TCP Congestion Control

Report for Assignment 2 as part of the *CS3205 : Introduction to Computer Networks* course

Shreesha G Bhat (CS18B103)
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

I. Aim

The aim of the assignment is to simulate a modified version of the TCP congestion control algorithm (also known as the *Additive Increase, Multiplicative Decrease* (AIMD) algorithm), and learn how the various parameters involved affect the algorithm.

II. Introduction

The TCP congestion control algorithm is a way of reducing congestion in the network by means of a *distributed*, *end-to-end* congestion control scheme. This means that each sender must control the rate at which segments are sent by perceiving the amount of congestion in the network.
The aim of the congestion control algorithm is to utilize the complete available network bandwidth without causing congestion.
This is done by *bandwidth probing* the network by increasing the *congestion window size* ($cwnd$) until a loss/timeout event occurs which is considered to be an indication of network congestion, at which point, $cwnd$ is reset and a threshold ($ssthresh$) is set based on the $cwnd$ value at the point of timeout. Further on, an aggressive (exponential) increase of $cwnd$ is performed until $ssthresh$ and a cautionary (linear) increase of $cwnd$ is performed beyond $ssthresh$ until another timeout event occurs and the cycle continues.
In this experiment, we make use of additional parameters K_i, K_m, K_n, K_f that govern the exact updates for initial value, exponential update, linear update and reset on timeout values for $cwnd$ respectively. P_s is another parameter introduced to simulate timeouts i.e. P_s represents the probability of not receiving the acknowledgement message for a given segment before it's timer expires.

III. Experimental Details

A. Experimental Setup

The following assumptions are considered for the simulation setup.

- The *receiver congestion window size* (RWS) is constant and is of size 1 MB.
- The *Maximum Segment Size* (MSS) for the sender is defined to be 1 KB. Each segment from the sender has a fixed size of 1 MSS.
- We also assume that the sender always has enough data to send.
- *Go-Back-N* scheme is considered, but with individual timeouts and acknowledgement for each segment sent, rather than cumulative acknowledgement.
- Initialization of $cwnd$

$$cwnd := K_i \times MSS$$

  Default value for $K_i = 1$, range of accepted values is $1 \leq K_i \leq 4$.
- Linear Update rule on successfully receiving an acknowledgement message for a previously sent segment

$$cwnd := \textsc{Min}(cwnd + K_n \times MSS \times \frac{MSS}{cwnd}, RWS)$$

  Default value for $K_n = 1$, range of accepted values is $0.5 \leq K_n \leq 2$.
- Exponential Update rule on successfully receiving an acknowledgement message for a previously sent segment

$$cwnd := \textsc{Min}(cwnd + K_m \times MSS, RWS)$$

  Default value for $K_m = 1$, range of accepted values is $0.5 \leq K_m \leq 2$.
- Reset on timeout/dropped acknowledgement message

$$cwnd := \textsc{Max}(1, K_f \times cwnd)$$

  range of accepted values is $0.1 \leq K_f \leq 0.5$
- $0 < P_s < 1$ indicates the probability of not receiving the ACK packet for a given segment before timeout occurs.
- The simulation stops when the number of updates reaches a given parameter value T.

B. Components Involved

The code section involves 2 components. A component to handle the input parameters and the actual simulational component.
The first component takes in the parameters and stores them in a class object for easy access.
The second component performs the actual simulation. The pseudocode for the second component is as given below

```
Initialize $gen$ = RandomNumberGenerator()
Let $cwnd\_values$ be an empty container
distribution = BinomialDistribution($P_s$)
$ssthresh = \infty$
$cwnd = K_i MSS$
while $cwnd\_values.len - 1 \leq T$
    $current\_cwnd = cwnd$
    for $i = 1$ to $(current\_cwnd/MSS)$
        if distribution($gen$) == $True$
            $ssthresh = 0.5 \times cwnd$
            $cwnd = \textsc{Max}(1, K_f cwnd)$
            $cwnd\_values.push(cwnd)$
            break
        if $cwnd \leq ssthresh$
            $cwnd = \textsc{Min}(cwnd + K_m MSS, RWS)$
        else
            $cwnd = \textsc{Min}(cwnd + K_n \frac{MSS^2}{cwnd}, RWS)$
        $cwnd\_values.push(cwnd)$
```

C. Additional Details

- Additional code is added to perform the mean bandwidth computation and to store the results.
- The random seed for the random number generator is fixed. This ensures that for a given P_s value, the timeout occurs at the roughly the same intervals from one run to the other, essentially ensuring that the simulated network conditions are the same from one run to the other. This was done inorder to produce reproducible results and for effective comparison between the plots for the different parameter combinations.

IV. Results & Observations

The simulation was run for 32 combinations of parameters based on the following values

$$\begin{aligned} K_i &= \{1, 4\} \\ K_m &= \{1, 1.5\} \\ K_n &= \{0.5, 1\} \\ K_f &= \{0.1, 0.3\} \\ P_s &= \{0.01, 0.0001\} \end{aligned}$$

For each of the 32 parameter combinations, a graph for *Congestion Window Size in kB* ($cwnd$) vs *Update Number* was plotted for $T = 3000$ updates. The 32 plots are submitted in a separate `Plots.pdf` file.

A. General Observations

For observation purposes, consider the following figure

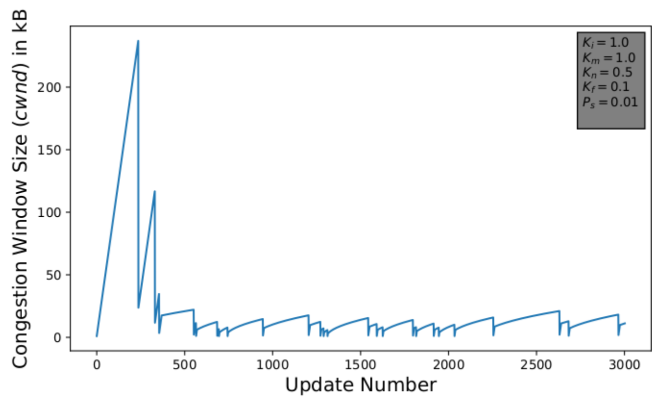

Fig. 1. Plot of *Congestion Window Size in kB* vs *Update Number* for parameters $K_i = 1.0$, $K_m = 1.0$, $K_n = 0.5$, $K_f = 0.1$, $P_s = 0.01$

In Figure 1, initially, according to the slow start phase, $cwnd$ increases exponentially until the first timeout. Note that there is no real threshold for this round as $ssthresh$ is set to ∞. After the first timeout, $ssthresh$ is a finite value, this limits further peaks from happening to a large extent as beyond $ssthresh$, $cwnd$ can only increase according to the linear update rule (as seen by the slightly curved sections of the plots), and usually, a timeout occurs before $cwnd$ can get too big. This characteristic shape is observed in the plots for most parameter values with $P_s = 0.01$. However, when the probability of the timeout is set to $P_s = 0.0001$, the following plot is obtained.

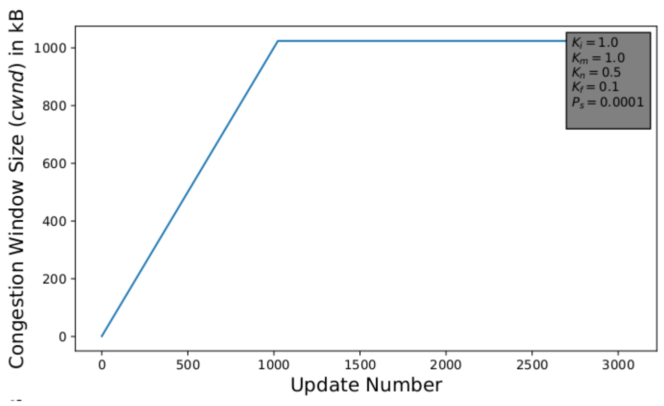

Fig. 2. Plot of *Congestion Window Size in kB* vs *Update Number* for parameters $K_i = 1.0$, $K_m = 1.0$, $K_n = 0.5$, $K_f = 0.1$, $P_s = 0.0001$

In Figure 2, the timeout probability $P_s = 0.0001$ is so low, that the slow start phase never ends. The $cwnd$ value eventually plateaus at *Reciever Window Size* (RWS) of 1 kB. In some cases, when using a different random seed for the distribution generator, we might obtain a timeout before/after plateauing, in which case a slight dip is seen in the plot. But the general behaviour remains similar.

B. Mean Throughput

Note that, at the beginning of each round, the transmission rate or send rate is roughly

$$\text{Throughput} = \frac{cwnd \times MSS}{RTT} \text{bytes/s}$$

Therefore, the mean send rate/throughput approximate can be obtained by averaging the above value over rounds, assuming RTT as a constant.
Using the above estimate, we obtain the following values for mean throughput for each of the 32 parameter combinations.

TABLE I
Table of Mean Throughput for each of the 32 parameter combinations

K_i	K_m	K_n	K_f	P_s	Mean Throughput ($1/RTT$) kB/s
1.0	1.0	0.5	0.1	0.01	10.3335
1.0	1.0	0.5	0.1	0.0001	255.917
1.0	1.0	0.5	0.3	0.01	11.7611
1.0	1.0	0.5	0.3	0.0001	255.917
1.0	1.0	1.0	0.1	0.01	12.954
1.0	1.0	1.0	0.1	0.0001	255.917
1.0	1.0	1.0	0.3	0.01	15.2812
1.0	1.0	1.0	0.3	0.0001	255.917
1.0	1.5	0.5	0.1	0.01	11.6499
1.0	1.5	0.5	0.1	0.0001	321.25
1.0	1.5	0.5	0.3	0.01	13.1959
1.0	1.5	0.5	0.3	0.0001	321.25
1.0	1.5	1.0	0.1	0.01	14.7254
1.0	1.5	1.0	0.1	0.0001	321.25
1.0	1.5	1.0	0.3	0.01	16.6595
1.0	1.5	1.0	0.3	0.0001	321.25
4.0	1.0	0.5	0.1	0.01	10.3957
4.0	1.0	0.5	0.1	0.0001	306.8
4.0	1.0	0.5	0.3	0.01	11.8561
4.0	1.0	0.5	0.3	0.0001	306.8
4.0	1.0	1.0	0.1	0.01	13.0532
4.0	1.0	1.0	0.1	0.0001	306.8
4.0	1.0	1.0	0.3	0.01	15.4375
4.0	1.0	1.0	0.3	0.0001	306.8
4.0	1.5	0.5	0.1	0.01	11.0044
4.0	1.5	0.5	0.1	0.0001	409.0
4.0	1.5	0.5	0.3	0.01	12.4841
4.0	1.5	0.5	0.3	0.0001	409.0
4.0	1.5	1.0	0.1	0.01	13.9437
4.0	1.5	1.0	0.1	0.0001	409.0
4.0	1.5	1.0	0.3	0.01	15.8042
4.0	1.5	1.0	0.3	0.0001	409.0

C. Effect of Parameters

- K_i controls the initial value of $cwnd$. In the plots this is visible as a higher starting point for $cwnd$. From Table *I*, we can see that when keeping other parameters constant, K_i has the effect of increasing the mean send rate.
- K_m controls the rate of the exponential growth phase for $cwmd$. Increasing K_m in general implies quicker growth in the exponential phase. This also means that the linear growth phase will be reached quickly as $cwnd$ will reach $ssthresh$ earlier. This is visible in the plots too, where a higher K_m value keeping the other parameters fixed results in a shorter exponential growth phase and a longer linear growth phase. In Table *I*, we can observe that keeping other parameters constant, increasing K_m leads to an increase in the mean throughput.
- K_n controls the rate of the linear growth phase for $cwnd$. Increasing K_n in general, results in quicker growth in the linear phase. As we have fixed the random seed, this implies that at the point of timeouts, keeping other factors fixed, increasing K_n results in a higher peak at these timeout points. In Table *I*, we can see that keeping other factors fixed, increasing K_n results in increased mean throughput.
- K_f controls the reset point for $cwnd$ on a timeout. Generally, a higher K_f results in a higher starting point for $cwnd$ after a timeout. This effect is visible in the plots. In Table *I*, we can see that keeping other factors fixed, increasing K_f results in increased mean throughput.
- P_s controls the network congestion frequency. A higher P_s implies higher frequency of lost acknowledgement messages and hence higher frequency of timeouts. In the plots, this has a very drastic effect. In case of $P_s = 0.01$, we can see dips in the graphs at regular intervals, which signifies timeouts happening at regular intervals. However, in case of $P_s = 0.0001$ which indicates a very rare chance of timeouts, there are no dips. On changing the random seed, at times, 1/2 dips were obtained in the graph. But in general, timeout being a very rare event, the graph stays in the initial slow start phase and plateaus after reaching a value of RWS. This naturally leads to a very high throughput, which is also visible in Table *I*, i.e. keeping other factors constant, a lower P_s results in significantly higher throughput.

V. Possible Improvements

- The TCP Congestion Control algorithm described above is a distributed algorithm, in the sense that each sender perceives the amount of congestion in the network and controls it's sending rate based on this. The sender as such does not provide any explicit feedback about whether congestion is occuring/not.
  In reality, we could expect that the congestion frequency is not really constant (as assumed in this assignment) but *depends on the sending rate of the sender*. Intuitively a higher sending rate implies a slightly higher chance of timeouts (dropping of acknowledgements) occurring and a lower sending rate implies a slightly lower chance of timeouts occurring. ***Therefore, one possible improvement is to make the distribution for lost acknowledgements a function of the current congestion window size***. This might result in a more accurate simulation of the TCP congestion control algorithm.
- Another alternative is to simulate many senders at a time, each controlling their congestion window size based on the network conditions. In such a case, we could define the distribution of dropped acknowledgements as a function of the congestion window size of each of the many senders involved in the simulation. This could potentially lead to even more accurate simulation of the TCP congestion control algorithm and give us a better idea about how the various parameters affect the performance.

VI. Learning Outcomes & Conclusion

- Learnt and understood the TCP congestion control (AIMD) algorithm.
- Built a realistic simulation of a modified version of the TCP congestion control algorithm.
- Explored how the various parameters introduced affect the behaviour and performance of the algorithm through plotting and other analyses.

References

[1] Kurose, J. F. Ross, K. W. (2016), Computer Networking: A Top-Down Approach , Pearson , Boston, MA .

[2] Slides provided by the TAs during class hours.

[3] CS3205 Lecture Slides