Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU33031 Computer Networks

Assignment #1: Protocols

Student name: YingZheng Pan

Student number: 19336862

Contents

1 Introduction **2**

2 Theory of Topic **2**

2.1 First Component: Client 4

2.2 Second Component: Server 4

2.3 Third Component: Workers 4

2.4 Communication and Packet Description 5

3 Implementation **6**

3.1 Component 1: Client 6

3.2 Component 2: Server 8

3.3 Component 3: Worker 10

3.4 User Interaction 12

3.5 Packet Encoding 13

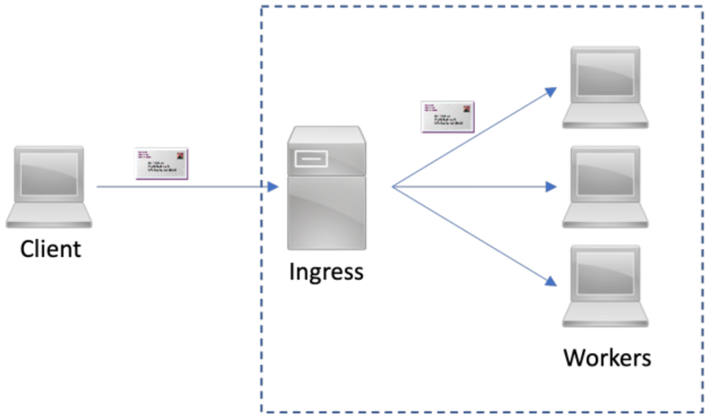
4 Summary **15**

5 Reflection **15**

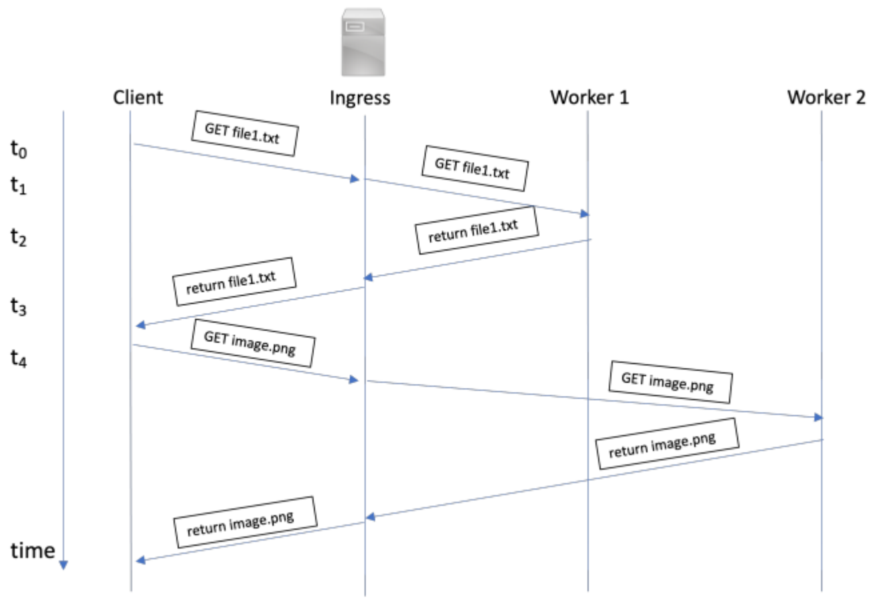
1 Introduction

The focus of this assignment is to learn about protocol development and the information saved in header files to support protocol functionality. For this assignment protocol, the aim of the protocol is to provide a mechanism to retrieve files from a service based on UDP datagrams. The header encoding of the protocol should be implemented in binary format. The protocol involves several actors: One or more clients, an ingress node (which can also call Server), and one or more workers. The client sends a file request to the ingress node and receives a reply from the node. The ingress node processes the requests, forwards them to a worker associated with it, and forwards the reply to the client that sent the reply. The header information contained in the package must support the identification of the requested action, the transfer of the file, which may consist of multiple packages, and the server's management of the worker.

2 Theory of Topic



The figure clearly show that a client sends a request to a server, called ingress here, which distributes requests to available workers.
The basic functionality that the file request protocol has to provide is to support requests for files from a server which then distributes the requests to workers. The description of the above scenario shows that the protocol may need to provide many additional functionalities, such as various file sizes, file contents etc. In my protocol functionality, my protocol provides various file sizes and file contents. I have several workers to deal with different types of files, such as txt and png. The following flow diagrams, which is the basic visualizations of network traffic between components. They show the sequence of messages exchanged between components with the start of their transmission at the sender and their arrival at the receiver.

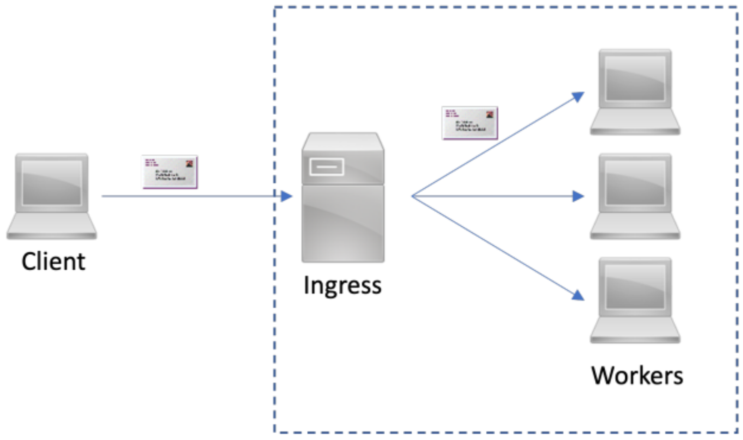


A client will request a file from the server. The server will select a worker and forward the request to the worker. Server will distinguish between different types of files and transfer the corresponding files to workers who specialize in processing these types of files. The worker will retrieve the file and send it to the server which then will forward it to the client. The client will then request another file from the server and the server will pick a different worker to return this specific file. During the transmission process, the protocol can also provide various file sizes and file contents.
Network traffic is the amount of data moving across a computer network at any given time. Network traffic, also called data traffic, is broken down into data packets and sent over a network before being reassembled by the receiving device or computer.

Understanding of UDP:

For UDP (User Datagram Protocol), which provides bidirectional message transport between the server and one or more clients. UDP is not connection-based, and each packet transmission is a separate event. Provides fast and lightweight data transfer for local packet broadcast and remote multicast. In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. The datagram on arrival contains the sender's address, which the server uses to send the data to the correct client. It is mainly used to establish low latency and fault tolerant connections between applications on the internet. UDP speeds up transmission by enabling data transfer before the recipient offers the protocol.

2.1 First Component: Client



Firstly, I used Docker, which can provide a mechanism to create a subnetwork on a local machine. I create a small, bridged network called csnet and then create a container image, named csjavaimg, based on a Dockerfile. The first component container I created is Client. As the figure shows above, a client sends a request to a server (Ingress). The server will select a worker and forward the request to the worker. The worker will retrieve the file and send it to the server which then will forward it to the client. The Client will receive a packet that is processed by the worker and server, which provides the contents of the corresponding file. The Client class provides simple methods for sending and receiving connectionless UDP datagrams in blocking synchronous mode. Because UDP is a connectionless transport protocol, so there is no need to establish a remote host connection prior to sending and receiving data.

