Performance Analysis of TCP Variants

Yifan, Zhang
College of Computer Science, Northeastern University
zhang.yifan2@husky.neu.edu

Xinyi, Zhang
College of Engineering, Northeastern University
zhang.xinyi@husky.neu.edu

1. Introduction

TCP, Transmission Control Protocol, is the most well-known protocol in current network mechanism and it was invented by Vint Cerf and Bob Kahn in 1974. However, the early stage TCP implementation fails to work out a way to solve the congestion problem that we meet in modern network environment. This implementation waits certain period for retransmitting packets, wastes a lot of network resources and makes it congested. As time changes, by now we have four typical TCP implementations that help us to solve or at least mitigate the network congestion and make network more efficient.

In this paper, we use NS2 to simulate the network congestion under different situations. In experiment 1, we explore how different TCP variants performs when we continuously increasing the noise's (CBR) rate. Plus, in experiment 2, we investigate about the fairness among different TCP variants pairs and in experiment 3, we compare the performance difference of TCP Reno and SACK when the queue parameter varies in NS2 scripts.

These differences of performance are related to which strategy the implementation takes. Basically Tahoe, Reno, New-Reno, are packet-loss-oriented which makes them perform similarly with others, but for Vegas, it is packet-delay-oriented, thus it should behave differ than other three variants. Also in the experiments, we could also observe different throughout patterns which show how algorithms like Slow Start, Congestion Avoidance, Fast Retransmission, Fast Recover etc. works.

2. Methodology

2.1 Network Topology

In this experiment, we set up a six-node network topology as Figure 1 shows. Each of the two adjacent nodes has connected with a link of bandwidth of 10MB and default delay 10ms.

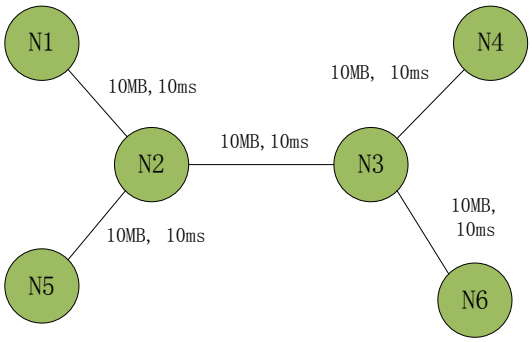

Figure 1. Experimental Topology

Thus, in experiment 1, we set up a CBR flow to get the network crowed from node N2 to N3 with varied rates, and a FTP TCP flow from N1 to N4. We test four TCP variants one by one through changing CBR rate start from 1MB and end at 10MB with increment of 0.5MB. For experiment 2, the basic settings are same as experiment 1 but we add another FTP TCP flow from node N5 to N6. We test how the two TCP flows impact each other when the network gradually become congested. For experiment 3, we test the variants' behavior and performance when different queue algorithms applied under a constant rate of CBR flow. Although there are so many queue algorithms and TCP variants, here we only focus on the performance of RED (Random Early Drop) and DropTail algorithms on TCP Reno and SACK (Selective Acknowledgement) respectively.

2.2 Tools used in experiments

Mainly in this experiments we use three tools to simulate and confirm our results. They are NS2,

Nam and NSWireless. Among these, NS2 is the major simulator and is responsible for generating both Nam readable and human readable trace file which records the process of flows. Nam can help us visualize the entire simulation process which can be used to verify whether the simulator works as we want. NSWireless is a Windows version tool which we can use it to verify the results (Through Output, Drop Rate and Latency) by simply import TCL scripts and corresponding trace file. Also we use GNUplot to make graphs according to our experiments results.

2.3 Measurement

Mainly in these all three experiments we measure TCP performance by using three typical network congestion indicators: Throughput, Drop Rate and Latency.

