Project 1 – TCP Throughput Measurements

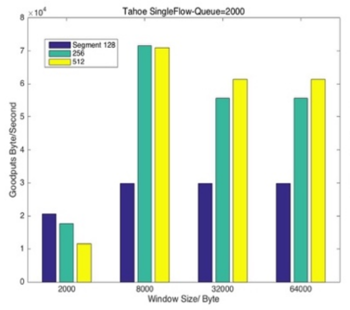
Zhongyi Luo
903141808

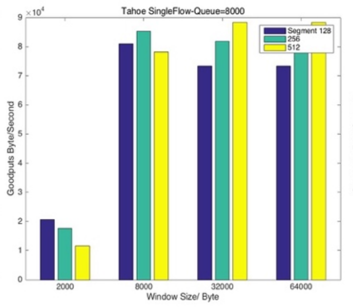
Simulator version: ns-3.24.1 Platform: Mac OSX 10.10.5, deepthought
Parameter:
Flow number {1,10}; Window size {2000,8000,32000,64000}; Queue Size {2000,8000,32000,64000}; Segment Size {128,256,512}; TCP-Tahoe/TCP-Reno.
Due to page limit, please refer to .cc file for topology and application settings. Due to multiple variables, analysis is done by fixing some of them in turns.

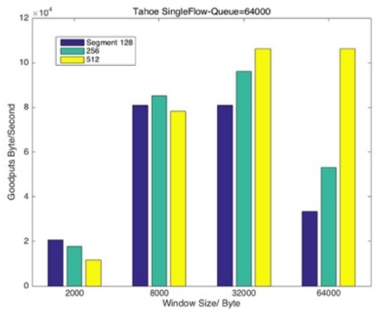
1. One Flow result

1.1 Analysis as Window Size is constant

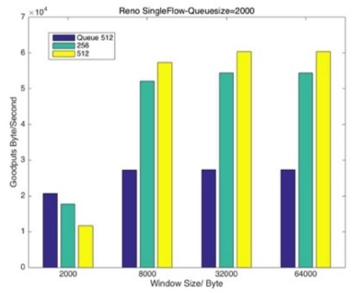
In the case of **TCP-Tahoe**, in which queue size is fixed at 2000, 8000 and 6400:

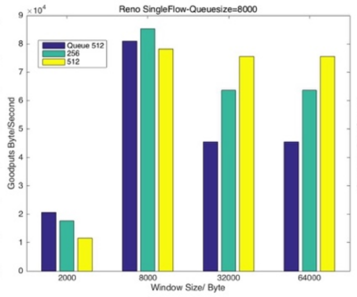


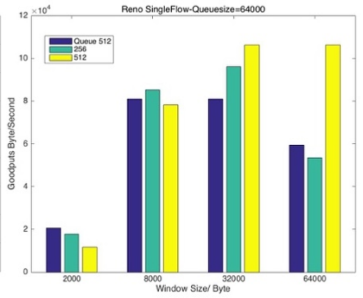




In the case of **TCP-Reno**, in which queue size is fixed at 2000, 8000 and 6400:







Observation:

(1), For all three cases, the goodput is much lower when Windowsize is 2000 Bytes.

(2), When the Windowsize reaches 8000, it doesn't make much difference on the average of different segment sets (especially when queue size is also 8000).

(3), Given low queue size (2000), the goodput of segment size 256 and 512 are at same scale in most cases. But the segment size 128 can be much lower.

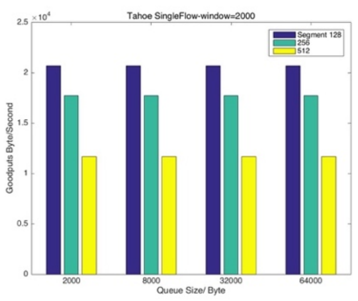
Consideration:

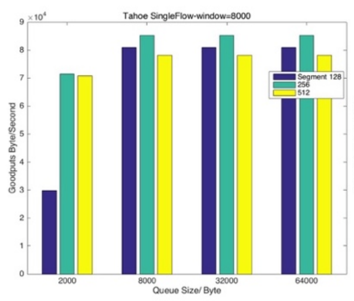
When Windowsize is 2000, it becomes the major constraint of the system. Due to this hard limit, though the traffic is greatly relieved, the utilization of bandwidth is low. Thus the goodput is much lower than other settings. As window grows to 8000 Bytes, the restriction posed by receiving side is not so stringent, queue size come to have effects. For observation (3), it can be the effect of low Queuesize and Segmentsize. Low Queuesize makes similar congestion situation and thus similar RTT for all three type of segments. In this case, 128 Byte segments do not generate much goodput for having less data per segment.

