CSC458 Programming Assignment 2 Report

Ziyue Zhang
University of Toronto
violetzy.zhang@mail.utoronto.ca

Bufferbloat Problem

What is bufferbloat?

Bufferbloat is the undesirable latency caused by excess buffering of packets [1]. We all know that buffer is a region where we store data temporarily while data packets are transmitting from one place to another since there is a difference between the rate at which data is received and the rate at which it can be processed. Buffer is used to prevent or decrease the packet loss caused by the difference between the arrival rate and departure rate and used to balance these two different rates.

With a larger buffer, we could store packets that we cannot process at the moment and save it for later to deal with and we add extra space to store them instead of letting them on the road, it sounds like we could prevent packet loss and alleviate traffic congestion. But the fact is that high latency is generated because of the large buffer size, this is what we called bufferbloat.

Why does bufferbloat occur?

Many congestion control algorithms are based on the packet loss to know the available bandwidth. These algorithms will increase the transmitting rate until packet loss is detected and then decrease the cwnd. Actually, TCP needs packet loss as feedback to adjust the cwnd to control congestion. The large buffer size makes it difficult for data packets to be dropped, they will be in the queue waiting for a long time. TCP does not know the occurrence of congestion and it will continue to increase the congestion window. The queue will get longer and the packet will wait for a longer time. This will cause high latency and decrease the network throughput.

What is the impact of bufferbloat?

High latency caused by bufferbloat is obvious when you use latency–sensitive applications like VOIP(Voice-over-IP service), video streamers, and online games, users suffer from the high latency and feel painful to use the slow service [2].

As the total traffic becomes heavier, network traffic patterns will grow burstier and more chaotic. Usage of individual links will swing rapidly and crazily between emptiness and overload cascades [2].

Another impact is the packet loss, it sounds ironic because the original intention of the large buffer size is to prevent the packet loss. Bufferbloat makes the packet loss increase dramatically once all the buffers are full [2]. It seems like the traffic realizes it should slow down until it runs out all of the buffers, but this time the congestion is more horrible and difficult to control.

Back to the initial motivation of the router, it is used to forward the packet instead of storing the packet, the reason why we add a buffer to the router is to store data packet temporarily to balance the rates instead of storing numerous packets without adjusting the traffic. We now know that unnecessary buffer does not make our life easier, it causes bufferbloat which harms the latency—the performance characteristic users care about most [2].

Simulation Setup

Simulation Topology

To learn the poor performance caused by the bufferbloat, we use the Mininet to simulate this problem. Within the Mininet, create the topology:

h1——(s0-eth1) S0 (s0-eth2)——h2

h1 represents the home computer with the 1Gb/s connection to s0 which has a slow uplink connection of 10Mb/s. h2 represents a server. The round-trip propagation delay between h1and h2 is 4ms.

We perform the simulation with three different queue sizes: Q=5 packets, Q=20 packets, and Q=100 packets.

Simulation Tasks

The performing tasks are: (1) Start a long–lived TCP flow sending data from h1 to h2, record the congestion windows size (cwnd) of this connection and plot the time–series of the long–lived TCP flow's cwnd. (2) Send 10 pings per second from h1 to h2, record their RTTs and plot the time–series of the RTT reported by ping. (3) Spawn a webserver on h1, download the index.html web page from h1 repeatedly every two seconds and measure download time and plot the time–series of webpage download time. (4) monitor the queue size and plot the time–series of queue size at the router.

Simulation Analysis

Difference in webpage fetch times with short and large router buffers

From the simulation, we have calculated the average and standard deviation of download time with different buffer sizes. The following table shows the different performances.

Table 1: The difference of download time between buffer size of 5 packets, 20 packets and 100 packets.

buffer size	5 packets	20 packets	100 packets
Average	1.0151	0.841	2.1748
Standard deviation	0.548987999262	0.195628280784	1.09942005621

The performance of fetch times with a buffer size of 20 packets is better than a buffer size of 100 packets in terms of whatever average and standard deviation.

It is because a larger buffer size makes packets uneasy to be dropped, instead, they will wait to be served in the queue slowly. TCP keeps sending packets and increasing cwnd, not knowing the occurrence of the congestion, so the queue becomes longer and the packets in the queue will wait for a longer time. So bufferbloat brings a longer queue, longer waiting time, and higher latency.

Also, we notice that the performance of the buffer size of 20 packets is slightly better than the buffer size of 5 packets. That is because the buffer size of 5 packets is too small, the loss-based congestion control misinterprets loss as a signal of congestion, leading to low throughput.

Different queue sizes and congestion windows

The plots of the time-series of the queue size and cwnd at the different sizes of routers are as follows.

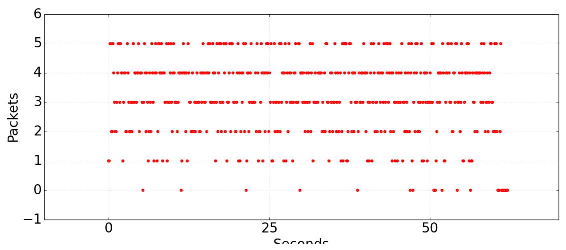


Fig. 1: queue size in 5 packets size buffer

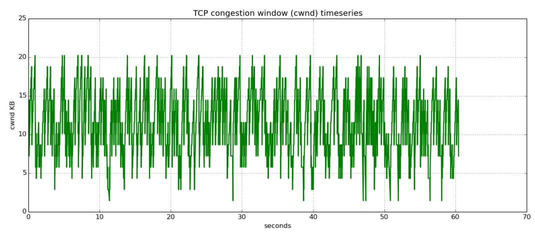


Fig. 2: congestion window in 5 packets buffer

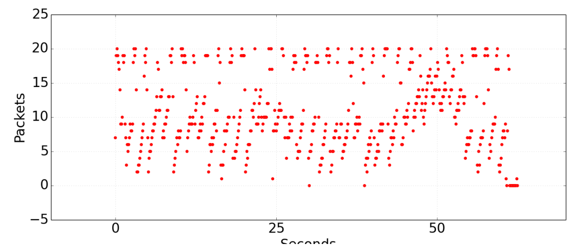


Fig. 3: queue size in 20 packets size buffer

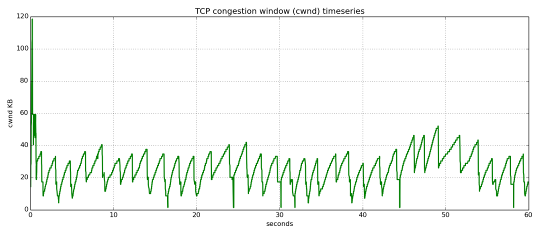


Fig. 4: congestion window in 20 packets buffer

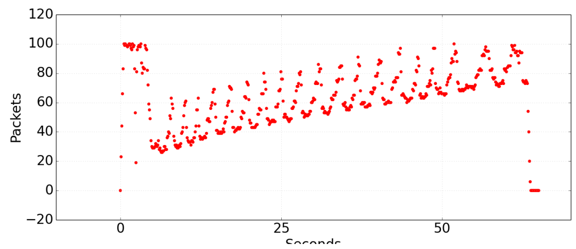


Fig. 5: queue size in 100 packets buffer

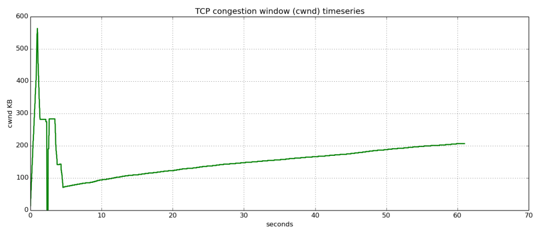


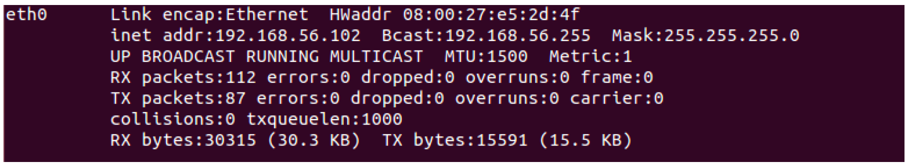
Fig. 6: congestion window in 100 packets size buffer

For the buffer size of 5 packets, we can see that the queue size in the buffer swings rapidly(see Fig.1). This is because the buffer size is small and packet loss is easier which will make TCP to adjust the cwnd frequently(see Fig.2) and the buffer size is small, so you can see that the queue size jump from empty to full frequently.

For the buffer size of 20 packets, we can see that the queue size oscillates regularly (see Fig.3)and most time, the queue is not filled up the buffer and it keeps the space for the future packet. Observe the corresponding congestion window, the cwnd turns sawtooth clearly(see Fig.4). In the buffer size of 20 packets, seeing from two figures, TCP controls the congestion steadily.

For the buffer size of 100 packets, we can see that the number of the congestion window changes are much smaller than the previous two, in most of the time, it increases its window continuously(see Fig.6). So, the queue in the buffer oscillates but the main trend is climbing up which means the packets in the buffer accumulate(see Fig.5) resulting in congestion sooner or later.

Bufferbloat can occur in other places such as network interface card (NIC). Check the output of ifconfig eth0 on the VirtualBox VM. The (maximum) transmit queue length on the network interface reported by ifconfig is 1000 packets(see Fig.7). For the queue sizes of 1000 packets, if we assume the queue drains at 100Mb/s, the maximum time a packet might wait in the queue before it leaves the NIC would be 0.12s(see eq.1)



```
eth0      Link encap:Ethernet  HWaddr 08:00:27:e5:2d:4f
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:112 errors:0 dropped:0 overruns:0 frame:0
          TX packets:87 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:30315 (30.3 KB)  TX bytes:15591 (15.5 KB)
```

Fig. 7: Output of iconfig eth0

The maximum transmit queue length(txqueuelen:1000) on the network is 1000 packets. The MTU(Maximum Transmission Unit) is 1500 bytes.

The condition of a maximum time a packet might wait is when the buffer is filled up, the packet is in the end of the queue and the transmission unit is MTU.

$$Maximum\ Time = \frac{1000 \cdot 1500 \cdot 8}{100 \cdot 10^6} = 0.12s \tag{1}$$

RTTs in different queue sizes.

Based on the simulation we do on the buffer size of 5 packets, 20 packets and 100 packets, observe the plots of the RTT.

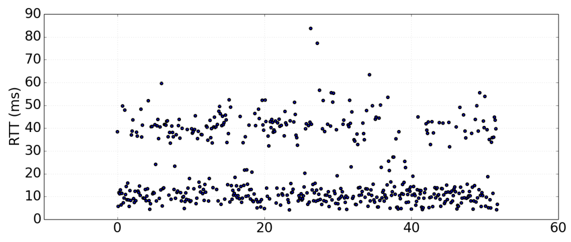


Fig. 8: RTT in 5 packets size buffer

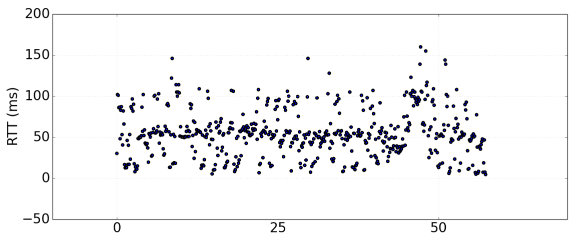


Fig. 9: RTT in 20 packets size buffer

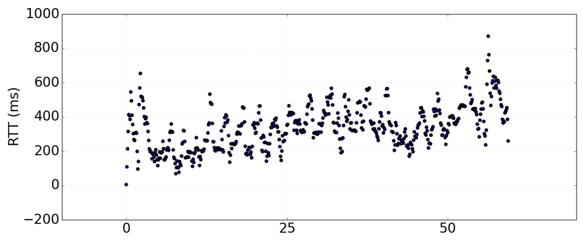


Fig. 10: RTT in 100 packets size buffer

Observe these plots of different buffer sizes, we could find some similarity in their corresponding queue size plots(Fig.1 and Fig.8, Fig.3 and Fig.9, Fig.5 and Fig.10), it seems it has the same upward trend and downward trend. That is because when a queue is generated, there is a linear dependence of RTT on inflight data(data sent but not yet acknowledged).
A symbolic equation to describe the relation between the two

$$RTT = \frac{max\ queue\ size \cdot MTU}{bottleneck\ bandwidth} + round\ trip\ propagation\ delay \tag{2}$$

Two ways to mitigate the bufferbloat problem.

