Answers to ns-3 training M/M/1 queue questions

I. Answers to questions

A. Question

1) For an M/M/1 queuing system, simulate with $\mu = 10$ and $\lambda =$ 1,3,5,7, and 9, and plot the queue idle proportion, mean queue length, and average packet delay, all as a function of the ratio λ/μ. For each trial, send 100,000 packets through the system. Set the queue limit to a large value to avoid drops.

B. Answer

There are a number of ways to obtain the data for this answer, including the post-processing of the trace file generated, or modifying the program to directly gather the information of interest.

The queue idle proportion can be defined as the proportion of time that the queue is idle, relative to the length of the simulation. In the trace file, there is a timestamp t_0 marking each dequeue event that sends the queue into a state of zero occupancy. Then there is a timestamp t_1 for an enqueue event. One can write a program to sum all the intervals (t_1 - t_0) in the trace file and divide by the total time. Let's call this quantity T_0.

The following command will yield a large data file `mm1queue.dat` for the first data point:

```
$ ./waf --run 'mm1-queue --mu=10 --lambda=1 --numPackets=100000'
```

This file begins with the following timestamped data, with the first column in units of seconds, and the third column the resulting queue length:

```
0.352958 + 1
0.528193 - 0
0.712375 + 1
```

```
0.764967 - 0
1.18865 + 1
1.20003 - 0
1.29859 + 1
1.44613 - 0
```

If we add a new first line to the plot `0.0000 - 0` to mark the start of the first idle time interval, we can then write a parsing program, or a spreadsheet, to find all instances of the queue length going to zero, and then record the time difference between that time stamp and the next time stamp where the queue size increments to 1 packet.

This Python program can do the processing:

```
#/usr/bin/env python
import sys
# The default file to read is 'mm1queue.dat', or you may provide a different
# filename as the first argument
x = []
filename = 'mm1queue.dat'
if len(sys.argv) > 1:
    filename= sys.argv[1]
with open(filename, 'r+') as f:
    lines = f.readlines()
    # Do not process the last line yet
    for i in range(0, len(lines) - 1):
        columns = lines[i].split()
        if columns[1] == '-' and columns[2] == '0':
            start_time = float(columns[0])
            nextline = lines[i+1]
            columns2 = nextline.split()
            stop_time = float(columns2[0])
            x.append(stop_time - start_time)
    # The last line will contain the total time
    columns = lines[len(lines) - 1].split()
    end_time = float(columns[0])
    print(sum(x) / end_time)
f.close()
```

Running it for the different input λ yields the following (all times in seconds):

λ	idle proportion (ns-3 experiment)
1	0.90
3	0.70
5	0.50
7	0.30
9	0.10

TABLE I: Queue idle proportion for different λ

These results align with the well-known equation for queue idle proportion in an M/M/1 queue: $\rho = 1 - \lambda/\mu$.

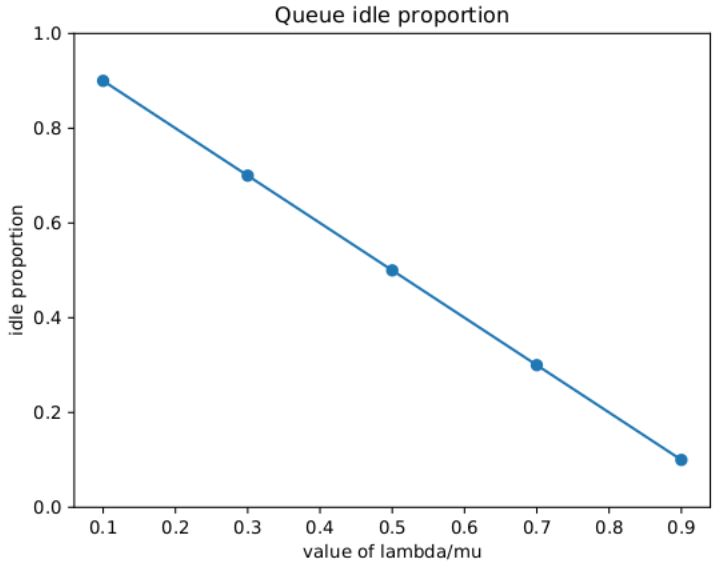

Fig. 1: Queue idle proportion vs λ/μ

The mean queue length is the mean number in the system (considering that the first in line packet is in the process of being served) and also follows from Little's law: $\rho/(1-\rho)$. We can compare against our experimental results either by working with the produced data file or the C++ program. Let T_0 be the proportion of time that the queue is empty, T_1 be the proportion of time that the queue has one packet in it, etc. The average queue length is simply the weighted sum of all of these proportions; i.e. T_0 * 0 + T_1 * 1 + T_2 * 2, etc.

