Measuring Internet Censorship in Iran from Multiple Vantage Points

Neeki Hushyar
UMass Amherst
nhushyar@cs.umass.edu

Angela Upreti
UMass Amherst
aupreti@cs.umass.edu

Keywords

TCP variants; SACK; Reno, Tahoe

Web traffic runs over TCP and given the scale at which the internet has grown and how much we rely on the web, it is important that the underlying transport protocol is able to operate at a reasonably low latency and provide high throughput. Network congestion causes packet loss which increases latency and hence, reduces throughput. Using the right TCP variant saves both money and time. Amazon found that every 100ms latency cost then 1 percent in sales [1]. Higher latency/wait-time in web applications discourages clients from revisiting the cite. This is specially true for video applications such as youtube. Clients usually close the page if the latency is high. Fairness in a the underlying transport layer is equally important. One flow should not hog most of the network bandwidth and throttle other flows. Otherwise, different customers paying the same amount for internet service would get widely different performance. Metrics such as throughput, average and end-to-end latency, packet drop rate, fairness are different across different tcp variants and the congestion control algorithms they use.

Congestion control algorithm for TCP was proposed by Van Jacobson in response to 1986's 'congestion collapse'. The original TCP proposed by Van Jacobson contained elements such as dynamic window sizing, slow start, additive increase and multiplicative deacrease (AIMD). Since then, several variants of TCP have been proposed. TCP Reno adds fast recovery and fast retransmit to TCP Tahoe. TCP Reno's fast recovery and fast retransmit suffer when multiple packets are dropped from the same window. In presence of multiple packet loss, the TCP pipe often gets drained. NewReno has to wait for retransmit timeout to be able to send data again. Consequently, multiple packet loss leads to low throughput. TCP NewReno was introduced to solve this. When in fast recovery, TCP NewReno interprets a single partial Ack as a signal to retransmit another packet rather. NewReno does not wait for 3 duplicate ACKs before retransmitting. TCP SACK is another TCP variant that solves the multiple packet drop problem in Reno. SACK allows the receiver to acknowledge any non sequential packets it has received. NewReno without SACK is unable to tell exactly which packets are missing. While NewReno can retransmit only one lost packet per RTT, TCP SACK can transmit as many lost packets as the congestion window allows. Variant Vegas uses RTT as an estimate of congestion rather than using packet loss.

We conduct three different simulation studies in this paper:

1. Compare performances of Tahoe, Reno, NewReno, and Vegas under congestion.
2. Compare fairness between different TCP variants.
3. Study influence of queuing disciplines (1)RED and (2) Drop Tail on TCP Sack and TCP Reno.

1. EXPERIMENT 1: ANALYZING PERFORMANCE OF TCP VARIANTS

subfig

This experiment analyzed the performance of different TCP variants over a setup consisting of a single TCP stream and a single CBR flow. The performance of the TCP variants were analyzed over a range of different packet sizes and transmission rates assigned to the CBR. We analyze the results of this experiment in two parts. First, by looking at the performance of the variants over an increasing range of CBR packet sizes, while the rate of the link is constant at 2Mbps. Second, we look at the performance of the variants over an increasing range of tranmission rates - from 1Mbps to 10Mbps, while the size of the packet is held constant at 3,000 bytes.

When holding the transmission rate steady, at 2Mbps, none of the variants experienced any packet loss over the range of increasing packet sizes, from 500 bytes to 8,000 bytes. The average and end-to-end latency as well as the throughput for TCPs Vegas, Reno, NewReno and Tahoe were all the same. This is consistent with the fact that these variants differ only in the ways they retransmit lost packets.

Vegas keeps the link as fully saturated as possible, to achieve higher bandwidth utilization. Given no congestion TCP Vegas maintains a higher throughput than TCPs Reno, NewReno and Tahoe.

