Reliable Data Transfer

Team: joy_maa

Anish Banerjee[*] Ankit Mondal[†]

August-September 2023

§1. Our Method

- First, we receive the file size from the server.
- To receive the data reliably, we maintain an `receivedlist` initialized with a dummy character. In each iteration, we update the `receivedlist` with the data received from the server. This helps us maintain the data's order, as it is not received sequentially from the server.
- However, the server may refuse to respond to some queries hence we do not expect to receive the entire file at the end of the first iteration. So, we repeat this process multiple times till we receive the entire file.
- We have also calibrated the timeout to a suitable value so that we don't wait too long for skipped requests.
- We have also added a wait period after every message to ensure that messages are sent at more or less uniform rate without resulting in squishing the data.

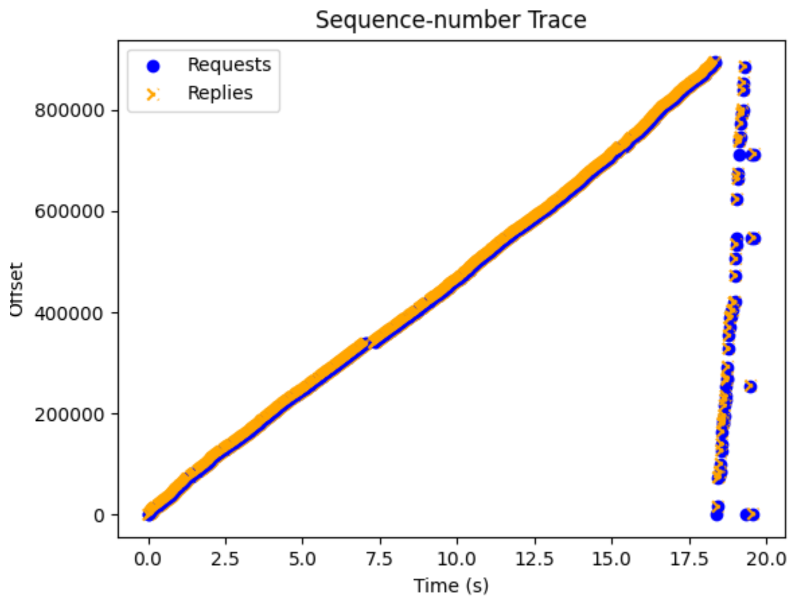

Figure 1: Sequence-number trace

[*] 2021CS10134
[†] 2021CS10229

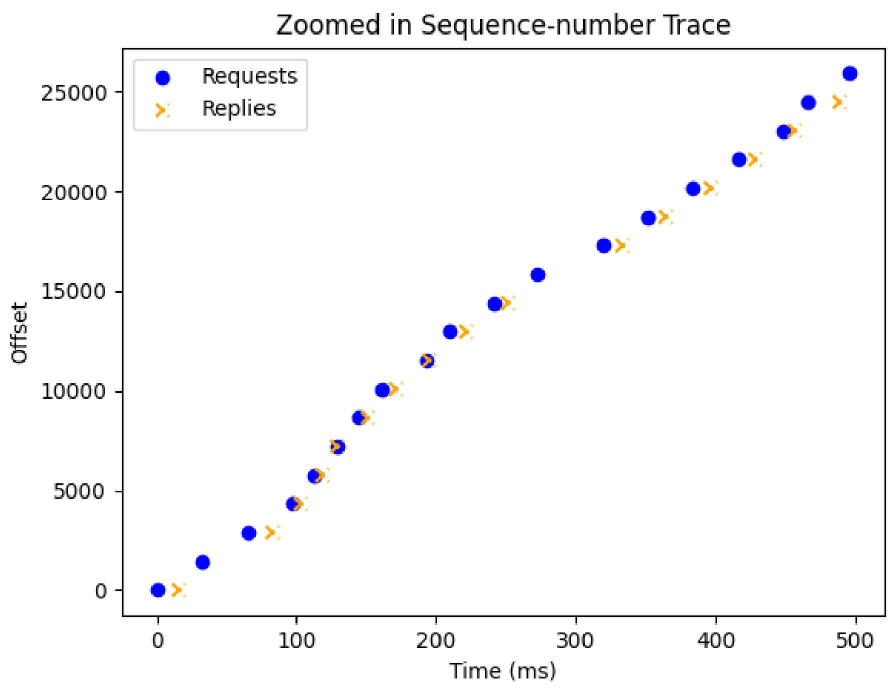

Figure 2: Zoomed in Sequence-number trace

§2. Graphs

- Fig. 1 shows the sequence number trace for an approx 0.01 MB file. Requests are shown in blue and replies in orange. The replies appear to almost overlap with the requests. As mentioned in our method above, we make multiple passes to obtain the entire file. This is evident in the graph too - approximately,
  - The first pass starts at 0s and continues till 17.5s,
  - The second pass starts at 17.5s and ends at 18s and,
  - The last pass starts at 18s and ends almost immediately as it sends just 4 requests.
- Fig. 2 shows a zoomed-in version of the above trace for requests and replies within 500ms from the initial time. From this too, we observe that the requests and responses are almost overlapping. There is a very small round trip time and in a few messages **there is a visible gap between the query and reply**. As the RTT increases the gap between 2 queries also rises (as time between successive queries is round trip time + sleep time per message). Some **queries are dropped midway**. These queries are raised again in the subsequent passes.

§3. Appendix: Other Attempts

We also tried implementing a stop-and-wait protocol in which the sender keeps sending requests till it receives a particular line. When a particular message is dropped, it keeps sending the same request till the response is received. We also tried sending the requests in burst sizes, decreasing the burst size if we had to wait to long for a particular message.

Result: The time to receive file decreased but penalty increased as some messages are squished while trying to find the correct rate to send the requests. Thus for the time being we move ahead with the safer code mentioned at the beginning of the report. The overall graph is also showing much less linearity as compared to previous uniform rate version.

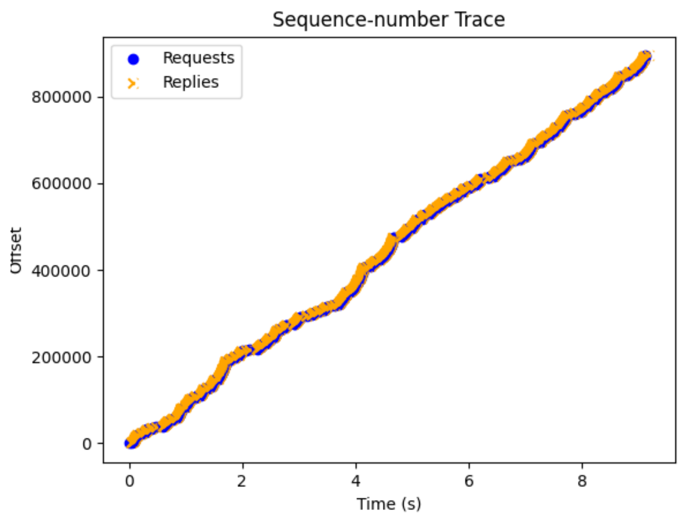

Figure 3: Sequence-number trace 2

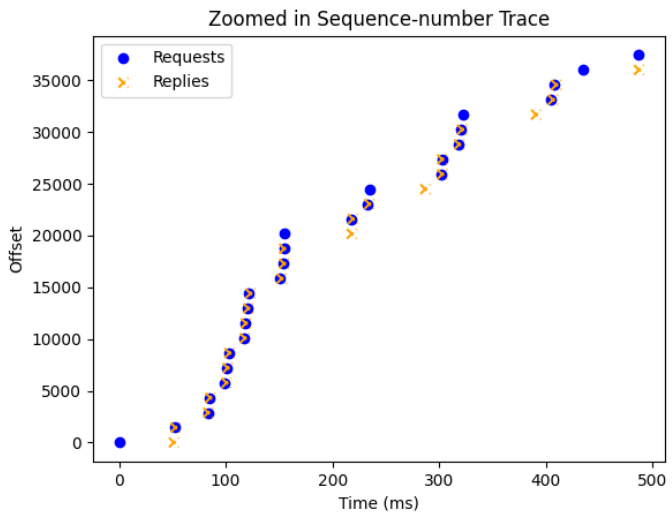

Figure 4: Zoomed main Sequence Number Trace 2