COL334 Assignment-3 Report

Amaiya Singhal 2021CS50598
Om Dehlan 2021CS10076

Milestone 3

Approach and Methodology

- We have implemented the **AIMD** (Additive Increase Multiplicative Decrease) policy for congestion control.
- **EWMA** (Exponentially Weighted Moving Average) has been used to calculate the appropriate interval between 2 bursts dynamically.
- We have used 2 threads for this, one thread is for sending the requests and another to simultaneously receive the responses from the Vayu server.
- Some modifications were made to the general AIMD policy to adapt to a **Variable** Rate Leaky Bucket Server, to give the best time while minimising the penalty.
- The specifics of our approach are described below:
- **Sending and Receiving Packets:**
  - There are 2 separate threads for sending requests and receiving responses.
  - On one thread, requests are send sequentially to the server, incrementing the offset each time by 1448 bytes (Packet Size).
  - On the other thread, the responses are being received from the server and the data received is stored in a dictionary with the offset as the key.
  - Requests are sent continuously until all the packets are received, i.e. the size of the dictionary equals the expected number of packets.
- **Variation in Burst Size:**
  - The burst size is decreased to half of its value when there is more than or equal to **20% drop** in the packets sent.
  - This is done because sometimes a packet is dropped but it does not mean that the rate is too high, hence the 20% value is used as tolerance.
  - Since this is a variable rate server, if we do not include this tolerance then the variations in rate which lead to varying number of packet drops would cause the AIMD policy to half the burst size very frequently and thus decreasing the time.
  - To avoid this and decrease the total time taken, a tolerance is introduced. This does not increase the penalty by any significant amount, yet improves the time taken.
  - In the case that the loss is less than 20% the burst size is incremented by 1 as per standard AIMD.
- **Gap between the Bursts:**
  - We have used the **EWMA** policy for finding the time interval between 2 consecutive bursts dynamically.

- The policy used is a modified version of the standard EWMA policy. The specifics of the policy are:
  - If we receive a response for all the requests sent in the burst then we decrease the RTT slightly because we can increase the rate in this situation.
  - If the burst size was 1 then we do not modify the RTT.
  - If the burst size was 2 then we apply the EWMA policy with $\alpha = 0.1$ and if it is greater than 2 then $\alpha = 0.2$ is used.
  - The value of α is kept small so as to slowly adjust the rate and not increase or decrease it very quickly as this can lead to either squishing or sending requests too slow.
  - If we do not receive a response for any requests in a time interval of 1 RTT then we increase the RTT by a factor of **1.01** to adjust for the fact that we might be sending packets too fast.
  - This dynamic policy ensures that the rate is **dynamically adjusted** after every burst which works very well with a variable rate server.

- **Ensuring Correctness:**
  - Before sending the data requests, we wait for a few milliseconds to receive any packet that was sent by the server in any previous run and was not yet taken care of. This is because this scenario was leading to conflicts in our implementation.
  - A dictionary is used to store all the data received and requests are sent to the server until the size of the dictionary equals the number of packets of size 1448 required.
  - This ensures that requests are sent until there exists an entry for each of the offset values in the dictionary.
  - Then the MD5 hash is calculated for the entire data and submitted to the server for verification.