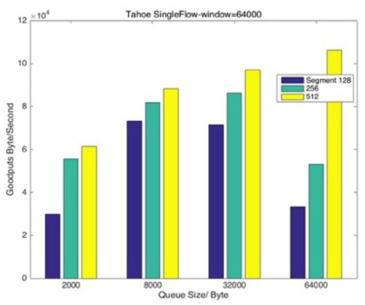
The basic feature of TCP-Reno findings is similar to that of TCP-Tahoe, except that Tahoe performs slightly better than Reno in some cases. This can be reasonable since Tahoe out performs Reno when congestion is severe. Details will be discussed later in Part 4.

1.2 Analysis as Queue Size is constant

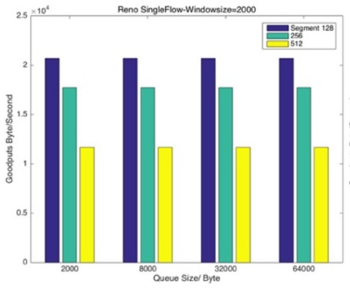
In the case of **TCP-Tahoe**, in which window size is set to 2000, 8000, 64000:

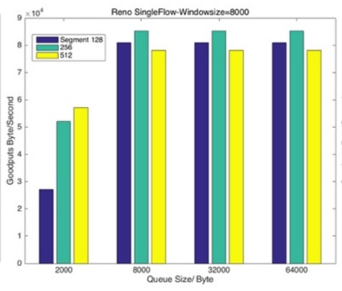


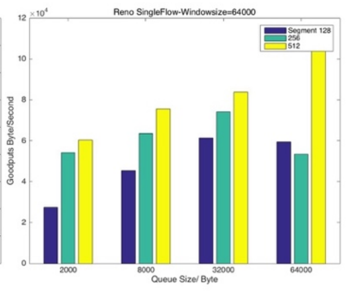




In the case of **TCP-Reno**, in which window size is set to 2000, 8000, 64000:







Observation:
(1), When Windowsize is low, I have goodput (128) > goodput (256) > goodput (512);
(2), When window size is high, I have goodput (128) < goodput (256) < goodput (512).
Consideration:

In (1), due to the low setting of window size, the limit of package on flight is constrained by window size instead of traffic. Since changing queue size does not change performance, it can be inferred that the queue is seldom used when window size is low (2000) – very low congestion.

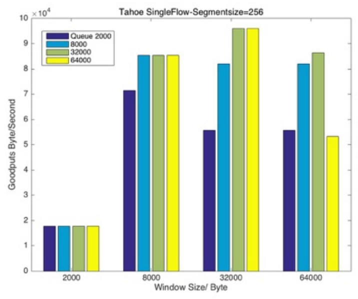
Assuming segment header is 24 Bytes: for 2000 Byte window, segment 128 can have at most 13 packets on flight (wasting 24 Bytes); segment 256 can have at most 7 packets on flight (wasting 40 Bytes); segment 512 at most 3 packets on flight (wasting 392 Bytes). Hence, the larger the segment is, the less efficient they make use of the bandwidth, when the window size is low.

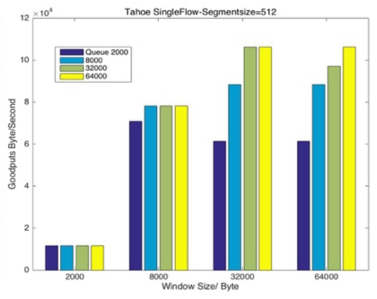
In (2), when the window size is high (64000), the performance of three segment types become reversed, and the gap between them varied according to queue size. As an extreme case, when queue size is also 64000, the packets travel with less "drop out" in the link. This makes RTT basically equivalent for all three segment sizes, making goodput nearly proportional to their segment sizes.

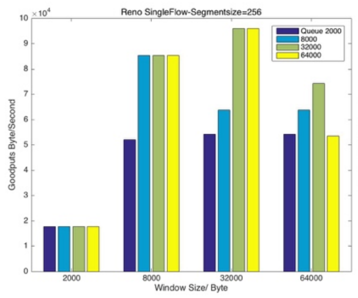
In addition to (2), when Queuesize move to lower values, due to the loose constraint of window size, the retransmission can be more often when queue size is smaller. Therefore, the performance of 3 size of segment packages are all lowered. Also, under the same queue size, the larger segments can fill up queue with less packets, making them dropped with higher possibility. This accounts for why the gap between 3 size of segment get smaller when queue size lowers (in (2)). Difference between Tahoe and Reno will be discussed in Part 4.

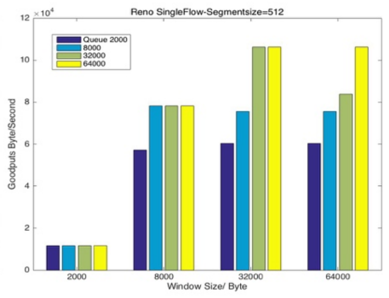
1.3 Analysis as Segment Size is constant

In the case of **TCP-Tahoe & TCP-Reno**, in which segment size is set to 256, 512:









Observation:

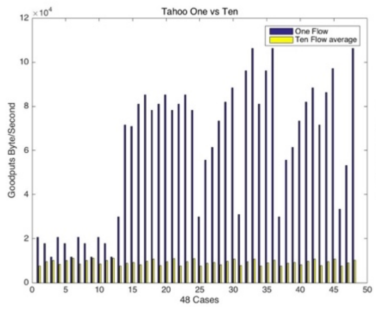
For both Tahoe and Reno, in the case of window size 6400, goodput of 256 package decreased when queue size increase to 6400; however, good put of 512 package continues to become higher as queue size increase.

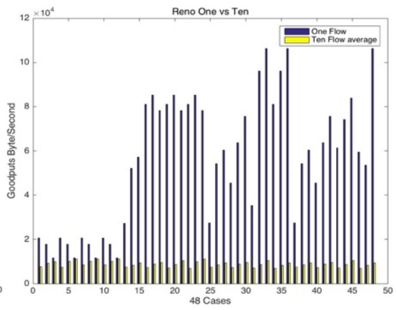
Consideration:

As mentioned in 2.2, when both window size and queue size reaches 64000, the window restrain is loose and large queue size reduce the possibility to drop out. Hence, for all three packet size, the traffic situation can be similar, and so does their RTT. But this does not mean a reduce in their RTT. When the queue is too long, a package might spend too much time waiting in the queue. I think there isn't a lower value between Time for queuing and time for retransmission for all cases. Waiting in a long queue is not efficient, and this is what happens to 256 segments when both window and queue size are 64000. But not for 512, they have more data per packet and also able to trigger retransmission with less packets.

2. Ten Flow result

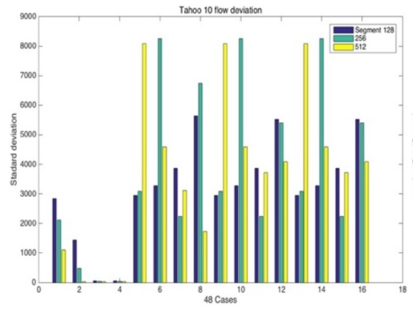
2.1 Fairness among multiple flows

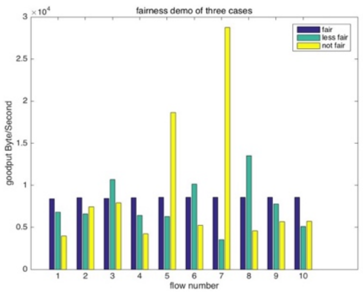




Results above are the single flow good put and average of ten flow (48 cases). The yellow bars are short since they are the average value of 10 flows. One thing worth attention is that, when the small window size greatly suppressed the goodput of one flow (as mentioned in former parts), it doesn't make much difference to the average ten flow goodput. This is natural, since sending ten flows essentially make better use of bandwidth resource.