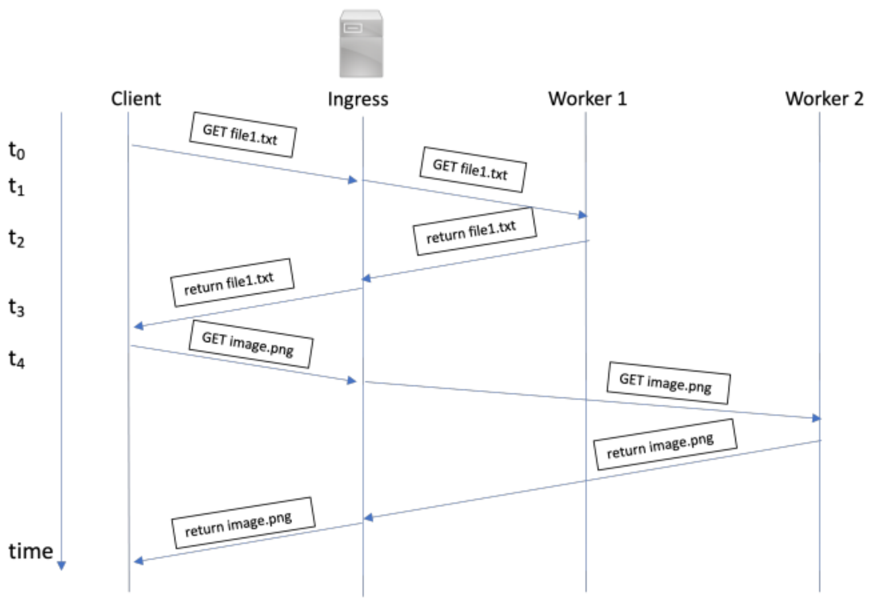
2.2 Second Component: Server

Secondly, I created a second container called Server in Docker. When the server receives the requests sent by Client, the server node processes requests, forwards them to one of the workers that are associated with it. In the server part, the server can make judgments and send the corresponding files to the corresponding worker for processing by distinguishing different types of files. And then the worker will retrieve the file and send it to the server. After that Server forwards replies to client that have send them.

2.3 Third Component: Workers

My third component are workers, A client will request a file from the server. The server will select a worker and forward the request to the worker. In the worker, different workers process different files, and the corresponding worker only processes files of its own category. When the worker retrieves the file, it will send back to the server, which then will forward it to the client.

2.4 Communication and Packet Description



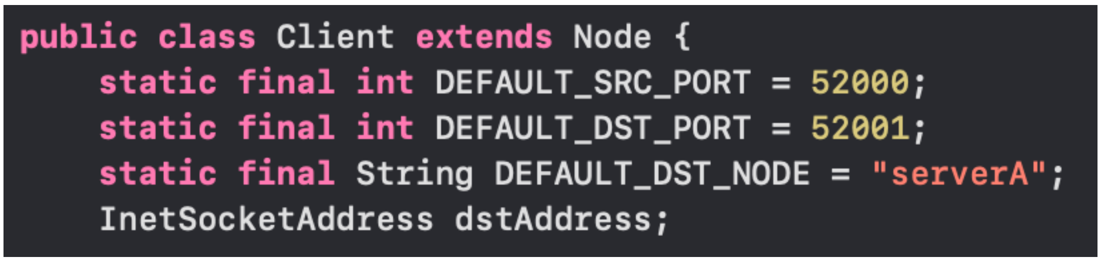
As the figure shows above, the basic and essential part of transmission data packet is a client will request a file from the server. The server will select a worker and forward the request to the worker. Server will distinguish between different types of files and transfer the corresponding files to workers who specialize in processing these types of files. The worker will retrieve the file and send it to the server which then will forward it to the client. The client will then request another file from the server and the server will pick a different worker to return this specific file. During the transmission process, the protocol can also provide various file sizes and file contents. In this assignment, I use UDP Socket and UDP Datagram to solve the problem and send the datagram packets.
My protocol and send packets process are all base on UDP Socket and UDP Datagram. Datagram packets are used to implement connectionless packet delivery services. Each message is routed from one machine to another, based solely on the information contained in the packet. Multiple packets sent from one machine to another may be routed differently and may arrive in any order. A datagram socket is a sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is addressed and routed separately. Multiple packets sent from one machine to another may be routed differently and may arrive in any order. UDP socket routines use the User Datagram Protocol (UDP) for simple IP communication. The User Datagram Protocol runs on top of the Internet Protocol (IP) and is developed for applications that do not require reliability, acknowledgment, or flow control features at the transport layer. UDP wraps datagrams with a UDP header, which contains totaling eight bytes. The fields in a UDP header are contains Source port - The port of the device sending the data. Destination port - The port of the device receiving the data. UDP port numbers can be between 0 and 65,535.

UDP header is an 8-bytes fixed and simple header. The first 8 Bytes contains all necessary header information and the remaining part consist of data. UDP port number fields are each 16 bits long, therefore the range for port numbers defined from 0 to 65535. port number 0 is reserved.

3 Implementation

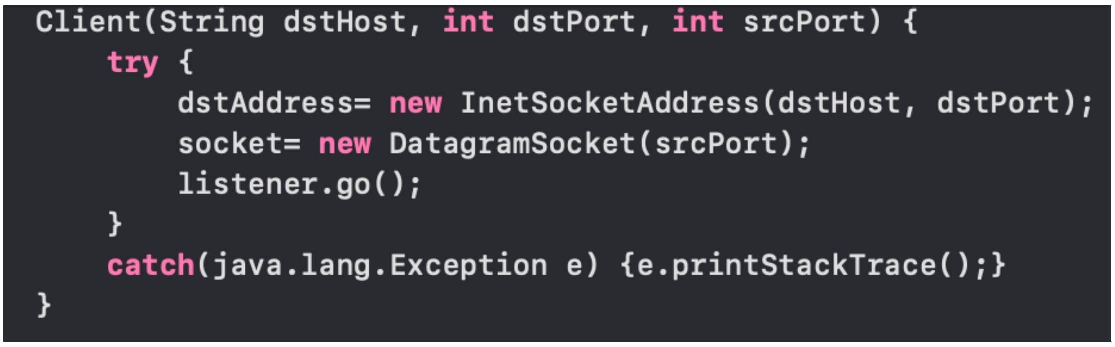
3.1 Component 1: Client

Firstly, in my Client.java, I create a host name called clientA and a client port number 52000. I also create a destination port 52001, which is server port, and the host name of server node is serverA. As the following figure shows.



