COIS 4310 Assignment 4 RFC

RFC Information

Authors: Michael Nichol & Bobby Horth
Application Name: Chatroom
RFC Version: 1.3
Date: February 3rd, 2021

Application Summary

This application allows for a server to be run on a machine, and have any amount of clients simultaneously connected on separate threads with the ability to send messages to one another, publicly or privately over TCP sockets. The server manages messaging between the clients while each user thread manages incoming packets from their client. When a message is sent, the server verifies the checksum of the message that the client sends to ensure that each packet is complete as sent from the client, and requests another one if found faulty. The server runs on localhost, and the clients have the ability to connect to the server from any machine on the same LAN as the server.

Application Architecture

Client

- Attempts connection to TCP socket with the hostname “localhost” and the port 54004
- Creates a locking thread that determines whether or not the write thread can send a packet
- Creates simultaneous read and write threads
    - Read thread
        1. Loops infinitely reading packets sent from the server
        2. Upon reading a packet, ensures the checksum is valid, if not request another packet
        3. If packet data is encrypted with ROT47, decrypt it
        4. Displays packet data to the client
    - Write thread
        1. Loops infinitely following the pattern:
            - accept user input
            - ensures locking thread is released
            - check if it is required to send another packet via the thread lock, if not then continue

            - converts user input to the associated packet with contents and type
            - encrypts packet data with ROT47
            - sends packet to the server through socket
            - wait until read thread allows another packet to be sent

Server

- Creates a TCP socket with the port 54004
- Loops infinitely accepting connections to the socket
    - For each connection creates a user thread
        - User thread loops infinitely awaiting packets from the client
        - Upon reading a packet, verifies the packet checksum for the data portion
            - If packet is corrupted, sends a request resend to the client
            - If packet is not corrupted, server processes the packet accordingly and sends acknowledgement packet to client verifying that the packet sent correctly

Packets

Packet Header

Protocol: TCP
Source: localhost
Source Port: 54004
Destination: localhost
Destination Port: 54004
ID: {packet id}
Version: 1
Data: {message}
Sender: {sender username}
Destination: {destination username/command}
Verb: {verb}
Checksum: {hashlib.md5 of ‘Data’}
EncryptionType: {name of encryption type}

Packet Types

ID	Verb	Purpose
1	username	Sent to the server when a client has chosen their username

2	broadcast	Sent to the server when the client wishes to broadcast a message to all other users Sent to the client upon receiving a broadcast from another user
3	private message	Sent to the server when the client wishes to send a private message to another user Sent to the client upon receiving a private message from another user
4	client list	Sent to server when a client wishes to know what other users are connected Sent to the client as a reply containing the client list
5	disconnect	Sent to the server before the client disconnects Sent to the client to notify them another user disconnected
6	user not found	Sent to the client as a reply if their destination user could not be found
7	valid packet	Sent to the client to verify that the packet sent correctly
8	corrupted packet	Sent to the client to inform them that the previous packet did not send correctly, and request the original packet again

Commands and Responses

Client Commands

Command	**Action**
all	1. Ensures user input matches command pattern 2. Separates user input into command and message 3. Creates a Packet containing the message with associated “disconnect” packet type 4. Sends packet to the server
who	1. Ensures user input matches command pattern 2. Creates a Packet with associated “clientList” packet type 3. Sends packet to server
username	1. Ensures user input matches command pattern 2. Separates user input into command and message 3. Creates a Packet with associated “privateMessage”

	packet type, containing target user, as well as the message contents 4. Sends packet to server
bye	1. Ensures user input matches command pattern 2. Creates a Packet with associated “disconnect” packet type 3. Sends packet to server

Server Response

Verb	**Action**
username	1. Gets the users username and connection from the packet 2. Adds the (username, connection) entry into dictionary of online users 3. Begins a thread on the server for that user
broadcast	1. Gets broadcast packet 2. Iterate over key, value pair in online users dictionary 3. Sends broadcast packet to each user containing the message
privateMessage	1. Determines if the target user exists in the dictionary of online users 2. If user exists a. Send privateMessage packet to the destination user 3. If user does not exist a. Send userNotFound packet to the sender user
clientList	1. Iterates over key, value pair in online users dictionary 2. Appends each users display name to string 3. Sends clientList packet to sender user containing the list of clients
disconnect	1. Removes the user from the dictionary of online users 2. Iterates over key, value pair in online users dictionary 3. Sends disconnect packet to each user containing the username of the user who disconnected

Client Response

Verb	**Action**
broadcast	1. Outputs a message telling the client they received a

	broadcast from the sender user along with the message
privateMessage	1. Outputs the username of the sender user along with their message
clientList	1. Checks the size of the client list sent by the server 2. If there are no clients in the list a. Outputs that the chat room is empty 3. If there are clients in the list a. Outputs the list of clients
disconnect	1. Outputs a message telling the client that the sender user has disconnected from the chat
userNotFound	1. Outputs a message telling the client the username they attempted to private message could not be found
acknowledge	1. Allow the write thread to send another packet by releasing the thread lock
corrupted	1. Inform the write thread to resend the last packet and activate the thread lock