Computer Networks Lab Assignment 3

Please use this link to view our code, ReadMe file and Makefile

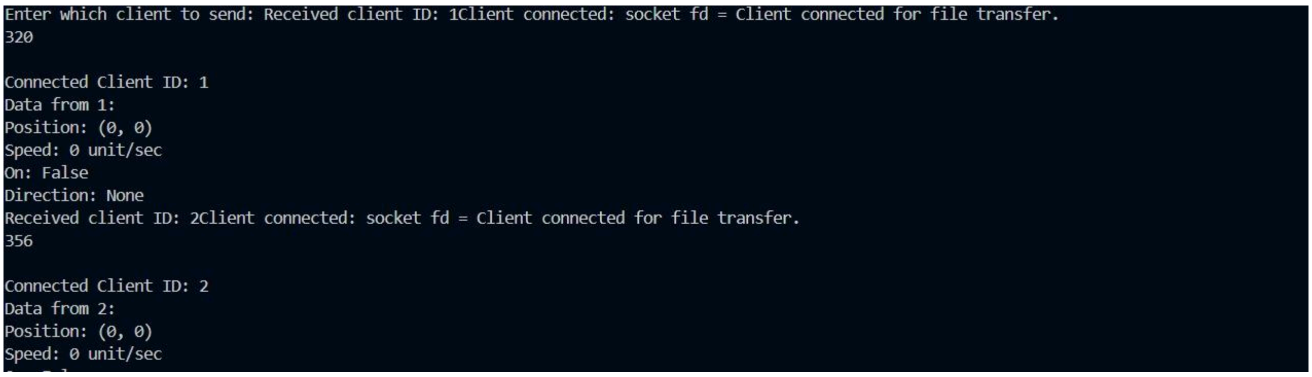
https://github.com/rambansal59/Autonomous-Drone-Monitoring-and-Control-System

Made by - Tanmay Mittal- 220101099
Ram Bansal- 220101085
Priyanshu Pratyay- 220123064
Simon Lalremsiama- 220101095

Q1

In this question , each client denotes a drone , with its unique specific id , and a server which receives constant updates from each drone , and it also sends various commands to drones .

(i) Here we are demonstrating multiple clients which are connected to the server.

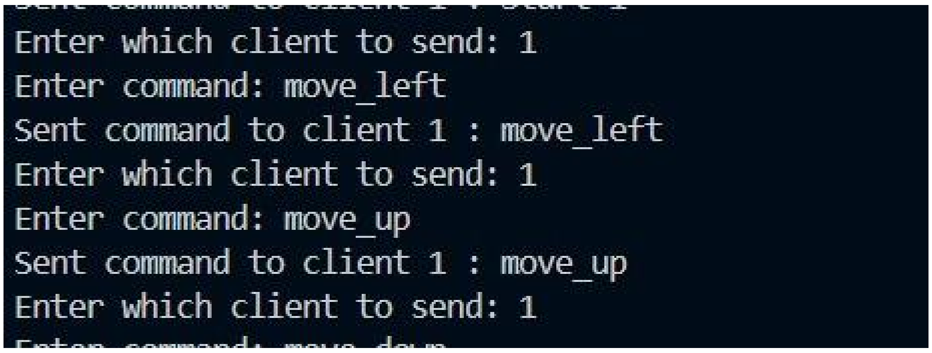


```
Enter which client to send: Received client ID: 1Client connected: socket fd = Client connected for file transfer.
320

Connected Client ID: 1
Data from 1:
Position: (0, 0)
Speed: 0 unit/sec
On: False
Direction: None
Received client ID: 2Client connected: socket fd = Client connected for file transfer.
356

Connected Client ID: 2
Data from 2:
Position: (0, 0)
Speed: 0 unit/sec
```

Here, *struct Movement{*
    int x;
    int y;
    int speed;
    bool On;
    string direction;
    Movement():x(0),y(0),speed(0),On(false),direction("None"){};
};

Where (x,y) denotes coordinates of drones, speed denotes current speed of drones , bool On denotes whether the drone is on or off , direction denotes which direction the drone is moving .

(ii)

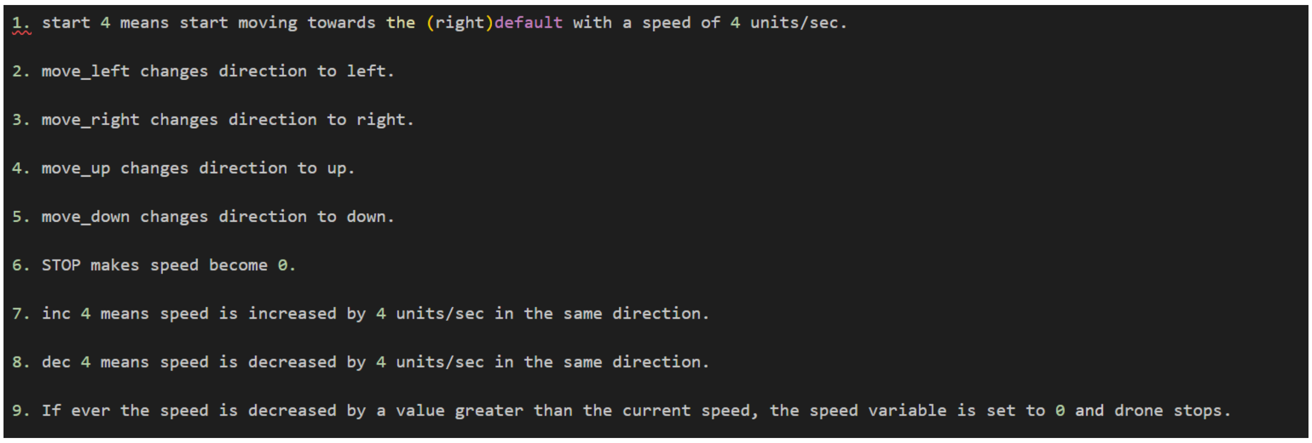


```
Enter which client to send: 1
Enter command: move_left
Sent command to client 1 : move_left
Enter which client to send: 1
Enter command: move_up
Sent command to client 1 : move_up
Enter which client to send: 1
```

Server has to choose which drone to send control data, this is achieved by typing the client id on the terminal, then the command is entered.
The control commands are sent by the server to the drone using the UDP protocol for **fast transfer** of information. There may be loss of packets and information may fail to reach the drone as UDP is not reliable.

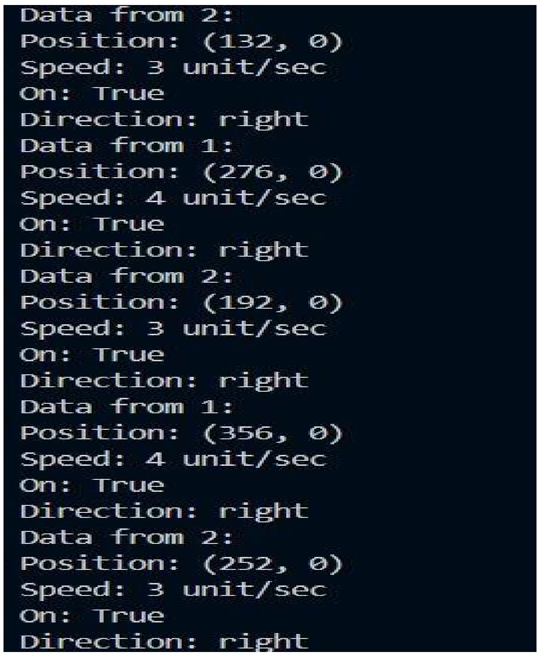
Note:- Directions like left, right, up,down are **absolute directions**. Imagine a 2-D space
Moving up increments y coordinate, down decreases y coordinate, right increases x coordinate and left decreases x coordinate.

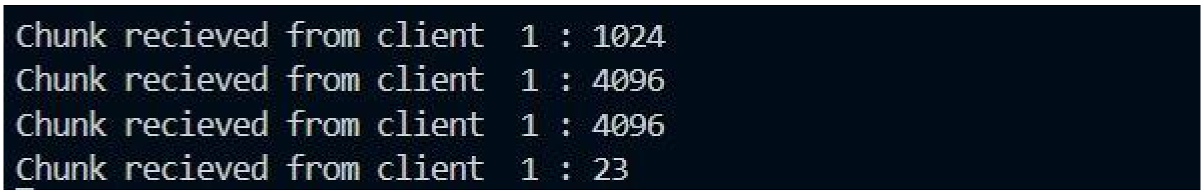
CONTROL COMMANDS



```
1. start 4 means start moving towards the (right)default with a speed of 4 units/sec.

2. move_left changes direction to left.

3. move_right changes direction to right.

4. move_up changes direction to up.

5. move_down changes direction to down.

6. STOP makes speed become 0.

7. inc 4 means speed is increased by 4 units/sec in the same direction.

8. dec 4 means speed is decreased by 4 units/sec in the same direction.

9. If ever the speed is decreased by a value greater than the current speed, the speed variable is set to 0 and drone stops.
```

(iii) TCP protocol is used to **reliably** send speed,location and status(ON/OFF) of the drones periodically every 20 seconds.


```
Data from 2:
Position: (132, 0)
Speed: 3 unit/sec
On: True
Direction: right
Data from 1:
Position: (276, 0)
Speed: 4 unit/sec
On: True
Direction: right
Data from 2:
Position: (192, 0)
Speed: 3 unit/sec
On: True
Direction: right
Data from 1:
Position: (356, 0)
Speed: 4 unit/sec
On: True
Direction: right
Data from 2:
Position: (252, 0)
Speed: 3 unit/sec
On: True
Direction: right
```

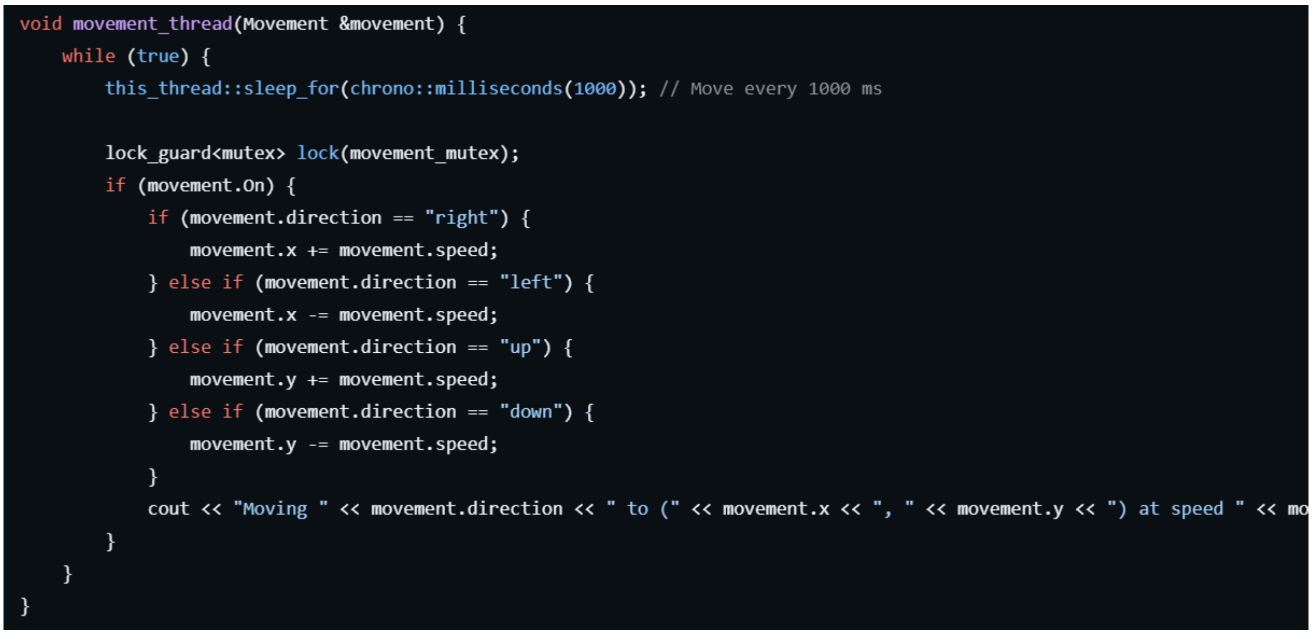
(iv) 
```
Chunk recieved from client  1 : 1024
Chunk recieved from client  1 : 4096
Chunk recieved from client  1 : 4096
Chunk recieved from client  1 : 23
```

File transfer is executed using QUIC protocol to reliably transfer data in a time efficient manner. Data is broken down into chunks before sending, the size of chunks is mentioned in the above screenshot

FUNCTION DESCRIPTION

CLIENT.CPP

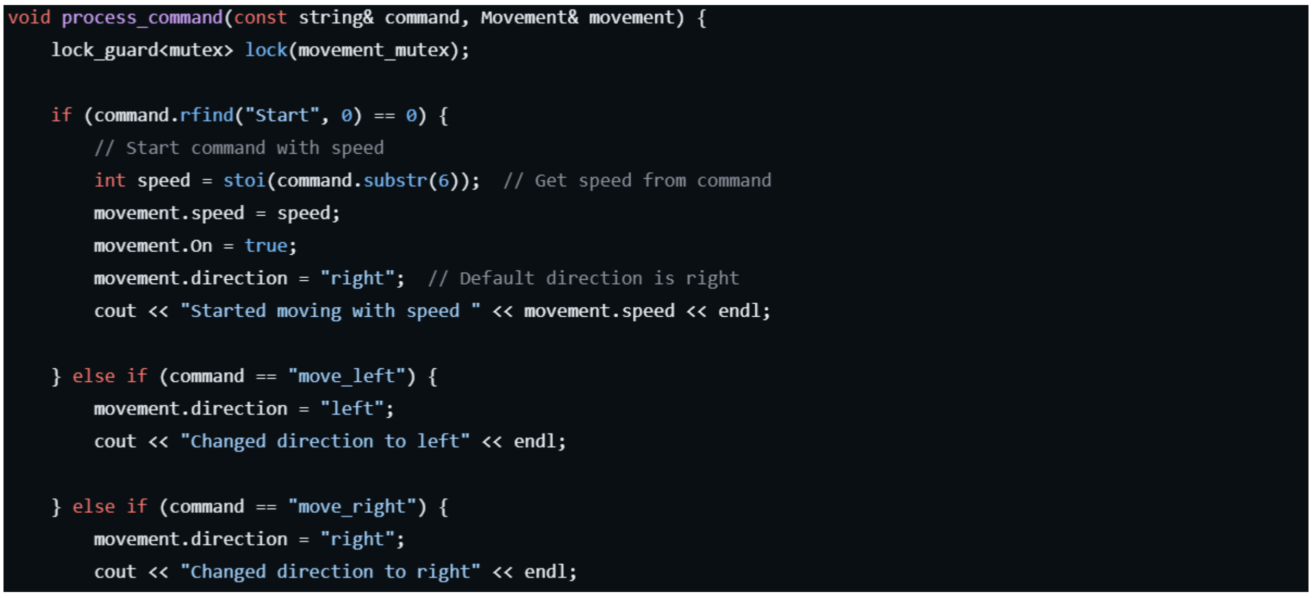
1.



```cpp
void movement_thread(Movement &movement) {
    while (true) {
        this_thread::sleep_for(chrono::milliseconds(1000)); // Move every 1000 ms

        lock_guard<mutex> lock(movement_mutex);
        if (movement.On) {
            if (movement.direction == "right") {
                movement.x += movement.speed;
            } else if (movement.direction == "left") {
                movement.x -= movement.speed;
            } else if (movement.direction == "up") {
                movement.y += movement.speed;
            } else if (movement.direction == "down") {
                movement.y -= movement.speed;
            }
            cout << "Moving " << movement.direction << " to (" << movement.x << ", " << movement.y << ") at speed " << mo
        }
    }
}
```

A thread function that simulates movement based on the `Movement` object's state. It updates the `x` and `y` coordinates based on the direction and speed every 1000 milliseconds. It uses a mutex to ensure thread-safe access to the `Movement` object.

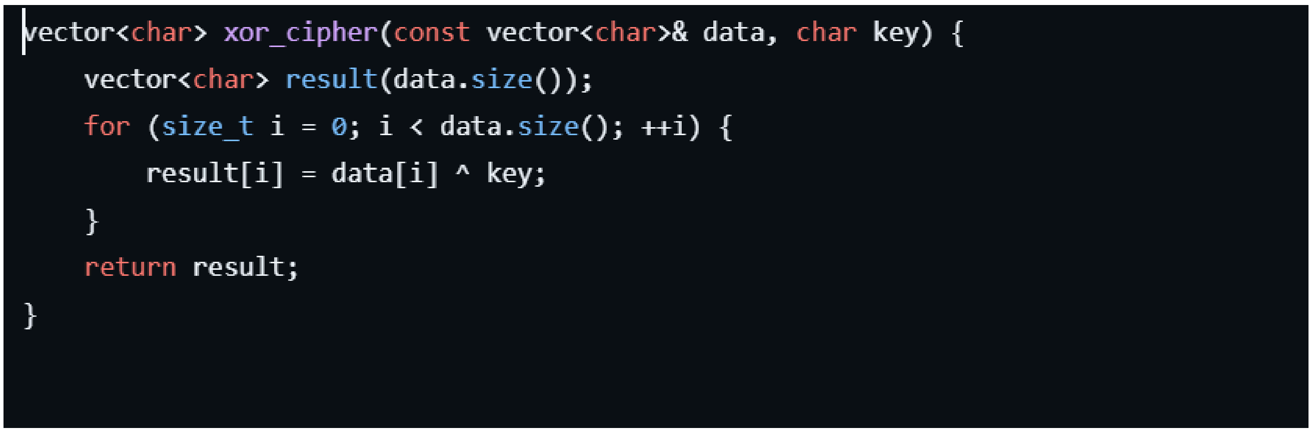
2.



```cpp
void process_command(const string& command, Movement& movement) {
    lock_guard<mutex> lock(movement_mutex);

    if (command.rfind("Start", 0) == 0) {
        // Start command with speed
        int speed = stoi(command.substr(6));  // Get speed from command
        movement.speed = speed;
        movement.On = true;
        movement.direction = "right";  // Default direction is right
        cout << "Started moving with speed " << movement.speed << endl;

    } else if (command == "move_left") {
        movement.direction = "left";
        cout << "Changed direction to left" << endl;

    } else if (command == "move_right") {
        movement.direction = "right";
        cout << "Changed direction to right" << endl;
```

Processes commands received (e.g., "Start", "move_left", "inc", etc.) to update the movement's direction, speed, and state (on/off). It locks the `movement` object using a mutex to ensure thread safety.

