Lab 1 Report
Network Simulation

Kyle Pontius

1 Introduction

This lab focuses on two general topics. First, the behavior of the file transfers in two- and three-node configurations. Second, comparing the simulator and its behavior to the anticipated results we learn from Queueing Theory. For each question in the file transfer section there were several parameters used to create the network: bandwidth, propagation delay, and packet size. Each of those will be specified for each question itself.

2 Two-Node Configuration

All questions in this section used a two-node configuration, using a bi-directional link. Additionally, for each question I'll calculate the anticipated delays and compare them to the results return by the simulator.

Question 1:

This configuration was 1 Mbps, 1 second propagation delay, with a packet size of 1000 bytes. One packet is sent over the network and its resulting creation time, identifier, and received time were printed. The code used to configure the network:

```
# setup network
net = Network('config-2n-1Mbps-1000ms.txt')

# setup routes
n1 = net.get_node('n1')
n2 = net.get_node('n2')
n1.add_forwarding_entry(address=n2.get_address('n1'),link=n1.links[0])
n2.add_forwarding_entry(address=n1.get_address('n2'),link=n2.links[0])

# setup app
d = DelayHandler()
net.nodes['n2'].add_protocol(protocol="delay",handler=d)

# send one packet
p = packet.Packet(destination_address=n2.get_address('n1'),ident=1,protocol='delay',length=1000)
Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)

# run the simulation
Sim.scheduler.run()
```

Simulator Results:

Current Simulator Time (sec)	Packet Identifier	Packet Created (sec)	Packet Received (sec)
1.008	1	0	1.008

Manual Calculation Results:

8000 bits (Packet Size) / 1000000 bps (Link Speed) = 0.008 seconds
0.008 seconds + 1 second (Prop. Delay) = 1.008 seconds

The simulator printed out a time of 1.008 seconds, which matches the time anticipated after manually calculating the time.

Question 2:

The configuration for this test was a bandwidth of 100 bps, a propagation delay of 10 ms, and a packet size of 1000 bytes. One packet is sent at time 0. All code was exactly the same as the Question 1, except I used a different network configuration file:

```
# setup network
net = Network('config-2n-100bps-10ms.txt')
```

Simulator Results:

Current Simulator Time (sec)	Packet Identifier	Packet Created (sec)	Packet Received (sec)
80.01	1	0	80.01

Manual Calculation Results:

8000 bits (Packet Size) / 100 bps (Link Speed) = 80 seconds
80 seconds + 10 ms (Prop. Delay) = 80.01 seconds

The results from the simulator match the anticipated calculations done by hand.

Question 3:

This configuration consisted of using a bandwidth of 1 Mbps, 10 ms propagation delay, and a packet size of 1000 bytes. For this test, 3 packets are sent at time 0 seconds, then 1 packet at time 2 seconds. My code for this configuration uses the same routes and app, but a different network and method of sending out packets. I simply set up the network, then manually sent each packet as follows:

```
# setup network
net = Network('config-2n-1Mbps-10ms.txt')

…

# send one packet
p = packet.Packet(destination_address=n2.get_address('n1'),ident=1,protocol='delay',length=1000)
Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)

# send one packet
p = packet.Packet(destination_address=n2.get_address('n1'),ident=2,protocol='delay',length=1000)
Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)

# send one packet
p = packet.Packet(destination_address=n2.get_address('n1'),ident=3,protocol='delay',length=1000)
```

```
Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)

# send one packet
p = packet.Packet(destination_address=n2.get_address('n1'),ident=4,protocol='delay',length=1000)
Sim.scheduler.add(delay=2, event=p, handler=n1.send_packet)
```

Simulator Results:

Current Simulator Time (sec)	Packet Identifier	Packet Created (sec)	Packet Received (sec)
0.018	1	0	0.018
0.026	2	0	0.026
0.034	3	0	0.034
2.018	4	2.0	2.018

Manual Calculation Results:

First Packet –
8000 bits (Packet Size) / 1000000 bps (Link Speed) = 0.008 seconds
0.008 seconds + 10 ms (Prop. Delay) = 0.018 seconds

Second Packet –
8000 bits (Packet Size) / 1000000 bps (Link Speed) = 0.008 seconds
0.008 seconds + 0.008 seconds (Queueing Delay) = 0.016 seconds
0.016 seconds + 10 ms (Prop. Delay) = 0.26 seconds

Third Packet –
8000 bits (Packet Size) / 1000000 bps (Link Speed) = 0.008 seconds
0.008 seconds + 0.016 seconds (Queueing Delay) = 0.024 seconds
0.024 seconds + 10 ms (Prop Delay) = 0.034 seconds

Fourth Packet –
8000 bits (Packet Size) / 1000000 bps (Link Speed) = 0.008 seconds
0.008 seconds + 0.0 seconds (Queueing Delay) = 0.008 seconds
0.008 seconds + 10 ms (Prop Delay) = 0.018 seconds
0.018 seconds + 2 seconds (Current Simulator Time) = 2.018 seconds

All simulator results match the anticipated calculated results.

3 Three-Node Configuration

Question 1:

This network uses two fast links, with identical configurations: 1 Mbps bandwidth and 100 ms propagation delay. The network sends a 1 MB file sent in 1 kB increments. The nodes are set up:

1 → 2 → 3

The code for the configuration for both Questions 1 and 2 are identical, except for the network configuration file they use:

```
# setup network
net = Network('config-3n-1Mbps-1Mbps-100ms-100ms.txt')

# setup routes
n1 = net.get_node('n1')
n2 = net.get_node('n2')
n3 = net.get_node('n3')

# setup forwarding entries
a2 = n2.get_address('n1')
a1 = n1.get_address('n2')
a3 = n3.get_address('n2')

n1.add_forwarding_entry(address=a2, link=n1.links[0])
n2.add_forwarding_entry(address=a1,link=n2.links[0])
n2.add_forwarding_entry(address=a3,link=n2.links[1])
n3.add_forwarding_entry(address=a2,link=n3.links[0])

# if packet is going from n1 -> n3
n1.add_forwarding_entry(address=a3,link=n1.links[0])

packetCount = 100

# setup app
d = DelayHandler(packetCount,outputFile)
net.nodes['n1'].add_protocol(protocol="delay",handler=d)
net.nodes['n2'].add_protocol(protocol="delay",handler=d)
net.nodes['n3'].add_protocol(protocol="delay",handler=d)

for i in range(0, packetCount):
    p = packet.Packet(destination_address=a3,ident=i,protocol='delay',length=1000)
    Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)

# run the simulation
Sim.scheduler.run()
```

Question 1 part 2 configuration file:

```
# setup network
net = Network('config-3n-1Gbps-1Gbps-100ms-100ms.txt')
```

Part 1 Partial Output (1 Mbps) –

```
Current Time: 0.216000, Packet Id: 0, Creation Time: 0.000000, Elapsed Time: 0.216000, TD: 0.016000, PD: 0.200000, QD: 0.000000
Current Time: 0.224000, Packet Id: 1, Creation Time: 0.000000, Elapsed Time: 0.224000, TD: 0.016000, PD: 0.200000, QD: 0.008000
Current Time: 0.232000, Packet Id: 2, Creation Time: 0.000000, Elapsed Time: 0.232000, TD: 0.016000, PD: 0.200000, QD: 0.016000
Current Time: 0.240000, Packet Id: 3, Creation Time: 0.000000, Elapsed Time: 0.240000, TD: 0.016000, PD: 0.200000, QD: 0.024000
Current Time: 0.248000, Packet Id: 4, Creation Time: 0.000000, Elapsed Time: 0.248000, TD: 0.016000, PD: 0.200000, QD: 0.032000
Current Time: 0.256000, Packet Id: 5, Creation Time: 0.000000, Elapsed Time: 0.256000, TD: 0.016000, PD: 0.200000, QD: 0.040000
…
```

Part 2 Partial Output (1 Gbps) –

```
 Current Time: 0.200016, Packet Id: 0, Creation Time: 0.000000, Elapsed Time: 0.200016, TD: 0.000016, PD: 0.200000, QD: 0.000000
Current Time: 0.200024, Packet Id: 1, Creation Time: 0.000000, Elapsed Time: 0.200024, TD: 0.000016, PD: 0.200000, QD: 0.000008
Current Time: 0.200032, Packet Id: 2, Creation Time: 0.000000, Elapsed Time: 0.200032, TD: 0.000016, PD: 0.200000, QD: 0.000016
Current Time: 0.200040, Packet Id: 3, Creation Time: 0.000000, Elapsed Time: 0.200040, TD: 0.000016, PD: 0.200000, QD: 0.000024
Current Time: 0.200048, Packet Id: 4, Creation Time: 0.000000, Elapsed Time: 0.200048, TD: 0.000016, PD: 0.200000, QD: 0.000032
Current Time: 0.200056, Packet Id: 5, Creation Time: 0.000000, Elapsed Time: 0.200056, TD: 0.000016, PD: 0.200000, QD: 0.000040
…
```

My results for this test were fairly straightforward. First, the file took 1.008 seconds to complete sending. Second, the because the bandwidth is relatively low the queueing delay builds up quickly over times, resulting in this queueing delay dominating the sending time. When the bandwidths are both bumped to 1 Gbps, the file takes 0.2008 seconds to complete sending. The Transmission Delay (TD) and Queueing Delay (QD) are both substantially less on the 1 Gbps links, both being exactly 1000x less since the link speed is 1000x faster. The decrease in QD can be attributed to the fact that because the connection processes so many more packets through per second on a 1 Gbps connection, each packet spends less time waiting to be sent. I'm also convinced that the simulator is working properly because the TD for the first packet is twice that of the measured in the two-node 1 Mbps configuration. The related QD also corresponds correctly for the first packet, if you take the twice the propagation delay and add it to the total TD, you get the new QD. That is:

0.016 (TD) + 0.2 (PD) = 0.216 (Time Received for Part 1, first packet)

Question 2:

This network uses one fast link and one slow link. 1 → 2 uses a 1 Mbps and 100 ms propagation delay configuration. 2 → 3 uses a 256 Kbps and 100 ms propagation delay. The code to set up the network is all identical to Question 1, except for the configuration file used:

```
# setup network
net = Network('config-3n-1Mbps-256Kbps-100ms-100ms.txt')
```

Partial Debugging Output –

```
Current Time: 0.239250, Packet Id: 0, Creation Time: 0.000000, Elapsed Time: 0.239250, TD: 0.039250, PD: 0.200000, QD: 0.000000
Current Time: 0.270500, Packet Id: 1, Creation Time: 0.000000, Elapsed Time: 0.270500, TD: 0.039250, PD: 0.200000, QD: 0.031250
Current Time: 0.301750, Packet Id: 2, Creation Time: 0.000000, Elapsed Time: 0.301750, TD: 0.039250, PD: 0.200000, QD: 0.062500
Current Time: 0.333000, Packet Id: 3, Creation Time: 0.000000, Elapsed Time: 0.333000, TD: 0.039250, PD: 0.200000, QD: 0.093750
Current Time: 0.364250, Packet Id: 4, Creation Time: 0.000000, Elapsed Time: 0.364250, TD: 0.039250, PD: 0.200000, QD: 0.125000
Current Time: 0.395500, Packet Id: 5, Creation Time: 0.000000, Elapsed Time: 0.395500, TD: 0.039250, PD: 0.200000, QD: 0.156250
…
```

It's clear that this output is correct, but it's obviously limited by the second link. By comparison, in Question 1 Part 1, there was no bottleneck so the network wasn't slowed by either node. In this configuration, the network is limited by the 256 Kbps link speed for the second link. As a result, the QD is significantly higher than the Question 1 Part 1 results. I'm convinced the simulator is working properly because the first packet correctly corresponds to the calculated results:

0.008 seconds (1st TD) + 0.03125 (2nd TD) = 0.03925 seconds (matches 1st packet's results)
0.03925 seconds + 0.2 seconds (total PD) = 0.23925 seconds (matches 1st packet's Received Time)

(see next page)

4 Queueing Theory

I used the *Delay.py* code to generate output from the simulator. Here is the code:

Configure Network –

```
# parameters
Sim.scheduler.reset()

# setup network
net = Network('../networks/one-hop.txt')

# setup routes
n1 = net.get_node('n1')
n2 = net.get_node('n2')
n1.add_forwarding_entry(address=n2.get_address('n1'),link=n1.links[0])
n2.add_forwarding_entry(address=n1.get_address('n2'),link=n2.links[0])

# setup app
d = DelayHandler()
net.nodes['n2'].add_protocol(protocol="delay",handler=d)

# setup packet generator
destination = n2.get_address('n1')
max_rate = 1000000/(1000*8)
load = 0.1*max_rate
g = Generator(node=n1,destination=destination,load=load,duration=20)
Sim.scheduler.add(delay=0, event='generate', handler=g.handle)

# run the simulation
Sim.scheduler.run()
```

