KOÇ UNIVERSITY
COMP 416 - COMPUTER NETWORKS, FALL 2020

Project 1: WeatNet

Baran Berkay Hökelek, 0060673 Berkay Barlas, 0054512
Cennet Tuğçe Turan, 0060605

November 8, 2020

1 Description of the Project

This project is a demonstration of our skills in developing computer network applications, in particular; it includes an application layer protocol design, a client/server protocol design, socket programming and multithreading.

Our goal was to develop an application that would request weather data from OpenWeatherMaps(OWM) and display them to the user. The server application takes a user input from client for the city and the date of the requested weather data.Server communicates with the OWM through its API (Application Programming Interface) and returns the results to client with defined protocol. Added on top of that task, there were other, smaller tasks such as authenticating new users, or using multithreading to respond to multiple users at the same time.

In the end, we built a system in which a user can authenticate by answering some questions; request weather data from a city at a particular time; and get either the results, or an error with explanation.

2 Explanation of the Philosophies

2.1 Post-Authentication Server-Client Protocol

We've built a server and a secure protocol that can handle the requests from multiple users simultaneously. We've indicated all the possible outcomes below.

During Querying phase our protocol includes token section in messages. When server receives a message from client, it checks TOKEN in message with the TOKEN generated for user.

For API_REQUEST message types we defined more specific protocol for handling different cases and creating more robust system. Also, our request protocol is flexible to handle multiple paramaters.

```
public enum MessageType {
    /** Undefined Message type */
    UNDEFINED(-1),
    /** Authentication request */
    AUTH_REQUEST(0),
    /** Authentication challange */
    AUTH_CHALLENGE(1),
    /** Authentication failed */
    AUTH_FAIL(2),
    /** Authentication successfull */
    AUTH_SUCCESS(3),
    /** Data connection request from client */
    DATA_CONNECTION_REQUEST(4),
    /** Data connection accepted by server */
    DATA_CONNECTION_ACCEPTED(5),
    /** Data connection declined by server */
    DATA_CONNECTION_DECLINED(6),
    /** API request by client */
    API_REQUEST(7),
    /** API request successful */
    API_RESPONSE_SUCCESS(8),
    /** A failure during API Response */
    API_RESPONSE_FAIL(9),
    /** Message type for requesting data from server */
    API_REQUEST_DATA(10),
    /** Message type that includes file hash */
    API_DATA_HASH(11),
    /** Data received correctly */
    API_DATA_RECEIVED(12),
    /** Data transfer failed due to hash mismatch */
    API_DATA_FAILED(13),
    /** Request process completed */
    API_PROCESS_COMPLETE(14),
    /** Connection closed */
    CONNECTION_CLOSED(90),
    /** Connection closed due timeout */
    TIMEOUT(99);
```

The first case is when everything goes as expected and the user manages to make a successful data connection & API request. The server always notifies the user about whether a certain request is successful or not, before proceeding with sending data. This is to ensure the prevention of data leak amidst illegal requests.

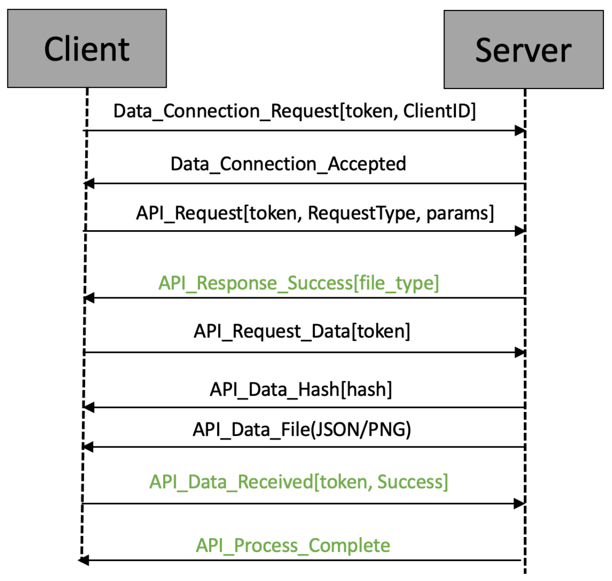

Client
Server
Data_Connection_Request[token, ClientID]
Data_Connection_Accepted
API_Request[token, RequestType, params]
API_Response_Success[file_type]
API_Request_Data[token]
API_Data_Hash[hash]
API_Data_File(JSON/PNG)
API_Data_Received[token, Success]
API_Process_Complete

Figure 1: Success scenario.