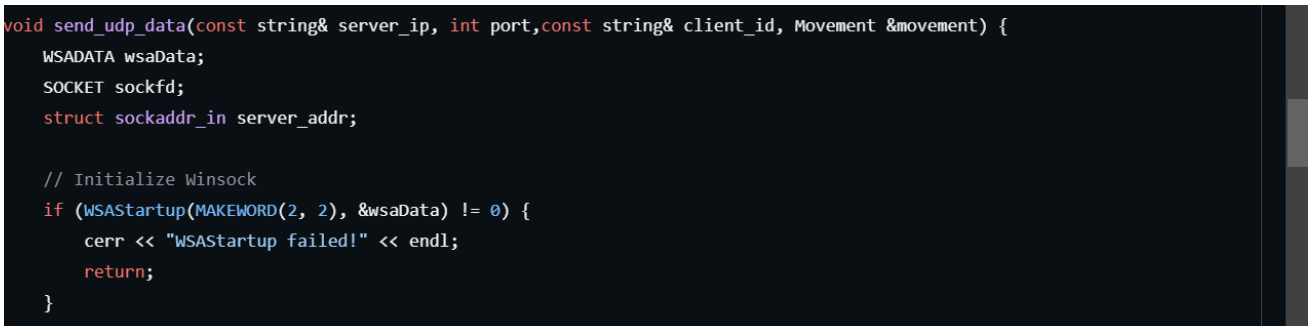
3.



```cpp
vector<char> xor_cipher(const vector<char>& data, char key) {
    vector<char> result(data.size());
    for (size_t i = 0; i < data.size(); ++i) {
        result[i] = data[i] ^ key;
    }
    return result;
}
```

Encrypts or decrypts a vector of characters using a simple XOR cipher. Each character in the input data is XORed with a given key.

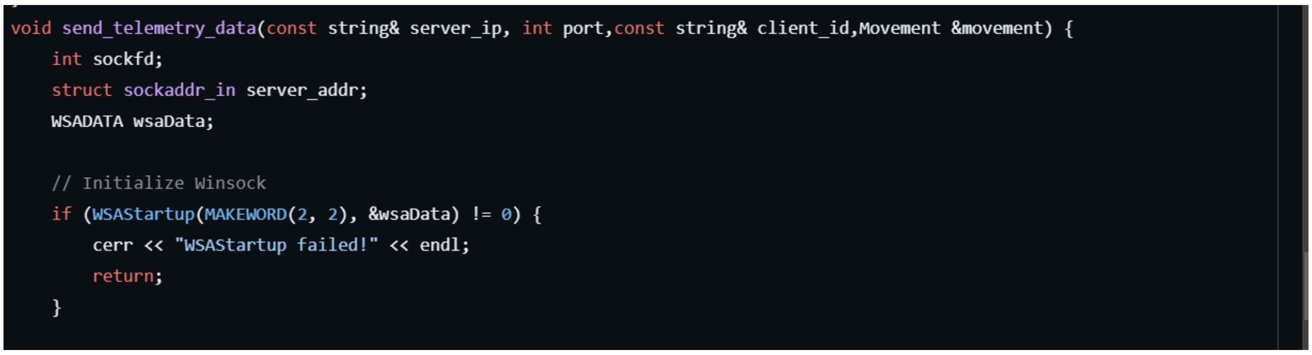
4.



```cpp
void send_udp_data(const string& server_ip, int port,const string& client_id, Movement &movement) {
    WSADATA wsaData;
    SOCKET sockfd;
    struct sockaddr_in server_addr;

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        cerr << "WSAStartup failed!" << endl;
        return;
    }
```

Sends control commands over UDP to a server. The function sends the client ID to the server and then listens for incoming commands, which are processed using `process_command()`. It operates in an infinite loop to receive and process commands continuously.

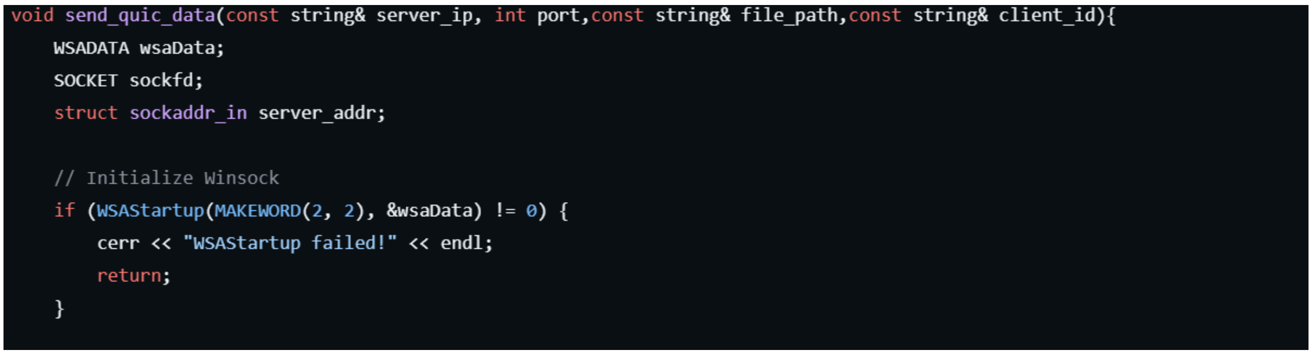
5.



```cpp
void send_telemetry_data(const string& server_ip, int port,const string& client_id,Movement &movement) {
    int sockfd;
    struct sockaddr_in server_addr;
    WSADATA wsaData;

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        cerr << "WSAStartup failed!" << endl;
        return;
    }
```

Sends telemetry data over a TCP connection to a server. It encrypts the serialized `Movement` data using the `xor_cipher()` function and sends it periodically to the server. It also sends the client ID during connection setup.

6.



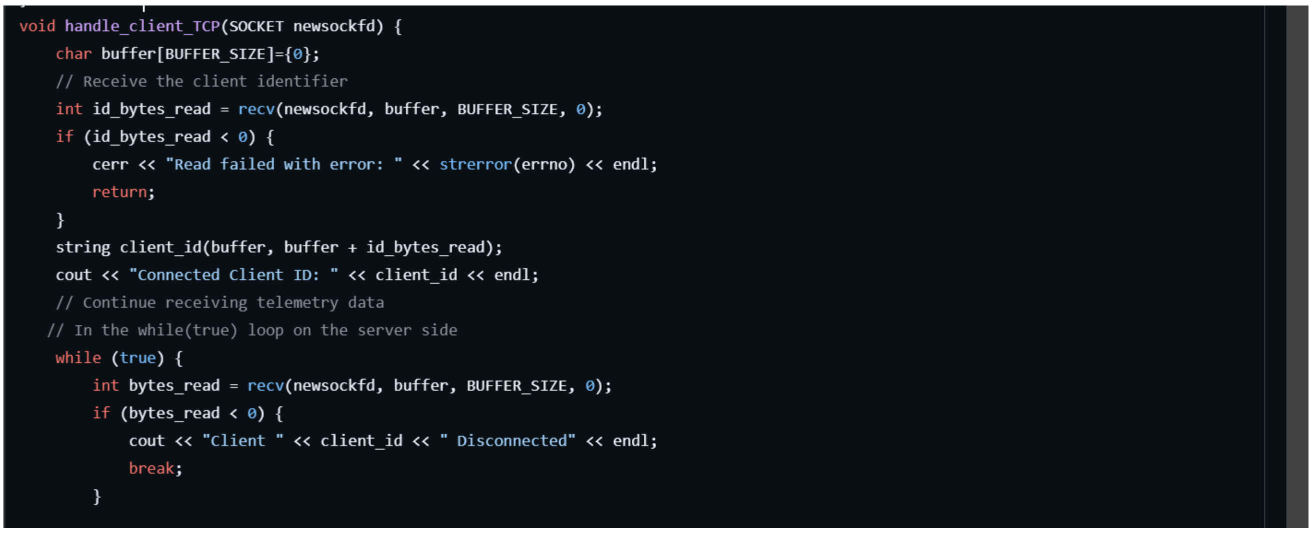
```
void send_quic_data(const string& server_ip, int port,const string& file_path,const string& client_id){
    WSADATA wsaData;
    SOCKET sockfd;
    struct sockaddr_in server_addr;

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        cerr << "WSAStartup failed!" << endl;
        return;
    }
```

Supports fast and reliable and fast transfer of files from drone to server.
In all 3 methods udp, tcp, quic client socket after making successful connection with server, it first sends its client id to server before going in while loop where it can constantly send and receive data in its socket .
This client id helps us differentiate different messages on server side , who actually send the data to it .

SERVER.CPP

Here, in tcp socket , for each server socket associated with distinct client , we map this socket with a seperate thread , so each server client in tcp can work independently .

1.

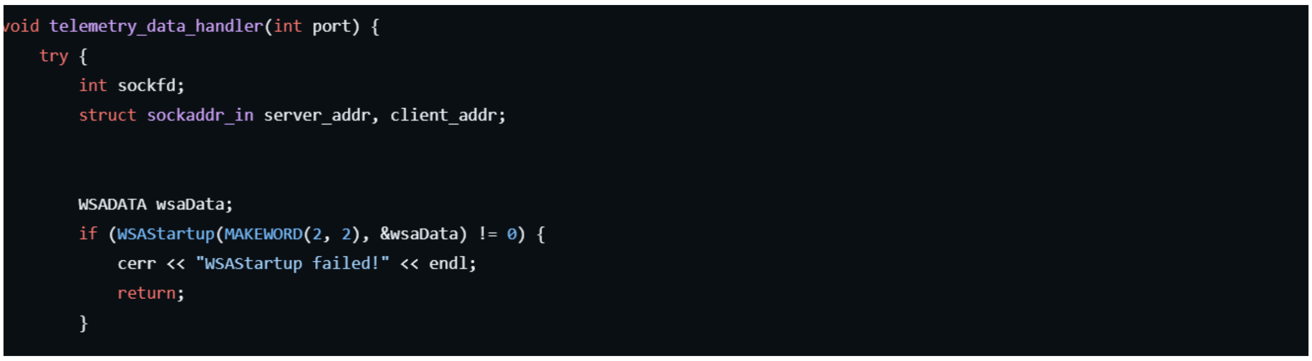


```
void handle_client_TCP(SOCKET newsockfd) {
    char buffer[BUFFER_SIZE]={0};
    // Receive the client identifier
    int id_bytes_read = recv(newsockfd, buffer, BUFFER_SIZE, 0);
    if (id_bytes_read < 0) {
        cerr << "Read failed with error: " << strerror(errno) << endl;
        return;
    }
    string client_id(buffer, buffer + id_bytes_read);
    cout << "Connected Client ID: " << client_id << endl;
    // Continue receiving telemetry data
   // In the while(true) loop on the server side
    while (true) {
        int bytes_read = recv(newsockfd, buffer, BUFFER_SIZE, 0);
        if (bytes_read < 0) {
            cout << "Client " << client_id << " Disconnected" << endl;
            break;
        }
```

Purpose: Manages telemetry data communication over TCP for a connected client.
Details: After establishing a connection, this function receives encrypted telemetry data, decrypts it using the XOR cipher, deserializes the data into a

`Movement` object, and prints the received information (position, speed, on/off status, direction). The function runs continuously in a loop, handling the telemetry data until the client disconnects.

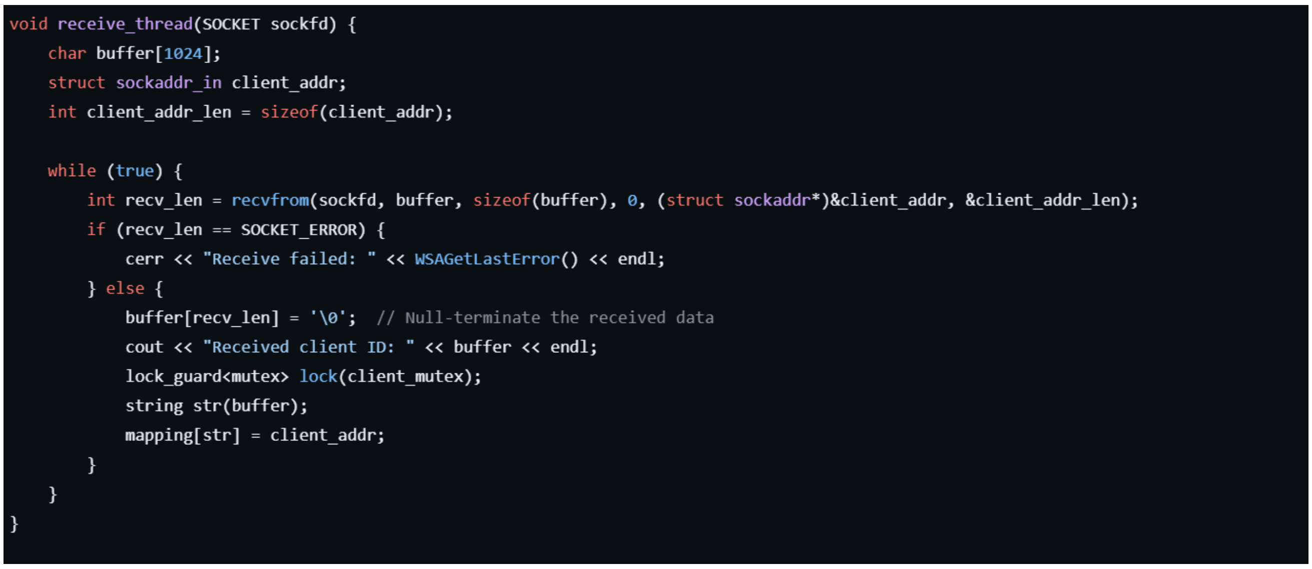
2.



```
oid telemetry_data_handler(int port) {
    try {
        int sockfd;
        struct sockaddr_in server_addr, client_addr;


        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            cerr << "WSAStartup failed!" << endl;
            return;
        }
```

Purpose: Handles incoming telemetry data connections via TCP.
Details: Sets up a TCP server that listens on a specified port (`port`). For each incoming client connection, it creates a new thread to handle the client using `handle_client_TCP`. This allows the server to handle multiple clients concurrently.

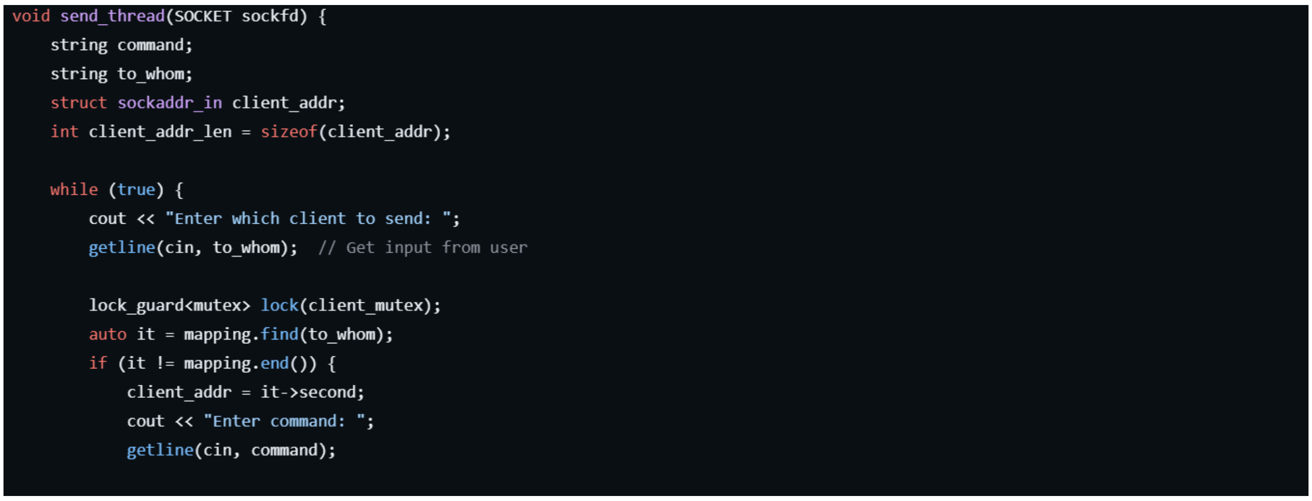
3.



```
void receive_thread(SOCKET sockfd) {
    char buffer[1024];
    struct sockaddr_in client_addr;
    int client_addr_len = sizeof(client_addr);

    while (true) {
        int recv_len = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&client_addr, &client_addr_len);
        if (recv_len == SOCKET_ERROR) {
            cerr << "Receive failed: " << WSAGetLastError() << endl;
        } else {
            buffer[recv_len] = '\0';  // Null-terminate the received data
            cout << "Received client ID: " << buffer << endl;
            lock_guard<mutex> lock(client_mutex);
            string str(buffer);
            mapping[str] = client_addr;
        }
    }
}
```

Purpose: Handles incoming control messages via UDP.
Details: Receives client identifiers sent via UDP and stores the mapping between the client ID and their address in a global map (`mapping`). This mapping is used to route control messages to the correct client.

4.

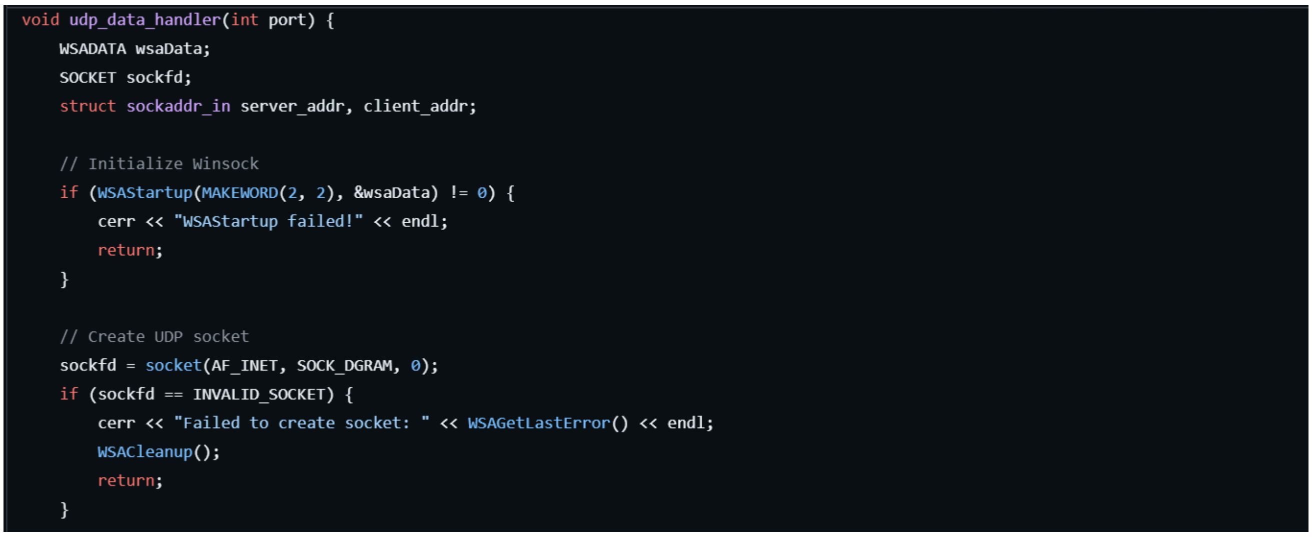


```
void send_thread(SOCKET sockfd) {
    string command;
    string to_whom;
    struct sockaddr_in client_addr;
    int client_addr_len = sizeof(client_addr);

    while (true) {
        cout << "Enter which client to send: ";
        getline(cin, to_whom);  // Get input from user

        lock_guard<mutex> lock(client_mutex);
        auto it = mapping.find(to_whom);
        if (it != mapping.end()) {
            client_addr = it->second;
            cout << "Enter command: ";
            getline(cin, command);
```

Purpose: Sends control commands to specific clients over UDP.

Details: Allows the user to specify a client ID and a command, then looks up the corresponding client address in the `mapping` and sends the command to that client via UDP

5..

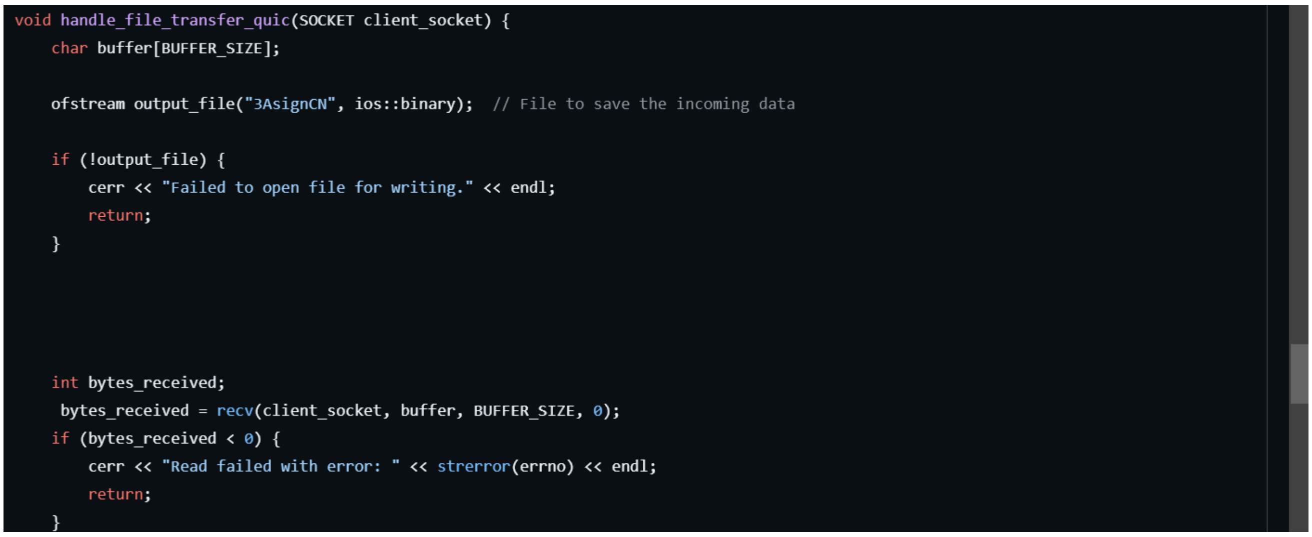


```
void udp_data_handler(int port) {
    WSADATA wsaData;
    SOCKET sockfd;
    struct sockaddr_in server_addr, client_addr;

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        cerr << "WSAStartup failed!" << endl;
        return;
    }

    // Create UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd == INVALID_SOCKET) {
        cerr << "Failed to create socket: " << WSAGetLastError() << endl;
        WSACleanup();
        return;
    }
```

Purpose: Manages UDP-based control message handling.
Details: Sets up a UDP server that listens on a specified port (`port`) and creates separate threads for receiving control messages (`receive_thread`) and sending commands to clients (`send_thread`). This function keeps the control communication running in parallel to other services.

Here in udp , we first receive client id from each client and map it to its client socket , so we can specify which client to send command later on.

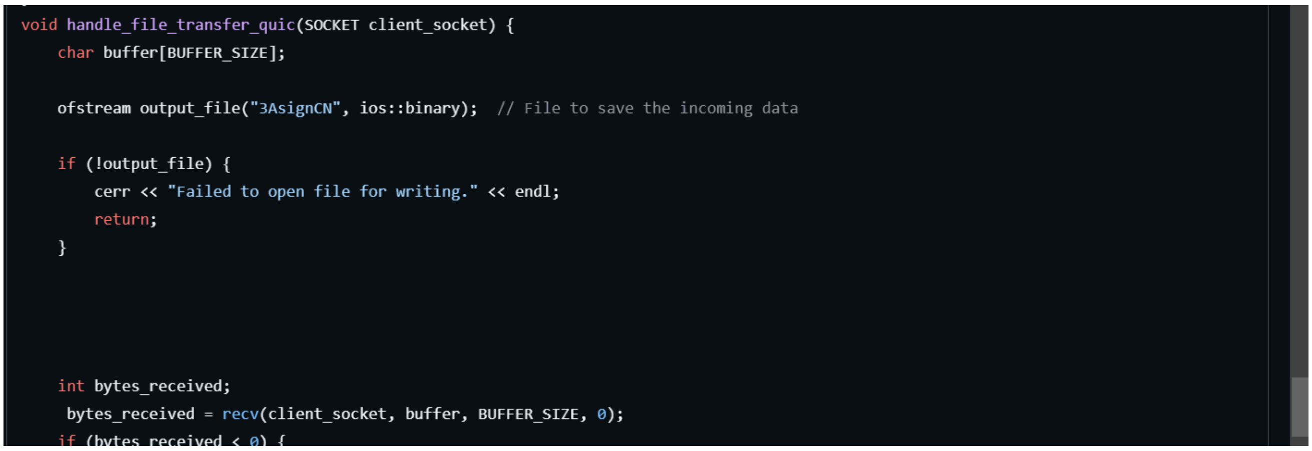
6.



```
void handle_file_transfer_quic(SOCKET client_socket) {
    char buffer[BUFFER_SIZE];

    ofstream output_file("3AsignCN", ios::binary);  // File to save the incoming data

    if (!output_file) {
        cerr << "Failed to open file for writing." << endl;
        return;
    }




    int bytes_received;
     bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
    if (bytes_received < 0) {
        cerr << "Read failed with error: " << strerror(errno) << endl;
        return;
    }
```

Purpose: Manages file transfer over TCP.
Details: Receives file data in chunks from a connected client and writes it to a local file ("3AsignCN"). The client identifier is printed along with each chunk of data received. It continues until the file is completely transferred or the connection is lost.

7.



```
void handle_file_transfer_quic(SOCKET client_socket) {
    char buffer[BUFFER_SIZE];

    ofstream output_file("3AsignCN", ios::binary);  // File to save the incoming data

    if (!output_file) {
        cerr << "Failed to open file for writing." << endl;
        return;
    }




    int bytes_received;
     bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
    if (bytes_received < 0) {
```

Purpose: Sets up a file transfer service.
Details: Listens for incoming connections from clients who want to transfer files. For each connection, it creates a new thread to handle the file transfer using `handle_file_transfer_quic`. This enables the server to handle multiple concurrent file transfers.

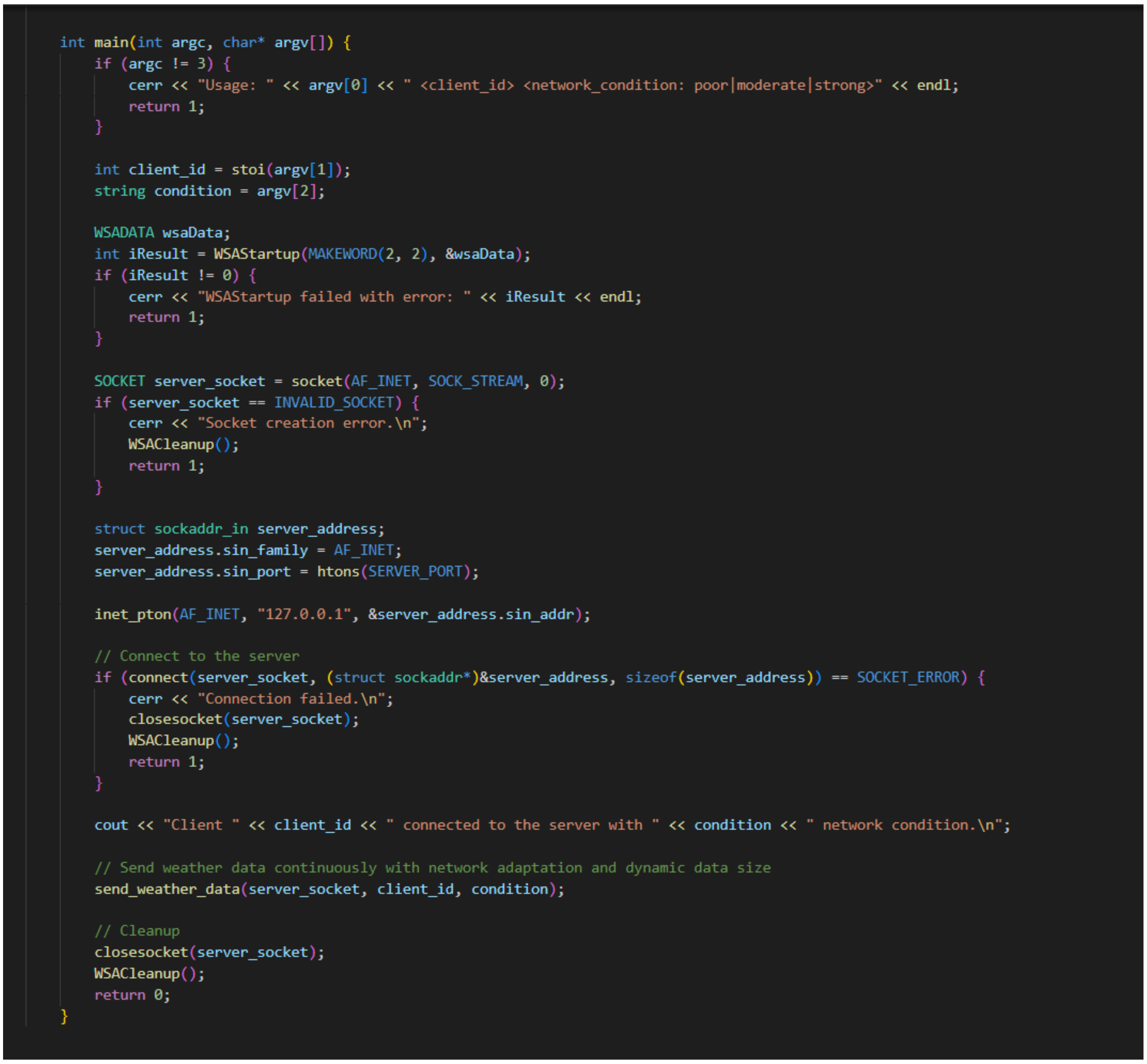
Q2. Report on Weather Station Data Transmission System

1. Client and Server Implementation

Client Main Function

In the client's main function, the client connects to the server and begins the process of generating and transmitting weather data. Key steps include:

- **Connection Setup**: The client first initialises the network (`WSAStartup`), creates a socket, and connects to the server using the server's address and port.
- **Weather Data Transmission**: The client generates random weather data based on predefined size types (small, moderate, large). This data is dynamically compressed depending on the size and transmitted to the server using the `send_weather_data()` function. The client also handles congestion control using the TCP Reno algorithm.
- **ACK Reception**: The client awaits acknowledgment from the server after each data transmission. If the same acknowledgment is received three times (indicating potential packet loss), the congestion window size (cwnd) is reduced by half.



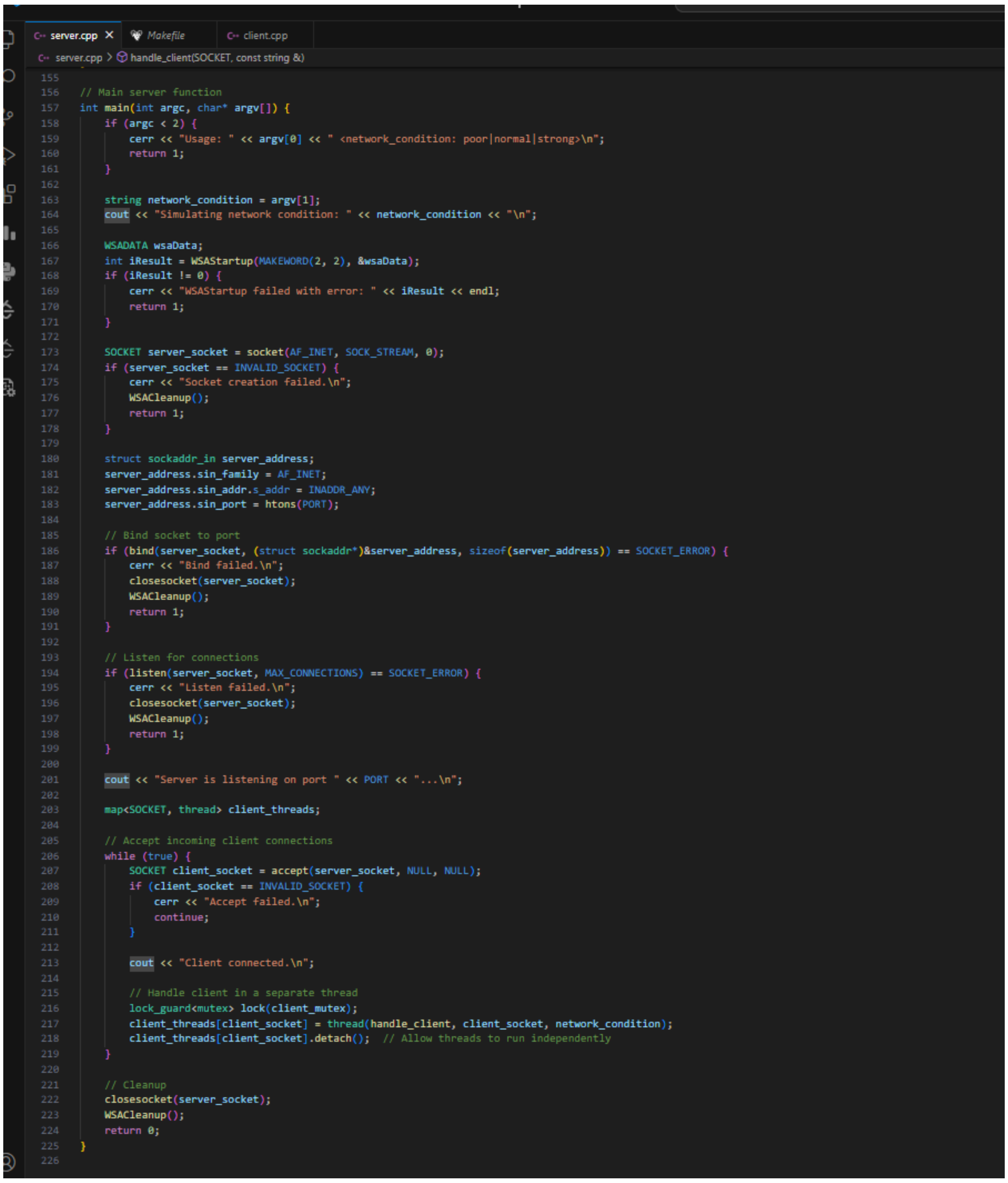
```cpp
int main(int argc, char* argv[]) {
    if (argc != 3) {
        cerr << "Usage: " << argv[0] << " <client_id> <network_condition: poor|moderate|strong>" << endl;
        return 1;
    }

    int client_id = stoi(argv[1]);
    string condition = argv[2];

    WSADATA wsaData;
    int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        cerr << "WSAStartup failed with error: " << iResult << endl;
        return 1;
    }

    SOCKET server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == INVALID_SOCKET) {
        cerr << "Socket creation error.\n";
        WSACleanup();
        return 1;
    }

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(SERVER_PORT);

    inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr);

    // Connect to the server
    if (connect(server_socket, (struct sockaddr*)&server_address, sizeof(server_address)) == SOCKET_ERROR) {
        cerr << "Connection failed.\n";
        closesocket(server_socket);
        WSACleanup();
        return 1;
    }

    cout << "Client " << client_id << " connected to the server with " << condition << " network condition.\n";

    // Send weather data continuously with network adaptation and dynamic data size
    send_weather_data(server_socket, client_id, condition);

    // Cleanup
    closesocket(server_socket);
    WSACleanup();
    return 0;
}
```

Server Main Function

In the server's main function, the server listens for incoming connections from multiple clients. Once a connection is established, the server handles data from each client using a dedicated thread to manage concurrent connections.

- **Multi-client Support**: The server uses the `accept()` function to establish connections with multiple clients. For each connection, a new thread is spawned to handle communication with that specific client. This allows the server to handle data from multiple weather stations simultaneously.

- **ACK Transmission**: For each data packet received from the client, the server sends an acknowledgment ("ACK") back to the client, simulating TCP Reno congestion control feedback.



```cpp
// Main server function
int main(int argc, char* argv[]) {
    if (argc < 2) {
        cerr << "Usage: " << argv[0] << " <network_condition: poor|normal|strong>\n";
        return 1;
    }

    string network_condition = argv[1];
    cout << "Simulating network condition: " << network_condition << "\n";

    WSADATA wsaData;
    int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        cerr << "WSAStartup failed with error: " << iResult << endl;
        return 1;
    }

    SOCKET server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == INVALID_SOCKET) {
        cerr << "Socket creation failed.\n";
        WSACleanup();
        return 1;
    }

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT);

    // Bind socket to port
    if (bind(server_socket, (struct sockaddr*)&server_address, sizeof(server_address)) == SOCKET_ERROR) {
        cerr << "Bind failed.\n";
        closesocket(server_socket);
        WSACleanup();
        return 1;
    }

    // Listen for connections
    if (listen(server_socket, MAX_CONNECTIONS) == SOCKET_ERROR) {
        cerr << "Listen failed.\n";
        closesocket(server_socket);
        WSACleanup();
        return 1;
    }

    cout << "Server is listening on port " << PORT << "...\n";

    map<SOCKET, thread> client_threads;

    // Accept incoming client connections
    while (true) {
        SOCKET client_socket = accept(server_socket, NULL, NULL);
        if (client_socket == INVALID_SOCKET) {
            cerr << "Accept failed.\n";
            continue;
        }

        cout << "Client connected.\n";

        // Handle client in a separate thread
        lock_guard<mutex> lock(client_mutex);
        client_threads[client_socket] = thread(handle_client, client_socket, network_condition);
        client_threads[client_socket].detach();  // Allow threads to run independently
    }

    // Cleanup
    closesocket(server_socket);
    WSACleanup();
    return 0;
}
```

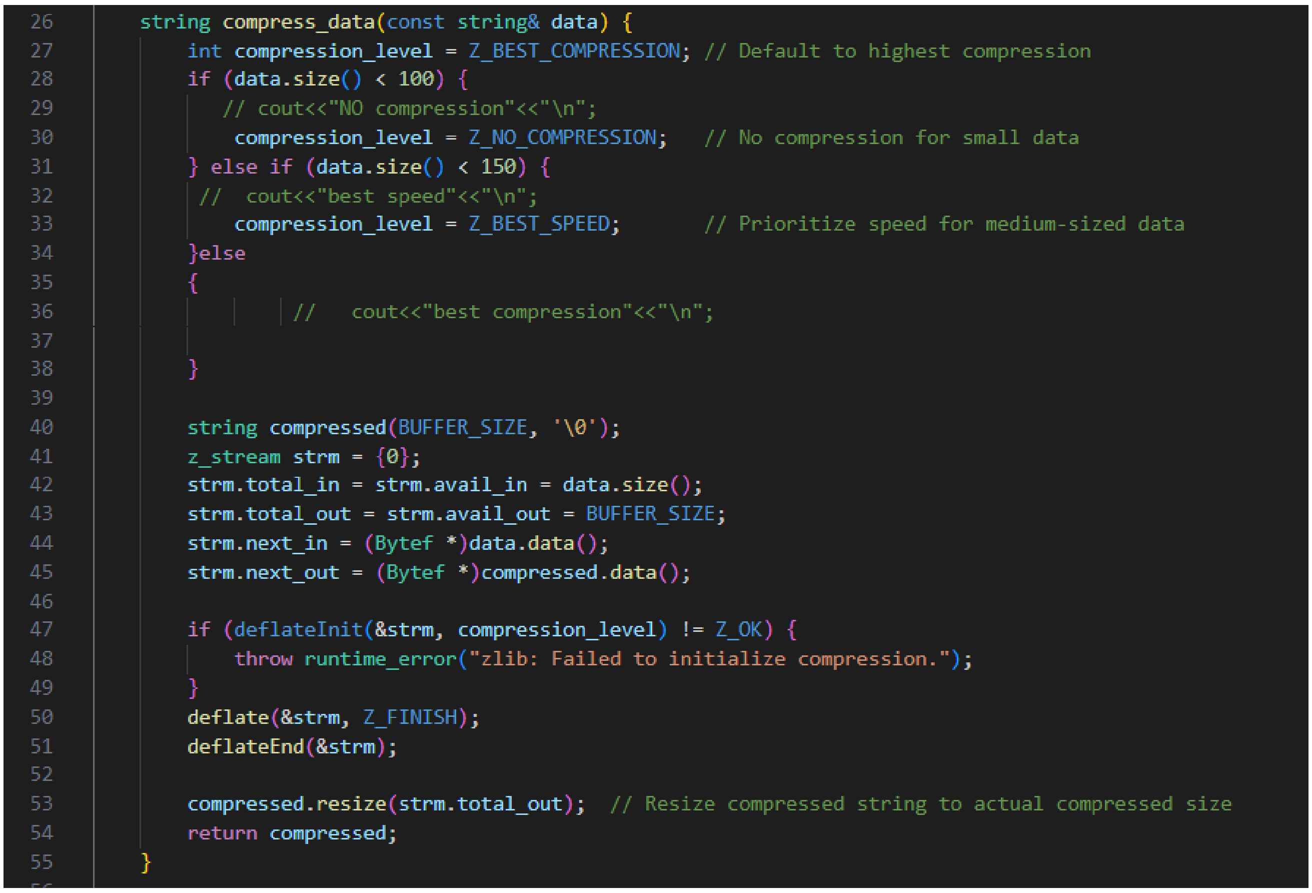
.

2. Feature Explanation

1. High Data Throughput

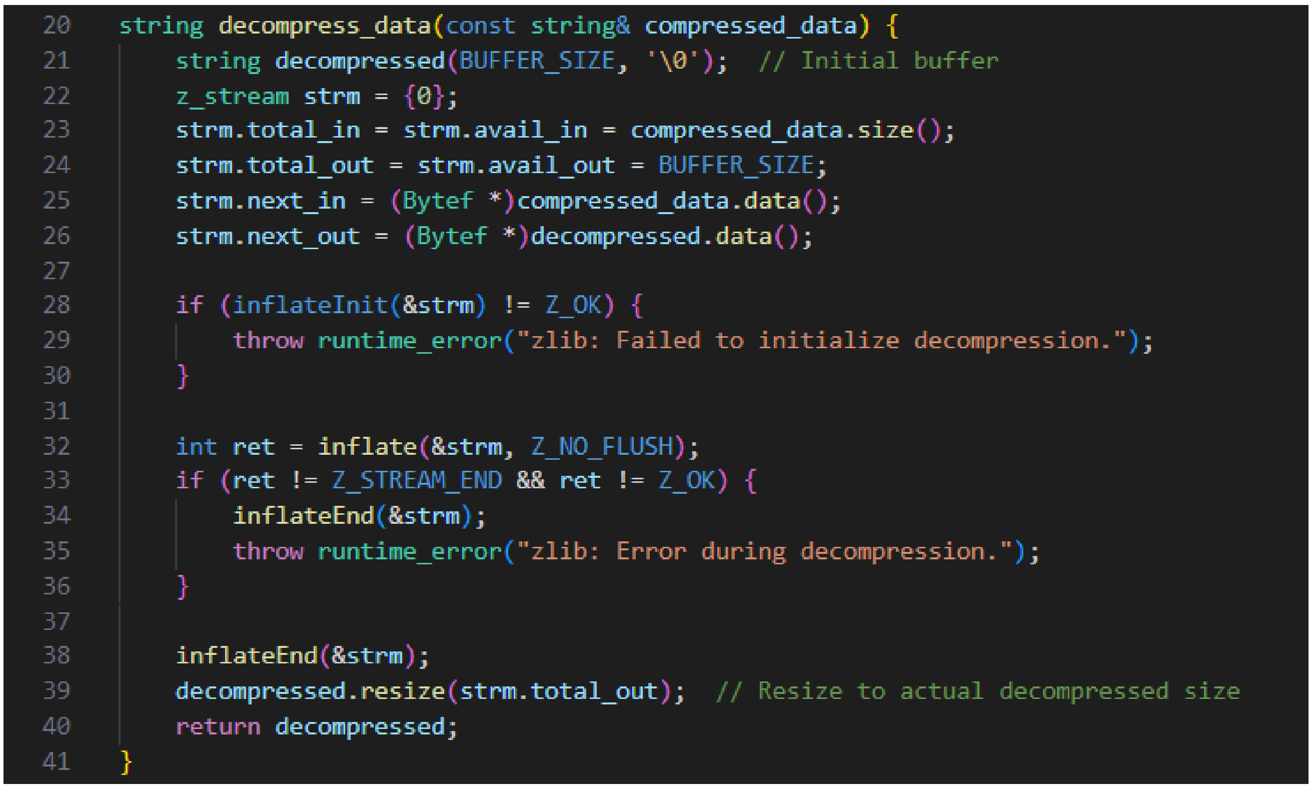
- **Feature Description**: Weather stations continuously generate large amounts of data. This system efficiently compresses and transmits the data without overwhelming the server, ensuring high throughput.

- **Implementation**:
    - In the `send_weather_data()` function of the client, the weather data is generated, compressed using zlib (`compress_data()`), and sent to the server. By compressing the data, the system reduces the amount of data that needs to be transmitted, ensuring higher throughput.



```
string compress_data(const string& data) {
    int compression_level = Z_BEST_COMPRESSION; // Default to highest compression
    if (data.size() < 100) {
       // cout<<"NO compression"<<"\n";
        compression_level = Z_NO_COMPRESSION;   // No compression for small data
    } else if (data.size() < 150) {
     //  cout<<"best speed"<<"\n";
        compression_level = Z_BEST_SPEED;       // Prioritize speed for medium-sized data
    }else
    {
                //   cout<<"best compression"<<"\n";

    }

    string compressed(BUFFER_SIZE, '\0');
    z_stream strm = {0};
    strm.total_in = strm.avail_in = data.size();
    strm.total_out = strm.avail_out = BUFFER_SIZE;
    strm.next_in = (Bytef *)data.data();
    strm.next_out = (Bytef *)compressed.data();

    if (deflateInit(&strm, compression_level) != Z_OK) {
        throw runtime_error("zlib: Failed to initialize compression.");
    }
    deflate(&strm, Z_FINISH);
    deflateEnd(&strm);

    compressed.resize(strm.total_out);  // Resize compressed string to actual compressed size
    return compressed;
}
```

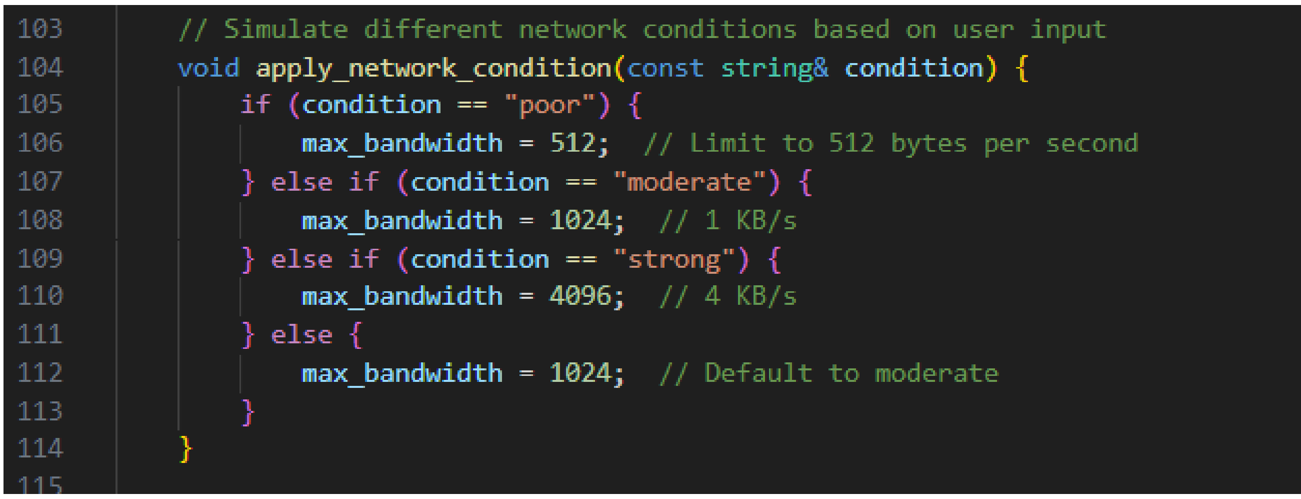
    - On the server side, the function `handle_client()` manages incoming data by decompressing it using `decompress_data()`, ensuring the server efficiently processes large amounts of incoming data without being overwhelmed.



```
string decompress_data(const string& compressed_data) {
    string decompressed(BUFFER_SIZE, '\0');  // Initial buffer
    z_stream strm = {0};
    strm.total_in = strm.avail_in = compressed_data.size();
    strm.total_out = strm.avail_out = BUFFER_SIZE;
    strm.next_in = (Bytef *)compressed_data.data();
    strm.next_out = (Bytef *)decompressed.data();

    if (inflateInit(&strm) != Z_OK) {
        throw runtime_error("zlib: Failed to initialize decompression.");
    }

    int ret = inflate(&strm, Z_NO_FLUSH);
    if (ret != Z_STREAM_END && ret != Z_OK) {
        inflateEnd(&strm);
        throw runtime_error("zlib: Error during decompression.");
    }

    inflateEnd(&strm);
    decompressed.resize(strm.total_out);  // Resize to actual decompressed size
    return decompressed;
}
```

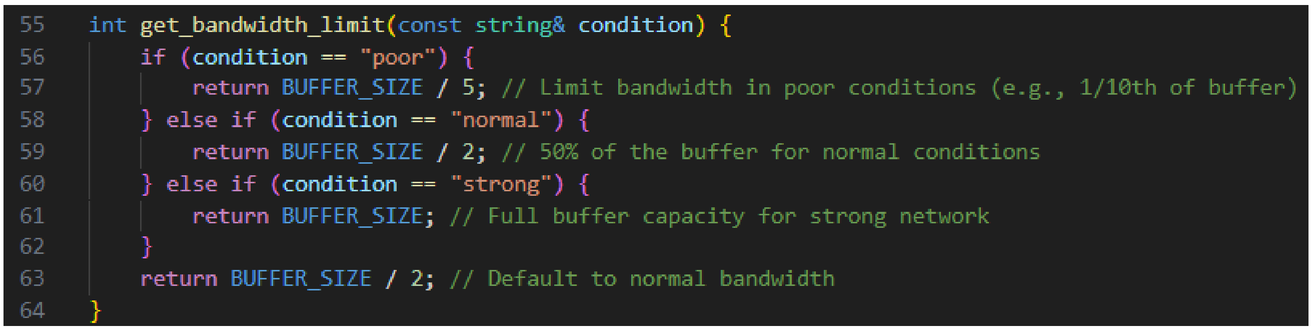
2. Network Adaptability

- **Feature Description**: The system adapts to different network conditions (poor, moderate, strong) to ensure efficient transmission without overwhelming the server.
- **Implementation**:
    - In the client code, the `apply_network_condition()` function dynamically adjusts the maximum bandwidth based on the network condition (poor, moderate, strong). This ensures that the client can send data at a rate that matches the network's capabilities, preventing congestion.

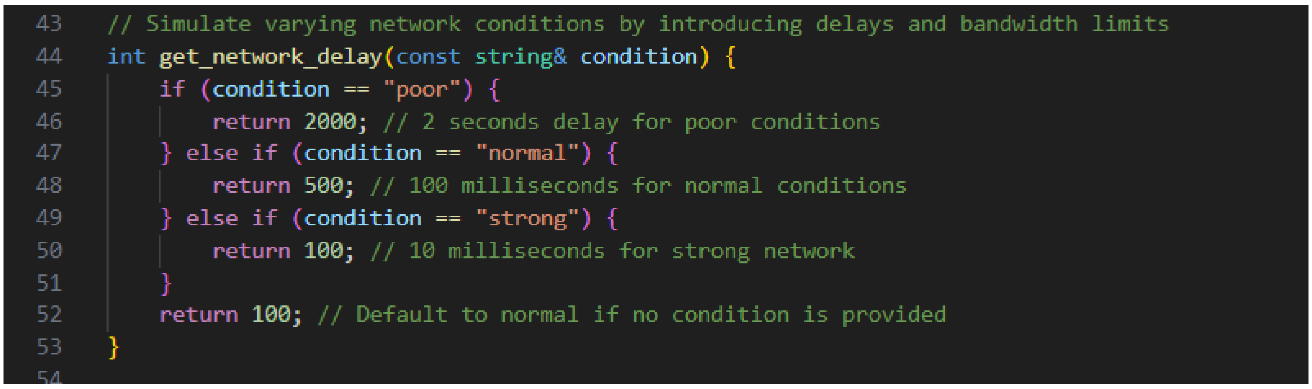


```cpp
    // Simulate different network conditions based on user input
    void apply_network_condition(const string& condition) {
        if (condition == "poor") {
            max_bandwidth = 512;   // Limit to 512 bytes per second
        } else if (condition == "moderate") {
            max_bandwidth = 1024;   // 1 KB/s
        } else if (condition == "strong") {
            max_bandwidth = 4096;   // 4 KB/s
        } else {
            max_bandwidth = 1024;   // Default to moderate
        }
    }
```

- The server simulates network conditions by limiting the bandwidth available to clients. In `handle_client()`, the server uses the `get_bandwidth_limit()` function to simulate bandwidth limitations and the `get_network_delay()` function to simulate network latency based on conditions.



