Lab 1 Report

Nathan Davis

Jan 20, 2016

1 Two Nodes

1.1 Network Setup

For the first set of networks, we set up two nodes. we used the same structure for each part and varied the speed and propagation delay of the link. My first network looked like this:

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

# link configuration
n1 n2 1Mbps 1000ms
n2 n1 1Mbps 1000ms
```

The nodes are named n1 and n2 respectively and can be referenced that way in the script:

```
# setup network
net = Network('p1.1-network.txt')

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

Then, a forwarding entry must be set up from node 1 to node 2 and vice versa. The script requires two links to connect the two nodes because the links are uni-directional.

```
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])
```

In the forwarding entries, the first parameter is the address that will be targeted by the node and the second parameter is the link that will be used. In this example, the forwarding address is realized by searching backwards from the receiving node to the sending node. After the network has been set up, a packet can be sent by creating the packet and adding it to the scheduler as an event.

```
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)
```

When a packet reaches it's final address, we have to tell that node what to do with the packet. Packets within the simulator keep track of useful information such as their creation time, propagation delay, queueing delay, etc. A Handler is used to access that information after the simulation's run. For example, here's a simple handler that prints out the time a packet was created, it's identifier, and the total time it took the packet to get to its destination.

```
class DelayHandler(object):
    def receive_packet(self,packet):
        print packet.created, packet.ident, Sim.scheduler.current_time() - packet.created
```

This handler can be initialized and added to node 2 as the handler for the "delay" protocol. Any time a packet reaches that node using that protocol, the delay handler is called with that packet.

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

The simulation is reset at the beginning to clear any existing data and finally, after the network has been set up, the handler is in place, and events are on the queue, the simulator can be started. All of it put together looks something like this:

```
    # Reset the simulation
    Sim.scheduler.reset()

    # Set up the network
    # Set up the handler
    # Queue a packet to send

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

1.2 Problem 1

For the first problem, we set the bandwidth of the link to 1Mbps with a 1 second propagation delay. The total time this packet will take to arrive will be $TransmissionDelay + PropagationDelay$. Transmission delay is given by the formula: $\frac{L}{R}$ where L is the length of the message in bits and R is the rate of the link in bits/second. For our problem, the propagation delay is given to us.

In problem 1, the transmission delay is $\frac{1000bytes}{1Mbps} = \frac{8000bits}{\frac{1000000bits}{second}} = 0.008seconds$. The propagation delay is $1second$ and so we expect the total delay to be $1.008seconds$.

Running the script, we get the following output:

```
    0 1 1.008
```

This means that there was one packet (ID:1) sent at time 0 that was received at time 1.008, exactly as predicted.

1.3 Problem 2

The setup to the second problem is very similar to the first. The only change is in the configuration of the network. The links are set to a speed of 100bps with a propagation delay of 10ms:

```
# link configuration
n1 n2 100bps 10ms
n2 n1 100bps 10ms
```

For this problem, the transmission delay is $\frac{1000bytes}{100bps} = \frac{8000bits}{\frac{100bits}{second}} = 80seconds$. The propagation delay is 10ms, so we expect the total delay to be 80.01 seconds.

The output we get from running the script is

```
0 1 80.01
```

This means that there was one packet (ID:1) sent at time 0 seconds that was received at time 80.01 seconds.

1.4 Problem 3

For problem 3, much of the network setup is the same. We alter our configuration slightly to get the correct rate and propagation delay.

```
# link configuration
n1 n2 1Mbps 10ms
n2 n1 1Mbps 10ms
```

Instead of sending one packet, we need to send 4 for this problem: 3 at time t=0 and 1 at time t=2. This is achieved by setting different events in the simulator's queue.