2.3.1 Throughput

In this case, we calculate throughput via using the total bits received by receiver node divide the total transfer time.

$$Throughput = \frac{Total\ Bits\ Received\ by\ Receiver\ Node}{First\ Packet\ Sent\ Time - Last\ Packet\ Received\ Time}$$

2.3.2 Drop Rate

Drop Rate is an indicator shows how many packets are lost during the transmission process. In the NS2 output trace file, the "d" event type means the packet was dropped during the queuing, and would not be sent. We make a guess that this scenario is controlled by queue algorithm. We calculate the Drop Rate with the following formula, in which the sent event in trace file is marked by "-" and the received event is "r".

$$DropRate = \frac{(Total\ Sent\ Packets\ \ Number - Total\ Received\ Packets\ \ Number) * 100}{Total\ Sent\ Packets\ \ Number}$$

2.3.3 Latency

For Latency calculation, in order to get more accurate and significant time scale, we choose to use average RTT (Round Trip Time) at microsecond level to present the overall latency.

$$Latency = \frac{\sum(ACK\ \ Time - Sent\ \ Time)}{Total\ \ Packet\ \ Number}$$

In this formula, the ACK time is the time that a single packet's ACK received by the original sender node (sent by the destination node), and sent time is the time that a single packet sent by sender node. We add these single latencies together and divide the total packet number.

3. TCP Performance with Congestion

As stated in the section of methodology, we compared the performances of four mentioned major TCP variants, Tahoe, Reno, NewReno and Vegas. In detail, we set up a CBR flow with changing rates that range from 1MB to 10MB with step of 0.5MB, which is used to generate traffic competing resources with TCP flow. The CBR flows from N2 to N3. And, we also set up FTP TCP flow from N1 to N4, which it will also trespass N2 to N3. It means that TCPs' performance will be impacted by the CBR flow from and might cause congestion. Our aim is to analysis how different variants perform under the same conditions. After running, we derive a bunch of data and calculate the overall average Throughput, Drop Rate and Latency.

3.1 Throughput Interpretation

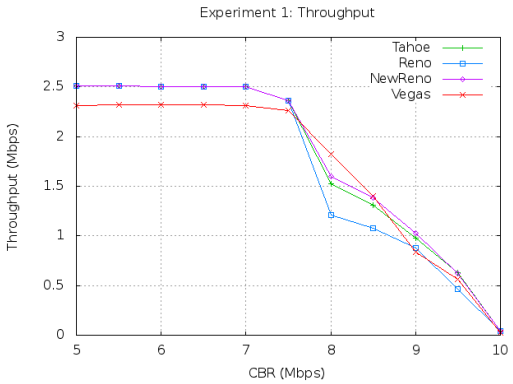

Figure 3.1 Throughput (Average) of 4 Variants

As the above figure shows, before CBR rate reaches to 7 MB, Tahoe, Reno, NewReno and Vegas are quite stable while Vegas's throughput is a little bit lower than other three. As the CBR rate continuously increases, Vegas' throughput drops at almost a constant speed. Generally speaking, NewReno has the highest average throughput.

We guess that the reason why these phenomena

take place is because Vegas choose a totally different strategy comparing with others. Vegas doesn't directly reduce the congestion window into half when there is a congestion, instead it only cares about the latency. Since that we can find Vegas decreases slower than the others.

3.2 Drop Rate Interpretation

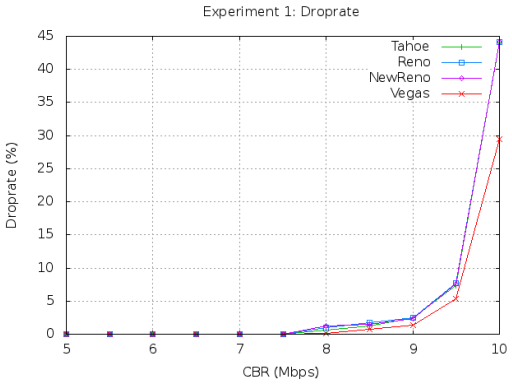

Figure 3.2 Packets Drop Rate of 4 Variants

We can see from the Figure 3.2 that before CBR change to 7.5MB, all of the four variants don't drop any packet because at that time, the congestion hasn't been formed yet. From range 7.5MB to 9.5MB, we found that Vegas has a lower drop rate than the other three. After 9.5MB, we see that all variants' drop rate climbs very quickly. However, Vegas' drop rate still keep lower when comparing with others, even if the entire bandwidth is occupied by CBR, Vegas still keeps its drop rate under 40% while almost half data was dropped by others in this case. So we can consider Vegas has the lowest drop rate.

