

Network Simulation (Lab 1) Report

Luke Dickinson

1/25/2017

1 Introduction

In order to measure the way a basic network performs, we used a network simulator called Bene to simulate a number of different situations. In each of these situations, we will show how the simulation is set up, what the simulation resulted to, and alternate calculations to show that the result from the simulation is correct.

1.0.1 Terms and Abbreviations

In order to increase readability, abbreviations will be used throughout the paper. The following terms may be use in the abbreviated form:

$$\textit{Packet} = p$$
$$\textit{Node} = n$$
$$\textit{CreationTime} = cTime$$
$$\textit{ProcessingDelay} = \textit{procDelay}$$
$$\textit{PropagationDelay} = \textit{propDelay}$$
$$\textit{TransmissionDelay} = \textit{tDelay}$$
$$\textit{QueueingDelay} = \textit{qDelay}$$

2 Two Nodes

This section includes a network that consisting of 2 nodes. The setup required for the simulations in this section is the same for each case. To setup the simulation, we first reset the scheduler for the simulation. Next, we set up the network by create a Network object by passing a network configuration file (which will be described in each simulation). After that, we set up the route information between the nodes. Finally, we setup a handler to print out information about packets received at node₂. The following code snippet performs the setup:

```
1 # parameters
2 Sim.scheduler.reset()
3
4 # setup network
5 net = Network('lab1-onehopX.txt')
6
7 # setup routes
8 n1 = net.get_node('n1')
9 n2 = net.get_node('n2')
10 n1.add_forwarding_entry(address=n2.get_address('n1'), link=n1.links[0])
```

```

11 n2.add_forwarding_entry(address=n1.get_address('n2'), link=n2.links[0])
12
13 # setup app
14 d = DelayHandler()
15 net.nodes['n2'].add_protocol(protocol="delay", handler=d)

```

2.1 First simulation of a two node network

2.1.1 Simulation configuration

For our first simulation of a two node network, we set the bandwidth between the nodes to 1 Mbps, and set the propagation delay between the nodes to 1 second. We send a single packet of size 1000 bytes from one node to the other at time set to 0 seconds. To do this we use the following configuration file:

```

1 # n1 — n2
2 n1 n2
3 n2 n1
4
5 # link configuration
6 n1 n2 1Mbps 1000ms
7 n2 n1 1Mbps 1000ms

```

This configuration file above is called 'lab1-onehopA.txt.' This is to be inserted into our Network constructor referred to above. Once the network is set up, we will send our single packet to node₂ and then run the simulation. This is shown in the following code snippet:

```

1 #send packet(s)
2 p = Packet(destination_address=n2.get_address('n1'), \
3             ident=1, protocol='delay', length=1000)
4 Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)
5
6 # run the simulation
7 Sim.scheduler.run()

```

2.1.2 Output of the simulation

```

1 (1.008, 1, 0, 1.008, 0.008, 1.0, 0)

```

This output shows that the first packet was created at 0 seconds, its identifier was 1, and the time it was received by node₂ was at 1.008 seconds.

2.1.3 Verifying calculations

We are able to calculate the time our packet is received at node₂ by adding the creation time of the packet, the processing delay, propagation delay, transmission delay, and queueing delay for this packet. In our simulation we are assuming that processing delay is 0. In situations of only a single packet, there is no queueing delay.

$$\begin{aligned}
cTime_{p1} &= 0 \text{ seconds} \\
procDelay &= 0 \text{ seconds} \\
propDelay_{n1} &= 1 \text{ second} \\
tDelay_{p1 \text{ at } n1} &= \frac{\text{size of the packet}}{\text{link rates}} \\
&= \frac{1000 \text{ bytes}}{1 \text{ Mbps}} \\
&= \frac{8000 \text{ bits}}{1 \text{ Mbps}} \\
&= 0.008 \text{ seconds} \\
qDelay_{p1 \text{ at } n1} &= 0 \text{ seconds}
\end{aligned}$$

$$\begin{aligned}
Time \text{ Packet}_1 \text{ Is Received At Node}_2 &= cTime_{p1} + procDelay + propDelay_{n1} + \\
&\quad + tDelay_{p1 \text{ at } n1} + qDelay_{p1 \text{ at } n1} \\
&= 0 + 0 + 1 + 0.008 + 0 \\
&= 1.008
\end{aligned}$$

From our calculations we see that packet₁ is received at node₂ at 1.008 seconds, which matches our simulation.

2.2 Second simulation of a two node network

2.2.1 Simulation configuration

For our second simulation of a two node network, we set the bandwidth between the nodes to 100 bps, and set the propagation delay between the nodes to 10 milliseconds. We send a single packet of size 1000 bytes from one node to the other at time set to 0 seconds. To do this, we use the following configuration file:

```

1 # n1 — n2
2 #
3 n1 n2
4 n2 n1
5
6 # link configuration
7 n1 n2 100bps 10ms
8 n2 n1 100bps 10ms

```

This configuration file above is called 'lab1-onehopB.txt.' This is to be inserted into our Network constructor referred to above. Once the network is set up, we will send our single packet to node₂ and then run the simulation. This is shown in the following code snippet:

```

1 #send packet(s)
2 p = Packet(destination_address=n2.get_address('n1'), \
3             ident=1, protocol='delay', length=1000)
4 Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)
5
6 # run the simulation
7 Sim.scheduler.run()

```

2.2.2 Output of the simulation

```
1 (80.01, 1, 0, 80.01, 80.0, 0.01, 0)
```

This output shows that the first packet was created at 0 seconds, its identifier was 1, and the time it was received by node₂ was at 80.01 seconds.