The first way to mitigate the bufferbloat is to be cautious when design the buffer size, minimize the buffer size under the premise that buffer size is enough for the transmitting. Also, when congestion happens, try to detect the fundamental problem instead of adding or enlarging the buffer roughly. Sometimes the reason for the congestion is not the limited buffer size, larger buffer size would just cause another problem—bufferbloat. Get rid of the zero tolerance of the packet loss and understand that the occasional packet loss is essential for avoiding congestion [2]. From where I stand, it is making a balance between latency and packet loss.

The second way to mitigate the bufferbloat is BBR congestion control. Some TCP congestion control interpret packet loss as congestion, the congestion control algorithm we have used in the simulation: TCP reno is also packet loss based. Algorithms like TCP Reno or TCP CUBIC use packet loss as an indication of congestion. However, loss only occurs when the buffers are close to full at the bottleneck (depending on the queue management used) [3]. The congestion is only detected when the bottleneck is already overloaded, leading to large delays. When bottleneck buffers are large, the loss-based congestion control keeps them full, causing bufferbloat. We need to realize that there is a delay between congestion and packet loss. As the Internet is moving more and more data and the memory chips go from KB to GB, the delay between the congestion and packet loss increases, using packet loss as an indication of congestion seems not appropriate because the relationship between packet loss and congestion became tenuous [4].

BBR is an alternative to loss-based congestion control, it is a congestion-based congestion control algorithm developed by Google. BBR cares about the amount of data inflight and BDP(bandwidth-delay product). $BDP = BtlBw * RTprop$($BtlBw$ is bottleneck bandwidth and $RTprop$ is round–trip propagation time). These two parameters bound transport performance. Inflight-BDP excess creates a queue at the bottleneck. Packet loss is when amount of data inflight-BDP excess the buffer capacity. Congestion is when the amount of data inflight excess BDP. Congestion control is some scheme to bound how far can inflight data can excess BDP[4]. BBR is a congestion control algorithm that reacts to actual congestion by measuring two parameters: bottleneck bandwidth, round-trip propagation time continuously since the $BtlBw$ and $RTprop$ vary over the life of the connection[4]. When the amount of data inflight excess BDP, BBR thinks congestion occurs. BBR estimates max $BtlBw$ and min $RTprop$ periodically and use their product as cwnd to use the full bandwidth. Since the problem caused by bufferbloat is due to the large buffer size which makes no packet loss to trigger the TCP congestion control, based on the congestion itself, BBR periodically estimates the available bandwidth and minimal round-trip time (RTT) and adjusts the cwnd, which will fix the delay problem caused by the large buffer to some extent. Even with no packet loss because of the large buffer size, traffic can still perceive congestion and control it in time preventing the occurrence of the bufferbloat with high latency.

References

[1] Pan, R., Natarajan, P., Piglione, C., Prabhu, M. S., Baker, F. J., VerSteeg, B. C., Subramanian, V. (2016). U.S. Patent No. 9,246,829. Washington, DC: U.S. Patent and Trademark Office.

[2] Information on https://www.bufferbloat.net/projects/bloat/wiki/Introduction/.

[3] Scholz, D., Jaeger, B., Schwaighofer, L., Raumer, D., Geyer, F., Carle, G. (2018, May). Towards a deeper understanding of TCP BBR congestion control. In 2018 IFIP Networking Conference (IFIP Networking) and Workshops (pp. 1-9). IEEE.

[4] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., Jacobson, V. (2016). BBR: Congestion-based congestion control. Queue, 14(5), 20-53.