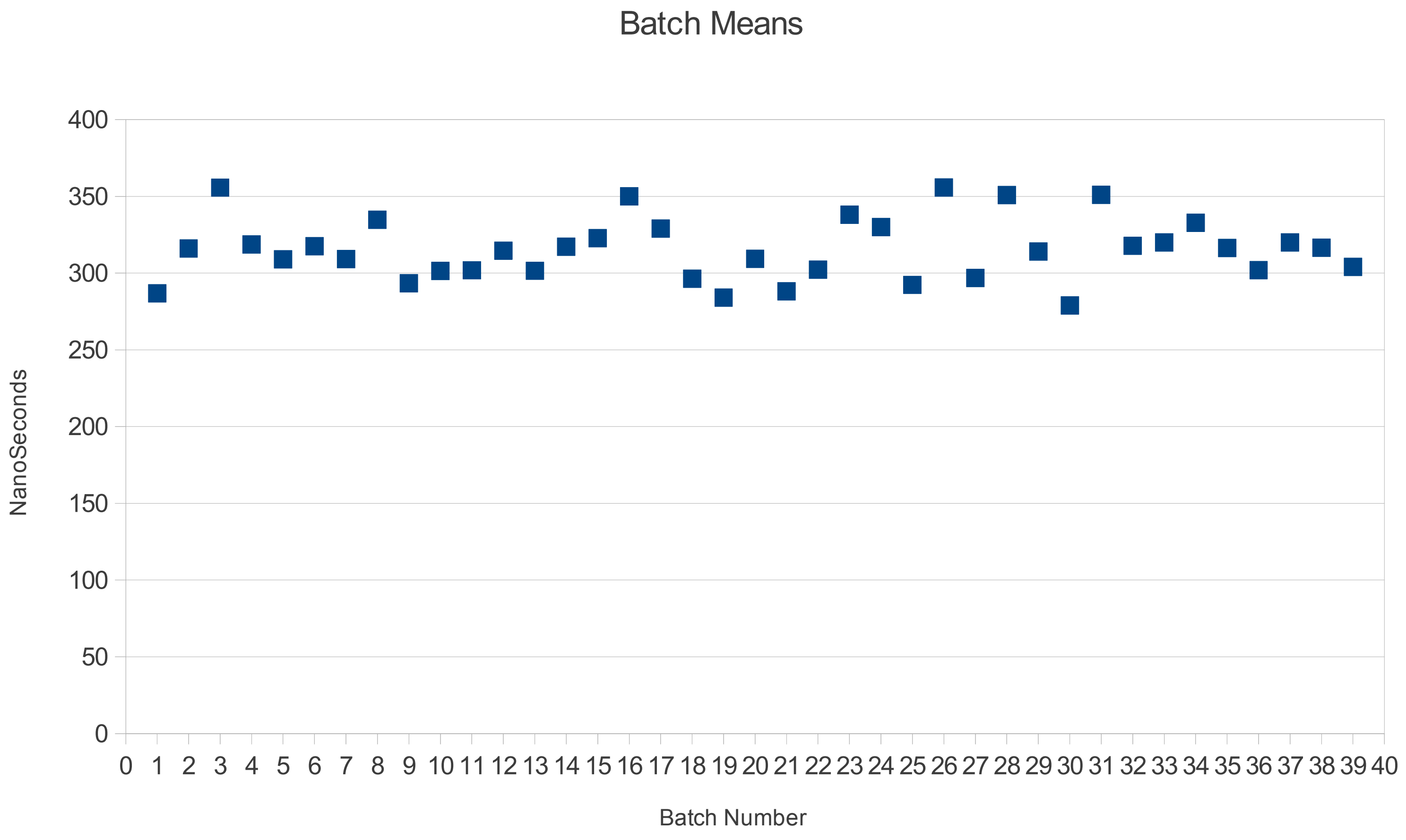
Section 3

The event for the SingleSimulation works as follows:

- The init() method is called first which sets globalTime and packetsTransmitted to 0. It also initializes the Source and the Queue.
- An arrival event is generated using the source to get us started and is added to the eventListManager.
- We know enter the loop which will run until a specified number of packets have arrived, 200000 in this case.
- Firstly the eventListManager is polled to get our next Event.
- The globalTime is the updated to match the time of this event.
- Next we check if it is an arrival or a departure packet.
    - If it is an arrival packet we add it to our Queue using the addPacket(Event e) method from earlier and keep track of the time this method returns.
    - This time called returnTime will be a value greater than zero, unless the packet has failed to be added to the Queue (if it's full) or the Queue is currently busy (isBusy is true).
    - If returnTime is greater -1 than this value represents when a departure event should be scheduled for the future and this departure is added to the EventListManager.
    - A new arrival packet is then generated for some time in the future and added to the EventListManager
    - Finally the number of packetsArrived is incremented to reflect the new arrival event.
- For a departure packet we do the following:
    - Call the removeQueue method from the Queue class.
    - This returns a value called returnTime which represents the time in the future when the next packet in the Queue will be transmitted. (As one packet has been removed from processing the next one is sent for processing and its transmission time is calculated.
    - This only happens if the Queue is not empty (returnTime>-1) if this is the case then the departure event for the next packet will be added to EventListManager.
- Finally when the number of packetsArrived reaches 200000 we print out the details of each of the batches using the Queues' printBatchDetails() method.

Simulation of Results for Single-Queue

Section 1



The average across all the batch means is: **315.28625ns**

Section 2

Based on these batch means I calculated the confidence interval to 95% as follows: firstly I calculated the mean of the batch means:

$$\overline{\overline{X}} = \frac{1}{k}\sum_{i=1}^{k} \overline{X}_i$$

where:

- $\overline{\overline{X}}$ = Mean of the Batch Means
- k = number of batch means
- $\overline{X}$ = Batch Mean

This works out to be :

$\overline{\overline{X}}$ = **315.28625ns**

Next the standard deviation was calculated using the following formula:

$$s^2 = \frac{1}{k-1}\sum_{i=1}^{k}(\overline{X}_i - \overline{\overline{X}})^2$$

where:

- s^2 = standard deviation^2 or the variance.

This works out to be:

s = 20.17273ns

Finally to compute the confidence interval at 95% we use the following formula:

$$(\bar{\bar{X}} - 1.96\frac{s}{\sqrt{k}}, \bar{\bar{X}} + 1.96\frac{s}{\sqrt{k}})$$

which works out at:

$$\pm 6.3312$$

or

$$\pm 2\%$$

Section 3

Using the formulas given in the notes we can calculate the theoretic result for the delay and compare it to our simulation result.

Firstly we calculate $\overline{N}$

$$\overline{N} = \frac{\rho}{1-\rho} - \frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}}$$

where:

- $\overline{N}$ = Average number of packets queueing, including packet being processed.
- P = Offered Load
- K = Queue Length (L+1)

Using the information associated with our Single Simulation we get the following the theoretic result:

$$\overline{N} = 3.1145$$

Next we calculate λeff which is the effective arrival rate:

$$\lambda_{eff} = \lambda(1 - p_K) = \lambda\left(1 - \frac{(1-\rho)\rho^K}{1-\rho^{K+1}}\right)$$

where:

- λeff = effective arrival rate
- λ = arrival rate

Using information relevant to our simulation we work this out at

$$\lambda\text{eff} = 9815500$$

Finally from our information above we can calculate the theoretic delay:

$$\overline{T} = \frac{\overline{N}}{\lambda_{eff}} = \frac{\overline{N}}{\lambda\left(\frac{1-\rho^K}{1-\rho^{K+1}}\right)}$$

$$\overline{T} = 317\text{ns}$$

From the mean we calculated earlier of **315ns** we can see that are simulation is accurate.

Section 4

From our batches we calculate the mean loss rate to be the following:

$$\text{Mean Loss Rate } = \overline{\overline{X}} = 1.77\%$$

Next we will calculate the confidence interval to 95% .

