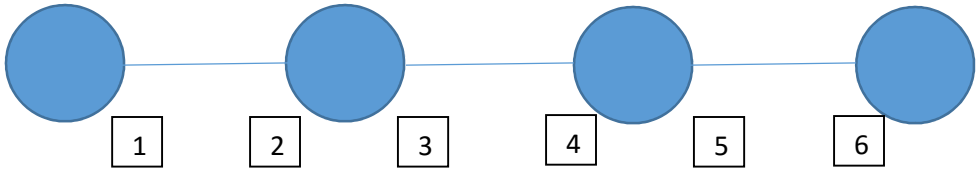


# Setup:

| Interface | IP Address |
|---|---|
| 1 | 10.1.1.1 |
| 2 | 10.1.1.2 |
| 3 | 10.2.1.1 |
| 4 | 10.2.1.2 |
| 5 | 10.3.1.1 |
| 6 | 10.3.1.2 |

There are three networks three different IP prefixes. Each of two routers have two interfaces. Source and sink have one interface each.

# Procedure:

A source and a sink are connected with two routers in between. Interface 3 is the one, where we analyze the queue because the link from interface 3 to 4 is the bottleneck. It has the lowest capacity of all the links. Different sets of experiments are performed by changing queue size, TCP segment size, maximum receiver advertised window and using either TCP Tahoe or Reno. TCP Reno has Fast Recovery mechanism in addition to all the features of Tahoe version. In another set of experiments, 10 flows are created between source and sink nodes and all the nodes start at time uniformly distributed between 0 and 0.1 seconds. A total of 192 experiments were performed by varying these parameters. TCP flow(s) at source send total of 100,000,000 bytes. Each simulation lasts for 10 seconds. *BulkSendHelper()* at source and *PacketSinkHelper()* at sink are used for sending and receiving of data. Goodput in Bytes per second is calculated at sink by dividing the total bytes received at sink divided by the difference in time between receiving the first byte and the last one. *FlowMonitorHelper* was used for this purpose. Goodput does not include packet retransmissions and is therefore easy to calculate at the sink by noting the number of received bytes.

Most of the values obtained for goodput are same for TCP Tahoe and Reno. But TCP Tahoe has better throughput in a few cases when window size is greater than queue size. TCP Reno includes Fast Recovery mechanism where slow start is not started after receiving triple duplicate acknowledgements. When there are not multiple packet losses in a round trip time, TCP Reno does not provide improvement.

## Single Flow:

### Increasing window size for fixed segment size and queue size

TCP Tahoe:

```
flow,0,windowSize,8000,queueSize,2000,segSize,128,goodput,29895
flow,0,windowSize,32000,queueSize,2000,segSize,128,goodput,29895
flow,0,windowSize,64000,queueSize,2000,segSize,128,goodput,29895
```

When queue size is less than window size, there is no improvement in goodput when we increase the window size. No matter how much we increase the maximum advertised window, goodput would be limited by the bottle neck router queue length.

```
flow,0,windowSize,2000,queueSize,32000,segSize,128,goodput,20668.8
flow,0,windowSize,8000,queueSize,32000,segSize,128,goodput,81038.6
```

When queue size is greater than window size, increasing window size results in better goodput. Sender is allowed to send more data, resulting in better goodput. The more data receiver allows sender to send, better would be throughput. Since queue size is larger than advertised window, packets will not be dropped as a result of overflow of queue.

TCP Reno

```
flow,0,windowSize,8000,queueSize,2000,segSize,128,goodput,27189
flow,0,windowSize,32000,queueSize,2000,segSize,128,goodput,27343.6
flow,0,windowSize,64000,queueSize,2000,segSize,128,goodput,27343.6
```

Similarly in TCP Reno, when queue size is less than window size, there is no improvement in goodput when we increase the window size. No matter how much we increase the maximum advertised window, goodput would be limited by the bottle neck router queue.

```
flow,0,windowSize,2000,queueSize,32000,segSize,128,goodput,20668.8
flow,0,windowSize,8000,queueSize,32000,segSize,128,goodput,81038.6
```

Similarly in TCP Reno, when queue size is greater than window size, increasing window size results in better goodput. The more data receiver allows sender to send, better would be throughput. Since queue size is larger than advertised window, packets will not be dropped as a result of overflow of queue.

