I. Documentation

Summary of the implementation of the UdpRelay class:

The UdpRelay object is implemented using three persistent threads (which run continuously while the object exists), and a variable number of connection specific threads (which run while a connection with a remote network remains open). The three persistent threads include: acceptsThread, commandThread, and relayInThread. The connection specific thread is the relayOutThread. The UdpRelay object uses exactly one instance of a Socket object, and exactly one instance of an UdpMulticast object. The Socket object is used to facilitate sending and receiving messages with a remote network, and the UdpMulticast object is used to facilitating sending and receiving message within the local network. Three standard library maps are used to keep track of connections that the UdpRelay object has accepted, added, and the relayOutThreads associated with each connection.

The acceptsThread uses an infinite loop to wait for and accept connection requests from a remote network through the Socket object. When a connection request is received (via Socket.getServerSocket()), the thread identifies the remote network and checks if a connection with that remote network already exists. If a connection already exists, the thread will close and eliminate the old connection before opening the new connection. The corresponding maps are then updated and the loop continues, waiting for the next connection request.

The commandThread uses a loop to wait for user entered input via the keyboard through standard input (cin). It uses standard library functions to process the command, breaking it down into tokens if necessary, and validating the format of the commands before calling a helper method to do the work associated with each command. The “add” command is similar to the acceptsThread, but instead of receiving a connection this command initiates the connection request to a remote network. The “delete” command removes a remote connection that was previously added by the same UdpRelay object.

The relayInThread first opens an UdpMulticast connection as the server with the local network. Then, it uses an infinite loop to wait for messages to be received from the local network. When a message is received, it checks the message header to determine if the message has already been processed by the local network. If the message is new to the local network, then the header is modified and the message is sent to any remote networks that are connected to the UdpRelay object. To synchronize the message processing with the relayOutThread, the relayInThread uses a mutex on the message buffer, and signals the relayOutThread once the message has been fully processed.

The relayOutThread uses an infinite loop to receive messages from a remote network. Using a mutex on the message buffer, it processes the message by first checking the header to see if the message has already been processed locally. If not, then the message is printed as received

and broadcast locally using the UdpMulticast object. The relayOutThread then waits for the relayInThread to finish processing the message buffer before printing its final message and returns to the beginning of the loop.

Several helper methods were implemented to help with the processing of the message buffer, as well as converting the address strings to and from network bytes.

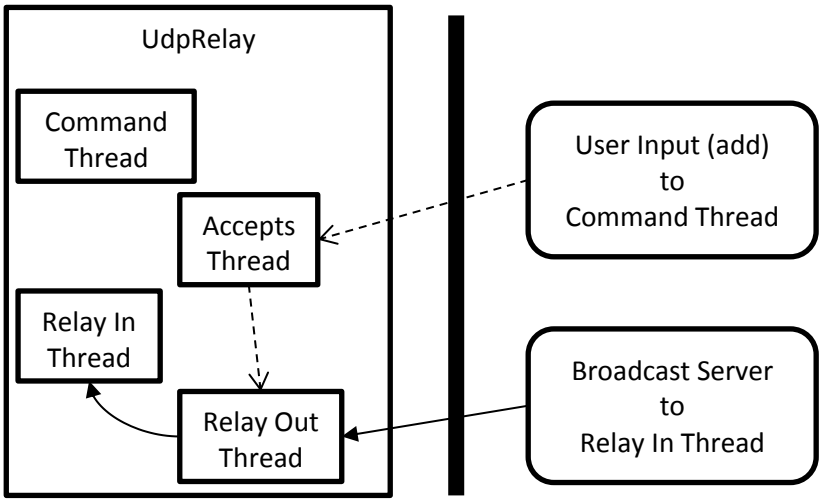

Figure 1.1: Sample Data Flow Diagram with UdpRelay Object

Figure 1.1 above shows a sample of how the UdpRelay object is used in communication with remote networks. First, the user on a remote network enters the "add" command for the local network. This triggers the local accepts thread to add the connection, spawning a new relay out thread. Then, a broadcast server on the remote network sends a message, which will be first picked up by the remote network's relay in thread. This thread then sends the message to its remote networks which is picked up by the relay out thread of the local UdpRelay object. The relay out thread broadcasts the message to its relay in thread for further processing.

II. Source Code

Please find the source code files `UdpRelay.h` and `UdpRelay.cpp` uploaded with the report. The source code is not included in this report because it would lose formatting, indentation, and other styles that improve readability.

III. Execution Output

Please find the execution output files `execution-script.txt`, and `execution-screenshot.png` uploaded with this report. The execution script text file show the order of command entered into each of the SSH terminals. The execution screenshot picture file show the results when the execution script is run. Also, the `typescript` file attached to this report show the successful execution of all the user commands.

IV. Discussion

Limitation of your program:

The implementation of the UdpRelay class has a few limitations. The first is a scenario where a message can take multiple paths from a BroadcastClient to a specific network group (see Figure 4.1). In the example provided below. A message could be sent directly from 1 to 3 or indirectly from 1 to 2 to 3. For this version of the UdpRelay program, the message would be received and processed twice by network 3. The message coming from 1 to 3 would be processed because 3's address would not be in the header of the message coming from 1. Also, the message from 2 to 3 would be processed because 3's address would not be in the message coming from 2. This is a problem because it is the same message. A potential solution would be to change how the UdpRelay program modifies and checks the header to see if it should accept and process a message. We may not easily be able to do this though, because it seems that an UdpRelay object needs to have a wider view of the entire network graph, not just the segments it is connected to but also the segments that other segments are connected to.

