# PROGRAMMING ASSIGNMENT 2
# HUBMB Computer Networking

**Subject:** Stack, Queue, and Dynamic Memory Allocation
**Advisors:** Assoc. Prof. Dr. Sevil ŞEN, Asst. Prof. Dr. Mustafa EGE, Dr. Cumhur Yiğit ÖZCAN
**TAs:** Merve ÖZDEŞ, Alaettin UÇAN, Pelin CANBAY, Selim YILMAZ
**Programming Language:** C
**Due Date:** 25.11.2018 (23:59:59)

## 1 Objectives

Your task is to design a simple version of network communication between peers within a network; that is, to implement a highly simplified computer networking protocol family similar to the Internet protocol suite.

The network protocol stack will consist of four layers each of which will have a different purpose as shown in the Table below:

| Layer | Protocol Services |
|---|---|
| **Application** | Communication between applications that use the network. |
| **Transport** | End-to-end data delivery. |
| **Network** | Defines packages and provides routing. |
| **Data Link / Physical** | Routines for accessing physical media. |

The network will have a peer-to-pear model. An example network topology is shown in the figure below.

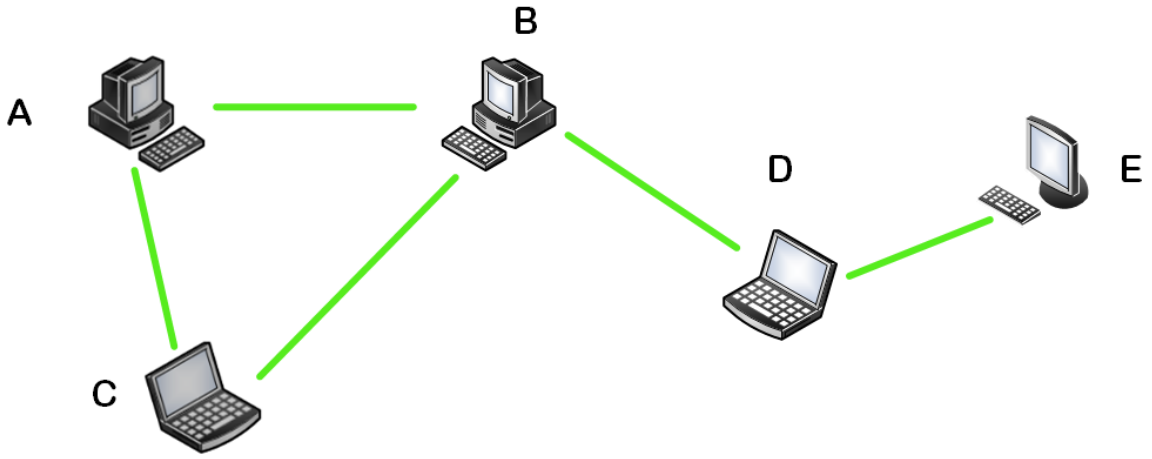



Figure 1: An example of a network topology

The objective is to enable the peers (we'll refer to them as clients) to communicate to each other by exchanging messages while abiding by a set of protocols.

## 1.1 Basic Layer Services

- A client application will place a request to send a message of an arbitrary length to another client in the network. The Application Layer will be responsible for dividing the message into message chunks of allowable size, since the links in the communication network will have a limited message size that can be delivered in a single frame over the physical media. At this point, the sender's and the receiver's IDs will be relevant to identify the communicating parties. Each message chunk will be sent in a separate frame.
- Transport Layer is responsible for appending the correct port numbers to the outgoing frames, based on which application is trying to communicate and whether it is an outgoing or an incoming socket.
- Network Layer needs to append the information regarding the IP addresses of the sender and the receiver to the frame.
- Data Link / Physical Layer will finally append the physical MAC addresses of the clients that are on both ends of the link over which frames will make the next hop.

## 1.2 A Summary of Network Services

A summary of network services that you need to implement is as follows:

- Enable communication between the clients within a network based on the set of protocols.
- Provide a simple routing of data packets using the clients' routing tables.
- Enable logging of all network activity.
- Print log reports as necessary.

## 1.3 Network Protocols

### 1.3.1 Frames as Stacks (Last In First Out)

Before data packets that carry message fragments can be sent, they will have to be encapsulated into Data Link/Physical layer **frames** which **must be implemented as stacks** of information related to each network layer (e.g. message fragment with sender and receiver IDs for the application layer, the sender and receiver's IP addresses for network layer, port numbers for transport layer, physical MAC addresses for the physical layer, etc.) as defined by the network protocol stack. A frame structure is shown in the figure below.

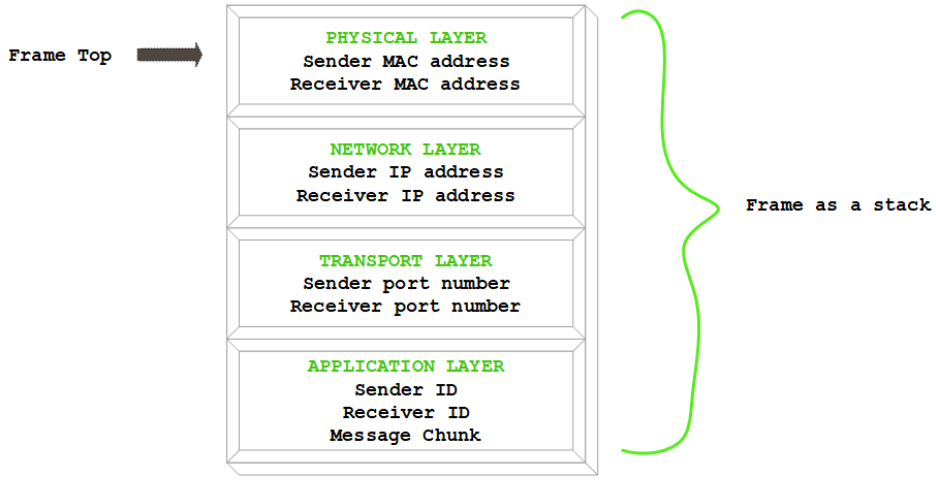



Figure 2: An example of a frame structure

### 1.3.2 Outgoing and Incoming Queues (First In First Out)

Each client will have send and receive buffers (**that must be implemented as queues**) for outgoing and incoming messages respectively (see the illustration below). You may assume that the buffer size is infinite; i.e. no packets should ever be dropped because there is not enough space on a queue; rather, the queue will expand as much as necessary to accommodate any frames that need to be placed. You need to dynamically allocate and manage (reallocate or free) memory used by all queues as necessary.

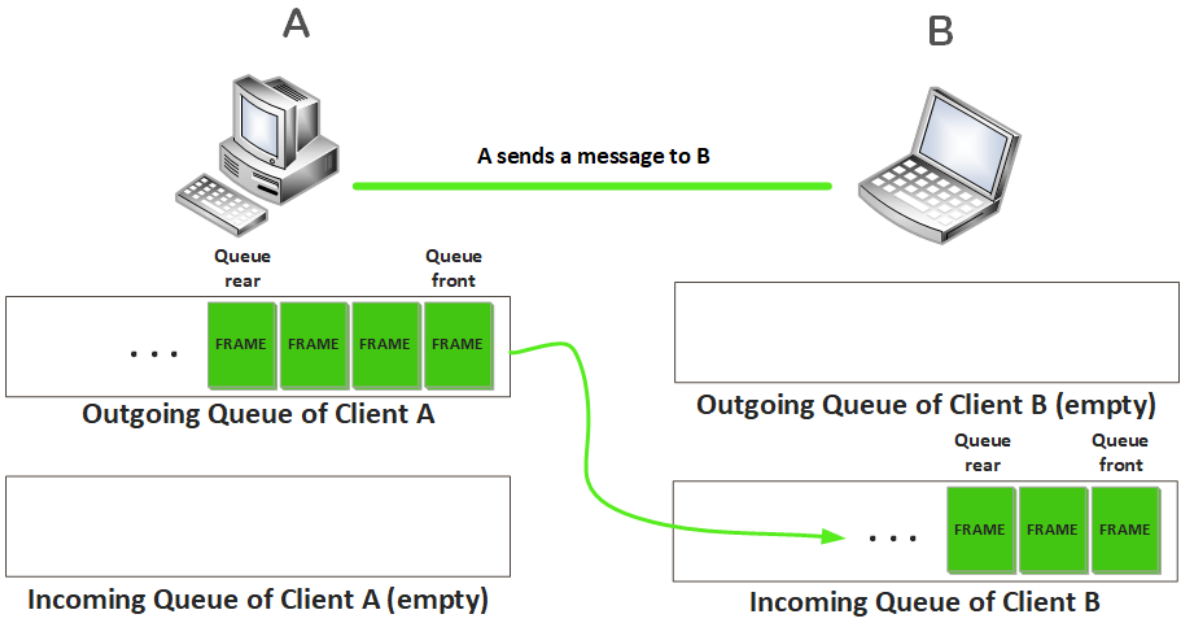



Figure 3: An example of communication between the clients A and B

The figure above illustrates an example communication between a client A and a client B in which A is messaging B. The message is first encapsulated in frames each of which carries a message fragment (when the message size exceeds the maximum allowable message length, one message will be sent in multiple frames). As a frame leaves the sender's outgoing queue, it is inserted onto the receiver's incoming queue. Frames must follow the FIFO method (the first message chunk will be sent in the first frame and received in the first frame on the receiver's side).

### 1.3.3 Routing

When a client receives message frames in its incoming queue, it must check who the intended receiver is. If the frame is addressed to the receiving client, it can proceed to unpack the message chunks as soon as the last message fragment has arrived, and pass them to the application layer based on the port number of its listening socket in order to reassemble the message. However, if the message is intended for someone else, the client must check its routing table to determine the next hop that the frames should take to their destination.

Routing tables of each client will have the following format:

| **Intended Destination** | **Neighbor to which the packet should be forwarded** |
|---|---|

Routing tables will have multiple rows, one for each client in the network. For instance, the entry | A | B | means that if a client needs to reach A, it should send the frame to B.

A routing example is illustrated in the figure below. In this scenario, client A is trying to send a message client D. The message is longer than the maximum allowed message length, so it is fragmented into four frames. A checks its routing table entry for D. Since A does not have a direct link to D, the routing table specifies which client is on a path that can eventually reach D, and if such client exists (in this case it is client B), A sends its message to that client for forwarding. If the information about the next hop is missing for some reason, the message will be dropped. When B receives the message intended for D, B must check its routing table to determine the next destination for the message frames. In this case, D is a direct neighbor of B (evident from B's routing table entry), so B forwards the message to D.

Frames may have to make multiple hops over the network before they reach they final destination. The information about the total number of hops will also need to be saved in the logs.

### 1.3.4 Logs

Every network activity must be logged. Thus, each client will have its own log which will store the information about the sent, received, and forwarded messages. Each log entry should store the following information:

- Timestamp: the date and time of the activity,
- Message that was carried in the frames,
- Total number of frames that carried the whole message,

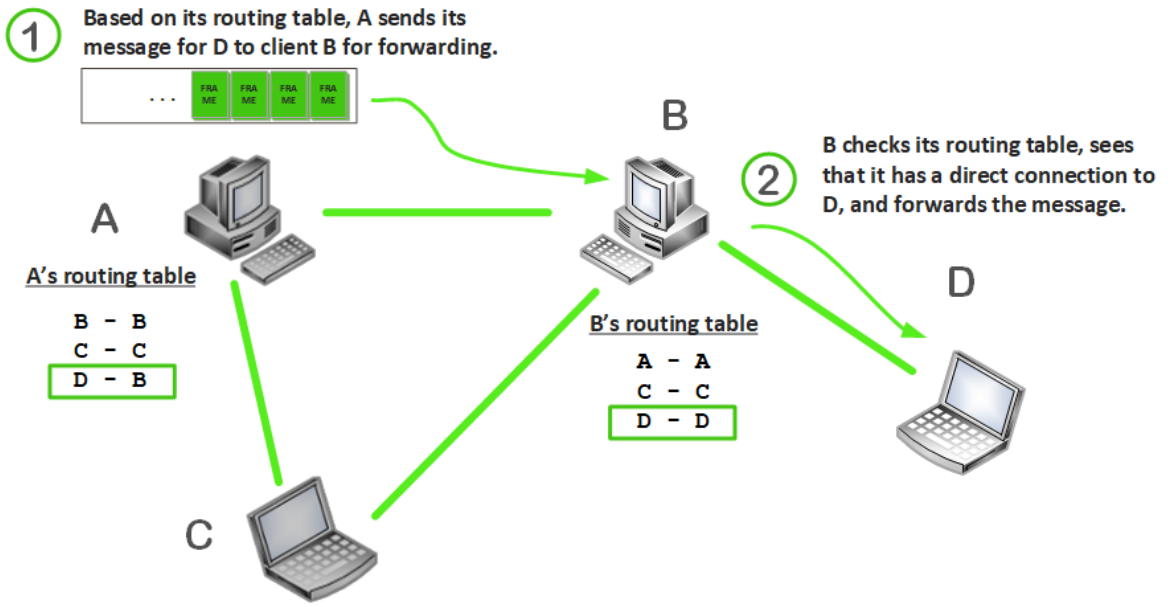



Figure 4: A routing example

- Total number of hops that frames have made so far through the network,
- Sender ID,
- Receiver ID,
- Activity type (message sent, received or forwarded), and
- Activity success status (successful or failed).

## 2 An Example Use Case

Assume we have a network shown in Fig. 1 in which the maximum message length supported by the links is 20 characters. The routing tables of each client would be as shown below.

Also assume that the client IDs, addresses, and the socket port numbers are as follows.

| Client ID | Client IP Address | Client MAC Address |
|---|---|---|
| A | 1.2.3.4 | AAAAAAAAAA |
| B | 4.3.2.1 | BBBBBBBBBB |
| C | 8.8.8.8 | CCCCCCCCCC |
| D | 9.9.9.9 | DDDDDDDDDD |
| E | 0.0.1.1 | EEEEEEEEEE |

| Service | Port Number |
|---|---|
| Sending socket | 0706 |
| Listening socket | 0607 |

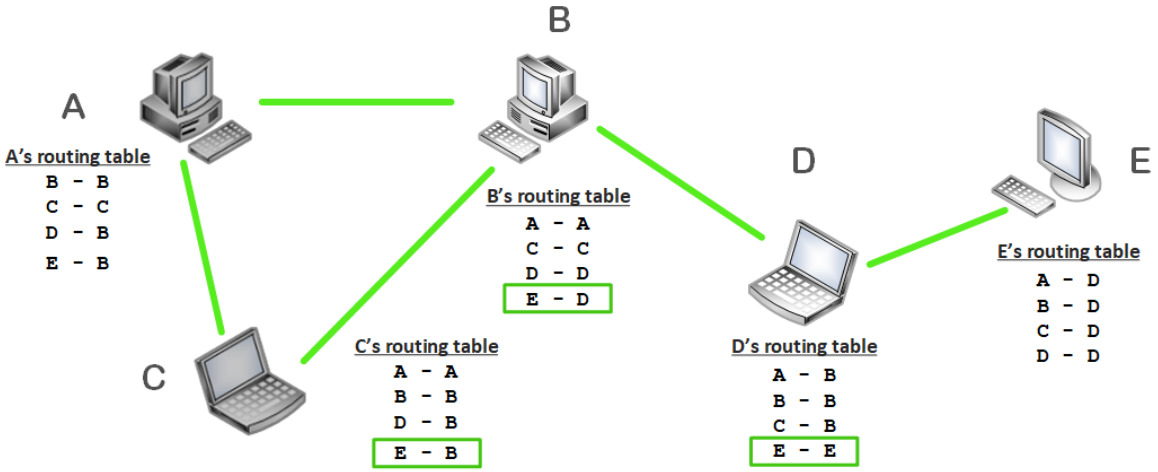


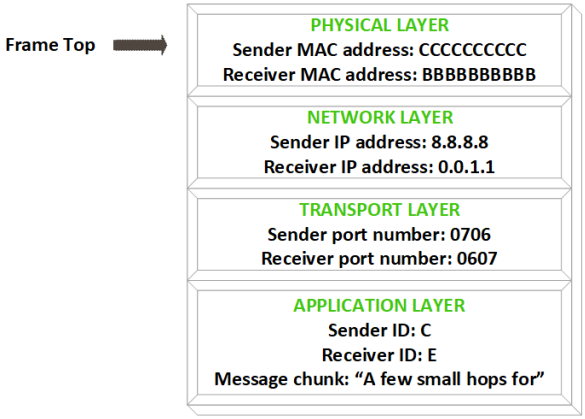

Figure 5: A demonstration of routing scenario

Let's suppose that client C wants to communicate with client E and send the message with the following content: "**A few small hops for frames, but a giant leap for this message.**" Since the length of the whole message is 63 characters and the maximum supported frame message length is 20, the message will have to be sent over four different frames with the following message chunks (note that spaces also count as characters):

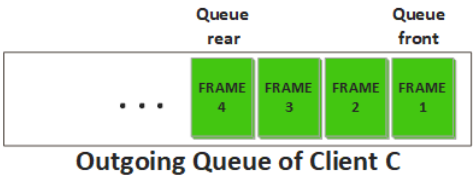
1. "A few small hops for"
2. " frames, but a giant"
3. " leap for this messa"
4. "ge."