### Increasing segment size for fixed window size and queue size

TCP Tahoe

```
flow,0,windowSize,8000,queueSize,32000,segSize,128,goodput,81038.6
flow,0,windowSize,8000,queueSize,32000,segSize,256,goodput,85303.7
flow,0,windowSize,8000,queueSize,32000,segSize,512,goodput,78242.3
```

```
flow,0,windowSize,2000,queueSize,8000,segSize,128,goodput,20668.8
flow,0,windowSize,2000,queueSize,8000,segSize,256,goodput,17730.8
flow,0,windowSize,2000,queueSize,8000,segSize,512,goodput,11648.7
```

For window size less than queue size, there are two cases. When window size is 2000, there is decrease in goodput as segment size is increased. For window sizes 8000 and 32000, increasing segment size first results in increase and then decrease in goodput, giving maximum value at 256. Therefore, increasing segment size may or may not result in improvement in goodput because queues may fill up, resulting in increase in delay and decrease in goodput.

```
flow,0,windowSize,8000,queueSize,2000,segSize,128,goodput,29895
flow,0,windowSize,8000,queueSize,2000,segSize,256,goodput,71516
flow,0,windowSize,8000,queueSize,2000,segSize,512,goodput,70923.2

flow,0,windowSize,32000,queueSize,8000,segSize,128,goodput,73332.9
flow,0,windowSize,32000,queueSize,8000,segSize,256,goodput,81953.4
flow,0,windowSize,32000,queueSize,8000,segSize,512,goodput,88396.3
```

When window size is greater than queue size, there are also two cases. When window size is 8000, increasing segment size first results in increase in goodput and then decrease, giving maximum value at 256. When window size is either 32000 or 64000, increasing segment size results in increase in goodput.

Interesting cases are when window size is equal to queue size. For 2000 case, increasing segment size results in degradation in goodput giving best value at 128. For 8000 case, increasing segment size results in first increase and then decrease in goodput, giving best value at 256. For 32000 and 64000 cases, increasing segment size results in increase in goodput giving maximum value at 512.

In general, when window size is equal to 2000, best value of goodput is at segment size 128. When window size is 8000, best value is at 256 and when window size is 32000 or 64000, best value of goodput is at 512 segment size.

Results for TCP Reno are similar.

**Increasing queue size for fixed segment size and receiver window**

For both TCP Reno and Tahoe

```
flow,0,windowSize,8000,queueSize,8000,segSize,128,goodput,81038.6
flow,0,windowSize,8000,queueSize,32000,segSize,128,goodput,81038.6
flow,0,windowSize,8000,queueSize,64000,segSize,128,goodput,81038.6
```

When queue size is larger than maximum receiver advertised window, there is no effect on goodput of changing queue size for both TCP Tahoe and Reno. Goodput is limited by maximum receiver advertised window, which is less than queue size of bottle neck link. No matter how large queue we place, it will never be filled up to result in decrease in goodput.

```
flow,0,windowSize,32000,queueSize,2000,segSize,256,goodput,55666.5
flow,0,windowSize,32000,queueSize,8000,segSize,256,goodput,81953.4
```

```
flow,0,windowSize,32000,queueSize,32000,segSize,256,goodput,96050.5
```

When queue size is less than window size, increasing queue size results in increase in throughput until we reach queue size equal to window size. Since maximum advertised window is allowing sender to send data, small queues will result in drop and degradation in throughput.

## Multiple Flows

For both Tahoe and Reno, the results are most fair (not much different from each other for the individual goodput of flows) for window size = 2000, queue size=32000 and for window size = 2000 and queue size = 64000 for any value of segment size. Also, when window size = 2000 and queue size = 8000, the results are quite fair for segment size = 512 as well. The values of goodput are the same for equivalent cases of TCP Tahoe and Reno. When queue is large and maximum advertised window is small, no flow will be allowed to put huge amount of data due to relatively small advertised window. Also large queue evens out the usage among flows. All the flows get almost equal share of bandwidth. Larger queues and smaller advertised windows result in greater fairness of goodput among flows.

```
flow,0,windowSize,2000,queueSize,32000,segSize,512,goodput,11133.3
flow,1,windowSize,2000,queueSize,32000,segSize,512,goodput,11162.8
flow,2,windowSize,2000,queueSize,32000,segSize,512,goodput,11169.6
flow,3,windowSize,2000,queueSize,32000,segSize,512,goodput,11179.5
flow,4,windowSize,2000,queueSize,32000,segSize,512,goodput,11119.7
flow,5,windowSize,2000,queueSize,32000,segSize,512,goodput,11123.3
flow,6,windowSize,2000,queueSize,32000,segSize,512,goodput,11122.8
flow,7,windowSize,2000,queueSize,32000,segSize,512,goodput,11190.9
flow,8,windowSize,2000,queueSize,32000,segSize,512,goodput,11125.6
flow,9,windowSize,2000,queueSize,32000,segSize,512,goodput,11191.3
```

Fairness can be calculated by using Raj Jain's fairness index:

$$F(x_0, x_1, \dots, x_9) = \frac{(\sum_{i=0}^{9} x_i)^2}{10.(\sum_{i=0}^{9} x_i^2)}$$
$$where, x_i \; is \; the \; goodputh \; of \; i^{th} \; flow$$

The value of index varies between 0 and 1 for large number of sources with 0 being most unfair and 1 being most fair allocation.