CS3205: Introduction to Computer Networks

Report: Assignment 2
TCP Congestion Control

Anirudh S
CS17B003

---

1 Problem Statement

The aim of this assignment is to emulate the TCP congestion control algorithm, *Additive Increase Multiplicative Decrease (AIMD)* [Section 2.4]. The Go-Back-N protocol [Section 2.3] is used with individual acknowledgement and timer for each data segment. It is assumed that the sender always has data to send with each message segment size (MSS) as 1 KB. The receievr window size(RWS) is set to 1 MB, i.e, it can hold up to 1024 segments. The update steps in AIMD are modified slightly for the assignment. The details are described in Section 2.5.

2 Introduction

2.1 TCP

Transmission Control Protocol (TCP) is an application layer protocol which ensures reliable bi-directional communication between sender and receiver. TCP is connection oriented, i.e, a connection must be established between sender and the receiver before the data transmission can occur. The connection is established through **three-way handshake** as shown in figure 1. Once the connection is established, the data is sent and received as a stream of bytes. After the communication process is completed, the TCP connection is closed using **four-way handshake** as shown in figure 1

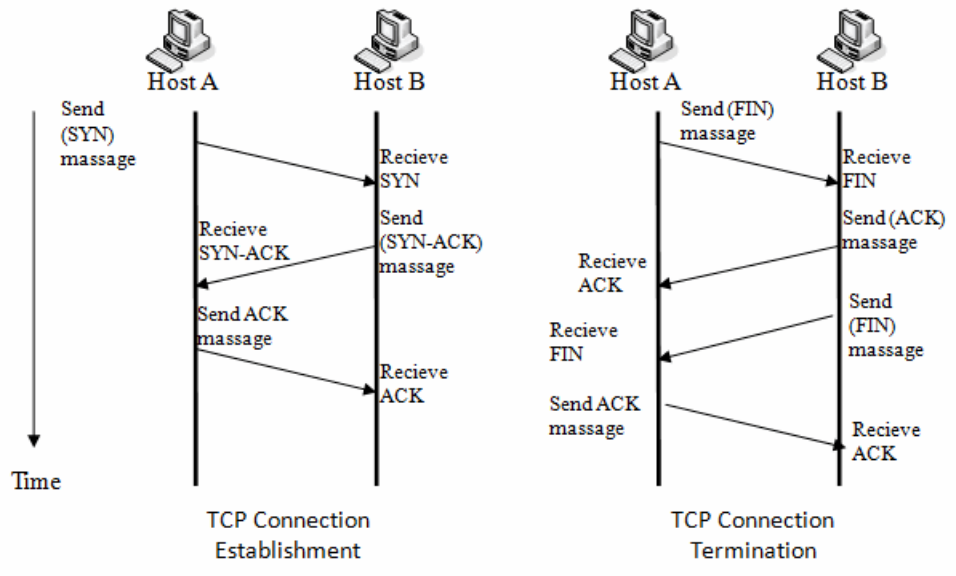


Figure 1: TCP three-way and four-way handshakes[1]

2.2 TCP error control

To ensure reliability TCP employs the following methods.

- **Checksum:** A checksum is included in the header of every TCP segment to ensure that the data transmitted is not corrupted.
- **Acknowledgement:** The receiver sends an acknowledgement back to the sender upon successful reception once per packet (or once every few packets). This ensures that the packets have reached the destination instead of being lost in the network.
- **Retransmission:** Retransmission occurs happens during two scenarios.
  1. **Timeout:** For every packet (or few packets) transmitted by the sender, a timer is maintained. If the acknowledgement does not arrive before the timer expires, it is assumed that the packet is lost and hence is retransmitted.
  2. **Three duplicate acknowledgements:** Due to some error in the network, one or more packets may get lost in between and the later packets are delivered to the receiver. To indicate this out-of-order reception to the sender, the acknowledgement message of the last successful reception is re-sent to the sender. In such a case, if the sender receives the same acknowledgement message three times in a row, it retransmits all the packets, starting from the one after the last successfully received packet.

2.3 Go-back-N (GBN) protocol

Go-back-N is one of the sliding window protocols used for efficient and reliable packet transmission in TCP. In this protocol, the sender can send up to **N** packets before it must wait for an acknowledgement from the receiver. If the first packet is acknowledged, the window slides and the new packet in the window is transmitted and so on. The receiver accepts packets in-order only. In case, a retransmission scenario occurs (as explained in 2.2), all packets in the current window are retransmitted. The protocol is summarised in figure 2.

