COMP416: Computer Networks
Project 1

Due: March 15, 11:59pm (Late submissions will not be accepted).

Submission of the project deliverables is via Blackboard.

This is an individual project. You are not allowed to share your codes with each other.

StratoNet: NASA-API based Network Application

Introduction and Motivation:

This project work is about the **application layer** of the network protocol stack. It involves application layer software development, client/server protocol, application layer protocol principles, socket programming, and multithreading.

Through this project, you are going to develop a network application protocol by interacting with the application layer abstract programming interface (API) of NASA (api.nasa.gov). The project will require you to work with the following APIs for information extraction from the NASA web server:

1. Astronomy Picture of the Day (APOD)
2. Insight (MARS weather)

These are two separate APIs that allow to interact with the NASA hosted data and retrieve the required elements

Project Overview:

In this project, you are asked to develop a NASA-API based network application protocol based on the client/server model. StratoNet server provides two types of TCP connections to interact with the clients: One connection for exchanging the protocol commands, and one for data transfers. Fig.1 shows connections for a sample StratoNet server and client interactions. As shown in this figure 1, only the StratoNet server interacts with the NASA API web server and provides the StratoNet client with the information extracted.

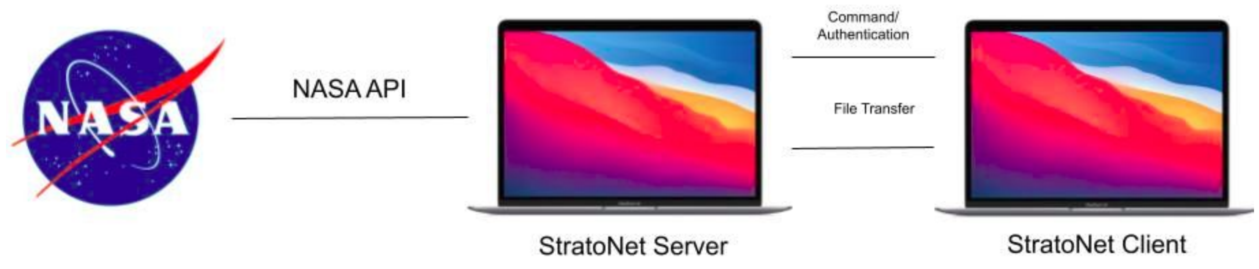


Figure 1. The StratoNet Client/Server connections.

Implementation Details:

The StratoNet reporting application has three main components:

- Interaction of the NASA API and the server
- Server side of the application
- Client side of the application

It is pertinent to note that the mentioned APIs allow for a limited number of API calls. The developed application and testing including for demonstration must keep these limits in perspective.

API Information and Interaction :
The Astronomy Picture of the Day (APOD) API works with the APOD website. The API parameters for accessing APOD can be specified as an HTTP Get request with the specified parameters. The http request contains the query parameters which when stated result in a JSON string with date, description, copyright, hdurl and url fields. The last two point to the image being hosted by the APOD website. Once working with the APOD API, StratoNet server is required to extract the hdurl/url from the returned JSON string, download the image and then pass on the image back to the concerned client over the File Socket.

The Insight API returns the MARS weather in the form of a JSON String. Each query results in the weather of the last 7 Sols (You have to find out what a 'Sol' is). While querying itself is straightforward, the data contained in the JSON object is detailed. You are required to provide the **Atmospheric Temperature and Wind Speed data** for the 1st, 4th and last sol. Conducting the extraction of this data may be done as StratoNet Server or Client, however, please keep in mind that the response to this query transmitted back to the client must be as a JSON object.

Phases:

This project has two communication phases: Authentication phase, and querying phase. During the authentication phase, the client provides its username and password to prove its identity. The protocol (the message flow, message types and their formats) is provided in the “Authentication” section. You need only to implement the provided protocol. During the querying phase, the authenticated clients communicate with the server to retrieve data from either APIs conforming to the specifications that will be explained in the following sections. **Unlike the authentication phase, we do not provide you a protocol but expect you to design your own which must be explicitly explained in the report as well.** You may use the authentication protocol as a starting point and build from there.

StratoNet Sequence overview:

1. Client/s authenticate with the server
2. After authentication, authenticated client/s send a query to the server.
3. The StratoNet server establishes connection with the required NASA API and fetches the required information (string or image)
4. The StratoNet server communicates the results and the hashes(for verification) back to the concerned client (string over the command connection and image through **File transfer socket. Hashes are always shared over command connection**).
5. Client acknowledges receiving the information. It also indicates to the server if another query is needed or closes connection.