If the user makes an illegal request (if the map, city or the request type made by the user doesn't exist), then the server would send a failure message and allows user to send a new request.

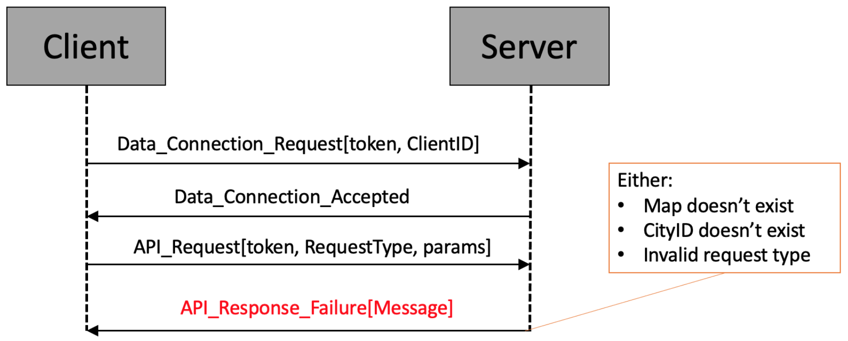

Client
Server
Data_Connection_Request[token, ClientID]
Data_Connection_Accepted
API_Request[token, RequestType, params]
API_Response_Failure[Message]
Either:
• Map doesn't exist
• CityID doesn't exist
• Invalid request type

Figure 2: Response failure scenario.

If the user makes a legal request, but the file is not received by the user due to some network error, or when the hash sent from the server doesn't match the user-calculated hash from the data; the client sends an API_Data_Failed flag; after which the server re-sends the data and hash until the transfer is successful.

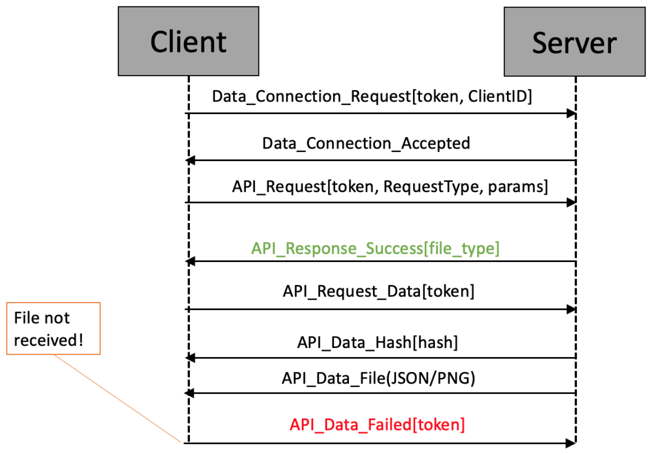


Figure 3: File transfer failure scenario.

The generic TCP payload for Query phase is illustrated below:

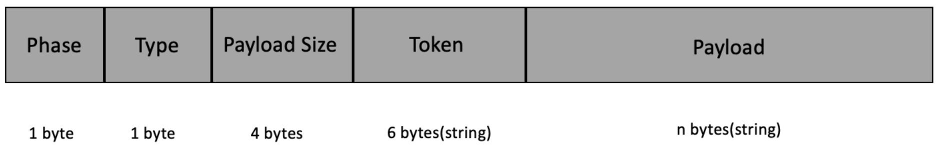
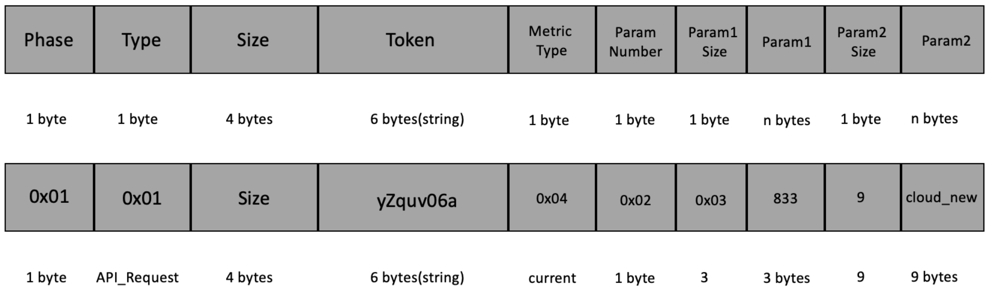


Figure 4: Generic TCP payload.

