Testing Different Data Sending Strategies in Multithreaded Socket Programs

Aiden Koknat, Kevin Patel

CS 371: Introduction to Computer Networking

Dr. Hana Khamfroush

Date: December 4, 2022

1. ABSTRACT –

We were able to create a client-server network with 2 different clients who are capable of the following tasks: CONNECT, UPLOAD, DOWNLOAD, DELETE, and DIR. This report will entail our results of various tests upon this client-server network by using text files, video files, and audio files.

2. INTRODUCTION & MOTIVATION –

The goal of this project is to create a client-server network where multiple clients can complete the appropriate task given by the user. These tasks consist of CONNECT, UPLOAD, DOWNLOAD, DELETE, and DIR. The clients should send and receive messages and files to and from the server, which will handle communication between different clients. The application should use multithreaded socket programming to establish and manage each connection.

3. DESIGN & IMPLEMENTATION –

To achieve a secure application that could safely manage and communicate to multiple clients, a multithreaded server was set up to store different threads that corresponded to each client's connection. Upon launch, the server creates storage for future client information, threads, and files. See the Python code snippet below:

```
if __name__ == "__main__":
    toBeDeleted = []  # array of file names that will be deleted after temporary use
    ThreadCount = 0  # placeholder for incrementing amount of client threads

    # Setting Up Client Threads
    clients = {}
```

```
    # Creating Server
    host = '127.0.0.1'  # local is 127.0.0.1
    port = 2004
    s = socket(AF_INET, SOCK_STREAM)
    s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
    s.bind((host, port))
    print('Server is set up!')
    print('Socket is listening..')
    s.listen(5)

    ACCEPT_THREAD = Thread(target=incoming_connections(toBeDeleted))
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    s.close()
```

Upon receiving the message from a client to connect, it runs the function incoming_connections. This function establishes the wireless connection and records the client's address for future use, then runs the function single_client (the bulk of the server code) to maintain the client requests. See code snippet below:

```
def incoming_connections(to_be_deleted):  # Handles each Client
    while True:
        client, addr = s.accept()
        print(f'A client has connected {addr}')
        thread_count = int(addr[1])
        Thread(target=single_client, args=(client, thread_count, to_be_deleted)).start()
```

The majority of the functionality of the program lies in how the messages are packaged. Messages are typically packaged (using pickle) with commands and file names along with the file data so the clients/server can properly understand how to handle them. For example, if the server were to read "UPLOAD" as the first index of a received message, it would then record the second index of the message as the incoming file information and the third index as the file name, all to be saved onto the

server (unless the fourth index tells it to be deleted after its use). See code sample from the single_client function of server.py below for reference:

```
# receivedCommand = receivedToken[0], which is the first index of the incoming message.
elif receivedCommand == "UPLOAD":
    print('Received data from Client')
    write_file = open("Server/" + receivedToken[2], "w")  # opens file for writing/saving sent file
    write_file.write(receivedToken[1])  # writes/saves file received
    print("File written.")
    if receivedToken[3] == "server_request":
        to_be_deleted.append("Server/" + receivedToken[2])
    write_file.close()
    print("File saved and closed.")
```

The client side of the application is fairly simple and light-weight. Upon launching, users are greeted with a menu of options:

```
--------------------------------------------------------------------------------------------
Welcome Client! Enter an Option:
CONNECT [host address] [server port] to connect to server: host is 127.0.0.1, port is 2004
UPLOAD fileName
DOWNLOAD fileName
DELETE fileName
DIR
--------------------------------------------------------------------------------------------
Input Command with arguments: CONNECT 127.0.0.1 2004
Connecting
Waiting for connection response...
```

This is where clients can manage their data, or connect and communicate with the server to upload or download files.

Client input is broken into tokens, where the command is put through a case of options. These options will then determine how the message that will be sent to the server will be built. Typically, the message is formatted in the order of ([command], [file data], [file name]).

```
tokens = input("Input Command with arguments: ").split(" ")
command = str(tokens[0])
if len(tokens) > 3:
    print("only one command and two arguments max please!")
    return 0
else:
    if command == "CONNECT":
        if is_connected:
            print("You're already connected, silly.")
            input("Click 'Enter' to continue.")
            return 0
        else:
            return tokens
    elif command == "UPLOAD":
        if is_connected:
            upload_file(tokens[1], "client_request")
            input("Click 'Enter' to continue.")
            return 0
        else:
            print("Please connect to a server first.")
            input("Click 'Enter' to continue.")
            return 0
    elif command == "DOWNLOAD":
        if is_connected:
            return tokens
        else:
            print("Please connect to a server first.")
            input("Click 'Enter' to continue.")
            return 0
```

4. EXPERIMENTS –

4.1 Experiment Overview: Using the above files, a timer will be set when the client requests for the files to be downloaded, and will be stopped when it receives both the files. As sending speed can vary from download to download (even if files and strategy are held constant), multiple experiments will be conducted for each strategy and will be averaged for a more accurate download speed representation. All strategies have a uniform period of 1 second for the server to receive the other clients file before sending it to the initial client.

Files used:

- serverFile.txt (will be sent from server)
- client2file.txt (will be sent from Client 2 to server, then to Client 1)

While the server is capable of sending large amounts of data such as audio and video files, text files were used so that the time saved from less of a download time could be spent on recording more trials.

4.2 Scenario 2: Files are sent with client and server devices housed in the same room.

4.2.1 Strategy 2-1: Files are sent one by one, and the server waits for an acknowledgement message before sending the second file.

Trial 1:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.20008635520935059 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.2243928909301758 seconds.
Sent acknowledgement message
```

Trial 2:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.9708490371704102 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.271618127822876 seconds.
Sent acknowledgement message
```

Trial 3:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.07559013366699219 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.0961425304412842 seconds.
Sent acknowledgement message
```

Trial 4:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.032280921936035156 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.141615867614746 seconds.
Sent acknowledgement message
```

Trial 5:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.06456565856933594 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.124612808227539 seconds.
Sent acknowledgement message
```

Average time: 1.171672 seconds

4.2.2 Strategy 2-2:

File from the second client is uploaded to the server first, then merged with file two.

Trial 1:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.1252448558807373 seconds.
Sent acknowledgement message
```

Trial 2:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0749013423919678 seconds.
Sent acknowledgement message
```

Trial 3:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0388996601104736 seconds.
Sent acknowledgement message
```

Trial 4:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.072620153427124 seconds.
Sent acknowledgement message
```

Trial 5:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0560967922210693 seconds.
Sent acknowledgement message
```

Average Speed: 1.073548 seconds

4.2.3 Strategy 2-3:

Server waits for files from Client 2 and sends both files back to back without merge.

Trial 1:

```
Input Command with arguments: DOWNLOAD serverFIle.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0170416831970215 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.0325419902801514 seconds.
Sent acknowledgement message
```

Trial 2:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0211822986602783 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.0211822986602783 seconds.
Sent acknowledgement message
```

Trial 3:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.9997661113739014 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.015233039855957 seconds.
Sent acknowledgement message
```

Trial 4:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.031177043914795 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.031177043914795 seconds.
Sent acknowledgement message
```

Trial 5:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0153932571411133 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.077876329421997 seconds.
Sent acknowledgement message
```

Average Speed: 1.035598

4.3 Scenario 3: Similar to Scenario 2, but Server and Client devices are farther away, with the clients' location at the Johnson Recreation Center and the server's location: Student Center.

4.3.1 Strategy 3-1:

Files are sent one by one, and the server waits for an acknowledgement message before sending the second file.

Trial 1:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.09821105003356934 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.1052751541137695 seconds.
Sent acknowledgement message
```

Trial 2:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.015452146530151367 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.1326861381530762 seconds.
Sent acknowledgement message
```

Trial 3:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.07946991920471191 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.1031765937805176 seconds.
Sent acknowledgement message
```

Trial 4:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.012026786804199219 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.0450751781463623 seconds.
Sent acknowledgement message
```

Trial 5:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 0.06334400177001953 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.1110115051269531 seconds.
Sent acknowledgement message
```

Average Speed = 1.09944

4.3.2 Strategy 3-2:

File from the second client is uploaded to the server first, then merged with file two.

Trial 1:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0088469982147217 seconds.
Sent acknowledgement message
```

Trial 2:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0461475849151611 seconds.
Sent acknowledgement message
```

Trial 3:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.026329517364502 seconds.
Sent acknowledgement message
```

Trial 4:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0846149921417236 seconds.
Sent acknowledgement message
```

Trial 5:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0674409866333008 seconds.
Sent acknowledgement message
```

Average Speed = 1.0466752

4.3.3 Strategy 3-3:

Server waits for files from Client 2 and sends both files back to back without merge.

Trial 1:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0676944255828857 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.0768663883209229 seconds.
Sent acknowledgement message
```

Trial 2:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.1305060386657715 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.1489672660827637 seconds.
Sent acknowledgement message
```

Trial 3:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0279695987701416 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.0397121906280518 seconds.
Sent acknowledgement message
```

Trial 4:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.167499303817749 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.1774697303771973 seconds.
Sent acknowledgement message
```

Trial 5:

```
Input Command with arguments: DOWNLOAD serverFile.txt client2file.txt
sent data!
Waiting... (enter 'quit' to quit early)
Saved and closed new file.
Time to receieve file since request: 1.0407025814056396 seconds.
Sent acknowledgement message
Saved and closed new file.
Time to receieve file since request: 1.0495753288269043 seconds.
Sent acknowledgement message
```

Average Speed: 1.0985178 seconds

4.4 Results –

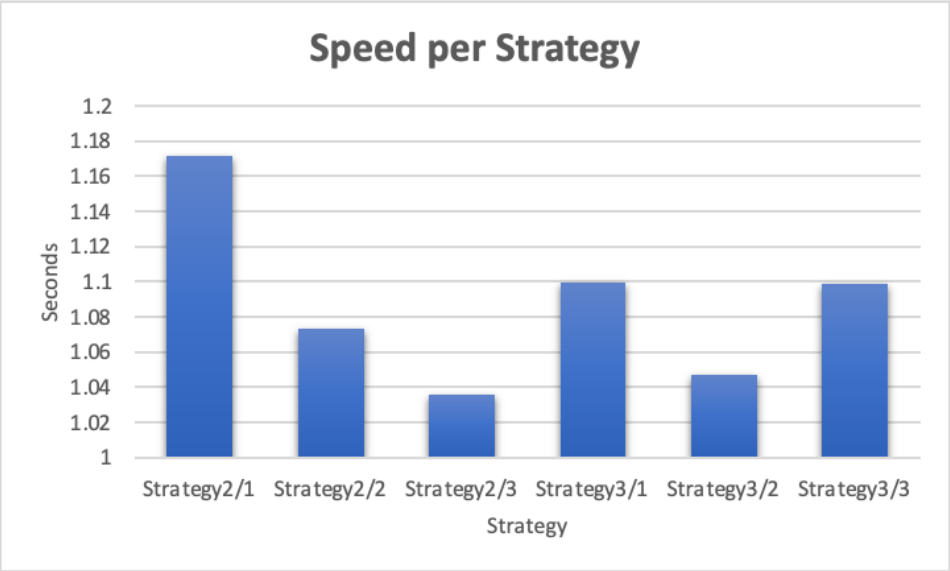

Table of Speeds in Seconds:

Trial #	2-1	2-2	2-3	3-1	3-2	3-3
1	1.22439	1.12524	1.03254	1.10527	1.008846	1.076866
2	1.27161	1.0749	1.02118	1.13268	1.046147	1.148967
3	1.09614	1.03889	1.01523	1.10317	1.026329	1.039712
4	1.14161	1.07262	1.03117	1.04507	1.084614	1.177469
5	1.12461	1.05609	1.07787	1.11101	1.06744	1.049575
Average	**1.171672**	**1.073548**	**1.035598**	**1.09944**	**1.0466752**	**1.0985178**

5. CONCLUSION –

Throughout the process it was found that Strategy 2-3 was able to produce the fastest results at averaging 1.035598 seconds. This was able to be accomplished by removing the need to merge the two files before sending them back. The process for merging the two files in Strategy 2-2 seemed to slow the overall process when compared to Strategy 2-3.

While testing, it was discovered that Strategy 2-1 takes the longest time because it is a one by one file transfer. Connections of nature take extra time to get back the ACK from client/server causing a higher time average. Strategy 2-2 involved the merging process of the files, which caused slower transfer speeds. Strategy 2-3 was faster and produced the lowest results because it skipped the merging process from Strategy 2-2 and sent both files back to back without merging causing less chance of congestion and not overwhelming the server with a large file size. Although Strategy 2-3 produced the fastest results, it was only during a local host connection.

When moved further away, such as the Johnson Center and Student center, we noticed that Strategy 3-2 produced the fastest results. We believe this was caused by the fact that over a longer distance, the files were received faster if merged and sent together instead of back to back. Since more time is taken on sending data longer distances, reducing the amount of trips vastly decreases the amount of time.