Examining TCP Performance with Network Simulator

Analyzing Congestion, Competition, and Queueing

Anthony Burwinkel
Computer Science Department
Northeastern University
Portland, ME, USA
burwinkel.a@northeastern.edu

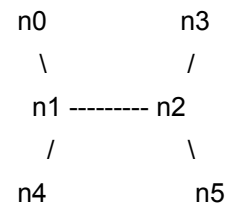
ABSTRACT

The performance of network hosts and networks in general depend largely on the strategies that the hosts themselves use to control traffic and congestion. This means designing transport layer protocols is more than a matter of ensuring performance for one client and server. Since the network's bandwidth must be shared by many hosts at once, the hosts themselves must choose a strategy that shares the bandwidth equitably. The strategy of any variant will have large effects on networks and their performance as a whole, since we will almost always have many hosts sharing bandwidth, and their decisions will all end up affecting each other.

To draw out these differences, three experiments were designed and carried out. The first experiment was concerned with the strategies of several TCP variants in dealing with network congestion caused by a non-connection oriented sender simply flooding the network with packets. The second experiment expands on the results of the first by pitting specific TCP variants against each other in this same network scenario, in an effort to gauge not only performance but fairness. The third experiment departed somewhat from the tested variables of the first two by changing the queueing algorithm of the intermediary nodes in the network topology to isolate differences in the variants.

Methodology

Each of the three experiments described in this paper were carried out in three primary phases. The first part of each experiment involved the simulation of a six node network with a set topology as pictured below:



To simulate this network, we used the open source tool NS-2, which can create network topologies, attach appropriate protocol agents to the nodes in the topologies, and simulate packet traffic moving through the network.

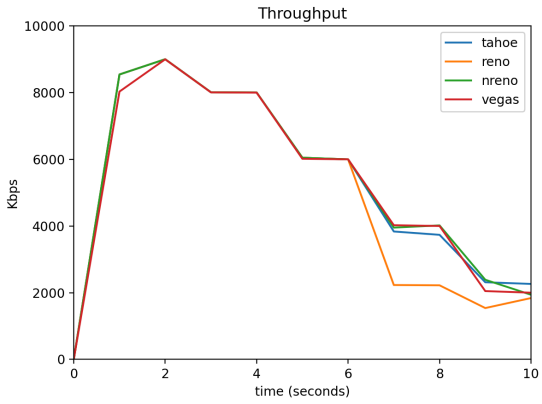
Many simulations were conducted to test the performance of different variants of the TCP protocol. After setting up the experiments and producing trace files that contained all relevant network events in our simulated network topologies, extensive scripts were developed for extracting particular data attributes of the produced trace file data sets. In particular, we were interested in measuring per-flow throughput, both over time and average, per-flow latency, both over time and average, and packet drop events over time. In addition to the standard events recorded by the NS-2 simulator, additional tracevar conditions were set to track the congestion window (cwnd) variables of the variants as the simulations ran, in the interest of investigating how their cwnd variable affected overall performance. The management of the congestion window variable is generally the most direct metric for explaining differences in performance between TCP variants.

After developing scripts for extracting the relevant attributes of each experiment, more scripts were developed for plotting the results in the popular

python library matplotlib. The produced graphs were analyzed with respect to the separate concerns of each experiment, which will be detailed more thoroughly in their individual sections. In general, a set of questions was posed at the beginning of the experiment, and the final graphs were used to propose answers to these questions.

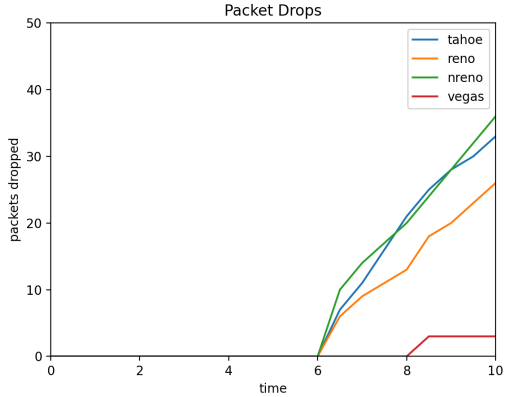
Experiment 1: Response to Congestion

For our first experiment, we set up the same six node topology in four different configurations. In every experiment, a constant bit rate flow of UDP packets was attached to node n1, and its corresponding sink was attached to node n2. The purpose of this flow was to simulate network congestion. One of four TCP agents (Tahoe, Reno, New Reno, or Vegas) was then attached to node n0 and connected to a sink at node n3. The performance of these variants was then measured as a function of the CBR flow by incrementing the CBR's send rate up for the duration of each experiment. For every 2 seconds that pass in the experiment, the CBR's send rate increases by 2 Mb, meaning that at t=2, the CBR's rate is 2Mb, t=4 CBR's rate is 4Mb, and so forth. The resulting graphs of the output traces are shown and discussed below. In particular, we tried to answer the following questions: Which variant achieves the highest throughput? The lowest latency? Which variant has the fewest drops? Is there a best overall variant, or does this metric depend on network context?