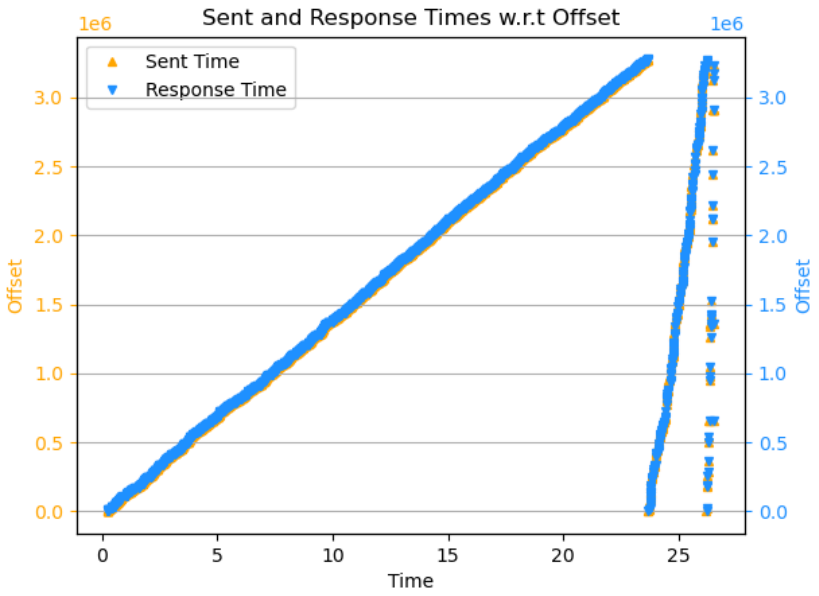
Observations

Run Number	Time (s)	Penalty
1	25.432	21
2	26.587	17
3	28.725	19
4	25.426	15
5	25.769	30

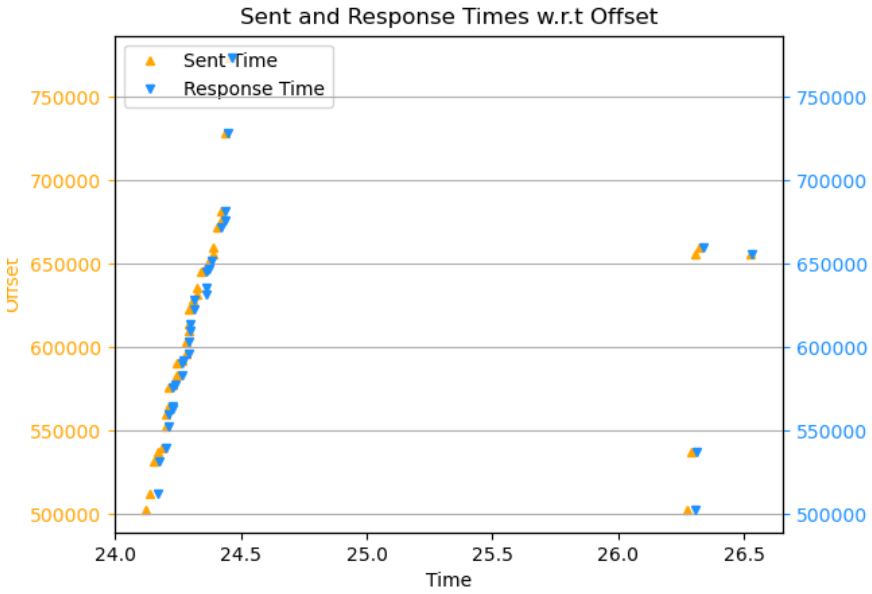
Time and Penalty with Run Numbers

- We are usually getting a time of around 25-30 seconds on the variable rate Vayu Server with this policy.
- There is slight variation in the time between runs which can be explained by the fact that this is a variable rate server.
- Even though the value of α is set in such a way that the variation between runs is minimised, there is still some amount of randomness in the RTT and packet drops hence leading to slight variations between runs.
- The penalty is **less than 30** in all the runs.
- The graphs below explain our approach in more detail.

