ASSIGNMENT REPORT

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Under the Guidance of:

Prof. Aaditeshwar Seth
Assot. Professor

CSE Department

Submitted by:

Karan Agrawal (2023MCS2479)
Annanya Mehta (2023MCS2484)

INDIAN INSTITUTE OF TECHNOLOGY, DELHI

Hauz Khas, New Delhi -110016, India

Detailed Report on the Provided Python Code: Client-Server Communication

Introduction

This report provides a comprehensive analysis of a Python script designed to function as a client in a client-server communication system. The primary goal of this script is to communicate with a remote server using the User Datagram Protocol (UDP). The client's responsibilities encompass retrieving data from the server in smaller segments, assembling these segments into a complete dataset, calculating an MD5 hash for data integrity checking, and submitting both the hash and data back to the server. Additionally, the script offers a feature to plot request and reply times along with corresponding offsets using the Matplotlib library. To gain a better understanding of this script, we will break it down into various sections.

Code Overview

Server Information

The script begins by defining essential server information, including the server's IP address (SERVER_HOST), port number (SERVER_PORT), and a timeout duration for socket operations (timeOut). These parameters are crucial for establishing a connection with the server.

Server Host: "10.17.7.218"
Server Port: 9802
Timeout: 0.01 seconds

Lists for Request and Reply Time & Offset

Timing and Offset Lists:
The script initializes several lists to record and analyze timestamps and offsets during client-server communication:

Request Time: Records timestamps when a request is initiated.
Reply Time: Records timestamps for server reply reception.
Request Offset: Records the offset when a request is made to track the requested data segment's position.
Reply Offset: Records the offset when a reply is received to analyze each segment's position in the final dataset.

These lists play a crucial role in understanding and visualizing the timing and sequencing of client-server interaction

Change Timeout Function

The change_timeOut function dynamically adjusts the timeout duration based on the client's recent interactions with the server. It considers the round-trip time (RTT) and updates the timeout value, ensuring the client can handle the server's responses within a reasonable timeframe.

Change Window Size Function:

The change_windowSize function plays a critical role in dynamically adjusting the window size used in client-server communication. The window size determines how many data segments the client can request and receive from the server simultaneously. This function takes into account several factors to determine the appropriate window size.

Parameters:

windowSize: The current window size.

requestRcvd: The number of requests received by the client.

squished: A flag indicating whether data segments have been squished (compressed) during transmission.

Function Explanation:

Global Threshold: The function accesses the global threshold variable. This threshold serves as a reference value and plays a crucial role in determining the appropriate window size. The threshold is initially set to a default value of 8.

Condition: windowSize <= requestRcvd + 1: The function first checks whether the current window size is less than or equal to the number of requests received plus one. This condition implies that the client is falling behind in processing received data segments. In such a scenario, the function needs to make adjustments to the window size to keep up with the incoming data.

Subcondition: windowSize <= threshold: If the current window size is less than or equal to the threshold, the function increases the window size. It does so by doubling the window size (up to a maximum of 16). This action is taken to accelerate data retrieval when the client is behind in processing.

Subcondition: windowSize > threshold: If the current window size exceeds the threshold, the function still increases the window size but by adding a fixed value of 2 (up to a maximum of 16). This adjustment aims to ensure steady progress in retrieving data segments.

Resetting the Window Size: In scenarios where the client is not falling behind (i.e., the requestRcvd count is satisfactory), the function reduces the window size by dividing it by 2. This reduction helps optimize the window size to match the current communication conditions.

Threshold Adjustment: The threshold value is then updated to half of the current window size. This adjustment helps regulate the behavior of the function in future iterations.

Handling Data Squishing: The function also considers whether data segments have been squished (compressed) during transmission. If squishing is detected (squished == 1), the function resets the window size to 1. This adjustment is applied to efficiently handle squished data segments. If no squishing is detected, the window size remains as calculated.

Connect Server Function

The connect_server function is a central component of the client-server communication process. It orchestrates the various steps required for data retrieval from the server. Let's break down each part of this function:

Setting Global Variables:

timeOut: The function accesses the global timeOut variable, which represents the timeout duration for socket operations.

predicted_rtt: This variable tracks the predicted round-trip time (RTT) for communication.

Creating a UDP Socket:

The function initializes a UDP socket using socket.socket(**socket.AF_INET, socket.SOCK_DGRAM**). This socket will be used for communication between the client and the server.

The socket's timeout is set to timeOut to handle potential timeouts during socket operations.

Sending "SendSize" Command:

The function creates a message with the content **"SendSize\nReset\n\n".** This message is sent to the server to inform it that data transfer is expected.

A loop is used to repeatedly send this message to the server until a response is received. In case of a socket timeout, the function prints a **"Sendsize timeout error."**

Data Size Calculation:

Once a response is received from the server, the function extracts the data size from the response message. The response format is assumed to include "SendSize" followed by the data size.

The data size is used to calculate the number of requests required to retrieve the entire dataset. It's also used to initialize data structures for storing the received data segments.

