**Performance Analysis of TCP Variants**

Zhongxi Wang, Shiyu Wang

Professor: David Choffnes

Feb. 25th, 2017

**Experiment 2**

This experiment mainly focused on fairness between different TCP variants. As we expected, all of these variants should be fair to one another, so different TCPs share equal bandwidth. However, we knew fairness is one of the problems of TCP variants in reality.

For this experiment, three flows are set in the network. In Fig -1, one CBR is added at N2 and sink at N3. Then, two TCPs are from N1 to N4 and N5 to N6, respectively. The bandwidth of each link is set at 10 Mbps. The performance of these TCP flows was measured by changing the CBR rate until it reached the bottleneck capacity.

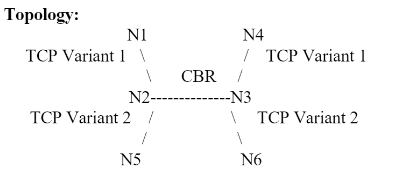


Fig – 1 Topology

In this experiment, we tested and measured average data for four pairs TCP variants, including:

1. Reno vs. Reno
2. New Reno vs. Reno
3. Vegas vs. Vegas
4. New Reno vs. Vegas

As in Experiment 1, we plot the average throughput, packet loss rate, and RTT of each TCP flow as a function of the bandwidth used by the CBR flow.

**Reno vs. Reno**

Fig – 2 Throughput of Reno vs. Reno

Fig – 3 RTT of Reno vs. Reno

Fig – 4 Packet Drop Rate of Reno vs. Reno

From the figures above, it shows the throughput of two TCP Reno are similar. As the CBR rate increase, the throughput goes done together with slightly different because of the bottleneck capacity and buffer limitation. This shows they share the bandwidth equally. Although the drop rate and latency figures show that Reno 2 has higher value, but it’s actually less than 0.05ms and 0.05%. On the other hand, the T-test was performed on the data of throughput of two TCP Reno:

|  |  |  |
| --- | --- | --- |
| T-Value | Reno 1 | Reno 2 |
| Reno 1 | 0 | 34.345 |
| Reno 2 | 34.345 | 0 |

Table – 1 T-Value of Reno vs. Reno

|  |  |  |
| --- | --- | --- |
| P-Value | Reno 1 | Reno 2 |
| Reno 1 | 1 | 2.2E-16 |
| Reno 2 | 2.2E-16 | 1 |

Table – 2 P-value of Reno vs. Reno

According to the Table – 1 and Table – 2, the P-value is way less than 0.1 and T-value is big enough to against null hypothesis. Therefore, we can conclude the data is significant that Reno and Reno are fair to each other.

**New Reno vs. Reno**

Fig – 5 Throughput of New Reno vs. Reno

New Reno has higher throughput than Reno when they are sharing the bandwidth. The Fig – 5 shows that this combination is unfair. As the CBR rate increase, it is obviously that New Reno always has higher throughput. Again, the T-test was performed against the null hypothesis with big T-value and tiny P value.

|  |  |  |
| --- | --- | --- |
| T-Value | New Reno | Reno |
| New Reno | 0 | 16.747 |
| Reno | 16.747 | 0 |

Table – 3 T-Value of New Reno vs. Reno

|  |  |  |
| --- | --- | --- |
| P-Value | New Reno | Reno |
| New Reno | 1 | 0.00003629 |
| Reno | 0.00003629 | 1 |

Table – 4 P-value of New Reno vs. Reno

Fig – 6 RTT of New Reno vs. Reno

Fig – 7 Packet Drop Rate of New Reno vs. Reno

In another aspect, Reno has higher Drop rate and RTT in the Fig – 6 and Fig – 7. It is because New Reno has the advantage of fast retransmit which can detect multiple packet losses and does not need to wait for retransmission of packet as in the case of Reno. This gives New Reno priority in bandwidth share.

**Vegas vs. Vegas**

Fig – 8 Throughput of New Reno vs. Reno

Fig – 9 RTT of Vegas vs. Vegas

Fig – 10 Packet Drop Rate of Vegas vs. Vegs

From the figures, although two variants have similar latency and drop rate, Vegas 1 has a little bit higher throughput in the bandwidth share. The reason is because Vegas is different from other TCP variants that it applies delay-based congestion avoidance. It considers queue building rather than loss of segment for congestion detection to adjust the sending rate. Hence, when it notices the rate is lower compared to the expected rate, it increases its rate of transmissions to utilize the bandwidth. Conversely, transmission rate becomes lower when rate is very to the expected rate, which helps to avoid over saturation of the bandwidth of the network. This is the reason that both of them have amazingly low Drop rate. In this case, one Vegas increased the sending rate. And the other one detected it and reduce the transmission rate correspondingly. As shown in Fig – 8, the difference value between two Vegas are changing over time. As soon as they get close, one will start trying to increase rate and the other acts oppositely. However, it won’t influence the RTT and Drop rate. Therefore, we conclude two Vegas variants are fair in bandwidth share, only slightly difference in throughput because of congestion avoidance algorithm. T-test is provided as well against null hypothesis. Again, the small P-value and large T-value can well prove the value is significant.

|  |  |  |
| --- | --- | --- |
| T-Value | Vegas 1 | Vegas 2 |
| Vegas 1 | 0 | 16.15 |
| Vegas 2 | 16.15 | 0 |

Table – 5 T-Value of Vegas vs. Vegas

|  |  |  |
| --- | --- | --- |
| P-Value | Vegas 1 | Vegas 2 |
| Vegas 1 | 1 | 0.00008732 |
| Vegas 2 | 0.00008732 | 1 |

Table – 6 P-Value of Vegas vs. Vegas

**New Reno vs. Vegas**

Fig – 11 Throughput of New Reno vs. Vegas

Fig – 12 RTT of New Reno vs. Vegas

Fig – 13 Packet Drop Rate of New Reno vs. Vegs

Comparing between New Reno and Vegas, the former is more aggressive. No matter about the start time, New Reno takes most of the bandwidth as the CBR rate gets closer to bandwidth. Because of the congestion avoidance algorithm talked in the last section, Vegas calculate have calculated the bandwidth while New Reno is still in the slow-start and has no idea about the bandwidth. During this period, Vegas reduces its transmission rate after every RTT, meanwhile, New Reno takes the majority of the bandwidth. Therefore, New Reno has a better throughput in Fig – 11. Therefore, in this pair, TCP New Reno is unfair to Vegas. T-test is provided below to against null hypothesis.

|  |  |  |
| --- | --- | --- |
| T-Value | New Reno | Vegas 2 |
| New Reno | 0 | 17.964 |
| Vegas | 17.964 | 0 |

Table – 7 T-Value of New Reno vs. Vegas

|  |  |  |
| --- | --- | --- |
| P-Value | New Reno | Vegas |
| New Reno | 1 | 0.000148858 |
| Vegas 2 | 0.000148858 | 1 |

Table – 8 P-Value of New Reno vs. Vegas

**Experiment 3**

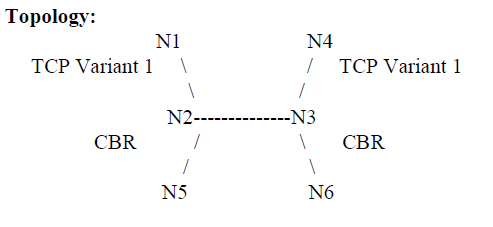


Fig – 13 Topology

In the last experiment we compare the influence of queuing displaces of two algorithms, Drop Tail and Random Early Drop (RED). As shown in Fig – 13, One TCP is set up from N1 to N4 and started at the start. After the TCP flows is steady in one to two seconds, the CBR will start running from N5 to N6.

During the forty times experiment with various starting time of CBR, average throughput, packet drop rate and latency are recorded every 0.5 seconds and analyzed with different queue displaces algorithms.

**Does each queuing discipline provide fair bandwidth to each flow?**

For this experiment, it is obvious the queuing discipline doesn’t provide fair bandwidth to each flow, because the CBR flow does not care about the packet drops. However, the TCP flow are affected.

**How does the end-to-end latency for the flows differ between DropTail and RED?**

Fig – 14 RTT of DropTail vs RED for Reno

Fig – 15 RTT of DropTail vs RED for Sack

From the Fig – 14 and Fig – 15, it shows no matter for Sack or Reno TCP, RED algorithms take more time during transmission. The reason is RED monitors the queue size and drops packets based on the queue capacity. Hence, there will be a small delay as the packet transmitted in the flow. By contrast, DropTail will increase the buffer size over time.

**How does the TCP flow react to the creation of the CBR flow?**

Fig – 16 Throughput of DropTail vs RED for Reno

Fig – 17 Throughput of DropTail vs RED for Sack

According to the Fig – 16 and Fig – 17, the throughput of TCP decrease dramatically when CBR flow is added into network. As starting of CBR flow, the packets start dropping because of the limited queue size. However, after a short time, DropTail gives a better throughput as started, then decrease again for both Sack and Reno. Oppositely, the throughput using RED algorithm keeps the low rate. The is because DropTail only drops packet when the buffer is full, so it can keep the high throughput when there is still space in buffer. However, RED drops packet randomly when it detects congestion based on its calculation. And the probability of dropping packets is proportional with the queue length.

**Is RED a good idea while dealing with SACK?**

As Fig – 17 shows, Yhe RED with Sack shows better than Red with Reno in Fig – 16, although it still has lower throughput than using DropTail algorithm. The reason is RED requires random and more resending than DropTail while transmission. Because Sack has a different sink using selective ACKs, which includes received, out-of-order sequence numbers in TCP header to tell sender about holes in the sequence. Therefore, it is much easier for sender to resend packets than using Reno.

**References**

R. Braden, V. Jacobson and L. Zhang, “TCP extensions for highspeed

TCP Vegas: New Techniques for Congestion Detection and Avoidance L. Brakmo, S. O'Malley and L. Peterson Proceedings of the SIGCOMM '94 Symposium, August 1994, pg. 24-35.

Fall, Kevin, and Sally Floyd. "Simulation-based comparisons of Tahoe, Reno and SACK TCP." *ACM SIGCOMM Computer Communication Review* 26.3 (1996): 5-21. Web.

"RED (Random Early Detection) Queue Management." *RED (Random Early Detection) Queue Management*. N.p., n.d. Web. 28 Feb. 2017.