Estimate Utilization –

```
def handle(self,event):
    # quit if done
    now = Sim.scheduler.current_time()
    if (now - self.start) > self.duration:
        return

    # generate a packet
    self.ident += 1
    p = packet.Packet(destination_address=destination,ident=self.ident,protocol='delay',length=1000)
    Sim.scheduler.add(delay=0, event=p, handler=self.node.send_packet)
    # schedule the next time we should generate a packet
    Sim.scheduler.add(delay=random.expovariate(self.load), event='generate', handler=self.handle)
```

Print Results –

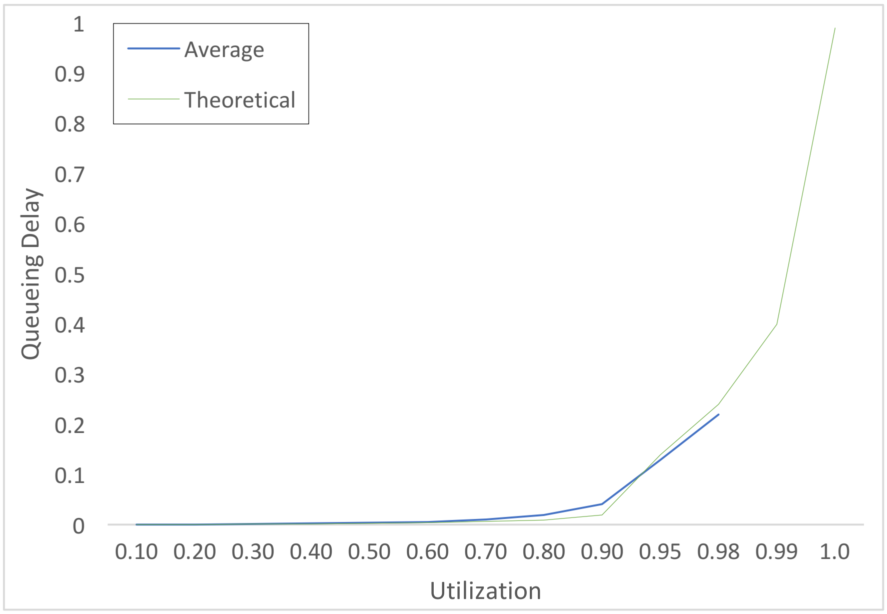
```
class DelayHandler(object):
    queueing_delay_times = []

    def __init__(self):
        print "# Time (sec), Packet Id, Packet Created, Packet Lifetime, TD, PD, QD"

    def receive_packet(self,packet):
        self.calculate_new_average_q_delay(packet.queueing_delay)
        print Sim.scheduler.current_time(),packet.ident,packet.created,Sim.scheduler.current_time() -
packet.created,packet.transmission_delay,packet.propagation_delay,packet.queueing_delay

    def calculate_new_average_q_delay(self, new_delay):
        self.queueing_delay_times.append(new_delay)
        print "AVERAGE Q DELAY: " + str(sum(self.queueing_delay_times) / len(self.queueing_delay_times))
```

Graph –



The data I gathered using this system was very interesting. One of the most notable parts of the data was the random nature of the what was returned. There was a random delay coded into the *Generator.handle()* method which effectively mimics the occurrence of traffic on the network. Interestingly enough, the nature of the data turned out to be pretty widely varied as I ran the simulator. Specifically, at 90% load, you may get a range of Queueing Delays that spans 0.5 seconds. For example, you may get an average delay of 0.06, but running the same code again may give you an average delay 0.11. Generally, though, the average delay was focused in one general area that aligned well with the theoretical graph. What that randomness does suggest, though, is that collisions on networks can occur in an almost infinite number of ways, resulting in wildly different delays on the network.

The graph above depicts the results of the average Queueing Delay that I observed after generating my results. As I noted in the previous paragraph, while the simulator would often keep its results close to the theoretical delay, its range of results were often much different. On average, though, the results matched the theoretical Queueing Delay very closely as can be seen from the graph itself.