2.2.3 Verifying calculations

We are able to calculate the time our packet is received at node₂ by adding the creation time of the packet, the processing delay, propagation delay, transmission delay, and queueing delay for this packet. In our simulation we are assuming that processing delay is 0. In situations of only a single packet, there is no queueing delay.

$$\begin{aligned} cTime_{p1} &= 0 \text{ seconds} \\ procDelay &= 0 \text{ seconds} \\ propDelay_{n1} &= 10 \text{ milliseconds} \\ tDelay_{p1 \text{ at } n1} &= \frac{\text{size of the packet}}{\text{link rates}} \\ &= \frac{1000 \text{ bytes}}{100 \text{ bps}} \\ &= \frac{8000 \text{ bits}}{100 \text{ bps}} \\ &= 80 \text{ seconds} \\ qDelay_{p1 \text{ at } n1} &= 0 \text{ seconds} \end{aligned}$$

$$\begin{aligned} Time \text{ Packet}_1 \text{ Is Received At Node}_2 &= cTime_{p1} + procDelay + propDelay_{n1} + \\ &\quad + tDelay_{p1 \text{ at } n1} + qDelay_{p1 \text{ at } n1} \\ &= 0 + 0 + 0.01 + 80 + 0 \\ &= 80.01 \end{aligned}$$

From our calculations we see that packet₁ is received at node₂ at 80.01 seconds, which matches our simulation.

2.3 Third simulation of a two node network

2.3.1 Simulation configuration

For our third simulation of a two node network, we set the bandwidth between the nodes to 1 Mbps, and set the propagation delay between the nodes to 10 milliseconds. We send 3 packets of size 1000 bytes from one node to the other at time set to 0 seconds. Then we send a single packet of size 1000 bytes from one node to the other at time set to 2 seconds. To do this we use the following configuration file:

```
1 # n1 — n2
2 #
3 n1 n2
4 n2 n1
5
6 # link configuration
7 n1 n2 1Mbps 10ms
8 n2 n1 1Mbps 10ms
```

This configuration file above is called 'lab1-onehopC.txt.' This is to be inserted into our Network constructor referred to above. Once the network is set up, we will send our packets to node₂ and then run the simulation. This is shown in the following code snippet:

```

1 #send packet(s)
2 p = Packet(destination_address=n2.get_address('n1'), \
3             ident=1, protocol='delay', length=1000)
4 Sim.scheduler.add(delay=0, event=p, handler=n1.send_packet)
5
6 p2 = Packet(destination_address=n2.get_address('n1'), \
7             ident=2, protocol='delay', length=1000)
8 Sim.scheduler.add(delay=0, event=p2, handler=n1.send_packet)
9
10 p3 = Packet(destination_address=n2.get_address('n1'), \
11            ident=3, protocol='delay', length=1000)
12 Sim.scheduler.add(delay=0, event=p3, handler=n1.send_packet)
13
14 p4 = Packet(destination_address=n2.get_address('n1'), \
15            ident=4, protocol='delay', length=1000)
16 Sim.scheduler.add(delay=2, event=p4, handler=n1.send_packet)
17
18 # run the simulation
19 Sim.scheduler.run()

```

2.3.2 Output of the simulation

```

1 (0.018000000000000002, 1, 0, 0.018000000000000002, 0.008, 0.01, 0)
2 (0.026000000000000002, 2, 0, 0.026000000000000002, 0.008, 0.01, 0.008)
3 (0.034, 3, 0, 0.034, 0.008, 0.01, 0.016)
4 (2.018, 4, 2.0, 0.0179999999999999794, 0.008, 0.01, 0.0)

```

This output shows that the first packet was created at 0 seconds, its identifier was 1, and the time it was received by node₂ was at 0.018 seconds. The second packet was created at 0 seconds, its identifier was 2, and the time it was received by node₂ was at 0.0260 seconds. The third packet was created at 0 seconds, its identifier was 3, and the time it was received by node₂ was at 0.340 seconds. The fourth packet was created at 2 seconds, its identifier was 4, and the time it was received by node₂ was at 2.018 seconds.

2.3.3 Verifying calculations

We are able to calculate the time each of our packets are received at node₂ by adding the creation time of the packet, the processing delay, propagation delay, transmission delay, and queueing delay for a packet. In our simulation we are assuming that processing delay is 0.

Calculating the time packet₁ is received at node₂

$$\begin{aligned}
cTime_{p1} &= 0 \text{ seconds} \\
procDelay &= 0 \text{ seconds} \\
propDelay_{n1} &= 10 \text{ milliseconds} \\
tDelay_{p1 \text{ at } n1} &= \frac{\text{size of the packet}}{\text{link rates}} \\
&= \frac{1000 \text{ bytes}}{1 \text{ Mbps}} \\
&= \frac{8000 \text{ bits}}{1 \text{ Mbps}} \\
&= 0.008 \text{ seconds} \\
qDelay_{pX \text{ at } nY} &= k_X * (tDelay_{pX \text{ at } nY} - tDelay_{packetX \text{ at } n(Y-1)}) \\
&\text{where } k_X = \text{the number of packets at node}_y \text{ when packet}_x \text{ arrives}
\end{aligned}$$

$$\begin{aligned}
k_1 &= 0 \\
&\text{Packet}_1 \text{ is the first packet and will not enter a queue, so } k_1 = 0 \\
qDelay_{p1 \text{ at } n1} &= k_1 * (tDelay_{p1 \text{ at } n1} - tDelay_{p1 \text{ at } n(1-1)}) \\
&= (0)(0.008 - 0) \\
&= 0
\end{aligned}$$

$$\begin{aligned}
\text{time packet}_1 \text{ is received at node}_2 &= cTime_{p1} + procDelay + propDelay_{n1} + \\
&\quad + tDelay_{p1 \text{ at } n1} + qDelay_{p1 \text{ at } n1} \\
&= 0 + 0 + 0.01 + 0.008 + 0 \\
&= 0.018
\end{aligned}$$