Graphs



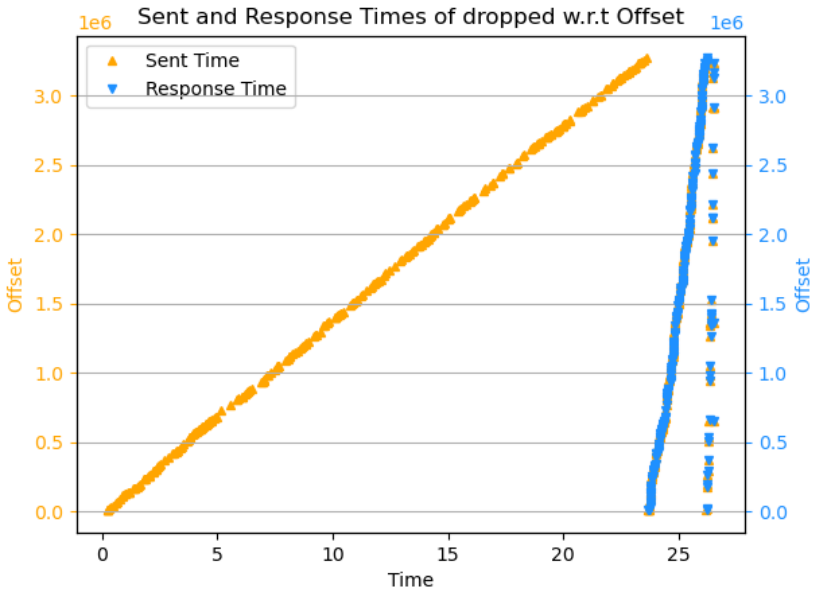
- The above graph shows the **Sequence Number Trace** for the requests sent and the responses received for the AIMD approach.
- All the requests are monotonically sent (in orange) and the responses are received for most of them (in blue). Due to the scale of this graph, it is not very clearly which responses are not received.
- Then the requests are sent again for the offsets for which we did not receive a response.
- This process had to be repeated until all the data was received and the MD5 hash could be submitted for verification.

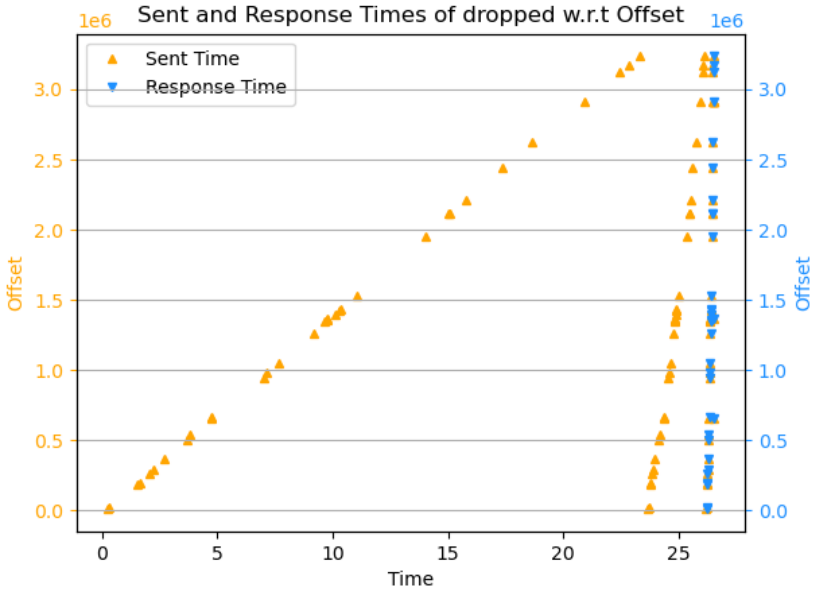


- The above graph shows a zoomed in version of the Sequence Number Trace.

- Here we can see that for some of the requests, a response was not received (Orange triangle present but no corresponding Blue triangle indicating that a response was not received).
- Even in the second iteration some of the packets were dropped and further iterations were needed until all the remaining data was finally received.
- The requests sent in bursts can be seen together in the graph, and the gap between the groups represents the gap between the bursts.

