Implementation Designs

P2P

For our peer-to-peer implementation, we focused on turning every node within the network into an independent data node. Each node would be able to connect to every other node to send and receive data. For this to happen, we modified our original client script to not only be able to send to one individual server, but to many different "servers", which in this case are our other nodes. While sending and receiving specified information from the other nodes, we also made sure to periodically send out a signal every few seconds to determine whether a node in the network is down or not. To connect all the nodes to each other, each node is provided with a config file that lists all the IPs and port numbers of the other nodes contained within the network. To make sure that everything is running at once, we also implemented multithreading to have all the nodes perform their respective duties at the same time.

Client-Server

For our client-server implementation, we focused on creating a file that would exclusively listen for packets and print out their information to accurately determine whether we'd be receiving packets from the other nodes within the network. Instead of each node being connected to each other like in our peer-to-peer implementation, each node was connected to the server. To accurately send out information about other nodes to nodes that aren't the server, we also made sure to give the server the ability to send packets back to the nodes it's connected to based on the data it receives from other packets. For the config file for the client-server implementation, we have the client nodes all pointing towards the server node to send and receive data, while for the server node we have a config file that points to the IPs and port numbers of every other node in the network. Just like with the peer-to-peer implementation, the client-server implementation also uses multithreading to accurately and efficiently run all its processes at once.

Testing Logs

Since we had unfortunate troubles with our virtual machines, most of the testing was performed on our local machines. Below are screenshots of console outputs that we received when testing different files and behaviors.

Example output from thread testing.

```
Thread A is running in Thread: Thread-0
Thread D is running in Thread: Thread-3
Thread B is running in Thread: Thread-1
Thread C is running in Thread: Thread-2
```

Example output from fetching a directory's file listing

Case 1: Directory points to java files

Case 2: Directory does not exist

```
ERROR: Directory not found: nonexistantdirectory
[]
```

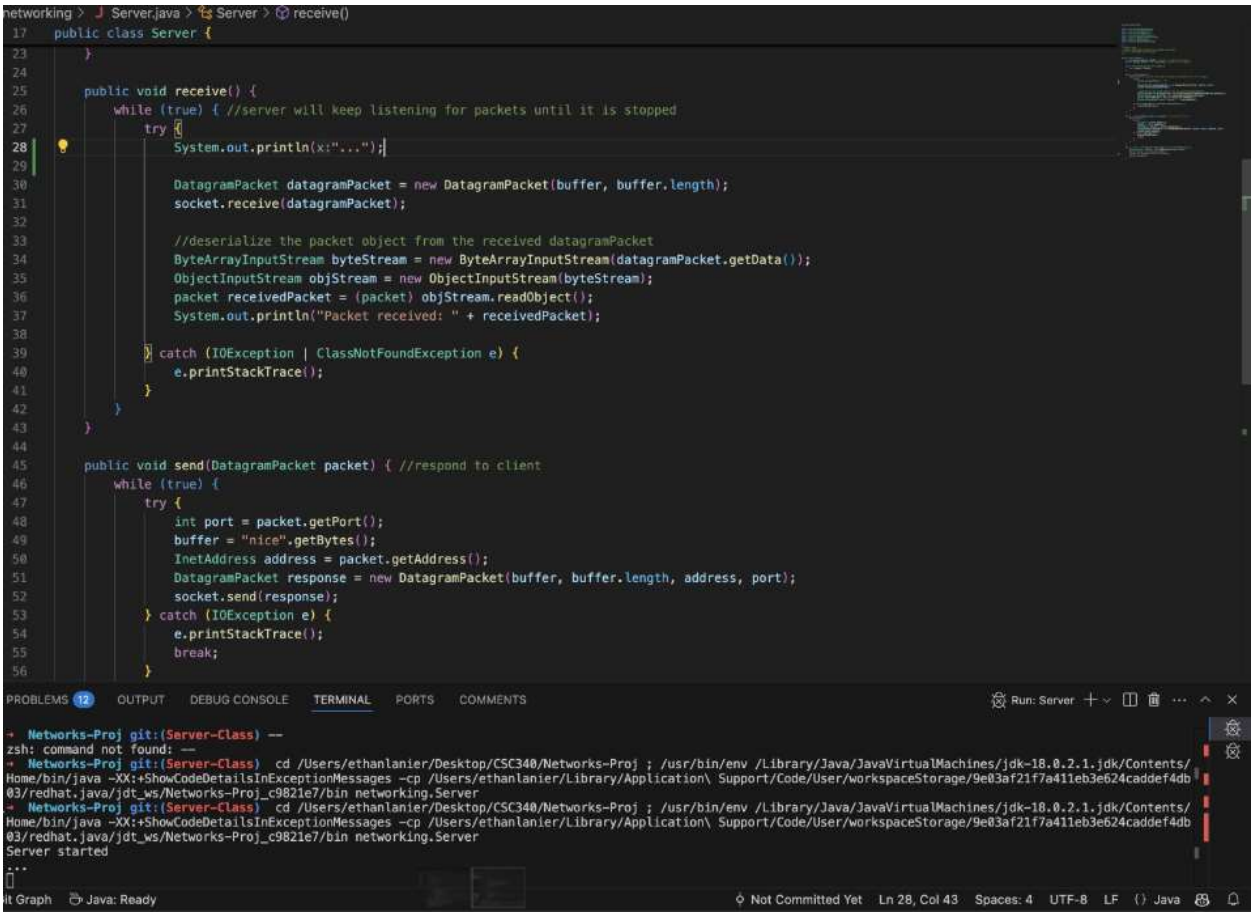
Case 3: Directory has no files or subdirectories

```
ERROR: No files in directory provided: C:\Users\kylem\OneDrive\Documents
[]
```