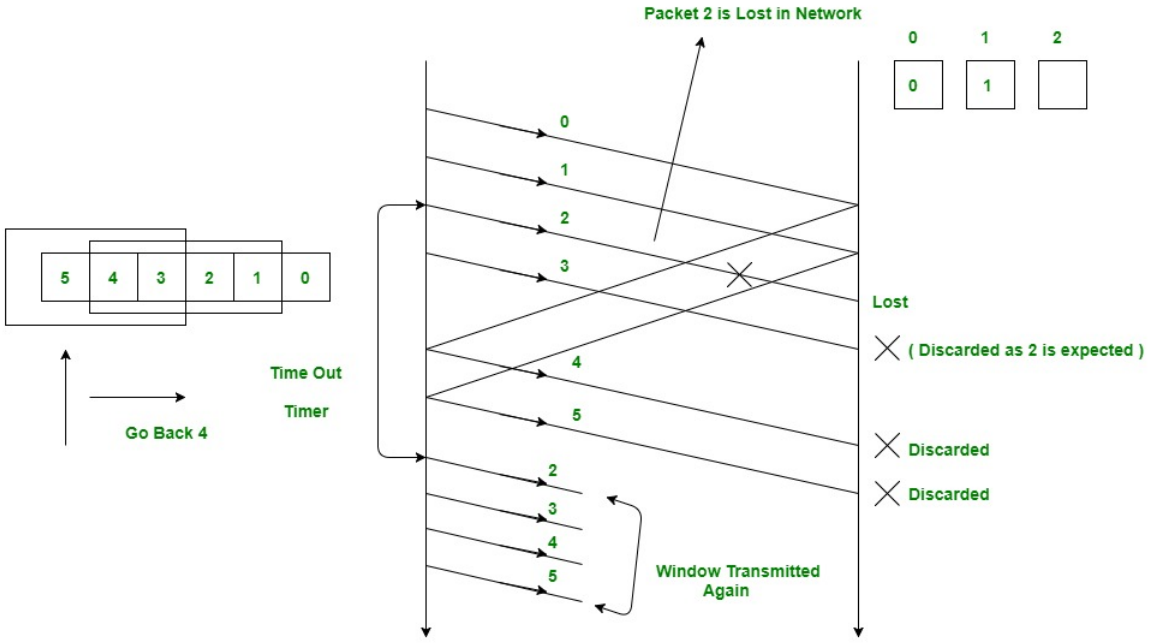


Figure 2: Go-back-N protocol[2]

2.4 TCP congestion control

Congestion is the situation in which the network is receiving more data packets than its handling capacity, thus leading to increased network delays. This causes a reduction in performance and increases the chance of a packet being timed out or lost, hence leading to multiple retransmissions. In order to avoid congestion, TCP employs **Additive Increase Multiplicative Decrease (AIMD)** algorithm. AIMD comprises of three phases.

1. **Slow start/Exponential growth phase:** The congestion window is incremented by 1 every time a packet is acknowledged. So in every round trip time (RTT), the congestion window doubles. This goes on till a threshold value is reached.

2. **Congestion avoidance/Linear growth phase:** Once the threshold value is reached, the command window is incremented by 1 every RTT. Thus, the command window grows linearly across RTTs.

3. **Congestion detection phase:** Congestion is detected through packet retransmission. Depending on the type of retransmission, different steps are taken.

   (a) **Timeout:** Timeout implies that the congestion in the network is high. So we perform the following steps (Fast retransmission):

      i. New threshold is set to half of current window size
      ii. Current window size is set to the initial value
      iii. Go to slow start phase

   (b) **Three duplicate acknowledgements:** This implies that the possibility of congestion is low and hence we take the following measures (Fast recovery):

      i. Current window size is set to the threshold value
      ii. Go to congestion avoidance phase

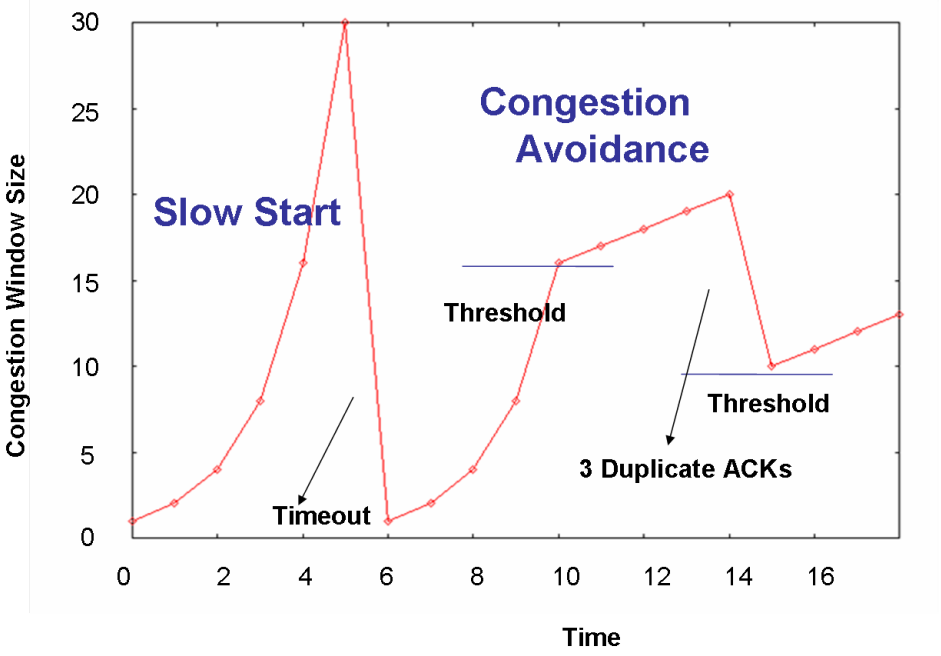


Figure 3: Additive Increase Multiplicative Decrease (AIMD)[3]

2.5 Assignment specific modifications

For the context of this assignment, few modifications have been introduced to the original AIMD algorithm.

