Overview:

You are tasked with building a **peer-to-peer (P2P) communication system** using C or C++, which simulates a botnet-like structure. The goal is to design a **store-and-forward message server** and a **client** that can participate in a distributed network, where servers route messages between each other in a decentralized way.

General Goals:

- Build a simple message passing system where servers exchange messages.
- Allow clients to connect to a specific server and send/receive messages.
- Implement a peer-to-peer architecture where servers communicate with each other and maintain connections to form a "botnet."

Key Parts of the Project:

1. Programming Language & Restrictions:

- **Language**: C or C++.
- **Library restrictions**: Boost.Asio is *not* allowed, though Boost libraries (for string handling, etc.) are permissible.
- You must implement the **TCP-based communication** using the BSD Socket API.

2. Group Setup:

- You need to join a group on **Canvas**, which determines your **Group ID**.
- You can work solo or in teams of 2.
- This **Group ID** is also used to identify your server.

3. Program Details:

- You need to develop a **client-server application**.
- **Client** connects to the server, issues commands, and interacts with other servers via the botnet network.
- **Server** accepts commands from both the client and other servers.

4. Networking Protocol Details:

- Communication is **TCP-based**.
- You'll define a simple **custom protocol** for commands and message exchanges.

- Commands/messages must use special delimiters: **ASCII 0x01 (SOH)** as the start of a message, and **ASCII 0x04 (EOT)** for the end of a message.

5. Project Structure:

- **Client**: Issues commands like sending or receiving messages, asking for server status, or listing connected servers.
- **Server**: Handles incoming messages from both the client and other servers. Manages message queues, routes messages to appropriate servers, and logs activity.

Server Specification:

Your server needs to:

1. **Listen on a TCP port** for connections from other servers and clients.
2. **Communicate with other servers** using predefined commands (see next section for details).
3. **Maintain connections** with 3 to 8 other servers at all times to form the botnet.
4. **Log** all commands and actions.

Commands Overview:

Here are the commands your server must handle:

Between Servers:

1. **HELO, <FROM GROUP ID>**: Initiates a connection between servers.
   - The server replies with the list of connected servers using the **SERVERS** command.
2. **SERVERS**: Provides a list of directly connected servers.
3. **KEEPALIVE, <No. of Messages>**: Sent periodically to indicate the number of messages waiting for a connected server. (At most once per minute).
4. **GETMSGS, <GROUP ID>**: Fetches messages for the specified server.
5. **SENDMSG, <TO GROUP ID>, <FROM GROUP ID>, <Message content>**: Sends a message to a specified server.
6. **STATUSREQ/STATUSRESP**: Server status check.

Between Client and Server:

1. **GETMSG, <GROUP ID>**: Client fetches a message from the server.
2. **SENDMSG, <GROUP ID>, <Message content>**: Client sends a message to the server.
3. **LISTSERVERS**: Lists servers connected to the botnet.

Submission Requirements:

1. **Client-Server Communication (4 points)**:
   - Build the client and server.
   - Implement commands for the client to send messages and get responses from the server.
2. **Wireshark Trace (1 point)**:
   - Capture a Wireshark trace showing communication between your client and server.
3. **Instructor Server Connection (1 point)**:
   - Your server should successfully communicate with an instructor's server.
4. **Group Communication (2 points)**:
   - Your server should receive and send messages to at least two other groups' servers.
5. **Code Submission (1 point)**:
   - Submit a single zip file with the source code, Makefile, README, and additional logs or traces.
   - README should explain how to compile and run the code.
6. **Code Documentation (1 point)**:
   - Write clean and well-documented code with meaningful logs to help debug.

Bonus Points (Optional):

You can earn up to **5 bonus points** beyond the regular 10 points. A few ways to earn bonus points:

- **Submit Early (1 point)**: Submit the basic implementation within the first week.
- Additional bonus points can be earned for extra functionalities like improved routing algorithms, handling large message volumes, etc.

Steps to Tackle the Assignment:

1. **Setup the Server-Client Skeleton:**
    - Start by setting up the server to listen on a TCP port and accept connections.
    - Implement basic client-server communication using sockets.
2. **Implement Core Commands:**
    - Implement the HELO and SERVERS command to facilitate connections with other servers.
    - Implement SENDMSG and GETMSGS for message passing.
3. **Log and Debug:**
    - Make sure to log all received and sent commands, as this will help in debugging and will be part of your submission.
4. **Test with Other Groups:**
    - Coordinate with classmates to test communication with at least two other servers.
    - Ensure you handle message expiry, keeping connections alive, and other network concerns.
5. **Wireshark Trace:**
    - Use Wireshark to capture traces of your client-server communication for submission.
6. **Submit Properly:**
    - Ensure the code is well-structured, documented, and that your README is clear on how to compile and run the project.