Apart from the mean shown above, what really reflects the fairness of 10 flows are the standard deviation shown:





Left shows the standard deviation of all 48 cases of TCP-Tahoe (TCP-Reno will be discussed in part4). Right is a demonstration of how 'fair' and 'unfair' 10 flow could look like.

The fluctuation can vary greatly in 48 cases, especially for 256 and 512 segments. It is because the larger the packet is, the faster "earlier" (actually can be more complicated than "earlier") sent flow took up most bandwidth resources. They leave the congestion heavier and break the fairness.

There are also two special cases that have really low standard deviation (below 100): Windowsize = 2000 & Queuesize =32000 as well as Windowsize =2000 & Queuesize = 64000.

This is reasonable. When Windowsize is largely limited, the congestion window of the "early" sent flows won't increase too much, leaving enough space for "later" coming flows. The large queue size also helps to reduce the chance of retransmission, together makes 10 flows share the link equally.

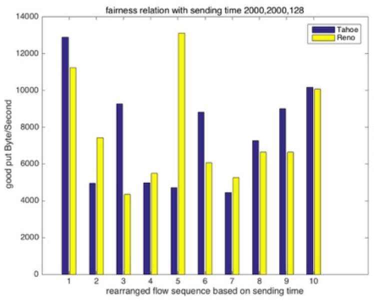
<u>Additionally, is it true that earlier sent packages always take up most bandwidth?</u>

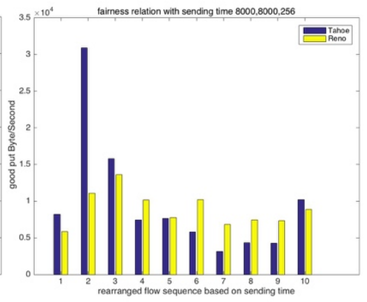
To look into this issue, I take 10 flow in 3 cases and rearrange them according to send time.

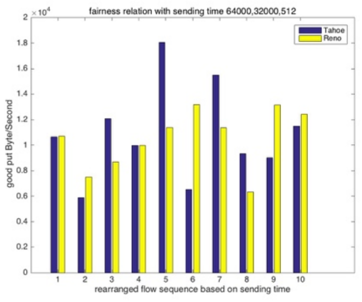
Given the seed and Stream attribute, the starting time of ten flows are fixed as:

0.0665559	0.00105638	0.0988327	0.0159708	0.0544278
0.057616	0.0571915	0.0261588	0.0596925	0.0265378

Rearranged result:





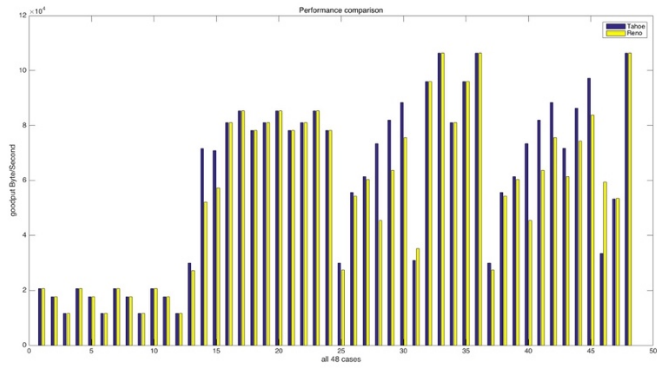


While other settings among ten flows are identical, the major source of difference is the sending time. However, the patterns above do not all show a monotonous decreasing trend according to sending time. It's also natural, since the actual packages in flight still under the constraint of window size and queue length. When a flow starts earlier than another, it does have the chance to increase toward window size earlier. But when congestion happens, since it takes up more portion of total packets on flight, it has greater possibility to be dropped by queue. This makes a greater chance to trigger fast retransmission/recovery. During the period of retransmission, the later flow might take up more bandwidth.

Hence, the mechanism is more sophisticated than the only factor of starting time.

4. Comparison between TCP-Tahoe and TCP-Reno

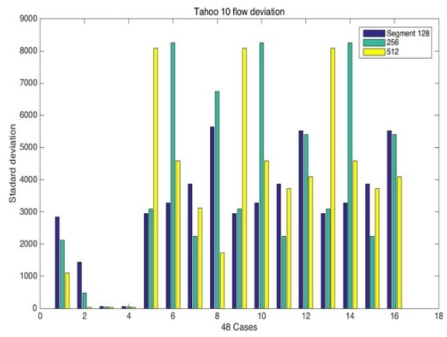
4.1 performance in one flow situation

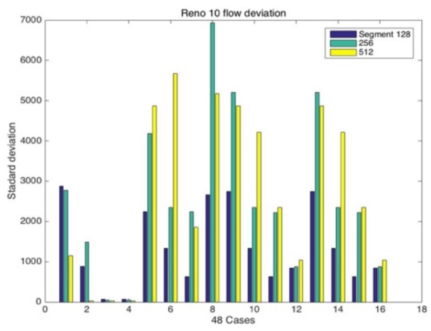


The major performance of TCP-Tahoe and TCP-Reno are consistent. By looking into detailed data, TCP-Reno's goodput is better or equal to that of TCP-Tahoe; when the queue size is lower than window size, TCP-Tahoe has more goodput.

Both Tahoe and Reno has Slow Start, Congestion Avoidance and Fast retransmit/recovery. However, in the case of 3 duplicate ack, Tahoe would retransmit and reduce Congestion Window to 1, when Reno would reduce the Congestion Window by half +3MSS. As my consideration, when window size is smaller than or equal to queue size, the number of packets on flight is limited by window and large queue size is less likely to invoke retransmission. Hence most congestion cases can be light. Reno performs better for it's not so aggressive scheme in decreasing congestion window, resulting in overall goodput. In the cases that window size is larger than queue size, congestion cases can be heavier, reducing to 1 is can be more helpful to keep traffic away from congestion. Therefore, in later situation, Tahoe outperforms Reno.

4.2 fairness in ten flow situation





This shows the standard diviation of ten-flow for Tahoe (Left), and Reno (Right) in all 48 cases. The case when standard diviation is suprisingly low is window size= 2000 and queue size= 32000/64000. This is common between Tahoe and Reno, for the reason discussed in Part 3.

Difference also exsists between them. In Tahoe, when window size increases over 2000, standard diviation is over 1500 for all cases(6 of them reach 8000). In Reno, after window size increases over 2000, there are still a few cases when standard diviation is below 1000, only one reaches peak value 7000.

Hence, within the scope of this experiment, Reno is proved to be more "fair" than Tahoe in Ten Flow link share. I think this might be the result of its less aggressive cwin reduce in 3 ack cases, thus more likely to avoid sudden change in flows.

5. Conclusion

In one flow cases:

- Best goodput is achived when segment size is 512 Bytes, Quesize = 32000/64000 & Windowsize = 32000 as well as Queuesize = 64000 & Windowsize =64000;
- In most cases, increasing segment size helps to improve goodput;
- window size as 2000 Bytes can be a major restrain on system traffic;
- higher window size means loose constraint of packts on flight and higher queue size means less probability to trigger retransmission, but tthey do not show a monotonous influence on goodputs;

It's hard to determine a gengeral trend for one flow. Analysis on the cases above can be more meaningful.

In ten flow cases:

- Best fairness is achived when Window size = 2000 and queue size= 32000/ 64000.
- Lower segment size usually makes more fairness.

As for TCP-Tahoe and TCP-Reno Comparison, when Windowsize <= Queuesize, Reno generate more goodput than or equal to Tahoe; when Windowsize > Queue size, Tahoe generate more goodput. In terms of fairness, Reno generally shows better fairness than Tahoe.