From our calculations we see that packet₁ is received at node₂ at 0.018 seconds,
which matches our simulation.

Calculating the time packet₂ is received at node₂

$$cTime_{p2} = 0 \text{ seconds}$$

$$procDelay = 0 \text{ seconds}$$

$$propDelay_{n1} = 10 \text{ milliseconds}$$

As packet 1 and 2, are the same size, the value for

tDelay_{p1 at n1} is the same as tDelay_{p2 at n1}

So,

$$tDelay_{p2 at n1} = 0.008 \text{ seconds}$$

$$qDelay_{pX at nY} = k_X * (tDelay_{pX at nY} - tDelay_{packetX at n(Y-1)})$$

where k_X = the number of packets at node_y when packet_x arrives

$$k_2 = 1$$

Packets 1, 2, and 3 arrive at node₁ at 0 seconds.

When these are the these packets created,

there are no other packets at node₁.

As packet₁ is the only packet in front of packet₂, $k_2 = 1$.

$$qDelay_{p2 at n1} = k_2 * (tDelay_{p2 at n1} - tDelay_{p2 at n(1-1)})$$

$$= (1)(0.008 - 0)$$

$$= 0.008$$

$$time \text{ packet}_2 \text{ is received at node}_2 = cTime_{p2} + procDelay + propDelay_{n1} +$$

$$+ tDelay_{p2 at n1} + qDelay_{p2 at n1}$$

$$= 0 + 0 + 0.01 + 0.008 + 0.008$$

$$= 0.026$$

From our calculations we see that packet₂ is received at node₂ at 0.026 seconds,

which matches our simulation.

Calculating the time packet₃ is received at node₂

$$cTime_{p3} = 0 \text{ seconds}$$

$$procDelay = 0 \text{ seconds}$$

$$propDelay_{n1} = 10 \text{ milliseconds}$$

As packet 1 and 3 are the same size, the value for

$tDelay_{p1 \text{ at } n1}$ is the same as $tDelay_{p3 \text{ at } n1}$

So,

$$tDelay_{p3 \text{ at } n1} = 0.008 \text{ seconds}$$

$$qDelay_{pX \text{ at } nY} = k_X * (tDelay_{pX \text{ at } nY} - tDelay_{packetX \text{ at } n(Y-1)})$$

where k_X = the number of packets at node_y when packet_x arrives

$$k_3 = 2$$

Packets 1, 2, and 3 arrive at node₁ at 0 seconds.

When these are the these packets created,

there are no other packets at node₁.

As packet₁ and packet₂ are the only packets in front of packet₃, $k_3 = 2$.

$$qDelay_{p3 \text{ at } n1} = k_3 * (tDelay_{p3 \text{ at } n1} - tDelay_{p3 \text{ at } n(1-1)})$$

$$= (2)(0.008 - 0)$$

$$= 0.016$$

$$time \text{ packet}_3 \text{ is received at node}_2 = cTime_{p2} + procDelay + propDelay_{n1} +$$

$$+ tDelay_{p2 \text{ at } n1} + qDelay_{p2 \text{ at } n1}$$

$$= 0 + 0 + 0.01 + 0.008 + 0.016$$

$$= 0.034$$

From our calculations we see that packet₃ is received at node₂ at 0.034 seconds,

which matches our simulation.

Calculating the time packet₄ is received at node₂

$$cTime_{p4} = 2 \text{ seconds}$$

$$procDelay = 0 \text{ seconds}$$

$$propDelay_{n1} = 10 \text{ milliseconds}$$

As packet 1 and 4 are the same size, the value for

$tDelay_{p1 \text{ at } n1}$ is the same as $tDelay_{p4 \text{ at } n1}$

So,

$$tDelay_{p4 \text{ at } n1} = 0.008 \text{ seconds}$$

$$qDelay_{pX \text{ at } nY} = k_X * (tDelay_{pX \text{ at } nY} - tDelay_{packetX \text{ at } n(Y-1)})$$

where k_X = the number of packets at node_y when packet_x arrives

$$k_3 = 2$$

Packet₄ arrives at node₁ at 2 seconds.

The packet previous to packet₄, packet₃, leaves node₁ at

(packet₃ arrival time to node₂) - (propDelay_{n1}) =

$$0.034 - 0.01 = 0.024 \text{ seconds.}$$

Because the previous packet to packet₄, packet₃, leaves node₁

before packet₄ arrives, packet₄ does not enter the queue. So $k = 0$.

$$qDelay_{p4 \text{ at } n1} = k_4 * (tDelay_{p4 \text{ at } n1} - tDelay_{p4 \text{ at } n(1-1)})$$

$$= (0)(0.008 - 0)$$

$$= 0$$

$$time \text{ packet}_4 \text{ is received at node}_2 = cTime_{p2} + procDelay + propDelay_{n1} +$$

$$+ tDelay_{p2 \text{ at } n1} + qDelay_{p2 \text{ at } n1}$$

$$= 2 + 0 + 0.01 + 0.008 + 0$$

$$= 2.018$$

From our calculations we see that packet₄ is received at node₂ at 2.018 seconds,

which matches our simulation.