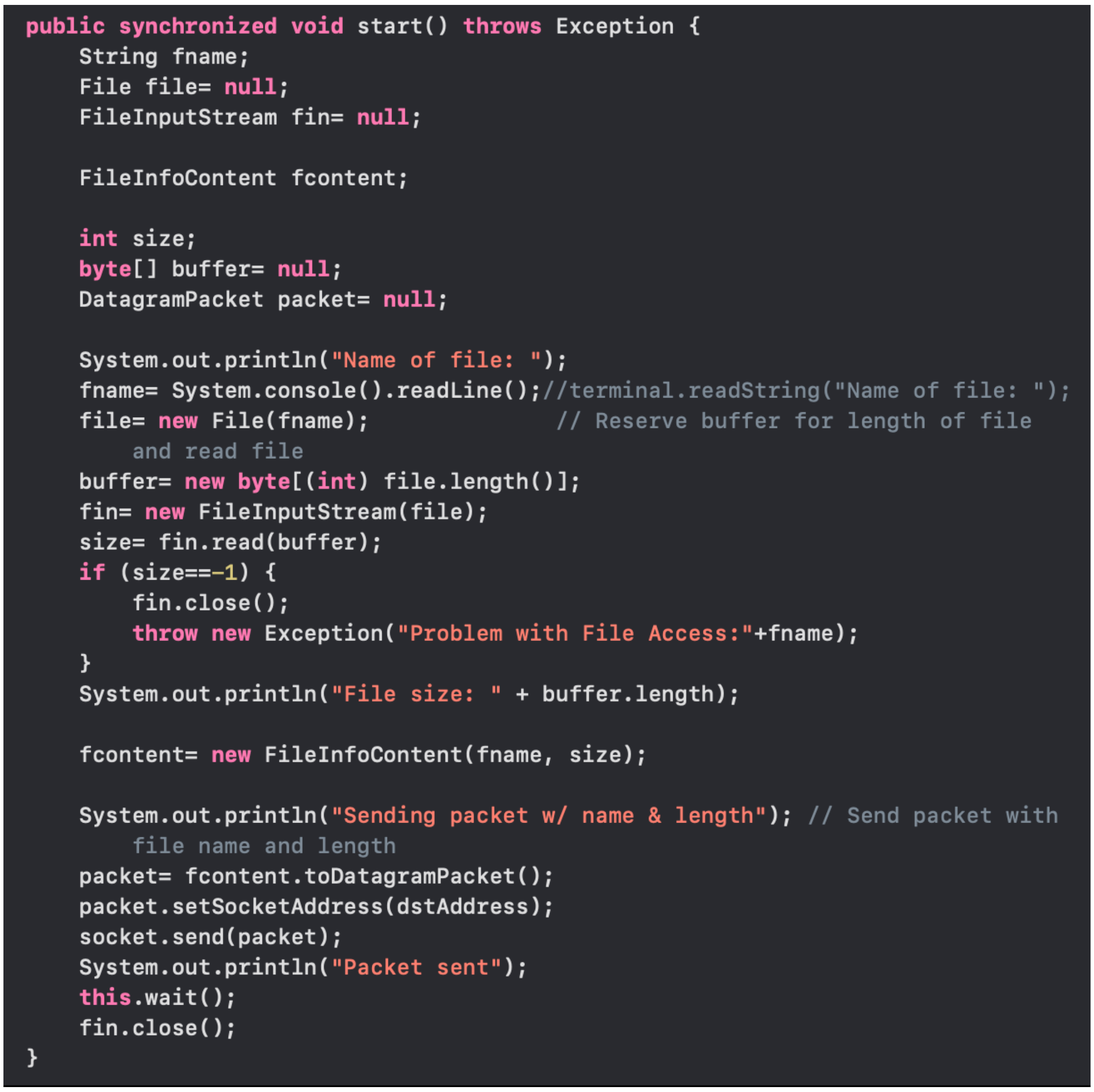
```
public class Client extends Node {
    static final int DEFAULT_SRC_PORT = 52000;
    static final int DEFAULT_DST_PORT = 52001;
    static final String DEFAULT_DST_NODE = "serverA";
    InetSocketAddress dstAddress;
```

Because for the client part, we need to let a client sends a request to a server (Ingress). The server will select a worker and forward the request to the worker. The worker will retrieve the file and send it to the server which then will forward it to the client. The Client will receive a packet that is processed by the worker and server, which provides the contents of the corresponding file. In my Client.java, the client will ask user to input the name of file(fname) they want to find. I used System.console().readLine() to get the file name that user want to find. And then reserve buffer for length of file and read file. And then I store the file content into a byte array. I also have a FileInfoContent.java, it will get the file content(fcontent), such as file name and file size that user choose the file for. Next, we need to make a file content into the packet and then sent it to the server, I attempts to create udp socket at given port and create an InetSocketAddress for the destinations. As the following figure shows.



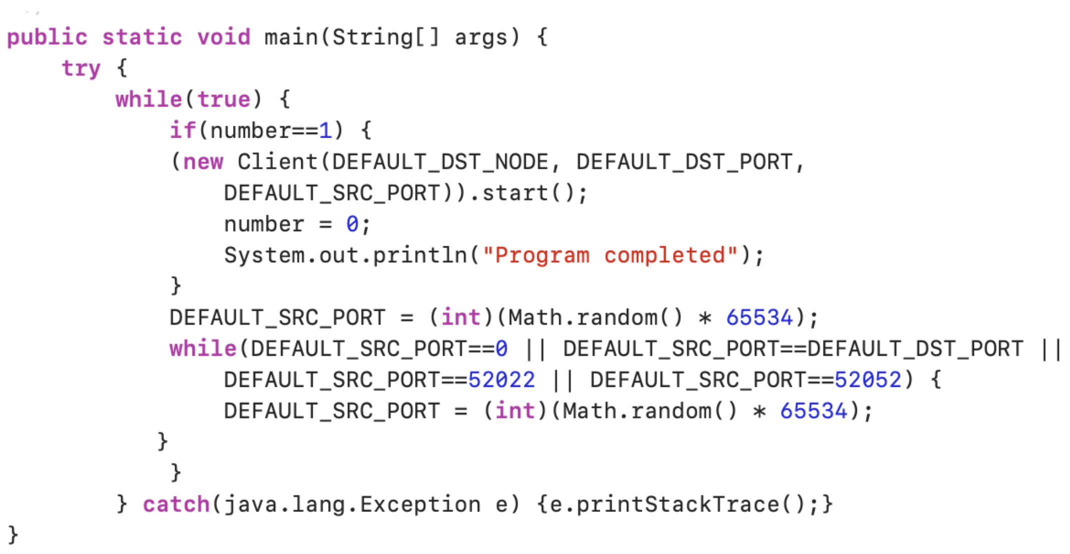
```
Client(String dstHost, int dstPort, int srcPort) {
    try {
        dstAddress= new InetSocketAddress(dstHost, dstPort);
        socket= new DatagramSocket(srcPort);
        listener.go();
    }
    catch(java.lang.Exception e) {e.printStackTrace();}
}
```

Furthermore, I set the packet socketAddress for the destination address, my destination address is the server. After I create the packet and set the destination address successfully, and then I use socket.sent() to send the file content packet to the server node. All the sending processes are shown in the following figure, the code in the figure is a demonstrating sender packet method.



```java
public synchronized void start() throws Exception {
    String fname;
    File file= null;
    FileInputStream fin= null;

    FileInfoContent fcontent;

    int size;
    byte[] buffer= null;
    DatagramPacket packet= null;

    System.out.println("Name of file: ");
    fname= System.console().readLine();//terminal.readString("Name of file: ");
    file= new File(fname);              // Reserve buffer for length of file
        and read file
    buffer= new byte[(int) file.length()];
    fin= new FileInputStream(file);
    size= fin.read(buffer);
    if (size==-1) {
        fin.close();
        throw new Exception("Problem with File Access:"+fname);
    }
    System.out.println("File size: " + buffer.length);

    fcontent= new FileInfoContent(fname, size);

    System.out.println("Sending packet w/ name & length"); // Send packet with
        file name and length
    packet= fcontent.toDatagramPacket();
    packet.setSocketAddress(dstAddress);
    socket.send(packet);
    System.out.println("Packet sent");
    this.wait();
    fin.close();
}
```

As explained above, this is the basic main building block of my Client.java file. For the Client main function, as the figure below shown.