StratoNet Server side overview:

The server for the StratoNet application should:

1) Establish a connection with the respective NASA web servers using the APIs and download the specified required information.

2) Authenticate any client before initiating any data exchange.

3) Allow multiple clients to connect using multi-threading with the same functionality.

4) Provides hashes of the files/string for error detection purposes.

Server Actions

For the interactions to take place, the client will pass the lat/long of a city for APOD and the weather trigger for Insight API. All this information shall be conveyed between the client and the server on the Command socket (the socket allocated to the client after acceptance and validation).

Based on this input, the server will parse the client input and use the API to extract the relevant information from the web server. This information will then be passed to the client in the form of an image file (APOD) over a File Transfer Socket. The server will be responsible for generating a hash value of each file/string before transmitting to the client and this will also be sent to the client for file verification over the Command Socket.

The process at server side would be:

1) Create welcoming socket
2) Accept incoming client connections at the welcoming socket, simultaneously if needed using multi-threading.
3) Authenticate any incoming client connection requests **after acceptance**. Create an additional Data socket with the client if authenticated. Terminate connection if authentication fails. (Demonstration should present both scenarios in which clients are validated as well as rejected).
4) Decipher the requests coming in from the clients and download the required files from the web server. (The protocol for forwarding requests has to be developed by yourself and explained in your report. A similar format is provided in the Authentication part).
5) Generate the hash values of the file/string requested by the client.
6) Send the hash value of the required file over the Command socket.
7) Send the requisite file over the File socket.
8) Terminate connection if a “file/JSON received” acknowledgment message is received from the client and no other files are requested **within timeout duration**.

StratoNet Client Side Overview:

This noapplication envisions multiple clients interacting with a single server. For this application, the clients should:

1) Confirm its authenticity with the server
2) Be able to submit requests to the server.
3) Be able to receive data in form of JSON/image files from the server over the Data socket.
4) Verify the file/JSON based on its hash value.
5) Display the data in tabular form or display the image on the client side

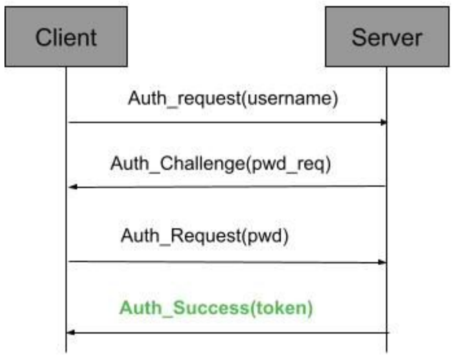
Client-Server Interaction
The client side must take care of the parameters to be passed once requesting any metric.

The process at the client side will follow the given steps:

1) Initiate connection with the server over Command Socket
2) Authenticate based on the server requirements.
3) Receive the parameters for File Socket and connect to that after authentication.
4) Pass the requests to the server
5) Receive the hash value of the file on the Command socket
6) Receive the file over the File socket
7) Confirm if the hash value corresponds to the file
8) Request retransmit from the server if mismatch between hash value and file or failure to receive file (Relevant String should be displayed in terminal). A scenario for this step may be specifically designed for demonstration purposes.)
9) Display the files in the appropriate manner.
10) Terminate the connection in case an appropriate file is received and no other request forwarded within the timeout duration. (Appropriate tests for demonstration purposes should be developed.)

Initialization Phase

In your implementation, the server does connect with everyone. The initialization phase requires the clients need to be authenticated in order to be able to query the server, which will be done by a simple password-based authentication. After the authentication is done, the client will receive a “token” from the server. The token will act as a “proof” of a valid client. From that point on, the client will need to append this token to its requests, and once the server receives a request, it responds only if the token is valid. Please note that the authenticated clients are authorized to perform all possible API queries.



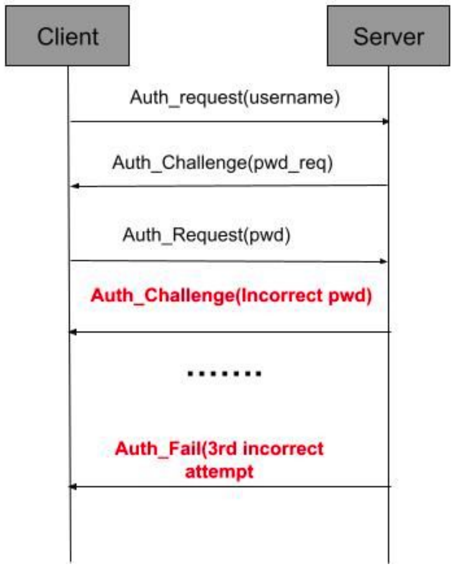


Fig. 2. The message flows for the initialization phase.

