Due Date: 10/30/2020

Upload to Canvas

# CS 5133: Data Networks
# Project-2
# Hamming Distance and Checksum using Message Relay System

Fall - 2020

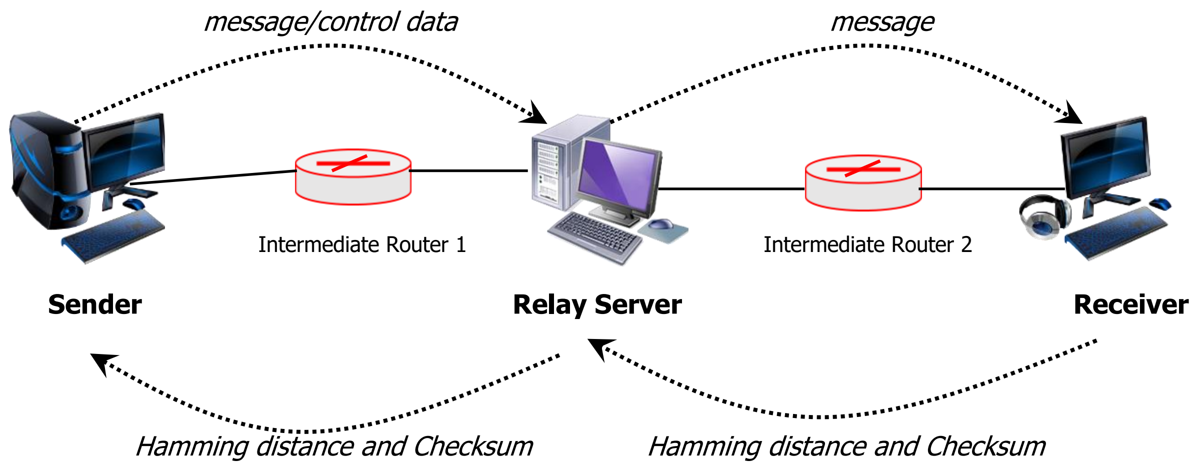
100 pts

## 1. Objective

The objective of this first programming project is to learn TCP iterative client-server interaction using socket interface in C programming language. After completing the project, you will have a basic understanding of the steps required to develop a networking application.

## 2. Project Specification

In this programming project, you are required to implement a *Hamming Distance and Checksum* based on simple message relay client-server application using C programming. The sender process accepts user-entered "messages" from the keyboard and sends these messages to the receiver process via the message-relay process. The receiver process displays the message and performs Hamming Distance and Checksum computations once the message has been received entirety. All communications among the processes are done using TCP sockets. There are two application-level messages defined: a DATA message (containing the text of a message) and a CLOSE message, which instructs the receiver of the CLOSE message to shut down. The relay process diagram and more specific project description are given below:



## 2.1. Sender process detail specifications:

- The sender process establishes a TCP connection with the relay server. Your sender program needs to take two arguments that specify the IP address and port number of the relay server that it is trying to connect.
- After a successful connection, your sender program will first prompt a welcome message that asks the user to enter a *username* using the keyboard. This username will then be sent to the relay server. After receiving the username from your sender, your relay server will send an acknowledgment message back to the sender.
- Your sender, after receiving the acknowledgment message from your server, will prompt a message that asks the user to enter the corresponding *password*. This password will then be sent to the server. Then, your server, after receiving the password from the sender, will verify the received pair of username and password against the list of legitimate pairs. If the result is positive, the server will send a success message to the sender. If the result is negative, the server will send an error message to the sender and wait for the new pair.
- After successful authentication, your sender program will take the name and port number of the receiver from the user using the keyboard. This pair will be sent to the server to verify the received pair of name and port number against the list of legitimate pairs for selecting the receiver. If the result is positive and a connection between the relay server and receiver has been set up using the IP address and port number of the receiver, the server will send a success message to the sender. If the result is negative, the server will send a failure message to the sender and wait for the new pair of name and port number to connect with the receiver.
- When connections between sender, relay server, and receiver are complete, the sender process prompts user for "**messages**" (e.g., containing characters entered by the user via keyboard) and sends each message to the relay and waits for reply from relay server. *Note*: because of the byte-stream semantics of the TCP socket, your sender process will need to define an application-layer DATA message that lets the relay process know how many characters are in the incoming byte stream.
- When the messages are sent from relay server to the receiver, the receiver will apply a simple algorithm to calculate the **Hamming Distance** for every two messages and **Checksum** for every message transmitted. This result will be sent back to the sender process. The sender process will print the result into the output.
- Finally, when the user wants to indicate that there are no more messages to send, the sender process should send a CLOSE message to the intermediate relay, and then terminate itself gracefully (i.e., releasing any sockets it has been using). User may type 'x' or 'q' to terminate from the sender program.