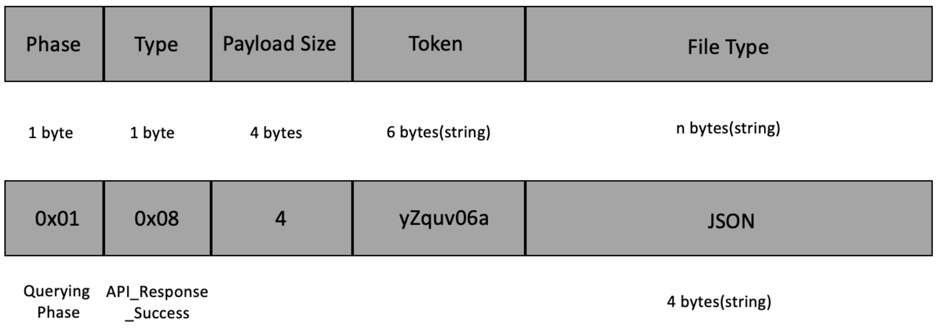
The TCP payload for API_REQUEST is illustrated below:



Phase	Type	Size	Token	Metric Type	Param Number	Param1 Size	Param1	Param2 Size	Param2
1 byte	1 byte	4 bytes	6 bytes(string)	1 byte	1 byte	1 byte	n bytes	1 byte	n bytes
0x01	0x01	Size	yZquv06a	0x04	0x02	0x03	833	9	cloud_new
1 byte	API_Request	4 bytes	6 bytes(string)	current	1 byte	3	3 bytes	9	9 bytes

Figure 5: TCP payload for API_Request.

The TCP payload for API Responses (success and failure) are illustrated below:



Phase	Type	Payload Size	Token	File Type
1 byte	1 byte	4 bytes	6 bytes(string)	n bytes(string)
0x01	0x08	4	yZquv06a	JSON
Querying Phase	API_Response _Success			4 bytes(string)

Figure 6: TCP payload for API_Response_Success.

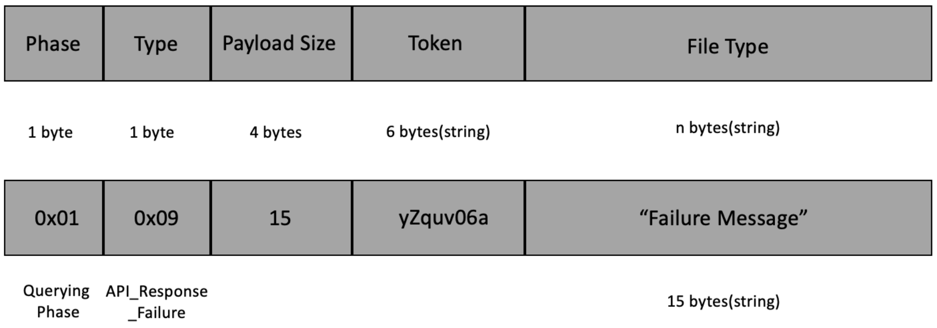


Figure 7: TCP payload for API_Response_Failure.

3 Overview of the Programming

3.1 Initial Authentication

For the initial authentication phase, we created a text file that contains all the questions and answers for each user. Then we developed a method (”retrieveUserQuestions”) that reads the file and builds a data structure that stores all the questions and answers for each individual user. Each question is linked to its answer, and each answer is linked to the next question. This is done to check the answers more easily. When a user submits an answer to a question, the program first checks whether the answer matches the value that current question is mapped to. Then, the program returns the value that the answer is mapped to, which is the next question. This process ends once the user answers the last question correctly; in that case the server returns the message ”Success” alongside a randomly generated token for the user.

If the user sends 3 wrong answers to authentication questions, server sends ”CONNECTION_CLOSED” message and closes socket and threads for that client.

3.2 Connection with the API

In the API connection side of the application, after server received request from client, it checks the parameters and according to request type, city ID and parameter types, it connects to OpenWeatherMap API and requests the necessary information. When a Json string is received from API, server saves it in a Json file so that it can send it to client afterwards. When an image received from the API, server saves it in a PNG file. Files are created in downloads folder under server.

