Computer Networks
CS3205
Assignment 2

TCP Congestion Control

Soumith Basina - CS18B005

Date Submitted: March 29th, 2021

Instructor ..... Prof. Siva Ram Murthy

1 Objective

The objective of this assignment is to emulate the TCP congestion control algorithm, as explained in the problem statement and observe the effects of the different parameters on the size of the congestion window.

2 Introduction

Transmission Control Protocol (TCP) uses a Congestion control algorithm using the Additive Increase/Multiplicative Decrease (AIMD) scheme and also using other methods like Slow Start and Congestion window. This algorithm is the primary basis of Congestion control in the internet. This algorithm mainly operates at the host side.

For each connection, TCP maintains a congestion window (CWND) to limit the total number of unacknowledged packets. TCP uses a mechanism called slow start to increase the congestion window after the connection is initialised or after a timeout. It starts with a window, a small multiple of the maximum segment size (MSS) in size. Although the initial rate is low, the rate of increase is very rapid; for every packet acknowledged, the congestion window increases by 1 MSS so that the congestion window effectively doubles for every round-trip time (RTT).

When the congestion window exceeds the slow-start threshold, ssthresh, the algorithm enters a new state, called congestion avoidance. In congestion avoidance state, as long as non-duplicate ACKs are received the congestion window is additively increased by one MSS every round-trip time.

Loss (indicated by duplicate ACKs) is handled differently on different algorithms. In TCP Tahoe, half of the current CWND is saved as ssthresh and CWND is set to its initial value. In TCP Reno, half of the current CWND is saved as ssthresh and as the new CWND value; while timeout is handled the same way in both cases, congestion window is set to 1 MSS, ssthresh is set to half the old congestion window.

The algorithm that is emulated in the assignment follows these principles but allows for changes in certain parameters.

3 Experimental details

The algorithm in this experiment has 5 adjustable parameters.

- K_i - Initial congestion window
- K_m - Multiplier of congestion window during exponential growth phase
- K_n - Multiplier of congestion window during linear growth phase
- K_f - Multiplier when timeout occurs
- P_s - Probability that a packet times out before receiving the ACK packet

3.1 Experimental/Simulation setup

The five parameters can be passed to the `cw` executable along with the number of segments to be sent and the name of the outfile to generate the outfile. The outfile consists of a list of doubles that represent the congestion window at every update. The outfile can then be used to plot the graph. This process can be automated using the `testcases.sh` bash script, which automatically generates outfiles and graphs of all the combinations of parameters which can be set within the script. Graph of an individual outfile can be viewed using the `plot.ipynb` notebook.

3.2 Entities involved and functions in each entity

`cw.cpp` has the code for the emulation of the algorithm. It takes in parameters through command line and writes the values of congestion window to an outfile of the specified name. For the probabilistic part of deciding whether a packet times out is done by generating a random number `rndnum` $\in [0, 1]$ for each attempt at sending a packet. If the number is less than the given P_s parameter, the packet is deemed to be timed out and congestion window is changed accordingly. The code for the random number generator is given below.

```
// random number generator
random_device rd;
default_random_engine generator(rd());
uniform_real_distribution<double> distribution(0.0, 1.0);
```

`testcases.sh` is a bash script that automatically generates the outfiles and graphs of the combinations of the parameters set as arrays in the script. For every combination, it acquires the outfile from `cw` and passes it to the python script `plot.py` to generate the plot.

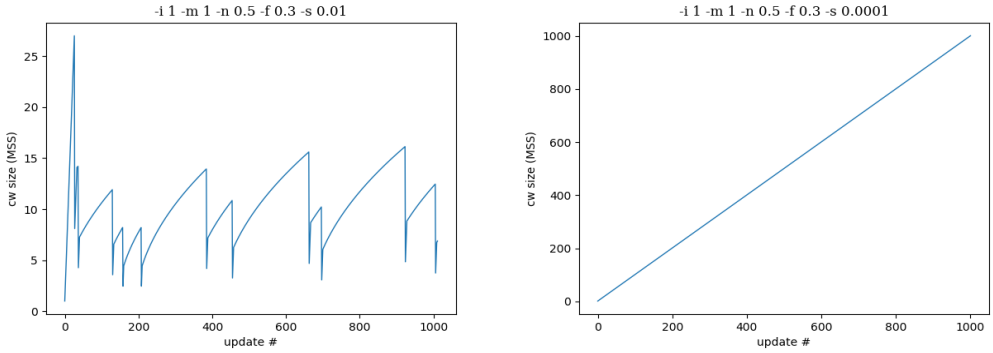
3.3 Additional details

`plot.ipynb` is a Jupyter notebook that can be used to view plot of an outfile. `scriptreplay` can be used on `shell_record` and `file_time` to replay the terminal session of testing the program.

4 Results and Observations

The effects of changes in various parameters on congestion window is detailed in the following subsections.

4.1 Effect of P_s

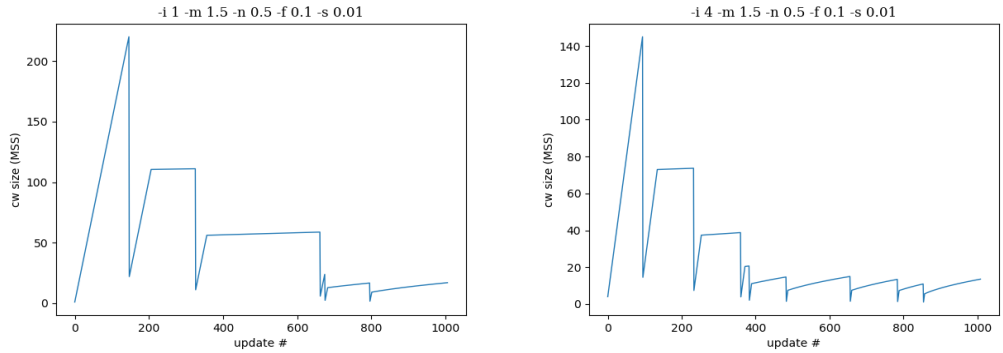


(a) $P_s = 0.01$ (b) $P_s = 0.0001$

Figure 1: Effect of P_s on the congestion window

Keeping all the other parameters same, changing P_s can affect the transfer rate drastically. If the chance of a packet timing out is less, there are less drops in the congestion window size. You will observe in most cases with $T = 1000$, there are rarely any drops for $P_s = 0.0001$. Whereas for $P_s = 0.01$, the congestion window stayed in the range of 5 - 15 most of the time.

4.2 Effect of K_i

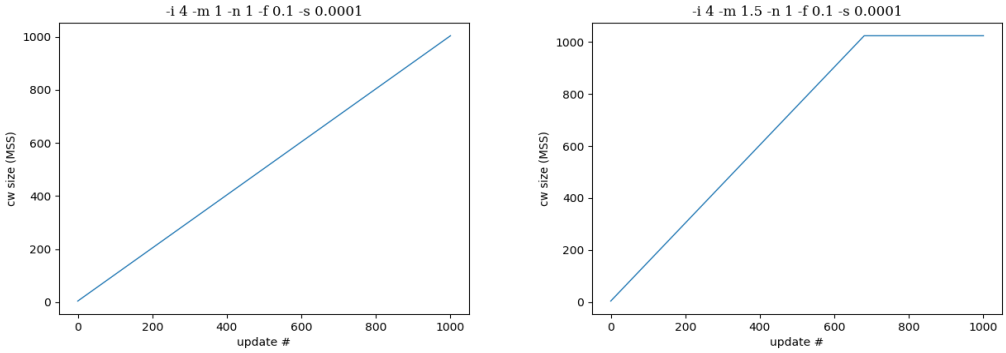


(a) $K_i = 1$ (b) $K_i = 4$

Figure 2: Effect of K_i on the congestion window

There is not much effect due to the change in K_i when T is large as even if the case with larger K_i might have an advantage of having a larger window at the beginning, its effect is limited after a few packet drops as window size is greatly reduced and congestion avoidance sets in resulting in much slower rise in the window size.

4.3 Effect of K_m

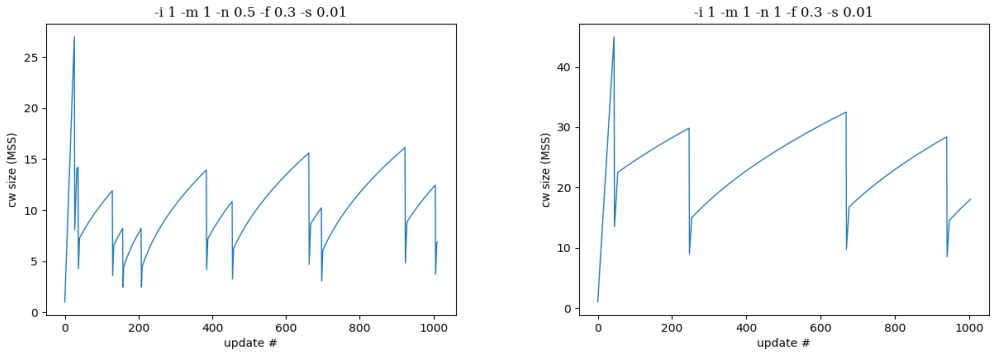


(a) $K_m = 1$ (b) $K_m = 1.5$

Figure 3: Effect of K_m on the congestion window

Having a higher K_m can lead to quicker recovery from the slow start phase and quicker initial rise of congestion window. As we can see in the graph (b), it reaches the receiver window size of 1024 MSS whereas graph (a) doesn't.

4.4 Effect of K_n

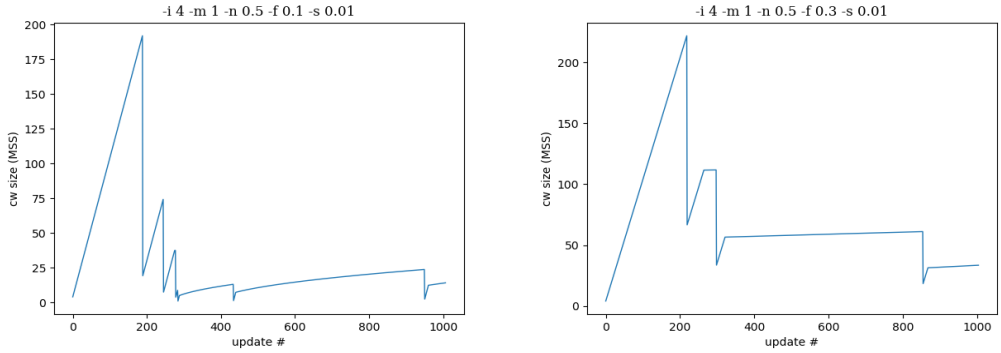


(a) $K_n = 0.5$ (b) $K_i = 1$

Figure 4: Effect of K_n on the congestion window

Having a higher K_n leads to a quicker increase of congestion window in the congestion avoidance phase which can be helpful to increase the rate of packets sent quickly between packet timeouts. Congestion window in graph (b) is increasing quicker than graph (a), therefore maintaining the bigger congestion window.

4.5 Effect of K_f



(a) $K_f = 0.1$ (b) $K_f = 0.3$

Figure 5: Effect of K_f on the congestion window

Having a higher K_f ensures that congestion window decreases less in the event of a timeout. This can help recover quickly from a timeout, maintaining higher transfer rates. As we can see in graph (b), cw does not fall as much as graph (a) in case of a timeout, hence maintaining a bigger window in comparison.

5 Learnings

This assignment has been helpful in

- Learning some insights about the TCP congestion control algorithm and how it helps avoid congestive collapse.
- Understanding how AIMD works and understanding the various effects of changing the parameters in the algorithm
- Learning technical skills such as bash scripting, the script command and python scripting.

6 Additional thoughts

Though this approach does emulate the behaviour of congestion window in real scenarios, as the outcome of whether a packet is going to timeout is immediately known, the effects on the algorithm are simplified as we cannot emulate the behaviour of duplicate ACKs, round trip time (RTT). We cannot observe the effects of such things in this experiment.

7 Conclusion

TCP congestion control is vital in preventing congestive collapse and maintaining a fair connection among all the hosts. By fine-tuning these parameters, we can optimise the algorithm to have a higher transfer rate and for quicker recovery or have a slower recovery in the case of congested networks to avoid drops. We have learned the various effects that different parameters have on the congestion window. This can be helpful in the case of optimising the algorithm.

8 References

- https://witestlab.poly.edu/blog/tcp-congestion-control-basics/
- https://en.wikipedia.org/wiki/TCP_congestion_control