COL334 Assignment 3 MileStone 3

Raja Kumar (2021CS10915)
Rishabh Kumar (2021CS10103)

1 Our Implementation

Logic 1: AIMD with Dynamic Timeout for Variable Rate Server

In this part of the project, we've extended our AIMD (Additive Increase, Multiplicative Decrease) logic from Part 2. However, we recognized that the server's rate isn't constant; it varies. To adapt, we've made some crucial modifications. One significant change is the dynamic allocation of timeouts based on the expected Round-Trip Time (RTT) and the Deviation in RTT (DevRTT). This approach allows us to adjust our timeout periods to better suit the current network conditions, ensuring more efficient communication with the variable rate server. This strategy optimizes both the transmission of requests and handling of responses in a way that minimizes penalties and enhances overall performance

Logic 2: Parallel Data Exchange for Minimum Penalty In the second implementation, we've taken a distinctly different approach. Here, we're running two key functions in parallel to handle data exchange with the variable rate server. The first function is dedicated to receiving data from the server. It ensures that we efficiently collect data as it becomes available, allowing us to stay in sync with the server's rate.

Simultaneously, the second function operates independently in parallel. Its role is to send requests to the server. However, what sets this strategy apart is the introduction of controlled time delays when sending these requests. By strategically timing our requests, we aim to minimize potential penalties and maintain a smooth interaction with the server. This approach helps in reducing the negative impacts on performance and ensures the most efficient utilization of resources for data exchange.