3 Three Nodes

This section includes a network that consisting of 3 nodes. The setup required for the simulations in this section is the same for each case. To setup the simulation, we first reset the scheduler for the simulation. Next, we set up the network by create a Network object by passing a network configuration file (which will be described in each simulation). After that, we set up the route information between the nodes. Finally, we setup a handler to print out information about packets received at node₃. The following code snippet performs the setup:

```

1 # parameters
2 Sim.scheduler.reset()
3
4 # setup network
5 net = Network('lab1-twohopX.txt')
6
7 # setup routes

```

```

8 n1 = net.get_node('n1')
9 n2 = net.get_node('n2')
10 n3 = net.get_node('n3')
11 n1.add_forwarding_entry(address=n2.get_address('n1'), link=n1.links[0])
12 n1.add_forwarding_entry(address=n3.get_address('n2'), link=n1.links[0])
13 n2.add_forwarding_entry(address=n1.get_address('n2'), link=n2.links[0])
14 n2.add_forwarding_entry(address=n3.get_address('n2'), link=n2.links[1])
15 n3.add_forwarding_entry(address=n2.get_address('n3'), link=n3.links[0])
16 n3.add_forwarding_entry(address=n1.get_address('n2'), link=n3.links[0])
17
18 # setup app
19 d = DelayHandler()
20 net.nodes['n3'].add_protocol(protocol="delay", handler=d)

```

3.1 First simulation of a three node network - Part 1

3.1.1 Simulation configuration

For our first simulation of a three node network, we set the bandwidth between the node₁ and node₂ to 1 Mbps, the bandwidth from node₂ to node₃ to 1 Mbps, set the propagation delay between the node₁ and node₂ to 100 milliseconds, and set the propagation delay between the node₂ to node₃ to 100 milliseconds. We send a 1000 packets, each 1000 bytes in size through the network with the source being node₁ and the destination being node₃. To do this we use the following configuration file:

```

1 # n1 --- n2 --- n3
2 n1 n2
3 n2 n1
4 n2 n3
5 n3 n2
6 # link configuration
7 n1 n2 1Mbps 100ms
8 n2 n1 1Mbps 100ms
9 n2 n3 1Mbps 100ms
10 n3 n2 1Mbps 100ms

```

This configuration file above is called 'lab1-twohopA.txt.' This is to be inserted into our Network constructor referred to above. Once the network is set up, we will send a single file in 1000 packets to node three and then run the simulation. In order to avoid queueing issues (like overflowing the simulated queue), we will control the time each packet is sent so that each packet will be begin its process as its previous packet leaves node₁. This is shown in the following code snippet:

```

1 trasmissionDelay = 8.0/1000.0
2 # send packets
3 for i in range(1,1001):
4     calculatedDelay = (i-1) * trasmissionDelay
5     p = Packet(destination_address=n3.get_address('n2'), ident=i, protocol='delay',
6               length=1000)
7     Sim.scheduler.add(delay=calculatedDelay, event=p, handler=n1.send_packet)
8 # run the simulation
9 Sim.scheduler.run()

```

3.1.2 Output of the simulation

Last 5 entries:

```

1 (8.1760000000000007, 996, 7.96, 0.21600000000000073, 0.016, 0.2, 6.217248937900877e-15)
2 (8.1840000000000006, 997, 7.968, 0.21600000000000064, 0.016, 0.2, 6.217248937900877e-15)
3 (8.1920000000000007, 998, 7.976, 0.21600000000000073, 0.016, 0.2, 6.217248937900877e-15)
4 (8.2000000000000006, 999, 7.984, 0.21600000000000064, 0.016, 0.2, 6.217248937900877e-15)
5 (8.2080000000000007, 1000, 7.992, 0.21600000000000073, 0.016, 0.2, 6.217248937900877e-15)

```

The entire file is transferred when the last packet arrives at node₃. This output shows that the last packet arrived at node₃ at 8.208 seconds. We can see that the true transmission delay dominates as the packet creation time (used to mimic transmission delay from the node₁) which is 7.992, plus the recorded transmission delay, which is 0.016, equals 8.008 seconds. This is much greater than the next closest delay, which is propagation delay, at 0.2 seconds.

3.1.3 Verifying calculations

We are able to calculate the time our file is received at node₃ by calculating the time our last packet, packet₁₀₀₀, is received at node₃. We calculate the time it takes for packet₁₀₀₀ to reach node₂ by adding the creation time, processing delay, propagation delay, transmission delay, and queueing delay for packet₁₀₀₀ going to node₂. We then can calculate the total time it takes to get to node₃ by adding the arrival time to node₂ for packet₁₀₀₀ to the processing delay, propagation delay, transmission delay, and queueing delay for packet₁₀₀₀ going to node₃. In our simulation we are assuming that processing delay is 0. As we are throttling our creation time of each packet, we will have no queueing delay at node₁. Also, because all of the links in our network are the same speed, every time a packet reaches a node the previous node would have just left. Therefore, there is no queuing delay for node₂ either.