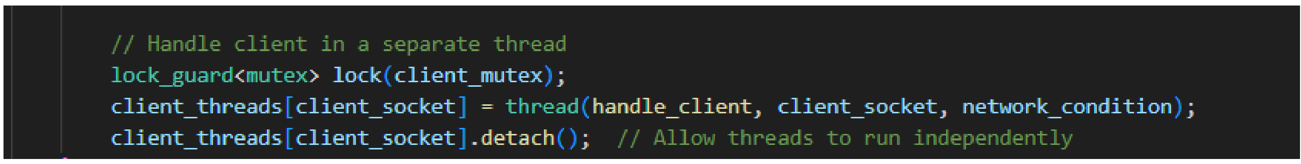
```cpp
int get_bandwidth_limit(const string& condition) {
    if (condition == "poor") {
        return BUFFER_SIZE / 5; // Limit bandwidth in poor conditions (e.g., 1/10th of buffer)
    } else if (condition == "normal") {
        return BUFFER_SIZE / 2; // 50% of the buffer for normal conditions
    } else if (condition == "strong") {
        return BUFFER_SIZE; // Full buffer capacity for strong network
    }
    return BUFFER_SIZE / 2; // Default to normal bandwidth
}
```



```cpp
// Simulate varying network conditions by introducing delays and bandwidth limits
int get_network_delay(const string& condition) {
    if (condition == "poor") {
        return 2000; // 2 seconds delay for poor conditions
    } else if (condition == "normal") {
        return 500; // 100 milliseconds for normal conditions
    } else if (condition == "strong") {
        return 100; // 10 milliseconds for strong network
    }
    return 100; // Default to normal if no condition is provided
}
```

3. Efficient Data Handling

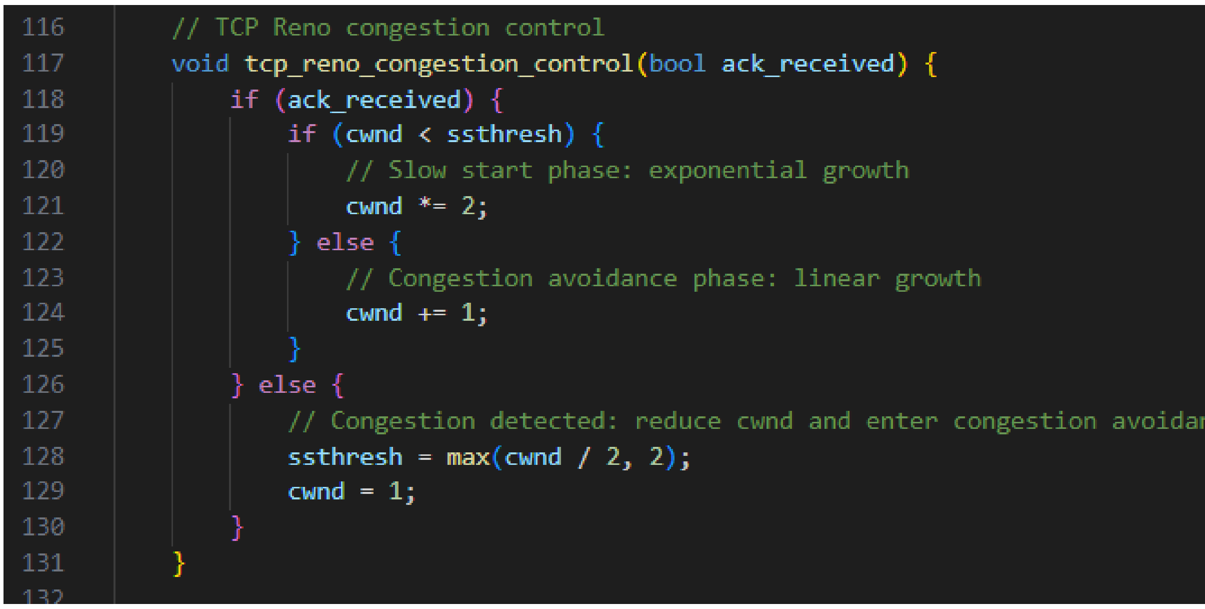
- **Feature Description**: The server needs to handle multiple concurrent clients efficiently, ensuring that data from each weather station is processed without overwhelming the system.
- **Implementation**:
  - The server handles multiple clients by spawning new threads for each incoming client connection, using `thread(handle_client, client_socket, network_condition).detach();`. This allows the server to handle multiple clients in parallel without being blocked by any single client.
  - The server ensures efficient data handling by decompressing incoming data (`decompress_data()`) and acknowledging the receipt of data, which allows the client to adjust its transmission rate dynamically.



```
// Handle client in a separate thread
lock_guard<mutex> lock(client_mutex);
client_threads[client_socket] = thread(handle_client, client_socket, network_condition);
client_threads[client_socket].detach();  // Allow threads to run independently
```

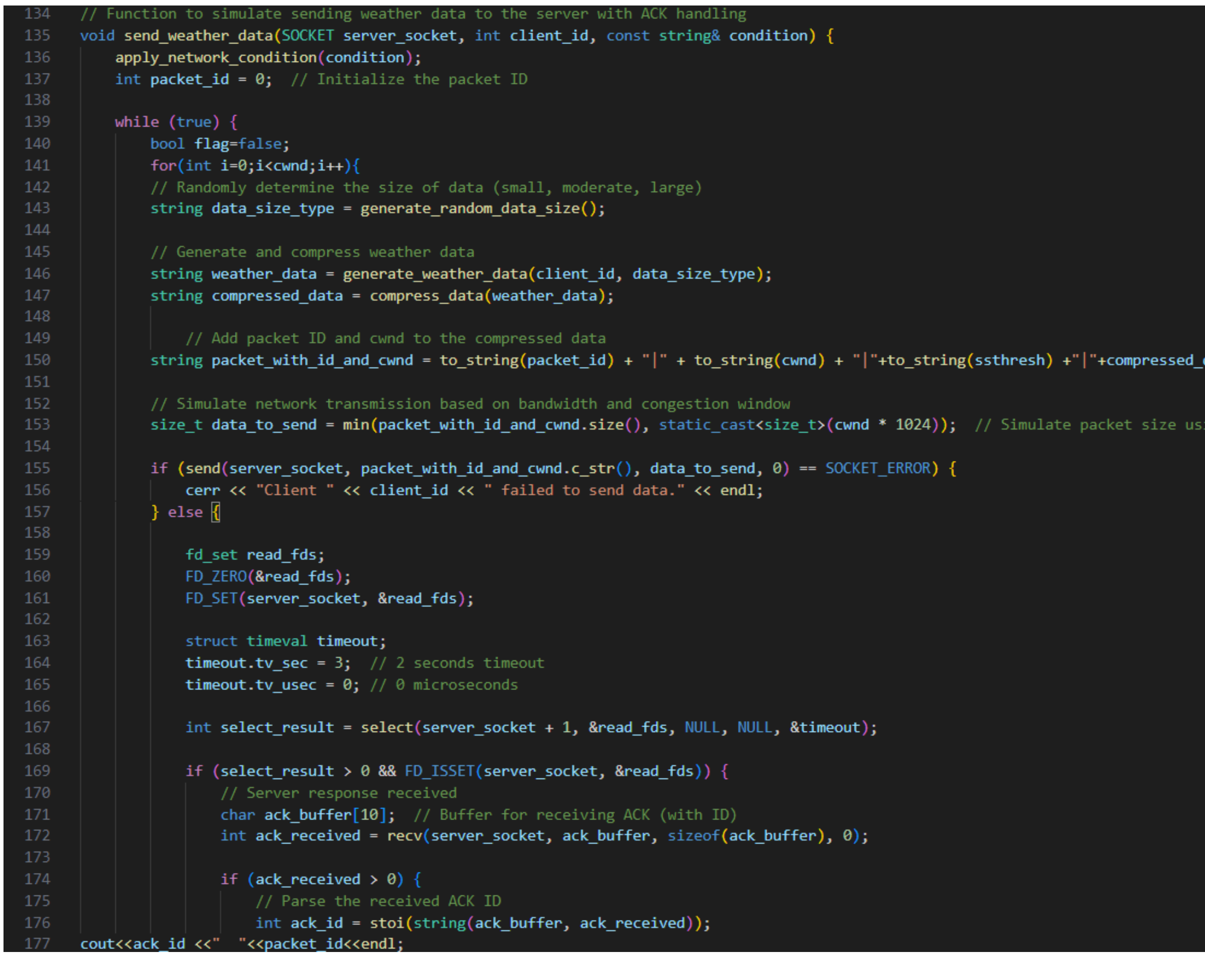
4. Adaptive Data Transmission (TCP Reno)

- **Feature Description**: The TCP Reno algorithm is implemented to manage congestion control at the weather stations. The congestion window (cwnd) is adjusted dynamically based on acknowledgments received from the server, ensuring efficient data transmission even under changing network conditions.
- **Implementation**:
  - The client uses the `tcp_reno_congestion_control()` function to adaptively adjust its data transmission rate. The function increases the congestion window size when acknowledgments are received and reduces the window size if no acknowledgment is received or packet loss is detected.



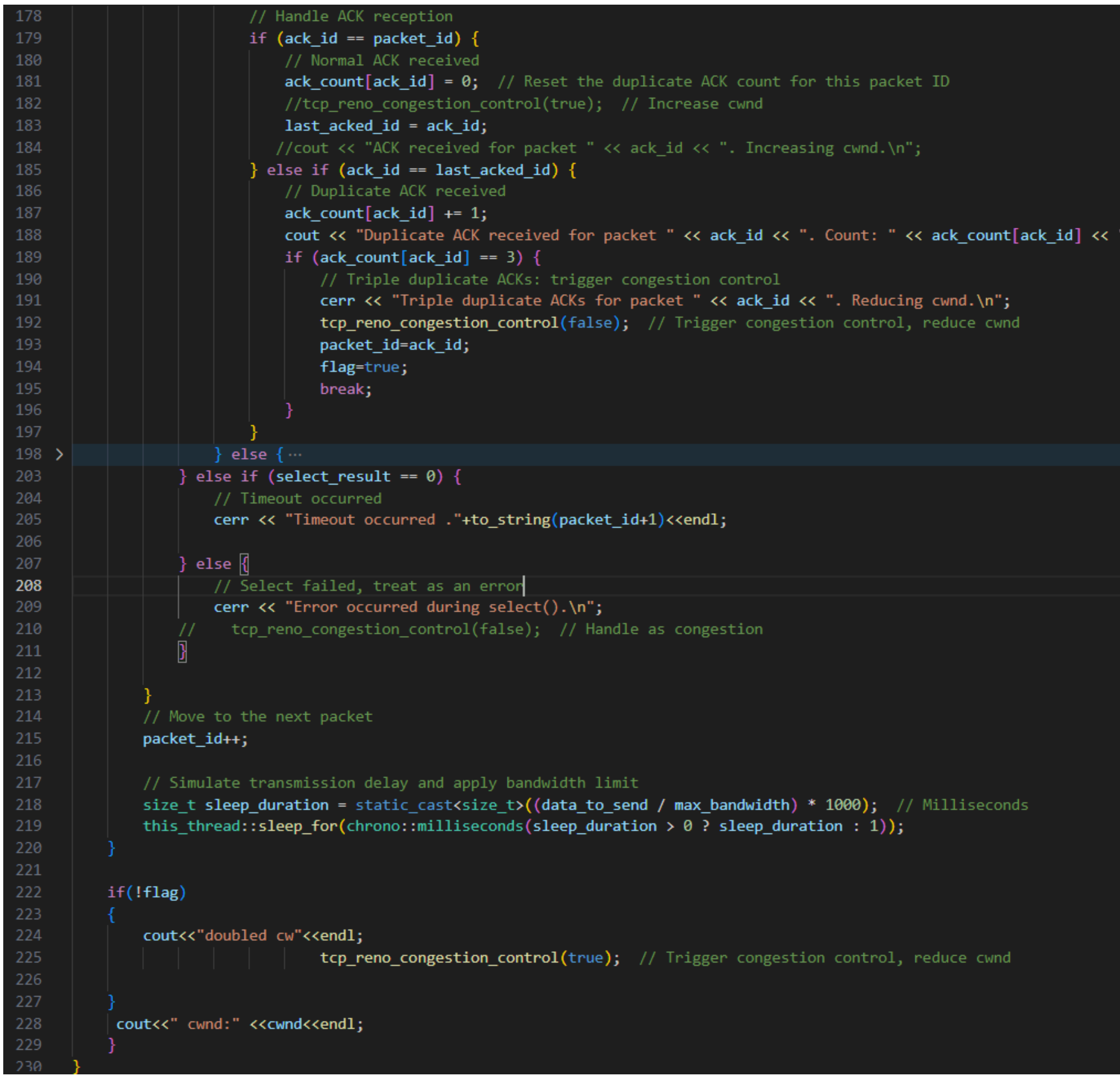
```cpp
// TCP Reno congestion control
void tcp_reno_congestion_control(bool ack_received) {
    if (ack_received) {
        if (cwnd < ssthresh) {
            // Slow start phase: exponential growth
            cwnd *= 2;
        } else {
            // Congestion avoidance phase: linear growth
            cwnd += 1;
        }
    } else {
        // Congestion detected: reduce cwnd and enter congestion avoida
        ssthresh = max(cwnd / 2, 2);
        cwnd = 1;
    }
}
```

- The server sends acknowledgments after receiving data from the client, simulating TCP Reno behavior. The client monitors these acknowledgments and adjusts its `cwnd` accordingly in `send_weather_data()`.



```cpp
// Function to simulate sending weather data to the server with ACK handling
void send_weather_data(SOCKET server_socket, int client_id, const string& condition) {
    apply_network_condition(condition);
    int packet_id = 0;  // Initialize the packet ID

    while (true) {
        bool flag=false;
        for(int i=0;i<cwnd;i++){
        // Randomly determine the size of data (small, moderate, large)
        string data_size_type = generate_random_data_size();

        // Generate and compress weather data
        string weather_data = generate_weather_data(client_id, data_size_type);
        string compressed_data = compress_data(weather_data);

            // Add packet ID and cwnd to the compressed data
        string packet_with_id_and_cwnd = to_string(packet_id) + "|" + to_string(cwnd) + "|"+to_string(ssthresh) +"|"+compressed_

        // Simulate network transmission based on bandwidth and congestion window
        size_t data_to_send = min(packet_with_id_and_cwnd.size(), static_cast<size_t>(cwnd * 1024));  // Simulate packet size us

        if (send(server_socket, packet_with_id_and_cwnd.c_str(), data_to_send, 0) == SOCKET_ERROR) {
            cerr << "Client " << client_id << " failed to send data." << endl;
        } else {

            fd_set read_fds;
            FD_ZERO(&read_fds);
            FD_SET(server_socket, &read_fds);

            struct timeval timeout;
            timeout.tv_sec = 3;  // 2 seconds timeout
            timeout.tv_usec = 0; // 0 microseconds

            int select_result = select(server_socket + 1, &read_fds, NULL, NULL, &timeout);

            if (select_result > 0 && FD_ISSET(server_socket, &read_fds)) {
                // Server response received
                char ack_buffer[10];  // Buffer for receiving ACK (with ID)
                int ack_received = recv(server_socket, ack_buffer, sizeof(ack_buffer), 0);

                if (ack_received > 0) {
                    // Parse the received ACK ID
                    int ack_id = stoi(string(ack_buffer, ack_received));
cout<<ack_id <<"  "<<packet_id<<endl;
```

-

- 

```
                    // Handle ACK reception
                    if (ack_id == packet_id) {
                        // Normal ACK received
                        ack_count[ack_id] = 0;  // Reset the duplicate ACK count for this packet ID
                        //tcp_reno_congestion_control(true);  // Increase cwnd
                        last_acked_id = ack_id;
                       //cout << "ACK received for packet " << ack_id << ". Increasing cwnd.\n";
                    } else if (ack_id == last_acked_id) {
                        // Duplicate ACK received
                        ack_count[ack_id] += 1;
                        cout << "Duplicate ACK received for packet " << ack_id << ". Count: " << ack_count[ack_id] <<
                        if (ack_count[ack_id] == 3) {
                            // Triple duplicate ACKs: trigger congestion control
                            cerr << "Triple duplicate ACKs for packet " << ack_id << ". Reducing cwnd.\n";
                            tcp_reno_congestion_control(false);  // Trigger congestion control, reduce cwnd
                            packet_id=ack_id;
                            flag=true;
                            break;
                        }
                    }
                } else {…
            } else if (select_result == 0) {
                // Timeout occurred
                cerr << "Timeout occurred ."+to_string(packet_id+1)<<endl;

            } else {
                // Select failed, treat as an error
                cerr << "Error occurred during select().\n";
            //    tcp_reno_congestion_control(false);  // Handle as congestion
            }

        }
        // Move to the next packet
        packet_id++;

        // Simulate transmission delay and apply bandwidth limit
        size_t sleep_duration = static_cast<size_t>((data_to_send / max_bandwidth) * 1000);  // Milliseconds
        this_thread::sleep_for(chrono::milliseconds(sleep_duration > 0 ? sleep_duration : 1));
    }

    if(!flag)
    {
        cout<<"doubled cw"<<endl;
                            tcp_reno_congestion_control(true);  // Trigger congestion control, reduce cwnd

    }
     cout<<" cwnd:" <<cwnd<<endl;
    }
}
```

5. Simulated Network Constraints

- **Feature Description**: The system simulates a constrained network environment, with limitations on bandwidth and network delays. The client and server must adapt to these constraints to maintain efficient communication.
- **Implementation**:
    - The client adjusts its transmission rate based on bandwidth limitations using `max_bandwidth` in `send_weather_data()`. This ensures the data is sent at an appropriate rate depending on the current network condition.

    - The server simulates network constraints in the `handle_client()` function using the `get_bandwidth_limit()` and `get_network_delay()` functions. These simulate real-world network issues, ensuring the system can handle varying conditions.

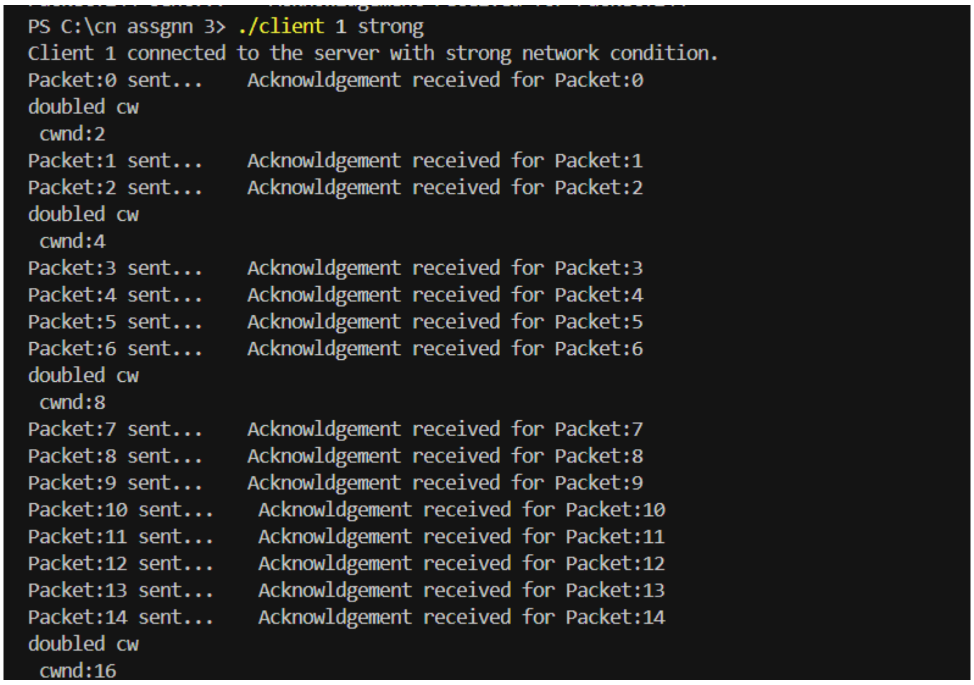
6. Dynamic Data Compression

- **Feature Description**: The system dynamically compresses weather data based on its size to balance between data reduction and transmission speed.
- **Implementation**:
    - The `compress_data()` function on the client dynamically adjusts the compression level based on the size of the data. For smaller data, no compression is applied (`Z_NO_COMPRESSION`), for medium-sized data, speed is prioritized (`Z_BEST_SPEED`), and for larger data, maximum compression (`Z_BEST_COMPRESSION`) is used.
    - On the server side, the data is decompressed using the `decompress_data()` function after being received.

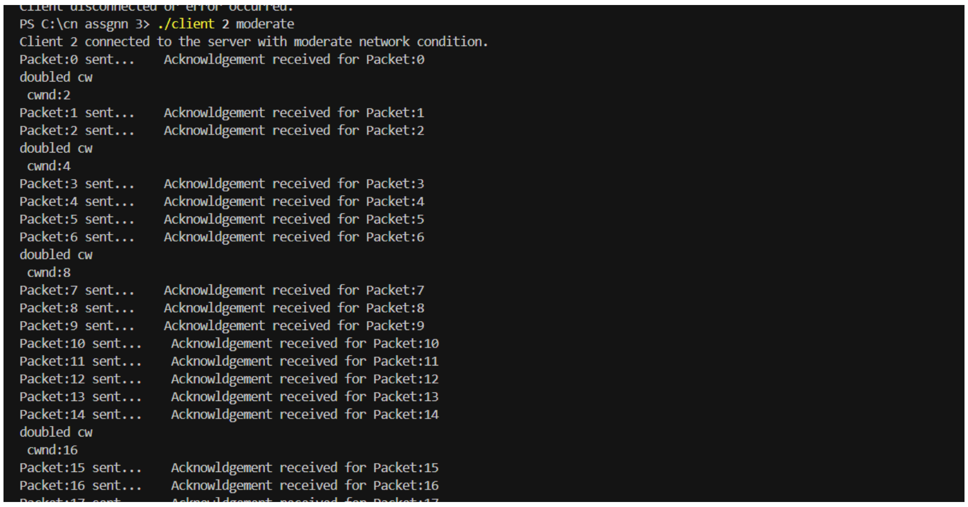
This system efficiently manages large volumes of weather data transmission from multiple weather stations, ensuring high data throughput while adapting to varying network conditions. By simulating real-world network constraints and implementing TCP Reno congestion control, the system ensures smooth data transmission under poor conditions without overwhelming the server. Dynamic data compression further optimizes the system by reducing data size while maintaining transmission speed.

Simulation

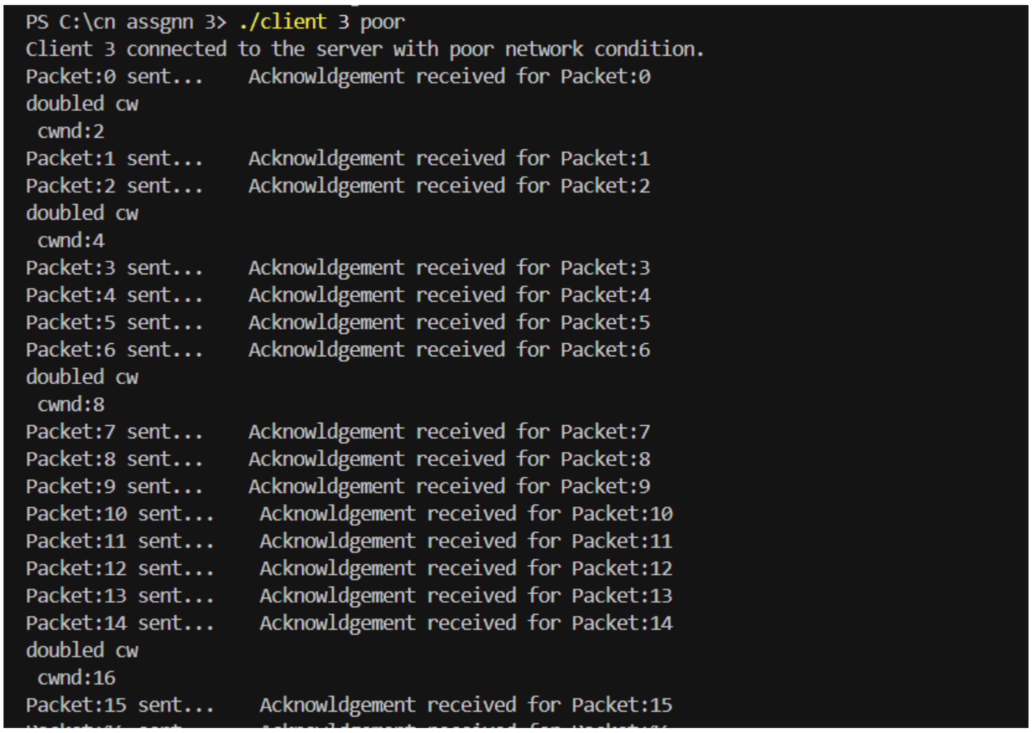
1. For simulation of clients with varying network connectivity, we have used arguments for specifying if the client is strong, moderate or poor as shown in below images.



```
PS C:\cn assgnn 3> ./client 1 strong
Client 1 connected to the server with strong network condition.
Packet:0 sent...    Acknowldgement received for Packet:0
doubled cw
 cwnd:2
Packet:1 sent...    Acknowldgement received for Packet:1
Packet:2 sent...    Acknowldgement received for Packet:2
doubled cw
 cwnd:4
Packet:3 sent...    Acknowldgement received for Packet:3
Packet:4 sent...    Acknowldgement received for Packet:4
Packet:5 sent...    Acknowldgement received for Packet:5
Packet:6 sent...    Acknowldgement received for Packet:6
doubled cw
 cwnd:8
Packet:7 sent...    Acknowldgement received for Packet:7
Packet:8 sent...    Acknowldgement received for Packet:8
Packet:9 sent...    Acknowldgement received for Packet:9
Packet:10 sent...    Acknowldgement received for Packet:10
Packet:11 sent...    Acknowldgement received for Packet:11
Packet:12 sent...    Acknowldgement received for Packet:12
Packet:13 sent...    Acknowldgement received for Packet:13
Packet:14 sent...    Acknowldgement received for Packet:14
doubled cw
 cwnd:16
```

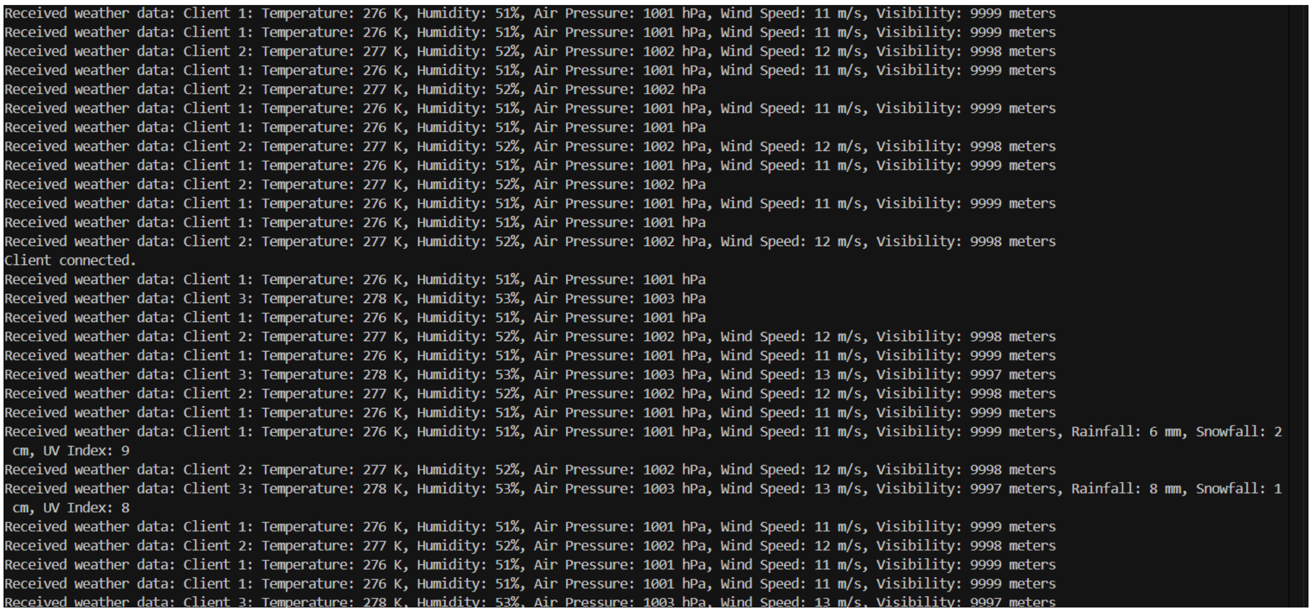


```
PS C:\cn assgnn 3> ./client 2 moderate
Client 2 connected to the server with moderate network condition.
Packet:0 sent...    Acknowldgement received for Packet:0
doubled cw
 cwnd:2
Packet:1 sent...    Acknowldgement received for Packet:1
Packet:2 sent...    Acknowldgement received for Packet:2
doubled cw
 cwnd:4
Packet:3 sent...    Acknowldgement received for Packet:3
Packet:4 sent...    Acknowldgement received for Packet:4
Packet:5 sent...    Acknowldgement received for Packet:5
Packet:6 sent...    Acknowldgement received for Packet:6
doubled cw
 cwnd:8
Packet:7 sent...    Acknowldgement received for Packet:7
Packet:8 sent...    Acknowldgement received for Packet:8
Packet:9 sent...    Acknowldgement received for Packet:9
Packet:10 sent...    Acknowldgement received for Packet:10
Packet:11 sent...    Acknowldgement received for Packet:11
Packet:12 sent...    Acknowldgement received for Packet:12
Packet:13 sent...    Acknowldgement received for Packet:13
Packet:14 sent...    Acknowldgement received for Packet:14
doubled cw
 cwnd:16
Packet:15 sent...    Acknowldgement received for Packet:15
Packet:16 sent...    Acknowldgement received for Packet:16
```



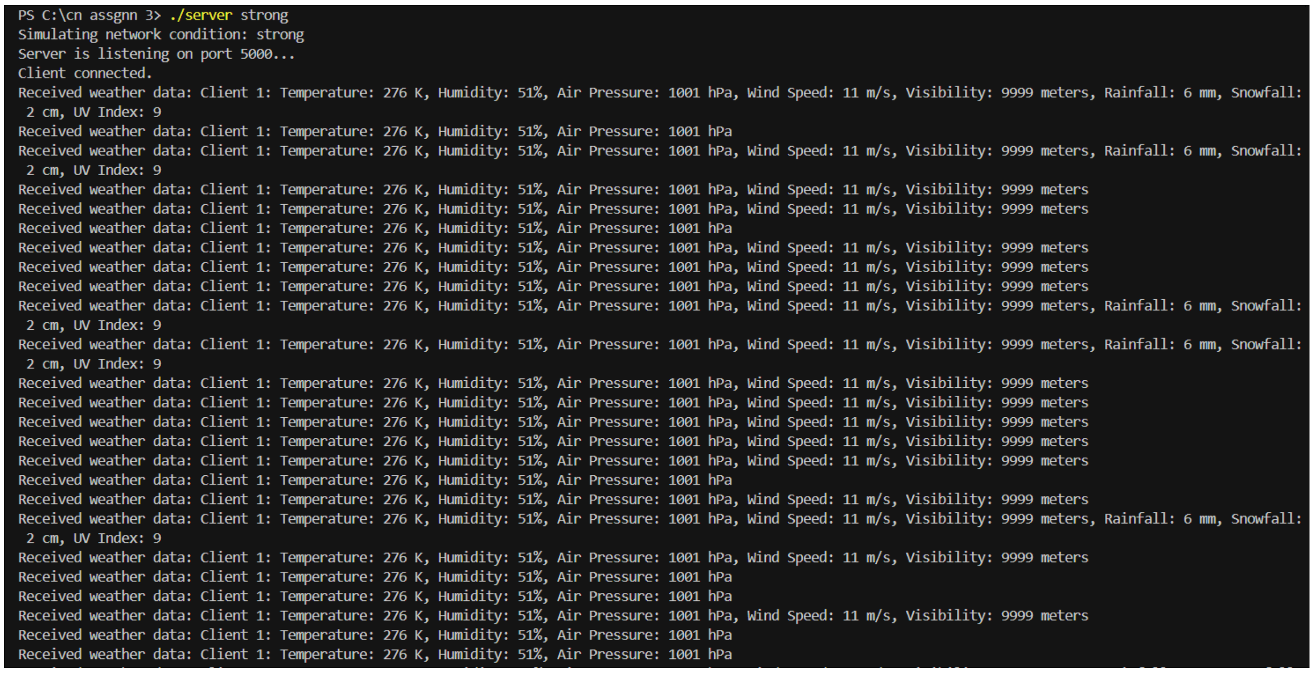
```
PS C:\cn assgnn 3> ./client 3 poor
Client 3 connected to the server with poor network condition.
Packet:0 sent...    Acknowldgement received for Packet:0
doubled cw
 cwnd:2
Packet:1 sent...    Acknowldgement received for Packet:1
Packet:2 sent...    Acknowldgement received for Packet:2
doubled cw
 cwnd:4
Packet:3 sent...    Acknowldgement received for Packet:3
Packet:4 sent...    Acknowldgement received for Packet:4
Packet:5 sent...    Acknowldgement received for Packet:5
Packet:6 sent...    Acknowldgement received for Packet:6
doubled cw
 cwnd:8
Packet:7 sent...    Acknowldgement received for Packet:7
Packet:8 sent...    Acknowldgement received for Packet:8
Packet:9 sent...    Acknowldgement received for Packet:9
Packet:10 sent...    Acknowldgement received for Packet:10
Packet:11 sent...    Acknowldgement received for Packet:11
Packet:12 sent...    Acknowldgement received for Packet:12
Packet:13 sent...    Acknowldgement received for Packet:13
Packet:14 sent...    Acknowldgement received for Packet:14
doubled cw
 cwnd:16
Packet:15 sent...    Acknowldgement received for Packet:15
```

WE can see on the server side that the number of packets received from client 1 (strong) is the highest , followed by client 2 (moderate) and client 3(poor) the least.

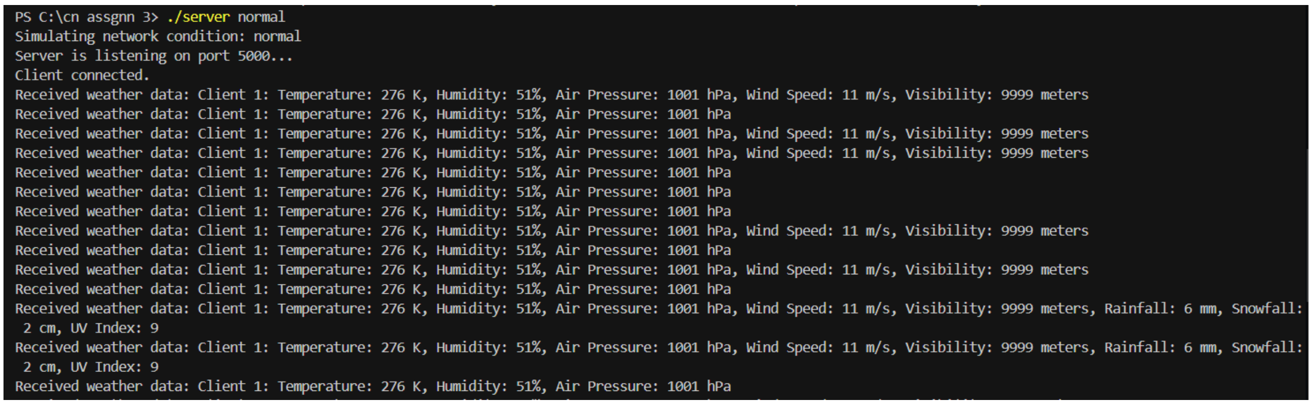


```
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Client connected.
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 3: Temperature: 278 K, Humidity: 53%, Air Pressure: 1003 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 3: Temperature: 278 K, Humidity: 53%, Air Pressure: 1003 hPa, Wind Speed: 13 m/s, Visibility: 9997 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters, Rainfall: 6 mm, Snowfall: 2
 cm, UV Index: 9
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 3: Temperature: 278 K, Humidity: 53%, Air Pressure: 1003 hPa, Wind Speed: 13 m/s, Visibility: 9997 meters, Rainfall: 8 mm, Snowfall: 1
 cm, UV Index: 8
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 3: Temperature: 278 K, Humidity: 53%, Air Pressure: 1003 hPa, Wind Speed: 13 m/s, Visibility: 9997 meters
```

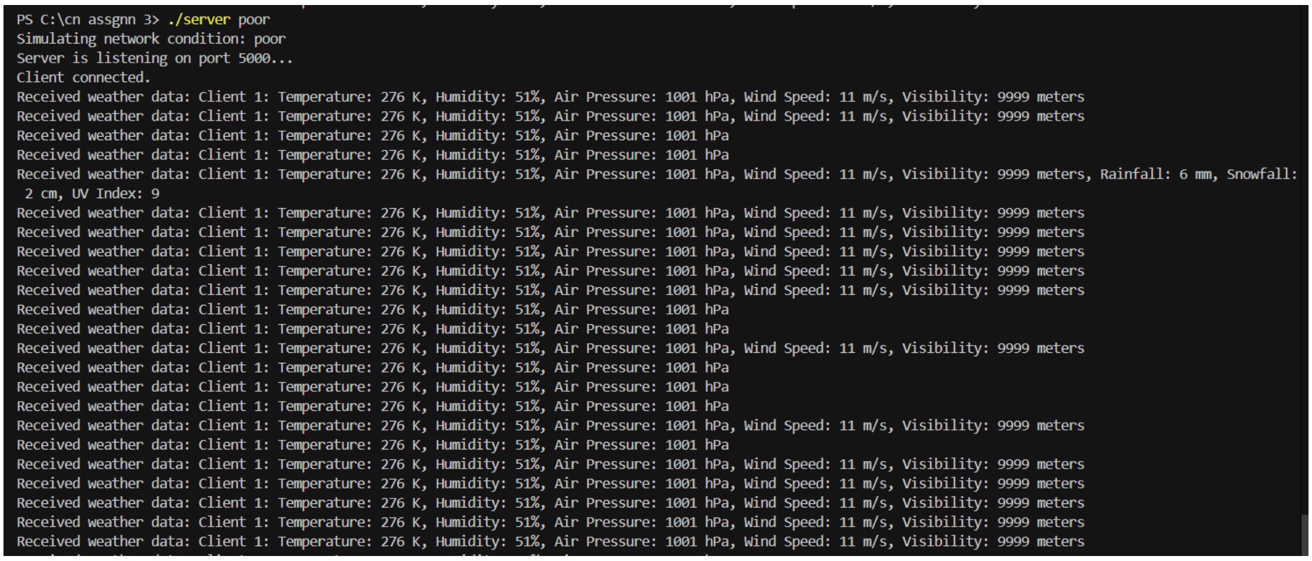
2. For simulating the strength of the server we can pass 3 arguments strength, normal and poor while running the executable file. The delay between receiving packets is in the order of: poor server > normal server > strong server .



```
PS C:\cn assgnn 3> ./server strong
Simulating network condition: strong
Server is listening on port 5000...
Client connected.
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters, Rainfall: 6 mm, Snowfall:
 2 cm, UV Index: 9
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters, Rainfall: 6 mm, Snowfall:
 2 cm, UV Index: 9
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters, Rainfall: 6 mm, Snowfall:
 2 cm, UV Index: 9
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters, Rainfall: 6 mm, Snowfall:
 2 cm, UV Index: 9
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters, Rainfall: 6 mm, Snowfall:
 2 cm, UV Index: 9
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
```

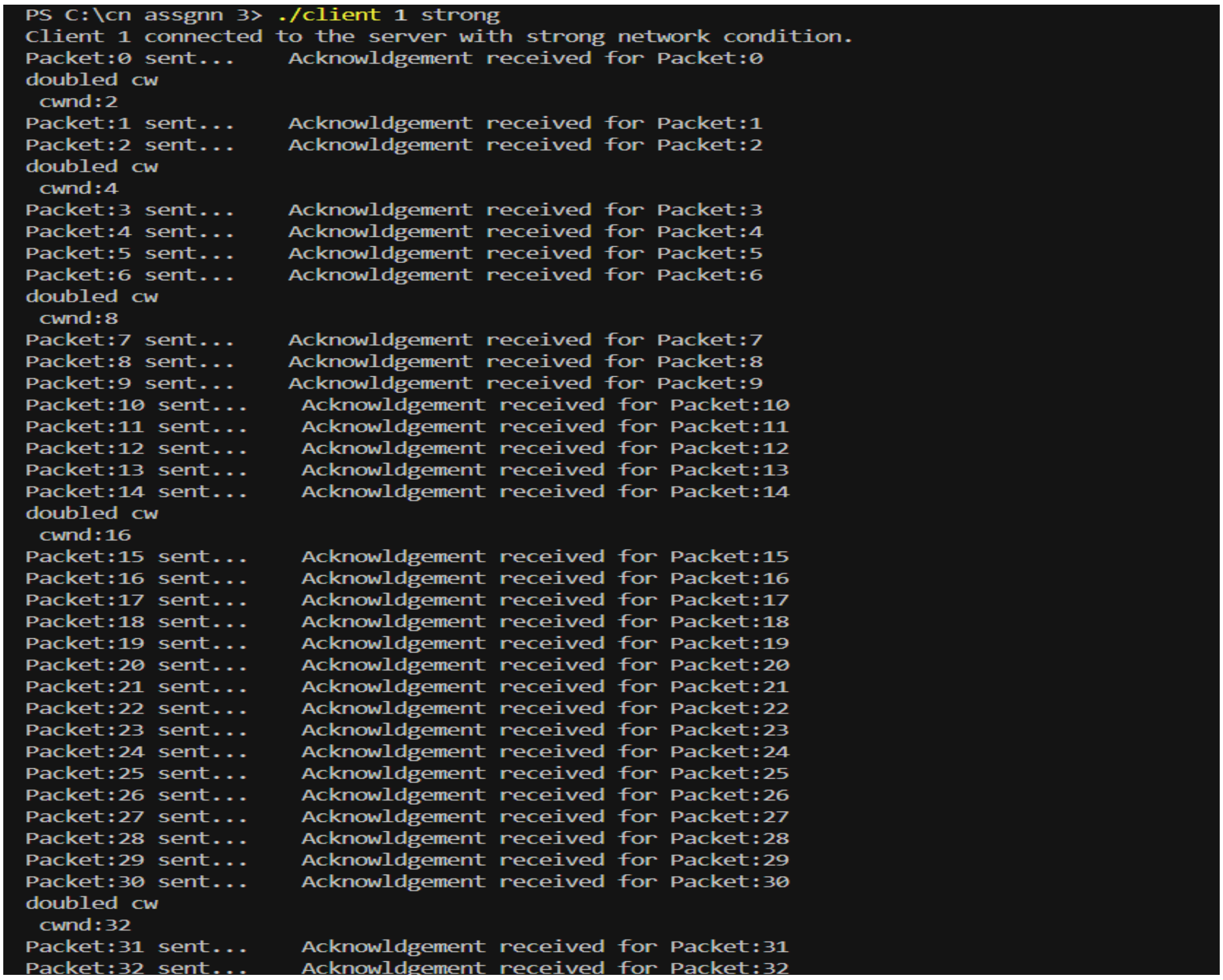


```
PS C:\cn assgnn 3> ./server normal
Simulating network condition: normal
Server is listening on port 5000...
Client connected.
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters, Rainfall: 6 mm, Snowfall:
 2 cm, UV Index: 9
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters, Rainfall: 6 mm, Snowfall:
 2 cm, UV Index: 9
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
```



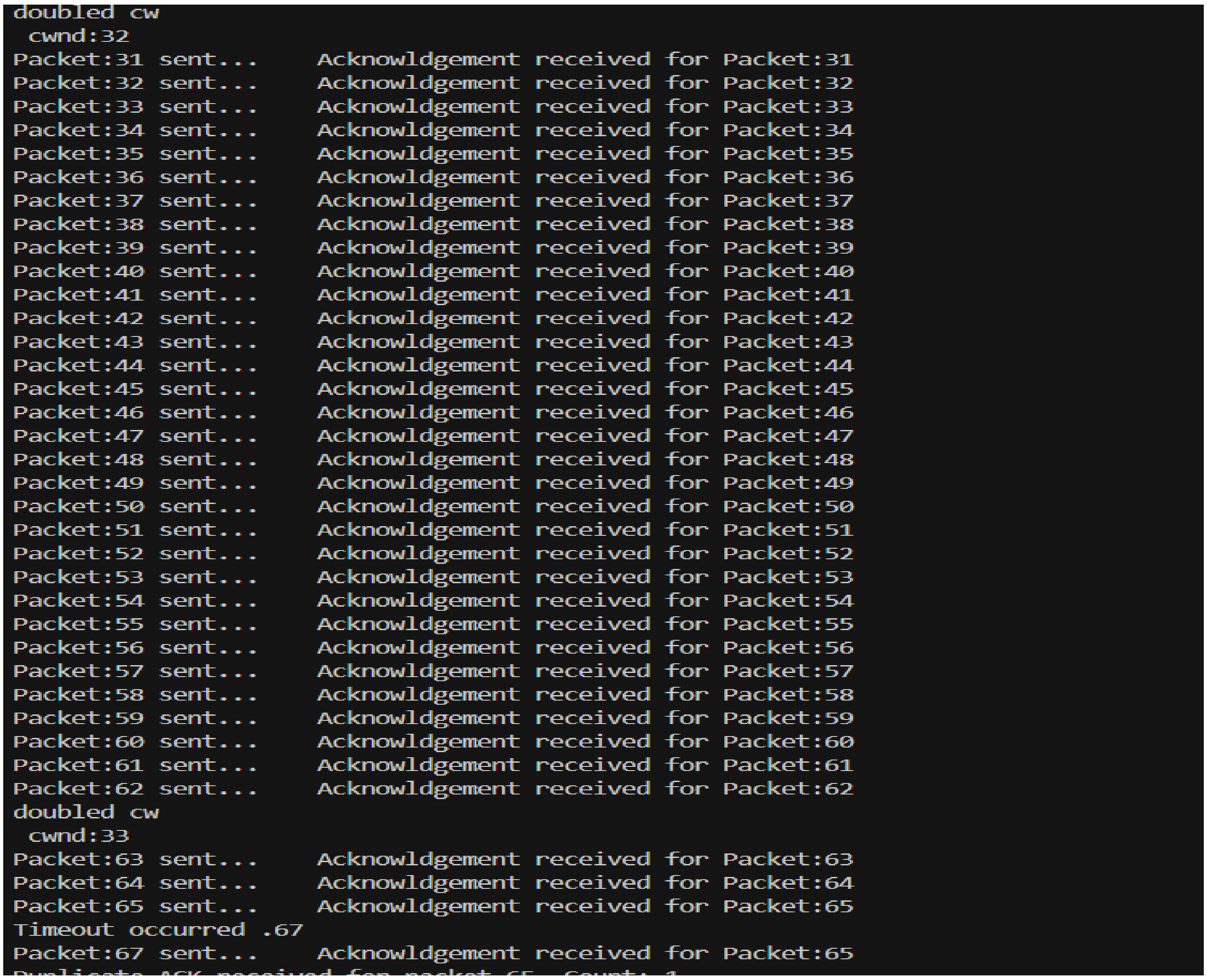
```
PS C:\cn assgnn 3> ./server poor
Simulating network condition: poor
Server is listening on port 5000...
Client connected.
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters, Rainfall: 6 mm, Snowfall:
 2 cm, UV Index: 9
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visibility: 9999 meters
```

3.**Working of TCP Reno:** Initially the client sends packets with congestion window =1 and doubles after the congestion window after it receives the acknowledgments for all the packets sent in that cwnd.This happens until congestion window is smaller than slow start threshold. This simulates the **slow start**.



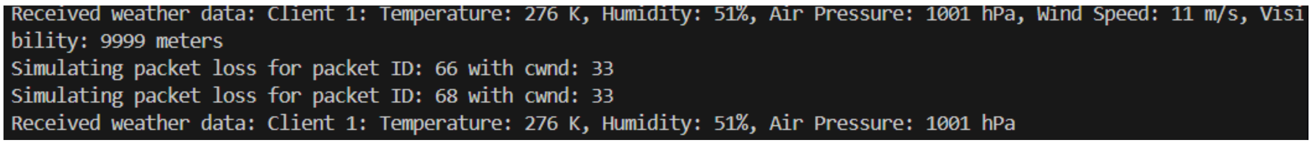
```
PS C:\cn assgnn 3> ./client 1 strong
Client 1 connected to the server with strong network condition.
Packet:0 sent...    Acknowldgement received for Packet:0
doubled cw
 cwnd:2
Packet:1 sent...    Acknowldgement received for Packet:1
Packet:2 sent...    Acknowldgement received for Packet:2
doubled cw
 cwnd:4
Packet:3 sent...    Acknowldgement received for Packet:3
Packet:4 sent...    Acknowldgement received for Packet:4
Packet:5 sent...    Acknowldgement received for Packet:5
Packet:6 sent...    Acknowldgement received for Packet:6
doubled cw
 cwnd:8
Packet:7 sent...    Acknowldgement received for Packet:7
Packet:8 sent...    Acknowldgement received for Packet:8
Packet:9 sent...    Acknowldgement received for Packet:9
Packet:10 sent...    Acknowldgement received for Packet:10
Packet:11 sent...    Acknowldgement received for Packet:11
Packet:12 sent...    Acknowldgement received for Packet:12
Packet:13 sent...    Acknowldgement received for Packet:13
Packet:14 sent...    Acknowldgement received for Packet:14
doubled cw
 cwnd:16
Packet:15 sent...    Acknowldgement received for Packet:15
Packet:16 sent...    Acknowldgement received for Packet:16
Packet:17 sent...    Acknowldgement received for Packet:17
Packet:18 sent...    Acknowldgement received for Packet:18
Packet:19 sent...    Acknowldgement received for Packet:19
Packet:20 sent...    Acknowldgement received for Packet:20
Packet:21 sent...    Acknowldgement received for Packet:21
Packet:22 sent...    Acknowldgement received for Packet:22
Packet:23 sent...    Acknowldgement received for Packet:23
Packet:24 sent...    Acknowldgement received for Packet:24
Packet:25 sent...    Acknowldgement received for Packet:25
Packet:26 sent...    Acknowldgement received for Packet:26
Packet:27 sent...    Acknowldgement received for Packet:27
Packet:28 sent...    Acknowldgement received for Packet:28
Packet:29 sent...    Acknowldgement received for Packet:29
Packet:30 sent...    Acknowldgement received for Packet:30
doubled cw
 cwnd:32
Packet:31 sent...    Acknowldgement received for Packet:31
Packet:32 sent...    Acknowldgement received for Packet:32
```

After the congestion window becomes larger than slow start threshold, the congestion window increases by one only after the client receives all the acknowledgements for all the packets sent in that congestion window.



