CSS 503

Assignment 4

Chang Liu

Cliu2015a@gmail.com

**Program Report**

1. **Documentation**

The UdpRelay object is implemented using three persistent threads (which run continuously while the object exists), and a relayOut threads (which run while a connection with a remote network remains open). The three persistent threads include: **acceptsThread**, **commandThread**, and **relayInThread**. The UdpRelay object uses one instance of a Socket object, and one instance of an UdpMulticast object. The Socket object is utilized to help sending and receiving messages with a remote network, and the UdpMulticast object is utilized to help sending and receiving message within the local network. UdpMulticast , UdpRelay and Socket these three standard library maps are used to set up connections.

The acceptsThread utilizes an infinite loop to wait for and accept connection requests from a remote network through the Socket object. When we receive a connection request (through 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 ListenToList is then updated and the loop continues, waiting for the next connection request.

The commandThread utilizes a loop to wait for user entered input via the keyboard through standard input (cin). It utilizes standard library functions to process the command, separating it into pieces of strings when necessary, and validating the format of the commands before calling a helper method to do the work linked with each command. The “add” command is like the acceptsThread. Rather than 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 opens an UdpMulticast connection as the server with the local network. Then, it utilizes 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 whether 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.

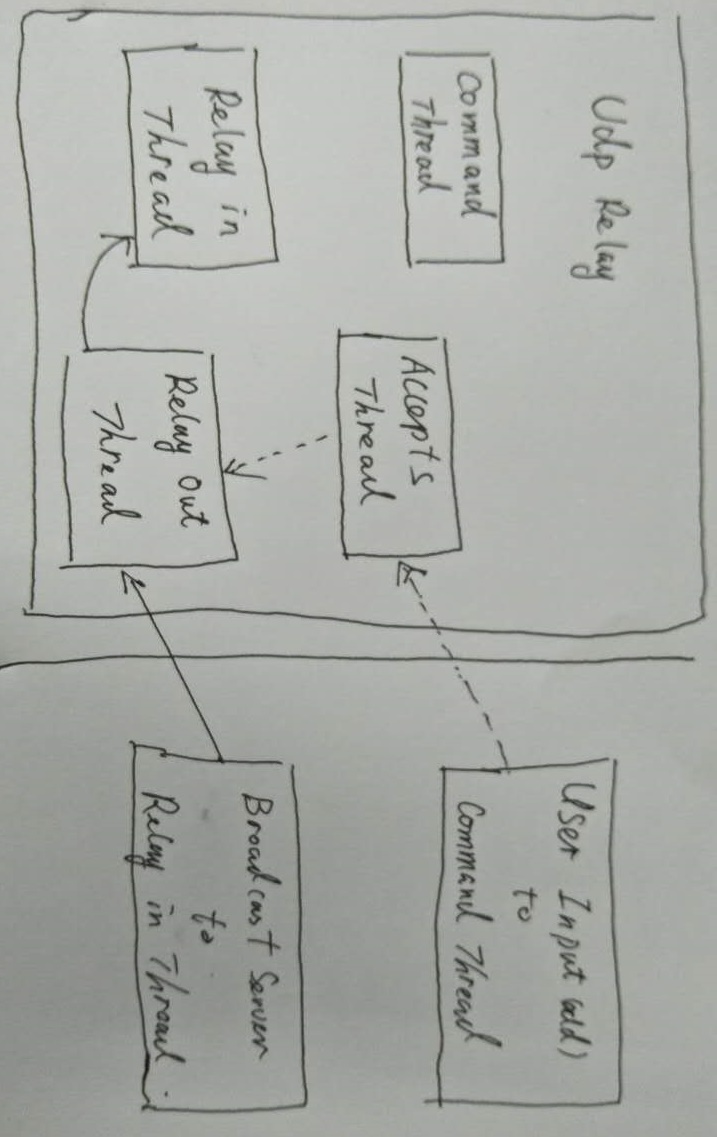
The relayOutThread utilizes an infinite loop to receive messages from a remote network. It processes the message by first checking the header to see whether the message has already been processed locally. If not, then the message is printed as received and broadcast locally using the UdpMulticast object. 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: Sample Data Flow Diagram with UdpRelay Object

Figure 1 shows a sample of how the UdpRelay object is utilized in communication with remote networks. Firstly, 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. Then this thread 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.

1. **Execution Output**

Please see the execution output screenshot.png uploaded with this report. The execution screenshot picture shows the results when the execution script is run.

1. **Discussion**
2. **Limitations**

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 2). 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. We could use a possible solution to change how the UdpRelay program modifies and checks the header to see whether it should accept and process a message. But it might be difficult for us to do this. Because an UdpRelay object usually needs to have a wider view of the entire network graph, not just the parts it is connected to but also the parts that other parts are connected to.

**1**

**2**

**3**

The second limitation might happen when the wide network becomes very large and many UdpRelay objects are 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. My message size is only 1024 which is not very large. It can not sent and very large messages.

**Possible Improvment:**

1. It seems like the idea of an UDP Relay could be utilized 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 complication than the implantation for this assignment, as the router must be able to handle incoming and outgoing messages with the same remote network.

1. The program has lots of memory leaks, which will affect the program performance in a long rum. However, there are not enough time to fix this problem.

3. relayOut threads got killed even they are not listening any messages any more. Whereas, the same reason, not enough time to improve that.