Calculating the time packet₁₀₀₀ is received at node₂

$$\begin{aligned}
tDelay_{p1000\text{ at }n1} &= \frac{\text{size of the packet}}{\text{link rates}} \\
&= \frac{1000 \text{ bytes}}{1 \text{ Mbps}} \\
&= \frac{8000 \text{ bits}}{1 \text{ Mbps}} \\
&= 0.008 \text{ seconds} \\
cTime_{p1000} &= (1000 - 1) * tDelay_{n1} \text{ seconds} \\
&= (999) * (0.008) \text{ seconds} \\
&= 7.992 \text{ seconds} \\
procDelay &= 0 \text{ seconds} \\
propDelay_{n1} &= 0.1 \text{ seconds} \\
qDelay_{p1000\text{ at }n1} &= 0 \\
\text{time packet}_{1000} \text{ is received at node}_2 &= cTime_{p1000} + procDelay + propDelay_{n1} + \\
&\quad + tDelay_{p1000,at\ n\ 1} + qDelay_{p1000\text{ at }n\ 1} \\
&= 7.992 + 0 + 0.1 + 0.008 + 0 \text{ seconds} \\
&= 8.1 \text{ seconds}
\end{aligned}$$

Calculating the time packet₁₀₀₀ is received at node₃

$$\begin{aligned} tDelay_{p1000 \text{ at } n2} &= \frac{\text{size of the packet}}{\text{link rates}} \\ &= \frac{1000 \text{ bytes}}{1 \text{ Mbps}} \\ &= \frac{8000 \text{ bits}}{1 \text{ Mbps}} \\ &= 0.008 \text{ seconds} \\ \text{time packet}_{1000} \text{ is received at node}_2 &= 8.1 \text{ seconds} \\ \text{procDelay} &= 0 \text{ seconds} \\ \text{propDelay}_{n2} &= 0.1 \text{ seconds} \\ qDelay_{p1000 \text{ at } n2} &= 0 \\ \text{time packet}_{1000} \text{ is received at node}_3 &= \text{time packet}_{1000} \text{ is received at node}_2 + \\ &\quad + \text{procDelay} + \text{propDelay}_{n2} + tDelay_{p1000 \text{ at } n2} \\ &\quad + qDelay_{p1000 \text{ at } n2} \\ &= 8.1 + 0 + 0.1 + 0.008 + 0 \text{ seconds} \\ &= 8.208 \text{ seconds} \end{aligned}$$

From our calculations we see that packet₁₀₀₀ is received at node₃ at 8.208 seconds, which matches our simulation.

3.2 First simulation of a three node network - Part 2

3.2.1 Simulation configuration

We repeated our first simulation of a three node network, but this time we set the bandwidth between the node₁ and node₂ to 1 Gbps, the bandwidth from node₂ and node₃ to 1 Gbps. We still set the propagation delay between the node₁ and node₂ to 100 milliseconds, and set the propagation delay between the node₂ and node₃ to 100 milliseconds. We send a 1000 packets, each 1000 bytes in size through the network with the source being node₁ and the destination being node₃. To do this we use the following configuration file:

```
1 # n1 — n2 — n3
2 n1 n2
3 n2 n1 n3
4 n3 n2
5
6 # link configuration
7 n1 n2 1Gbps 100ms
8 n2 n1 1Gbps 100ms
9 n2 n3 1Gbps 100ms
10 n3 n2 1Gbps 100ms
```

This configuration file above is called 'lab1-twohopA2.txt.' This is to be inserted into our Network constructor referred to above. Once the network is set up, we will send a single file in 1000 packets to node three and then run the simulation. In order to avoid queueing issues (like overflowing the simulated queue), we will control the time each packet is sent so that each packet will be begin its process as its previous packet leaves node₁. This is shown in the following code snippet:

```
1 #8kbit/1Gbps = 8kbit/1000000kbps
2 trasmissionDelay = 8.0/1000000.0
3 # send packets
4 for i in range(1,1001):
```

```

5     calculatedDelay = (i-1) * trasmissionDelay
6     p = Packet(destination_address=n3.get_address('n2'), ident=i, protocol='delay',
7         length=1000)
8     Sim.scheduler.add(delay=calculatedDelay, event=p, handler=n1.send_packet)
9 # run the simulation
10 Sim.scheduler.run()

```

3.2.2 Output of the simulation

Last 5 entries:

```

1 (0.207976000000000013, 996, 0.00796, 0.200016000000000014, 1.6e-05, 0.2,
2     1.3010426069826053e-16)
3 (0.20798400000000001, 997, 0.007968, 0.20001600000000001, 1.6e-05, 0.2,
4     1.3010426069826053e-16)
5 (0.207992000000000012, 998, 0.007976, 0.20001600000000001, 1.6e-05, 0.2,
6     1.2836953722228372e-16)
7 (0.208000000000000013, 999, 0.007984, 0.200016000000000014, 1.6e-05, 0.2,
8     1.2836953722228372e-16)
9 (0.208008000000000014, 1000, 0.007991999999999999, 0.200016000000000014, 1.6e-05, 0.2,
10    1.2836953722228372e-16)

```

The entire file is transferred when the last packet arrives at node₃. This output shows that the last packet arrived at node₃ at about 0.208008 seconds. We can see that the true transmission delay is the packet creation time (used to mimic transmission delay from the node₁) which is about 0.007992 seconds, plus the recorded transmission delay, which is 0.000016, equals 0.008008 seconds. This time we can see that propagation delay is much higher and is the dominate delay at 0.2 seconds.