Data Retrieval Loop Analysis:

The section of code you provided represents the core of the data retrieval process within the connect_server function. This loop is responsible for efficiently requesting and receiving data segments from the server. Let's break down this loop and analyze its various components:

Loop Initialization:

while len(data_remains) > 0:: The loop continues as long as there are remaining data segments to be retrieved. The loop iterates over these segments until all data is collected.

Request Adjustment and Time Management:

cnt and **cnt2** are counters used to manage the timing and request intervals.

The loop begins by incrementing these counters. If squished is not detected (i.e., data segments are not compressed), the loop adjusts the request interval based on the following conditions:

If **cnt** exceeds 20 and the difference between the previous and current data remains (prelength) is less than or equal to 10, the request interval is increased by a small increment (0.000001). This adjustment likely indicates that the data retrieval process is slower than expected.

If **cnt2** is less than or equal to 20 and the difference between the previous and current data remains (**prelength2**) is greater than 200, the request interval is decreased by the same small increment. This adjustment suggests that data retrieval is faster than expected.

In case data is squished (**squished == 1**), the loop pauses for a fixed time of 0.10 seconds and significantly increases the request interval by multiplying it by 10. This handling is intended to manage the retrieval process effectively when compressed data segments are encountered.

Request Transmission:

The loop proceeds to request data segments from the server. It calculates the window size (**windowSize**) and iterates through the data segments based on this size.

For each data segment requested, it records the request time and offset, then sends the request message to the server.

Time Tracking:

The code records the time t1 when the last request in the current window was sent. This timestamp is used to calculate the destination transmission time.

The current RTT (Round-Trip Time) is initialized to a default value of 0.01 seconds (**currentRtt=0.01**).

Reply Reception:

The loop then attempts to receive server responses for the requested data segments. It accounts for the possibility of timeouts.

If a response is successfully received, it records the reply time and offset, as well as the data itself. If data segments are compressed (squished), it handles this condition accordingly.

Window Size and Time Adjustments:

After processing a window of data segments, the code adjusts the window size (windowSize) based on the number of requests received (requestRcvd) and whether data squishing was detected.

The code also records the burst size (window size) and the time at which it was applied.

Data Remainder and RTT Adjustment:

The loop maintains a list of remaining data segments (data_remains) and updates it as segments are successfully received.

It calculates the current RTT (currentRtt) based on the time difference between request and reply.

Timeout Handling:

If any socket timeouts occur during the process, the loop handles them, although the specific code for adjusting the timeout is not shown in this snippet.

Data Assembly and Integrity Checking:

The received data segments are assembled into a complete dataset.

An MD5 hash is computed for the entire dataset to ensure data integrity.

Submission to the Server:

The computed MD5 hash and the data are submitted to the server using a "Submit" command.

Similar to the "SendSize" command, the function sends the submission message in a loop until

a response is received. In case of a timeout, it prints a "Submission timeout error."

Plotting Request and Reply Times:

Matplotlib is used to create two plots:

"**Figure 1:** Sequence-number trace" displays the request and reply times along with offsets, providing a visual representation of the communication timeline.

"**Figure 2:** Burst-size-trace" shows the evolution of the window size over time, offering insights into the efficiency of data retrieval.

This connect_server function serves as the core of the client's interaction with the server, managing data retrieval, assembly, integrity checking, submission, and performance analysis.

Script Execution

The script's execution starts with the main function, which serves as the entry point. The conditional block if __name__ == '__main__': ensures that the main function is executed when the script is run. This function, in turn, initiates the client-server communication process.