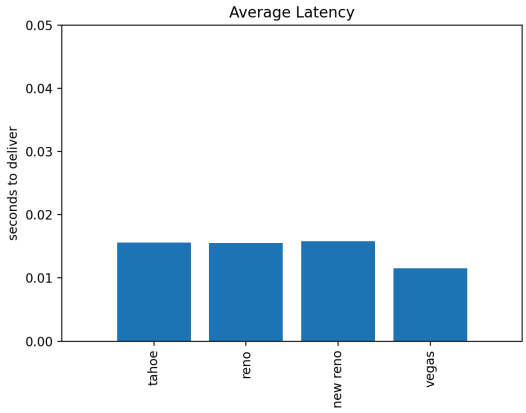
As can be seen in the graph above, each TCP variant behaves very similarly as the CBR's flow is increased. Tahoe, Reno, and New Reno each mimic each other for the first 6 seconds, likely because each one adjusts its cwnd variable down only after packet loss. Regardless, all variants' throughput per second sinks as the CBR send rate goes up, and the available bandwidth goes down. These reductions in throughput are a result of more packets being queued at nodes n1 and n2, as both the TCP flow and the CBR flow share this thoroughfare.

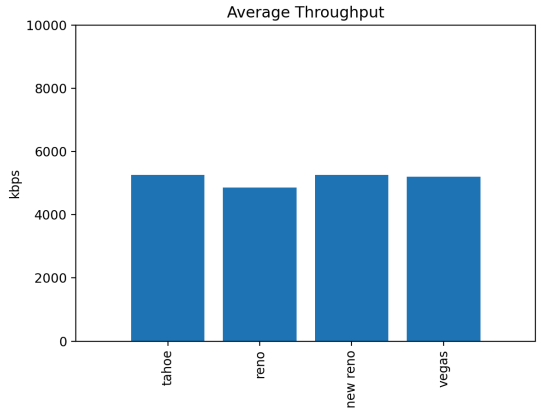
The most interesting distinctions in behavior can be observed after the network has reached a bottleneck condition, which occurs at t=6 in our set of experiments. This is when we start to see packet loss and real divergence in throughput.



Tahoe, Reno, and New Reno all shape their sending behavior based on duplicate ACKs and packet loss, whereas Vegas instead modifies its sending behavior based on expected round trip time. In particular, we can see that Reno's throughput suffers after the bottleneck, even though its packet loss is markedly lower than either Tahoe or New Reno. This is because Reno is waiting for retransmit timeouts before adjusting its sending rate upwards, a noted weakness of Reno that was addressed in the design of New Reno.

Vegas, the most conservative of the TCP variants, avoids packet loss almost entirely, while still maintaining a relatively high throughput compared to the others. Tahoe, Reno, and New Reno begin to see significant packet loss at the bottleneck, and their losses grow linearly for the remainder of the experiment as the CBR eats up more of the bandwidth.





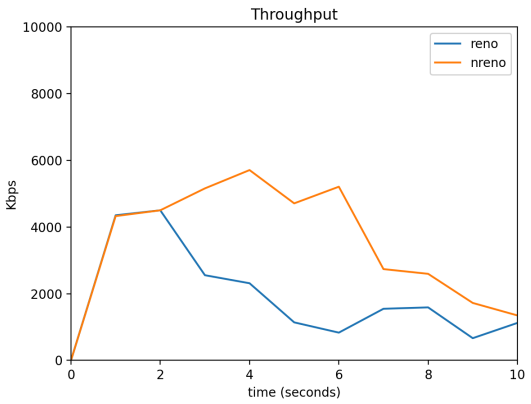
It seems that in the presence of generic network traffic like the CBR flow, which takes no pains to avoid congestion, Vegas is the clear winner. Vegas maintains a significantly lower latency than the other variants, and overall manages to maintain approximately the same throughput. While Tahoe achieves better throughput, when we consider the state of the network overall, Vegas is better. We know that in the other experiments with Tahoe and the Renos, network bandwidth is being spent shipping packets that will ultimately be dropped and need to be retransmitted. Vegas achieves almost the same throughput and utilizes less network resources.

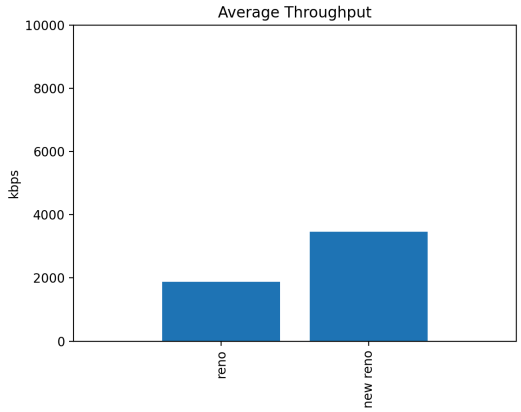
Experiment 2: Competition and Fairness

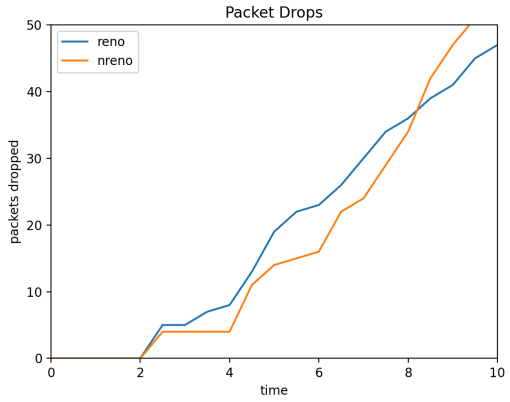
For the second experiment, competitions were arranged between the CBR flow and either two matching or two different TCP clients. The same setup and behavior was defined for a CBR flow sending from node n1 to node n2, with the same step up procedure being conducted over time, that is an increase of 2Mb in the send rate of the CBR flow every 2 seconds. Once again, a TCP connection was added between node n0 and node n3, but for this experiment a second TCP connection was established between node n4 and node n5. The two TCP clients were thus in competition with not only the CBR flow but each other as well.

The purpose of this series of experiments was to investigate the fairness of different TCP variants that are competing with each other, a relevant consideration in TCP host design. In general, designing fairer TCP variants will produce a net gain not only in overall network performance, but consequently individual performance, since any variant of TCP will typically be in competition with other instances of itself on a network. We conducted 4 separate experiments using different pairings to gauge fairness between competing variants: Reno vs. Reno, New Reno vs. Reno, Vegas vs. Vegas, and New Reno vs. Vegas. After running these simulations, we tried to answer the following questions: Are the different combinations fair to each other? Are there combinations that are unfair? If so, why?

In general, the results of the competitions in this experiment supported the findings of the first. The flaws of the Reno variant of TCP showed themselves consistently when in competition with other variants. However, when pitted against a variant like Vegas, whose strategy is focused on preventing congestion rather than responding to it, the results of competing with a TCP client are different than with a CBR flow.



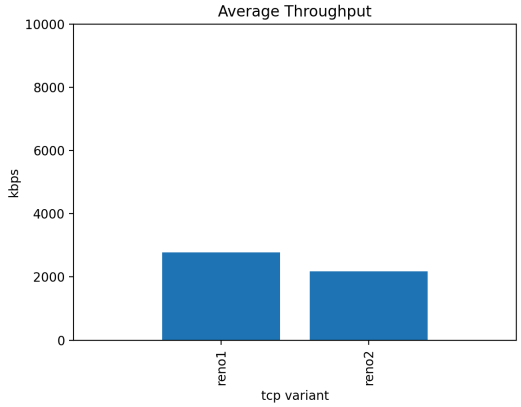


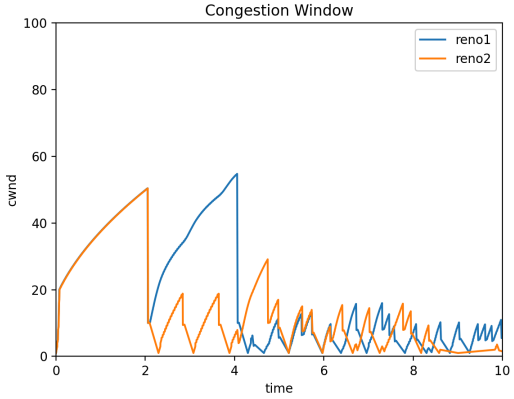


Since both Reno and New Reno use packet loss to shape their sending rates, but Reno waits for retransmission timeouts to modify its congestion window, when Reno competes with New Reno, New Reno will take advantage of this lag in throughput on Reno's part and pull ahead.