First, the client will send its username to request to be authenticated (i.e., acquire a token). Then, the server will authenticate the client by requesting the password using the **Auth_Challenge** message. The client will prompt the user, only in case of a valid username, the user will enter the password through standard input, and the client will send the answer by including it into an **Auth_Request** message.

If the correct password is supplied by the client, the server will send back an **Auth_Success**, including a unique token for the client. If the client provides incorrect password, the server will resend the **Auth_Challenge** two more times requesting the correct password. If the three attempts fail, the server sends an **Auth_Fail** with the reason of failure message as “Incorrect password”. Please note that the server may also send back an **Auth_Fail** when the first **Auth_Request** includes a nonexistent username. In this case, the reason for failure should be “User does not exist”.

Figure 2 illustrates the intended message flow

a. Message format

In protocol design, (1) the message types, and (2) the format & the meaning of the values in each type of message must be clearly specified. The client and the server will handle the received messages according to their types. Similarly, they will construct the messages adhering to the protocol.

Message types

During the authentication phase, the client is able to send only **Auth_Request** messages, and the server is able to send **Auth_Challenge**, **Auth_Fail** and **Auth_Success** messages.

Message type	Value	Payload
Auth_Request	0	Username/Answer (String)
Auth_Challenge	1	Question (String)
Auth_Fail	2	Reason of failure (String)
Auth_Success	3	Token (String)

Deconstructing the TCP data
The TCP data received from the socket must be deconstructed correctly. The first six bytes are designated as the application header, where the first byte represents the "phase" (either the Initialization (0) or querying (1) phase.), and the second byte represents the "type" of the message. If the "phase" byte is set to 0, then the messages will be handled by the authentication module of our implementation. Otherwise, your implementation should hand over the handling of the request to the API querying module. The remaining four bytes of the header are designated as an integer (4 bytes) denoting the length of the payload in bytes. Your application should use this value to read the correct number of bytes from the TCP stream as the payload, since we do not know the length of the payload beforehand. Figure 3 shows the deconstruction of an authentication message.

Here are some useful links:

https://docs.oracle.com/javase/7/docs/api/java/io/DataInputStream.html
https://docs.oracle.com/javase/7/docs/api/java/io/DataOutputStream.html
https://docs.oracle.com/javase/7/docs/api/java/io/FilterOutputStream.html

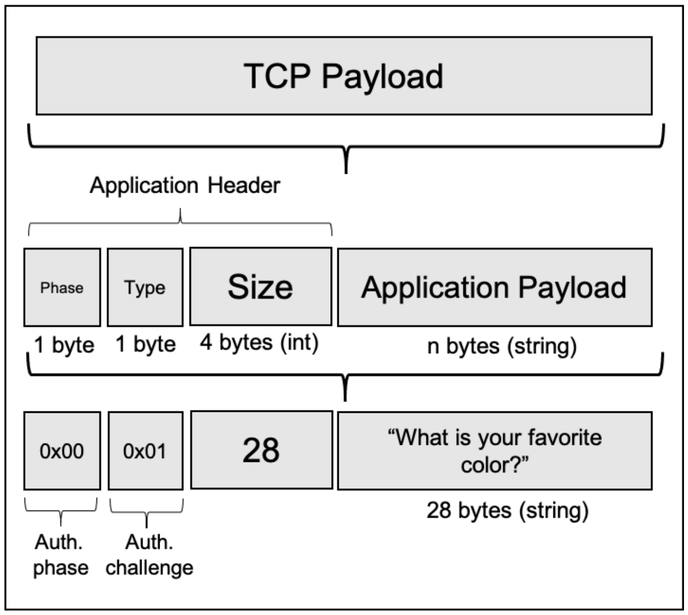


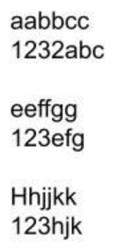
Fig. 3. Deconstructing the TCP data.

b. Timeout mechanism

You also need to handle the cases where the client is unresponsive to a password request. After sending a challenge, the server should wait only for a predetermined amount of time (e.g. 10 seconds) before sending an **Auth_Fail** to the client with the appropriate reason message and closing the connection.

c. Implementation details

Storing users, questions, and answers
For simplicity, you are required to keep all the usernames and passwords in an easily parsable text file and not hardcode them in your code. Figure 4 illustrates an example of such a file, where we have three users with different passwords.



```
aabbcc
1232abc

eeffgg
123efg

Hhjjkk
123hjk
```

Fig. 4. An example of a text file storing usernames and passwords.