## 2.2. Intermediate-relay server process specifications:

- During the initial startup, the server program needs to take an argument that specifies the port that it is listening to. You have to use (*5000+last 4 digits* of your student-id number) to avoid requesting same port by multiple students.
- Your relay server process is responsible for verifying the user authentication and receiver name to IP address resolution. User and receiver list text files are shown below in which data are separated by a black space (i.e., between username and password there is a blank space to separate them) and each line is separated by a newline:

**userList.txt**

| Username | Password |
|---|---|
| Anna | a86H6T0c |
| Louis | G6M7p8az |
| Cathie | Pd82bG57 |
| Ken | jO79bNs1 |
| Sam | Cfw61RqV |
| Bailey | Kuz07YLv |

**receiverList.txt**

| Receiver Name | IP Address | Port Number |
|---|---|---|
| gpel8.cs.ou.edu | 129.15.78.11 | ServerPortNumber+1 |
| gpel9.cs.ou.edu | 129.15.78.12 | ServerPortNumber+1 |
| gpel10.cs.ou.edu | 129.15.78.13 | ServerPortNumber+1 |
| gpel11.cs.ou.edu | 129.15.78.14 | ServerPortNumber+1 |
| gpel12.cs.ou.edu | 129.15.78.15 | ServerPortNumber+1 |
| gpel13.cs.ou.edu | 129.15.78.16 | ServerPortNumber+1 |

- After establishing a connection with the sender and receiver processes, your server process reads messages sent by the sender process. It forwards DATA messages to the receiver process over a TCP socket.
- Similarly, your server process reads messages sent by the receiver process. It forwards the resulting data messages to the sender process over a TCP socket.
- When the intermediate relay server receives a CLOSE message from the sender process, it should send a CLOSE message to the receiver process so that the receiver process can gracefully terminate itself.
- The intermediate relay server process continues to run regardless of the CLOSE message send by the sender process.

### 2.3. Receiver process specifications:

- During the initial startup, your receiver program will take one argument that specifies the port number that it is listing to. You have to use your *relay server port number + 1* as the receiver port number. Your receiver program acts more like a server.
- This process reads messages sent by the intermediate-relay process. It displays the content of each DATA message once that message has been received entirety.
- Next, your receiver program requires implementing a simple algorithm to calculate the Hamming Distance of every two messages and Checksum for every message transmitted by the sender. *Note:* you may refer to *https://www.geeksforgeeks.org/hamming-distance-two-strings/* *https://en.wikipedia.org/wiki/IPv4_header_checksum*

  **<u>Hint</u>**: First, the relay server will forward the message from the sender to the receiver. The receiver needs to wait for the <u>second message</u> from the sender to calculate Hamming distance while it calculates Checksum. This can be done by using **arrays**. The receiver can store every message received from the sender in arrays, and whenever the array index reaches a *multiple of 2*, the receiver should calculate the **hamming distance** between the current and the previous message received from the sender through relay server.

- These results need to be sent back to the sender process via the relay server. It's a good idea to deploy structures for handling messages and results.
- Finally, on receipt of a CLOSE message, the receiver process should terminate itself gracefully.

## 3. Programming Notes:

I would strongly suggest that everyone begin by writing the sender and relay processes first, i.e., just getting the two of them to interoperate corrections. Then modify the relay and program the receiver. You will need to know your machine's IP address, when one process connects to another. You can telnet to your own machine and see the dotted decimal address displayed by the telnet program. You can also use the either *ifconfig or hostname -I or ping (e.g., ping gpel8.cs.ou.edu)* commands to figure out the IP address of the machine you are working on. If you need to kill a process after you have started it, you can use the *kill* command. Use the *ps* command to find the process id of your server.

Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port ID's, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use though it's prohibited.

You should program your each process to print an informative statement whenever it takes an action (e.g., sends or receives a message, detects termination of input, etc.), so that you can see that your processes are working correctly (or not!). One should be able to determine from this output if your processes are working correctly. You should hand in screen shots (or file content, if your process is writing to a file) of these informative messages.

Since the goal of the programming project is for you to learn socket programming (not to build a hardened application), you may also assume that the relay and the receiver will only be receiving messages from a single sender or relay, respectively. You are required to work on this project alone. Please use the "*Projects*" discussion group in the canvas if you have any general questions regarding this project.

### File names:

Make sure you follow the file name guideline given below for your project:

*lastNameP1Sender.c, lastNameP1Server.c, and lastNameP1Receiver.c, userList.txt, and receiverList.txt*

## 4. Points Distribution:

| Bits and pieces | Points |
|---|---|
| Client Program | 20 |
| Relay Server | 35 |
| Receiver Program | 25 |
| Program Style (Coding style, comments etc.) | 10 |
| Documentation | 10 |

## 5. Submission Instructions:

This project requires the submission of a *soft copy.*

**Soft Copy (Due October 30, 2020, 11:59 pm)**

The soft copy should consist of:

- source code of the Client program,
- source code of the Relay Server program,
- source code of the Receiver program,
- any header file(s), and
- detailed documentation should consist of:
    - discussion of your problem-solving approach
    - detailed analysis of data structures, algorithms, and user define functions
    - any legitimate assumption(s) with justification, and
    - screen shots of outputs

These must be submitted through Canvas (http://canvas.ou.edu).

## 6. Late Penalty:

You have to submit your project on or before the due date to avoid any late penalty. **A late penalty of *15% per day* will be imposed after the *due date***. After one week from the due date, you will not be allowed to submit the project under any circumstances.

**Good Luck!!**