```
# send three packets at t=0

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)
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)

# send one packet at t=2
p4 = packet.Packet(destination_address=n2.get_address('n1'),ident=4,
    protocol='delay',length=1000)
Sim.scheduler.add(delay=2, event=p4, handler=n1.send_packet)
```

Each of the packets is sent from node 1 to node 2 using the same method as problems 1 and 2. Each packet is given a unique identifier ranging from 1 to 4. Packets 1-3 are sent with a delay of 0 and packet

4 is sent with a delay of 2.

In order to estimate the arrival time of each packet, we have to factor in some queueing delay as well now. The first packet should send at a normal rate, that is $TransmissionDelay = \frac{1000bytes}{1Mbps} = \frac{8000bits}{\frac{1000000bits}{second}} = 0.008seconds$ $Totaldelay = TransmissionDelay + PropagationDelay = 0.008seconds + 10ms = 0.018seconds$. The second packet will have to wait until the first packet has been put on the line and so it has a Queueing delay equal to the first packet's transmission delay. The total delay for the second packet should be $Totaldelay = QueueingDelay + TransmissionDelay + PropagationDelay = 0.008s + 0.008s + 0.01s = 0.026s$. Similarly, the third packet will have queueing delay equal to twice the transmission delay. For the third packet, $Totaldelay = (0.008s \times 2) + 0.008s + 0.01s = 0.034s$. The fourth packet is sent at time t=2 which means that the first three packets should have been sent already and there should be no queuing delay so it should arrive after the same delay as the first packet, 0.018 seconds. However, since it was sent at time t=2, it will arrive at time t=2.018.

Here we can see the results from the script and compare them to our expected delay times.

```
0 1 0.018
0 2 0.026
0 3 0.034
2.0 4 2.018
```

We see our estimations confirmed, with packet 1 arriving at t=0.018 and packets 2 and 3 arriving at 0.008s intervals afterwards. Packet 4 arrives at t=2.018.

2 Networks with Three Nodes

2.1 Network Setup

The setup for a network with three nodes is very similar to the one with two nodes except that, well, it has three nodes. Each connection must be listed in the configuration, so node 2 must have connections to both nodes 1 and 3. Node 3 must have a connection back to node 2. Each link's speed and propagation delay must be specified, similar to the network with two nodes configuration. Here is an example configuration for the first network with three nodes.

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

# link configuration
n1 n2 1Mbps 100ms
n2 n1 1Mbps 100ms
n2 n3 1Mbps 100ms
n3 n2 1Mbps 100ms
```

Similarly, the nodes are added in a similar fashion inside the script that sets ups the simulator. One extra consideration here is that a forwarding table must be set up from every node to every other node. For example, if you want to send packets from node 1 through node 2 to node 3, you must have an entry in node 1's forwarding table for node 3, even though node 1 doesn't have a direct connection to node 3.

Here's a list of all the forwarding entries for the three node network above.

```
    # From n1 to n2
    n1.add_forwarding_entry(address=n2.get_address('n1'),link=n1.links[0])
    # From n1 to n3
    n1.add_forwarding_entry(address=n3.get_address('n2'),link=n1.links[0])

    # From n2 to n1
    n2.add_forwarding_entry(address=n1.get_address('n2'),link=n2.links[0])
    # From n2 to n3
    n2.add_forwarding_entry(address=n3.get_address('n2'),link=n2.links[1])

    # From n3 to n2
    n3.add_forwarding_entry(address=n1.get_address('n2'),link=n3.links[0])
    # From n3 to n1
    n3.add_forwarding_entry(address=n2.get_address('n3'),link=n3.links[0])
```

Notice the forwarding entry in node 1's table to node 3. It says that when node 1 gets a packet addressed to the port on node 3 where node 2 is connected, use link 0 from node 1 to send the packet. This logic is a little difficult to follow and it may take a few times reading that line to understand it fully.

The handler, which we call DelayHandler again will be added to node 3 this time since that will be the final node for our simulations.

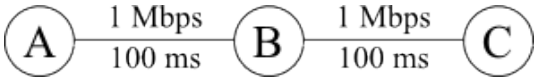
```
    # setup app
    d = DelayHandler()
    net.nodes['n3'].add_protocol(protocol="delay",handler=d)
```

Also, we want to print out more information for each packet received this time. We'll print out the time the packet was received, the identifier, when it was created, the total delay, transmission delay, propagation delay, and queueing delay. These will help us more accurately see what's happening.

```
    print Sim.scheduler.current_time(),packet.ident,packet.created,
        Sim.scheduler.current_time() - packet.created,
        packet.transmission_delay, packet.propagation_delay, packet.queueing_delay
```

This concludes the configuration of the network. The simulator must be reset at the beginning and run in each simulation.

