AIMD and BBR scribe

Sai Anirudh Kondaveeti

2018/05/02

1 Congestion Avoidance and Control

'Congestion collapse' is the state of network where the network is delivering packets at its capacity, but most of it is useless as most of the packets are duplicates (caused by retransmissions). These 'congestion collapses' started happening around 1886 and this paper was published in 1888 to avoid this problem. This algorithm was built on the ideas introduced by the DECbit where they tried to avoid congestion by updating a bit to signal congestion. But this approach was not implementable as it involved changing functionality of routers. So a more general way to detect congestion through estimation of packet losses is suggested. These changes were just required at the end host. Also this is the first implementation of congestion control on Unix kernel(BSD 4.3). This implementation is one of the key reasons for the wide adoption of this algorithm. The other reason for its wide adoption is the dire need of a solution to avoid 'congestion collapse'.

1.1 Conservation of packets

The condition that "A new packet isn't put into the network until an old packet leaves", ensure that a collapse will be avoided. This principle only fails in the following ways:

- The connection doesn't get to equilibrium, or
- A sender injects a new packet before an old packet has exited, or
- The equilibrium can't be reached because of resource limits along the path.

It is to be noted that the older version of TCP wasn't following this principle. It was just dumping a large window of packets onto the network. This led to overflowing of buffers causing persistent retransmissions.

1.2 Algorithmic Ideas

The main contributions of this paper are:

- Slow start algorithm to get to the equilibrium
- Retransmission timeout estimation using rtt variance so as to maintain the equilibrium.
- Exponential back-off which describes how the retransmits be spaced when a packet has to be retransmitted more than once. This technique helps in maintaining equilibrium.
- Congestion avoidance which describes how the congestion window has to increase when there is no congestion and how the congestion window has to decrease in case of congestion.

These ideas were very fundamental and almost every Internet connected computer in the world ran a slightly modified version of this algorithm between 1988 and 2004.

1.3 Fixing this large window problem

Instead of using a large window size at the start of transmission we need to find the right window size at the current network conditions. To do that we start our window size by 1 and increase it by 1 on every ACK. By following this rule our window size gets doubled every RTT. This is an exponential increase in size of window over time but still it is considered slow compared to a fixed large window size. This increase in window size stops when we detect a packet loss through a timeout. Whether the bottleneck link capacity is 56 Kbps or 10 Mbps, we get to the optimal window size very fast as it is exponential increase.

1.4 Retransmission timeout

Estimating the retransmission timeout is essential. If we estimate it to a large value and retransmit the packet late then we are holding up the delivery of data. We are holding up the data because TCP provides an in-order reliable bytestream. If we estimate it too small and retransmit packets then we are hogging up the network resources which will lead to congestion collapse.

In this paper a new method for retransmission timeout was suggested using rtt variance. The previous approaches were not estimating the rtt variance and were using a constant factor beta over mean RTT to estimate retransmission timeout. Here the beta accounted to rtt variance. This led to wrong estimation of retransmit timeout interval. In this paper they suggested a faster (Optimizing RTT mean and variance calculation using integer arithmetic instead of floating point) and accurate way to estimate variance which led to setting a better retransmission timeout. This retransmission timeout is adaptive as it tracks the mean RTT and variance of RTT more closely (compare Figure 6 and 5).This led to avoiding unnecessary retransmits and maintain the network at equilibrium.

1.5 Exponential back-off

Another scenario which previous TCP implementations didn't deal correctly is the scenario when the retransmitted packet is lost. The loss of a retransmitted packet means the estimates could be way off. So this problem is dealt by doubling the retransmission timeout until an ACK is received. Once the ACK is received the timeout estimation logic is reset and restarted. In practice there is an upper bound on the maximum size of retransmission timeout (typically 60 seconds). This exponential back-off nature of retransmission timeout can be explained by the analogy that the network is a linear system. If a linear system is unstable (about to have congestion), stability is achieved through exponential damping (exponential timer back-off) to its primary excitation (senders, traffic sources).