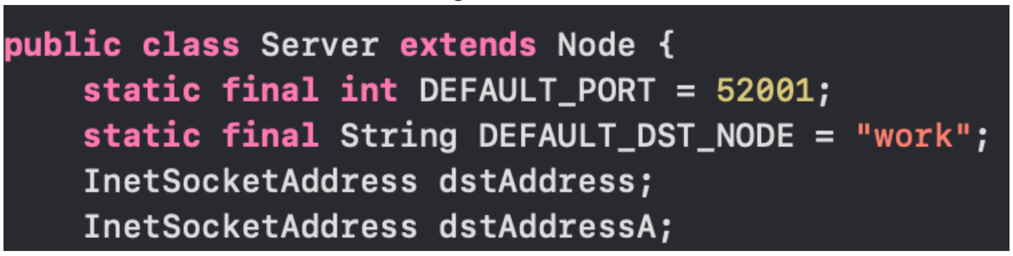


```java
public static void main(String[] args) {
    try {
        while(true) {
            if(number==1) {
            (new Client(DEFAULT_DST_NODE, DEFAULT_DST_PORT,
                DEFAULT_SRC_PORT)).start();
                number = 0;
                System.out.println("Program completed");
            }
            DEFAULT_SRC_PORT = (int)(Math.random() * 65534);
            while(DEFAULT_SRC_PORT==0 || DEFAULT_SRC_PORT==DEFAULT_DST_PORT ||
                DEFAULT_SRC_PORT==52022 || DEFAULT_SRC_PORT==52052) {
                DEFAULT_SRC_PORT = (int)(Math.random() * 65534);
           }
            }
        } catch(java.lang.Exception e) {e.printStackTrace();}
}
```

This is the main function of Client. I have an outer while loop here, which is use for create a new Client each time when I looping the project. The while loop is use to letting user finding and getting multiple files and file contents so that the program will keep asking user Name of file: . What's more, we use Math.random() function here because we need to change the source port of client each time. In order to make the source port take a different value in each loop, if the source port is the same throughout the loop, the program will report the error of repeating the address. the inner while loop is use to when the random number of source port randomly equal to server, worker1 or worker2 port number, we need to let it keeps randomly choose another port number within the 65535, which is the largest range of number in UDP port number, until the source port is not equal to any number I mentioned in the while loop conditions, and then we can pick that random number to our new source port number and keep going the program. When a loop ends each time, I also let the Client print program completed message to tell user one loop process are completed.

3.2 Component 2: Server

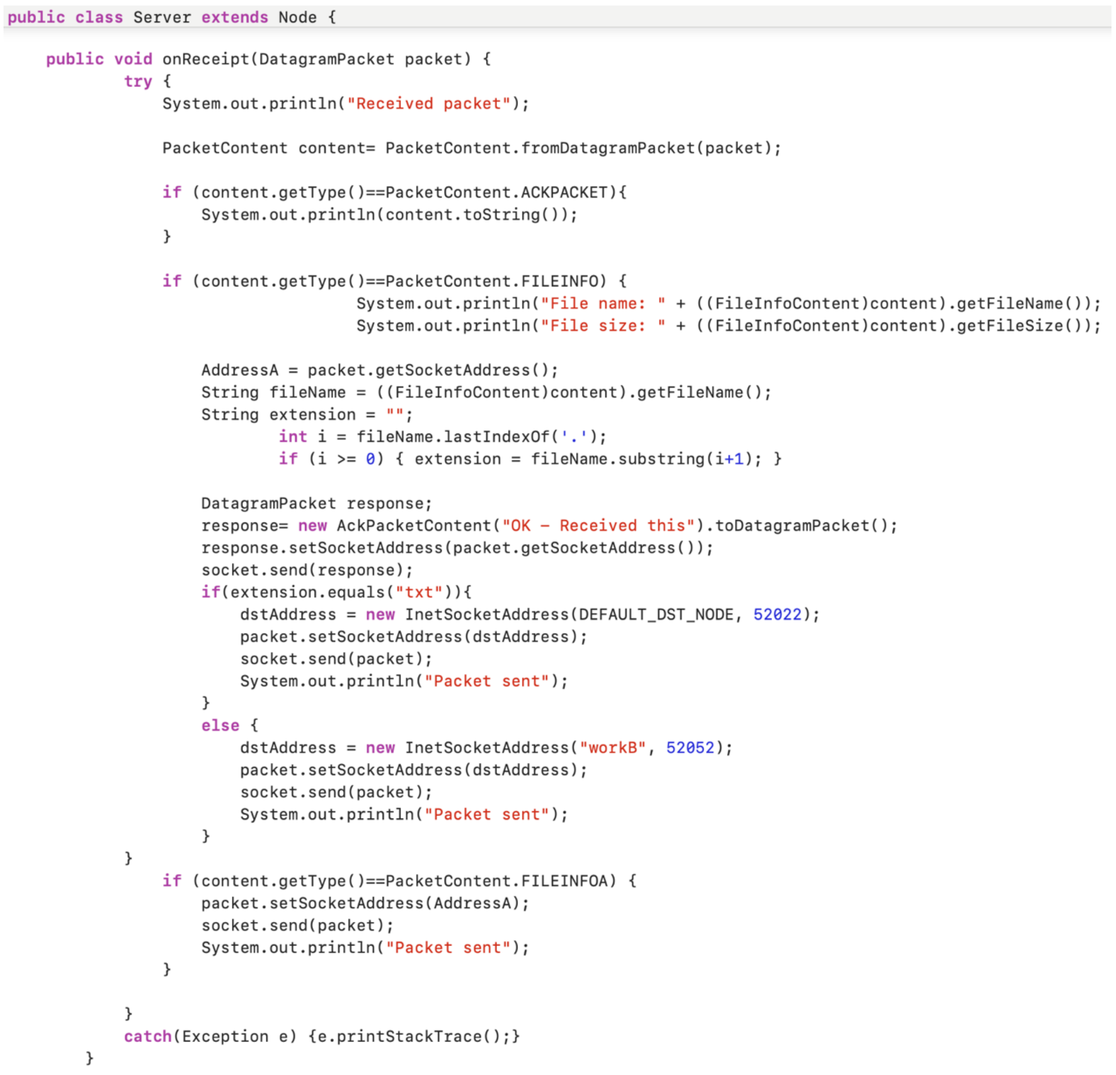
Secondly in my Server.java, I create a host name serverA and a port number of Server which is 52001. As the figure below shown.



```java
public class Server extends Node {
    static final int DEFAULT_PORT = 52001;
    static final String DEFAULT_DST_NODE = "work";
    InetSocketAddress dstAddress;
    InetSocketAddress dstAddressA;
```

For the server part, When the server receives the requests sent by Client, the server node processes requests, forwards them to one of the workers that are associated with it. And then the worker will retrieve the file and send it to the

server. After that Server forwards replies to client that have send them. Therefore the destination port in Server.java is worker, called work.
As the below figure shown, that’s the basic and main process of the work of Server.