2.2 Problem 1



Problem 1 asks us to send a 1MB file in 1000 packets of 1000kB each. We'll do this in a for loop to avoid using excess space. We'll loop for 1000 times and give that loop number to the packet as the identifier. Each packet will be sent to node 3's address for the link to node 2. Once again, the logic there is difficult to follow, but know that $n3.get_address('n2')$ is the way to get node 3's address because the packet will be sent from node 2 to node 3. The protocol is 'delay' which means node 3 will give the packet to the DelayHandler once it reaches node 3. Each packet is set to 1000 bytes.

We add all those packets to the scheduler as events and we give them to node 1 by calling it's $send_packet$ method as the packet handler. All the packets are queued at time t=0 which means that the entire file is ready to send and should send as fast as the network can handle it.

```
# send 1000 packet (1000 bytes each)

for i in range(0, 1000):
    p = packet.Packet(destination_address=n3.get_address('n2'),ident=i,
        protocol='delay',length=1000)
    Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)
```

Finally we'll run the simulation:

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

Before running the simulation, we'll calculate the delay by hand. Since the 2 links have the same rate and propagation delay, there shouldn't be any delay caused from queueing. Let's find each of the parts on their own. The transmission delay for one packet over one link is

$\frac{L}{R} = \frac{1kB}{1Mbps} = \frac{8000bits}{1\times10^6 bps} = 0.008s$

The propagation delay for one link is 0.1s. The total delay is going to be the transmission delay times 1000 (one for each packet), add the propagation delay twice, and add one more transmission delay because there are two links.

$0.008s \times 1000packets + 0.1s \times 2 + 0.008s = 8.208s$

Running the simulation, we get 1000 lines of print out. The last few lines are

```
8.176  995 0 8.176  0.016  0.2  7.96
8.184  996 0 8.184  0.016  0.2  7.968
8.192  997 0 8.192  0.016  0.2  7.976
8.2    998 0 8.2    0.016  0.2  7.984
8.208  999 0 8.208  0.016  0.2  7.992
```

The first column tells us the arrival time for each packet. We see that the last packet arrives at 8.208 seconds as expected. The last column in the simulation should show us the queueing delay. However, we see that it considers any packet that's waiting to be in a queue. In a sense this is correct, but because the packet hasn't travelled "into" the network until it's gone across at least one link, we consider the queueing delay to be zero. The total transmission time for each packet is 0.016 seconds and the total propagation delay for each packet is 0.2 seconds.

Here we can see that the transmission delay dominates. It accounts for 8.008 seconds of the total delay. (We include queueing delay in this case because transmission is what causes the queueing delay)

Now we'll change the two link rates to 1Gbps each. Everything else stays the same, so the only change is in the network configuration.

```
# link configuration
n1 n2 1Gbps 100ms
n2 n1 1Gbps 100ms
n2 n3 1Gbps 100ms
n3 n2 1Gbps 100ms
```

To calculate the total delay, we'll use a similar formula to last time but with a new transmission delay using the new rate. The transmission delay for a single packet over a single link is

$\frac{L}{R} = \frac{1kB}{1Gbps} = \frac{8000bits}{1\times10^9 bps} = 0.000008s$

The total delay can then be calculated to

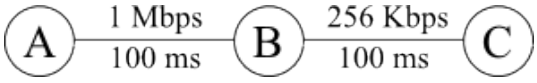
$0.000008s \times 1000packets + 0.1s \times 2 + 0.000008s = 0.20808s$

When we run the simulation, here's the last few lines:

```
0.207976 995 0 0.207976 1.6e-05 0.2 0.00796
0.207984 996 0 0.207984 1.6e-05 0.2 0.007968
0.207992 997 0 0.207992 1.6e-05 0.2 0.007976
0.208    998 0 0.208    1.6e-05 0.2 0.007984
0.208008 999 0 0.208008 1.6e-05 0.2 0.007992
```

We can see from the first column of the last row that our expectation has been verified. The total delay was 0.20808 seconds with 0.008008 seconds being transmission delay and 0.2 seconds being propagation delay. In this case propagation delay dominates.

