Can Gözpınar
68965
Comp416

Report

This project aims to design and implement an application level protocol using readily available java socket functionalities. This project makes use of multi-threading to enable multiple client connections to occur concurrently to a given server namely, StratoNet Server. Project is comprised of two modules, first being the authentication and second being querying API's via performing http GET requests. During the authentication phase, client and server exchanges packets which are structured according to the protocol design that was decided on the project guidelines. A successful authentication process ends with client acquiring a token which will be used to validate the client's requests from the server by attaching that token to the packets that are being sent to the server. In the second phase client is prompted to choose and supply the parameters for the API calls that the StratoNet Server would be making for the client. Client can choose between the APOD API and the Insight API. Both cases end up with client receiving the corresponding information from the server.

Code Examination →

Running the main() function of the server starts up by creating the welcoming socket of the server to which client's establishes a connection. Client's main() passes socket related information(such as port) to the ClientSocket class. This class is responsible for establishing the connections for the first time with the command socket(via startSocket() method) and the data socket(via startDataSocket() method). At first main() calls the ClientSockets startSocket() instance which connects to the command socket of the server and declares parameters related to this socket such as inputStream(is), OutputStream(os). One important variable that this method establishes is the localDataSocketPort variable which is used later on by the startDataSocket() method to find the port of the server's data socket to connect to. In my implementation localDataSocketPort is set to 1+localCommandSocketPort. This is due to a choice I made while designing my implementation to figure out that the target client has established a connection with the data socket. The server understands that its(that thread's) client has connected to its data socket after a successful authentication by making sure that the client's port that connected to the dataSocket is 1 more than the port client has used to connect to the command socket.

```
//opens a command socket and initiates StratoNetProtocol
public  void startSocket() throws IOException {

    try{

            s = new Socket ( host, port );
            os = new DataOutputStream (s.getOutputStream());
            is = new DataInputStream(s.getInputStream() );
            stdIn = new BufferedReader (new InputStreamReader ( System.in ) );

            localCommandSocketPort = s.getLocalPort();
            localDataSocketPort = localCommandSocketPort + 1;
            localCommandSocketIp = s.getLocalAddress();
            //hand in socket properties to requestHandler and it will take care of the protocol
            RequestHandler requestHandler = new RequestHandler(is,os,s, stdIn);
            requestHandler.handleRequest();//act according to receivedPackage
        }catch (Exception e){
        e.printStackTrace();
    }

}
```

Later, startSocket() method initializes an instance of the RequestHandler class which functions to listen for the incoming packets from the server and acts according to the StratoNet Protocol to further a healthy communication between the two. RequestHandler class's handleRequest() method is called which listens to command socket's inputStream and parses it according to the header design to parse information from it such as phase, type, size, payload. Then by using a switch statement acts according to the phase value that it has extracted from the incoming packet. Phase value of 0 leads to authentication modules being involved and phase value of 1 which happens after a successful authentication, leads query module to be called.

```
//listen to command socket
try{
    AuthenticationModule authenticationModule = new AuthenticationModule(is,os,s, stdIn);

    System.out.println("Please enter your StratoNet username.");
    authenticationModule.authUname();//authenticates username for the first time

    //parse replies from the server
    byte[] b = new byte[2000];//byte array to incoming socket input message
    while ( (is.read(b)) != 0 ) {

        //convert inputStream to byte[] and parse packet infe
        int phase = Arrays.copyOfRange(b,  i: 0, i1: 1)[0];
        int type = Arrays.copyOfRange(b,  i: 1, i1: 2)[0];
        int size;
        String payload;

        //act accordingly
        switch(phase){
            case 0: //for Authentication
                size  = ByteBuffer.wrap(Arrays.copyOfRange(b,  i: 2, i1: 6)).getInt();
                payload = new String(Arrays.copyOfRange(b,  i: 6, i1: 6 + size));

                authenticationModule.authRequestHandler(type, size, payload);
                break;
            case 1: //for Querying
                size  = ByteBuffer.wrap(Arrays.copyOfRange(b,  i: 2, i1: 6)).getInt();
                int dsSize  = ByteBuffer.wrap(Arrays.copyOfRange(b,  i: 6, i1: 10)).getInt();//size of
                payload = new String(Arrays.copyOfRange(b,  i: 10, i1: 10 + size));

                //attain the data socket
                Socket ds = ClientSocket.getDataSocket();
                //attain the validation token
                String token = authenticationModule.getToken();
                QueryModule queryModule = QueryModule.getInstance(is, os, s, ds, token, stdIn);
                //handles request coming from the server and responds accordingly
                queryModule.queryRequestHandler(type, size, payload, dsSize);
                //TODO: implement query module
                break;
```

