Project 01

Simple Application Protocol Design
(Implementation Using Socket Programming)

1 Overview

This exercise provides hands-on experience in designing a simple application protocol and implementing the same using socket programming. This task is recommended to be done in a team of two. One team member should implement client-side application and other team member should implement server-side application. Alternatively, you can do it individually as well (Note: doing individually will not yield any relaxation in assignment submission).

The project has two submission deadlines. The first submission corresponds to submitting the design document and second submission corresponds to protocol implementation.

1.1 Learning Objectives

- Develop familiarity with Application Protocol design and implementation.
- Develop basic knowledge of TCP socket programming.
- Working as a team and develop communication skills.

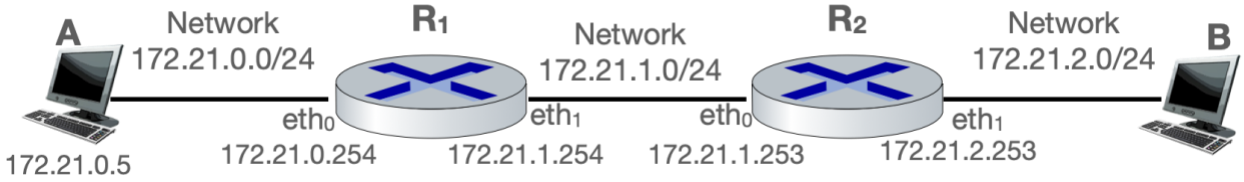
2 Learning Resources

1. Socket Programming HowTo (Python3): https://docs.python.org/3/howto/sockets.html
2. Netcat: Test and establish TCP/UDP connections. https://www.digitalocean.com/community/tutorials/how-to-use-netcat-to-establish-and-test-tcp-and-udp-connections

3 Prerequisites and environment familiarity

For this exercise, you can use the VM 130.85.121.14 using your assigned account.

For this project implementation, create a network (as shown in Figure 1) consisting of two hosts connected via two routers. The IP addresses shown are for illustrative purposes and should be changed as per assigned IP addresses based on your UMBC Id.



Create a docker compose file that contain details of IP addresses for the respective interface and their network routes. An example configuration file (`mynetwork.yml`) is provided. Please feel free to change the docker file name or container names in the docker file as per your choice.

Note: This setup requires use of three IP subnets. Let us denote them as 172.21.X.0/24, 172.21.Y.0/24 and 172.21.Z.0/24, where X corresponds to last 2 digits of your UMBC id, Y=X+1,

and Z=X+2. Corresponding address of host A and host B would be 172.21.X.5 and 172.21.Z.5. Similarly, IP addresses of router R1 would be 172.21.X.254 (eth0), and 172.21.Y.254 (eth1). The IP addresses of router R2 would be 172.21.Y.253(eth0) and 172.21.Z.253(eth1). In the example file and Figure 1 shown above, values of X, Y, and Z are taken as 0, 1, and 2.

4 Assignment Description

4.1 Simple Protocol Design

Design a simple Application Layer protocol to accomplish some basic service functionality. A few examples are given below. You can design and implement your own application on similar lines or use any one of these.

a. Manage a To-Do List or Calendar activities. An entry would have following (or more) fields of information
    i. Start Date/time
    ii. Duration or end time
    iii. Task/To-do description
    iv. Notes or miscellaneous information
b. Manage a class schedule or time table. An entry would have following (or more) fields of information
    i. Subject Code e.g. CMSC481
    ii. Room number
    iii. Day and time slot
    iv. Start and end date
c. Managing grades. An entry would have following (or more) fields of information.
    i. Semester e.g., Spring-2024
    ii. Subject code
    iii. Grade obtained
    iv. Credit hours

The protocol should have following 3 (or more) phases of communication:

a. Session establishment i.e., Authentication.
b. Implementation of Application functionality by exchange of Application messages.
c. Session termination

The implementation is to be carried out using TCP connection. For simplicity of debugging and testing, design the protocol in clear text i.e., all keywords, etc. should be in clear ASCII text and capital letters. Implement using your preferred programming language e.g., python3, C, C++, Java, etc., for implementation of the protocol. Brief details of each phase are given below:

i. Phase 1: Session Establishment. The client should be able to authenticate to the Application server using a simplified variant of CHAP (Challenge Handshake Authentication Protocol). For example, after the client establishes a TCP Connection with the server, it should identify itself using some keyword e.g.

    ID: <user identity>

Choose your own keyword instead of ID.