λ	predicted by theory	observed (ns-3 experiment)
1	0.11	0.11
3	0.43	0.43
5	1	1.00
7	2.33	2.29
9	9	9.00

TABLE II: Mean queue length as function of λ

A Python program to tabulate this on the data file is:

```
#/usr/bin/env python
import sys
import numpy as np
# The default file to read is 'mm1queue.dat', or you may provide a different
# filename as the first argument
x = []
a = np.zeros(100)
filename = 'mm1queue.dat'
if len(sys.argv) > 1:
    filename= sys.argv[1]
with open(filename, 'r+') as f:
    lines = f.readlines()
    previous = lines[0].split()
    start_time = float(previous[0])
    length = int(previous[2])
    for i in range(1, len(lines)):
        current = lines[i].split()
        end_time = float(current[0])
        a[length] += (end_time - start_time)
        start_time = float(current[0])
        length = int(current[2])
        previous = current
mean_queue_length = 0
for i in range (0, len(a)):
    mean_queue_length += i * a[i]/end_time
print(mean_queue_length)
f.close()
```

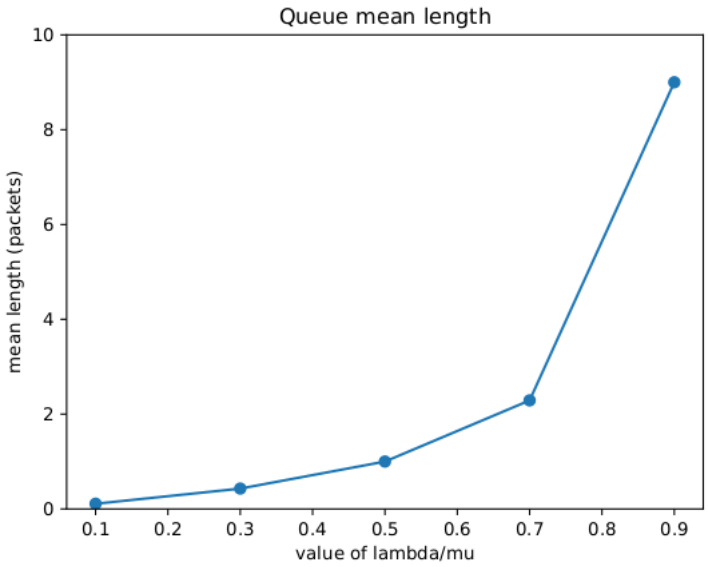

Fig. 2: Queue mean length vs λ/μ

The average packet delay is the average wait time in the system, which also follows from Little's Law: $1/(\mu - \lambda)$.

Tabulating this is a bit more involved to read from the trace file. The sample average is simply the sum of the time of each packet in the system, divided by the number of packets. However, when a packet is enqueued at a given time, there is no explicit label of the dequeue time for that specific packet in the trace file. It can be inferred if one writes a script to track the queue occupancy and figure out the enqueue event that led to each eventual dequeue, but it is probably more straightforward to track this from within the C++ program.