2 Plots for AIMD with dynamic timeout

2.1 Plot of Dynamically Changing Burst size vs time

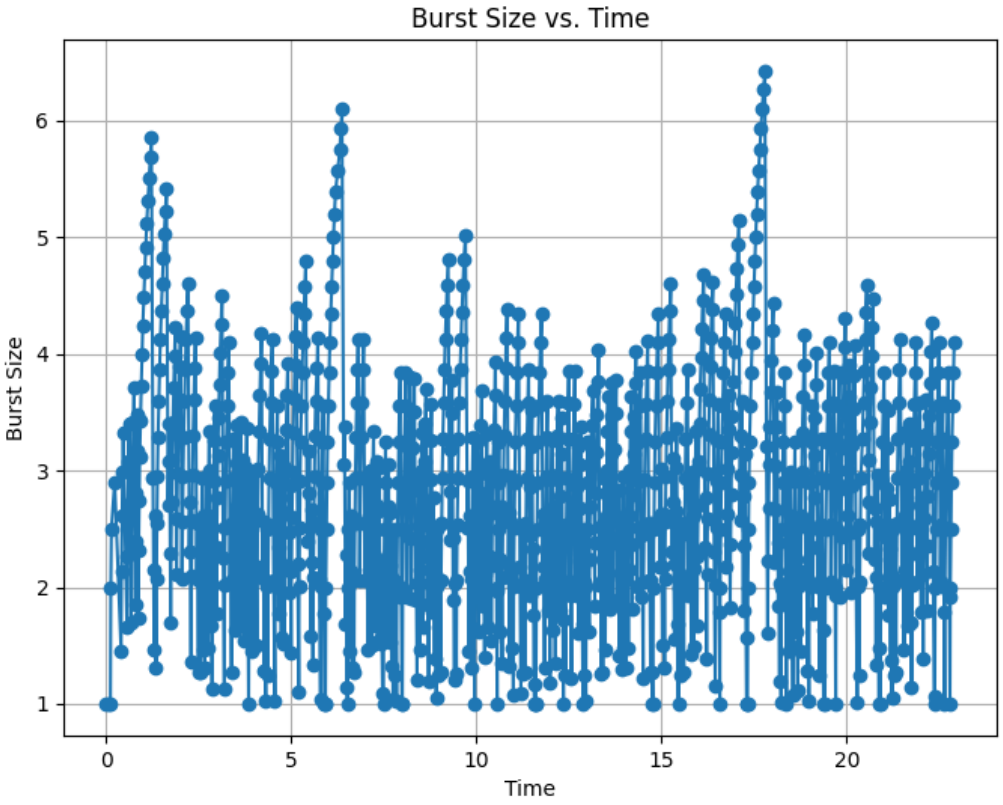


Figure 1: Burst Size vs Time

2.2 Number of Squishing over time

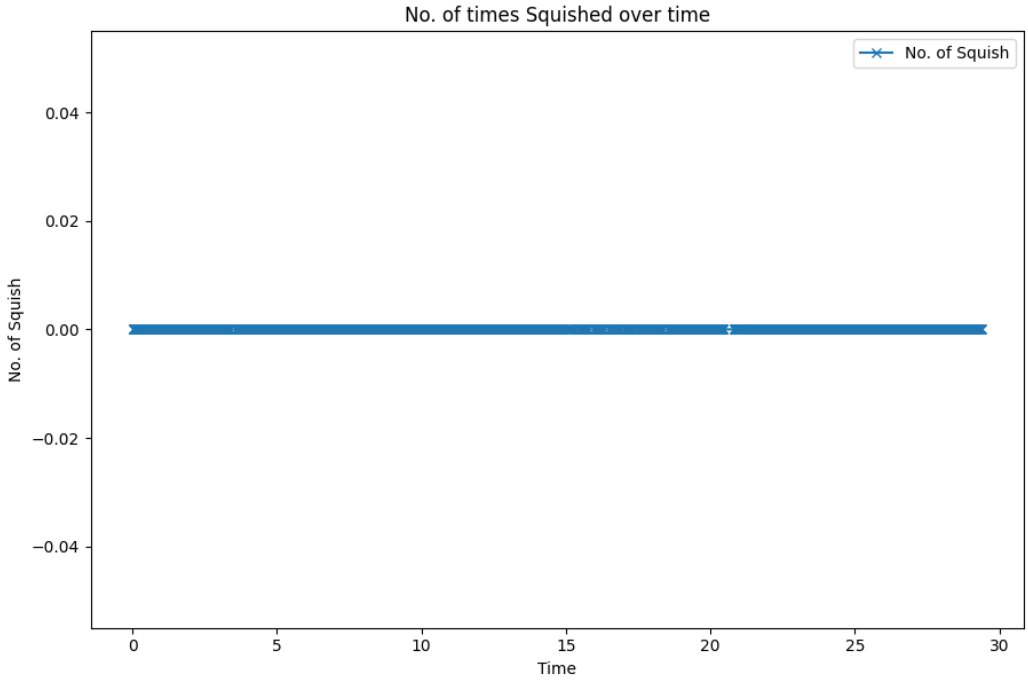


Figure 2: Squish vs Time

2.3 Plot showing the offset and the time for requests sent and replies received

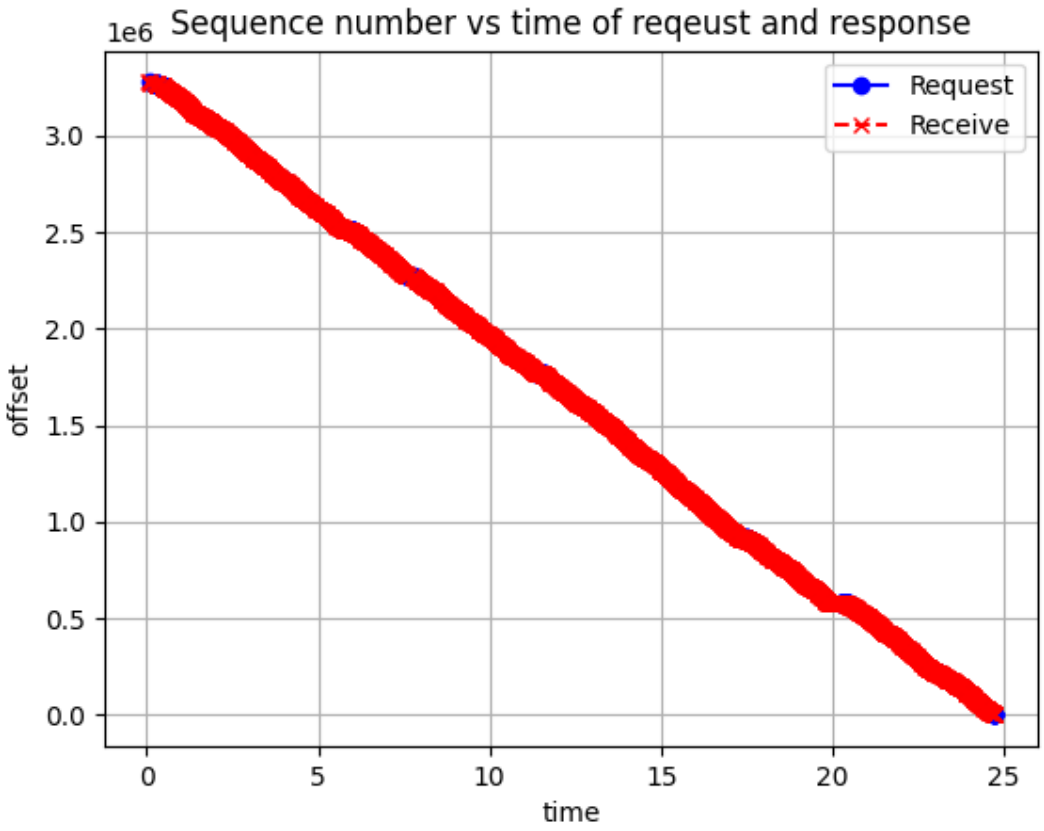
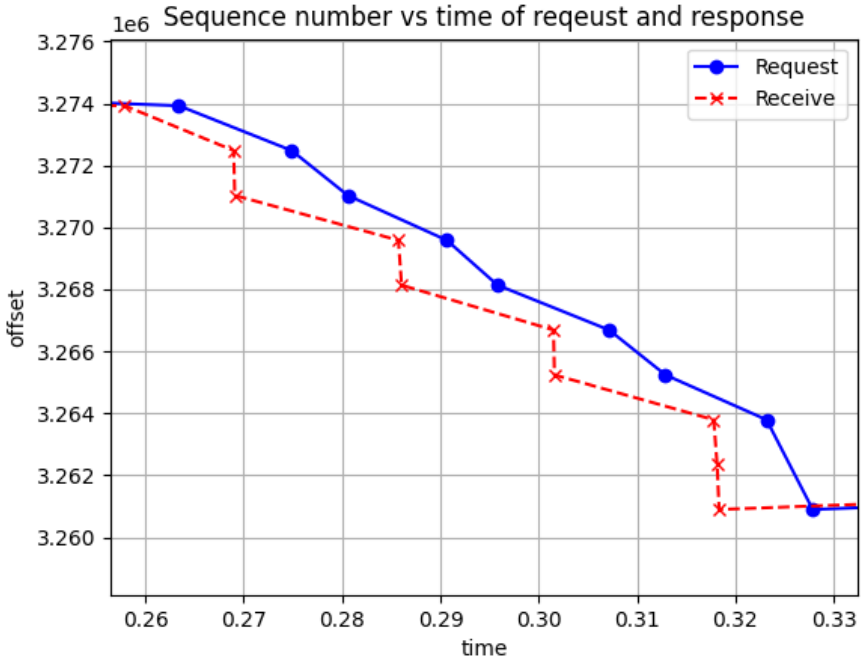
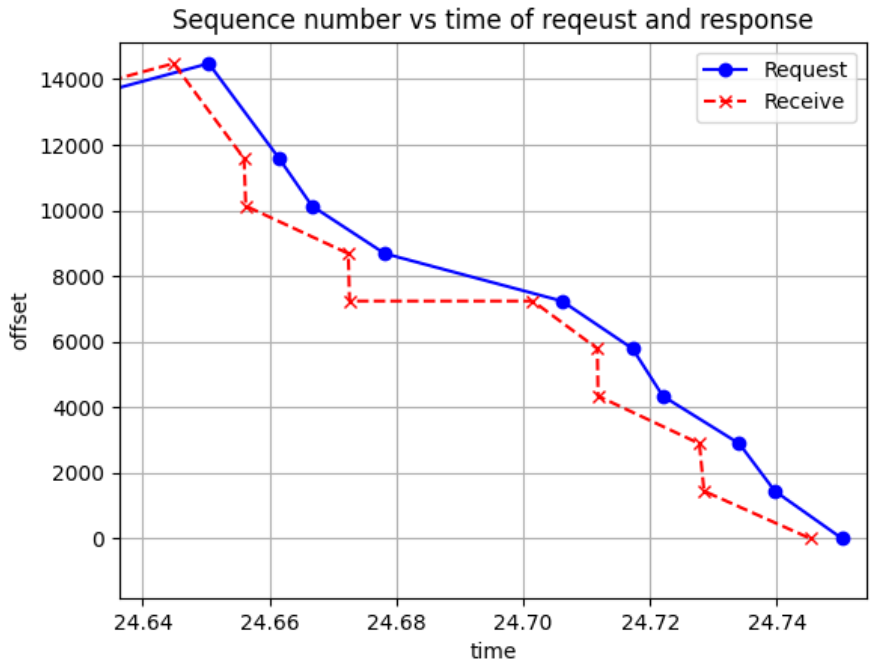