2.3 Problem 2



For the second problem, we notice that the links have different speeds. This involves a change to our configuration file.

```
# link configuration
n1 n2 1Mbps 100ms
n2 n1 1Mbps 100ms
n2 n3 256Kbps 100ms
n3 n2 256Kbps 100ms
```

The forwarding entries, delay handler, and protocol all remain the same as problem 1's network setup as well as the sending of the 1000 packets in the loop. The difference is in the results we expect.

We're expecting the queueing delay to be a factor this time. The first packet shouldn't experience queueing delay and each subsequent packet should experience more and more, with the 1000th packet experiencing the most. The equation for the queueing delay as a function of k where k is the packet number out of 1000 can be given with the equation $(k-1) \times 23.24ms$. For the very last packet, this means

that the queueing delay will be $999 \times 23.24ms = 23.22675s$. We can find the total delay by taking the transmission delay for all 1000 packets over the first link, adding the propagation delay, the maximum queueing delay, the transmission delay for the last packet over the second link, and the propogation delay over the second link. This gives us

$$8ms \times 1000 + 100ms + 23.22675s + 31.25ms + 100ms = 31.458seconds$$

The last 5 lines of the results from running the simulation are

```
31.333    995 0 31.333    0.03925 0.2 31.09375
31.36425  996 0 31.36425  0.03925 0.2 31.125
31.3955   997 0 31.3955   0.03925 0.2 31.15625
31.42675  998 0 31.42675  0.03925 0.2 31.1875
31.458    999 0 31.458    0.03925 0.2 31.21875
```

We can see that our prediction of 31.458 seconds was correct as that is the time that the last packet was received by node 3.

3 Queueing Theory

For this portion of the assignment we were asked to set up an experiment involving queueing theory. We chose to set up a three node network where the first link would be a 10Gbps link with no propogation delay and the second link would be a 1Mbps link with 1ms propagation delay. The thinking behind this is that the packets would get through the first link very quickly and then be bottlenecked and queued on the second link.

```
# link configuration
n1 n2 10Gbps 0ms
n2 n1 10Gbps 0ms
n2 n3 1Mbps 1ms
n3 n2 1Mbps 1ms
```

Configuring the network was very similar to the three node network problems.

In order to simulate packets being sent at a certain rate, we used the generator from the example delay.py code. The generator will run for a certain amount of time and generate packets with an exponential distribution, using the load as the seed.

```
# 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)
```

We then created a loop that would go through every load from 10 to 90 by 10's and run the generator

