COL334 Assignment 3 (Milestone 3)

Aastha Rajani (2021CS10093), Disha Shiraskar (2021CS10578)

November 1, 2023

1 Objective

The primary objective of this assignment was to implement a reliable file transfer protocol over UDP using different methods. UDP, being a connectionless and best-effort protocol, doesn't guarantee data order or even delivery, making it a challenge to ensure reliability. To combat this issue and come up with a client that sends requests at an optimal rate, we tried various possible methods. The report discusses the methods used, corresponding observations and the method which worked out to be the best.

2 Methods Implemented

2.1 Constant Burst Size

- Sending requests in bursts of constant size means we send data in fixed-sized bursts . This approach helps maintain predictable packet sizes and simplifies the implementation. However, it might introduce a bit more delay, and choosing the right burst size can be tricky.
- This method is unable to adapt the changing network conditions. In situations where there is ample available bandwidth, using constant-sized bursts may not fully utilize the network's capacity, potentially leading to slower data transfer rates. Also when the traffic in the network is high it would be beneficial to reduce the burst size, avoid excessive squishing and high penalty.
- We have considered each burst to have a constant size of 5. From the offset vs time graph, it is clear that irrespective of the number of replies we receive for the burst of size 5 , we continue to send the requests at the same rate.
- From the Burst Size and Squish Events over time graph, it is visible that our client is getting multiple times over the entire duration after short durations of time.

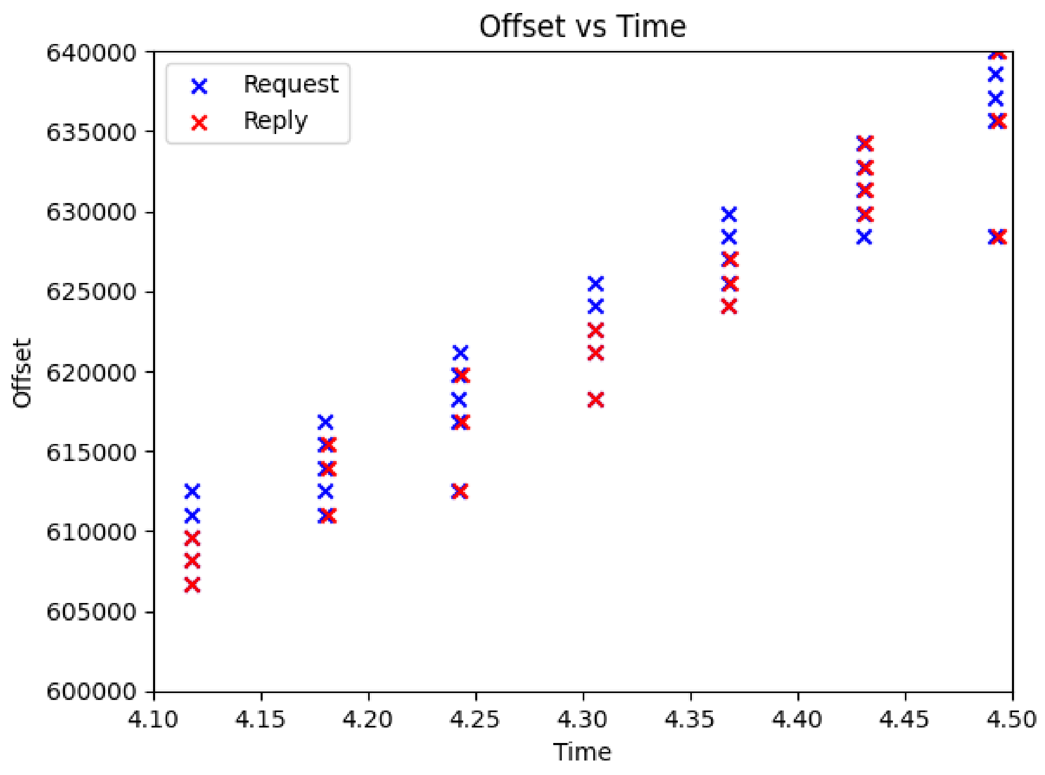


Figure 1: Offset Vs Time Graph.

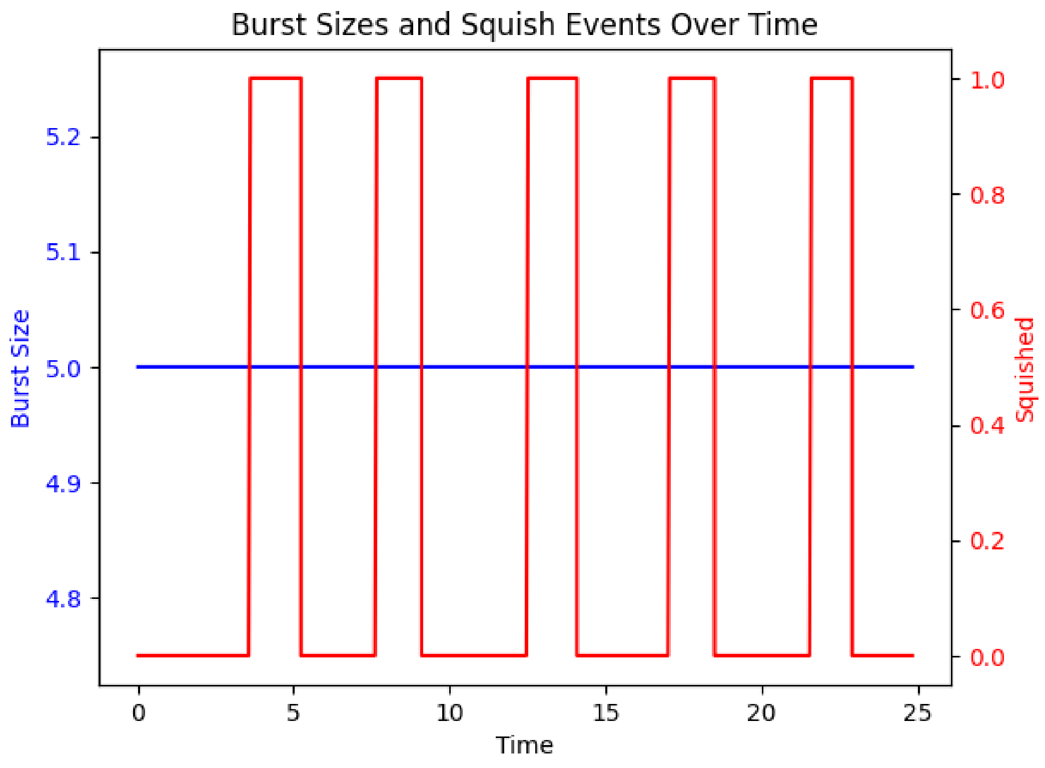


Figure 2: Burst Size and Squish Events over Time .

2.2 Variable Burst Size (AIMD)

- Sending requests in bursts of variable size, also known as Additive Increase Multiplicative Decrease (AIMD), is an adaptive approach for implementing a reliable file transfer protocol over UDP. In AIMD, the size of data bursts varies dynamically based on network conditions.
- When packets are successfully delivered, the sender gradually increases the burst size. When there's packet loss, the sender makes the burst size half to alleviate congestion. AIMD can provide better throughput than constant-sized bursts when the network conditions are favorable.
- It can take advantage of available bandwidth while avoiding congestion. By decreasing burst size in response to congestion or packet loss, AIMD can help prevent network congestion and reduce the risk of packet loss. This, in turn, improves reliability.