- Three duplicate acknowledgements are ignored and only timeout retransmissions are considered.
- After congestion detection, the threshold is always set to 50% of the congestion window size. Hence, after the congestion detection phase, the algorithm restarts from slow start phase.
- New values are defined for updating command window size according to the different scenarios.

  1. **Initial congestion window size**, K_i, $1 \leq K_i \leq 4$ (default value 1). Hence the initial CW:
  $$CW_{new} = K_i * MSS$$
  2. **Exponential phase multiplier**, K_m, $0.5 \leq K_m \leq 2$ (default value 1). Hence during the exponential phase when an acknowledgement is received:
  $$CW_{new} = min(CW_{old} + K_m * MSS, RWS)$$
  3. **Linear phase multiplier**, K_n, $0.5 \leq K_n \leq 2$ (default value 1). Hence during the linear phase when an acknowledgement is received:
  $$CW_{new} = min(CW_{old} + K_n * \frac{MSS}{CW_{old}} * MSS, RWS)$$
  4. **Timeout multiplier**, K_f, $0.1 \leq K_m \leq 0.5$. Hence whenever a timeout occurs:
  $$CW_{new} = max(1, K_f * CW_{old})$$
  5. **Failure probability** P_s, $0 \leq P_s \leq 1$, denotes the probability that a timeout occurs for a particular packet.

3 Implementation details

3.1 Simulator

The simulator consists of two parts: **simulate method**, which performs the simulation and **Command line interface**, which accepts the different parameters required for the simulation as command line arguments.

3.1.1 Simulate method

The simulate method accepts the different parameters as arguments, runs the modified AIMD algorithm and outputs the command window size at each stage to the output file provided.

The number of updates is counted and ensured that when it reaches the required number (received from the user), he simulation stops. Until then, the following happens continuously: segments from the current window are 'sent' (no actual sending takes place; but logically it can be considered to have happened). For each segment a random number is generated between 0 and 1. If this number is less than P_s, then the segment is considered to have timed out. In this case, the threshold and congestion window sizes are updated accordingly, and the process of sending restarts with the sending of segments in slow start phase. Otherwise, based on which phase(slow start or congestion avoidance) the algorithm is in, updates to congestion window size are made accordingly.

3.1.2 Command line interface

The simulation accepts the following command line arguments [argument: flag, type]

- K_i: `-i`, `double`
- K_m: `-m`, `double`
- K_n: `-n`, `double`
- K_f: `-f`, `double`
- P_s: `-s`, `double`
- Number of updates: `-T`, `integer`
- Output file: `-o`, `string` (file name)

The command to invoke the simulation program would appear as

```
$ ./cw -i <double> -m <double> -n <double> -f <double> -s <double> -T <int>
-o <string>
```

3.2 Script

The script runs the above simulator and plots a graph on based on the output. It is written `python3` and uses the `matplotlib` library for plotting graphs. The script can be run for particular parameter values using the same flags described in section 3.1.2. The command would appear as

```
$ python3 script.py -i <double> -m <double> -n <double> -f <double> -s <double>
-T <int> -o <string>
```

Apart from that, the script accepts an additional '-a' flag to run the simulator for all the combinations mentioned in section 4. The outputs of the simulation are stored in `outputs` folder and the graphs in `graphs` folder. Also, the values of T used are: for $P_s = 0.01, T = 1000$ and for $P_s = 0.0001, T = 20000$.

```
$ python3 script.py -a
```

4 Experiments

The TCP congestion control is simulated on the following parameter combinations.

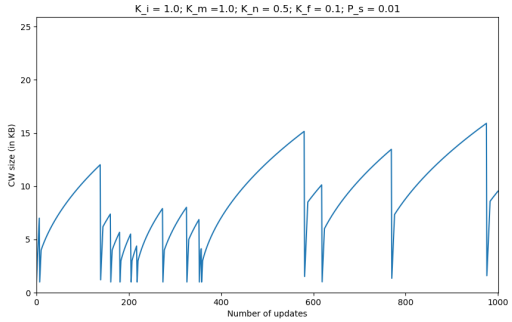
- $K_i \in \{1, 4\}$
- $K_m \in \{1, 1.5\}$
- $K_n \in \{0.5, 1\}$
- $K_f \in \{0.1, 0.3\}$
- $P_s \in \{0.01, 0.0001\}$

The congestion value size after each update is printed to an output file. For each simulation, a graph of congestion window size vs update number is plotted.

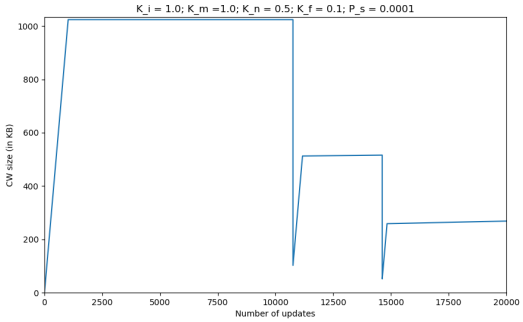
5 Results and Observations

5.1 Graphs

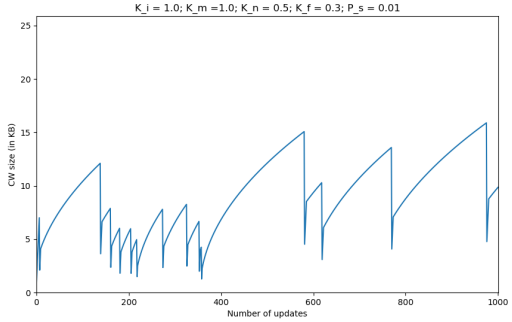
The graphs obtained by the above experiments are represented below followed by observations on them in section 5.2.