Running Time Graphs

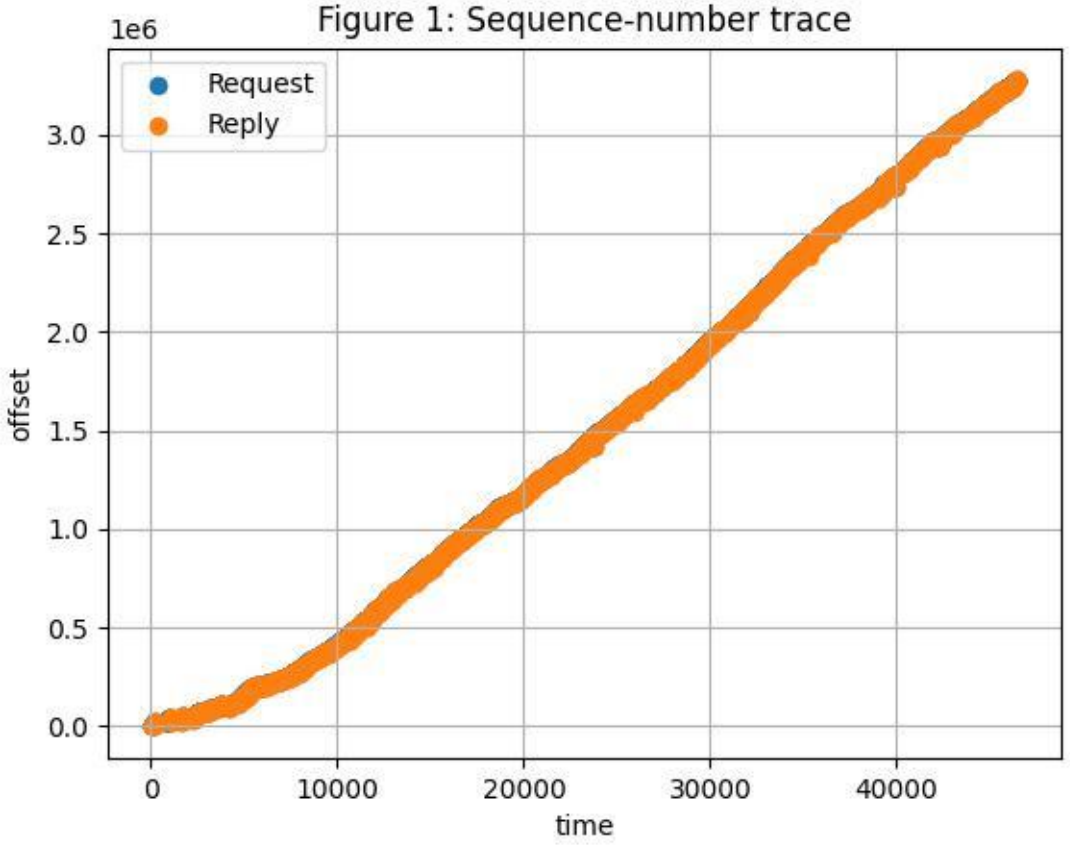
Figure 1: Sequence-number trace
1e6
Request
Reply
3.0
2.5
2.0
1.5
1.0
0.5
0.0
offset
0
10000
20000
30000
40000
time

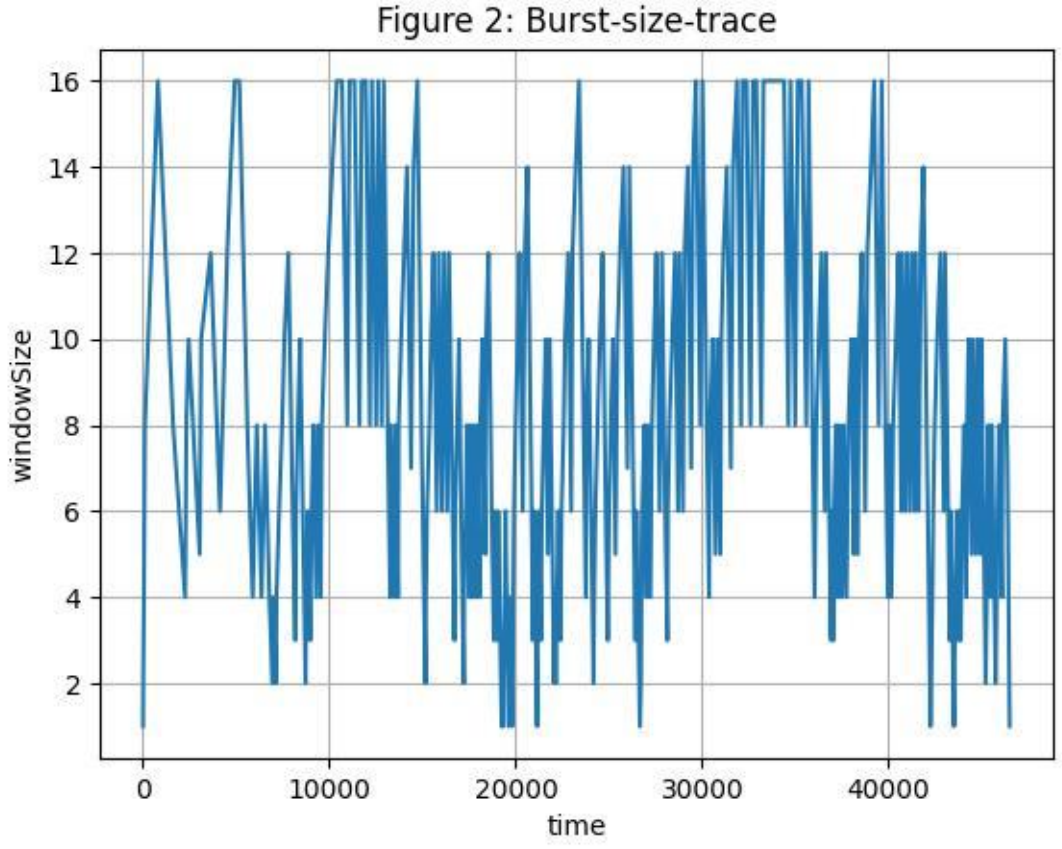
Figure 2: Burst-size-trace
16
14
12
10
8
6
4
2
windowSize
0
10000
20000
30000
40000
time

sjdkljfs