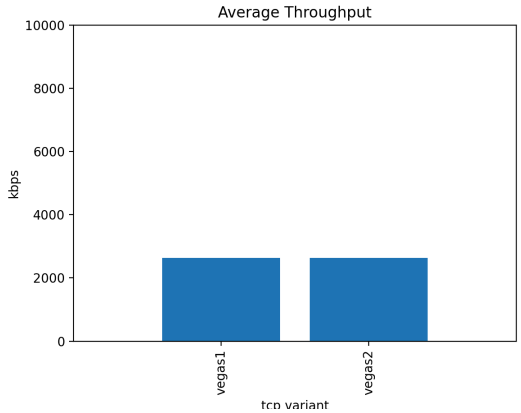
New Reno consistently dominates Reno in both metrics, despite a general similarity in their packet drop rates. Both begin losing packets at the same time, around t=2, but since New Reno remains in Fast Recovery until all of the data outstanding has been acknowledged, it will achieve higher throughput without being significantly more wasteful with network resources than Reno.

This is not the case, however, when Reno competes with itself. When Reno competes with another instance of Reno, since both delay further sending until retransmission timeout after packet loss, they achieve more similar throughput than Reno vs. New Reno. When examining their congestion windows, we can see that they both travel through the same states at different times, and thus track each other more closely than Reno and New Reno do when competing. Even so, whichever Reno variant experiences significant packet loss first will inevitably be at a disadvantage overall. As we can see, Reno 1's throughput is higher because its congestion window reinflates at t=2, unlike its competitor.

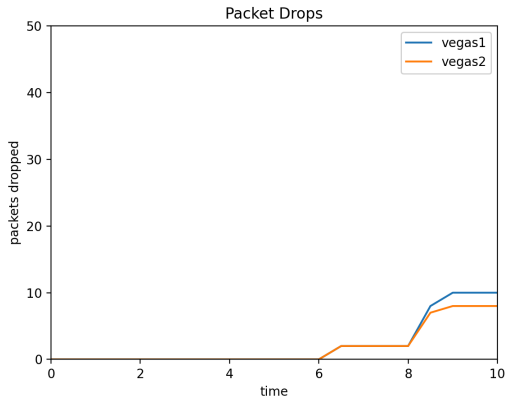




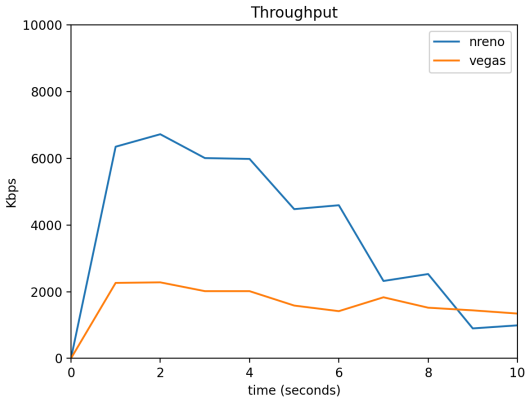
In contrast to the competitions between the different flavors of Reno TCP, the experiments involving the Vegas TCP protocol were remarkably different. In general, Vegas is the most courteous of any of the protocols that were used in experiment 2, and this shows in the results. In terms of answering the questions about fairness, the simulations involving Vegas were of the most interest.



When Vegas competes with another instance of Vegas, it is remarkably fair. Average throughput and latency for this experiment were identical. As an added benefit, Vegas also used the resources of its network the most efficiently when competing with itself. In addition to nearly identical throughput, Vegas dropped remarkably few packets. Where flavors of Reno consistently started wasting bandwidth on lost packets at t=2, Vegas only began to see losses at t=6, meaning the CBR's flow was triple what was necessary to cause packet loss for Reno.



To contrast, when competing with a TCP variant whose congestion control strategy relies on packet loss, Vegas appeared to be at a distinct disadvantage because of its conservative approach. When New Reno shares a network with Vegas, its advantage in throughput is very similar to when it competes with Reno, and for similar reasons: both Reno and Vegas are reluctant to hog network bandwidth when experiencing greater congestion.