One can modify the `mm1=queue.cc` program to have the dequeuer record the difference between 'Now' and enqueue time, and add this sample to a running total of observed delays. The total can then be divided by the total number of packets dequeued (which can also be counted). The main challenge is to somehow tag each packet with a timestamp of when it was enqueued. There are a few ways to do this; the way depicted below is to make use of the Packet Unique ID (Uid) as the index to a vector which stores the Time of enqueue, and to then use this packet UID to fetch the enqueue time. Below is an example of how the program can be patched:

```
diff --git a/contrib/training-2019/examples/mm1-queue.cc
b/contrib/training-2019/examples/mm1-queue.cc
```

```
index af0796383..d4ca55980 100644
--- a/contrib/training-2019/examples/mm1-queue.cc
+++ b/contrib/training-2019/examples/mm1-queue.cc
@@ -18,6 +18,7 @@
 #include "ns3/core-module.h"
 #include "ns3/network-module.h"
+#include <vector>
 using namespace ns3;
@@ -32,6 +33,7 @@ public:
   void SetGenerator (Ptr<ExponentialRandomVariable> var) { m_var = var; }
   bool IsRunning (void) { return m_running;}
   void Start (void);
+  Time GetEnqueueTime (uint64_t packetId);
 private:
   void Generate (void);
@@ -40,6 +42,7 @@ private:
   uint32_t m_maxPackets {0};
   uint32_t m_numPackets {0};
   bool m_running {false};
+  std::vector<Time> m_enqueueLog;
 };
 void
@@ -60,6 +63,9 @@ Enqueuer::Generate (void)
     {
       NS_LOG_DEBUG ("Generating a packet at time " << Simulator::Now ().GetSeconds ());
       Ptr<Packet> p = Create<Packet> ();
+      NS_ABORT_MSG_UNLESS (p->GetUid () == m_enqueueLog.size (), "Queue size mismatch");
+      // Store simulation time for enqueue of a packet of a given ID
+      m_enqueueLog.push_back (Simulator::Now ());
       m_queue->Enqueue (p);
       Simulator::Schedule (Seconds (m_var->GetValue ()), &Enqueuer::Generate, this);
       m_numPackets++;
@@ -70,6 +76,12 @@ Enqueuer::Generate (void)
     }
 }
+Time
+Enqueuer::GetEnqueueTime (uint64_t index)
+{
+  return m_enqueueLog[index];
+}
+
 class Dequeuer : public Object
 {
 public:
@@ -77,11 +89,14 @@ public:
   void SetGenerator (Ptr<ExponentialRandomVariable> var) { m_var = var; }
   void SetEnqueuer (Ptr<Enqueuer> enqueuer) { m_enqueuer = enqueuer; }
   void Start (void);
+  Time GetAverageTimeInSystem (void);
```

```
 private:
   void Consume (void);
   Ptr<Queue<Packet> > m_queue;
   Ptr<ExponentialRandomVariable> m_var;
   Ptr<Enqueuer> m_enqueuer;
+  uint64_t m_numDequeued {0};
+  Time m_timeInSystem {Seconds (0)};
 };
 void
@@ -99,6 +114,8 @@ Dequeuer::Consume (void)
     {
       NS_LOG_DEBUG ("Dequeue successful");
       Simulator::Schedule (Seconds (m_var->GetValue ()), &Dequeuer::Consume, this);
+      m_timeInSystem += (Simulator::Now () - m_enqueuer->GetEnqueueTime (p->GetUid ()));
+      m_numDequeued++;
     }
   else
     {
@@ -115,6 +132,12 @@ Dequeuer::Consume (void)
     }
 }
+Time
+Dequeuer::GetAverageTimeInSystem (void)
+{
+  return (m_timeInSystem / m_numDequeued);
+}
+
 class QueueTracer
 {
 public:
@@ -214,6 +237,7 @@ int main (int argc, char *argv[])
     }
   Simulator::Run ();
+  std::cout << deq->GetAverageTimeInSystem ().GetSeconds () << std::endl;
   Simulator::Destroy ();
   return 0;
```

The results from the modified program are shown below, in comparison with theoretical results.

λ	predicted by theory	observed (ns-3 experiment)
1	0.11	0.112
3	0.143	0.143
5	0.2	0.200
7	0.33	0.328
9	1	1.003