1.6 Congestion avoidance

After the slow start which gets to the optimal window size we need to cautiously explore for more bandwidth or decrease our bandwidth depending on network load. This congestion avoidance phase allows different flows to be in equilibrium and consequently allows them to converge to their fair share. In the steady state the window size is increased by 1/cwnd on every ACK. This can also be seen as increasing the window size by 1 packet every RTT (additive increase). In case of packet loss during this increase the window size is decremented by a factor of 2 (multiplicative decrease).

1.7 Why Additive Increase Multiplicative Decrease (AIMD)?

This is the right combination because this algorithm converges all the flows to an equilibrium (Chiu-Jain plots). The reasoning behind multiplicative decrease is that the queue lengths in the router increase exponentially which can be countered by multiplicative decrease in windows size. By doing this multiplicative decrease the queues will stop growing and consequently drain. The paper doesn't explain the reasoning behind additive increase but most probably the reason could be that the alternative which is multiplicative increase leads to congestion.

1.8 Gateway congestion control

There is a need for congestion control algorithms at gateways as the end hosts can't ensure fairness at a granularity less than a second. Also it is not expected of the every user to use the congestion control algorithm which gives rise to some user unfairly using other user's share of bandwidth. To mitigate these problems we need to introduce some intelligence in gateways. These problems are tackled in the next lecture through protocols like XCP and WFQ.

1.9 Overall summary of the paper

The paper successfully demonstrates a new congestion control algorithm which achieves better effective bandwidth (figure 11).

2 BBR

This paper was published in 2016 which is nearly 30 years after the TCP Tahoe. Most of the TCP congestion algorithms after TCP Tahoe were interpreting packet loss as congestion. But this premise isn't valid in the current Internet with speeds in the order of Gbps and memory chip sizes in the order of GB. With the high link capacities the current implementations of TCP are not able to achieve optimal bandwidth due to packet losses on a path with multiple hops (Handley's paper "Only just works" provides an example of this in Section 4.2.1). With large buffer sizes it takes a long time to see a loss. Especially on the cellular links which maintain large buffers to handle extreme swings in link capacity.

2.1 Symptoms of the problem with TCP

Because of the changes in technology new problems have arised and were solved case by case. For example TCP window was limited to a maximum value by the OS vendors on cellular networks. This was done to counter the effect where a TCP download filled up the buffers and caused delays to other traffic on mobile devices. Another example would be development of alternative congestion control algorithms for intercontinental links because the current implementations only give few Mbps worth of throughput on a link which supports Gbps.

2.2 Cause of the problem

The above solutions are makeshift and aren't dealing with the actual cause behind them. The crux of their problem is due to interpretation of loss as congestion signal. All these loss-based congestion control algorithms are just a sustained operation to the right of BDP line (figure 1) and congestion control is a scheme which bounds how far right of this line can a connection operate. This point is neither optimal in terms of bandwidth, delay and loss for a individual user nor the network as a whole.

Aside: In this paper BBR is compared against CUBIC which is a variant of TCP Tahoe (different increase rule). But it doesn't matter whether we compare it against CUBIC or TCP Tahoe as both of them can be seen as a loss-based TCP.

2.3 Now, onto BBR

Figure 1 shows RTT and delivery rate variation with the amount of data inflight. It has three regions namely app limited, bandwidth limited, buffer limited. App limited is the region where the application can't sent enough data to saturate the link. In this region we will have optimal RTT but we don't know about the bottleneck bandwidth (BtlBw). Bottleneck bandwidth limited region is the region where the link is saturated and the queues start absorbing any excess traffic. In this region we know BtlBw but we see that the RTT increases with increasing queue size. Buffer limited region is the region where the application has started overflowing the buffer. In this region we know the BtlBW but we also see losses because of packet dropping. Loss-based TCP operates at the intersection of bandwidth limited and buffer limited regions. But a much more desirable point is at the intersection of the bandwidth limited and app limited regions. Because at this point we have maximum bandwidth and minimum delay. This is also the point which maximizes power i.e throughput/delay.