As we already have $\overline{\overline{X}}$ we the standard deviation using the formula in section 2:

$$s = 0.423\%$$

And again using the formula from before we calculate the confidence interval at 95%:

$$\pm 0.132\%$$

Next we can compare the theoretic result to the mean which we got from our simulation:

$$p_k = \frac{(1-\rho)\rho^k}{1-\rho^{K+1}} \qquad 0 \le k \le K \qquad \rho < 1$$

here we let k = 11 (Queue length +1)

$$p_K = 1.84\%$$

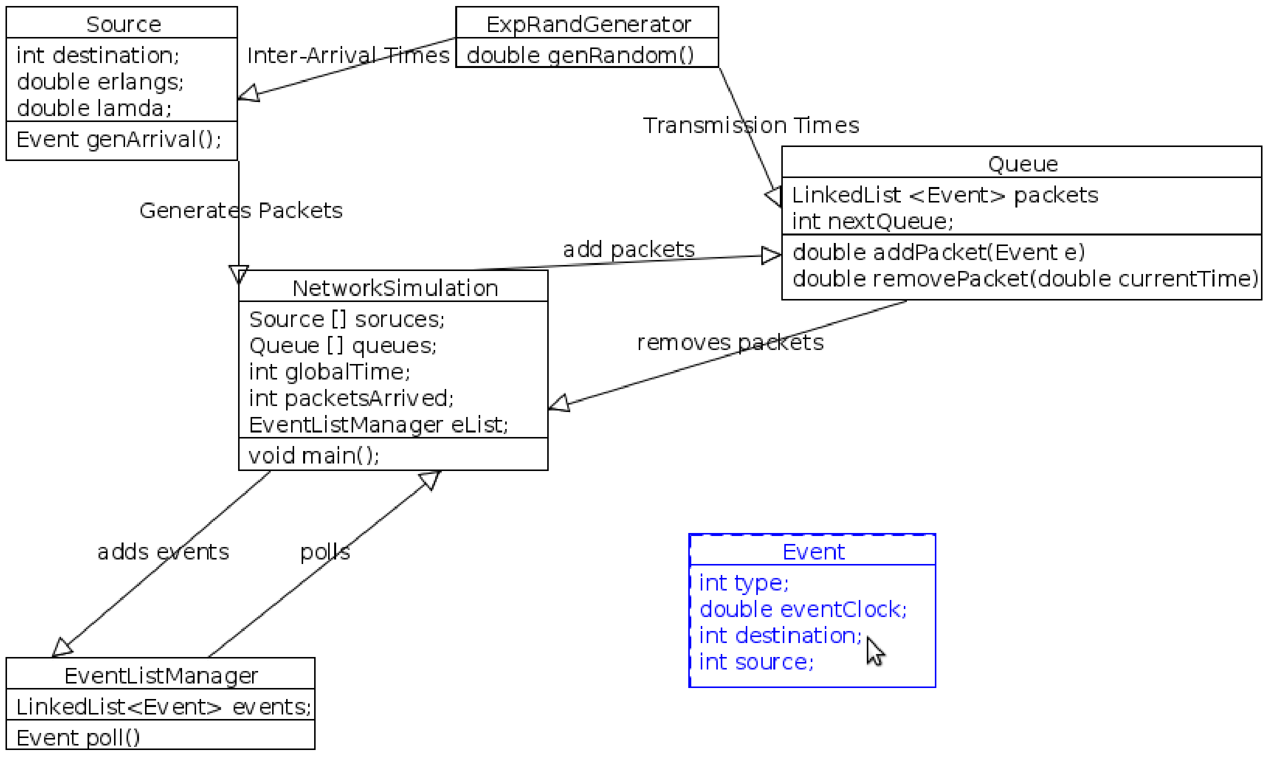
our mean was calculated to be 1.77% so this shows again the simulation is accurate.

Section 5

For this assignment I found that a lot more effort was required to implement the discrete-event simulator than getting the results by mathematical calculations. If I was to give a ratio I would estimate it to be in the range of 6:1.

Specification of Network Simulator Design

The design of the Network Simulator is not all that different from the single simulation model except with a few additional parts to handle packets departing one Queue and arriving at another. Firstly lets have a look at some UML to give an idea of how the Simulator is laid out.



Next lets take a closer look some of the classes that we use:

Source

The source class is responsible for generating fresh packets into the network i.e. packets that are not from another Queue. In this network there are 7 which correspond to the assignment specification and each one has a different offered load and path through the network.

The class itself has the variables erlangs, destination and lamda. The method genArrival(int sourceNum) returns an arrival Event with the specified destination, an appropriate inter-arrival time and the source of the packet.

Event

The event class is a modified version of the code supplied with the assignment, destination specifies which queue the packet is heading to, type indicates whether it is an arrival or departure and source specifies which source the packet came from. It is important to note that this value is only relevant if the packet is fresh (has not come from another queue).