All these steps are implemented in ”OWMManager” class. First OWMMAnager initilizes the download path for the files received from API. then a HashMap is created from city.list.json file (It is from OWM website.) to get latitude and longitude od cities from their city IDs. Then, in ”requestData” function, request type is examined to decide on which request will be done to API. Additionally, existence of first parameter (city ID) is controlled. If a history data is requested, second parameter is controlled as number of days to request from API. If map data is requested, type of map is on the second parameter again.

If parameters are missing or incorrect, error messages are thrown. After the checks, data is requested according to request type and "requestData" function returns the file location of the file that is saved to server.

For current, daily and minutely weather requests, "getCityWeather" function takes first parameter as reqType (current, daily, minutely) and second parameter as cityID. It gets the longitude and latitude of the city and creates location to save the received file. It connects to OWM API with "ConnectToOWM" function and creates a Json file out of received string from API in the created file location. It returns file location for "requestData" function.

For historical data, "getCityWeatherHistory" function takes cityID and number of days for to return historical data. day number is bounded with 5; if user requests more than 5 days, the function will give 5 days since it is the possible maximum. In the function weather data for each day is requested separately and added to a Json Object. From that object a Json file is created in the location specified in the function as it was in getCityWeather". File location is returned.

For weather map, "getCityWeatherMap" takes city ID and map type ("cloudsnew", "precipitationnew", "pressurenew", "windnew", "tempnew"). According to city ID lat and lon is obtained from HashMap like other methods. However, this time x and y map tile coordinates are obtained from lan and lon. The zoom level is set to a proper level so that a approximately a city is visible on the map. Afterwards, image is received from API byte by byte and saved to a png file in created file location.

"getCityWeather" and "getCityWeatherMap" uses "ConnectToOWM" function to connect to API. it takes url as string and returns the received Json string. It first opens a connection, requests "GET" method, then creates the input stream.

So, all the connection with API is handled in "OWMManager" class. After requested data is known, Weather reports are obtained and saved to files in this class.

3.3 File Transfer Mechanism

File transfer occurs in DataServer. Server socket is initialized on input port and server listens the line, in the case of an incoming connection, it creates and starts a DataThread in client. Socket is created according to id.

Output and input streams are created in "DataServerThread" class. To send file, a function is established which contains write command in it. After the file transfer, connection is closed.

```java
 /**
     * Starts a single direction connection
     * The connection is started and initiated as a DataThread object
     * @return Data server thread
     */
private DataServerThread acceptConnection() throws IOException
    {
        Socket s = getSocket().accept();
        // TODO: CHECK REMOTE ADDRESS WITH AUTHENTICATED ADRESSES
        String socketId = "" + s.getRemoteSocketAddress();
        System.out.println("A data connection was established with a client on the
            address of " + s.getRemoteSocketAddress());
        DataServerThread st = new DataServerThread(s);
        st.start();
        dataServerThreads.put(socketId, st);
        return st;

    }
```

3.4 File Verification

There are mainly, two data verification mechanisms. If connection is properly working, in the "ServerThread" class, token is checked in every step after authentication is reached. such as" data connection request", "API request", "API request data" cases. If a mismatch in tokens is observed, server end the connection with client.

Second verification mechanism is in the "ClientMain". In the case "API DATA HASH", data recieved and hash value is checked. and returned as true or false. If hash value is true, verification messaje is created as "API DATA RECIEVED", if hash value is false, "API DATA FAILED" message is created and client tries again.

```
//in class ClientMain.java
// Check hash
    boolean isDataValid = HashUtils.checkSHA256Integrity(serverResponse.payload,
         receivedData);
    System.out.println("Does hash value match: " + isDataValid);

// Send received type message
    if (isDataValid) {
        message = new MessageProtocol(MessageType.API_DATA_RECEIVED,token, "Data
            hash matches.");
    } else {
        message = new MessageProtocol(MessageType.API_DATA_FAILED,token, "Data hash
            does not match!");
    }
```

```
//in class HashUtils.java
//function to check hash
 public static Boolean checkSHA256Integrity(String hash, byte[] message) {
        if (hash == null) {
            return false;
        }
        return generateSHA256(message).equals(hash);
    }
```

4 Test Routines

In the testing part, first we run the server, after we are dure that server socket is opened, we run the client main. Client connects with localhost on port 4444. Then, server asks username and client writes a username. If it doen not give a user name of gives a nonexisting username, server responds as "User does not exist". If client gives a existing user name, server asks the fist question for that client. If client cannot answer the question, server returns "Incorrect answer" and gives 2 more chances and if client connot get connected in 3 trials, server closes the socket. If client gives correct answers to all question, data socket is opened and and server send a token to client. Token also checked on the client side, if recieved tokens does not match, servers closes the connection. If tokens match, client is able to send request to server. When server gets request, connect with Open Weather Map API and saves the recieced file. Then, it sends the file to client. File type is keeped so that if json is recieved, it is printed on the console. Client also saves the file it recieves from server.

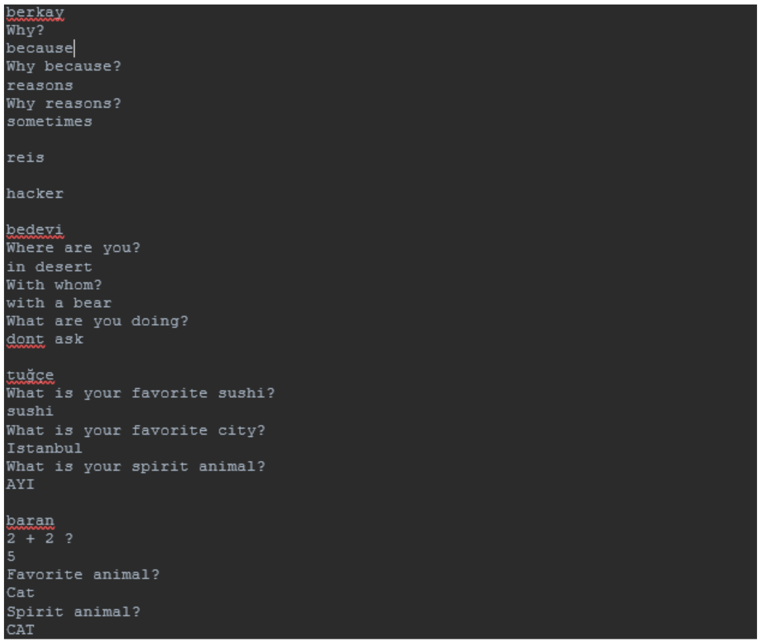
Symentaneously, many clients can connect to server and make several request. Different clients are connected to different sockets of server.

Also the hash values are controlled so that in the case of mismatch, error is reported on the console but second time, hash is corrected.

If in the request phase, city ID is given incorrect, server gives error and asks again.

If client does not answer for 1200 seconds, timeout occurs and server closes the connection.

We have 5 users introduced to server but it can be extended. User information is stored in "question-answers.txt" file under server. "reis" enters witous authentication. Hacker breaks the token. Bedevi gets a hash error. Other users are regular users and one user can connect from several times from different sockets.



```
berkay
Why?
because
Why because?
reasons
Why reasons?
sometimes

reis

hacker

bedevi
Where are you?
in desert
With whom?
with a bear
What are you doing?
dont ask

tuğce
What is your favorite sushi?
sushi
What is your favorite city?
Istanbul
What is your spirit animal?
AYI

baran
2 + 2 ?
5
Favorite animal?
Cat
Spirit animal?
CAT
```

Figure 8: Questions and answers.

There are mainly 5 request types: "current" , "daily" , "minutely", "history", "map". First three, needs the city ID for the request also. Example request: "current 833". History takes city ID and day number (Ex: "history 833 3"). Map takes city ID and map type as "cloudsnew", "precipitationnew", "pressurenew", "windnew", "tempnew" (Ex: "map 833 clouds-new").

5 Conclusion

In this project, we successfully developed an application that can handle multiple users over a network, fetch weather data from OWM on their requests, and provide a necessary authentication mechanism to determine whether a given user is authorized or not. We successfully completed the project requirements and created the necessary tests to show that our code works without any issues. We believe that this project was a proper way to obtain necessary introductory coding skills for network programming, and we believe that our success indicates a clear understanding of the course material up until this point.