Bufferbloat Problem

CSC458 Computer Networks Programming Assignment 2

Author: Tingfeng Xia, University of Toronto, mailto:tingfeng.xia@mail.utoronto.ca
Date: Saturday 28th November, 2020

Introduction

Bufferbloat refers to the existence of excessively large and frequently full buffers inside a network. [Sivov and Sokolov, 2012]

Most TCP congestion control algorithms rely on packet drops to determine the available bandwidth between two ends of a connection.[Allman and Paxson, 2009, El-Sayed et al., 2011] In general, TCP congestion control algorithms speed up the data transfer (via increasing congestion window size) until packets start to drop, then slow down the transmission rate. Under ideal conditions, we expect an equilibrium speed to be reached after a period of time of adjustments.

In a fast to slow transition hop, bufferbloat can easily occur. Let's consider the internet topology illustrated in the assignment handout (Assignment Topology). TCP will continue to increase the cwnd size since packets sent out are being buffered inside the intermediate router s_0 and with no packet being dropped. It will only decrease the cwnd size when buffer of s_0 is saturated, but that is already too late. In other words, the buffer in the intermediate router has turned the packet drops into an **un-timely** indication of congestion, which is bad since TCP congestion control algorithms rely on timely communication of congestion via packet drops.

Bufferbloat causes increase in queueing delay, and thus causes end users to experience increase in latency, which is the sum of transmission delay, processing delay, and queueing delay. [Sivov and Sokolov, 2012] Bufferbloat also causes jitters and decreases the overall throughput of the network.

Methods

We emulate our Assignment Topology using mininet. Then, we simultaneously perform the following three tasks

- start a long lived TCP flow sending data from h_1 to h_2, and
- send 1 ping per 0.1 second from h_1 to h_2, and
- spawn an web server on h_1, and download the webpage from h_1 once every two seconds.

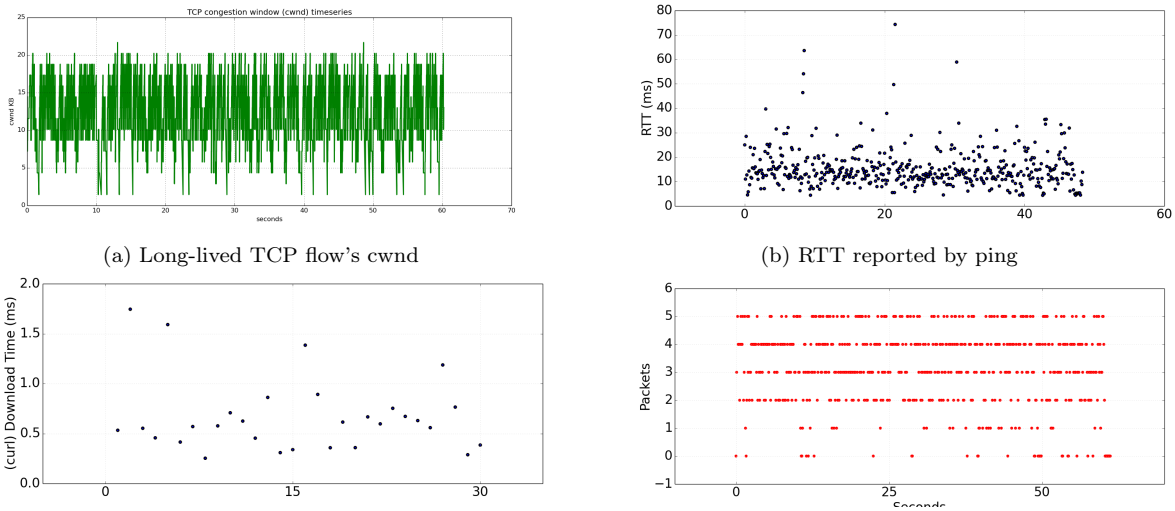
This simulation is repeated for three different queue sizes, $Q = 5/20/100$ pkts. Then, for each max queue size, we plot the time-series of the long-lived TCP flow's cwnd, the RTT reported by ping, the webpage download time, and the queue size at the router s_0.

Results

Table 1 provides an summary of mean and standard deviation of the fetch time in all three experiments with queue size 5, 20, and 100 pkts respectively, while Figures 1, 2 and 3 are plotted time-series for the long lived TCP flow's cwnd, RTT reported by ping, webpage fetch time, and queue size at the router.

Queue Size	5	20	100
Mean(ms)	0.67010	0.75517	1.25273
Stddev(ms)	0.36301	0.20127	0.62726

Table 1: Mean and standard deviation in fetch time for all three experiments.



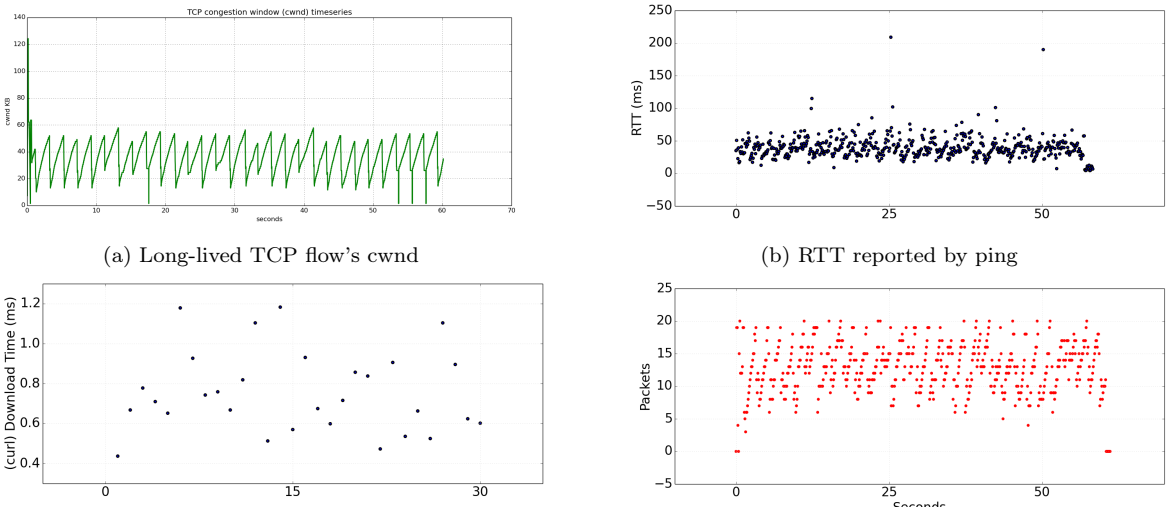
(a) Long-lived TCP flow's cwnd

(b) RTT reported by ping

(c) Webpage download time

(d) Queue size at the router

Figure 1: Long-lived TCP flow's cwnd, RTT reported by ping, webpage download time, and queue size at the router with max buffer size of 5.



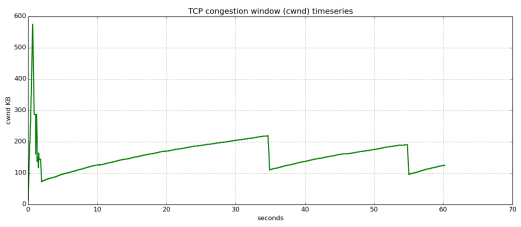
(a) Long-lived TCP flow's cwnd

(b) RTT reported by ping

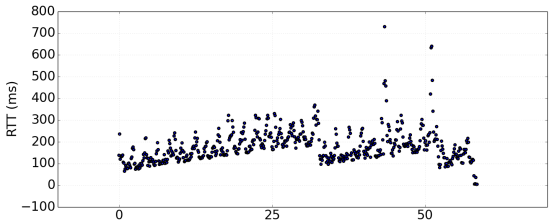
(c) Webpage download time

(d) Queue size at the router

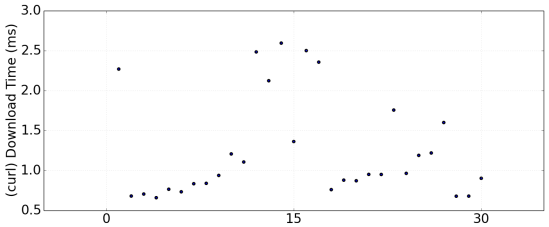
Figure 2: Long-lived TCP flow's cwnd, RTT reported by ping, webpage download time, and queue size at the router with max buffer size of 20.



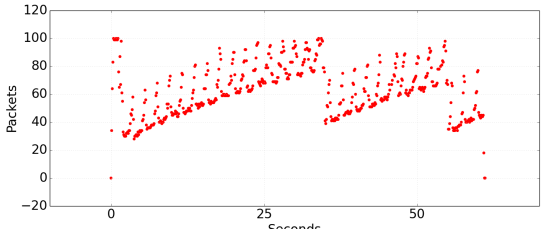
(a) Long-lived TCP flow's cwnd



(b) RTT reported by ping



(c) Webpage download time



(d) Queue size at the router

Figure 3: Long-lived TCP flow's cwnd, RTT reported by ping, webpage download time, and queue size at the router with max buffer size of 100.

Discussion As we can see from Figures 1d, 2d and 3d, buffer tend to be full regardless of our queue size choice. Recall that delay is the sum of transmission delay, processing delay, and queueing delay. When the buffer is larger, we are likely to have a larger queuing delay. In the case of fetching a webpage, when h_1 sends the webpage file to h_2, it is likely that the packet will be queued since it will be travelling from a fast link to a slow one. Thus, the fetch time increases as the buffer size increases.

When the queue size is 5, the buffer is small. This means that we are unlikely to suffer from bufferbloat problem, but we are more likely to drop packets due to buffer out of space. This explains the high standard deviation for fetch time presented in 1 when queue size is 5. On the contrary, when the queue size is set to 100, we have a high fetch time mean. In this case, bufferbloat is happening. Figure 3d illustrates the queue occupancy time-series at the router, and we can see that the buffer is unnecessarily full a lot of the time. When the max buffer is at 20, it is more balanced. Although the buffer is also leaning towards the saturated side, since the buffer size is small, we are seeing in Figures 2b, 2c both faster download time and shorter RTT. Also comparing Figures 1a, 2a, 3a show that as the queue size increase, TCP congestion avoidance algorithm is stepping in less and less.

Ipconfig reports a maximum queue length of `txqueuelen:1000` packets, and a MTU of 1500 bytes. Then, if we have a saturated buffer, we need

$$(1000 \times 1500 \times 8)/(100 \times 10^6) = 0.12\,\text{second} \tag{1}$$

Figure 4 shows[1] the three data points that we have, which are pairs of queue size and the average RTT time taken reported by ping. On the plot, the coordinates are marked as text

[1]This plot is generated with my script `plot_relationship.py`. However, since I can't submit the file on Markus, I will add it to the appendix of this write up. You will need NumPy and Scipy.stats to run this script, I used them for the fitting of linear regression model.

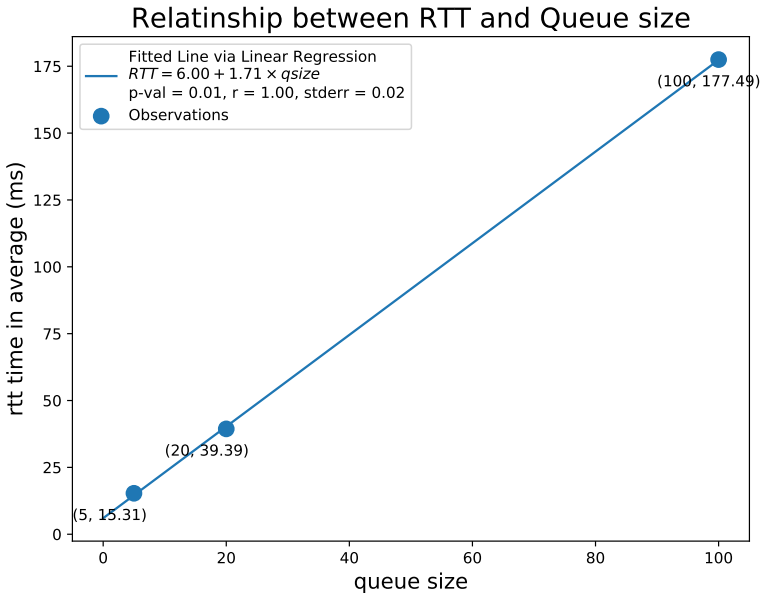


Figure 4: Scatter plot of RTT (in ms) and the max queue size used respectively. This plot is generated with my script `plot_relationship.py`. However, since I can't submit the file on Markus, I will add it to the appendix of this write up. You will need NumPy and Scipy.stats to run this script, I used them for the fitting of linear regression model.

next to the data points. We than fit a linear regression model to our three data points, and then the fitted line is plotted on the same plot. In my case, the fitted line is described algebraically as

$$\text{RTT(ms)} = 1.71 \times \text{queueSize} + 6 \tag{2}$$

Also notice that the $p-value$ is very close to zero, r is 1, and the standard error is also very close to zero. This means that we are statistically confident that the linear relationship is described well with our fitted model.

For the sake of mitigating the bufferbloat problem, we can try

- The probably simplest approach is to decrease the buffer sizes at each hop. This way, when congestion happen, buffers fill up quickly, and then rely on packet drops as a timely indication of congestion. This approach is, however, not optimal. We originally introduced buffers to deal with bursts of packets, and make networking more smooth in general. Reducing buffer size will cost us the ability to deal with bursts in the network.
- Use a delay based congestion avoidance algorithm rather than packet drop based. [Sivov and Sokolov, 2012] In this way, if the delay is too high we know that the packet might have been buffered somewhere, signalling a congestion. We can then control the cwnd size based on this information

Appendix Code

Figure 5 is the code used to produce Figure 4.

Source Code `plot_relationship.py`

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

def get_ping_vals(filename):
        with open(filename, 'r') as f:
            lines = f.readlines()
            ret = []
            num = 0
            for line in lines:
                if 'bytes from' not in line:
                    continue
                try:
                    rtt = line.split(' ')[-2]
                    rtt = rtt.split('=')[1]
                    rtt = float(rtt)
                    ret.append(rtt)
                except:
                    break
            return np.array(ret)

plt.figure(figsize=(8, 6))
# plt.style.use("seaborn")

qs = np.array([5, 20, 100])
filenames = ["./bb-q5/ping.txt", "./bb-q20/ping.txt", "./bb-q100/ping.txt"]
rtts = np.array([np.mean(get_ping_vals(i)) for i in filenames])

slope, intercept, r_value, p_value, std_err = stats.linregress(qs, rtts)
plt.scatter(qs, rtts, s=100, label="Observations")
for x, y in zip(qs, rtts):
text = '(' + str(x) + ', {:.2f})'.format(y)
plt.text(x - 10, y - 10, text)

plt.plot(
np.arange(0, 101),
np.arange(0, 101) * slope + intercept,
label="Fitted Line via Linear Regression \n$RTT$" +
      "$= {:.2f} + {:.2f} \\times qsize$\np-val = {:.2f}, r = {:.2f}, stderr = {:.2f}".format(
          intercept, slope, p_value, r_value, std_err)
)

plt.title("Relatinship between RTT and Queue size", size=18)
plt.xlabel("queue size", size=14)
plt.ylabel("rtt time in average (ms)", size=14)
plt.legend(loc="best")
plt.savefig("./fit.pdf")
```

Figure 5: Appendix Code, `plot_relationship.py`. I was not able to include this file to MarkUs submission, so I am including it here in Appendix. It produces Figure 4. Sorry for the small font size, I had to fit this inside 5 pages limit.

References

[Allman and Paxson, 2009] Allman, M. and Paxson, V. (2009). Tcp congestion control. Rfc, RFC Editor.

[El-Sayed et al., 2011] El-Sayed, A., HAGGAG, S., and EL-FESHAWY, N. (2011). A survey for mechanisms for tcp congestion control. *International Journal of Research and Reviews in Computer Science (IJRRCS)*, 02:676–682.

[Sivov and Sokolov, 2012] Sivov, A. and Sokolov, V. (2012). The bufferbloat problem and tcp: Fighting with congestion and latency.