The token
The token should be unique for each session. Ideally, they should be constructed on-the-fly. You can construct a token by performing a hash on the concatenation of the username and the last two digits of your KUSIS ID. Then, the token will be the first n (e.g., 6) characters of the output. After that, you should save the token along with the corresponding client IP + port and username, so that the server can verify the token in the future during the querying phase.

Architecture
The authentication module of the server and client would communicate with each other (through the command socket) to agree on a token in the authentication phase. In the querying phase, the only responsibility of the authentication module would be as following:

- **At the client:** Append the token to the message received from the NASA API module before sending the message through TCP to the server.
- **At the server:** Verify the received token appended to the received message from the TCP before supplying it to the NASA API module. Then, the Server API module could simply focus on communicating with the Nasa API.

Execution Scenario

Due to the fact that the classes are being conducted online, the client and server are expected to reside on a single machine for simplicity of both execution and demonstration. However, any student preferring connection to a Remote device is free to do so albeit with the necessary consideration given to the overall execution and demonstration scenarios.

The project envisions you to implement at least five unique clients and one server.

Project deliverables:

You should submit your source code and project report (<u>in a single .rar or .zip file</u>) via Blackboard.

Report

The report should start with a very brief description of the project. This should be followed by an explanation of the philosophies especially post-authentication server-client protocol. Following this, you should provide an overview of the programming of the client and the server side including the initial authentication, connection with the API, the file transfer mechanism and the file verification. Instead of attaching complete code, it is strongly recommended to attach code Snippets and images of the results of their execution. **The report should explicitly describe the test routines developed to evaluate the full range of features required for the StratoNet.**

- Source Code: A .zip or .rar file that contains your implementation in a single Eclipse or Intellij IDEA project. If you aim to implement your project in any IDE rather than the mentioned one, you should first consult with TA and get confirmation.

- The **report** is an ***<u>important part of your project</u>*** presentation, which should be submitted as both a .pdf and Word file. Your report should show the step by step NASA API configuration and connection with your code. As well as your server-client communication initiation. **Report acts as a proof of work for you to assert your contributions to this project.** Everyone who reads your report should be able to reproduce the parts we asked to document without hard effort and any other external resource. If you need to put the code in your report, segment it as small as possible (i.e. just the parts you need to explain) and clarify each segment before heading to the next segment. For codes, you should be taking ***screenshots*** instead of copy/pasting the direct code. Strictly avoid ***replicating the code*** as whole in the report, or leaving code ***unexplained***. You are expected to **<u>provide detailed explanations</u>** of the following clearly in your report:

Demonstration:

You are required to demonstrate the execution of StratoNet Application for the defined requirements. Your demo sessions will be announced by the TAs. Attending the demo session is required for your project to be graded. You are supposed to attend your individual demo session. **The on time attendance at the demo session is considered as a grading criteria.**

During your demonstration of the authentication phase, you will be asked to demonstrate **two** clients being authenticated at the same time. You can assume that all the clients will present different usernames. <u>You need to make sure that all the users have different correct passwords.</u> The clients and the servers will be running on the same machine, so different clients should be running at different ports. In this vein, the server will differentiate the clients not only by their IP address, but also with their port number. For the demonstration, various modes of authentication may be tested. First, you will need to show an unsuccessful authentication, then you will need to show a successful authentication.

During the demonstration, you will be asked to display all the operations of the application starting from initiating client connections and authentication to passing requests for the listed metrics. It is strongly recommended that the appropriate test routines may be developed to present an effective demonstration and display the full range of features as defined for StratoNet application.

You have the creative freedom to present any additional features you may have built into the application in any way you deem feasible. However, please note that you will be given 5-10 minutes for your demonstration.

Following is a detailed but not exhaustive list of the test routines you should develop to aid them in testing and demonstration:

1. Client creation. The application should be designed with client scalability in mind even though the testing would be done with max 5 clients.
2. Single/multiple client connection with the server
3. Connection refused due to timeout/ incorrect password on 3rd attempt
4. The process from initiating a request by the client side to the final verification of the received file at the client side and file display.
5. Process in case of a mismatch between received file and received hash.

Good Luck!