Network Simulation

Chase Robertson

20 October 2017

My test protocol is called transfer.py in the bene/lab2 directory. It can be run with args: -f filename (no default), -l loss (default 0.0), -w window size (default 3000 bytes), and flag -r to enable fast retransmit (default disabled). Two network configuration files were interchanged in transfer.py to model tcp with and without queueing: network.txt and networkWithQueue.txt.

1 Basic Tests

A simple network consisting of two nodes and and one bidirectional link was simulated. The bandwidth of the links was set to 10 Mbps, with a propagation delay of 10 milliseconds. Maximum packet size was set to 1,000 bytes.

1. A test file "test.txt" of 10,000 bytes was sent from n1 to n2 at time 0, with a window size of 3,000 bytes. Different loss rates were tested in each scenario.

   (a) Test of 0% packet loss. This test went as fast as the transmission rate and window size would allow, with no loss to increase delay.

```
0.0362 n1 (1) received ACK from 2 for 8000
0.037 n1 (1) received ACK from 2 for 9000
0.0462 n2 (2) received TCP segment from 1 for 9000
0.0462 application got 1000 bytes
0.0462 n2 (2) sending TCP ACK to 1 for 10000
0.0472 n1 (1) received ACK from 2 for 10000

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

   (b) Test of 10% packet loss. Though this test had only 1 packet lost per 10 sent, the timeout of 1 second ended up increasing total delay by 20 times!

```
1.059 n1 (1) received ACK from 2 for 9000
1.059 n1 (1) sending TCP segment to 2 for 9000
1.0698 n2 (2) received TCP segment from 1 for 9000
1.0698 application got 1000 bytes
1.0698 n2 (2) sending TCP ACK to 1 for 10000
1.0708 n1 (1) received ACK from 2 for 10000

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

(c) Test of 20% packet loss. Each packet lost added 1 second of delay waiting for the timeout to signal loss.

```
2.0706 n2 (2) sending TCP ACK to 1 for 9000
2.0708 n1 (1) received ACK from 2 for 8000
2.0714 n2 (2) received TCP segment from 1 for 9000
2.0714 application got 1000 bytes
2.0714 n2 (2) sending TCP ACK to 1 for 10000
2.0716 n1 (1) received ACK from 2 for 9000
2.0724 n1 (1) received ACK from 2 for 10000

File transfer correct!
byu718179wks:lab2 chaserobertson$
```

(d) Test of 50% packet loss. The nature of such heavy packet loss increased the total time of this test far beyond my expectations. Only 10 packets of data ended up causing far more than 5 timeout delays, because each packet lost must be resent, and each resending has its own chance of being lost.

```
16.0708 n1 (1) retransmission timer fired
16.0708 n1 (1) sending TCP segment to 2 for 8000
16.0816 n2 (2) received TCP segment from 1 for 8000
16.0816 application got 0 bytes
16.0816 n2 (2) sending TCP ACK to 1 for 10000
16.0826 n1 (1) received ACK from 2 for 10000

File transfer correct!
byu718179wks:lab2 chaserobertson$
```

2. A test file "internet-architecture.pdf" of 514,520 bytes was sent from n1 to n2 at time 0, with a window size of 10,000 bytes. Different loss rates were tested in each scenario.

(a) Test of 0% packet loss. As expected, total delay of transfer was only limited by transmission speed.

```
0.6152 n1 (1) received ACK from 2 for 513000
0.615416 n2 (2) received TCP segment from 1 for 514000
0.615416 application got 520 bytes
0.615416 n2 (2) sending TCP ACK to 1 for 514520
0.616 n1 (1) received ACK from 2 for 514000
0.616416 n1 (1) received ACK from 2 for 514520

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

(b) Test of 50% packet loss. Of the 515 packets of new data sent, 402 were lost. This is because each packet lost must be resent, as well as packets sent after the lost packet, each having its own 50% chance of also being lost.

```
402.749416 application got 0 bytes
402.749416 n2 (2) sending TCP ACK to 1 for 513000
402.750216 n2 (2) received TCP segment from 1 for 513000
402.750216 application got 1520 bytes
402.750216 n2 (2) sending TCP ACK to 1 for 514520
402.750416 n1 (1) received ACK from 2 for 513000
402.751216 n1 (1) received ACK from 2 for 514520

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

2 Fast Retransmit

The same simple network consisting of two nodes and and one bidirectional link was simulated. The bandwidth of the links was set to 10 Mbps, with a propagation delay of 10 milliseconds. Maximum packet size was set to 1,000 bytes.

1. A test file "internet-architecture.pdf" of 514,520 bytes was sent from n1 to n2 at time 0, with a window size of 10,000 bytes. The same loss rate of 20% was tested with and without fast retransmit functionality. My implementation of fast retransmit depended on receiving 4 ACK responses with the same sequence number, then immediately resending the next outstanding packet as though a timeout had occurred.

