Lab 1 Report

Nate Fox

1 Two Nodes

First I will explore a simple network of two nodes with a bidirectional link. Here is the code that sets this up:

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

# setup routes
n1 = net.get_node('n1')
n2 = net.get_node('n2')
link_1_2 = n1.links[0]
link_2_1 = n2.links[0]
n1.add_forwarding_entry(address=n2.get_address('n1'), link=link_1_2)
n2.add_forwarding_entry(address=n1.get_address('n2'), link=link_2_1)
```

Note that the link from n1 to n2 is named link_1_2 and the reverse link is named link_2_1. Also note that two forwarding entries were created to direct network traffic to the correct destination. The one-hop.txt file that is referenced in the above code is used to create the simple network:

```
# n1 -- n2
#
n1 n2
n2 n1
```

For the first experiment, the bandwidth of the link is set to 1 Mbps and the propagation delay is set to 1 second. One packet of 1000 bytes is sent from n1 to n2 at time 0:

```
# send packet
link_1_2.bandwidth = 1000000.0
link_1_2.propagation = 1.0
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)
```

Running this simulation yields the following output:

Time Created	Packet ID	Time Received
0	1	1.008

The delay can be calculated by hand to show that the simulator is accurate:

$$D_{total} = D_{prop} + D_{trans} = 1 + \frac{L}{R} = 1 + \frac{8000}{1E6} = 1.008$$

For the next experiment, the bandwidth of the link changes to 100 bps and the propagation delay changes to 10 ms. One packet of 1000 bytes is sent from n1 to n2 at time 0:

```
# send packet
link_1_2.bandwidth = 100.0
link_1_2.propagation = 0.010
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)
```

Running this simulation yields the following output:

Time Created	Packet ID	Time Received
0	1	80.01

The delay can be calculated by hand to show that the simulator is accurate:

$$D_{total} = D_{prop} + D_{trans} = 0.010 + \frac{L}{R} = 0.010 + \frac{8000}{100} = 80.01$$

Now the bandwidth of the link changes back to 1 Mbps and the propagation delay remains at 10 ms. Three 1000-byte packets are sent from n1 to n2 at time 0, then one packet at time 2 seconds:

```
# send packets
link_1_2.bandwidth = 1000000.0
link_1_2.propagation = 0.010
p1 = packet.Packet(destination_address=n2.get_address('n1'),
        ident=1, protocol='delay', length=1000)
p2 = packet.Packet(destination_address=n2.get_address('n1'),
        ident=2, protocol='delay', length=1000)
p3 = packet.Packet(destination_address=n2.get_address('n1'),
        ident=3, protocol='delay', length=1000)
p4 = packet.Packet(destination_address=n2.get_address('n1'),
        ident=4, protocol='delay', length=1000)
Sim.scheduler.add(delay=0, event=p1, handler=n1.send_packet)
Sim.scheduler.add(delay=0, event=p2, handler=n1.send_packet)
Sim.scheduler.add(delay=0, event=p3, handler=n1.send_packet)
Sim.scheduler.add(delay=2, event=p4, handler=n1.send_packet)
```

Running this simulation yields the following output:

Time Created	Packet ID	Time Received
0	1	0.018
0	2	0.026
0	3	0.034
2.0	4	2.018

The delay can be calculated by hand to show that the simulator is accurate. Only one packet can transmit at a time, so any packet that is created at the same time as another packet has to wait until the previous packet is done transmitting before it can be transmitted. Packet 1 has 8 ms of transmission delay (L/R = 8000/1E6 = 0.008 s) and 10 ms of propagation delay, so it takes 18 ms. Packet 2 can't start transmitting packet 1 is done transmitting at time = 8 ms. It takes 8 ms longer, making 26 ms. Packet 3 can't start until packet 2 is done transmitting at time = 16 ms. It takes 16 ms longer than packet 1, making 34 ms. Packet 4 can start transmitting when it is created at time = 2 s because no other packet is transmitting, so it takes 18 ms and finishes at time = 2.018 s.

2 Three Nodes

Now that I have verified that the simulator works on a simple two-node network, I will explore a three-node network with nodes 1, 2, and 3. Nodes 1 and 2 are connected by a bidirectional link, as are nodes 2 and 3. Nodes 1 and 3 are only connected through node 2. Here is the code that sets this up:

```
# setup network
net = Network('three-nodes.txt')