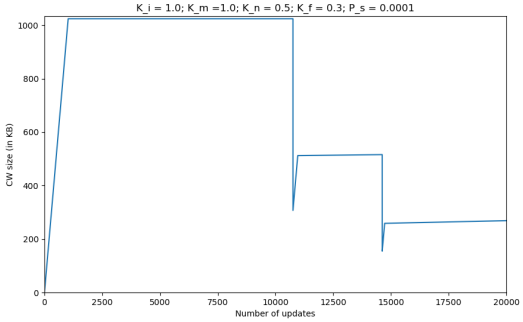
(a) $K_i = 1.0, K_m = 1.0, K_n = 0.5, K_f = 0.1, P_s = 0.01$



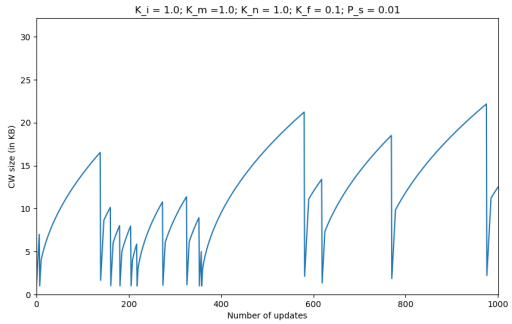
(b) $K_i = 1.0, K_m = 1.0, K_n = 0.5, K_f = 0.1, P_s = 0.0001$



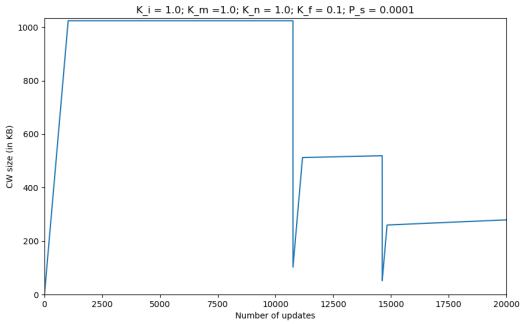
(c) $K_i = 1.0, K_m = 1.0, K_n = 0.5, K_f = 0.3, P_s = 0.01$



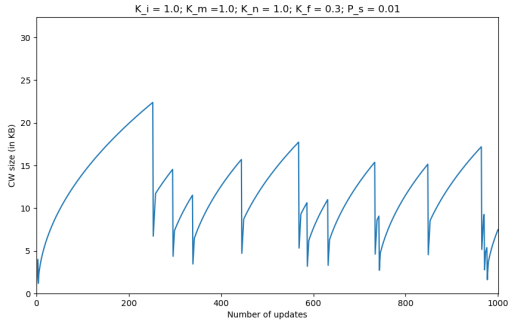
(d) $K_i = 1.0, K_m = 1.0, K_n = 0.5, K_f = 0.3, P_s = 0.0001$



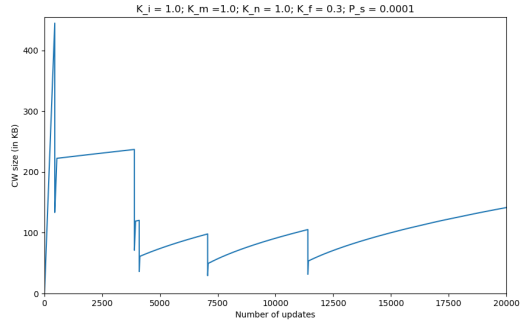
(e) $K_i = 1.0, K_m = 1.0, K_n = 1.0, K_f = 0.1, P_s = 0.01$