   (a) Test without fast retransmit. Significant delay occurred because it takes a full second to recognize each lost packet.

```
67.8292 n2 (2) sending TCP ACK to 1 for 514000
67.8294 n1 (1) received ACK from 2 for 513000
67.829616 n2 (2) received TCP segment from 1 for 514000
67.829616 application got 520 bytes
67.829616 n2 (2) sending TCP ACK to 1 for 514520
67.830616 n1 (1) received ACK from 2 for 514520

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

   (b) Test with fast retransmit. Delay is reduced by a factor of 12, simply by recognizing duplicate cumulative ACK's as a sign of packet loss, thereby short circuiting timeout in most cases.

```
4.103664 application got 0 bytes
4.103664 n2 (2) sending TCP ACK to 1 for 514520
4.103864 n1 (1) received ACK from 2 for 514520
4.10408 n2 (2) received TCP segment from 1 for 514000
4.10408 application got 0 bytes
4.10408 n2 (2) sending TCP ACK to 1 for 514520
4.10508 n1 (1) received ACK from 2 for 514520

File transfer correct!
byu718179wks:lab2 chaserobertson$
```

3 Experiments

The same simple network consisting of two nodes and and one bidirectional link was simulated. The bandwidth of the links was set to 10 Mbps, with a propagation delay of 10 milliseconds. Maximum packet size was set to 1,000 bytes. A queue size of 100 was added, and loss rate was set to 0%.

The throughput of each test transfer was computed as the total bits sent divided by the total time to send the file. The average queueing delay of each test transfer was also calculated (from simulator's queueing log csv file).

1. A test file "internet-architecture.pdf" of 514,520 bytes was sent from n1 to n2 at time 0, using varying window sizes.

   (a) Test with 1,000 byte window size

```
6.0642 application got 1000 bytes
6.0642 n2 (2) sending TCP ACK to 1 for 514000
6.0652 n1 (1) received ACK from 2 for 514000
6.0652 n1 (1) sending TCP segment to 2 for 514000
6.075616 n2 (2) received TCP segment from 1 for 514000
6.075616 application got 520 bytes
6.075616 n2 (2) sending TCP ACK to 1 for 514520
6.076616 n1 (1) received ACK from 2 for 514520

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

$$throughput = 4,116,160bits/6.0766sec = 677Kbps$$
$$d_{queue} = 0sec$$

   (b) Test with 2,000 byte window size

```
3.0324 application got 1000 bytes
3.0324 n2 (2) sending TCP ACK to 1 for 514000
3.0326 n1 (1) received ACK from 2 for 513000
3.0326 n1 (1) sending TCP segment to 2 for 514000
3.0334 n1 (1) received ACK from 2 for 514000
3.043016 n2 (2) received TCP segment from 1 for 514000
3.043016 application got 520 bytes
3.043016 n2 (2) sending TCP ACK to 1 for 514520
3.044016 n1 (1) received ACK from 2 for 514520

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

$$throughput = 4,116,160bits/3.044sec = 1.35Mbps$$
$$d_{queue} = 0.0016sec$$

(c) Test with 5,000 byte window size

```
1.2168 application got 1000 bytes
1.2168 n2 (2) sending TCP ACK to 1 for 514000
1.217 n1 (1) received ACK from 2 for 513000
1.217216 n2 (2) received TCP segment from 1 for 514000
1.217216 application got 520 bytes
1.217216 n2 (2) sending TCP ACK to 1 for 514520
1.2178 n1 (1) received ACK from 2 for 514000
1.218216 n1 (1) received ACK from 2 for 514520

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

$$throughput = 4,116,160/1.218 = 3.38Mbps$$
$$d_{queue} = 0.0136sec$$

(d) Test with 10,000 byte window size

```
0.615 application got 1000 bytes
0.615 n2 (2) sending TCP ACK to 1 for 514000
0.6152 n1 (1) received ACK from 2 for 513000
0.615416 n2 (2) received TCP segment from 1 for 514000
0.615416 application got 520 bytes
0.615416 n2 (2) sending TCP ACK to 1 for 514520
0.616 n1 (1) received ACK from 2 for 514000
0.616416 n1 (1) received ACK from 2 for 514520

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

$$throughput = 4,116,160/0.6164 = 6.68Mbps$$
$$d_{queue} = 0.032sec$$

(e) Test with 15,000 byte window size

```
0.4212 application got 1000 bytes
0.4212 n2 (2) sending TCP ACK to 1 for 514000
0.4214 n1 (1) received ACK from 2 for 513000
0.421616 n2 (2) received TCP segment from 1 for 514000
0.421616 application got 520 bytes
0.421616 n2 (2) sending TCP ACK to 1 for 514520
0.4222 n1 (1) received ACK from 2 for 514000
0.422616 n1 (1) received ACK from 2 for 514520