At first since the client will have to go through the authentication phase case 0 get executed fist. It parses and passes packet related information that would be of use such as payload to AuthenticatonModule class's instance by calling authRequestHandler() method. This class is specifically made to handle authentication related processes by acting on the type parameter of the packet that is passed to its methods to figure out the next feasible action that conforms to the StratoNet Protocol. authRequestHandler() method is called to handle authenticating via the password processes by sending packets to the server that again conforms to the protocol.

```java
//handles first incoming initialization message
public void authRequestHandler(int type, int size, String payload) throws IOException {
    switch(type){
        case 1://received Auth_Challenge
                //reply with Auth_Request(pwd)

            if(failCount != 0){//if user has sent a wrong pwd before
                System.out.println(payload);
            }
            failCount++;

            //prompt user to enter pwd
            System.out.println("Enter your password:");
            String pwd = "";
            pwd = stdIn.readLine();

            //create TCPPayload
            TCPPayload sendPacket = new TCPPayload( phase: 0, type: 0, pwd.length(), pwd);
            //send packet
            os.write(sendPacket.toStratonetProtocolByteArray());

            break;
        case 2://Auth_fail
            System.out.println(payload);
            terminateSocket();
            break;
        case 3://Auth_Success(token)
            //retrieve the token and save it for queryModule to authenticate its queries later on
            token = payload;

            //connect to DataSocket at port 5555
            ClientSocket.startDataSocket();
            break;
    }
}
```

Let’s take a look at the server side code to get a better picture of the other half of the process that takes places during this authentication phase. On the server side just like the client side’s ClientSocket class that called in the main(), server side has Server class which initiates a server socket via the parameters passed(port) to its Server() constructor. This constructor invokes its listenAndAccept() method in an infinite loop which utilizes multi-threading to allow multiple client’s to be served concurrently by letting them operate on separate threads.

```java
*/
public Server(int port)
{
    try{
        serverSocket = new ServerSocket(port);//command socket
        dataSocket = new ServerSocket(dataSocketPort);

        System.out.println("StratoNet opened up a server socket on " + Inet4Address.getLocalHost());
    }
    catch (IOException e){
        e.printStackTrace();
        System.err.println("Server class.Constructor exception on opening a StratoNet server socket");
    }
    while (true){
        ListenAndAccept();
    }
}
```

```
    private void ListenAndAccept(){
        Socket s;
        try
        {
            s = serverSocket.accept();
            System.out.println("A connection was established with a client on the address of " + s.getRemot
            ServerThread st = new ServerThread(s, dataSocket); //handles multithreading of multiple connect
            st.start();

        }

        catch (Exception e)
        {
            e.printStackTrace();
            System.err.println("Server Class.Connection establishment error inside listen and accept functi
        }
    }

}
```

