NETWORK

PROBLEM DEFINITION:
The aim of the project is to design a simple version of network communication between peers within a network. The network protocol stack for this project consists of application, transport, network and data link layers.

METHODS AND SOLUTION:
Frames are implemented as stacks and teach client will have an outgoing and incoming queue. Both are defined using structures in C.

The source code is divided into three functions:
1. main()
2. For frame – push() and pop()
3. For queue – insert() and remove()
4. add_log_entry()

The **main()** function takes care of:
1. Processing of input arguments.
    a. There will be six input arguments to the executable, namely:
        i. Client related info in a file. Eg: clients.dat
        ii. Routing table in a file. Eg: routing.dat
        iii. Commands to the network in a file. Eg: commands.dat
        iv. Maximum message size that a frame can store. Eg: 20
        v. Incoming port for a client. Eg: 607
        vi. Outgoing port for a client. Eg: 706

    ```
    $ ./code clients.dat routing.dat commands.dat 20 607 706
    ```

    The above line is the command line execution of the executable 'code'.
    b. We load the client related info into the arrays: 'client_ids', 'clients_ips' and 'client_macs'.
    c. We load the routing table in the array: 'routing' and maximum message size, incoming and outgoing ports into the variables 'max_msg_size', 'in_port' and 'out_port' respectively.
    d. We start reading in line-by-line from the commands.dat file and process each one as shown below.
2. Iterating over all the commands and processing them.
    a. Command "MESSAGE"
        i. This takes care of dividing the message into sub messages, if the message length is more than a frame can handle, before populating the frames in the outgoing queue of the sender.
        ii. It is also responsible for the frame creation for the sender and then insertion of each build up frame into the outgoing queue.
    b. Command "SHOW_FRAME_INFO"

        i. This gets the frame being specified by the command arguments and displays information about it. If the frame is not found, it displays the corresponding error messages.
    c. Command “SHOW_Q_INFO”
        i. Similar to the SHOW_FRAME_INFO, this command searches for the specified queue for the specified client and then display it’s information.
    d. Command “SEND”
        i. This takes care of sending the frames from the sender to the destination. It starts off with an infinite loop, which breaks when the frames have reached the destination or there’s a packet loss.
        ii. Within the loop, each time the receiver MAC is checked with the destination MAC, and if they are not same, the frames will be forwarded accordingly, or else either the frames have reached destination or lost.
        iii. In order for the frames to be forwarded, the frames in the incoming queue of the receiver are inserted into outgoing queue of the receiver, before making the current receiver as sender for the next iteration.
        iv. Whatever may be the case (packet lost/forwarding/message reached the destination) with the receiver, the corresponding log is added to the all_log_entries array using the utility function add_log_entry(), which is explained below.
    e. Command “PRINT_LOG”
        i. This command takes care of displaying the log messages corresponding to the specified client. This takes information stored in the all_log_entries array and simply displays it as a list.
    f. For any other invalid commands
        i. If any command other than the above, the program will display an invalid command message.
3. All these commands are checked using nested if-else-if conditions and processed accordingly.

The **push_*()** and **pop_*()** functions:
1. Takes care of inserting and removing from the frames layer-wise.
2. Each push or pop checks for the validity of the stack top, before performing respective task.

The **q_insert()** and **q_remove()** functions:
1. Takes cares of inserting and removing frames from the incoming and outgoing queues of the clients.
2. The frames corresponding to every individual queue are implemented as a simple array.
3. Each insertion will append a new frame to the end of the existing frame array. Each removal will take out the frame at the beginning of the frame array and adjust the remaining frame array accordingly.

The **add_log_entry()** function:
1. Takes care of creating and appending new logs to the existing all_log_entries array.

2. This is a utility function, which can be called from anywhere with necessary arguments for creating a new log entry for the specified client.

FUNCTIONS IMPLEMENTED:

1. Dynamic allocation using malloc() – all the arrays are dynamically allocated.
2. Multidimensional arrays are implemented.
3. Frames (stacks) and incoming, outgoing queues are implemented using C structures with corresponding functions for push, pop, insert and remove capabilities.
4. Necessary commenting in the source code.