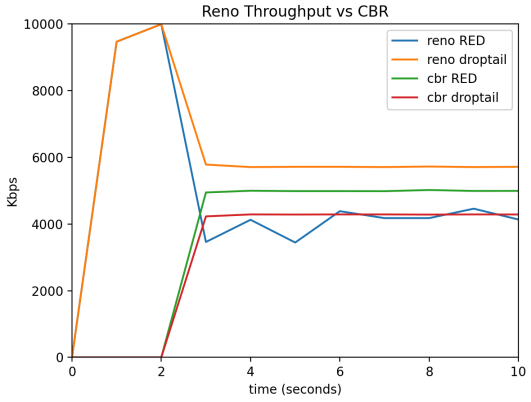


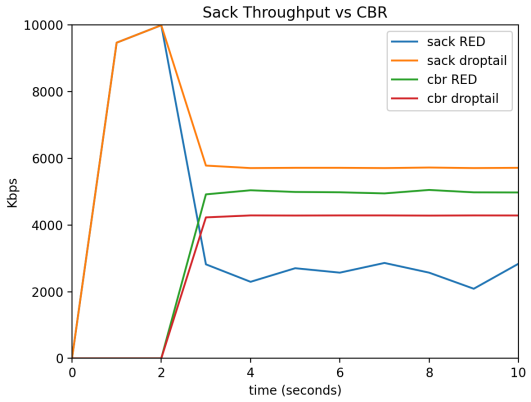
As a result, we concluded that the most fair competition was Vegas vs. Vegas, while the most unfair competition was Vegas vs. New Reno.

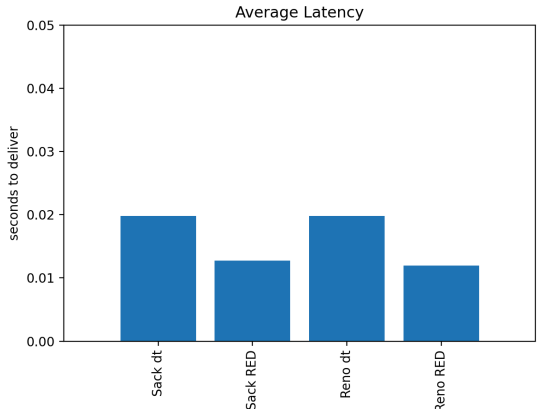
Experiment 3: Influence of Queueing

For our third experiment, the goal was to understand the influence that different queueing strategies would have on TCP clients. The same topology from experiment 1 and 2 was used, but with one TCP flow connecting node n0 to node n3, and a CBR flow connecting node n4 and n5. Rather than vary the CBR's rate, the difference between the experiments was the enqueueing method of the intermediary network nodes. For one set of simulations, all nodes used the Droptail algorithm, and for the other, all used RED. We wanted to answer the following questions: Is one queueing algorithm more fair? How do they affect latency? How does TCP react to a CBR flow in this situation?

To answer these questions, we tested two TCP variants, Reno and SACK. in each experiment, the TCP client was allowed to reach its ssthresh state before the CBR flow began to send packets. The CBR flow was given a send rate of 5Mbps and started sending at t=2, at which point we can see both Reno and SACK's throughput begin to decline as they compete for bandwidth.







In general, the CBR flow's performance in each experiment fared better in a network whose intermediary nodes use RED. This is because the CBR's UDP agent doesn't care about dropped packets, and ignores this indicator that the network is congested. TCP clients, on the other hand, adjust their sending rates whenever packet loss occurs. Since a network whose nodes use RED will begin dropping packets as soon as their queues are sufficiently full, instead of waiting like Droptail nodes would for the queues to fill completely, this results in earlier packet loss, and a net loss in throughput for TCP clients.

The most dramatic finding of this set of experiments was the low performance of SACK TCP in a network that utilizes RED queueing. We can see that SACK achieves much lower throughput than Reno in this context. The key difference between SACK and Reno is that SACK keeps track of which packets are dropped specifically, while Reno can only intuit that packet loss began at a certain segment of its sending window. Since RED is dropping random packets from its queue, this behavior confuses the SACK sender and causes it to interpret the packet loss as greater and greater congestion, so it pulls back on its sending more than Reno does. Additionally, RED implies better overall latency for all network scenarios, since a RED queueing algorithm will never fully consume network resources.

Conclusion

In this paper, we have set up three distinct scenarios in order to better understand the way different variants of the TCP protocol deal with congestion, competition over network bandwidth, and differences in the queueing algorithms employed by the network. What has been shown is that no TCP protocol can really be considered superior in all contexts, nor can any particular queueing algorithm. Network design is a delicate matter that requires careful consideration of all of the elements, and no perfect solution exists in all scenarios.

In general, we found that modifying send rate based on packet loss can be a good metric for producing high throughput, but will often lead to unfair competition between different or even the same kind of TCP variant. The queueing algorithm employed by intermediary nodes is also a factor. In particular, RED can support lower overall network latency, but can also sabotage the performance of certain TCP clients.

REFERENCES

[1] Kevin Fall, Sally Floyd. "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP." July 1996, ACM conference

[2] Lixia Zhang, David D. Clark. "Oscillating Behavior of Network Traffic: A Case Study Simulation." 1990