3.3 Latency Interpretation

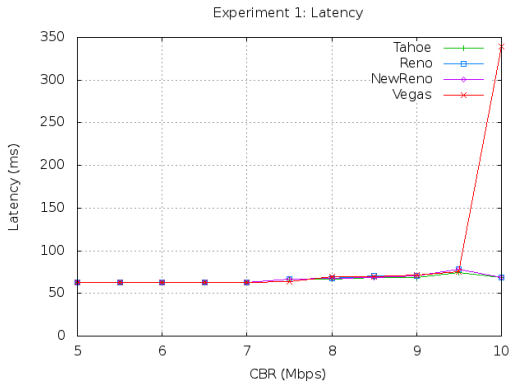

Figure 3.3 Average Latency of 4 Variants

As shown in the above figure, we can see that before CBR changes to 9.5MB, all four variants have an almost same and constant latency between 50ms to 90ms. After 7.5MB, although the four variants' latencies are same they have incremental trend. After CBR reaches 9.5MB, Tahoe, Reno, NewReno keep almost no change on the latency, but for Vegas, the latency increases dramatically. We can consider the Tahoe, Reno and NewReno have the same latency and lower than Vegas.

3.4 Summary

From above experimental data we can draw a conclusion that each TCP has its own advantages in specific field and there is no best TCP implementation to solve all problem. When we combine those three figures together for an analysis, we find that the bottleneck of current topology settings is around CBR rate of 9.5MB, which after that all three indicator changes dramatically. This is due to the bandwidth setting of 10 Mb of each link. Plus, we also find that Vegas uses totally different strategies internally when comparing with other three variants, which makes it perform completely different.

If in a network doesn't have problem of congestion, Tahoe, Reno, NewReno are better choices because they have higher average throughput than Vegas which means that they might be more efficient than Vegas. For instance, in the same condition, they might deliver more data reliably. But if we come to congestion situation, Vegas perform better than the other three implementations because Vegas has lower packet drop rate when network resources become limited. In the latency figure, we can see that Vegas has unbelievable latency when the network becomes extremely congested, comparing with other implementations. But at the same time, Vegas' drop rate is below 30% which means that if we are willing to wait enough time we finally get 70% of total data, while others might only get less than 50% of data. However, the longer the latency is, the poorer satisfaction of customer

experience. So, actually in the real world, we need to make a tradeoff decision about which is the best-fit TCP implementation in business.

4. Fairness Between TCP Variants

4.1 Fairness among TCP Variants Pairs

In this part, we make experiments on the fairness test of four different pairs of TCP variants. All the experimental settings are the same with section 3 except we add a new TCP flow from N5 to N6. We still focus on the three metrics: throughput, drop rate and latency.