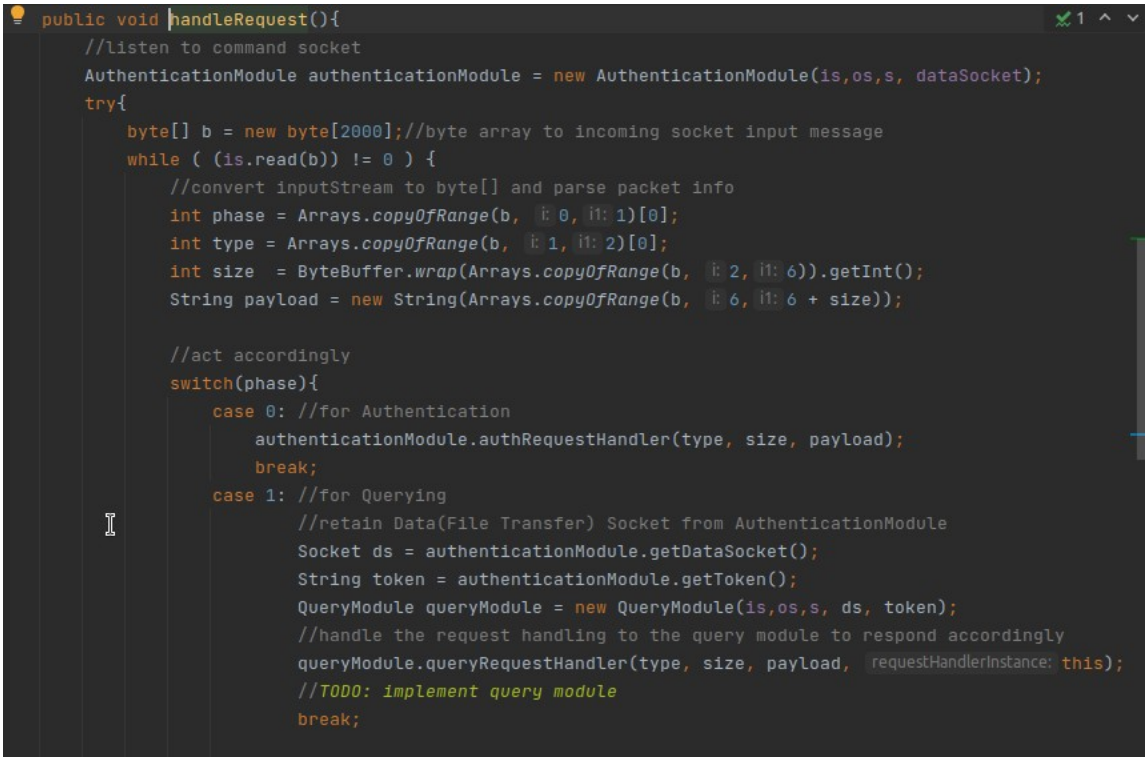
Each newly connected client leads to a new thread being run, namely the ServerThread. These instances of ServerThreads initializes RequestHandler instances and calls its handleRequest() method to listen for and handle incoming packets from the clients and act accordin to the StratoNet Protocol.

```
/**
 * The server thread, communicates with the client according to the StratoNet protocol
 */
public void run(){
    try{
        is = new DataInputStream(s.getInputStream());
        os = new DataOutputStream(s.getOutputStream());

    }
    catch (IOException e){
        System.err.println("Server Thread. Run. IO error in server thread");
    }

    try{
        //Handle Authentication functionality by passing it to Authentication instance
        RequestHandler requestHandler = new RequestHandler(is,os,s, dataSocket);
        requestHandler.handleRequest();
    }catch (NullPointerException e){
```

Just like client side's handleRequest() method, server's handleRequest() parses the information from the arriving packets and delegates information to the corresponding modules(authentication → phase 0 & query → phase 1) by checking the phase value of the arriving packet.



```
public void handleRequest(){
    //listen to command socket
    AuthenticationModule authenticationModule = new AuthenticationModule(is,os,s, dataSocket);
    try{
        byte[] b = new byte[2000];//byte array to incoming socket input message
        while ( (is.read(b)) != 0 ) {
            //convert inputStream to byte[] and parse packet info
            int phase = Arrays.copyOfRange(b,  i: 0, i1: 1)[0];
            int type = Arrays.copyOfRange(b,  i: 1, i1: 2)[0];
            int size  = ByteBuffer.wrap(Arrays.copyOfRange(b,  i: 2, i1: 6)).getInt();
            String payload = new String(Arrays.copyOfRange(b,  i: 6, i1: 6 + size));

            //act accordingly
            switch(phase){
                case 0: //for Authentication
                    authenticationModule.authRequestHandler(type, size, payload);
                    break;
                case 1: //for Querying
                        //retain Data(File Transfer) Socket from AuthenticationModule
                        Socket ds = authenticationModule.getDataSocket();
                        String token = authenticationModule.getToken();
                        QueryModule queryModule = new QueryModule(is,os,s, ds, token);
                        //handle the request handling to the query module to respond accordingly
                        queryModule.queryRequestHandler(type, size, payload,  requestHandlerInstance: this);
                        //TODO: implement query module
                        break;
```

For the authentication phase, it delegates information to authenticationModule by invoking its authRequestHandler() method. This method makes use of logic operators to pin point the client's situation given at that time such as how many password trials are left and sends packets accordingly. This method calls getValidUsers() method to get a HashMap<uname,pwd> which is used to both identify if a username is valid and if so to check the correctness of the password supplied to login. In my implementation valid usernames and passwords reside in apodAPIKey.txt file and has a syntax of "Username:" followed by the user's name and "Password:" followed by the registered password of the user. This method makes use of regex patterns to extract these fields from the mentioned text file.

```
    String text = "";//text of validUsers.txt
    // /read the validUsers.txt
    while (sc.hasNextLine()){
        text += sc.nextLine()+"\n";
    }

    //extract username and passwords using regex
    Pattern unamePattern = Pattern.compile("Username:(.*)\\b");
    Matcher unameMatcher = unamePattern.matcher(text);

    Pattern pwdPattern = Pattern.compile("Password:(.*)\\b");
    Matcher pwdMatcher = pwdPattern.matcher(text);

    //store valid username(key) pwd(value) combination in validUsersMap
    HashMap<String, String> validUsersMap = new HashMap<>();//
    while(unameMatcher.find() & pwdMatcher.find()){
        validUsersMap.put(unameMatcher.group( i: 1),pwdMatcher.group( i: 1)); // key=username, value=passw
    }

    return validUsersMap;
}
```