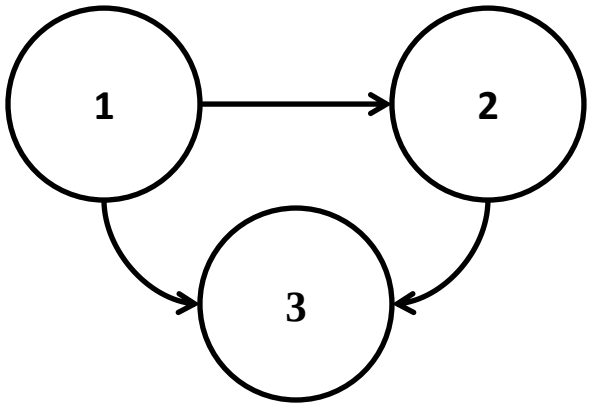

Figure 4.1: Multiple Paths from Source to Destination

Another limitation may occur when the wide network becomes very large and there are many UdpRelay objects connected to each other. The overhead associated with the UdpRelay may slow down the message transmission, and the size of the header of the message could grow very large. Also, if there are a lot of commands to add and delete remote networks, there could be problems with the timing of messages received by threads whose existence is being changed. This UdpRelay implementation is also limited by the direction which messages can be sent, it does not allow for 2-way communication between remote networks.

Possible extensions of your program:

It almost seems like the idea of an UDP Relay could be used to facilitate network traffic through a router from the internet (WAN) to some host on the LAN. The router is the machine that runs the UdpRelay program, and it is connected to other routers over the WAN which area also

running the UdpRelay program. A machine on the LAN connected to a router can send a message (similar to the BroadcastClient program in our assignment), which could be sent through the router to another router and its LAN. This router implementation of the UdpRelay is more complex than the implantation for this assignment, as the router must be able to handle incoming and outgoing messages with the same remote network.

Execution Screenshot Analysis:

The execution screenshot picutre included in this report shows the UdpRelay program working to facilitate the transmission of messages from 1 machine to 1 or 2 other machines. Figure 4.2 below is a simplified diagram that shows some of the data flow between 2 machies.

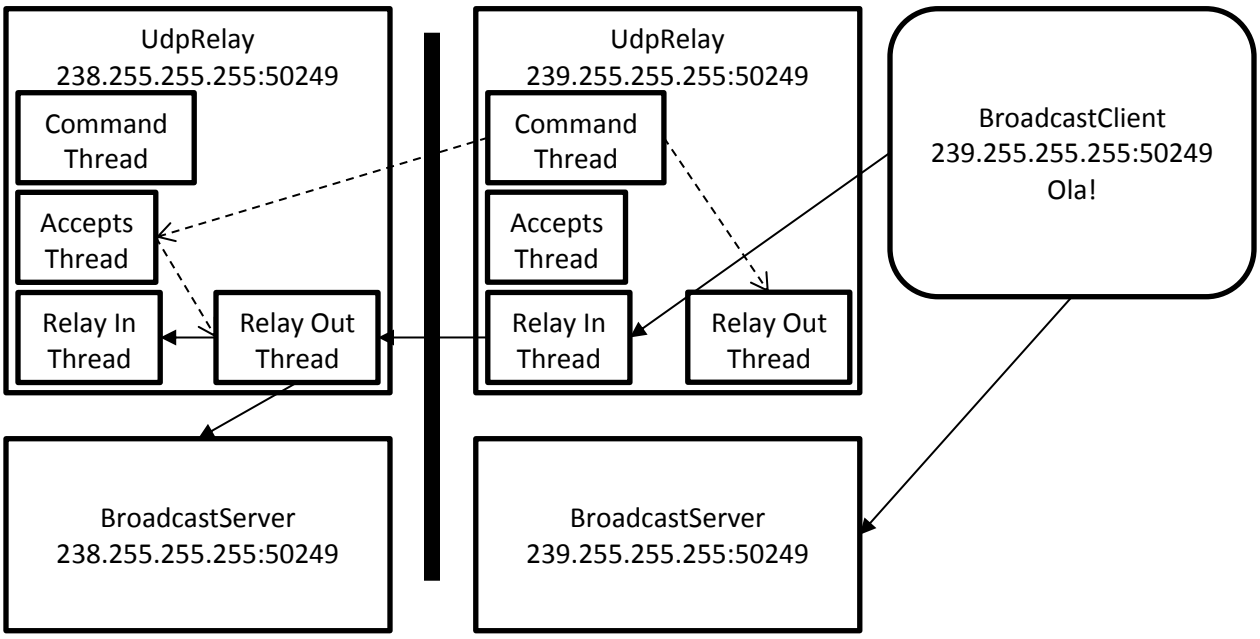

Figure 4.2: Simplified Execution Output Data Flow Diagram

First, the command thread for one of the UdpRelay programs receives the add command from the user to add a remote network (e.g. “add uw1-320-10:50247”). This triggers the acceptsThread on uw1-320-10 to open the connection and spawn a relayOutThread associated with that connection. Then, a BroadcastClient on the same network as the first UdpRelay machine (uw1-320-13) generates a message to be sent. The message is picked up by the BroadcastServer on the same network group, and the message is picked up by the relayInThread on the UdpRelay on uw1-320-13, which forwards the message to the remote network UdpRelay via TCP socket. The remote UdpRealy picks up the message in the relayOutThread, where it is broadcast locally and picked up by the BroadcastServer on that network, as well as the relayInThread on the same machine as the UdpRelay program.