TABLE III: Average time in system for different λ

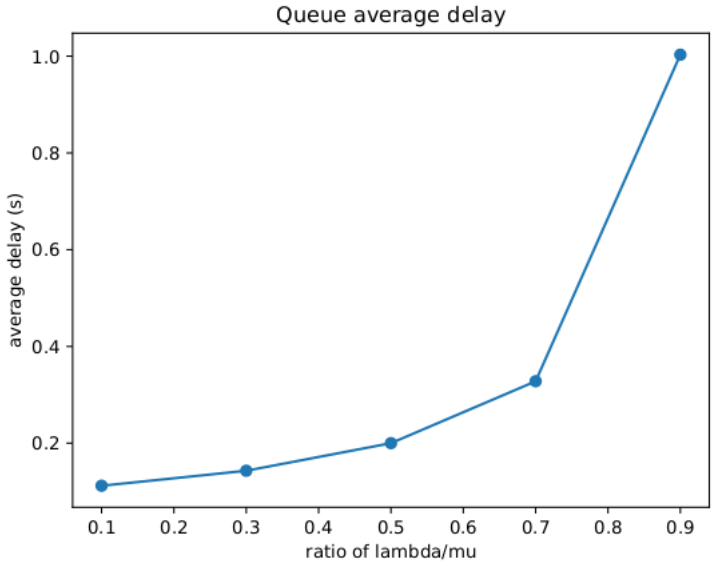

Fig. 3: Average queue delay vs. λ/μ

C. *Question*

4) Experiment with the buffer overflow outcome of a 100 packet M/M/1 queue by setting $\mu = 10$ and $\lambda = 9.5$, 9.7, and 9.9. Starting from an empty queue, run a number of trials to measure how long it takes for the queue to overflow. Provide your estimate of the average time until overflow, sample standard deviation, and number of trials conducted for each $lambda$.

D. Answer

Define a trial to be the execution of one simulation and the measurement of time until the first drop is observed. When $\lambda = 9.5$, it will generally take a few thousand packets until a drop is observed.

One can easily instrument the simulator to simply stop at the first drop occurrence, and output the time:

```
diff --git a/contrib/simple-wireless/examples/mm1-queue.cc
b/contrib/simple-wireless/examples/mm1-queue.cc
index af0796383..3a8e4e9a4 100644
--- a/contrib/simple-wireless/examples/mm1-queue.cc
+++ b/contrib/simple-wireless/examples/mm1-queue.cc
@@ -154,6 +154,7 @@ QueueTracer::DropTracer (Ptr<OutputStreamWrapper> stream, Ptr<Queue<Packet> > qu
 {
   NS_LOG_DEBUG ("Trace drop at time " << Simulator::Now ().GetSeconds ());
   *stream->GetStream () << Simulator::Now ().GetSeconds () << " d " << queue->GetNPackets () << std::endl;
+  Simulator::Stop ();
 }
@@ -161,11 +162,15 @@ int main (int argc, char *argv[])
 {
   bool quiet = false;
   uint32_t numInitialPackets = 0;
-  uint32_t numPackets = 0;
+  uint32_t numPackets = 100000;
   uint32_t queueLimit = 100; // packets
   double lambda = 1;
   double mu = 2;
+  std::ofstream outfile;
+
+  outfile.open("mm1queue-time-until-drop.dat", std::ios_base::app);
+
   CommandLine cmd;
   cmd.AddValue ("lambda", "arrival rate (packets/sec)", lambda);
   cmd.AddValue ("mu", "departure rate (packets/sec)", mu);
@@ -214,6 +219,8 @@ int main (int argc, char *argv[])
     }
   Simulator::Run ();
+  outfile << Simulator::Now ().GetSeconds () << std::endl;
+  outfile.close ();
   Simulator::Destroy ();
   return 0;
```

Then, answer is a matter of running this for some number of times with different random

seeds:

```
waf --run 'mm1-queue --lambda=9.5 --mu=10 --numPackets=100000 --RngRun=1'
waf --run 'mm1-queue --lambda=9.5 --mu=10 --numPackets=100000 --RngRun=2'
...
```

For example, using a shell script to run the simulation 100 times for each λ, one can then process the resulting 100 values with another script or a spreadsheet to compute the sample mean and std. deviation of the 100 values. It can be observed that, as expected, the average time until a queue overflow decreases as lambda increases. However, note that the standard deviation is still quite large despite 100 trials. We are not looking for a specific number of trials here but, in general, more trials are most likely needed for an experiment such as this because the time to drop has a high variance.

λ	trials	avg. time (s)	std. dev
9.5	100	5282	3828
9.7	100	1852	1623
9.9	100	840	721

TABLE IV: Statistics on time until queue overflow