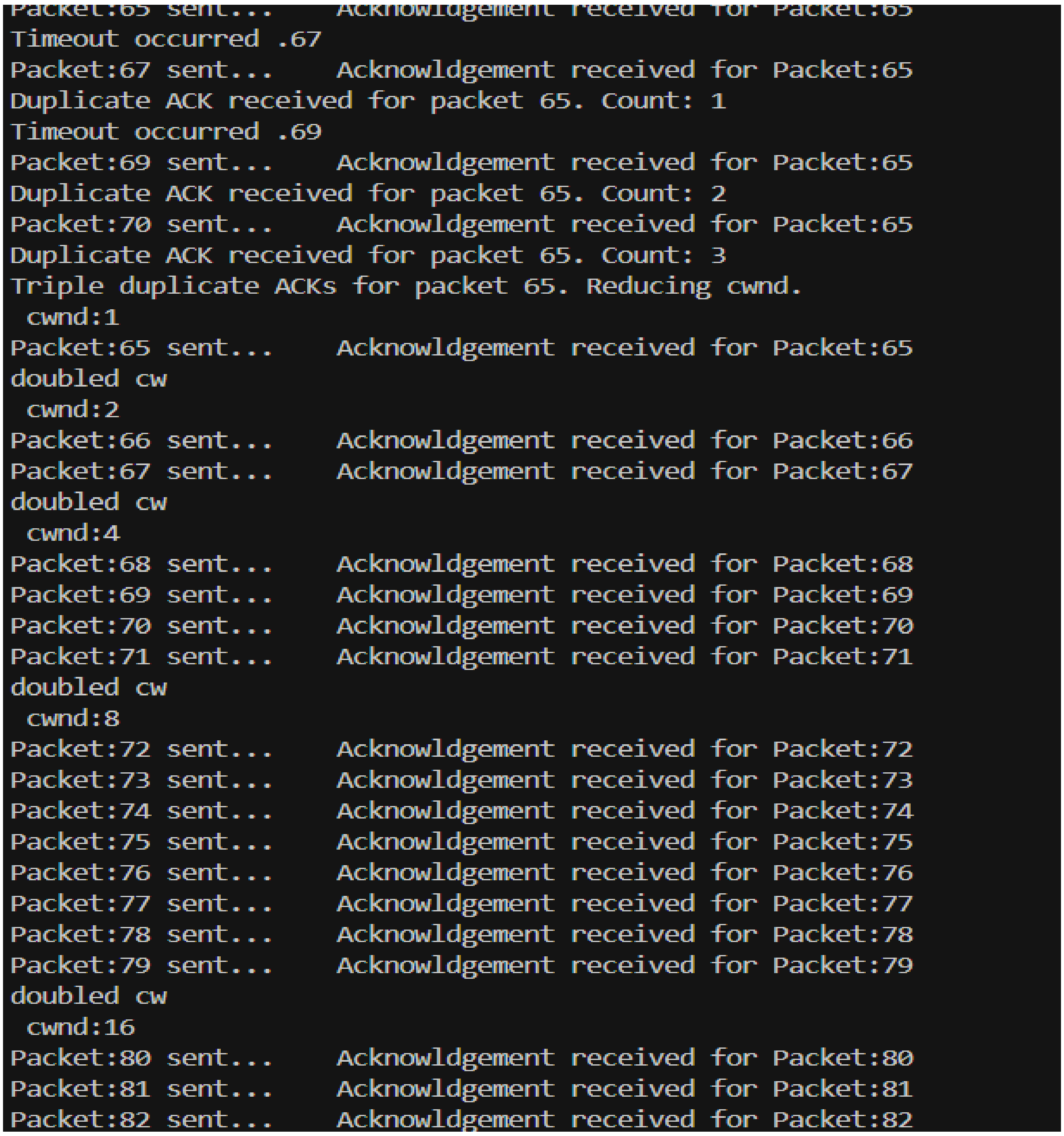
```
doubled cw
 cwnd:32
Packet:31 sent...    Acknowldgement received for Packet:31
Packet:32 sent...    Acknowldgement received for Packet:32
Packet:33 sent...    Acknowldgement received for Packet:33
Packet:34 sent...    Acknowldgement received for Packet:34
Packet:35 sent...    Acknowldgement received for Packet:35
Packet:36 sent...    Acknowldgement received for Packet:36
Packet:37 sent...    Acknowldgement received for Packet:37
Packet:38 sent...    Acknowldgement received for Packet:38
Packet:39 sent...    Acknowldgement received for Packet:39
Packet:40 sent...    Acknowldgement received for Packet:40
Packet:41 sent...    Acknowldgement received for Packet:41
Packet:42 sent...    Acknowldgement received for Packet:42
Packet:43 sent...    Acknowldgement received for Packet:43
Packet:44 sent...    Acknowldgement received for Packet:44
Packet:45 sent...    Acknowldgement received for Packet:45
Packet:46 sent...    Acknowldgement received for Packet:46
Packet:47 sent...    Acknowldgement received for Packet:47
Packet:48 sent...    Acknowldgement received for Packet:48
Packet:49 sent...    Acknowldgement received for Packet:49
Packet:50 sent...    Acknowldgement received for Packet:50
Packet:51 sent...    Acknowldgement received for Packet:51
Packet:52 sent...    Acknowldgement received for Packet:52
Packet:53 sent...    Acknowldgement received for Packet:53
Packet:54 sent...    Acknowldgement received for Packet:54
Packet:55 sent...    Acknowldgement received for Packet:55
Packet:56 sent...    Acknowldgement received for Packet:56
Packet:57 sent...    Acknowldgement received for Packet:57
Packet:58 sent...    Acknowldgement received for Packet:58
Packet:59 sent...    Acknowldgement received for Packet:59
Packet:60 sent...    Acknowldgement received for Packet:60
Packet:61 sent...    Acknowldgement received for Packet:61
Packet:62 sent...    Acknowldgement received for Packet:62
doubled cw
 cwnd:33
Packet:63 sent...    Acknowldgement received for Packet:63
Packet:64 sent...    Acknowldgement received for Packet:64
Packet:65 sent...    Acknowldgement received for Packet:65
Timeout occurred .67
Packet:67 sent...    Acknowldgement received for Packet:65
Duplicate ACK received for packet 65 , Count: 1
```

On the server side, we simulate loss of packet by sending acknowledgement of a previous packet.On the client side, timeout signifies that the acknowledgement was not received for a particular packet.



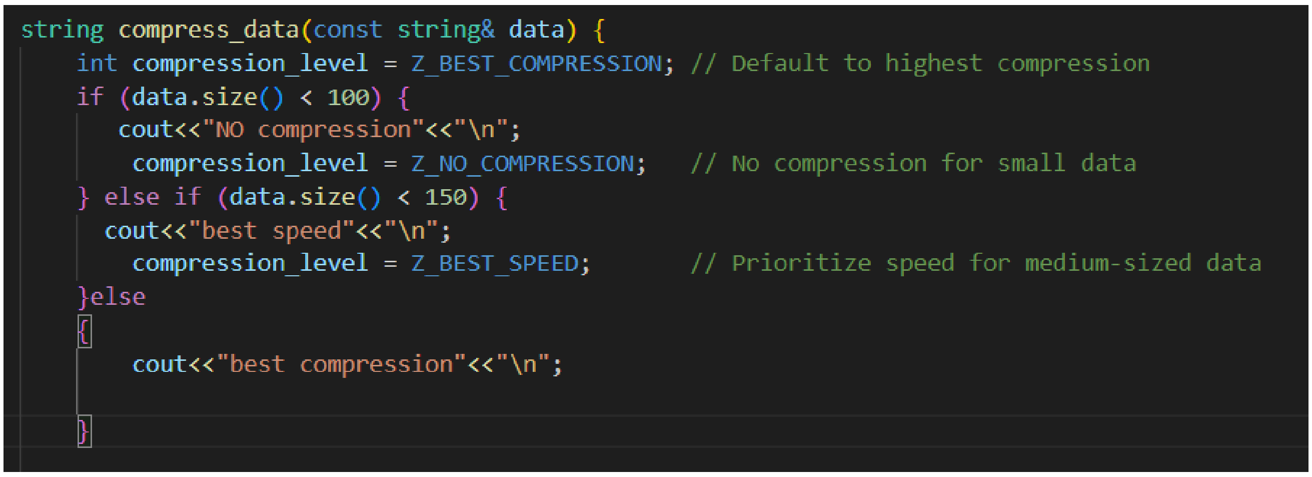
```
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa, Wind Speed: 11 m/s, Visi
bility: 9999 meters
Simulating packet loss for packet ID: 66 with cwnd: 33
Simulating packet loss for packet ID: 68 with cwnd: 33
Received weather data: Client 1: Temperature: 276 K, Humidity: 51%, Air Pressure: 1001 hPa
```

If client receives 3 such duplicate acknowledgements then it makes
Slow start threshold equal to (current congestion window)/2 and congestion window =1 and resends the packet lost. It again goes into the slow start part.



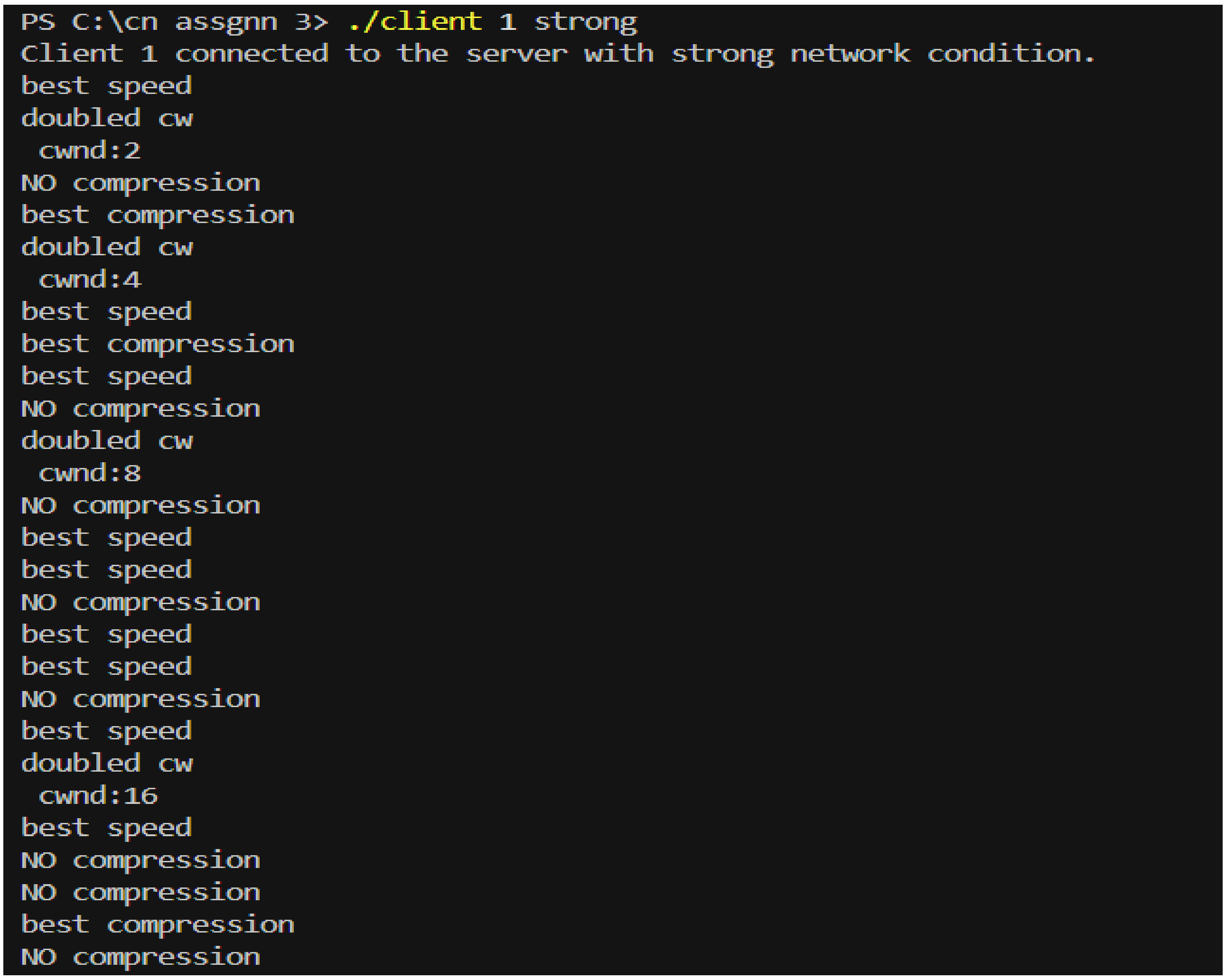
```
Packet:65 sent...    Acknowldgement received for Packet:65
Timeout occurred .67
Packet:67 sent...    Acknowldgement received for Packet:65
Duplicate ACK received for packet 65. Count: 1
Timeout occurred .69
Packet:69 sent...    Acknowldgement received for Packet:65
Duplicate ACK received for packet 65. Count: 2
Packet:70 sent...    Acknowldgement received for Packet:65
Duplicate ACK received for packet 65. Count: 3
Triple duplicate ACKs for packet 65. Reducing cwnd.
 cwnd:1
Packet:65 sent...    Acknowldgement received for Packet:65
doubled cw
 cwnd:2
Packet:66 sent...    Acknowldgement received for Packet:66
Packet:67 sent...    Acknowldgement received for Packet:67
doubled cw
 cwnd:4
Packet:68 sent...    Acknowldgement received for Packet:68
Packet:69 sent...    Acknowldgement received for Packet:69
Packet:70 sent...    Acknowldgement received for Packet:70
Packet:71 sent...    Acknowldgement received for Packet:71
doubled cw
 cwnd:8
Packet:72 sent...    Acknowldgement received for Packet:72
Packet:73 sent...    Acknowldgement received for Packet:73
Packet:74 sent...    Acknowldgement received for Packet:74
Packet:75 sent...    Acknowldgement received for Packet:75
Packet:76 sent...    Acknowldgement received for Packet:76
Packet:77 sent...    Acknowldgement received for Packet:77
Packet:78 sent...    Acknowldgement received for Packet:78
Packet:79 sent...    Acknowldgement received for Packet:79
doubled cw
 cwnd:16
Packet:80 sent...    Acknowldgement received for Packet:80
Packet:81 sent...    Acknowldgement received for Packet:81
Packet:82 sent...    Acknowldgement received for Packet:82
```

4. For simulating dynamic data compression we have made use of different levels of compression using zlib.

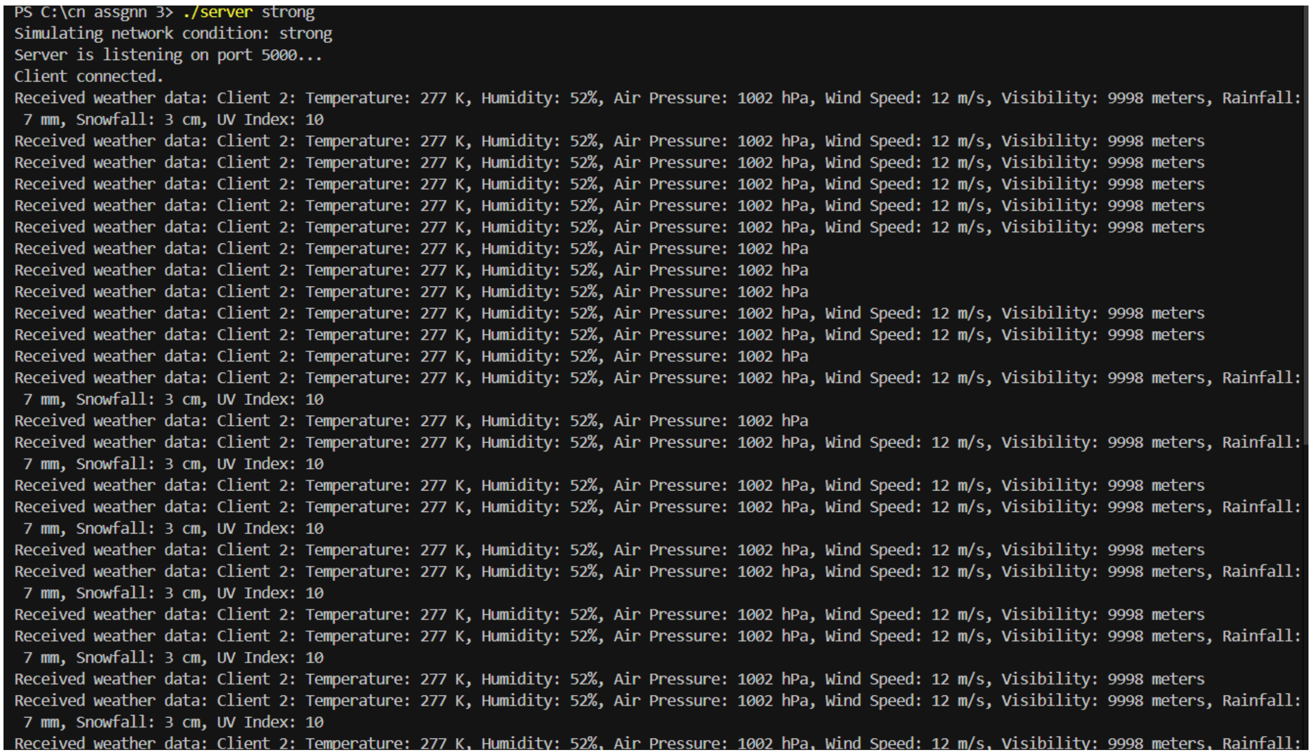


```cpp
string compress_data(const string& data) {
    int compression_level = Z_BEST_COMPRESSION; // Default to highest compression
    if (data.size() < 100) {
       cout<<"NO compression"<<"\n";
        compression_level = Z_NO_COMPRESSION;    // No compression for small data
    } else if (data.size() < 150) {
      cout<<"best speed"<<"\n";
        compression_level = Z_BEST_SPEED;        // Prioritize speed for medium-sized data
    }else
    {
        cout<<"best compression"<<"\n";

    }
```

If the size of data is less than 100 characters then no compression is done. If the size of data is between 100 to 150 characters then we use best speed compression. For data greater than 150 characters then we use best compression level.



```
PS C:\cn assgnn 3> ./client 1 strong
Client 1 connected to the server with strong network condition.
best speed
doubled cw
 cwnd:2
NO compression
best compression
doubled cw
 cwnd:4
best speed
best compression
best speed
NO compression
doubled cw
 cwnd:8
NO compression
best speed
best speed
NO compression
best speed
best speed
NO compression
best speed
doubled cw
 cwnd:16
best speed
NO compression
NO compression
best compression
NO compression
```



```
PS C:\cn assgnn 3> ./server strong
Simulating network condition: strong
Server is listening on port 5000...
Client connected.
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters, Rainfall:
 7 mm, Snowfall: 3 cm, UV Index: 10
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters, Rainfall:
 7 mm, Snowfall: 3 cm, UV Index: 10
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters, Rainfall:
 7 mm, Snowfall: 3 cm, UV Index: 10
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters, Rainfall:
 7 mm, Snowfall: 3 cm, UV Index: 10
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters, Rainfall:
 7 mm, Snowfall: 3 cm, UV Index: 10
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters, Rainfall:
 7 mm, Snowfall: 3 cm, UV Index: 10
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters, Rainfall:
 7 mm, Snowfall: 3 cm, UV Index: 10
Received weather data: Client 2: Temperature: 277 K, Humidity: 52%, Air Pressure: 1002 hPa, Wind Speed: 12 m/s, Visibility: 9998 meters, Rainfall:
```