This is the optimal operating point for both individual user and the network as whole. Though this paper doesn't go into the details it suggests that this algorithm converges to this optimal operating point in a distributed settings (figure 6).

Aside: Other algorithms have shared similar goals of high throughput and low delay over the last ten years, e.g., DCTCP, TIMELY, Sprout, Verus, and Remy. But they have different models of the network, i.e., they are tuned for different network settings.

2.4 BBR: A decentralized algorithm that attempts to optimize power

Regardless of how many links a connection traverses or what their individual speeds are, from TCP's viewpoint an arbitrarily complex path behaves as a single link with same propagation delay (RTprop) and bottleneck bandwidth (BtlBw). In case of multiple flows the RTprop is same as previous one but the BtlBw is the fair amount that particular flow was supposed to get. Though the algorithm converges to the fair point this paper doesn't discuss how fast it gets to that fair point.

2.5 BBR: the details

As described earlier RTprop and BtlBw are the two parameters which were needed to be estimated so that the optimal amount of data is sent. To estimate RTprop, the minimum over a time window which is typically tens of seconds to minutes is used. The minimum is used because RTprop will be minimum when it is at the optimal operating point. To estimate BtlBw, the maximum over a time window which is typically six to ten RTTs is used. The maximum is used because the bandwidth will be maximum at the optimal operating point. Since RTprop is visible only to the left of BDP and BtlBw oly to the right (figure 1), they obey an uncertainty principle: whenever one can be measured the other cannot. This is because when we are hitting the bandwidth limits, we can't measure the propagation delay accurately, and similarly when we can measure the propagation delay accurately, it's unlikely we are saturating the link.

Also, there is some more bookkeeping to keep track of when the application had no traffic to offer so that we don't underestimate the bottleneck bandwidth because the traffic is application limited.

2.6 BBR: measurement and control

Typically the measurement and control are separated but BBR combined these two. It used two gain parameters pacing gain and cwnd gain to move between Startup, Drain, and ProbeBW states. Which parameter to use is decided by the state machine (see Figure 4 for these states).

2.7 BBR vs. CUBIC

From figure 4 it can be seen that BBR starts up rapidly (similar to CUBIC's slow start). But then BBR drains it queue size by reducing its sending rate (but CUBIC keeps increasing). Once the queue drains to zero, BBR starts probing in ProbeBW state (but CUBIC is still increasing until the buffer overflows). In figure 5 it can be seen that the CUBIC flow will ultimately have loss due to buffer overflow whereas BBR sends data optimally without any losses.

2.8 Fairness

Similar to TCP Tahoe, BBR doesn't directly tackle fairness. Fairness is implicit in the sense that each sender backs off in Tahoe/BBR in response to negative externalities to other senders (high delay in case of BBR or loss in Tahoe). Because of this implicit fairness these can only guarantee fairness at multi-second timescales. For finer time scales, we need router intelligence. In figure 6 it can be seen that it takes around 35 seconds to converge.

2.9 Real-world experiments comparing wide-area throughput

BBR was tested on B4 which is Google's high-speed private network connecting its large datacenters. From figure 7 it can be seen that BBR has gains from 2 to 25 times relative to CUBIC. In figure 8 it can be seen that BBR performed far better than CUBIC when the loss rate was increased.

2.10 YouTube experiments

In case of YouTube deployment, the gains were not very substantiative. The reason could be mostly due to the fact that YouTube was highly optimized. The BBR algorithm only reduced the mean RTT by 53%.

2.11 Experiments on mobile networks

Figure 10 describes the steady state median RTT variation with link buffer size on a cellular link. It can be seen that as buffer size increases, the latency for CUBIC flows increase. This leads to failed connections, whereas BBR performs well as it tries to maintain zero queue size.

2.12 Overall summary

This paper assumed a clean and reasonable model of the network. It has fewer handwaving arguments than the congestion avoidance paper and also has real experiments with Google traffic. But, it has too many parameters and mechanisms. The application level gains were a bit more limited in some contexts. Overall, BBR has delivered on the promise of high throughput and low delay.