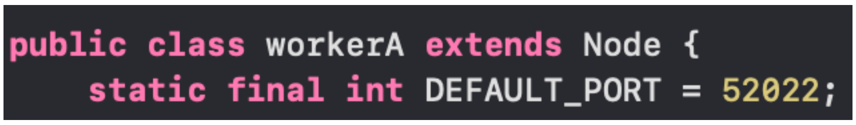


```
public class Server extends Node {

    public void onReceipt(DatagramPacket packet) {
            try {
                System.out.println("Received packet");

                PacketContent content= PacketContent.fromDatagramPacket(packet);

                if (content.getType()==PacketContent.ACKPACKET){
                    System.out.println(content.toString());
                }

                if (content.getType()==PacketContent.FILEINFO) {
                                    System.out.println("File name: " + ((FileInfoContent)content).getFileName());
                                    System.out.println("File size: " + ((FileInfoContent)content).getFileSize());

                    AddressA = packet.getSocketAddress();
                    String fileName = ((FileInfoContent)content).getFileName();
                    String extension = "";
                            int i = fileName.lastIndexOf('.');
                            if (i >= 0) { extension = fileName.substring(i+1); }

                    DatagramPacket response;
                    response= new AckPacketContent("OK - Received this").toDatagramPacket();
                    response.setSocketAddress(packet.getSocketAddress());
                    socket.send(response);
                    if(extension.equals("txt")){
                        dstAddress = new InetSocketAddress(DEFAULT_DST_NODE, 52022);
                        packet.setSocketAddress(dstAddress);
                        socket.send(packet);
                        System.out.println("Packet sent");
                    }
                    else {
                        dstAddress = new InetSocketAddress("workB", 52052);
                        packet.setSocketAddress(dstAddress);
                        socket.send(packet);
                        System.out.println("Packet sent");
                    }
            }
                if (content.getType()==PacketContent.FILEINFOA) {
                    packet.setSocketAddress(AddressA);
                    socket.send(packet);
                    System.out.println("Packet sent");
                }

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

The important part of my Server is I use String.lastIndexOf(‘.’) and get the index of ‘.’ So that I can extract the file type and then make a comparison by using if statement to send packages to workers that specialize in processing the corresponding files. In my protocol program, my workerA solve txt files and my workerB solve png file or any other type of files. In the server, I also have a section where the server transmits to the client. This is because when the worker finishes processing the file of the corresponding file type, the worker will transmit it back to the server, and then the server needs to send the new packet, which included file contents in it, that has been processed back to the client. At the beginning of this part of code, I also have an if statement, which is content.getType() == PacketContent.ACKPACKET. This prints out the message from workers that acknowledge receipt, when the worker receive the packet from the Server, the corresponding worker which receive the packet will send back a receive packet message to Server and the server will print out the confirmation message. That’s the basic and main process of my Server part.

3.3 Component 3: Worker

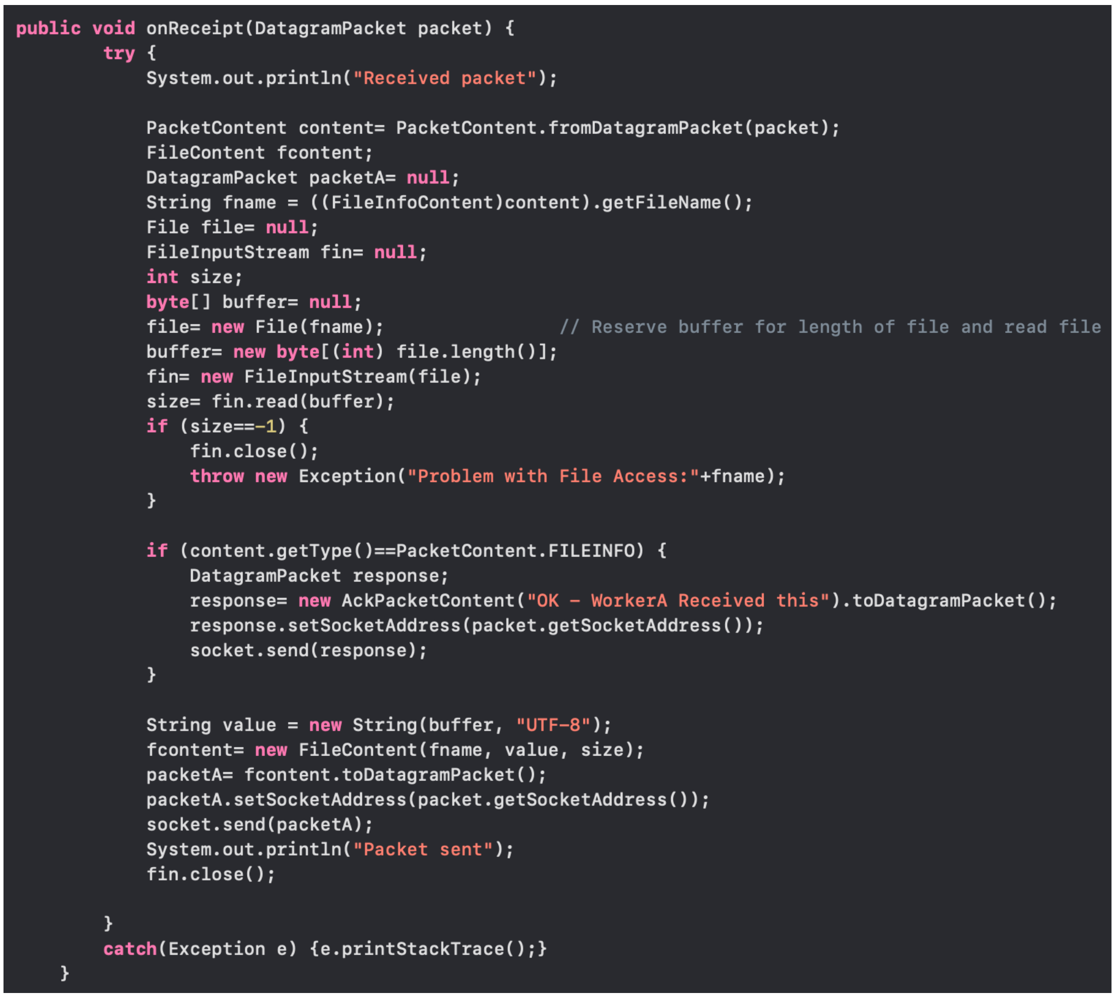
In this assignment, I have two workers in my program, one is workerA and another one is workerB. As the following figure shows this is the port number of my workerA which is 52022.



```
public class workerA extends Node {
    static final int DEFAULT_PORT = 52022;
```

Furthermore, there are two important java files which are FileInfoContent.java and FileContent.java. These two files are important for my two workers. FileInfoContent.java file is used to extract the file size and file name from the file. By the way the FileContent.java is used to extract the specific information and file contents in the file.

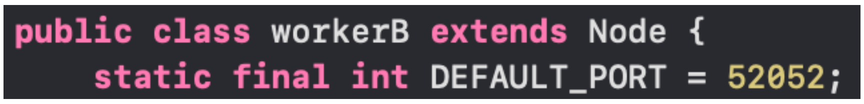
As the following figure show this is the basic and main part of my workerA.



```
public void onReceipt(DatagramPacket packet) {
        try {
            System.out.println("Received packet");

            PacketContent content= PacketContent.fromDatagramPacket(packet);
            FileContent fcontent;
            DatagramPacket packetA= null;
            String fname = ((FileInfoContent)content).getFileName();
            File file= null;
            FileInputStream fin= null;
            int size;
            byte[] buffer= null;
            file= new File(fname);                  // Reserve buffer for length of file and read file
            buffer= new byte[(int) file.length()];
            fin= new FileInputStream(file);
            size= fin.read(buffer);
            if (size==-1) {
                fin.close();
                throw new Exception("Problem with File Access:"+fname);
            }

            if (content.getType()==PacketContent.FILEINFO) {
                DatagramPacket response;
                response= new AckPacketContent("OK - WorkerA Received this").toDatagramPacket();
                response.setSocketAddress(packet.getSocketAddress());
                socket.send(response);
            }

            String value = new String(buffer, "UTF-8");
            fcontent= new FileContent(fname, value, size);
            packetA= fcontent.toDatagramPacket();
            packetA.setSocketAddress(packet.getSocketAddress());
            socket.send(packetA);
            System.out.println("Packet sent");
            fin.close();