Figure 3: Sequence-number trace



(a) Zoomed in sequence-number trace at the beginning

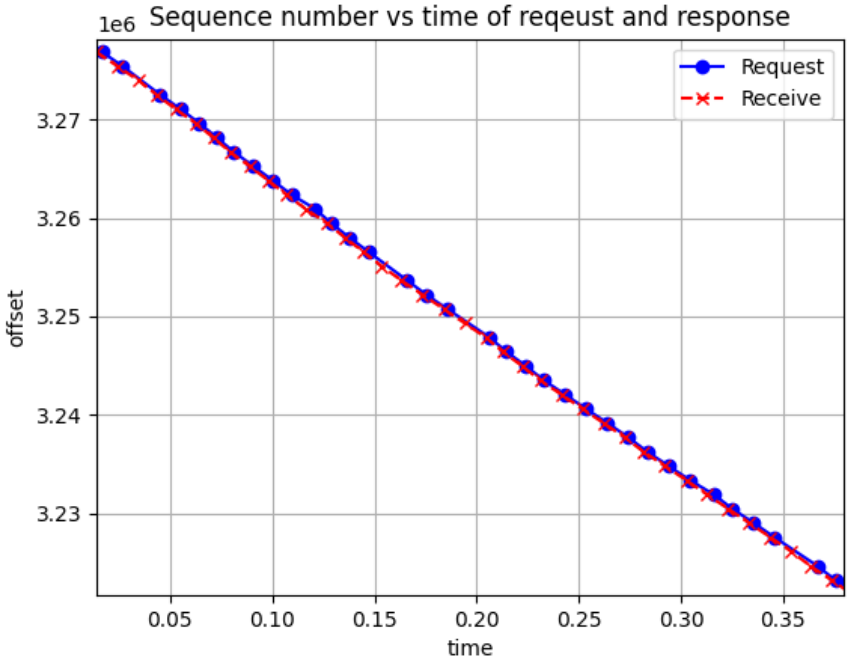


(b) Zoomed in sequence-number trace at the ending

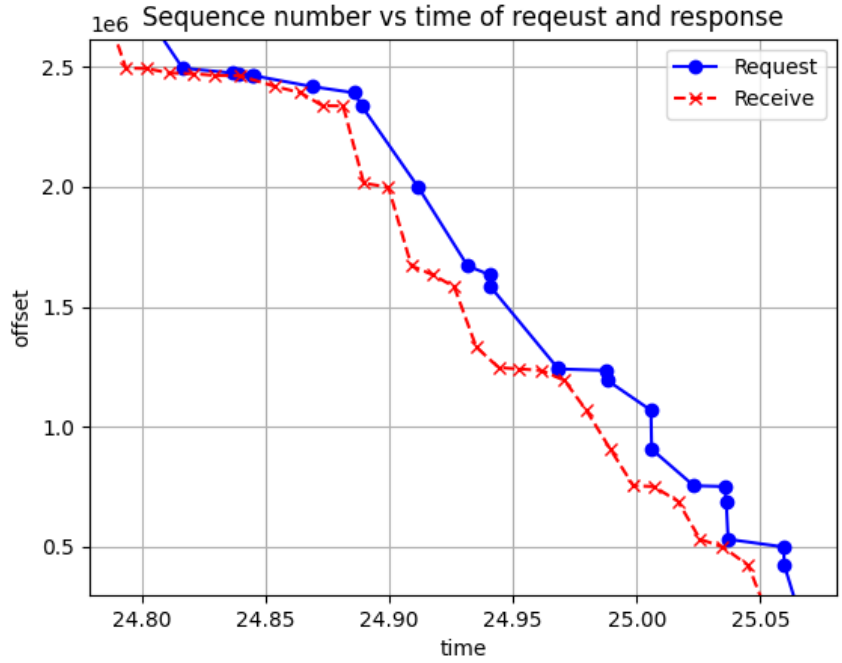
3 Plots for Threading Implementation

3.1 Plot showing the offset and the time for requests sent and replies received

Figure 5: Sequence-number trace



(a) Zoomed in sequence-number trace at the beginning



(b) Zoomed in sequence-number trace at the ending

3.2 Number of Squishing over time

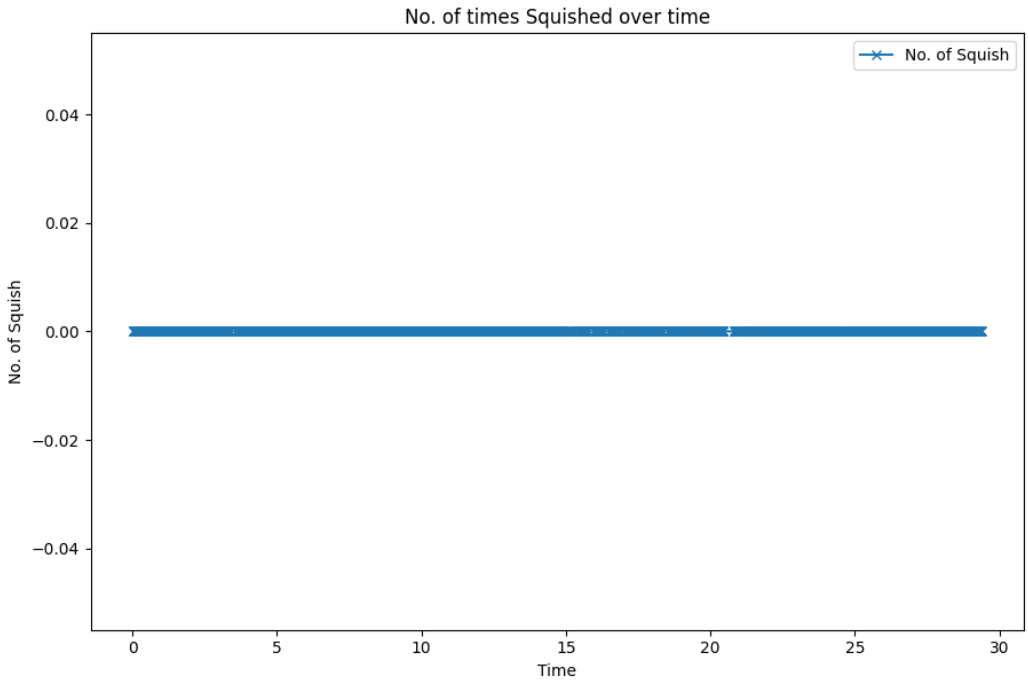


Figure 7: Squish vs Time

4 A sample run result

AIMD	Vayu server	Local server
Total size	3276886	3298151
Result	true	true
Time taken (ms)	23482.0	22199.0
Penalty	26.0	24.0

Parallel	Vayu server	Local server
Total size	3276886	3298151
Result	true	true
Time taken (ms)	23733.0	24252.0
Penalty	0.0	0.0

5 Observations

Below, are the observations we get from above plots:

1. Figure 1. shows the burst-size over time. Every time we get all the requests the burst size has been incremented accordingly and if it is not so then we have halved the burst size.
2. Figure 2. shows that there is no squishing over the entire interval. Our findings show that with the 0.02-second timeout setting, we usually experience a very small penalty, typically between 20 and 30. What's even better is that we don't encounter any squishing issues at all. Additionally, we achieve a high maximum burst size of about 6 to 7.
3. Figure 3. In the sequence number trace we have requested highest offset first which leads the plot to come downwards. Also, due to the scale of the axes, the replies appear to almost overlap with the requests. In the zoomed-in version plot which has been provided below, we can see that each time we didn't received the same number as requested our request number has become halved otherwise has got increased.
4. Figure 7. shows that there is no squishing over the entire interval in threading implementation
5. In thread Impelementation : we have observed total penalty to be 0 and there is no squishing at all.