- Server responds with a token string of 10+ random characters, called challenge.
- Client appends the password to this token, generates a md5sum (hash) and responds.
- Server compares the received the response, generates its own hash using this token string and user's password and if the two matches, returns success else fails authentication.
- The server should maintain a predefined set of identifiers, e.g., these could be a set of UMBC IDs, phone numbers, or any other values and if the identifier matches among the known values, it returns a success message otherwise failure. For example, response to authentication could be a status code and its description.
    - 200 SUCCESS
    - 400 FAIL

Note: You should preconfigure the server with some authentication credentials (username/password). These can be simply kept in server memory or configuration files which can be read at start time.

ii. Phase 2: Perform few (or even none) transactions. These would correspond to retrieving existing information or adding/deleting information about authenticated user. Keywords identifying such messages from clients could be as below. Again, define your own keywords.

a. RETRIEVE: obtain the task information already stored with server for the user
b. ADD: Create new task info and store with server
c. DELETE: Removed the specified task info

Note: All the information can be maintained in memory and no persistent storage required. Thus, when server application starts, it is acceptable that it has no information (except credential) about tasks performed by the user. It would be helpful if each response server contains the status code followed by the actual data.

Along with the message keyword, you need to define the required syntax and semantics for the messages. The server should respond according to a message received from the client. For example, RETRIEVE should provide details of all (default) or specific requested items stored in the server repository, whereas ADD should create and new information item etc. Make your own decision whether you would like to permit duplicate items.

For any unrecognized keyword, the server should return an error. Define your own error codes or messages. Similarly, requesting a non-existing specific item should return a corresponding error.

Implement a session time out, that is, if for some time (configurable at runtime), there is no communication from the client, the server should simply close the session. Any communication received from the client after time out should result in an error response.

iii. Phase 3: The last phase of this task management is logout. This could be a BYE (or EXIT/LOGOUT) message and the server would respond accordingly.

Note:

- Ensure your programs are robust and neither client nor server should crash for any invalid or unexpected input.
- The protocol communication should be in clear text (ASCII) thus one should be able to mimic both client and server using Netcat. The use of Netcat (nc) would be a helpful tool to debug the implementation of client and server. Netcat works as both client and server, your client and server should be able to communicate with Netcat as server/client respectively.

4.2 Concurrent access

The server should support multiple different concurrent clients. That means if 3 different users are connected to the server, each should be able to work independently. It is fine if servers disallows multiple sessions for the same user (i.e. same credentials).

4.3 Bonus work

i. Use of select()/poll()/epoll() for handling concurrent TCP connections i.e. a single server thread handling all TCP connections.
ii. Implement communication over SSL. Thus, all communication should occur in a secure way.
iii. Implementation of client and server in different programming languages. For example, client in C and server in Java. This demonstrates clarity of protocol communication.
iv. Implement the Wireshark plugin for your protocol. This plugin should add the capability to wireshark so as to decode the application layer protocol and show messages as per protocol format. (Explore Lua)
v. Handling of pipelined request and response that makes use of TCP streaming. For example, consider the request message “RETRIEVE <data>”, this could be sent by client in two different send() API calls as “RET” and “RIEVE <data>”. Essentially, Both clients and server should be work with receiving partial data in on TCP recv() call and thus need to invoke multiple recv() calls to get the desired data.

5 Assessment

5.1 Protocol Design

1. A Readme.txt file provides the following:
    a. Team details: UMBC IDs and Names of both team members.
    b. Brief description of application task.
    c. Summary of your learning
    d. Approximate duration (in days) spent in completing the assignment.
    e. Challenges faced in doing the assignment and how did you address it.

2. A document (.pdf or Word) providing detailed protocol design i.e., messages/requests being sent by the client and corresponding responses from the server. Provide syntax and semantics details of both request and response.
3. Timeline sequence diagram for at least following 3 cases.
    a. Successful authentication and logout. No transaction
    b. Successful authentication with two or more requests (for adding and one request for retrieving information)
    c. Timeout after successful authentication

5.2 Protocol Implementation

1 A Readme.txt file provides the following:

a. Team details: UMBC ID, Name of both the team partners
b. Summary of your learning
c. Challenges faced in doing the assignment and how did you address it.
d. Approximate duration (in days) spent in completing the assignment.

2 Both the client and server programs:

3 Evaluation:

i. Protocol Design and time line sequence diagrams: 20 marks
    a. 10 marks for the protocol design document
    b. 10 marks for Timeline Sequence diagrams.
ii. Protocol Implementation
    a. 10 marks for client
    b. 20 marks for server side
iii. Bonus marks
    a. 5 marks for using select()/poll()/epoll().
    b. 5 marks for SSL-based implementation for secure communication.
    c. 5 marks of implementing client and server in different programming languages.
    d. 5 marks for handling of pipelined requests and streaming data.
    e. 10 marks for the Wireshark plugin/addon. This should show protocol level formatting in wirehshark protocol pane.

4 Note

Any plagiarism activity will result in penalties of being awarded 0 marks.

<end of Project 01>