Some More Graphs

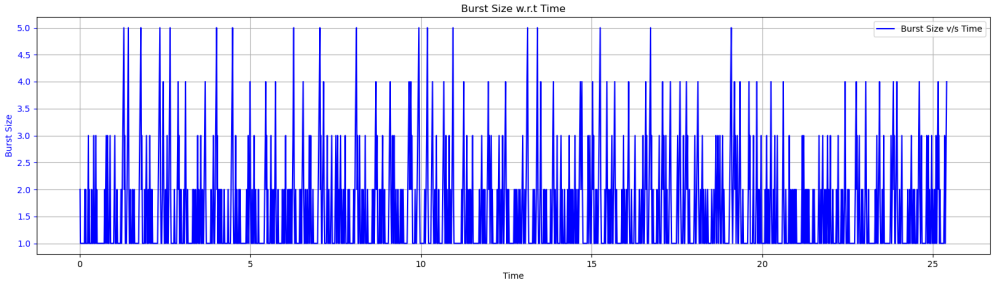


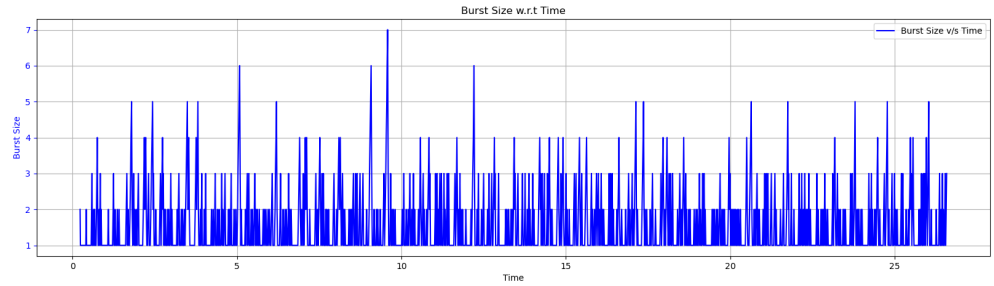


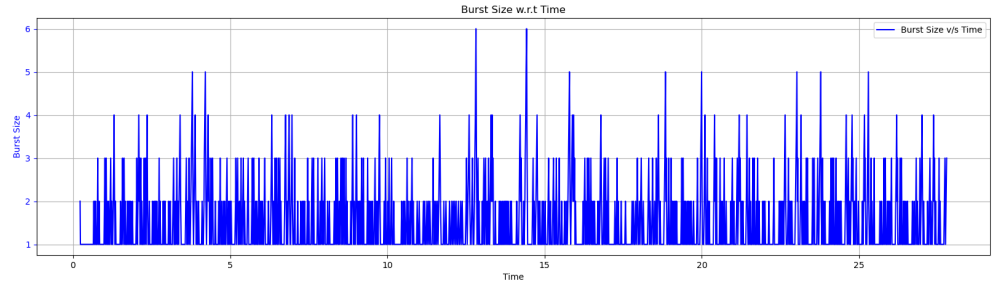
- The first graph shows all the packets that were dropped in the first run. (There is no blue triangle for any of them indicating that no response was received).

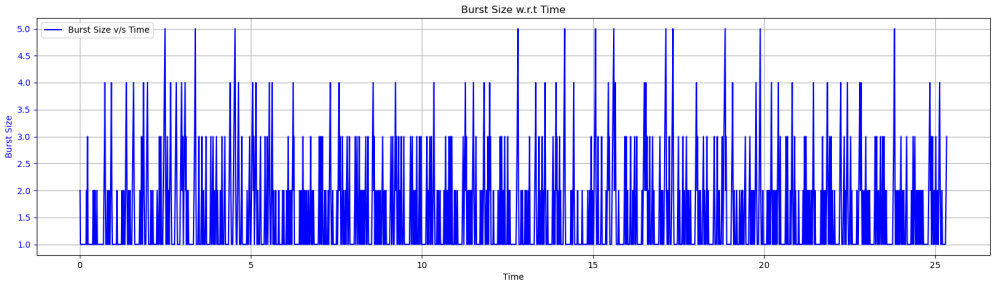
- Therefore further iterations are called for these to request these packets again.
- The second graph shows all the packets that were dropped in both the first and second runs. (There is no blue triangle for any of them indicating that no response was received).
- Therefore further iterations are called for these.

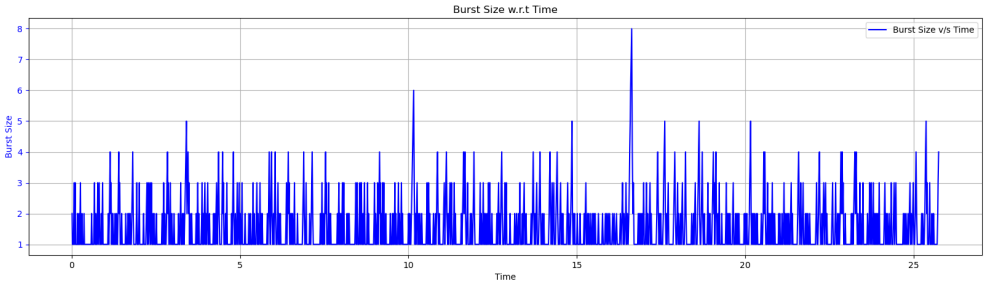
Burst Size v/s Time





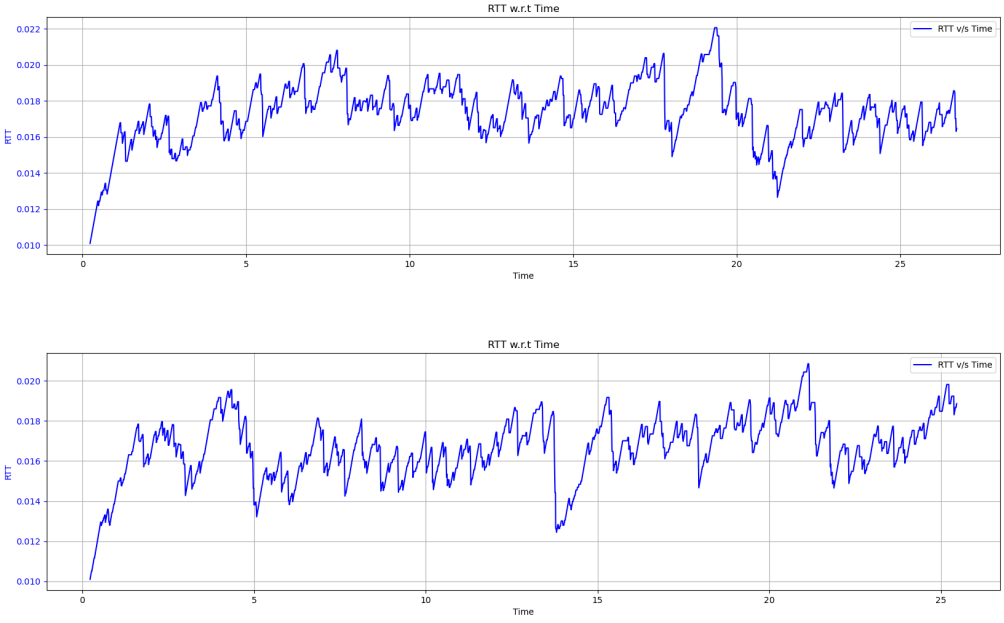






- The above graphs show the **Change in Burst Size of requests w.r.t Time** for our approach.
- We can observe that the graphs goes up and down representing the AIMD approach.
- Since this is against a variable rate server, the window sizes are changed very frequently and the variation is significant as well depending on the variation in the rate at the server.
- We also observe that the window size changes frequently also because the RTT is changing dynamically and keeps increasing or decreasing hence the window size also adjusts.
- Window size is lower when RTT is low and as RTT is increased the window size also increases.

RTT vs Time



- The above graphs show the variation in RTT with time.
- The large hills reflect the variation in the rate of the Vayu server, wherease the spikes represent the RTT being adjusted slightly as and when there are packet drops.
- This dynamic change in RTT ensures that our approach is suited for a variable rate server.

Conclusion

- We also tried to modify and use the alternate approach that we had tried in milestone 2 for a constant rate server, i. e., an adaptive clever hack which increases/decreases sending rate based on a ratio of successful responses received among every 20 requests.
- This however was not a good strategy because on a variable rate server when there is a sudden decrease of rate of server, this approach sometimes get squished before it can adapt to the new rate.
- We also experimented with different implementations of AIMD and found the best one which gives the best result consistently on a variable rate server.
- Therefore using AIMD with EWMA and our chosen conditions seemed to be the best choice as we do not get squished, have a low penalty and a good time to receive the complete file.