# setup routes
n1 = net.get_node('n1')
n2 = net.get_node('n2')
n3 = net.get_node('n3')
link_1_2 = n1.links[0]
link_2_3 = n2.links[1]
n1.add_forwarding_entry(address=n2.get_address('n1'), link=link_1_2)
n1.add_forwarding_entry(address=n3.get_address('n2'), link=link_1_2)
n2.add_forwarding_entry(address=n3.get_address('n2'), link=link_2_3)
n2.add_forwarding_entry(address=n1.get_address('n2'), link=n2.links[0])
n3.add_forwarding_entry(address=n2.get_address('n3'), link=n3.links[0])
```

Note that the link from n1 to n2 is named link_1_2 and the link from n2 to n3 is named link_2_3. Also note that the forwarding entries were created to direct network traffic to the correct destination. The three-nodes.txt file that is references is used to create the three-node network:

```
# n1 -- n2 -- n3
#
n1 n2
n2 n1 n3
n3 n2
```

For the first experiment with the three-node network, both links have a propagation delay of 100 ms and a bandwidth of 1 Mbps. Node 1 transmits a 1 MB file as a stream of 1 kB packets to node 3:

```
# send packets
link_1_2.bandwidth = 1000000.0
link_2_3.bandwidth = 1000000.0
link_1_2.propagation = 0.100
link_2_3.propagation = 0.100
for i in xrange(1000):
        p = packet.Packet(source_address=n1.get_address('n2'),
                destination_address=n3.get_address('n2'),
                ident=i, protocol='delay', length=1000)
        Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)
```

Running this simulation yields the following output:

Time Created	Packet ID	Time Received
0	0	0.216
0	1	0.224
0	2	0.232
0	3	0.24
0	4	0.248
...	...	...
0	995	8.176
0	996	8.184
0	997	8.192
0	998	8.2
0	999	8.208

The delay can be calculated by hand to show that the simulator is accurate. First, the transmission delay for each 1000-byte packet can be calculated:

$$D_{trans} = \frac{L}{R} = \frac{8000}{1E6} = .008s$$

The transmission delay is 8 ms, and the propagation delay is given as 100 ms. Both links are the same, so there is no queueing delay. The total delay of the entire 1 MB stream can be computed by taking the time it takes to send the first packet over the first link, adding the transmission delay for each packet over the second link, and finally adding the propagation delay for the last packet over the second link:

$$D_{total} = (D_{trans} + D_{prop}) + (< packetcount > * D_{trans}) + D_{prop} = 108 + 8000 + 100 = 8208$$

The total delay is 8,208 ms, or 8.208 s. Only 0.200 s of that time is propagation delay, leaving 8.008 s of transmission delay. Changing both links to an upgraded rate of 1 Gbps drastically reduces the total delay:

```
# send packets
link_1_2.bandwidth = 1000000000.0
link_2_3.bandwidth = 1000000000.0
link_1_2.propagation = 0.100
link_2_3.propagation = 0.100
for i in xrange(1000):
        p = packet.Packet(source_address=n1.get_address('n2'),
                destination_address=n3.get_address('n2'),
                ident=i, protocol='delay', length=1000)
        Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)
```

Running this simulation yields the following output:

Time Created	Packet ID	Time Received
0	0	0.200016
0	1	0.200024
0	2	0.200032
0	3	0.20004
0	4	0.200048
...	...	...
0	995	0.207976
0	996	0.207984
0	997	0.207992
0	998	0.208
0	999	0.208008

Because the bandwidth was multiplied by 1000, the transmission delay for each 1000-byte packet is 1/1000 of what it was before. Before it was 8 ms; now it is 0.008 ms. The propagation delay remains at 100 ms. Using the same equation as before, the total delay can be calculated:

$$D_{total} = (D_{trans} + D_{prop}) + (< packetcount > * D_{trans}) + D_{prop} = 100.008 + 8 + 100 = 208.008$$

The total delay is 208.008 ms, or .208008 s. The propogation delay takes up .200 s of that time, leaving only .008008 s of transmission delay. Transmission delay was the dominant delay before, but upgrading the bandwidth led to the propagation delay dominating.

Now the bandwidth of the first link changes back to 1 Mbps, and the second link's bandwidth goes as low as 256 Kbps. The propagation delay remains at 100 ms for each link. Node 1 transmits a 1 MB file as a stream of 1 kB packets to node 3:

```
# send packets
link_1_2.bandwidth = 1000000.0
link_2_3.bandwidth = 256000.0
link_1_2.propagation = 0.100
link_2_3.propagation = 0.100
for i in xrange(1000):
        p = packet.Packet(source_address=n1.get_address('n2'),
                destination_address=n3.get_address('n2'),
                ident=i, protocol='delay', length=1000)
        Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)
```

Running this simulation yields the following output:

Time Created	Packet ID	Time Received
0	0	0.23925
0	1	0.2705
0	2	0.30175
0	3	0.333
0	4	0.36425
...	...	...
0	995	31.333
0	996	31.36425
0	997	31.3955
0	998	31.42675
0	999	31.458

The total delay can be calculated by taking the transmission delay for each packet over the first link, adding the propagation time of the last packet over the first link, adding the queueing delay of the last packet, adding the transmission delay of the last packet over the second link, and finally adding the propagation delay of the last packet over the second link. The queueing delay of a packet can be defined by d(p) = p * (dt2-dt1), where p is a 0-indexed packet number, dt2 is the transmission delay over the second (slower) link, and dt1 is the transmission delay over the first link. The delay dt1 is 8 ms, and dt2 = 8000/256000 = 31.25 ms.

$$D_{total} = (< packetcount > * D_{trans1}) + D_{prop} + D_{queue} + D_{trans2} + D_{prop} =$$
$$8000 + 100 + (999 * (31.25 - 8)) + 31.25 + 100 = 31458$$

The total delay is 31,458 ms, or 31.458 s. The propogation delay takes up .200 s of that time. The transmission delay takes up 8.03125 s of that time. The queueing delay dominates with 23.22675 s, which is 73.8% of the total time.

3 Queueing Theory

Now that it has been shown that queueing delay can be a significant percentage of total delay, there is a need to look deeper into queueing theory. An M/D/1 queue assumes an exponential distribution on the arrival rate, a deterministic service time, and a single queue. Looking at utilizations from 10% to 98% on a network simulating an exponential distribution for the arrival rate and meausuring the average queueing

delay for each packet will show how queueing delay is affected as utilization approaches 100%. Below is the code used to set up this experiment:

```
import sys
sys.path.append('..')

from src.sim import Sim
from src import node
from src import link
from src import packet

from networks.network import Network

import random

class Generator(object):
        def __init__(self,node,destination,load,duration):
                self.node = node
                self.load = load
                self.duration = duration
                self.destination = destination
                self.start = 0
                self.ident = 1

        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=self.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)

class DelayHandler(object):
        def receive_packet(self,packet):
                print packet.queueing_delay

def write_queue_average(util):

        print "UTIL:", util

        # reset scheduler
        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])
```

```
55          n2.add_forwarding_entry(address=n1.get_address('n2'),link=n2.links[0])
56
57          # setup app
58          d = DelayHandler()
59          net.nodes['n2'].add_protocol(protocol="delay",handler=d)
60
61          # setup packet generator
62          destination = n2.get_address('n1')
63          max_rate = 1000000/(1000*8)
64          load = util * max_rate
65          g = Generator(node=n1,destination=destination,load=load,duration=20)
66          Sim.scheduler.add(delay=0, event='generate', handler=g.handle)
67
68          # run the simulation
69          Sim.scheduler.run()
70
71  utils = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,0.95,0.98]
72  for util in utils:
73          write_queue_average(util)
```

Notice that the utilization values start at 0.1 (10%) and steadily increase up to 0.98 (98%). Also notice that the max_rate is calculated as bandwidth (1 Mbps) divided by packet size (1000 bytes). The load is defined as the utilization multiplied by the max rate. Running this simulation outputs a file that labels the util value and then lists the queueing delay that each packet experienced:

```
1  UTIL: 0.1
2  0.0
3  0.00101857862775
4  0.0
5  0.00798810853694
6  0.0
7  0.00590925711805
8  ...
9  UTIL: 0.98
10 0.00571336401862
11 0.0
12 0.0391423214485
13 0.040467162235
14 0.0100265130185
15 0.284257539877
16 0.315644164604
17 0.34917504953
18 0.411160811664
19 0.430353853684
20 ...
```

This output can be used to calculate the average queueing time for each utilization percentage. Assuming that the output is written to a file named experiment_data.txt, the following code will graph the data using the matplotlib library:

```
# Graph experimental average
data = [line[:-1] for line in open("experiment_data.txt", "r")]
util = []
delay = []
list_num = []
for line in data:
        if line[0] is "U":
                util.append(float(line[6:]))
                if list_num:
                        delay.append(sum(list_num) / len(list_num))
                list_num = []
        else:
                list_num.append(float(line))
delay.append(sum(list_num) / len(list_num))
average_line = plot(util, delay, label='Average')
```

We can graph the theoretical line on the same graph using the following code:

```
max_rate = 1000000.0/8000
mean_service_rate = 0.8 * max_rate
p = np.arange(0.0,1.0,0.01)
theory_line = plot(p,(1/(2*mean_service_rate))*(p/(1-p)), label='Theory')
```

Finally, running this code yields this graph, showing that the simulator is pretty close to matching the theoretical results: