COL334 Assignment - Reliable and congestion friendly yet speedy file transfer Part 2

Protocol Approach: -

1. **Sending "SendSize" Request to the Server:**
   - The program initiates communication with the server by sending a "SendSize\nReset\n\n" request.
   - It repeats this request with a timeout duration of 0.05 seconds until it successfully receives the necessary packet from the server.
   - The purpose of this request is to obtain the total size of the data that the client needs to receive from the server.

2. **Calculating the Number of Requests and Initializing Data Structures:**
   - Once the total size is obtained from the server, the code calculates the total number of requests that are required to retrieve the entire dataset.
   - This calculation is based on the total size of the data and the fixed chunk size of 1448 bytes per request.
   - The code then initializes a boolean array called **offset_received** with all values set to False. This array is used to keep track of the offsets for which data chunks have not yet been received from the server.

3. **Initialization of Frontier:**
   - The code initializes an empty array called "frontier" that can hold a limited number of requests.
   - In this specific code, the "frontier" array is designed to hold a maximum of 10 requests that can be sent to the server in a burst.

`

4. **Managing Burst Requests:**
    - The code uses the "frontier" array to manage the burst transmission of requests to the server.
    - It populates the "frontier" array with requests that need to be sent to the server, ensuring that the number of requests in the array does not exceed the predefined limit of 10.
    - The program processes each request in the "frontier" array and sends it to the server.

5. **Ensuring Efficiency and Control:**

    - By limiting the number of requests in the "frontier" array, the code ensures that the server is not overwhelmed with an excessive number of requests at any given time.
    - It allows the program to maintain control over the rate of requests sent to the server, preventing potential bottlenecks and ensuring a smoother data retrieval process.

6. **Use of the receive_data Function:**

    - The receive_data function is designed to continuously listen for incoming data from the server and process the received data appropriately.
    - It uses a while loop along with a socket.timeout setting to control the duration of the receiving operation. This ensures that the program doesn't get stuck indefinitely while waiting for data.
    - The function handles data received from the server, updates the relevant data structures.

7. **Threading for Managing Data Reception:**

    - Threading is utilized to enable concurrent execution of the data reception process alongside other operations in the program.
    - A separate thread is created for the receive_data function to run independently, allowing the main program to continue sending requests while data is being received.

`

- Threading helps to prevent the program from being blocked during the data reception process, ensuring efficient handling of data chunks from the server.

8. **Timeout Timings in Managing Data Reception and Threading:**

    - Timeout timings play a critical role in managing the duration of specific operations, such as waiting for data from the server and controlling the intervals between different actions in the code.
    - The use of timeout timings ensures that the program doesn't wait indefinitely for data, helping to prevent potential bottlenecks and ensuring the smooth execution of the data retrieval process.
    - By setting appropriate timeout values, the program can effectively balance the time spent waiting for data with the need to perform other necessary tasks, maintaining overall program efficiency.

9. **Data Concatenation and MD5 Hash Calculation:**

    - After receiving all the necessary data chunks, the code concatenates them into a complete string.
    - It calculates the MD5 hash of the concatenated string using the **"hashlib"** library, which ensures data integrity and provides a unique identifier for the concatenated data.

10. **Submitting Data Back to the Server:**

    - The program sends the calculated MD5 hash, along with a specific identifier, back to the server using the "Submit" command.
    - It ensures that the submitted data is in the required format and follows the expected protocol for communication with the server.

Fig 1. Trace of Offset in y-axis with request time in x-axis

Fig 2. Trace of Offset in y-axis with receive time in x-axis

`

Congestion Control Approach:

- To Implement the Congestion control we applied the idea of slow start in which we start from the burst size of 1 and double the burst size of requests up to the threshold value and the increment it thereafter by 1 or constant number.
- We store the burst size for which we had successfully received acknowledgements for all requests sent and keep updating with the greater burst size with successful acknowledgements received in variable min_burst.
- We maintain a variable called threshold for careful increment of burst size for slow start method and update it to half of burst size for which any packets were dropped and increment burst_size by 1 or any constant when burst size equals or greater than threshold value while doing slow start.
- At any point if the client experiences any dropped packet, we decrease the burst size to the size where all packets can be received which has been saved in a variable named min_burst
- If the client gets squished, we also decrease the threshold and the min_burst to the half.

`

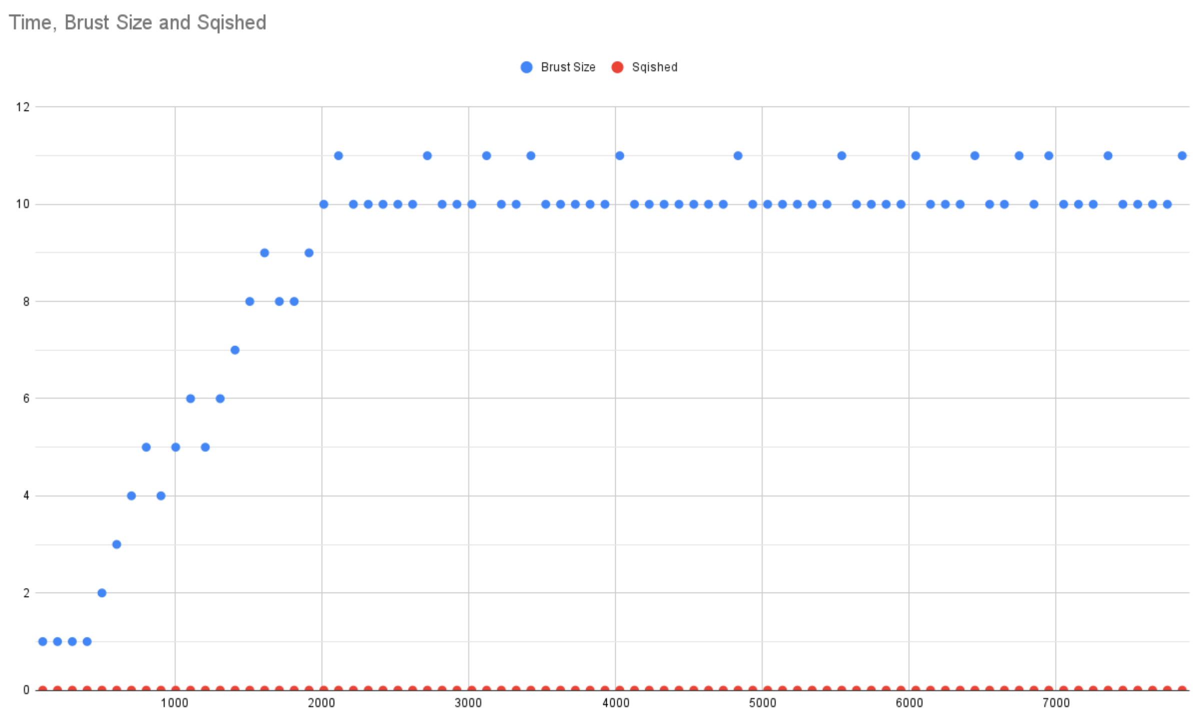


Figure 1: Time vs Brust size and No of Squished for Local Server

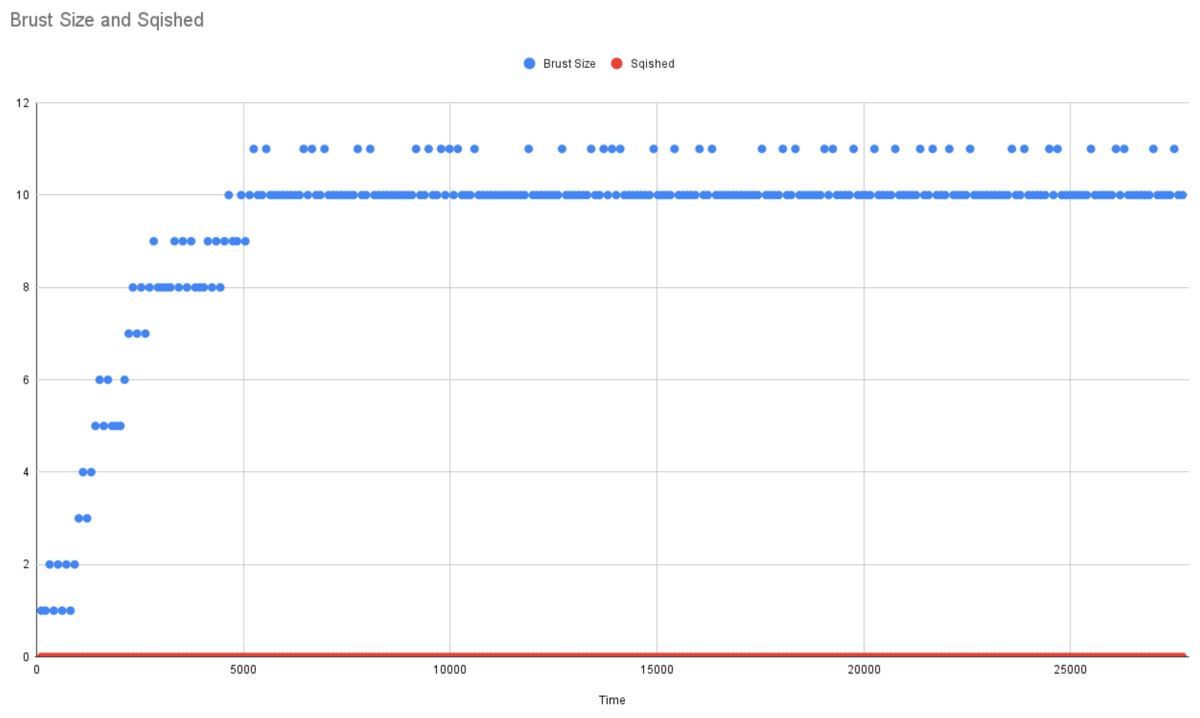
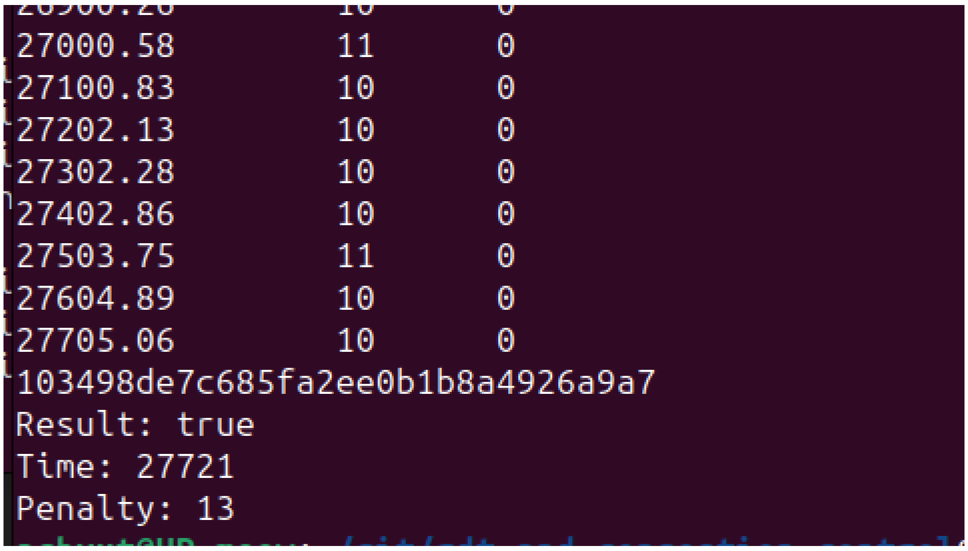


Figure 2: Time vs Brust size and No of Squished for Vayu Server

`



```
27000.58        11      0
27100.83        10      0
27202.13        10      0
27302.28        10      0
27402.86        10      0
27503.75        11      0
27604.89        10      0
27705.06        10      0
103498de7c685fa2ee0b1b8a4926a9a7
Result: true
Time: 27721
Penalty: 13
```

Figure 3: Response Received after Run on Vayu Server

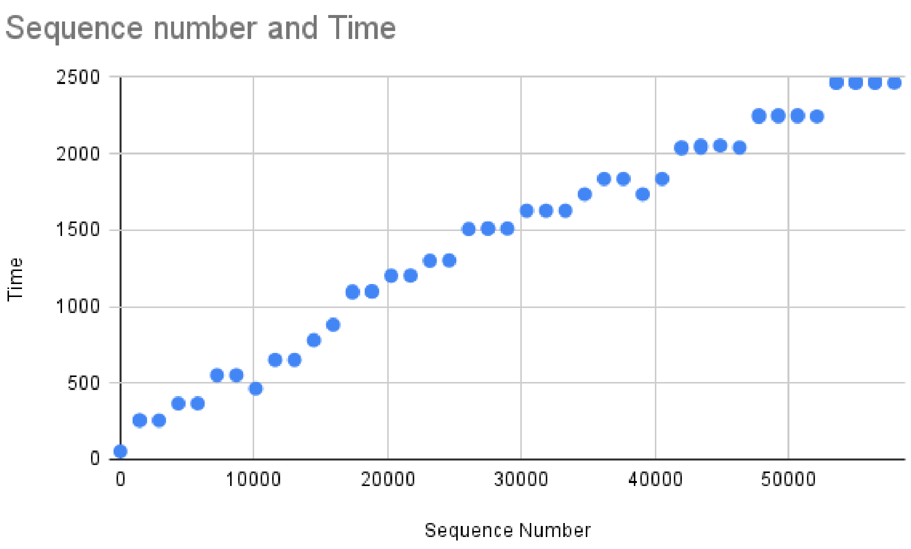


Figure 4: Zoomed Image showing that how burst size is increasing as the client is receives all the requests it has sent

`