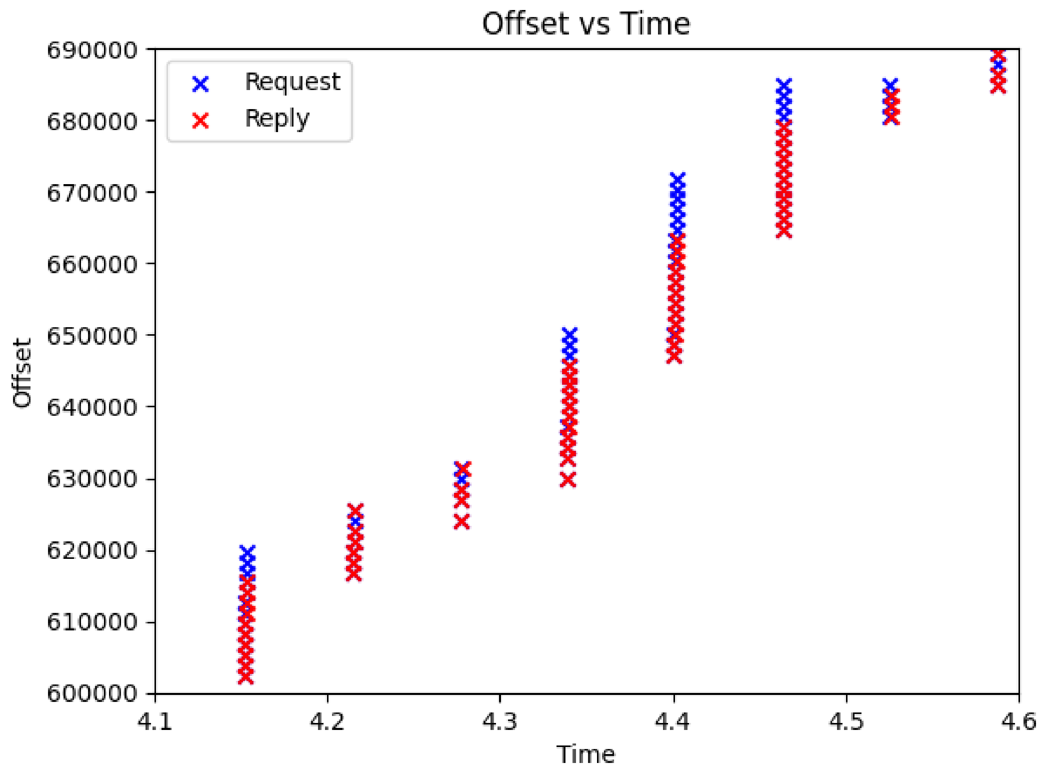


Figure 3: Offset vs Time.

- The above graph shows a zoomed in view of offset vs time for time duration between 4.1 to 4.6 sec. From the graph, it is clear that the burst size varies with time and network conditions. When any packet faces timeout or we receive replies for lesser number of packets than requested the burst size is reduced in the next iteration. Between 4.5 and 4.6s we donot receive same number of replies as replies sent and therefore in the burst just after 4.6s the burst size has been reduced .

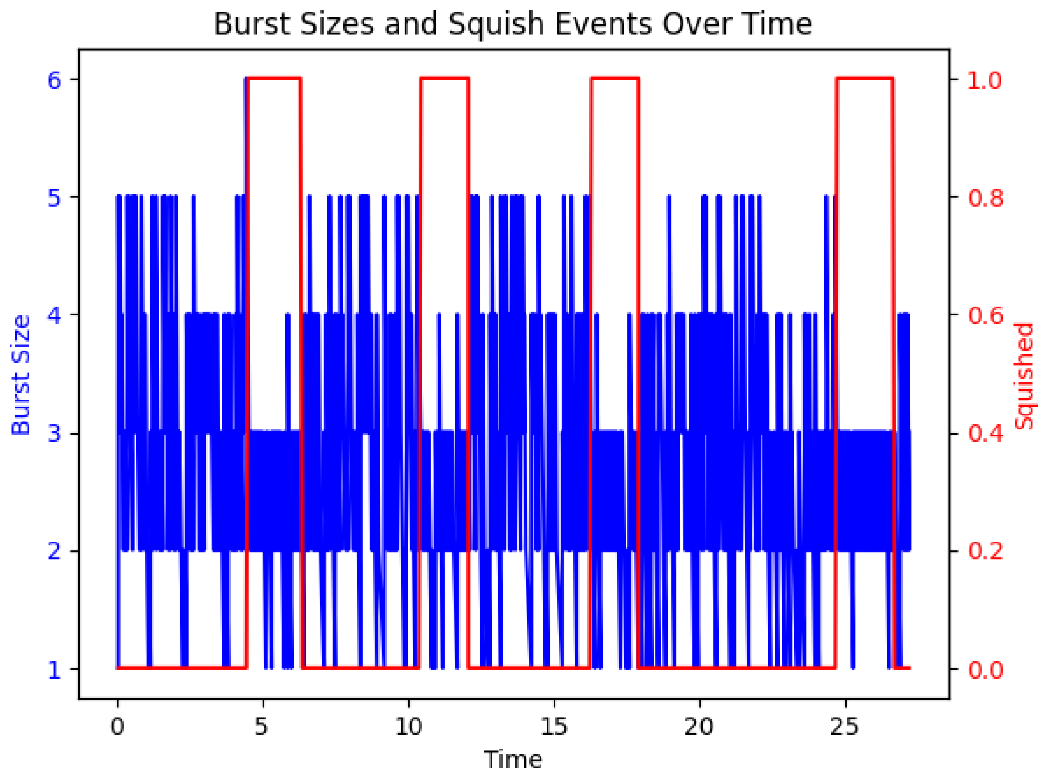


Figure 4: Burst Size and Squish Events over Time.

- The above graph shows the variation of Burst Size and Squish Events over Time. Burst Size changes with time and therefore shows some kind of adaptability towards network conditions. The burst size varies from 1 to 5. Our client gets squished by the server more than once over the entire duration of data collection. However, compared to the previous graph, the squished periods are more largely spaced (i.e client has been squished for lesser number of times compared to constant burst size.)

2.3 AIMD with Self Clocking

- In this scenario, self-clocking helps control the number of packets in the network at a given time. When an acknowledgment for a packet is received, a new request is sent immediately, ensuring there are always N packets ”in transit” in the network. AIMD (Additive Increase Multiplicative Decrease) is applied to this, increasing N when a packet is received and adjusting the congestion window size accordingly.
- The congestion window size (cwnd) is increased by adding 1/N for every acknowledgment (ACK) received. N is a variable representing the number of packets that have been acknowledged. As each ACK is received, the cwnd is incremented by 1 divided by the current value of N. This is the additive increase part of the AIMD algorithm with self-clocking.
- The multiplicative decrease part of AIMD comes into play when packet loss or congestion is detected, and it reduces the cwnd in response. This has been implemented using multithreading. Locks have added delays in the overall time.

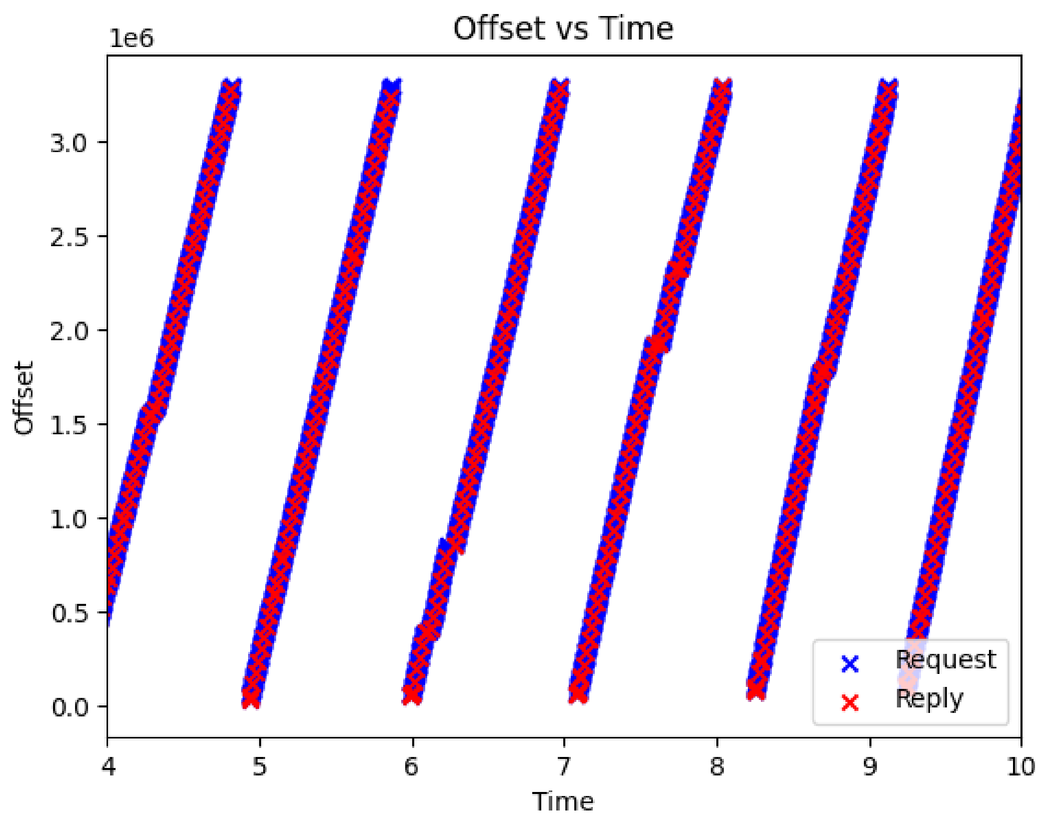


Figure 5: Offset vs Time.

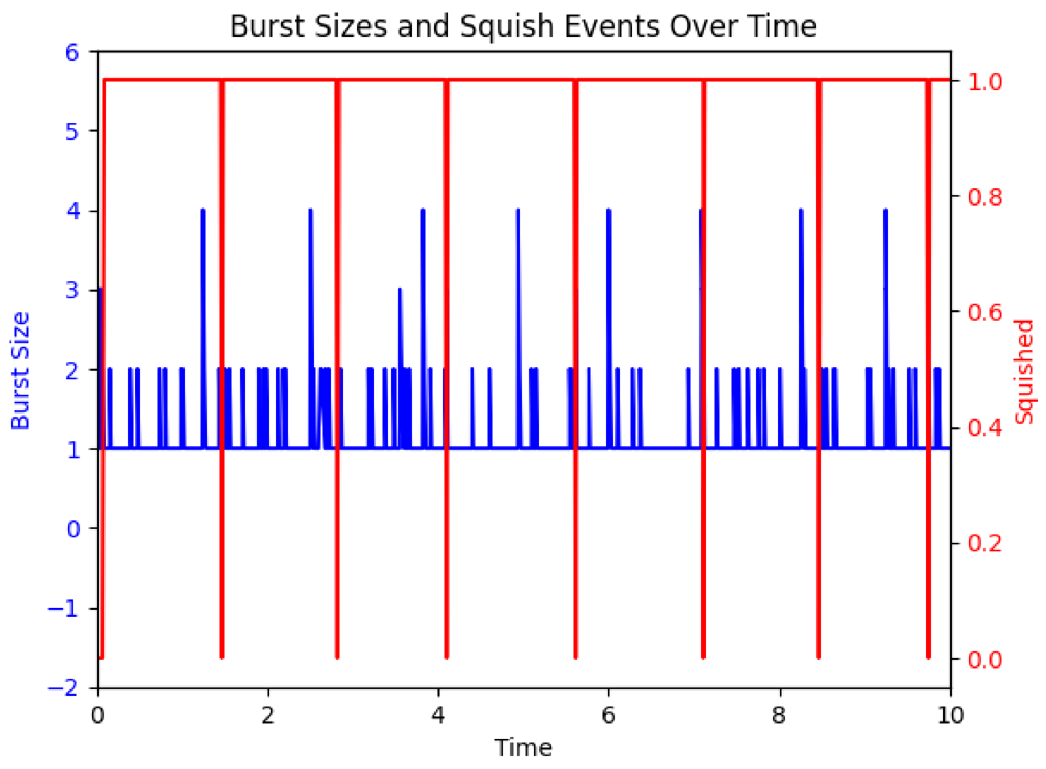


Figure 6: Burst Size and Squish Events over Time.

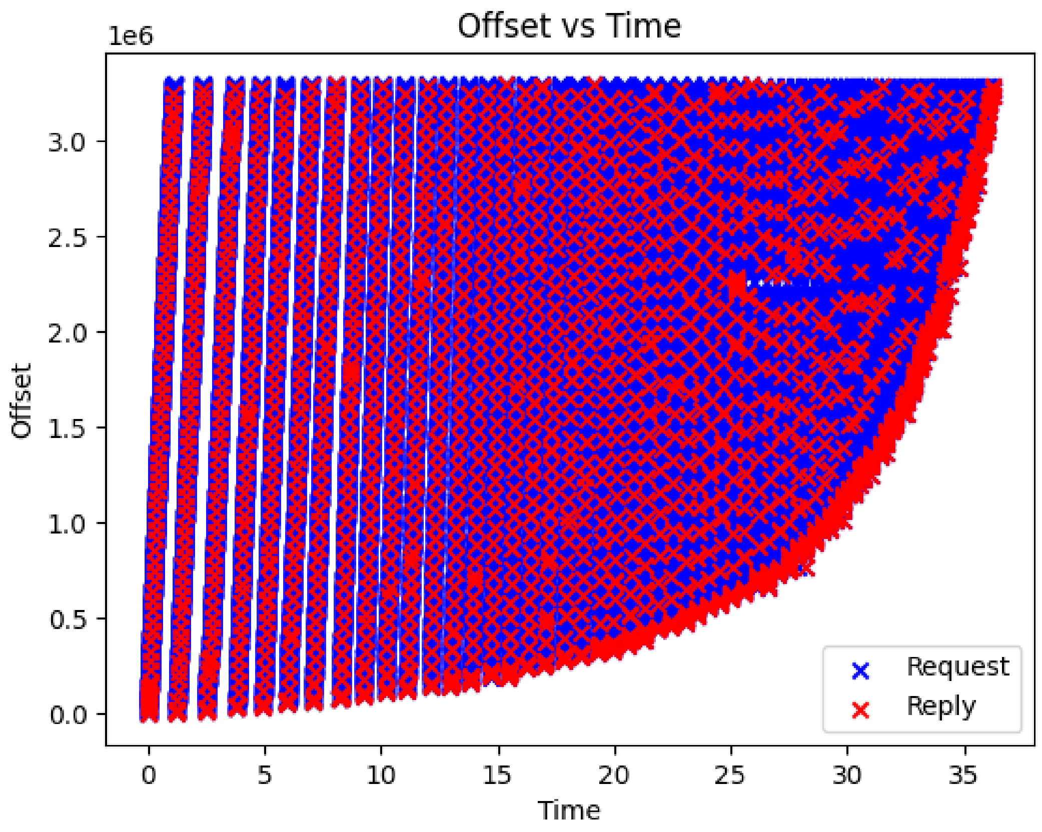


Figure 7: Burst Size and Squish Events over Time.

- In our code we have maintained a list of pending offsets. Graph 1 shows zoomed-in view of time between 5s to 10s. From graph 1, it is clear that approximately once in every 0.5s the requests for all of these pending offsets is being sent. The offsets which are received simultaneously are being removed from the list of pending offsets. Moreover, replies are received at extremely lower rate compared to requests sent.
- Graph 2 shows the squish periods between the time 10s to 20s. It depicts that for most of the time the client is getting squished due to extremely high rate of sending requests. Congestion window size varies from 1 to 5. Client has been squished and penalized for maximum times in this implementation.
- Graph 3 shows a zoomed-out view of offset vs time for this implementation. The offsets which are received simultaneously are being removed from the list of pending offsets. Since the size of this list reduces over time, lesser requests are being sent as time passes. T

2.4 AIMD with Dynamic Timeouts

- This approach combines the original variable burst size (AIMD) implementation along with dynamic timeouts. The timeout is calculated and updated once in every burst using the estimated RTT and deviation RTT calculated till then. This ensures even more adaptability towards network conditions.

- The timeout for packet acknowledgment is dynamic, depending on the average Round Trip Time (RTT) and the deviation RTT. It involves 4 tunable parameters - ALPHA (for Exponential Moving Average), BETA (for deviation), MARGIN (extra waiting time based on deviation), AIMD Increase (alpha) and Decrease (beta) factors. By appropriate approximations and experimentation with vayu and the local server , we came up with the best possible values of these parameters.

- Apart from dynamic timeout and AIMD, we have slightly modified the code. By using only AIMD and dynamic timeouts , we require a time.sleep() to reduce sending rate slightly because if we don't then it leads to increased penalty. However inserting time.sleep() after every packet sent, led to increased delays. To balance this issue and ensure optimization of both time and penalty, we executed time.sleep() command once in every 3 requests sent. This led to reduced time and penalty.

- The detailed explaination of our implementation is as below:

  - AIMD and Data Retrieval: The AIMD algorithm is a fundamental congestion control strategy often utilized in networking. Initially, data requests, or 'bursts', are sent out in small sizes. If responses to these requests are successfully received, the code infers that the network can handle more traffic and thus, the burst size is increased additively. However, if a timeout occurs, indicating potential network congestion or packet loss, the burst size is multiplicatively decreased. This approach ensures that the client doesn't flood the network with too many requests during congestion, yet scales up requests when the network conditions are favorable. The code determines the total data size first, divides it into packets, and continuously fetches these packets while adapting the request rate based on network feedback.
  - Dynamic Timeouts for Response Handling: The dynamic timeout strategy complements AIMD. Instead of relying on a static wait time for server responses, the code employs an Exponential Moving Average (EMA) to continuously update the anticipated Round-Trip Time (RTT) based on previously observed RTTs. This average RTT, augmented with a margin calculated from the deviation from this average, determines the wait time for a response. The dynamic nature of this timeout ensures the client reacts promptly to varying network delays. If responses are coming in faster than anticipated, the wait time is adjusted downwards, speeding up data retrieval. Conversely, if network conditions deteriorate and responses slow down, the timeout increases, preventing premature request retries and further network congestion.
  - In essence, the combination of AIMD for adaptive data request rates and Dynamic Timeouts for intelligent response anticipation ensures the client efficiently retrieves data even under fluctuating network conditions. This approach maximizes data throughput while minimizing potential network congestion and packet loss.

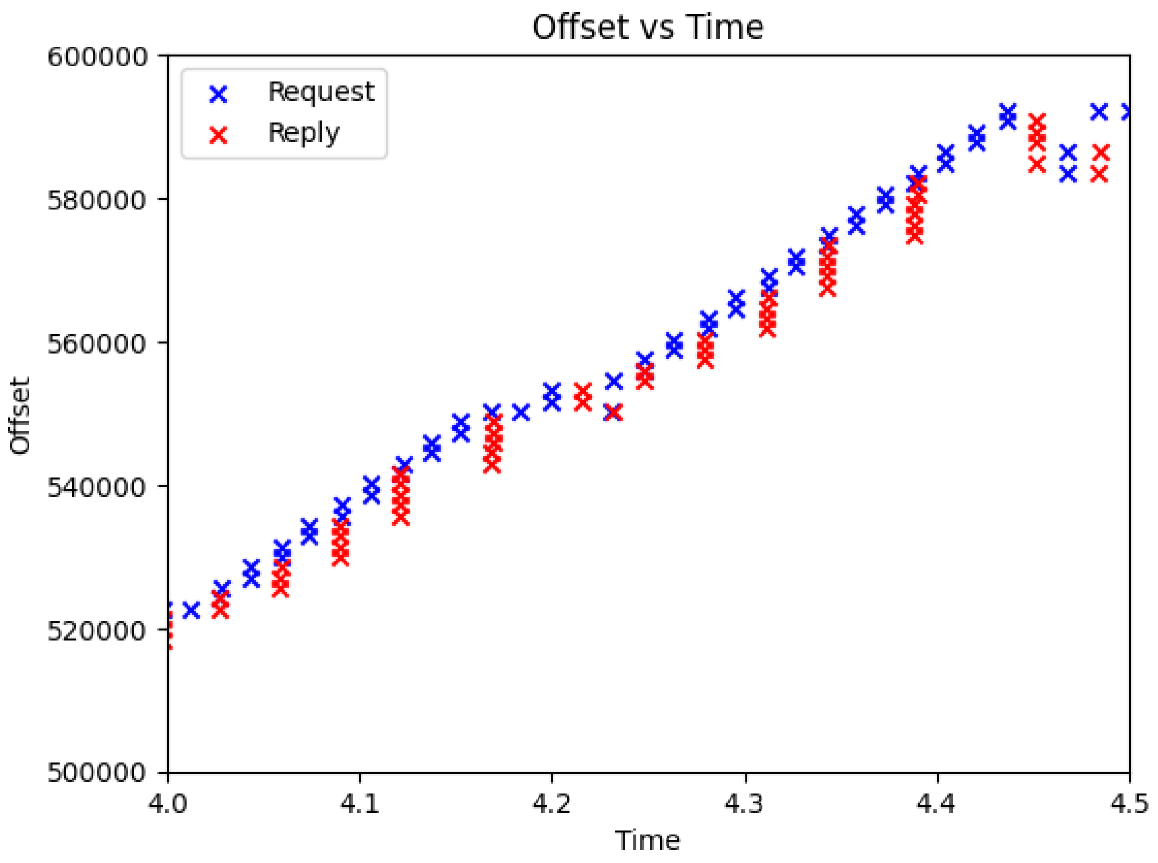


Figure 8: Burst Size and Squish Events over Time.

- This graph shows the variations in requests sent and replies received between the time 4.5s to 5.0s. It is similar to the graph in case of variable burst size (AIMD) implementation. Note that though burst size may go beyond 3, it is not clearly visible in the graph due to insertion of time.sleep() command once every 3 requests sent.

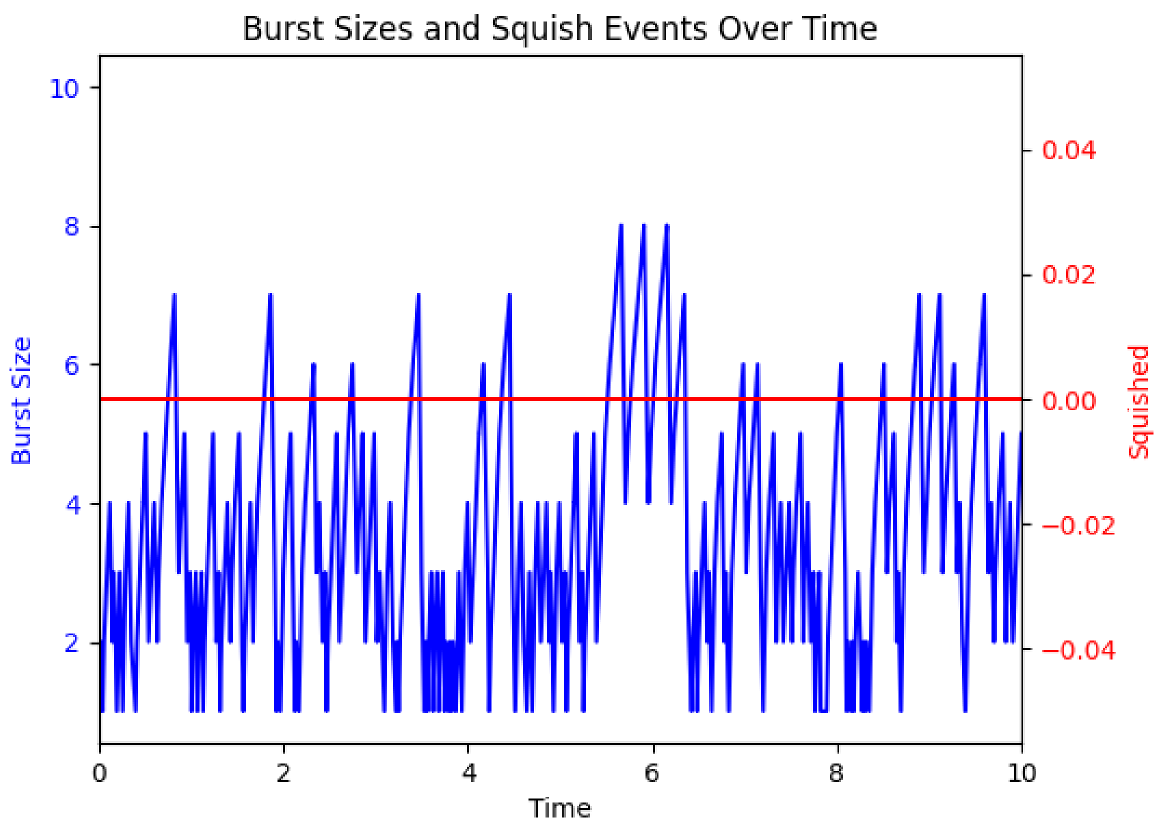


Figure 9: Burst Size and Squish Events over Time.

- This graph shows the Burst size and squished period vs time for 0s to 18s. Since we have figured out the optimal values of our 4 tunable parameters by experimentation, our penalty is always less than 100, thus ensuring that our client never gets squished over the entire period. This implementation leads to optimal time and penalty.

2.5 Multithreading

- This involves multithreading for collecting data in an optimal way. It defines 2 functions one for sending requests and other for receiving. The main thread runs the send burst function which send requests corresponding to all packets in the pending packets list. The other thread runs the receive data function which collects the responses from the server, stores them and removes the corresponding offset from received packets list.
- The receiving thread is first started and continues to run throughout the entire duration of the program. The main thread runs a while loop until the pending packets gets empty. During each iteration pending packets is updated to received packets.
- This method incurred zero squish events , zero penalty and completed execution in 39 seconds.

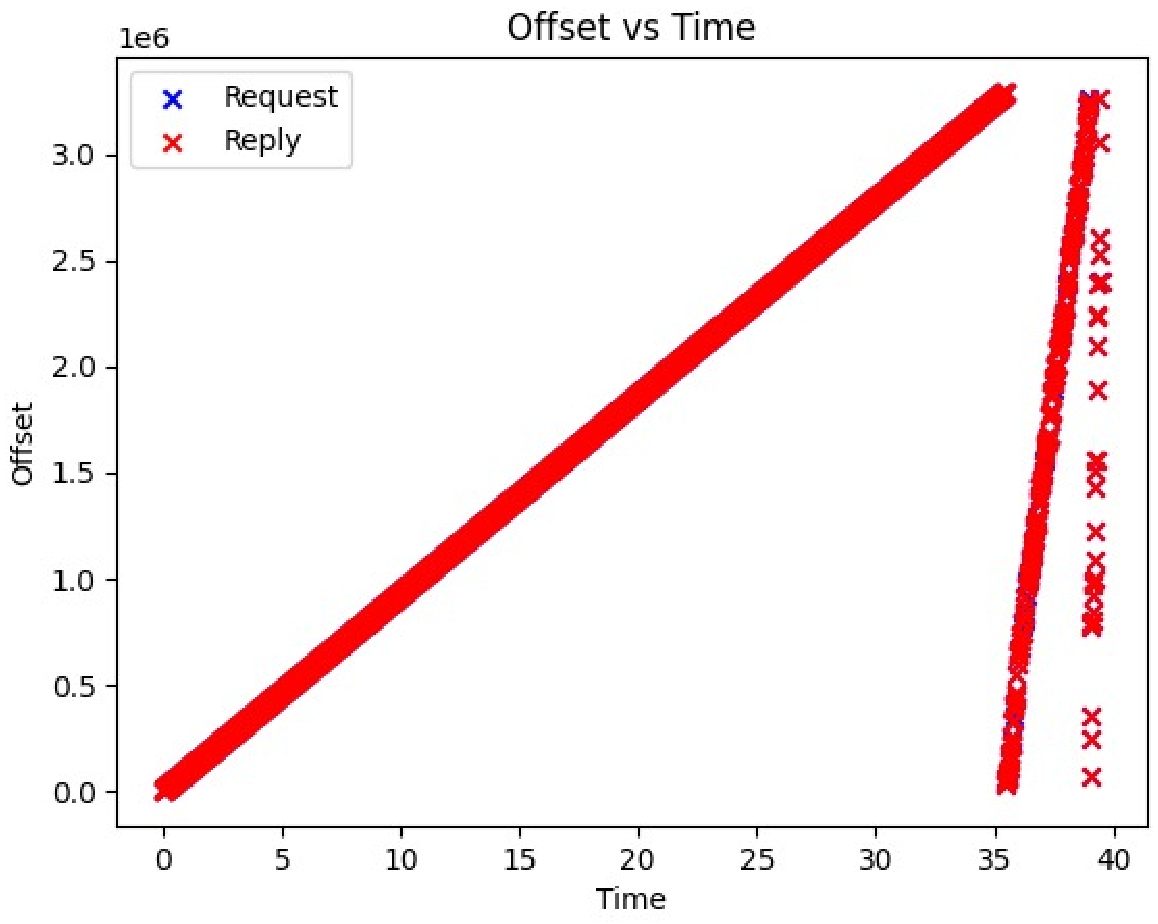


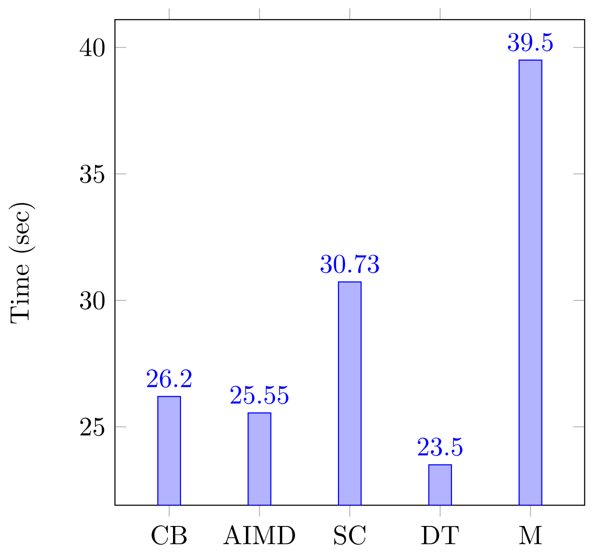
Figure 10: Offset over Time.

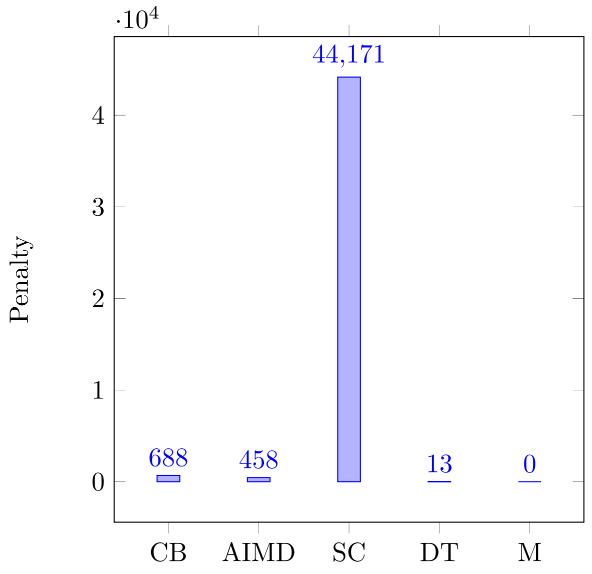
3 Conclusion

The time and penalty on variable rate local JAVA Server with 50000 lines taken by each of the 4 implementations is listed below.

Method	Time (sec)	Penalty
Constant Burst	26.20	688
Variable Burst (AIMD)	25.55	458
Self Clocking	30.73	44171
Dynamic Timeout	23.5	13
Multithreading	39.5	0

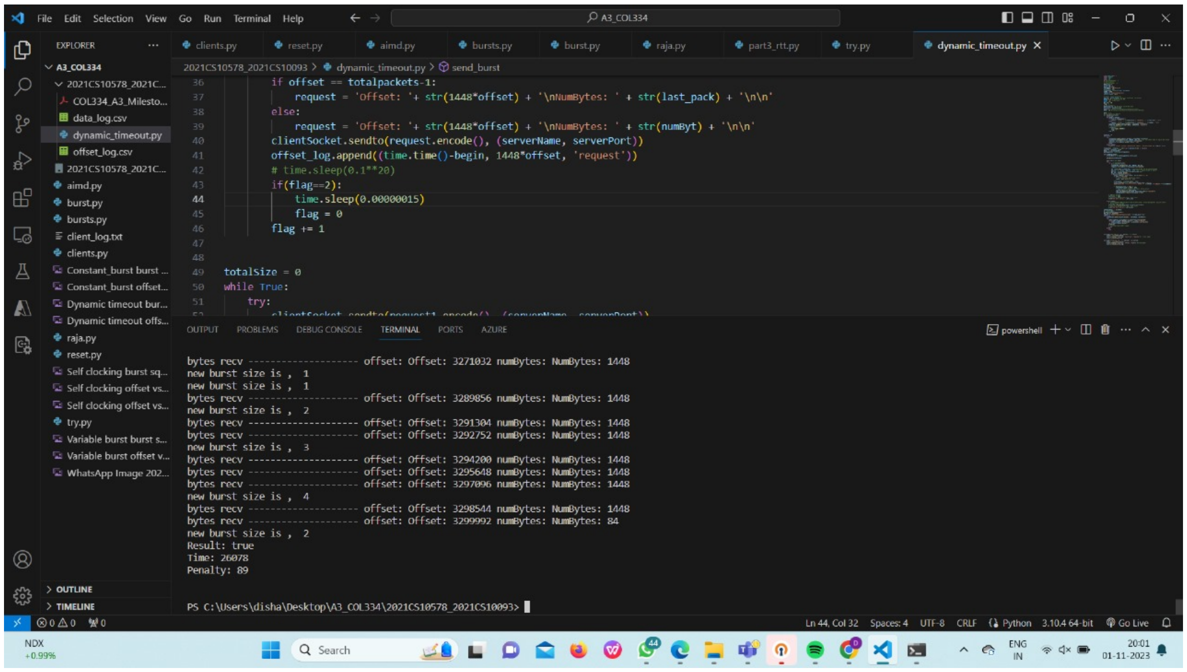
Table 1: Observation.





3.1 Dynamic Timeouts and Multithreading is the best !

From the above 2 graphs , it is clear that AIMD with dynamic timeouts and Multithreading is the best method. For the vayu servers , AIMD with Dynamic Timeouts took 26.42 seconds and incurred a penalty of 71.3. We couldn't take readings for multithreading on vayu server due to reset issues. The below screenshot shows the same :



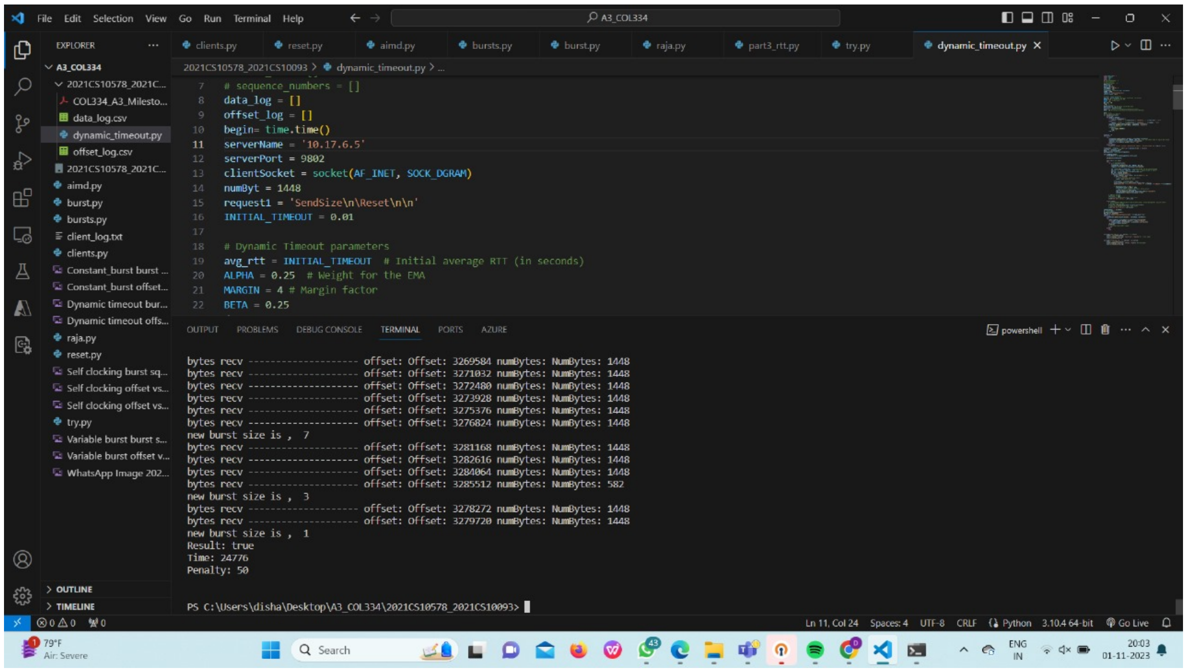
```python
        if offset == totalpackets-1:
            request = 'Offset: '+ str(1448*offset) + '\nNumBytes: ' + str(last_pack) + '\n\n'
        else:
            request = 'Offset: '+ str(1448*offset) + '\nNumBytes: ' + str(numByt) + '\n\n'
        clientSocket.sendto(request.encode(), (serverName, serverPort))
        offset_log.append((time.time()-begin, 1448*offset, 'request'))
        # time.sleep(0.1**20)
        if(flag==2):
            time.sleep(0.00000015)
            flag = 0
        flag += 1


totalSize = 0
while True:
    try:
```

```
bytes recv -------------------- offset: Offset: 3271032 numBytes: NumBytes: 1448
new burst size is ,  1
new burst size is ,  1
bytes recv -------------------- offset: Offset: 3289856 numBytes: NumBytes: 1448
new burst size is ,  2
bytes recv -------------------- offset: Offset: 3291304 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3292752 numBytes: NumBytes: 1448
new burst size is ,  3
bytes recv -------------------- offset: Offset: 3294200 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3295648 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3297096 numBytes: NumBytes: 1448
new burst size is ,  4
bytes recv -------------------- offset: Offset: 3298544 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3299992 numBytes: NumBytes: 84
new burst size is ,  2
Result: true
Time: 26078
Penalty: 89

PS C:\Users\disha\Desktop\A3_COL334\2021CS10578_2021CS10093>
```

Figure 11: Time Taken and Penalty Incurred for 10.17.51.115 using DT



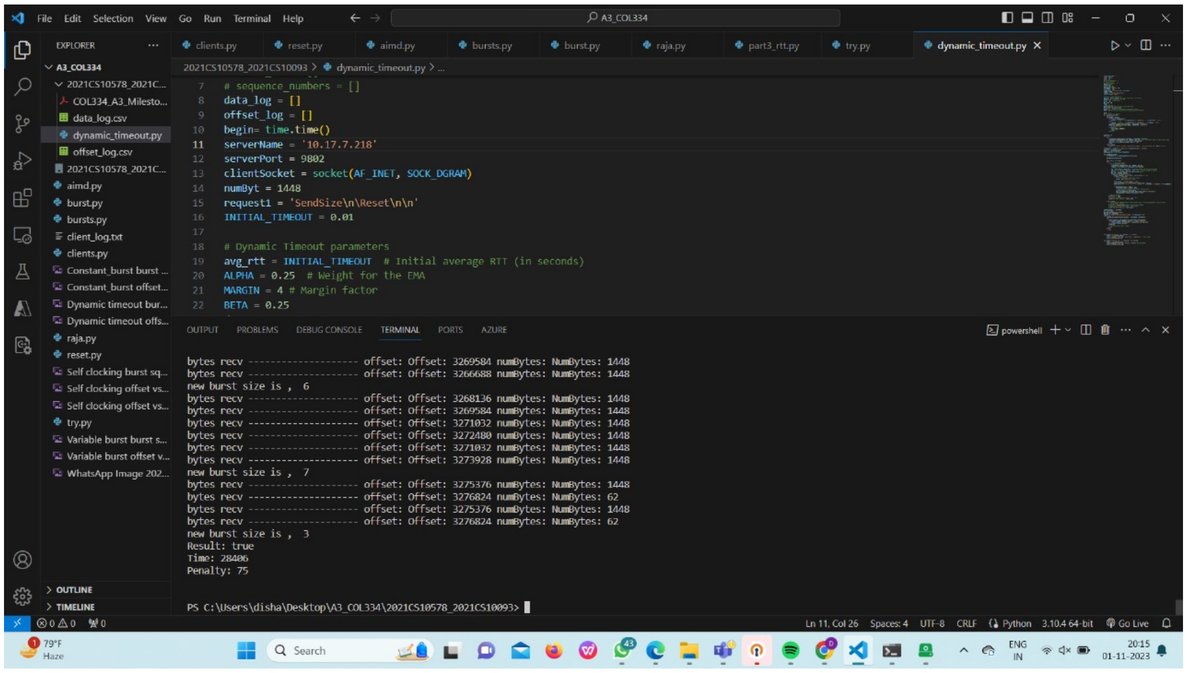
```python
# sequence_numbers = []
data_log = []
offset_log = []
begin= time.time()
serverName = '10.17.6.5'
serverPort = 9802
clientSocket = socket(AF_INET, SOCK_DGRAM)
numByt = 1448
request1 = 'SendSize\n\Reset\n\n'
INITIAL_TIMEOUT = 0.01

# Dynamic Timeout parameters
avg_rtt = INITIAL_TIMEOUT  # Initial average RTT (in seconds)
ALPHA = 0.25  # Weight for the EMA
MARGIN = 4 # Margin factor
BETA = 0.25
```

```
bytes recv -------------------- offset: Offset: 3269584 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3271032 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3272480 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3273928 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3275376 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3276824 numBytes: NumBytes: 1448
new burst size is ,  7
bytes recv -------------------- offset: Offset: 3281168 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3282616 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3284064 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3285512 numBytes: NumBytes: 582
new burst size is ,  3
bytes recv -------------------- offset: Offset: 3278272 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3279720 numBytes: NumBytes: 1448
new burst size is ,  1
Result: true
Time: 24776
Penalty: 50

PS C:\Users\disha\Desktop\A3_COL334\2021CS10578_2021CS10093>
```

Figure 12: Time Taken and Penalty Incurred for 10.17.6.5 using DT



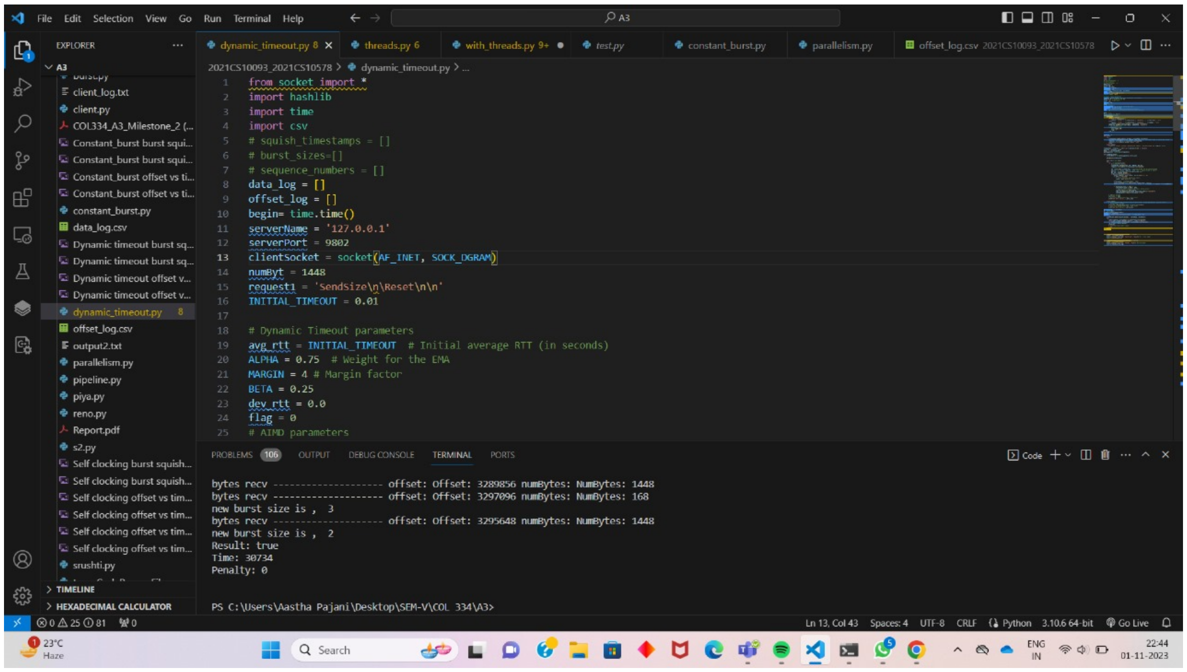
```python
# sequence_numbers = []
data_log = []
offset_log = []
begin= time.time()
serverName = '10.17.7.218'
serverPort = 9802
clientSocket = socket(AF_INET, SOCK_DGRAM)
numByt = 1448
request1 = 'SendSize\n\Reset\n\n'
INITIAL_TIMEOUT = 0.01

# Dynamic Timeout parameters
avg_rtt = INITIAL_TIMEOUT  # Initial average RTT (in seconds)
ALPHA = 0.25  # Weight for the EMA
MARGIN = 4 # Margin factor
BETA = 0.25
```

```
bytes recv ------------------- offset: Offset: 3269584 numBytes: NumBytes: 1448
bytes recv ------------------- offset: Offset: 3266688 numBytes: NumBytes: 1448
new burst size is ,  6
bytes recv ------------------- offset: Offset: 3268136 numBytes: NumBytes: 1448
bytes recv ------------------- offset: Offset: 3269584 numBytes: NumBytes: 1448
bytes recv ------------------- offset: Offset: 3271032 numBytes: NumBytes: 1448
bytes recv ------------------- offset: Offset: 3272480 numBytes: NumBytes: 1448
bytes recv ------------------- offset: Offset: 3271032 numBytes: NumBytes: 1448
bytes recv ------------------- offset: Offset: 3273928 numBytes: NumBytes: 1448
new burst size is ,  7
bytes recv ------------------- offset: Offset: 3275376 numBytes: NumBytes: 1448
bytes recv ------------------- offset: Offset: 3276824 numBytes: NumBytes: 62
bytes recv ------------------- offset: Offset: 3275376 numBytes: NumBytes: 1448
bytes recv ------------------- offset: Offset: 3276824 numBytes: NumBytes: 62
new burst size is ,  3
Result: true
Time: 28406
Penalty: 75

PS C:\Users\disha\Desktop\A3_COL334\2021CS10578_2021CS10093>
```

Figure 13: Time Taken and Penalty Incurred for 10.17.7.218 using DT



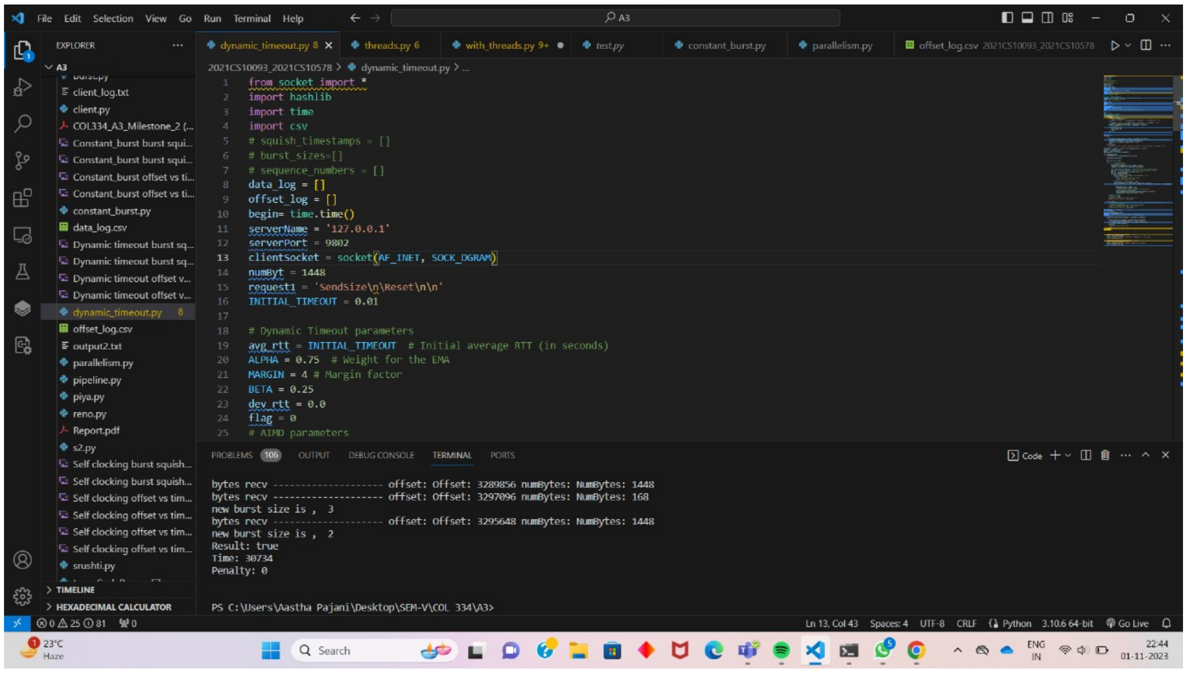
```python
from socket import *
import hashlib
import time
import csv
# squish_timestamps = []
# burst_sizes=[]
# sequence_numbers = []
data_log = []
offset_log = []
begin= time.time()
serverName = '127.0.0.1'
serverPort = 9802
clientSocket = socket(AF_INET, SOCK_DGRAM)
numByt = 1448
request1 = 'SendSize\n\Reset\n\n'
INITIAL_TIMEOUT = 0.01

# Dynamic Timeout parameters
avg_rtt = INITIAL_TIMEOUT  # Initial average RTT (in seconds)
ALPHA = 0.75  # Weight for the EMA
MARGIN = 4 # Margin factor
BETA = 0.25
dev_rtt = 0.0
flag = 0
# AIMD parameters
```

```
bytes recv ------------------- offset: Offset: 3289856 numBytes: NumBytes: 1448
bytes recv ------------------- offset: Offset: 3297096 numBytes: NumBytes: 168
new burst size is ,  3
bytes recv ------------------- offset: Offset: 3295648 numBytes: NumBytes: 1448
new burst size is ,  2
Result: true
Time: 30734
Penalty: 0

PS C:\Users\Aastha Pajani\Desktop\SEM-V\COL 334\A3>
```

Figure 14: Time Taken and Penalty Incurred for Local server using DT



```python
from socket import *
import hashlib
import time
import csv
# squish_timestamps = []
# burst_sizes=[]
# sequence_numbers = []
data_log = []
offset_log = []
begin= time.time()
serverName = '127.0.0.1'
serverPort = 9802
clientSocket = socket(AF_INET, SOCK_DGRAM)
numByt = 1448
request1 = 'SendSize\n\Reset\n\n'
INITIAL_TIMEOUT = 0.01

# Dynamic Timeout parameters
avg_rtt = INITIAL_TIMEOUT  # Initial average RTT (in seconds)
ALPHA = 0.75  # Weight for the EMA
MARGIN = 4 # Margin factor
BETA = 0.25
dev_rtt = 0.0
flag = 0
# AIMD parameters
```

```
bytes recv -------------------- offset: Offset: 3289856 numBytes: NumBytes: 1448
bytes recv -------------------- offset: Offset: 3297096 numBytes: NumBytes: 168
new burst size is ,  3
bytes recv -------------------- offset: Offset: 3295648 numBytes: NumBytes: 1448
new burst size is ,  2
Result: true
Time: 30734
Penalty: 0

PS C:\Users\Aastha Pajani\Desktop\SEM-V\COL 334\A3>
```

Figure 15: Time Taken and Penalty Incurred for Local server using DT