1) Reno/Reno, 2) NewReno/Reno,

3) Vegas/Vegas, 4) NewReno/Vegas

4.2 Throughput fairness

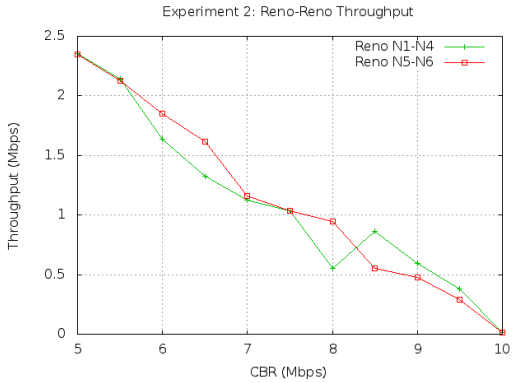

Figure 4.1 Reno-Reno throughput

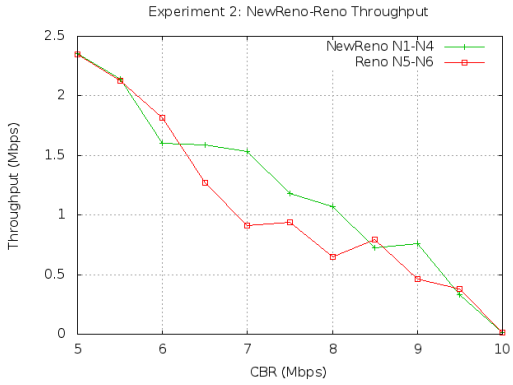

Figure 4.2 NewReno-Reno throughput

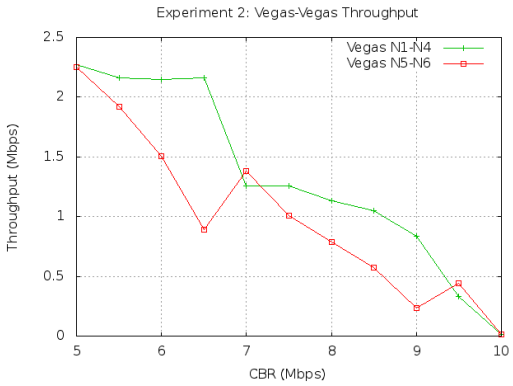

Figure 4.3 Vegas-Vegas throughput

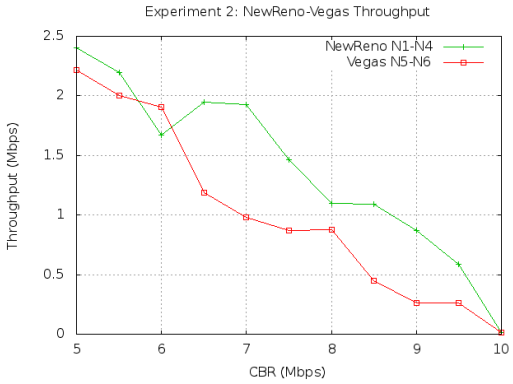

Figure 4.4 NewReno-Vegas throughput

Figure 4.1-Figure 4.4 shows the throughput of different TCP variant pairs when operating simultaneously over the same link with CBR flow. We plot these figures start at CBR rate 5 Mb, since we find that the TCP flow are steady before CBR reaches 5Mb because there is no congestion presented. We can consider the Reno-Reno pair is fair because they have similar throughput all the time. While the other three pairs are not fair because we can see one has higher average throughput in each picture. As for NewReno/Reno and NewReno/Vegas pairs, we can see the NewReno win the competition, occupying more bandwidth. As a result, we find NewReno has better congestion avoidance strategy than Vegas and Reno.

What interests us is the Vegas-Vegas pair. In this situation one Vegas performs better than the other one. We guess the reason is when two Vegas meets the congestion and have to compete with the other, they have internal algorithm that would

maintain the higher throughput flow all the time.

4.3 Drop Rate fairness

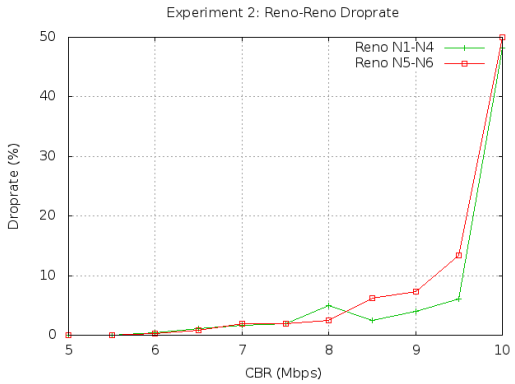

Figure 4.5 Reno-Reno Drop Rate

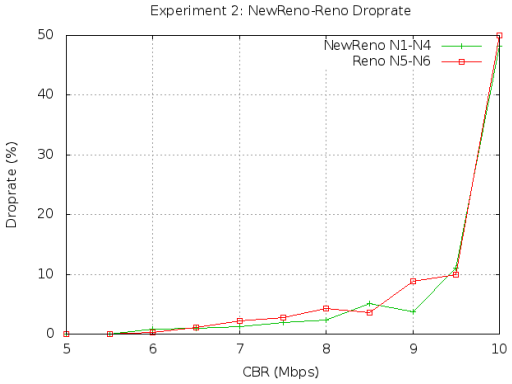

Figure 4.6 NewReno-Reno Drop Rate

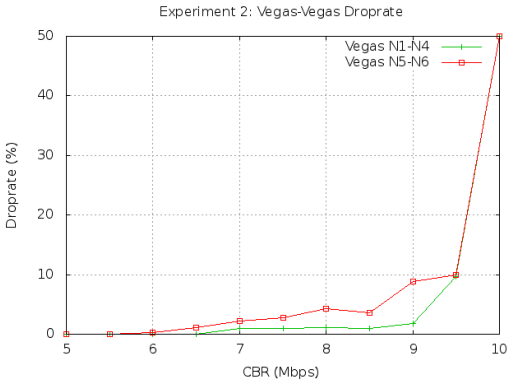

Figure 4.7 Vegas-Vegas Drop Rate

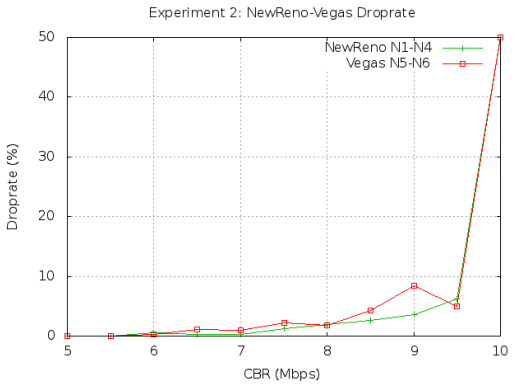

Figure 4.8 NewReno-Vegas Drop Rate

Figure 4.5-4.8 show the drop rate of different TCP pairs. We can find the corresponding results comparing with the throughput. Reno-Reno pair is fair and the other three pairs unfair. The reason of unfairness is the same as section 4.2.