Queue

The Queue class at its heart is comprised of a LinkedList of type which keeps track of the packets currently in the Queue and more importantly the size of the Queue and the arrival times of packets. A boolean isBusy indicates if a packet is currently being processed. There are also a number of other variables used for statistical information. The addPacket(Event e) method is used to add a packet to the Queue, there a 3 possible outcomes, first the Queue may be full and the packet will have to dropped, secondly the Queue may not be busy and the packet can be processed right away, in this case the transmission time is returns the the departure event can be scheduled by the simulator. Finally the packet may be added to the queue to wait for processing, the linkedList stores the time of arrival.

The removePacket(int currentTime) is called on a departure event it has two outcomes, if there are no packets waiting in the Queue then isBusy is set to false so the next arrival packet can go straight to processing. Otherwise the next packet in the Queue is sent for processing and a departure event is set based on its transmission time. Statistical data based on the waiting time is also calculated.

NetworkSimulation

NetworkSimulation is the main class where to program is run. It keeps track of the number of packets which arrive to the network, the globalTime and a linkedList of the sources and the queues.

The program functions as a loop:

- Firstly a packet from each source is generated from each source and the packet is scheduled.
- The program now enters the loop
- The eventListManager is polled to get the next Event.
- If it is an arrival event:
    - add the packet to the appropriate queue and if the queue was empty schedule the departure event (returnTime>-1)
    - then if the packet was a fresh packet (from a source, not a queue) schedule the next packet from that source
- If it is a departure event:
    - Check what Queue the packet is heading to next
    - call the appropriate queues removePacket() method
    - schedule a departure event if required
    - schedule and arrival event at nextQueue if required.
- Details are printed at the end.

Raised Questions

I think that most points have been covered above but I will answer the questions asked in the specification:

What information is associated with an event ans how is that information used: The information associated with an Event is the time of the event, it's type (arrival or departure) the packets source if it is an arrival event and the destination queue. This information is used for a number of reasons:

- adding arrivals to appropriate Queue
- removing packets from appropriate queues
- Calculating the waiting time of a packet
- scheduling a new arrival from a source

When a packet completes service (i.e. completes transmission), how is it decided where it should be routed to next (i.e. for which queue should a corresponding arrival event occur)?: Each queue has a record of what Queue it leads onto, when a departure event occurs in a queue then if the nextQueue variable is not -1 (packet left network) if will schedule an arrival event at appropriate queue.

How does the main event loop operate to handle arrivals and departures from different queues? The event class has been modified to include the destination variable which allows us to see what Queue the event is for.

How is generation of packets from multiple independent sources handled? Again the event class has been modified to include what source the packet came from. So when an arrival event from a source occurs we can schedule the next arrival for that source.

How is loss rate calculated and stored/batched for each queue (link)? Each queue maintains a record of statistical data and batch data. For example every time a packet is dropped packetsDropped is incremented.

How is end-to-end delay calculated and stored/batched for each end-to-end path? End to end delay is a combination of wait time and transmission time. Again each queue maintains statistical data which is incremented when appropriate and calculated to produce the correct results.

Network Simulation and Network Design Exercise

Queue 1A

Mean End-to-end Delay: **113.8ns**

Mean Loss Ratio: **0.0005**

95% Confidence Interval for End-to-end Delay

$\overline{\overline{X}}$ = **113.8ns**

s = **2.88ns**

95% confidence interval = $\pm$**.808ns or** $\pm$**0.7%**

49 batches of size **5000** were used here.

95% Confidence Interval for Loss Ratio

$\overline{\overline{X}}$ = **0.000508**

s = **0.00043**

95% confidence interval = $\pm$ **0.000069 or** $\pm$ **13%**

139 batches of size **5000** were used here.

- **Note that although the largest confidence interval is greater than 5% the offered load for this queue is so low that too many simulations would be needed for accurate data. It is enough to note that the loss ratio is very low and often 0.**

Queue 1B

Mean End-to-end Delay: **254.2ns**

Mean Loss Ratio: **0.101**

95% Confidence Interval for End-to-end Delay

$\overline{\overline{X}}$ = **254.2ns**

s = **8.02ns**

95% confidence interval = ± **3.6ns or** ± **1.4%**

39 batches of size **5000** were used here.