3.2.3 Verifying calculations

We are able to calculate the time our file is received at node₃ by calculating the time our last packet, packet₁₀₀₀, is received at node 3. We calculate the time it takes for packet₁₀₀₀ to reach node₂ by adding the creation time, processing delay, propagation delay, transmission delay, and queueing delay for packet₁₀₀₀ going to node₂. We then can calculate the total time it takes to get to node₃ by adding the arrival time to node₂ for packet₁₀₀₀ to the processing delay, propagation delay, transmission delay, and queueing delay for packet₁₀₀₀ going to node₃. In our simulation we are assuming that processing delay is 0. As we are throttling our creation time of each packet, we will have no queueing delay at node₁. Also, because all of the links in our network are the same speed, every time a packet reaches a node the previous node would have just left. Therefore, there is no queueing delay for node₂ either.

Calculating the time packet₁₀₀₀ is received at node₂

$$\begin{aligned}
tDelay_{p1000 \text{ at } n1} &= \frac{\text{size of the packet}}{\text{link rates}} \\
&= \frac{1000 \text{ bytes}}{1 \text{ Gbps}} \\
&= \frac{8000 \text{ bits}}{1 \text{ Gbps}} \\
&= 0.000008 \text{ seconds} \\
cTime_{p1000} &= (1000 - 1) * tDelay_{n1} \text{ seconds} \\
&= (999) * (0.000008) \text{ seconds} \\
&= 0.007992 \text{ seconds} \\
procDelay &= 0 \text{ seconds} \\
propDelay_{n1} &= 0.1 \text{ seconds} \\
qDelay_{p1000 \text{ at } n1} &= 0 \\
\text{time packet}_{1000} \text{ is received at node}_2 &= cTime_{p1000} + procDelay + propDelay_{n1} + \\
&\quad + tDelay_{p1000 \text{ at } n1} + qDelay_{p1000 \text{ at } n1} \\
&= 0.007992 + 0 + 0.1 + 0.000008 + 0 \text{ seconds} \\
&= 0.108 \text{ seconds}
\end{aligned}$$

Calculating the time packet₁₀₀₀ is received at node₃

$$\begin{aligned}
tDelay_{p1000 \text{ at } n2} &= \frac{\text{size of the packet}}{\text{link rates}} \\
&= \frac{1000 \text{ bytes}}{1 \text{ Mbps}} \\
&= \frac{8000 \text{ bits}}{1 \text{ Mbps}} \\
&= 0.000008 \text{ seconds} \\
\text{time packet}_{1000} \text{ is received at node}_2 &= 0.108 \text{ seconds} \\
procDelay &= 0 \text{ seconds} \\
propDelay_{n2} &= 0.1 \text{ seconds} \\
qDelay_{p1000 \text{ at } n2} &= 0 \\
\text{time packet}_{1000} \text{ is received at node}_3 &= \text{time packet}_{1000} \text{ is received at node}_2 + \\
&\quad + procDelay + propDelay_{n2} + tDelay_{p1000 \text{ at } n2} \\
&\quad + qDelay_{p1000 \text{ at } n2} \\
&= 0.108 + 0 + 0.1 + 0.000008 + 0 \text{ seconds} \\
&= 0.208008 \text{ seconds}
\end{aligned}$$

From our calculations we see that packet₁₀₀₀ is received at node₃ at 0.208008 seconds, which matches our simulation.

3.3 Second simulation of a three node network

3.3.1 Simulation configuration

For our second simulation of a three node network, we set the bandwidth between the node₁ and node₂ to 1 Mbps, the bandwidth from node₂ and node₃ to 256 Kbps, set the propagation delay between the

node₁ and node₂ to 100 milliseconds, and set the propagation delay between the node₂ and node₃ to 100 milliseconds. We send a 1000 packets, each 1000 bytes in size through the network with the source being node₁ and the destination being node₃. To do this we use the following configuration file:

```

1 # n1 — n2 — n3
2 n1 n2
3 n2 n1 n3
4 n3 n2
5
6 # link configuration
7 n1 n2 1Mbps 100ms
8 n2 n1 1Mbps 100ms
9 n2 n3 256Kbps 100ms
10 n3 n2 256Kbps 100ms

```

This configuration file above is called 'lab1-twohopB.txt.' This is to be inserted into our Network constructor referred to above. Once the network is set up, we will send a single file in 1000 packets to node three and then run the simulation. In order to avoid queueing issues (like overflowing the simulated queue), we will control the time each packet is sent so that each packet will be begin its process as its previous packet leaves node₁. This is shown in the following code snippet:

```

1 trasmissionDelay = 8.0/1000.0
2 # send packets
3 for i in range(1,1001):
4     calculatedDelay = (i-1) * trasmissionDelay
5     p = Packet(destination_address=n3.get_address('n2'), ident=i, protocol='delay', length=1000)
6     Sim.scheduler.add(delay=calculatedDelay, event=p, handler=n1.send_packet)
7 # run the simulation
8 Sim.scheduler.run()

```

3.3.2 Output of the simulation

Last 5 entries:

```

1 (31.3330000000000002, 996, 7.96, 23.373, 0.03925, 0.2, 23.13375)
2 (31.3642500000000002, 997, 7.968, 23.3962500000000002, 0.03925, 0.2, 23.1570000000000004)
3 (31.3955000000000002, 998, 7.976, 23.4195000000000003, 0.03925, 0.2, 23.18025)
4 (31.4267500000000002, 999, 7.984, 23.4427500000000004, 0.03925, 0.2, 23.2035000000000002)
5 (31.4580000000000002, 1000, 7.992, 23.466, 0.03925, 0.2, 23.2267500000000003)

```

The entire file is transfered when the last packet arrives at node₃. This output shows that the last packet arrived at node₃ at about 31.458 seconds.