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

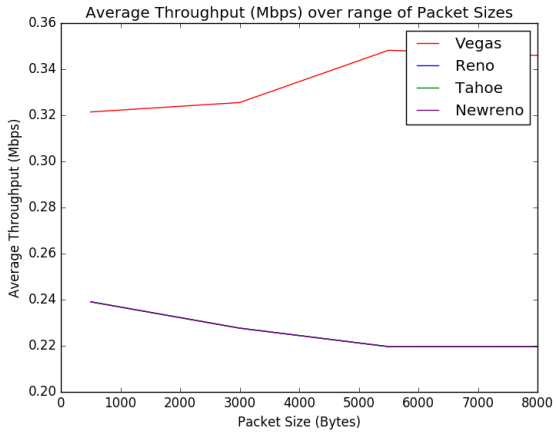

Figure 1: TCP Vegas (red) maintains a higher throughput than TCPs Reno, NewReno, and Tahoe (all purple) in a network with no congestion.

In a network with congestion, TCP Vegas takes preventative measures in avoiding it. The other variants make changes only in the aftermath of packet loss. To prevent congestion, Vegas records the round trip time (RTT) of packets and adjusts its congestion window according to any delay. As shown in Figure 2, TCPs Reno, NewReno and Tahoe experience their first packet loss when the CBR rate is set to 6Mbps. Vegas does not experience packet loss until the CBR rate is 9Mbps.

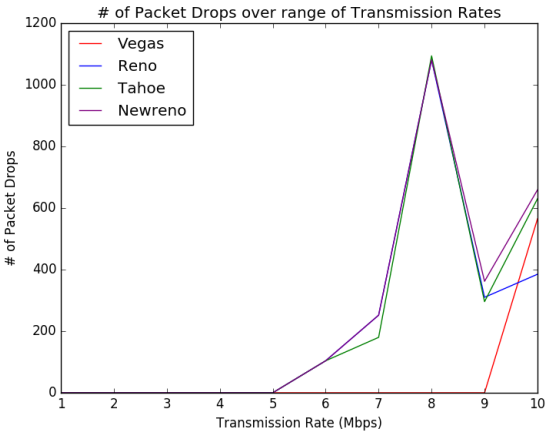

Figure 2: This figure shows the number of packet drops when the CBR flow has a rate of 10Mbps and the packet size is increasing.

Despite packet loss occuring at a rate of 9Mbps in TCP Vegas, Figure 3 shows the throughput decreasing, before experiencing packet loss. This is because Vegas is sensing the increased congestion on the network as a result of increasing RTTs and decreasing its congestion window accordingly as a preventative measure, before experiencing packet loss. As a result, the latency also increased.

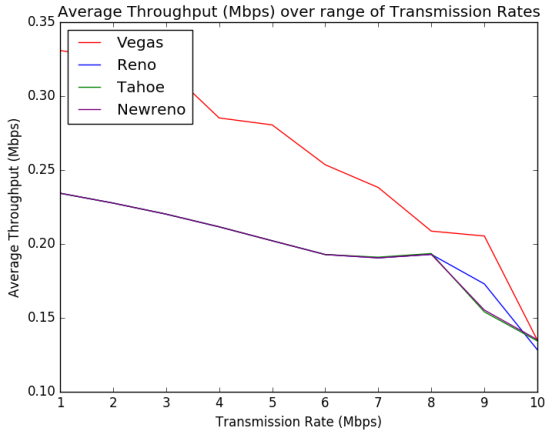

Figure 3: Throughputs of each of the TCP Variants as the CBR flow rate increases, causing contention.

Each variant is consistent, only in that they wait until receiving 3 duplicate-acknowledgements or retransmit timeouts before retransmitting the first lost packet. Given a network with a single TCP variant, which experiences little congestion, TCP Vegas experiences higher throughput and lower latency.

Given low-to-moderate levels of congestion, TCP Vegas is able to efficiently decrease its congestion window size, before experiencing packet loss, and thus is able to maintain 0 packet loss while its counterparts are unable to. However, once some threshhold is reached, for example a CBR flow rate of 10Mbps in our experiments, Vegas inevitabley suffers packet loss, and at this point it is no longer the most efficient variant in terms of dealing with loss. NewReno, handles multiple packet losses better than its counterparts. After the first packet loss is identified by a triple duplicate acknowledgement, or by a retransmission timeout, NewReno assumes that any packet following a partial acknowledgment is lost. This makes the retransmission of packets much faster and accounts for the increase in throughput experienced by NewReno in Figure 3. Given moderate to high amounts of congestion, NewReno has higher throughputs and lower latency than the other variants.