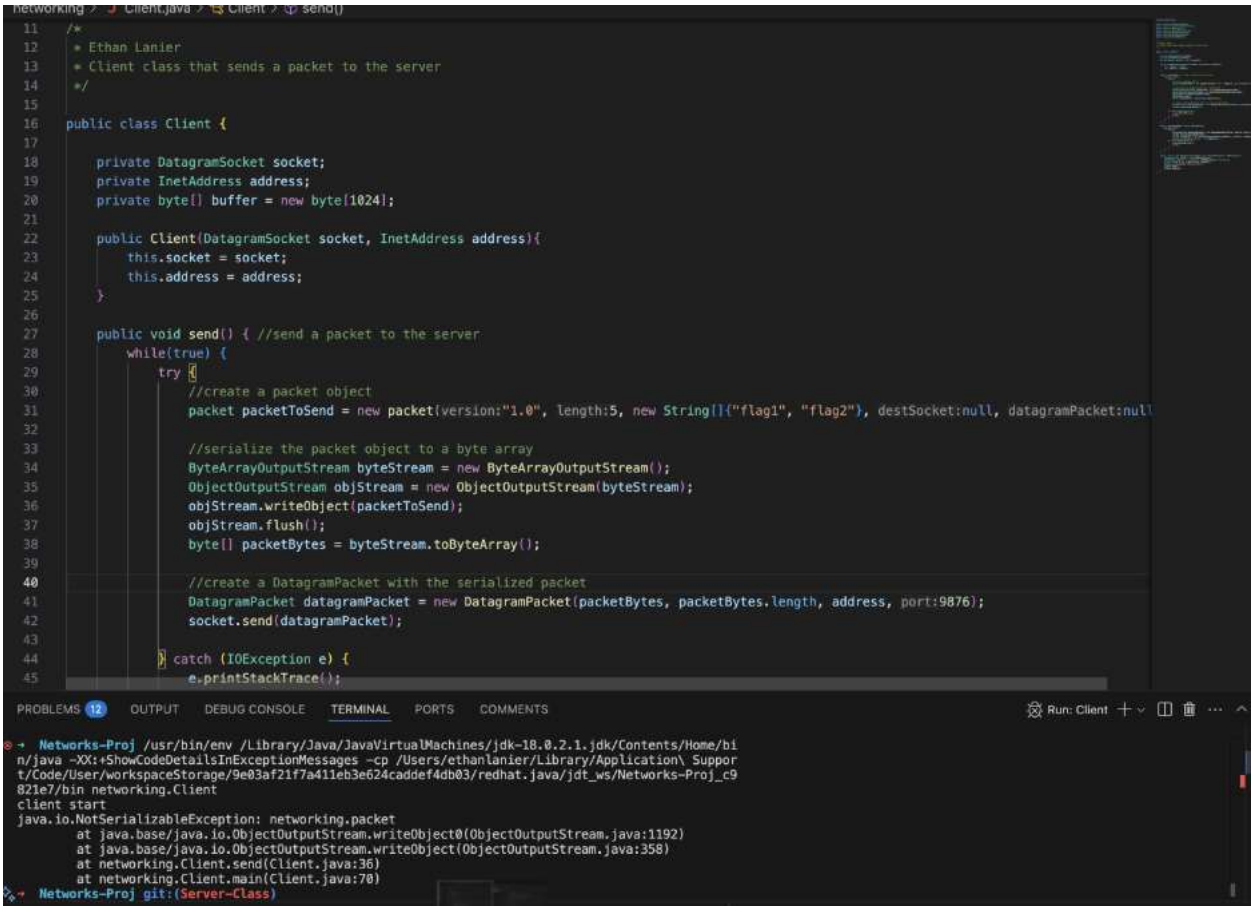
Case 4: Directory has both files and subdirectories

```
[File: blah.txt, File: superman.html, Directory: whatevs]
```

Test run from our server class



Test run from the client class



```
Networking > Client.java > Client > send()
/*
 * Ethan Lanier
 * Client class that sends a packet to the server
 */

public class Client {

    private DatagramSocket socket;
    private InetAddress address;
    private byte[] buffer = new byte[1024];

    public Client(DatagramSocket socket, InetAddress address){
        this.socket = socket;
        this.address = address;
    }

    public void send() { //send a packet to the server
        while(true) {
            try {
                //create a packet object
                packet packetToSend = new packet(version:"1.0", length:5, new String[]{"flag1", "flag2"}, destSocket:null, datagramPacket:null

                //serialize the packet object to a byte array
                ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
                ObjectOutputStream objStream = new ObjectOutputStream(byteStream);
                objStream.writeObject(packetToSend);
                objStream.flush();
                byte[] packetBytes = byteStream.toByteArray();

                //create a DatagramPacket with the serialized packet
                DatagramPacket datagramPacket = new DatagramPacket(packetBytes, packetBytes.length, address, port:9876);
                socket.send(datagramPacket);

            } catch (IOException e) {
                e.printStackTrace();
```

```
PROBLEMS 12  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS                Run: Client

→ Networks-Proj /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bi
n/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/ethanlanier/Library/Application\ Suppor
t/Code/User/workspaceStorage/9e03af21f7a411eb3e624caddef4db03/redhat.java/jdt_ws/Networks-Proj_c9
821e7/bin networking.Client
client start
java.io.NotSerializableException: networking.packet
        at java.base/java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1192)
        at java.base/java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:358)
        at networking.Client.send(Client.java:36)
        at networking.Client.main(Client.java:70)
→ Networks-Proj git:(Server-Class)
```

Instructions

To run the implementation, follow the instructions listed below.

1. Download the zip file
2. Extract the zip file
3. Replace the IPs and port numbers present in the config file of the node with the IPs and ports of the other nodes
4. Run the client app file on nodes you wish to be clients, and the server app file on the node you wish to be the server
5. If running P2P, run the client file on each node instead
6. Use the console to view the status of the network once all the nodes are running