4.4 Latency fairness

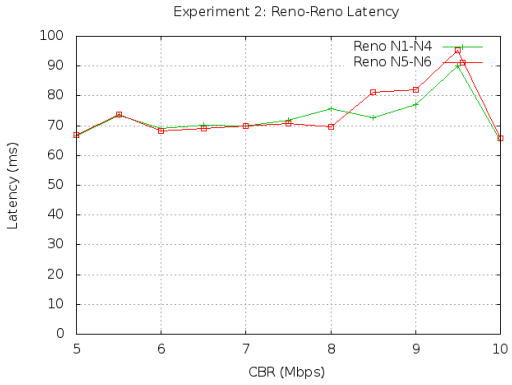

Figure 4.9 Reno-Reno Latency

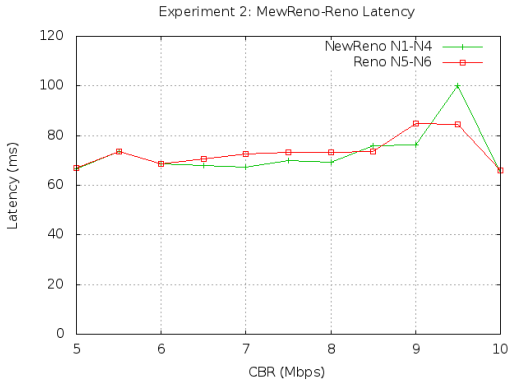

Figure 4.10 NewReno-Reno Latency

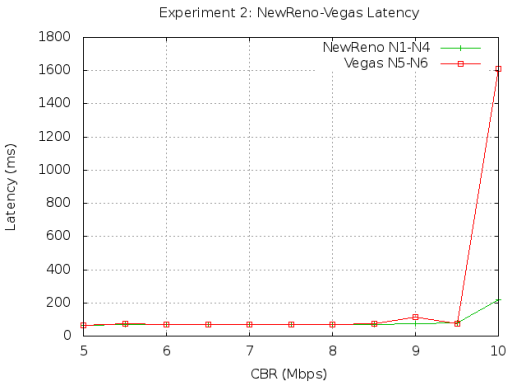

Figure 4.11 NewReno-Vegas Latency

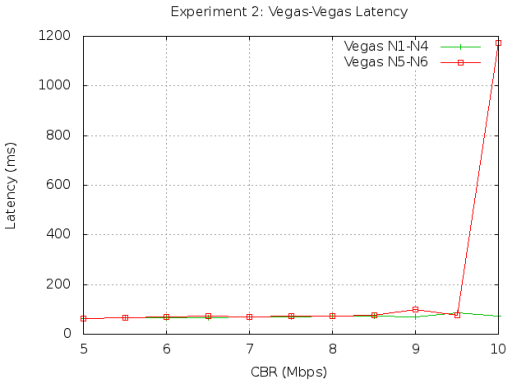

Figure 4.12 Vegas-Vegas Latency

Figure 4.9-4.12 shows the latencies of different TCP pairs. We can consider Reno-Reno and NewReno-Reno pairs are fair, meanwhile NewReno-Vegas and Vegas-Vegas are fair before CBR reaches 9.5Mb, but unfair at 10Mb. Vegas is latency-oriented, so when the CBR reaches 9.5, Vegas doesn't realize the link is very congested. At this time Vegas continue sending lots of packets, so the latency becomes very large at CBR 10Mb.

5. Influence of Queuing

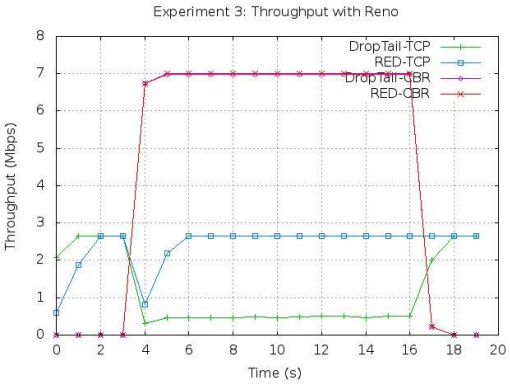

Figure 5.1 Throughput with Reno

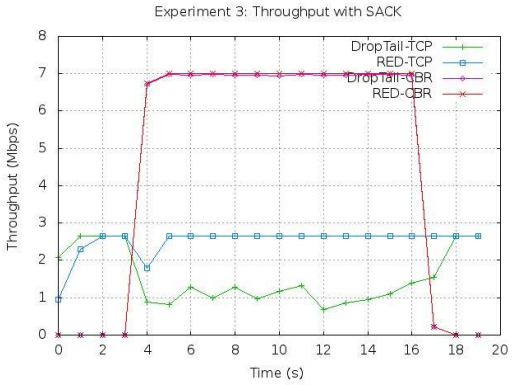

Figure 5.2 Throughput with SACK

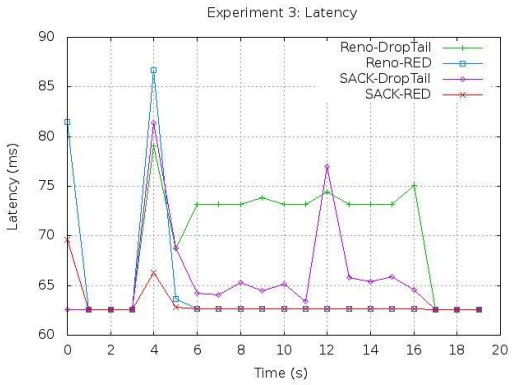

Figure 5.3 Latency

Figure 5.1-5.3 shows the influence between two TCP queuing disciplines. When using Reno, RED queue algorithm recovers quickly after CBR start, while DropTail fails to recover and stay at a very low throughput. As for SACK, RED recovers as well but DropTail starts to oscillate after the CBR starts, meanwhile has a better average throughput than the Reno situation. So the queue disciplines are unfair. When talking about latency, we find RED has lower latency than DropTail in both TCP variants, together with throughput we think RED is a better queue algorithm in Reno and SACK.

6. Conclusion

When facing congestion, each variants has its own pros and cons, we need to trade off when implementing. As for fairness, we find only Reno-Reno pair performs fair, and the unfairness comes from the congestion avoidance algorithm. Plus, RED performs better than DropTail.