2. EXPERIMENT 2: ANALYZING FAIRNESS BETWEEN TCP VARIANTS

This experiment used two TCP flows and a single CBR flow, to observe the fairness or lack there of, between the TCP flows over different pairs of variants. We tested different combinations of variants, including assigning TCP Vegas to both flows, TCP Reno to both flows, NewReno to one flow

and Reno to another, and lastly, NewReno to one flow and Vegas to another. Further, we ran these tests twice. The first test analyzes fairness in a network with a CBR flow rate of 1Mbps and a packet size of 1000 Bytes. The second test analyzes fairness is a network with a CBR flow rate of 10Mbps and a packet size of 4000 bytes. The first test simulates a network with no congestion, and the second test simulates a network with congestion and packet loss.

In the network with no congestion (test 1), the only combination of TCP flows which did not exhibit the same throughput rates, was that of TCP NewReno and TCP Vegas. TCP Vegas had a higher throughput than TCP NewReno, similarly to what we found in experiment 1. Because there was no congestion on the network, the flows acted as they would independenltly of each other. The combinations of NewReno/Reno and Reno/Reno, experienced identical throughputs, latencies, etc. because there was no packet loss, so they essentially followed the same protocol in this simulation. Vegas/Vegas also split the throughputs evenly, because they both follow the same protocol, however their throughputs were significantly lower than any other combination we tested, again consistent with what we found in experiment 1.

This experiment analyzes the performance of TCP Reno and TCP Sack using the queuing algorithms: DropTail and RED. TCP Sack, increases and decreases the congestion window the same as TCP Reno does, when dealing with the first packet loss. Upon the first sign of packet loss, both variants cut the congestion window in half, and proceed to retransmit the lost packet.

In situations involving multiple packet losses, TCP Sack waits until the number of outstanding, or yet to be acknowledged packets, is less than the congestion window. While in fast recovery, TCP Sack keeps a record of outstanding packets as well as a corresponding counter. Reno only has the foresight to keep track of a single lost packet. Sack uses partial acknoweldgements to decrease the counter of outstanding packets at a rate which increases the congestion window faster than slowstart, which is why the performance of Sack benefits from a slightly higher throughput and a slightly lower latency than Reno.

DropTail fills up the queue, and drops any further packets when the queue is full. It is first come, first serve. Random Early Detection (RED) accepts all packets when the queue is below some threshhold, and randomly accepts packets when the queue is above the threshhold but still less then the maximum capacity. This allows it to reduce the power of large and fast packet burst which would otherwise monoplize the bandwidth and fill up the queue (in the case of DropTail).

Our experiment held constant a CBR flow rate of 8Mbps and a packet size of 4,000 bytes, which cause congestion and packet loss in the network. The results of the experiment showed an average throughput using RED in TCP Reno and SACK was .214 Mbps and .225 Mbps, respectively. The average throughput using the DropTail queuing algorithm in Reno and SACK, were .148 Mbps and .150 Mbps, respecitvely. The increased throughput using the RED queuing algorithm is because the network is able to maintain a more steady flow of traffic, and avoid being under or over utilized. Random dropping however, did double the drop rate - because packets were dropped when they were above some threshold to encourage fairness among the tcp traffic. TCPs Sack and Reno experienced 118 and 116 dropped packets using DropTail and experienced 216 and 267 dropped packets using RED.

The combination of TCP Sack using the RED queuing algorithm, consisted of the highest throughput, lowest average and end-to-end latencies of the four combinations of variants and queuing algorithm. As previously mentioned, it must be noted, there was a relatively high number of drop packets in comparison to results computed when using DropTail. It comes down to the amount of congestion on the network. If there is moderate congestion, it is efficient to use TCP Sack and RED together. If there is a rate of congestion, far exceeding the threshhold using in RED, at which point, packets are randomly dropped, it may be less efficient to use this queuing algorithm.

3. ACKNOWLEDGMENTS

4. REFERENCES

[1] R. Kohavi and R. Longbotham. Online Experiments: Lessons Learned. *Computer*, 40(9):103–105, Sept. 2007.