95% Confidence Interval for Loss Ratio

$\overline{\overline{X}}$ = **0.101**

s = **0.008**

95% confidence interval = ± **0.002 or** ± **2.4%**

44 batches of size **5000** were used here.

Queue 2A

Mean End-to-end Delay: **210.95ns**

Mean Loss Ratio: **0.0405**

95% Confidence Interval for End-to-end Delay

$\overline{\overline{X}}$ = **210.95ns**

s = **5.8ns**

95% confidence interval = ± **2.79ns or** ± **1.3%**

34 batches of size **5000** were used here.

95% Confidence Interval for Loss Ratio

$\overline{\overline{X}}$ = **0.0405**

s = **0.0048**

95% confidence interval = ± **0.0015 or** ± **3.8%**

36 batches of size **5000** were used here.

Queue 2B

Mean End-to-end Delay: **165.2ns**

Mean Loss Ratio: **0.0128**

95% Confidence Interval for End-to-end Delay

$\overline{\overline{X}}$ = **165.2ns**

s = **5.38ns**

95% confidence interval = $\pm$ **1.6ns or** $\pm$ **0.9%**

88 batches of size **5000** were used here.

95% Confidence Interval for Loss Ratio

$\overline{\overline{X}}$ = **0.0128**

s = **0.0029**

95% confidence interval = $\pm$ **0.0006 or** $\pm$ **4.7%**

89 batches of size **5000** were used here.

Queue 3A

Mean End-to-end Delay: **228.3ns**

Mean Loss Ratio: **0.066**

95% Confidence Interval for End-to-end Delay

$\overline{\overline{X}}$ = **228.3ns**

s = **8ns**

95% confidence interval = $\pm$ **3.74ns** or $\pm$ **1.6%**

36 batches of size **5000** were used here.

95% Confidence Interval for Loss Ratio

$\overline{\overline{X}}$ = **0.066**

s = **0.0066**

95% confidence interval = $\pm$ **0.002** or $\pm$ **3.1%**

39 batches of size **5000** were used here.

Queue 3B

Mean End-to-end Delay: **99.6ns**

Mean Loss Ratio: **see below**

95% Confidence Interval for End-to-end Delay

$\overline{\overline{X}}$ = **99.6ns**

s = **2ns**

95% confidence interval = ± **0.84ns** or ± **0.8%**

45 batches of size **5000** were used here.

95% Confidence Interval for Loss Ratio

Again as before as the loss ratio is so low, 0% for the majority in this queue it is not feasible to calculate the confidence interval accurately.

Queue 3C

Mean End-to-end Delay: **228.4ns**

Mean Loss Ratio: **0.057**

95% Confidence Interval for End-to-end Delay

$\overline{\overline{X}}$ = **228.4ns**

s = **7.57ns**

95% confidence interval = ± **3.49ns** or ± **1.5%**

37 batches of size **5000** were used here.

95% Confidence Interval for Loss Ratio

$\overline{\overline{X}}$ = **0.057**

s = **0.0053**

95% confidence interval = ± **0.0016** or ± **2.9%**

39 batches of size **5000** were used here.

Queue 4A

Mean End-to-end Delay: **212ns**

Mean Loss Ratio: **0.045**

95% Confidence Interval for End-to-end Delay

$\overline{\overline{X}}$ = **212ns**

s = **7ns**

95% confidence interval = $\pm$ **3.37ns** or $\pm$ **1.6%**

34 batches of size **5000** were used here.

95% Confidence Interval for Loss Ratio

$\overline{\overline{X}}$ = **0.045**

s = **0.0059**

95% confidence interval = $\pm$ **0.0019** or $\pm$ **4.2%**

36 batches of size **5000** were used here.

Section 2

Finally for the last section we are required to find the minimum value for L (Queue length) that reduces the average packet loss in a queue to <1%. This is to be done by trial and error.

Let's take **Queue 4A** as our example here.

- L = **5** => Loss Rate = **4.8%**
- L = **6** => Loss Rate = **3.4%**
- L = **7** => Loss Rate = **2.3%**
- L = **8** => Loss Rate = **1.7%**
- L = **9** => Loss Rate = **1.3%**
- *L = **10** => Loss Rate = **0.9%***

The method used was to increment the value until a value or <1% was found. The I ran the simulation repeatedly with that value to ensure it remained below 1%.