3.3.3 Verifying calculations

We are able to calculate the time our file is received at node₃ by calculating the time our last packet, packet₁₀₀₀, is received at node 3. We calculate the time it takes for packet₁₀₀₀ to reach node₂ by adding the creation time, processing delay, propagation delay, transmission delay, and queueing delay for packet₁₀₀₀ going to node₂. We then can calculate the total time it takes to get to node₃ by adding the arrival time to node₂ for packet₁₀₀₀ to the processing delay, propagation delay, transmission delay, and queueing delay for packet₁₀₀₀ going to node₃. In our simulation we are assuming that processing delay is 0. As we are throttling our creation time of each packet, we will have no queueing delay at node₁. There will be queueing delay at node₂ because the bandwidth from node₂ to node₃ is less than the bandwidth from node₁ to node₂. Because there are no gaps in the streaming of our 1000 packets, we are able to use a simpler formula to calculate queueing delay. The formula will be shown below is as follows:

$$qDelay_{pk\ at\ n2} = (k - 1)(tDelay_{pk\ at\ n2} - tDelay_{pk\ at\ n1})$$

Calculating the time packet₁₀₀₀ is received at node₂

$$\begin{aligned} tDelay_{p1000\ at\ n1} &= \frac{\text{size of the packet}}{\text{link rates}} \\ &= \frac{1000\ bytes}{1\ Mbps} \\ &= \frac{8000\ bits}{1\ Mbps} \\ &= 0.008\ seconds \end{aligned}$$

$$\begin{aligned} cTime_{p1000} &= (1000 - 1) * tDelay_{n1}\ seconds \\ &= (999) * (0.008)\ seconds \\ &= 7.992\ seconds \end{aligned}$$

$$procDelay = 0\ seconds$$

$$propDelay_{n1} = 0.1\ seconds$$

$$qDelay_{p1000\ at\ n1} = 0$$

$$\begin{aligned} \text{time packet}_{1000}\ is\ received\ at\ node_2 &= cTime_{p1000} + procDelay + propDelay_{n1} + \\ &\quad + tDelay_{p1000\ at\ n1} + qDelay_{p1000\ at\ n1} \\ &= 7.992 + 0 + 0.1 + 0.008 + 0\ seconds \\ &= 8.1\ seconds \end{aligned}$$

Calculating the time packet₁₀₀₀ is received at node₃

$$\begin{aligned}
 tDelay_{p1000 \text{ at } n2} &= \frac{\text{size of the packet}}{\text{link rates}} \\
 &= \frac{1000 \text{ bytes}}{256 \text{ Kbps}} \\
 &= \frac{8000 \text{ bits}}{256 \text{ Kbps}} \\
 &= 0.03125 \text{ seconds}
 \end{aligned}$$

$$\text{time packet}_{1000} \text{ is received at node}_2 = 8.1 \text{ seconds}$$

$$\text{procDelay} = 0 \text{ seconds}$$

$$\text{propDelay}_{n2} = 0.1 \text{ seconds}$$

$$\begin{aligned}
 qDelay_{p1000 \text{ at } n2} &= (k - 1)(tDelay_{p1000 \text{ at } n2} - tDelay_{p1000 \text{ at } n1}) \\
 &= (1000 - 1)(0.03125 \text{ seconds} - 0.008 \text{ seconds}) \\
 &= 23.22675
 \end{aligned}$$

$$\begin{aligned}
 \text{time packet}_{1000} \text{ is received at node}_3 &= \text{time packet}_{1000} \text{ is received at node}_2 + \\
 &\quad + \text{procDelay} + \text{propDelay}_{n2} + tDelay_{p1000 \text{ at } n2} \\
 &\quad + qDelay_{p1000 \text{ at } n2} \\
 &= 8.1 + 0 + 0.1 + 0.03125 + 23.22675 \text{ seconds} \\
 &= 31.458 \text{ seconds}
 \end{aligned}$$

From our calculations we see that packet₁₀₀₀ is received at node₃ at 31.458 seconds, which matches our simulation.

4 Queueing Theory

In this section, we explore how well our simulation matches up to queueing theory. We will simulate a M/D/1 queue. The "M" means that there is an exponential distribution of arrival times. The "D" means there is a deterministic service time. The "1" means that there is only 1 queue. It is expected that as our utilization percentage of our maximum rate approaches 100%, the average queueing delay will go to infinity.

Our simulator streams packets in between two nodes for 10 seconds. The rate that the packets are created and sent is determined by passing a utilization percentage of the maximum rate into a random exponential distribution function. We chose a number of different utilization percentages. We ran these utilization percentages through our experiment, and then graphed the average queueing delay for each utilization percentage.

We also graphed the theoretical values for each of the utilization percentages using the following formula:

$$w = \frac{1}{2u} * \frac{p}{1 - p}$$

where u = service rate

and where p = utilization percentage

4.1 Simulation configuration

To setup the simulation, we first reset the scheduler for the simulation. Next, we set up the network by create a Network object by passing a network configuration file (which will be described below). After that,

we set up the route information between the nodes. Finally, we setup a handler to print out information about packets received at node₂ to a file. The following code snippet performs the setup:

```

1  # parameters
2  Sim.scheduler.reset()
3
4  # setup network
5  net = Network('../networks/one-hop.txt')
6
7  # setup routes
8  n1 = net.get_node('n1')
9  n2 = net.get_node('n2')
10 n1.add_forwarding_entry(address=n2.get_address('n1'), link=n1.links[0])
11 n2.add_forwarding_entry(address=n1.get_address('n2'), link=n2.links[0])
12
13 # setup app
14 d = DelayHandler()
15 net.nodes['n2'].add_protocol(protocol="delay", handler=d)

```

Our simulated network had two nodes. We set the bandwidth between the nodes to 1 Mbps, and set the propagation delay between the nodes to 1 second. To do this we use the following configuration file:

```

1 # n1 --- n2
2 n1 n2
3 n2 n1
4
5 # link configuration
6 n1 n2 1Mbps 1000ms
7 n2 n1 1Mbps 1000ms

```

This configuration file above is called 'lab1-queueingtheory.txt.' This is to be inserted into our Network constructor referred to above. Once the network is set up, we then need to set up a way to continually send packets as we need them. This can be done with a Generator class. This class generates packets based on the load parameter passed in and the random.expovariate function. This shown in the following code snippet:

```

1 class Generator(object):
2     def __init__(self, node, destination, load, duration):
3         self.node = node
4         self.load = load
5         self.destination = destination
6         self.duration = duration
7         self.start = 0
8         self.ident = 1
9
10    def handle(self, event):
11        # quit if done
12        now = Sim.scheduler.current_time()
13        if (now - self.start) > self.duration:
14            return
15
16        # generate a packet
17        self.ident += 1
18        p = Packet(destination_address=self.destination, ident=self.ident,
19                  protocol='delay', length=1000)
20        Sim.scheduler.add(delay=0, event=p, handler=self.node.send_packet)
21        # schedule the next time we should generate a packet
22        Sim.scheduler.add(delay=random.expovariate(self.load), event='generate',

```

Lastly, we need to generate our load parameter and start our simulator. First, we calculate the maximum rate our link can handle (in packets/second), which is 1Mbps/8Kbit packet size. Then we multiply this value by the utilization percentage chosen for the experiment. After we have this value, we create our Generator object by passing in the source node, destination node, the load value, and 10 seconds of duration time. Lastly, we start the simulation. This is shown in the following code snippet:

```

1 # setup packet generator
2 destination = n2.get_address('n1')
3 max_rate = 1000000 // (1000 * 8)
4 myLoadPercent = 98
5
6 load = myLoadPercent/100.0 * max_rate
7 g = Generator(node=n1, destination=destination, load=load, duration=10)
8 Sim.scheduler.add(delay=0, event='generate', handler=g.handle)
9
10 # run the simulation
11 Sim.scheduler.run()
```

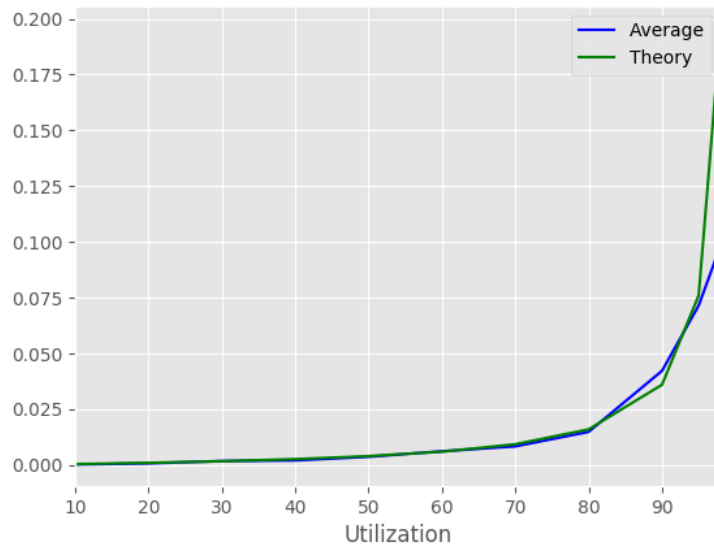
4.2 Experiment

In our experiment we used the following utilization values: 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 95%, and 98%.

For each utilization value, we recorded the queueing time for each packet sent. The number of packets sent varied from about 100 to 1000 depending on how high the utilization was. We then calculated the average queueing delay for each utilization value. The averages were created by running the data through a python script. The averages were as following:

10% = 0.000246186799194 *seconds*
 20% = 0.000744426315041 *seconds*
 30% = 0.00180887562312 *seconds*
 40% = 0.00207776042834 *seconds*
 50% = 0.00370991960505 *seconds*
 60% = 0.00617988242069 *seconds*
 70% = 0.0083786921238 *seconds*
 80% = 0.0148861562441 *seconds*
 90% = 0.0422212596784 *seconds*
 95% = 0.0714921688987 *seconds*
 98% = 0.0985421009293 *seconds*

In the graph below, we graphed our experimental data in blue and the theoretical data in green. The theoretical data was created by using the function described above.



As you can see from our data, our experimental data closely matches the theoretical data. The only variation is when utilization reaches 98%.