for 1000 seconds and save every packet's queueing delay to a file. The file would be named by the rate. For example, for rate 0.50, the file would be named '50.txt'.

```
    for r in [1, 10, 20, 30, 40, 50, 60, 70, 80, 85, 90, 92, 95, 98]:

        # setup network
        # ...

        # setup app
        d = DelayHandler(str(r) + '.txt')
        net.nodes['n3'].add_protocol(protocol="delay",handler=d)

        # setup packet generator
        destination = n3.get_address('n2')
        max_rate = 1000000/(1000*8)
        load = (float(r)/100.0)*max_rate
        g = Generator(node=n1,destination=destination,load=load,duration=1000)
        Sim.scheduler.add(delay=0, event='generate', handler=g.handle)

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

The result was a list of files organized by utilization rate containing the queueing delays for every packet generated in that 1000 seconds of simulator time.

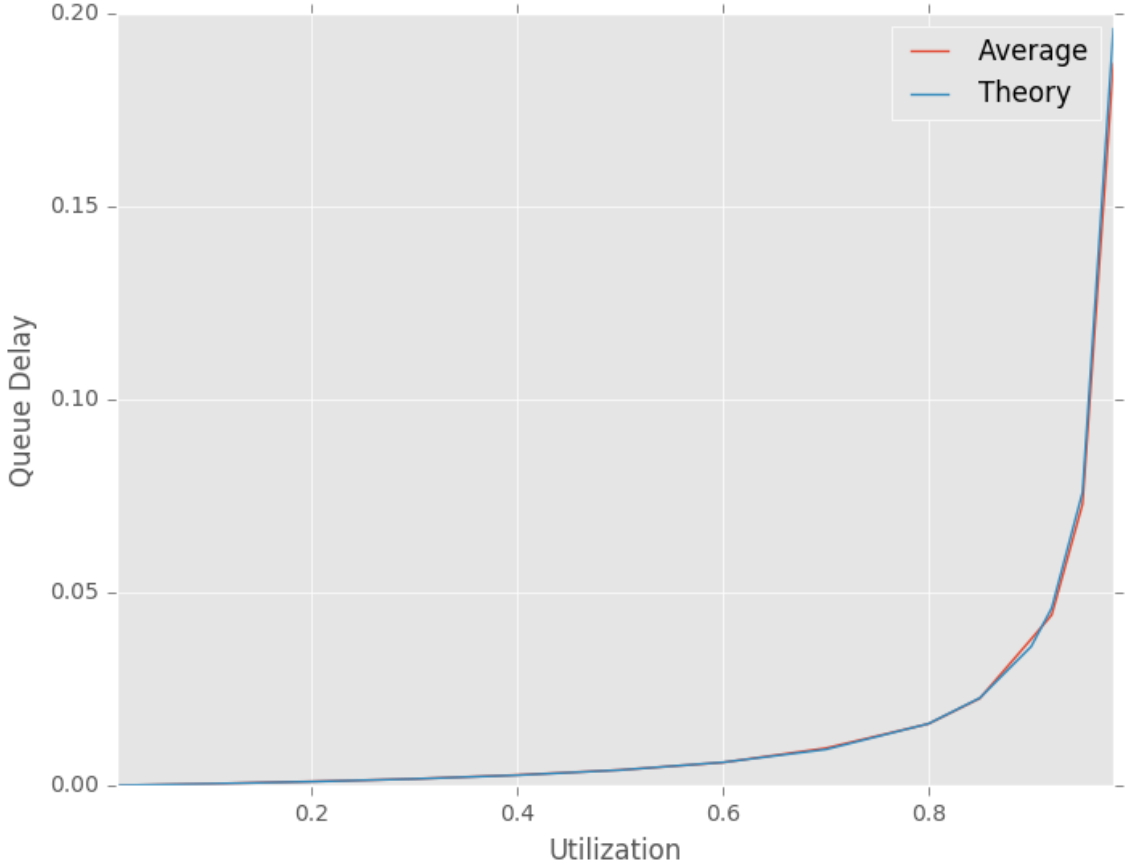
```
$ ls results/
1.txt   10.txt  20.txt  30.txt
40.txt  50.txt  60.txt  70.txt
80.txt  85.txt  90.txt  92.txt
95.txt  98.txt
```

We then created a script that would average the queueing delays for each rate and output them to a csv along with the expected rate given by the theoretical queueing delay equation.

```
    Utilization,Average,Theory
    0.01,0.000034,0.000040
    0.1,0.000434,0.000444
    0.2,0.001035,0.001000
    0.3,0.001715,0.001714
    0.4,0.002723,0.002667
    0.5,0.004020,0.004000
    0.6,0.005966,0.006000
    0.7,0.009670,0.009333
    0.8,0.015975,0.016000
    0.85,0.022571,0.022667
    0.9,0.037933,0.036000
    0.92,0.044196,0.046000
    0.95,0.072916,0.076000
    0.98,0.186995,0.196000
```

These results were easy to put into a graph, with one line representing the average from the simulator

and the other representing the theory.



As shown by the graph. The results from the simulator very closely match those given by the equation. This would support the queueing theorem which states that as utilization approaches 1.00, queue waiting time approaches infinity.