(f) $K_i = 1.0, K_m = 1.0, K_n = 1.0, K_f = 0.1, P_s = 0.0001$

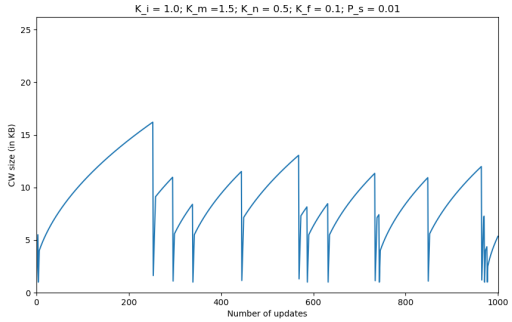


(g) $K_i = 1.0, K_m = 1.0, K_n = 1.0, K_f = 0.3, P_s = 0.01$

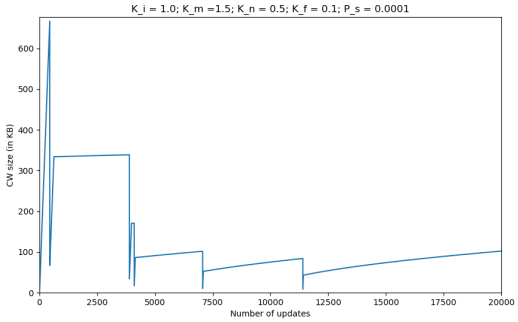


(h) $K_i = 1.0, K_m = 1.0, K_n = 1.0, K_f = 0.3, P_s = 0.0001$

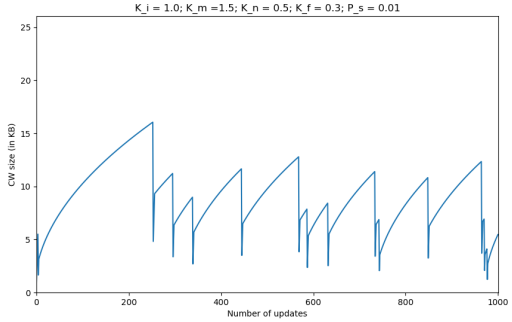
Figure 4: Graphs obtained from experiments: Set 1



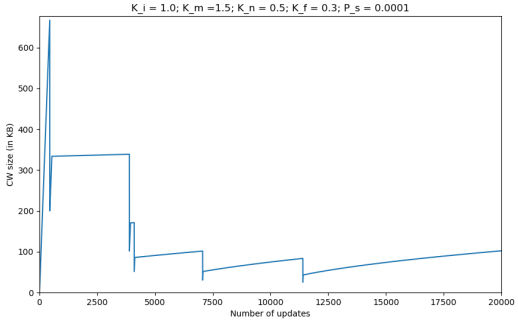
(a) $K_i = 1.0, K_m = 1.5, K_n = 0.5, K_f = 0.1, P_s = 0.01$



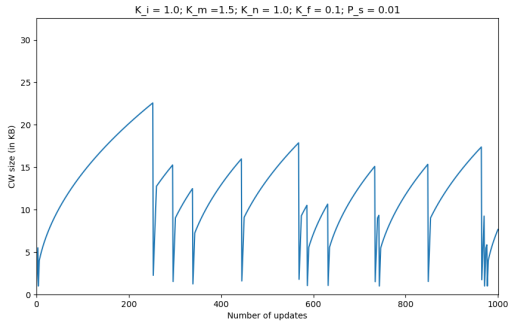
(b) $K_i = 1.0, K_m = 1.5, K_n = 0.5, K_f = 0.1, P_s = 0.0001$



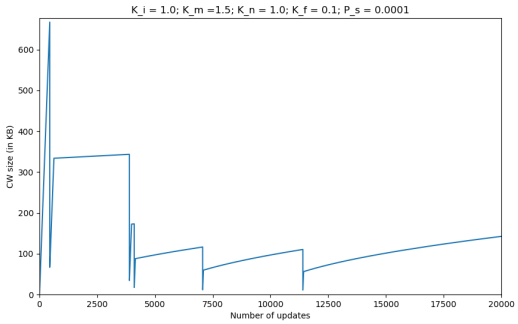
(c) $K_i = 1.0, K_m = 1.5, K_n = 0.5, K_f = 0.3, P_s = 0.01$



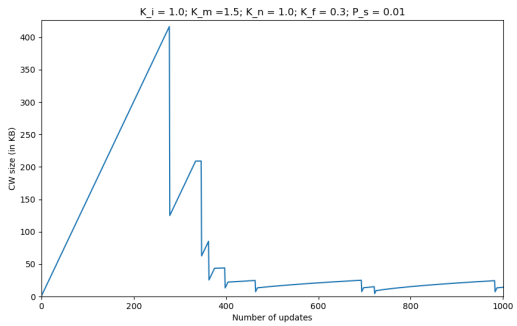
(d) $K_i = 1.0, K_m = 1.5, K_n = 0.5, K_f = 0.3, P_s = 0.0001$



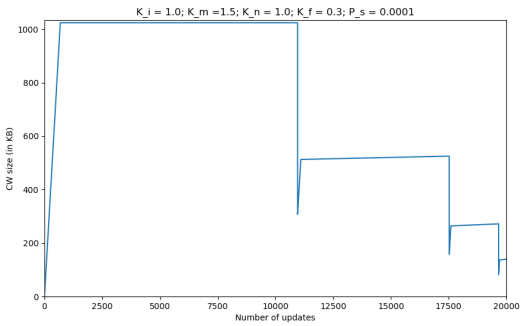
(e) $K_i = 1.0, K_m = 1.5, K_n = 1.0, K_f = 0.1, P_s = 0.01$