There is also a generateToken() method which returns a String of length 6 which is created unique to every user and used as a token to validate the user during querying phase.

```
//generates token
public String generateToken(){
    String stringToBeHashed = username + "68"; // hash username and last two digits of my kusisID(6896
    String hashedString = Integer.toString(stringToBeHashed.hashCode());
    //take first 6 digits of hashedString as the token
    String token = hashedString.substring( i: 0, i1: 7);
    return token;
}
```

Having finished this phase with a successful login, client is granted a token which will be appended to the packets it will be sending during the querying phase to prove its validity. It is at that point that client establishes a connection with the server's data socket by utilizing the aforementioned startDataSocket() method.

```
//opens a DataSocket and initiates StratoNetProtocol
public static void startDataSocket() throws IOException {

    try{
        dataSocket = new Socket ( host, dataSocketPort, localCommandSocketIp,  localDataSocketPort);
        System.out.println("Connected to Data Socket at: " + host + " " + dataSocketPort + ".");

        //prompt user to query the server
        Socket ds = ClientSocket.getDataSocket();
        //attain the validation token
        String token = AuthenticationModule.getToken();
        QueryModule queryModule = QueryModule.getInstance(is,os,s, ds, token, stdIn);//initializes the
        //prompt user to make the first query to the server
        queryModule.initiateQueryModule();
```

This method calls QueryModule's(which is a class specialized in handling packet exchanges that take place during the querying phase) initiateQueryModule(). This method prompts the user about the possible querying options(APOD and Insight) and according to the input from the user it initiates the communiation with the server by sending a packet.

```
//prompt user and start querying the StratoNetServer
public void initiateQueryModule() throws IOException {
    //prompt the user about querying syntax
    System.out.println("For querying APOD type \"APOD <YYYY-MM-DD>\"");
    System.out.println("For querying Insight API type \"insight \"");

    boolean validQuery = false;//to check if a valid query request is made

    while (!validQuery){
        //read and parse the user input
        String queryString = "";
        queryString = stdIn.readLine();


        Pattern apodPattern = Pattern.compile("APOD (\\d{4})-(\\d{2})-(\\d{2})$");
        Pattern insigthApiPattern = Pattern.compile("insight");

        Matcher apodMatcher = apodPattern.matcher(queryString);
        Matcher insightApiMatcher = insigthApiPattern.matcher(queryString);
```

```
        if(apodMatcher.find()){//valid APOD request is made
            String year = apodMatcher.group( i: 1);
            String month = apodMatcher.group( i: 2);
            String day = apodMatcher.group( i: 3);
            //make the request to the server
            queryAPOD(year, month, day);
            validQuery = !validQuery;
        }else if(insightApiMatcher.find()){//valid insight API request is made
            queryInsight();
            validQuery = !validQuery;
        }else{//non valid request is made
            System.out.println("Invalid request type. Try again.");
        }
    }
```

This class uses regex to extract the query parameters(if present) from the user input. And if APOD is chosen then, queryAPOD() is invoked with the relevant parameters being passed to it or queryInsight() is called to ask server for the Insight API related tasks. QueryAPOD() sends a packet which is structured very similarly to the server. Again it consists of phase, type, size and payload which are identical except that payload consists of token + query(i.e YYYY-MM-DD). At the server side token and query are separated by assigning the first 6 chars of the string acquired from the payload since token has a fixed size of 6 and always appears at the beginning of the payload in our implementation.

```
public void queryAPOD(String year, String month, String day) throws IOException {
    int phase = 1;
    int type = 0;
    //payload is token+the query param for APOD
    String replyPayload = token + year+"-"+month+"-"+day; //send token
    int replySize = replyPayload.length();

    TCPPayload replyMessage = new TCPPayload(phase, type, replySize, replyPayload);
    os.write(replyMessage.toStratonetProtocolByteArray()); // send the Auth_Success to client

}
```