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

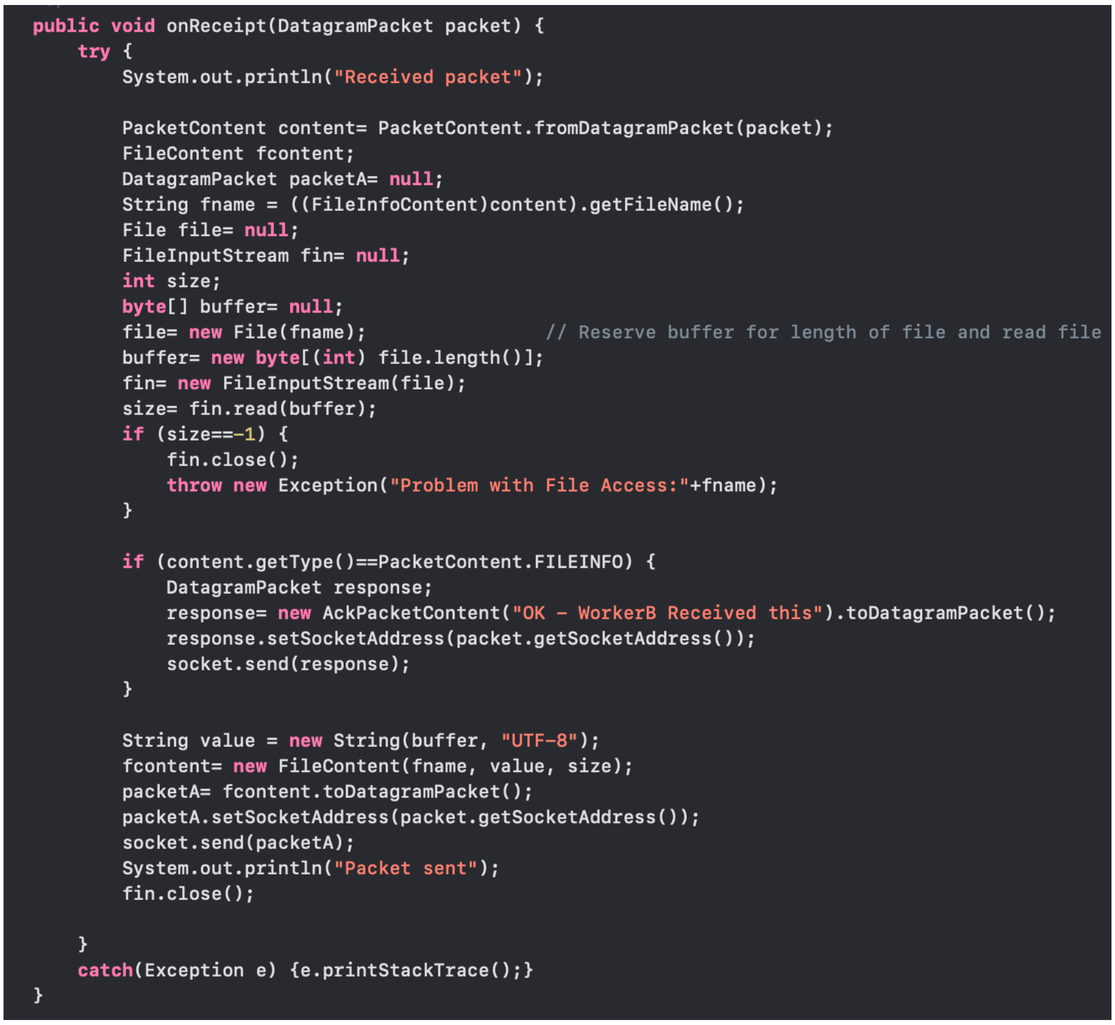
For the first part of my worker code, I use FileInfoContent.java and FileContent.java to extract the file contents of file and then put it into the byte

array[ ]. After this I have an if statement which is used to send an ACKContent message(“OK- WorkerA Received this.”). I put the message into the packet and then send this response message back to Server, and then in the server part, I have an if statement to receive the ACK knowledge message part, it will print out the ACK knowledge content message for confirmation from workers. What’s more, I have string value after this, which is used to change the byte array to string by using “UTF-8”. This line of code will change the byte array info to string value and then I use FileContent to get my file content of file. After this, I set the socket address to Server, then I set the new packet and then put the file content into it. Then I send the packet to the Server. When the worker retrieves the file, it will send back to the server, which then will forward it to the client. That’s the main and essential part of my workerA file.
For my workerB, its principle is similar to workerA. As the figure shows below this is the port number of workerB, which is 52052.



```java
public class workerB extends Node {
    static final int DEFAULT_PORT = 52052;
```

As the figure below shows this is the main part of my workerB.

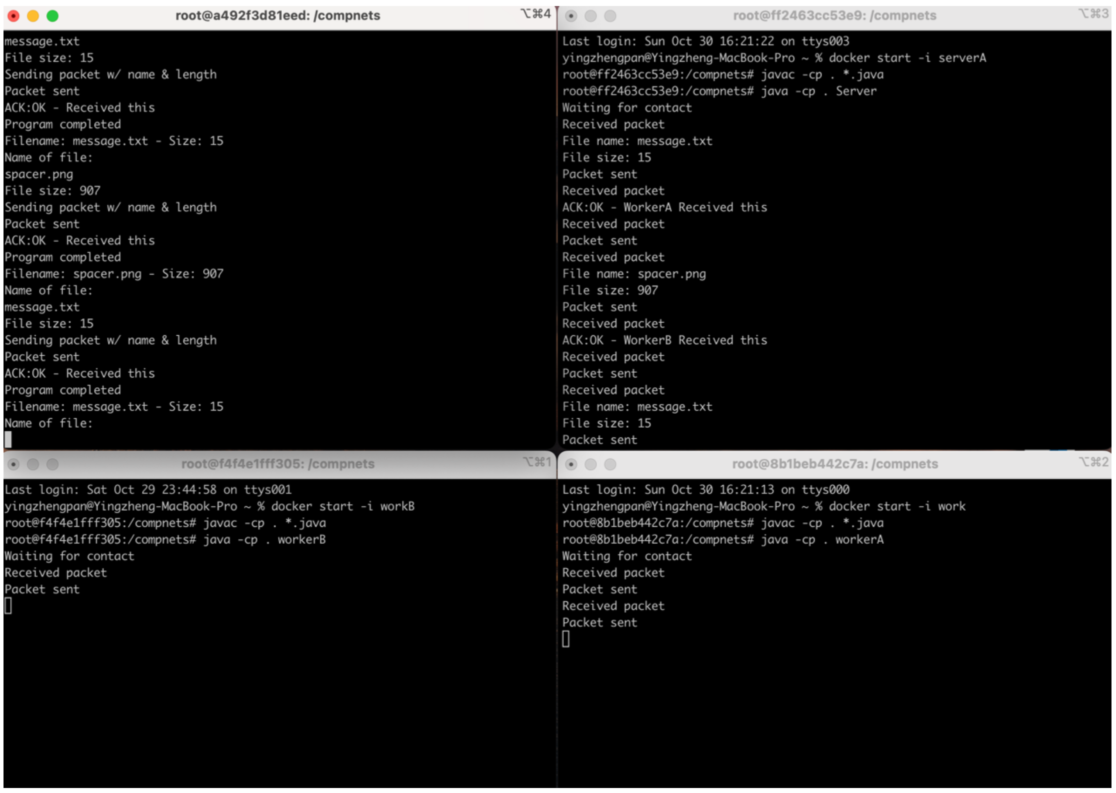


```java
public void onReceipt(DatagramPacket packet) {
    try {
        System.out.println("Received packet");

        PacketContent content= PacketContent.fromDatagramPacket(packet);
        FileContent fcontent;
        DatagramPacket packetA= null;
        String fname = ((FileInfoContent)content).getFileName();
        File file= null;
        FileInputStream fin= null;
        int size;
        byte[] buffer= null;
        file= new File(fname);                // Reserve buffer for length of file and read file
        buffer= new byte[(int) file.length()];
        fin= new FileInputStream(file);
        size= fin.read(buffer);
        if (size==-1) {
            fin.close();
            throw new Exception("Problem with File Access:"+fname);
        }

        if (content.getType()==PacketContent.FILEINFO) {
            DatagramPacket response;
            response= new AckPacketContent("OK - WorkerB Received this").toDatagramPacket();
            response.setSocketAddress(packet.getSocketAddress());
            socket.send(response);
        }

        String value = new String(buffer, "UTF-8");
        fcontent= new FileContent(fname, value, size);
        packetA= fcontent.toDatagramPacket();
        packetA.setSocketAddress(packet.getSocketAddress());
        socket.send(packetA);
        System.out.println("Packet sent");
        fin.close();

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

Similarly, the first part is getting the file content from FileContent and

FileInfoContent of file and then put it into the byte array[ ]. After this, I have if statement to send the ACK knowledge content response message to server for confirmation of receive. What's more, I use "UTF-8" to change the file content byte array to string. And then when I get the file content as a String, I set the Datagram packet and then put the file content into it. And then send the new file content packet to Server. That's my main part of my workerB java file.
The main difference between workerA and workerB are workerA solve the txt file and workerB solve png file or any other type of file.

3.4 User Interaction


root@a492f3d81eed: /compnets