(f) $K_i = 1.0, K_m = 1.5, K_n = 1.0, K_f = 0.1, P_s = 0.0001$

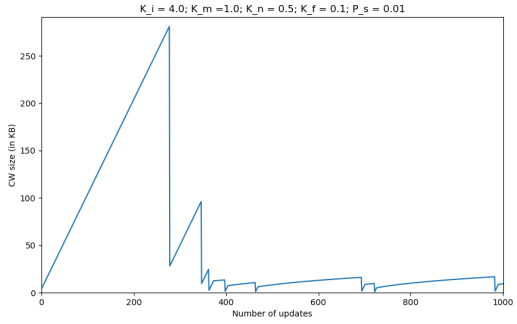


(g) $K_i = 1.0, K_m = 1.5, K_n = 1.0, K_f = 0.3, P_s = 0.01$

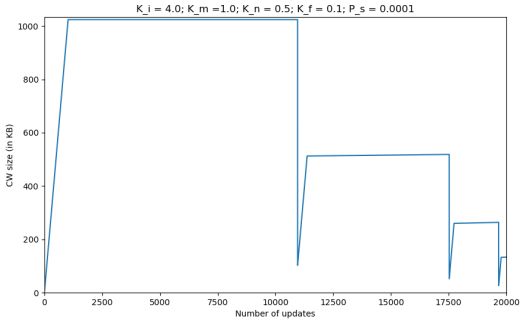


(h) $K_i = 1.0, K_m = 1.5, K_n = 1.0, K_f = 0.3, P_s = 0.0001$

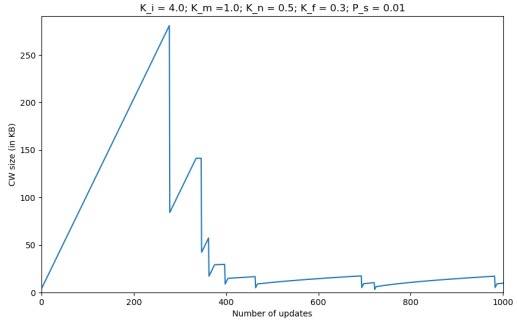
Figure 5: Graphs obtained from experiments: Set 2



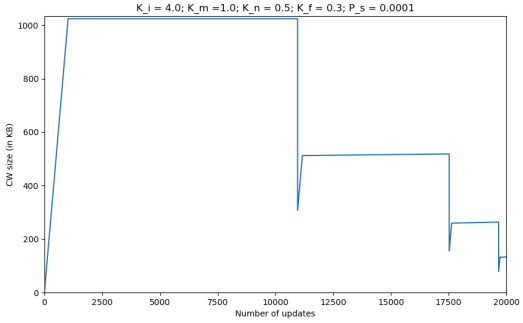
(a) $K_i = 4.0, K_m = 1.0, K_n = 0.5, K_f = 0.1, P_s = 0.01$



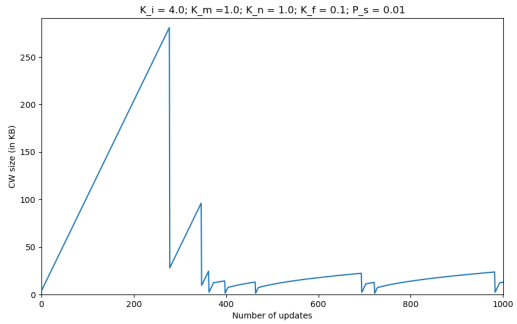
(b) $K_i = 4.0, K_m = 1.0, K_n = 0.5, K_f = 0.1, P_s = 0.0001$



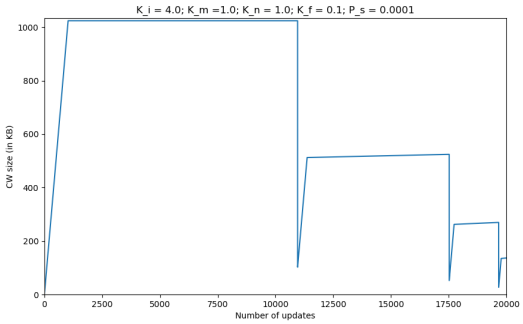
(c) $K_i = 4.0, K_m = 1.0, K_n = 0.5, K_f = 0.3, P_s = 0.01$



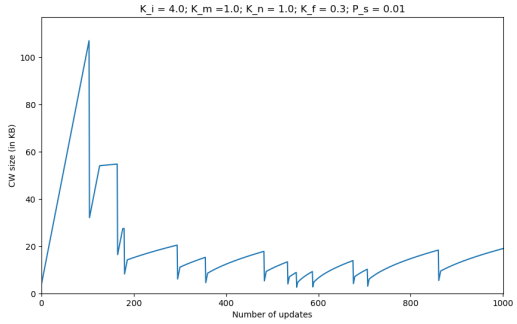
(d) $K_i = 4.0, K_m = 1.0, K_n = 0.5, K_f = 0.3, P_s = 0.0001$



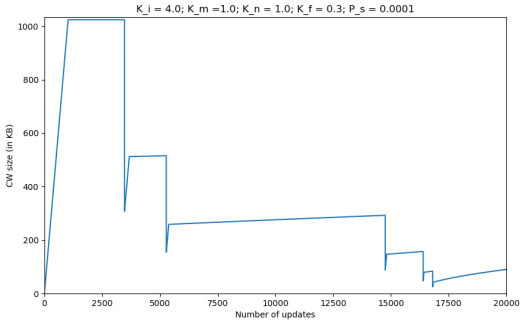
(e) $K_i = 4.0, K_m = 1.0, K_n = 1.0, K_f = 0.1, P_s = 0.01$



(f) $K_i = 4.0, K_m = 1.0, K_n = 1.0, K_f = 0.1, P_s = 0.0001$

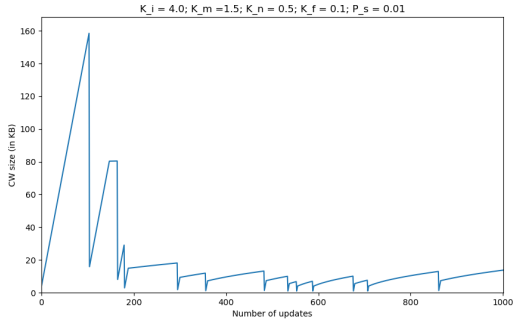


(g) $K_i = 4.0, K_m = 1.0, K_n = 1.0, K_f = 0.3, P_s = 0.01$

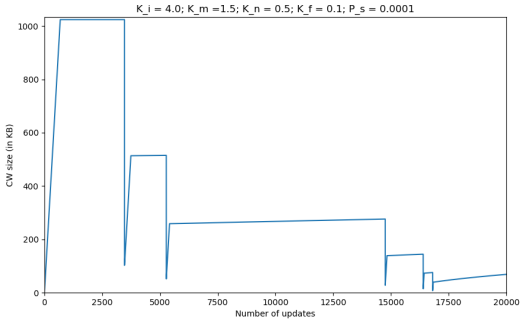


(h) $K_i = 4.0, K_m = 1.0, K_n = 1.0, K_f = 0.3, P_s = 0.0001$

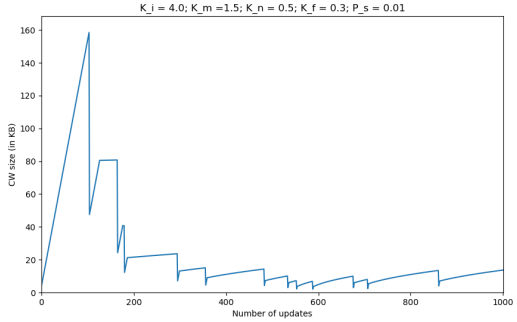
Figure 6: Graphs obtained from experiments: Set 3



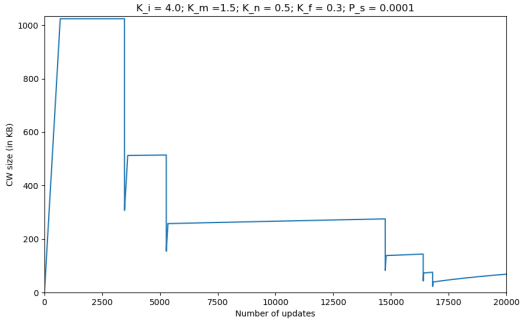
(a) $K_i = 4.0, K_m = 1.5, K_n = 0.5, K_f = 0.1, P_s = 0.01$



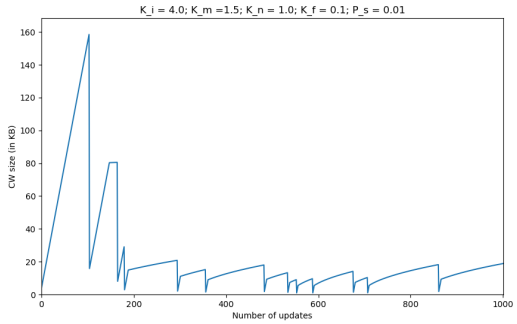
(b) $K_i = 4.0, K_m = 1.5, K_n = 0.5, K_f = 0.1, P_s = 0.0001$



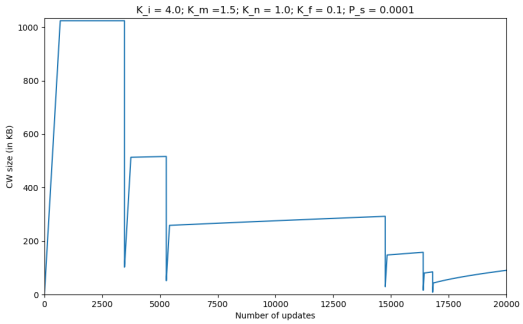
(c) $K_i = 4.0, K_m = 1.5, K_n = 0.5, K_f = 0.3, P_s = 0.01$



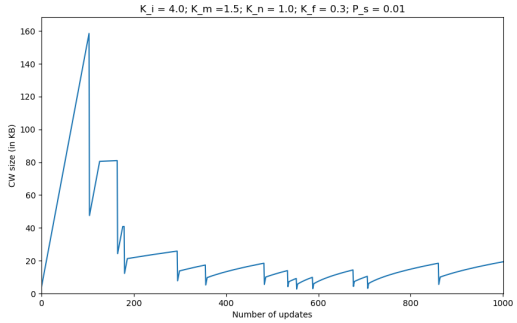
(d) $K_i = 4.0, K_m = 1.5, K_n = 0.5, K_f = 0.3, P_s = 0.0001$