In the server side as explained earlier, RequestHandler class's method interprets a packet with phase 1 as being related to the query module so it delegates it to the server's QueryModule class. This class has queryRequestHandler() which parses information from the received packet and by inspecting the type attained from the packet it interprets which query related action to take. A general overview of the meanings of these types and such could be found at the end of this document where the information about my protocol design is showcased. Actions for a type of 0 which means that client has made APOD related query to the server is shown below. Just like this case code for other cases can be found in the source code provided with this document. One important thing to note in this code snippet is that both the server's and the client's QueryModule classes makes use of another class named TCPPayload which provides a mold for the structure of the packet protocol that we are using. It converts messages to bytes given its parameters are provided in the constructor. It can achieve both the authentication protocol and the querying protocol designs requirements. It's methods are explained using comments in depth in the code so I am not going to clutter this document by going in any further detail. Also checkSum() method is used to generate hash functions from payload's byte array to cross check any data corruption that could happen during the data transfer. If one were to happen client requests a 'resend' from the server to make up for the loss as explained in the assignment's pdf guide.

```java
//interprets the incoming msg from the client and responds accordingly
public void queryRequestHandler(int type, int size, String payload, RequestHandler requestHandlerInsta
    this.requestHandlerInstance = requestHandlerInstance;
    //parse the load into token and query parameter
    String payloadToken = (String)payload.subSequence( i: 0, token.length());//token send by the client
    String payloadMsg = (String)payload.substring(token.length());
    switch (type){

        case 0://Query params received from client for APOD query
            String queryParam = payloadMsg;
            //check token validit
            if(tokenValidation(payloadToken)){//if user is valid
                String host = "https://api.nasa.gov";
                String urlExtension = "/planetary/apod?api_key="+APODKEY+"&date="+queryParam;
                //make HTTP GET request to apodGetUrl
                String jsonString = "";
                BufferedImage image;
                try {
                    jsonString = HttpGetRequest.makeHttpGetRequest( urlAddr: host+urlExtension);
                    //parse the json Object and attain the image url
                    /*JSONObject json = new JSONObject(jsonString);*/
                    JSONParser parser = new JSONParser();
                    JSONObject json = (JSONObject) parser.parse(jsonString);
                    String imageUrl = (String) json.get("url");
```

```java
//download the image from the image url                                    ⚠ 10  ✓ 9
URL url = new URL(imageUrl);
image = ImageIO.read(url);
//convert downloaded image to String
ByteArrayOutputStream aos = new ByteArrayOutputStream();
ImageIO.write(image,  s: "jpg", aos);
byte[] bytes = aos.toByteArray();

//send command socket
int replyPhase = 1;
int replyType = 1;
String imageHash = String.valueOf(getCRC32Checksum(bytes));
String replyPayload = token + imageHash; //send image as payload
int replySize = replyPayload.length();

TCPPayload replyMessage = new TCPPayload(replyPhase, replyType, replySize, replyPayload, bytes.length);

//take backups of the packets
requestHandlerInstance.setsBackUpFile(replyMessage.toAPIStratonetProtocolByteArray());
requestHandlerInstance.setDsBackUpFile(bytes);

os.write(replyMessage.toAPIStratonetProtocolByteArray()); // send the Auth_Success to client

//send the image through the data Socket
dos.write(bytes); // send the Auth_Success to client
```

```java
//convert object to StratoNet protocol String message(used for phase 0)
public byte[] toStratonetProtocolByteArray(){
    //convert info to bytes
    byte[] phaseByteArray = ByteBuffer.allocate(1).put((byte)phase).array();
    byte[] typeByteArray = ByteBuffer.allocate(1).put((byte)type).array();
    byte[] payloadByteArray = payload.getBytes();
    byte[] sizeByteArray = ByteBuffer.allocate(4).putInt(payloadByteArray.length).array();//byte size

    //concatenate byte arrays to one packet
    byte[] packet = new byte[phaseByteArray.length + typeByteArray.length + sizeByteArray.length + pa
    System.arraycopy(phaseByteArray, i: 0, packet, i1: 0, phaseByteArray.length);
    System.arraycopy(typeByteArray, i: 0, packet, phaseByteArray.length, typeByteArray.length);
    System.arraycopy(sizeByteArray, i: 0, packet, i1: phaseByteArray.length + typeByteArray.length, si
    System.arraycopy(payloadByteArray, i: 0, packet, i1: phaseByteArray.length + typeByteArray.length

    return packet;
}
```

Querying Protocols: