Basant Solanky

12040430

Assignment- 2

Transport Layer and Network Simulations using NS-3

PART 2

Study of TCP Congestion Control Algorithms

TCP CUBIC

Cubic is designed to achieve high-bandwidth connections faster and more reliable over long-distance networks. In the cubic algorithm, a function differentiates it from the standard TCP protocol. The cubic function increases the window size, which solves the congestion problem (slow increment in congestion window) in the standard TCP network with the large bandwidth-delay product.

Algorithm:

To make the network more scalable and ultimate, the Cubic algorithm uses a cubic window increase function to track the elapsed time from the last congestion event.

- CUBIC starts with a slow start algorithm when the congestion window is less than the ssthresh. Depending on the network's speed and distance, it may choose either TCP standard slow start or hybrid slow start.

- If congestion is detected by duplicate ACKs or Explicit Congestion Notification-Echo (ECN-Echo) ACKs, Cubic registers the congestion window size where it gets the congestion event as ‘W_max’ and performs a multiplicative decrease of the congestion window.
- After entering congestion avoidance, it widens the congestion window using the cubic function's concave profile. The **concave** window rise continues until the window size reaches W_max since the cubic function is configured to peak at W_max. After that, the convex window increase starts, and the cubic function transforms into a **convex** profile.

Window increase function-

$$\text{cwnd}= W_{cubic}(t) = C * (t - K)^3 + W_{max}$$

cwnd:- The congestion window at the current time.

t:- The elapsed time since the current congestion avoidance began.

C:- Constant.

K:- The time to increase the current window size to W_max.

□:- Multiplicative decrease factor

$$K= \sqrt[3]{\frac{Wmax * (1-\square)}{C}}$$

TCP Friendly Region

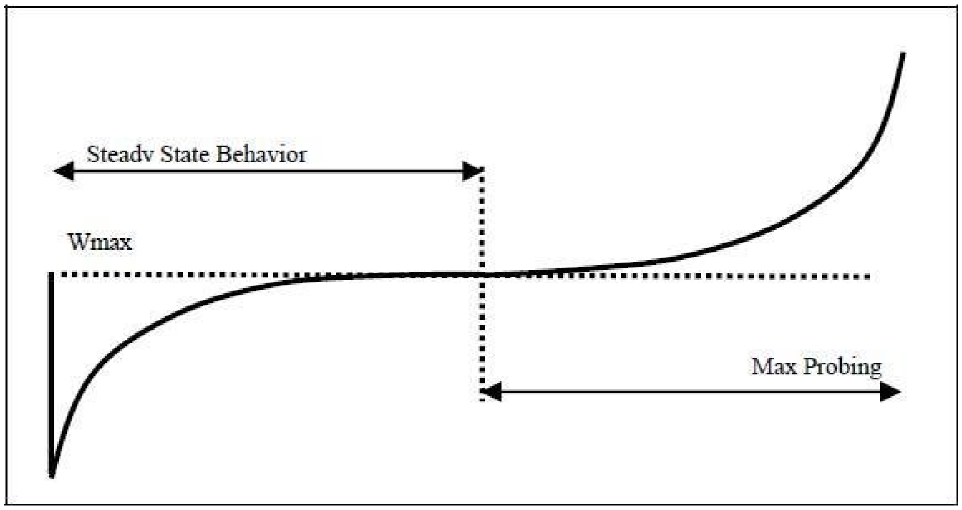
Standard TCP functions well on some networks, such as those with short RTT and little bandwidth. We employ the TCP-friendly region in these networks to be sure. This area makes sure that CUBIC delivers a throughput that is at least equal to Standard TCP.

Concave Region

Cubic is in the concave region if cubic is not in the TCP-friendly zone, and cwnd is smaller than W_max when receiving an ACK in congestion avoidance. In this region, the congestion window must be incremented $\frac{Wcubic\ (t+RTT) - cwnd}{cwnd}$ for each received ACK.

Convex Region

When the Cubic crosses the W_max, it moves through the concave region. CUBIC is carefully and gradually increasing its window size in this area. In this region, the congestion window must be incremented $\frac{Wcubic\ (t+RTT) - cwnd}{cwnd}$ for each received ACK.



CUBIC window growth function

There are some limitations to using Cubic algorithms according to the above figure:-

- It could become slow if the new W_max is too much bigger than the previous W_max since it will take a long time to approach the inflection point.

- The Previous W_max determines the TCP Cubic's aggressiveness. The congestion window will initially rise slowly if the former W_max is less. In networks with a short lifespan, it could cause slow responsiveness.

Pros of CUBIC Algorithm

- TCP Cubic uses a cubic function's concave and convex profiles to scale the congestion window size for optimal network utilization.
- We now understand that TCP Cubic suits networks with high capacity and great distances. It adheres to those standards for networks with limited bandwidth or short distances, where the TCP standard functions well.
- Cubic is built to achieve bandwidth sharing across flows with various RTTs to improve RTT fairness.
- Cubic balances scalability and convergence speed by choosing an appropriate multiplicative window decrease factor.

<u>TCP New Reno</u> (Loss-Based)

This algorithm for congestion is loss-based. To get around TCP reno's constraints, TCP Newreno was created. Let's examine the TCP Reno's limitations.

For each packet lost in TCP Reno, we drop our cwnd by 50%. Assume that the current size of our congestion window is 1024. Then, it would only take 10 packet losses to reduce it to 1 and another 10 RTT to return to 1024, which is inefficient. Multiple packet losses in the same congestion window must be detected slowly in TCP Reno.

The following restrictions are addressed by the TCP new Reno. It informs the sender that it must only cut the cwnd by 50% once to account for all packet loss in the same congestion window. This can be achieved using something called partial ACKs.

Algorithm:
When a packet is lost, it will switch to fast recovery mode, much like TCP Reno. After retransmitting a packet, it will determine whether the ACK is new or not, considering whether or not all packets in that particular cwnd have been acknowledged by the receiver. The sender will see ACK as a new, otherwise partial ACK if the receiver receives all packets of that specific cwnd. If only a portion of the message is acknowledged, the sender won't reduce the cwnd by 50% once more and won't exit the fast recovery phase.
TCP New Reno can identify numerous packet losses in the same window using the partial ACK again when the New Reno sender receives the first retransmitted packet's ACK. It waits for the ACKs for all the other packets in the same window rather than treating it as a fresh ACK like Reno does. It assumes several packets are lost in the same window if it doesn't receive those acks.

Pros

- In the same window, it can detect multiple packet loss.
- For multiple packet losses, it can resist decreasing the window size by 50% in the same window.

Cons

- If the window size is too tiny, we won't get enough duplicate ACKs for quick retransmission, and we'll have to wait for a timeout before sending a lost packet again.
- We have to wait for 1RTT to detect each packet loss.

FAST-TCP (Delay-Based)

It is a congestion control technique based on delay. Fast TCP uses queueing delay as a congestion indicator. Since delay is a continuous quantity and packet loss only provides partial information about the congestion level, delay is considered a better signal for checking congestion.

Algorithm:

Throughout the network, FAST TCP works to keep a constant volume of packets in the queue. The number of packets in queues is considered when comparing the observed RTT to the base RTT (i.e., the RTT when no queueing is present). The smallest observed RTT for the connection is roughly considered to be the base RTT. The rate changes depending on how many packets are in the queue; if there are few packets, the rate increases; if there are many, the rate decreases.

Pros

- It can keep the size of the window constant.
- It can detect congestion faster than loss-based algorithms since delay represents partially filled buffers and loss results from filled buffers.

Cons

- When combined with TCP Reno, it needs to receive its fair share.
- Operating system scheduling makes delay measurements jitter-sensitive.

TCP Compound (Hybrid-Based)

This algorithm uses loss and delay signals for congestion, making it a hybrid congestion control algorithm. In contrast, if we were to use the delay-based approach, we would not receive a fair share of bandwidth compared to other loss-based approaches, which would be unfair to other Reno users. We need to develop a strategy that falls somewhere between the two options.

Algorithm:

Compound TCP maintains two congestion windows, one using a delay-based technique and the other a simple AIMD approach. The total of these two windows determines the size of the actual sliding window used. As in the TCP Reno technique, the AIMD window grows. The delay-based window quickly expands to maximize the network's bandwidth while the delay is minor. The delay window drops when the queuing delay is raised to balance the rise in the AIMD window. Keeping is the goal.

Pros

- In a network with large values of the bandwidth-delay product, it can produce a throughput that is significantly higher than the TCP New Reno.

Cons

- Compared to other connections corresponding to TCP, TCP Compound may become unfair.

<u>Comparison of all four variants:</u>

Four TCP congestion control techniques have been demonstrated. One utilized delay as a signal, one (FAST) used packet loss as a signal, two (CUBIC and NEW RENO) utilized packet loss as a signal, and the final one (COMPOUND) utilized both packet loss and delay as a signal to regulate congestion.
TCP Cubic is far superior to the New Reno since it uses the bandwidth considerably more effectively thanks to the cubic function than the New Reno does.
Additionally, it makes excellent use of TCP Standard for networks with short RTT or minimal bandwidth delays. When TCP Cubic and TCP Fast are compared, FAST is superior because it uses delay as a signal, which is a better way to judge congestion. Still, CUBIC triumphs because FAST encounters unfair bandwidth compared to straightforward Reno algorithms, whereas CUBIC doesn't encounter that problem in the first place.
TCP CUBIC and Compound TCP are now. Compound defeats CUBIC because it employs both loss and delay as a congestion signal, giving it the best of both worlds regarding congestion signal quality and bandwidth utilization. Again, CUBIC is superior to Compound because it makes greater use of a vast BDP network while Compound uses the AIMD window.