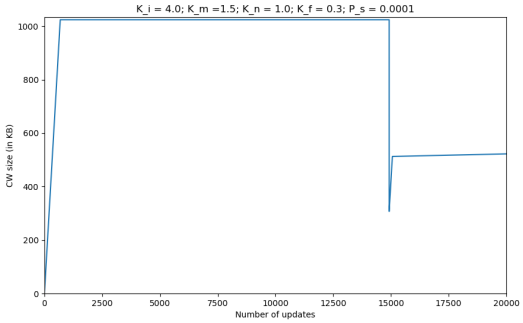
(e) $K_i = 4.0, K_m = 1.5, K_n = 1.0, K_f = 0.1, P_s = 0.01$



(f) $K_i = 4.0, K_m = 1.5, K_n = 1.0, K_f = 0.1, P_s = 0.0001$



(g) $K_i = 4.0, K_m = 1.5, K_n = 1.0, K_f = 0.3, P_s = 0.01$



(h) $K_i = 4.0, K_m = 1.5, K_n = 1.0, K_f = 0.3, P_s = 0.0001$

Figure 7: Graphs obtained from experiments: Set 4

5.2 Observations

The different parameters utilised for the simulator have varying degrees of impact on the outcome graph. We will examine the influence of each parameter separately and then finally we make additional observation on the obtained results.

1. **Failure probability P_s:**
   - This parameter has the highest influence on the congestion window size in this experiment.
   - Lower failure probability implies higher number of acknowledgements. This means that network congestion is directly proportional to P_s. This is results in lesser number of timeouts for $P_s = 0.0001$ as compared to $P_s = 0.01$ in the same interval. It can observed from the graphs that hardly any timeouts occur in the first 1000 updates for the lower probability case.
   - As can be observed, for $P_s = 0.0001$, 20000 updates had to be performed to get some significant number of timeouts, whereas for $P_s = 0.01$, many timeouts occur just in the first 1000 updates.
   - In many cases when $P_s = 0.0001$, the congestion window size reached the saturation value of 1024 MSS. This is mainly attributed to significantly lower timeouts due to a non-congested network.
2. **Timeout multiplier K_f:**
   - This parameter represents to what size the congestion window is reduced to after a timeout. In a sense, lower value of K_f is an indicator of our reservation about the network traffic, assuming that it might be very high and hence starting from a much lower value.
   - It can be observed from the graphs with varying K_f with all other parameters kept constant (such as figures 4b and 4d), that the one with $K_f = 0.1$ has a higher drop in congestion window size after timeout as compared to the one with $K_f = 0.3$. This is in accordance with our expectation of how this parameter is supposed to influence the outcome.
3. **Linear phase multiplier K_n:**
   - This parameter influences the growth of congestion window in congestion avoidance phase. Higher the value of K_n implies faster growth in the congestion window size. Hence a higher value is reflective of confidence in comparatively lower network traffic.
   - As can be seen from plot 7a where $K_n = 0.5$ and plot 7e where $K_n = 1.0$ with all other parameters constant, the linear growth is comparatively steeper in the latter case.
4. **Exponential phase multiplier K_m:**
   - The congestion window size growth in slow start phase is influenced by this parameter. The reasoning and rationale behind K_m are the same as those behind K_n, as they perform similar functions but in different phases of the AIMD algorithm.
   - We observe a steeper rise when the K_m is higher. In plot 7c where $K_m = 1.5$, the congestion window size rises to approximately 160 in around 100 updates. Comparatively, in plot 6c where all other parameters are the same but $K_m = 1.0$, we observe that it takes around 150 iterations to reach a congestion window size of 160.

5. **Initial congestion window size K_i:**
   - This parameter only determines the starting size of the congestion window and does not participate in any way during the simulation.
   - Hence, as expected, K_i has no effect on the graph apart from the Y-axis offset in the beginning.
6. **Additional observations:**
   - Random numbers between 0 and 1 were generated to check against the failure probability. The random number generator was seeded using system time and it was observed that, when the simulator was run on all the parameter configurations, same seed was used across few runs. This resulted in the same sequence of random numbers generated.
   - As a consequence, when the P_s was the same, as in the case of plots 5a, 5c and 5e with $P_s = 0.01$, timeout occurs at the same place across all the three plots.
   - Due to this implementation detail, plots across few runs look similar with small changes due to the influence of K_m, K_n and K_f as already discussed above.

6 Conclusion

TCP is omnipresent in the current-day networks due to the widespread incorporation of the TCP/IP protocol suite. Due to more and more being connected to the internet, network traffic is increasing. Thus, congestion control is of central importance to provide reliable performance despite this increasing traffic. The exercise of implementing a simulator for TCP congestion control for this assignment provided in-depth understanding of the Additive Increase Multiplicative Decrease (AIMD) algorithm. Due to prominence of TCP congestion control in the current day computer networks, this assignment proved to be an excellent platform to augment my skill-set with this relevant knowledge. In conclusion, implementing the simulator not only provided me with interesting insights of the protocol working, but also resulted in an immense amount of learning and experience gained in these critical computer networks concepts.

References

[1] Manual:Connection oriented communication (TCP/IP)
URL: http://www.cablefree.net/support/radio/software/Manual:Connection_oriented_communication_(TCP/IP)

[2] Sliding Window Protocol — Set 2 (Receiver Side)
URL: https://www.geeksforgeeks.org/sliding-window-protocol-set-2-receiver-side/

[3] DCCP URL: http://www.cs.nccu.edu.tw/ lien/TALK/DCCP/hardcopy.htm