COL334 Assignment 3

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

1 Our Implementation

We have implemented an Adaptive Increase Multiplicative Decrease (AIMD) technique to manage burst sizes, but we've introduced a modification. Instead of incrementing the burst size by a fixed value like 1 upon receiving a successful response, we're increasing it fractionally by a factor of 1 divided by the current burst size. Furthermore, if any of the burst's requests are not received within the estimated time, we are reducing the burst size by half. This approach allows us to dynamically adjust burst sizes in a more fine-grained and responsive manner.

In our implementation of this logic, we have standardized the packet size at 1448 bytes. We maintain a list that contains all conceivable offset values that can be requested. Based on the current burst size, we extract offsets from the end of this list. If a response is not received for a specific offset, we recycle it back into the list. This process continues until the list is exhausted, ensuring that we request data continuously until all available offsets have been processed.

The previous logic has indeed enhanced the overall time efficiency. However, concerns regarding congestion, penalties, and squishing still persist. To address these issues, we've incorporated a strategy involving the introduction of a minimum sleep duration using system sleep commands. This break in action gives the system more space to preventing congestion, minimizing penalties, and handle squishing issues (which finally comes to 0, we can see in the graph below as well). Notably, this approach has proven to be highly effective in resolving these concerns.

2 Plots

2.1 Plot of Dynamically Changing Burst size vs time

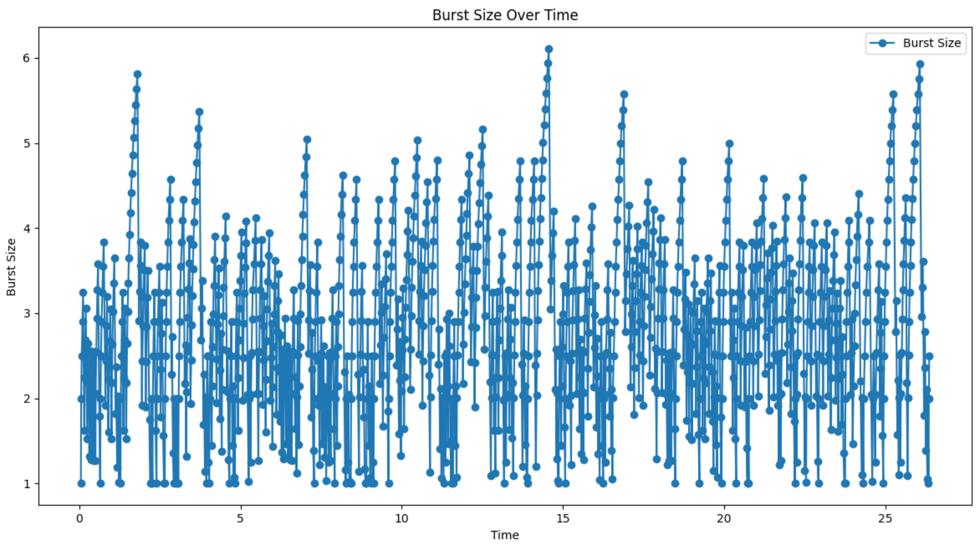


Figure 1: Burst Size vs Time

2.2 Number of Squishing over time

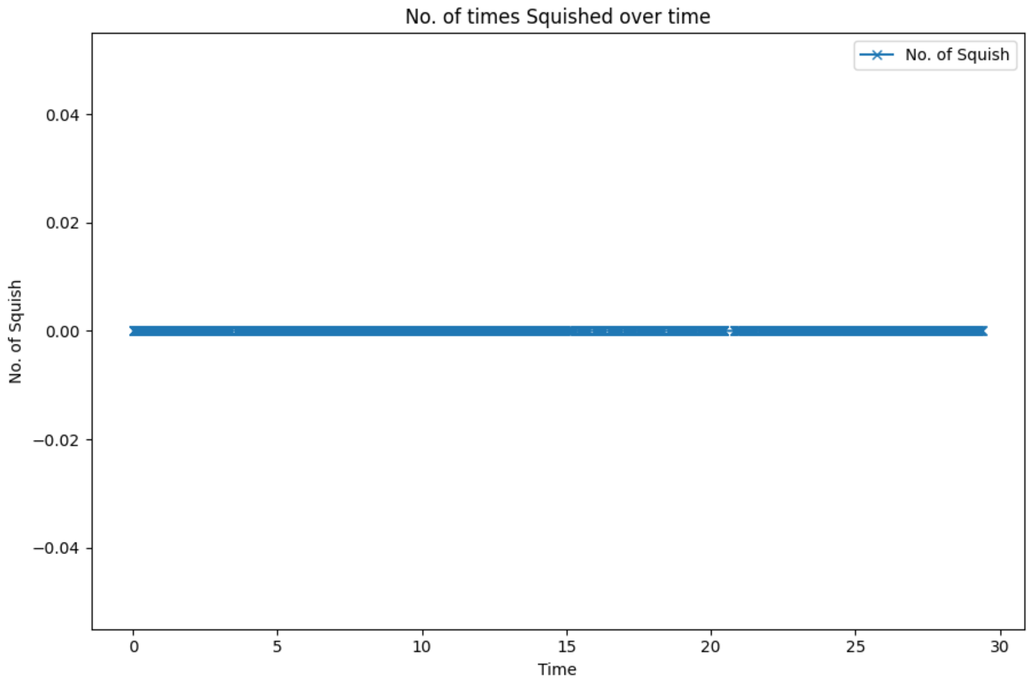


Figure 2: Squish vs Time

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

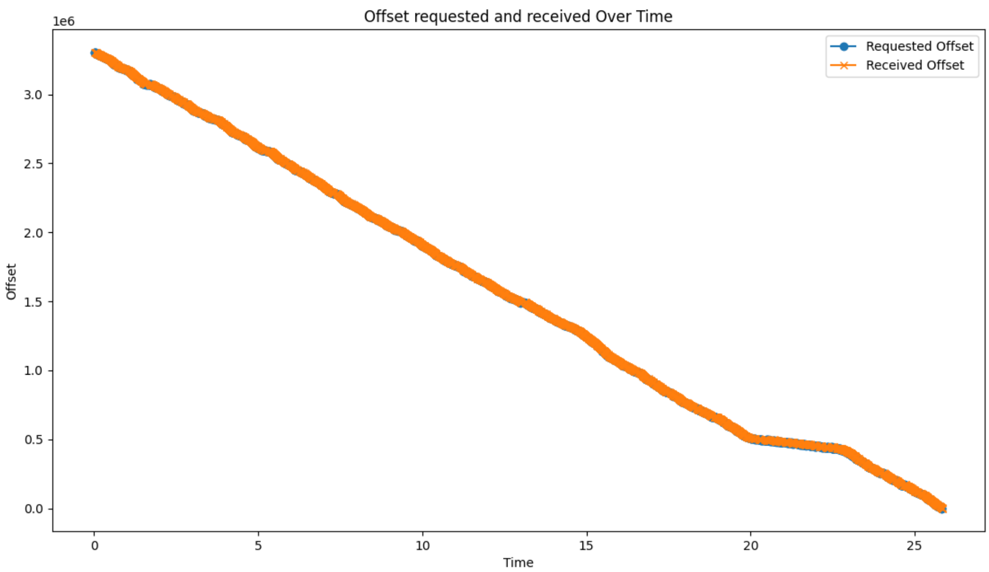
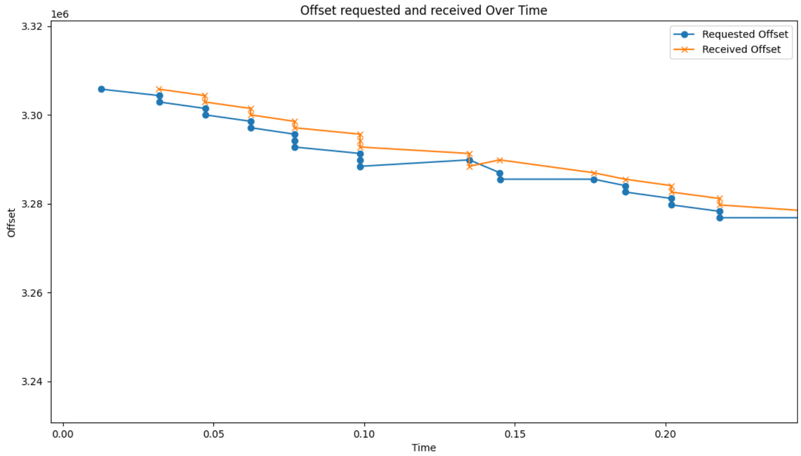
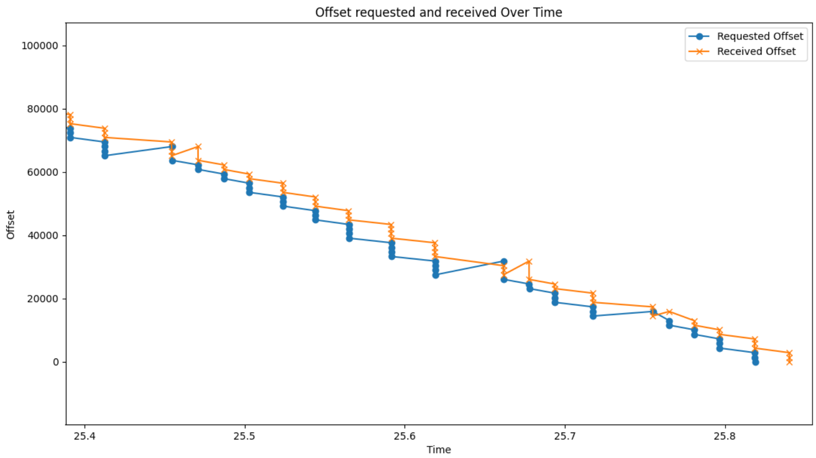


Figure 3: Sequence-number trace



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



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

3 A sample run result

	Vayu server	Local server
Total size	3306804	3295700
Result	true	true
Time taken (ms)	24451.0	22588.0
Penalty	0.0	0.0

4 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 according to the strategy explained in the implementation 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 0 and 5. 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.

   When we don't receive a response for a particular offset, we recycle it by putting it back into the list. In the next request, we'll ask for that specific offset again, but this time with a modified burst size. This creates a kind of continuous sequence of decreasing offset request and response times, where we keep track of the sequence numbers over time, ensuring that no data is lost and that we adapt our data retrieval strategy as needed.

4. After making several adjustments, including modifications to the timeout, burst size, and other parameters, we have determined that a timeout period of 0.02 seconds is the optimal setting for both the Vayu and local servers when it comes to maintaining a constant data transfer rate. This particular timeout duration strikes the right balance and ensures that data is neither excessively delayed nor sent too quickly. In simpler terms, it's like finding the perfect spot where data flows smoothly without overloading or lagging on both the Vayu and local servers.