```
message.txt
File size: 15
Sending packet w/ name & length
Packet sent
ACK:OK - Received this
Program completed
Filename: message.txt - Size: 15
Name of file:
spacer.png
File size: 907
Sending packet w/ name & length
Packet sent
ACK:OK - Received this
Program completed
Filename: spacer.png - Size: 907
Name of file:
message.txt
File size: 15
Sending packet w/ name & length
Packet sent
ACK:OK - Received this
Program completed
Filename: message.txt - Size: 15
Name of file:
```

root@ff2463cc53e9: /compnets

```
Last login: Sun Oct 30 16:21:22 on ttys003
yingzhengpan@Yingzheng-MacBook-Pro ~ % docker start -i serverA
root@ff2463cc53e9:/compnets# javac -cp . *.java
root@ff2463cc53e9:/compnets# java -cp . Server
Waiting for contact
Received packet
File name: message.txt
File size: 15
Packet sent
Received packet
ACK:OK - WorkerA Received this
Received packet
Packet sent
Received packet
File name: spacer.png
File size: 907
Packet sent
Received packet
ACK:OK - WorkerB Received this
Received packet
Packet sent
Received packet
File name: message.txt
File size: 15
Packet sent
```

root@f4f4e1fff305: /compnets

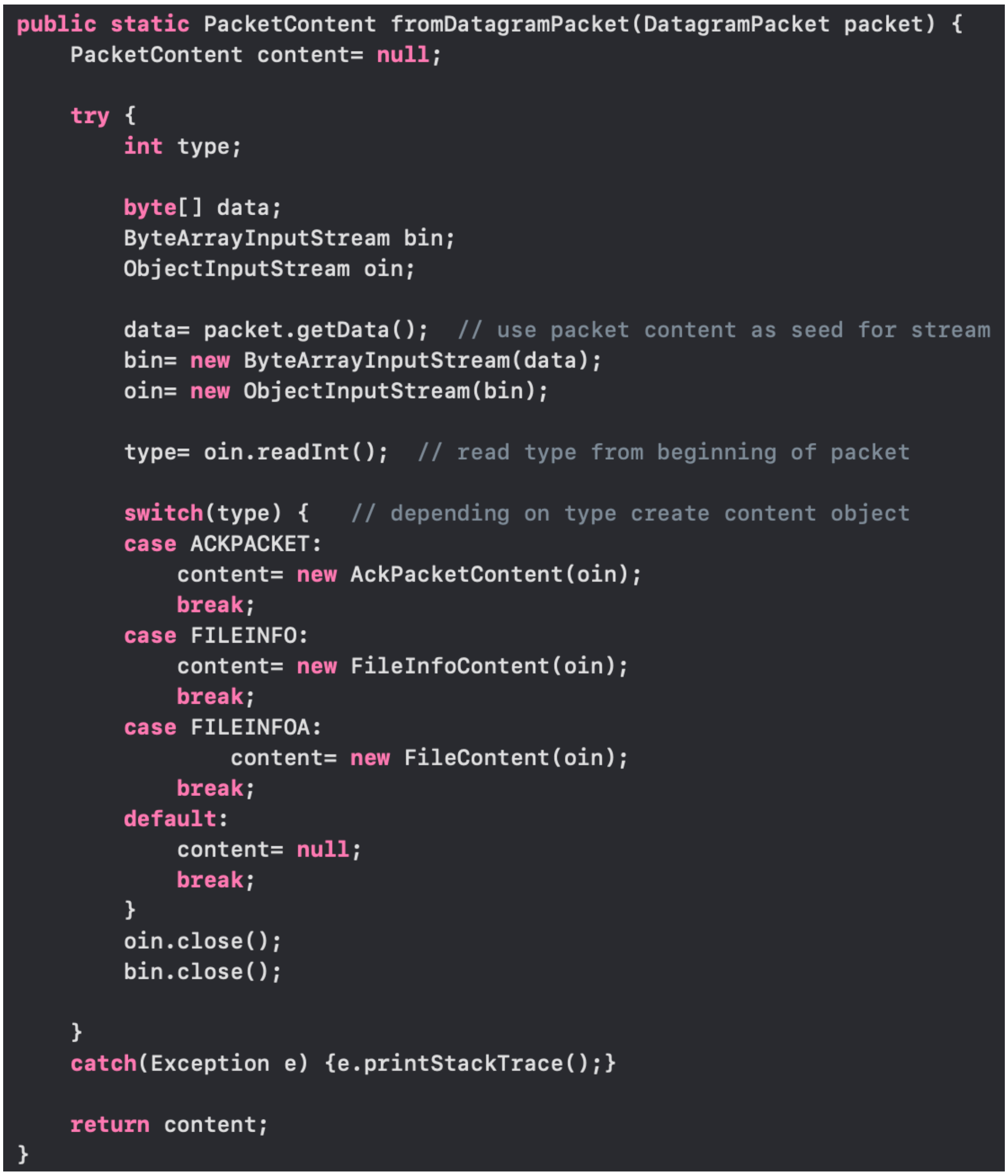
```
Last login: Sat Oct 29 23:44:58 on ttys001
yingzhengpan@Yingzheng-MacBook-Pro ~ % docker start -i workB
root@f4f4e1fff305:/compnets# javac -cp . *.java
root@f4f4e1fff305:/compnets# java -cp . workerB
Waiting for contact
Received packet
Packet sent
```

root@8b1beb442c7a: /compnets

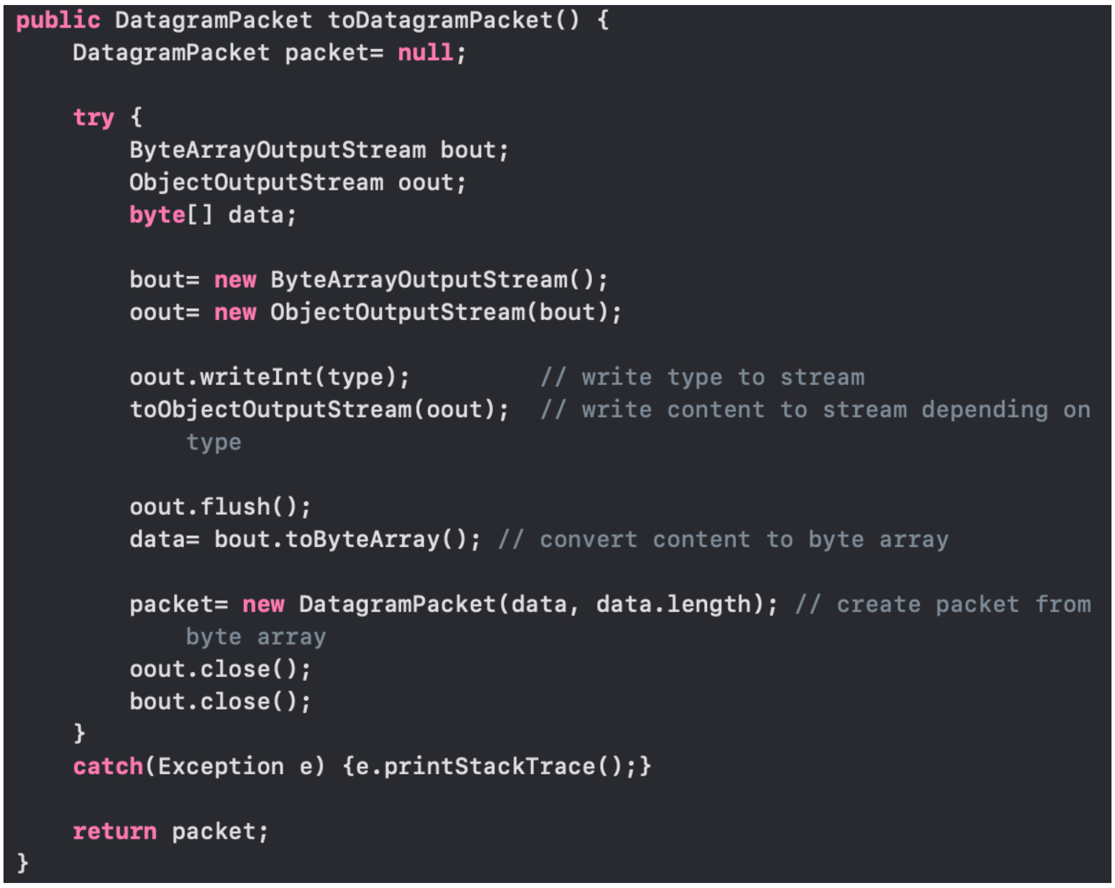
```
Last login: Sun Oct 30 16:21:13 on ttys000
yingzhengpan@Yingzheng-MacBook-Pro ~ % docker start -i work
root@8b1beb442c7a:/compnets# javac -cp . *.java
root@8b1beb442c7a:/compnets# java -cp . workerA
Waiting for contact
Received packet
Packet sent
Received packet
Packet sent
```

As the figure above shows, in my protocol program, firstly the Client will ask user for Name of file: , they want to find for. When the user input the file name for example message.txt or spacer.png. The client sends a request to server. The server will select a worker and forward the request to the worker. When the server receives the requests sent by Client, the server node processes requests, forwards them to one of the workers that are associated with it. In the server part, the server can make judgments and send the corresponding files to the corresponding worker for processing by distinguishing different types of files. For example workerA solve the message.txt file and workerB solve the spacer.png or any other type of files. When the worker receives the packet from server, the worker which receive the corresponding file, will send a ACK knowledge content message to server for confirmation of receive files. In worker part, the worker will process the corresponding file, such as extracting the file name, file size and file content. Then put the extracted things into a new packet, and finally send the new packet back to the Server. And then the worker will retrieve the file and send it to the server. After that Server forwards replies to client that have send them. Finally, the Client will receive a packet that is processed by the worker and server, which provides the contents of the corresponding file.

3.5 Packet Encoding



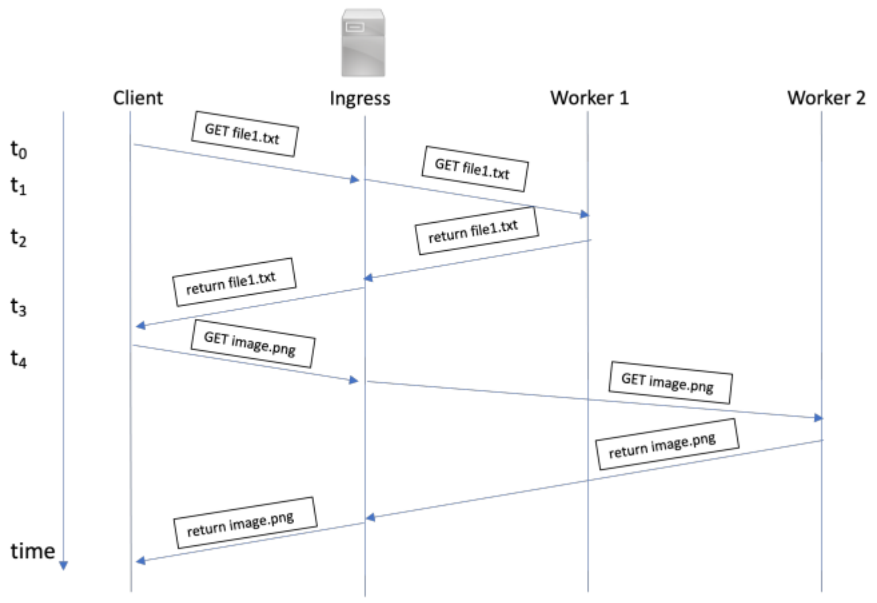
```java
public static PacketContent fromDatagramPacket(DatagramPacket packet) {
    PacketContent content= null;

    try {
        int type;

        byte[] data;
        ByteArrayInputStream bin;
        ObjectInputStream oin;

        data= packet.getData();  // use packet content as seed for stream
        bin= new ByteArrayInputStream(data);
        oin= new ObjectInputStream(bin);

        type= oin.readInt();  // read type from beginning of packet

        switch(type) {   // depending on type create content object
        case ACKPACKET:
            content= new AckPacketContent(oin);
            break;
        case FILEINFO:
            content= new FileInfoContent(oin);
            break;
        case FILEINFOA:
                content= new FileContent(oin);
            break;
        default:
            content= null;
            break;
        }
        oin.close();
        bin.close();

    }
    catch(Exception e) {e.printStackTrace();}

    return content;
}
```



```java
public DatagramPacket toDatagramPacket() {
    DatagramPacket packet= null;

    try {
        ByteArrayOutputStream bout;
        ObjectOutputStream oout;
        byte[] data;

        bout= new ByteArrayOutputStream();
        oout= new ObjectOutputStream(bout);

        oout.writeInt(type);         // write type to stream
        toObjectOutputStream(oout);  // write content to stream depending on
            type

        oout.flush();
        data= bout.toByteArray(); // convert content to byte array

        packet= new DatagramPacket(data, data.length); // create packet from
            byte array
        oout.close();
        bout.close();
    }
    catch(Exception e) {e.printStackTrace();}

    return packet;
}
```

In PacketContent.java, I use this java file to put the file content into the packet and then make it to a Datagram packet. In my program, in client, server and worker java file, I have onReceipt function, which is used to put the file content into the Datagram packet. As the figure shows, those code are used to convert content to byte array and then put it into the Datagram packet and then get the new content packet. In my program I use udp Datagram packet as my packet encoding.

4 Summary



For my protocol program, as the figure shows that's the main logic and approach for the project. the basic and essential part of transmission data packet is a client will request a file from the server. The client will ask user to input the name of file. The server will select a worker and forward the request to the worker. Server will distinguish between different types of files and transfer the corresponding files to workers who specialize in processing these types of files. When the server sends the packets to worker, in worker part, the worker will send the ACK knowledge content message to server for confirmation of receive packet. The worker will retrieve the file and send it to the server which then will forward it to the client. The client will then request another file from the server and the server will pick a different worker to return this specific file. As the figure shows, my workerA will solve txt file and my workerB will solve png file or any other type of files. This is the main and essential part of my protocol program.

5 Reflection

In conclusion, I have learned a lot from this assignment, I know more knowledge about UDP in this module. I expanded my understanding of network and protocol. From this assignment, I made my own protocol program, which made me more familiar with and better understand udp. I like the protocol program very much, which makes me learn a lot about udp. I am looking forward to and will work harder to complete the future assignments.