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

$$throughput = 4,116,160/0.4226 = 9.74Mbps$$
$$d_{queue} = 0.412sec$$

(f) Test with 20,000 byte window size

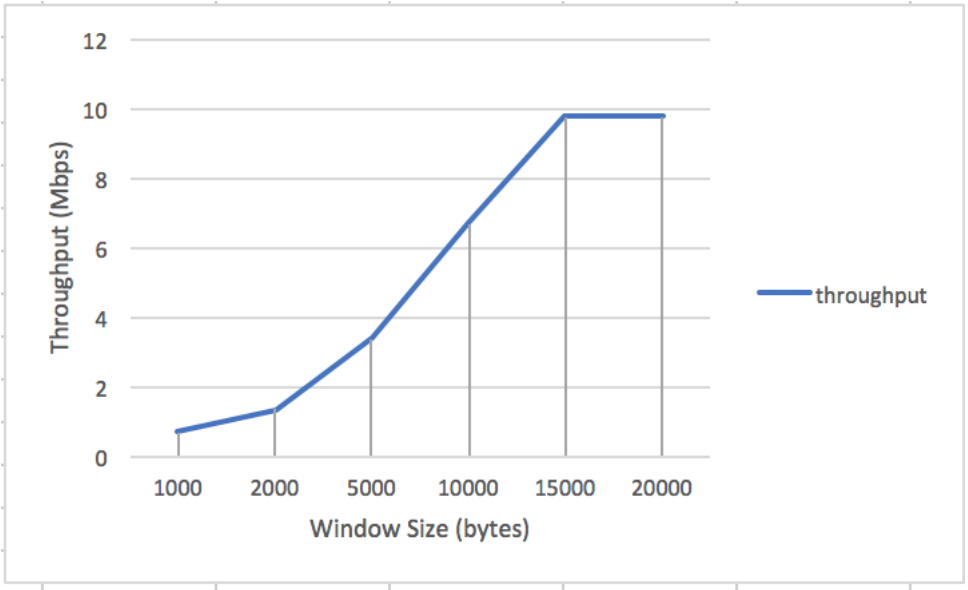
```
0.4212 application got 1000 bytes
0.4212 n2 (2) sending TCP ACK to 1 for 514000
0.4214 n1 (1) received ACK from 2 for 513000
0.421616 n2 (2) received TCP segment from 1 for 514000
0.421616 application got 520 bytes
0.421616 n2 (2) sending TCP ACK to 1 for 514520
0.4222 n1 (1) received ACK from 2 for 514000
0.422616 n1 (1) received ACK from 2 for 514520

File transfer correct!
Chases-MBP-010024223033:lab2 chaserobertson$
```

$$throughput = 4,116,160/0.4226 = 9.74Mbps$$
$$d_{queue} = 0.420sec$$

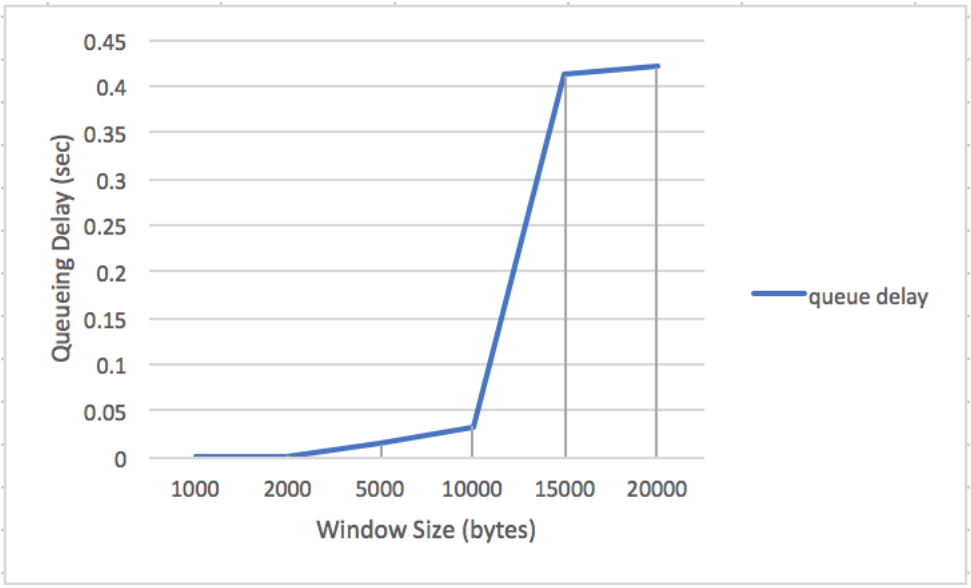
2. The throughput and average queueing delay of each test transfer were each plotted as a function of window size.

(a) Throughput vs Window Size



As window size increases, link utilization also increases because more packets are in transfer at once. Once maximum link utilization is reached, increasing window size doesn't decrease total delay anymore. This can be seen in the difference between total delay time between window sizes 15,000 and 20,000 bytes: there is no difference!

(b) Queueing Delay vs Window Size



As window size increases, the queue starts to fill with packets and cause some queueing delay. Much more interesting experiments can be conducted with larger window sizes, as the queue fills much more than in these cases. Experiments that include random loss and random packet arrival or readiness would also be more akin to reality.