The following steps will need to occur within the network for this communication to happen:

1. C will fragment the message and encapsulate it within four frames to be sent, and place the frames in its outgoing queue. For example, the first frame would have the following format:

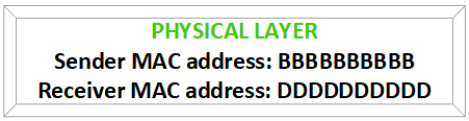


The remaining three frames will be similar, with the only difference in the message chunk they carry. After this process, the outgoing queue of client C will look as shown below.



Outgoing Queue of Client C

2. C checks its routing table to determine the next receiver. From C's routing table it is evident that the next hop for packets intended for E should be client B, since C is not E's direct neighbor. Therefore, C will send the frames to B, adjusting the Physical layer information (MAC addresses) accordingly, to reflect the fact that the frames will make the next hop over the C-B link.
3. B will receive the frames in its incoming queue in the same order they are removed from C's outgoing queue. This process must be performed using queue operations; i.e. as a frame is removed from the sender's outgoing queue, it will be inserted onto the receiver's incoming queue, until all frames have been transferred.
4. B will inspect the receiver information within the frames to check the intended receiver. It will see that the message is intended for another client in the network, namely E. So, B will remove the frames from its incoming queue and place them on its outgoing queue for forwarding.
5. B will check its routing table to determine the next hop the frames should take, and see that the frames should be forwarded to D. At this point, MAC addresses of the frames should again be adjusted accordingly to reflect the fact that the frames will make the next hop over the B-D link. Physical layer of each frame becomes:




6. Similarly to the previous receiving case, D will receive the frames in its incoming queue in the same order as they are removed from B's outgoing queue.
7. D will inspect the receiver information within the frames to check the intended receiver. It will see that the message is intended for E, so D will remove the frames from its incoming queue and place them on its outgoing queue for forwarding.
8. D will check its routing table to determine the next hop the frames should take, and see that the E is its direct neighbor. Hence, D will forward the frames to E in a similar manner (again adjusting MAC addresses as necessary).
9. Finally, when E receives the frames in its incoming queue, it will notice that the message has reached its intended destination. E will unpack the message and pass it to its application.

Note that each activity whether it is sending, reception or forwarding must be logged by the corresponding client in the format described above.

To sum up, in case of a reception of frames, clients need to first check the intended receiver. If the message has reached its destination it can be unpacked, otherwise it should be forwarded. On the other hand, in case of sending, routing tables need to be consulted to determine the next hop for the frames.

# 3 Input Files and Parameters

There will be three input files (two for initializing the network and one with the commands), and three additional arguments specifying the maximum supported message size, outgoing and incoming port numbers for applications that will communicate respectively.

Clients and the network topology will be initialized through the first two input files.

## 3.1 Client Info File Format

```
number_of_clients
Client_ID<space>Client_IP_Address<space>Client_MAC_Address
```

For example, to initialize a network with five clients from our example scenario the input file will have the following content:

```
5
A 1.2.3.4 AAAAAAAAAA
B 4.3.2.1 BBBBBBBBBB
C 8.8.8.8 CCCCCCCCCC
D 9.9.9.9 DDDDDDDDDD
E 0.0.1.1 EEEEEEEEEE
```

## 3.2 Routing Tables File Format

```
Client_1_routing_table
-
Client_2_routing_table
-
...
-
Client_n_routing_table
```

Where each client_routing_table has the following format:

```
Destination_Client_ID<space>Client_ID_of_nearest_neighbor
```

For example, our sample use case scenario would have the following routing info file contents:

```
B B
C C
D B
E B
```

```
-
A A
C C
D D
E D
-
A A
B B
D B
E B
-
A B
B B
C B
E E
-
A D
B D
C D
D D
```

### 3.3 Commands File Format

Your works will be tested through the commands that will be given in the third input file. The commands will specify network activity your program must simulate (e.g. which client wants to communicate with whom and what message needs to be sent, what output is expected regarding network status, etc.). The file format is given below.

```
number_of_commands
COMMAND<space>command_parameter_1<space>command_parameter_2...
```

For a sample command file and the instructions how to execute commands, check the following section Network Status Check Commands.

### 3.4 Program Execution

When your program is executed, all of these parameters will be given as command-line arguments in the following order:

```
.\HUBBMNET clients.dat routing.dat commands.dat max_msg_size outgoing_port incoming_port
```

Note that file names may change, but the order will always stay the same. Your program must be able to accommodate this.

## 4 Network Status Check Commands

If we were to request a network status check in the example use case discussed above, here is a sample command input file that would accomplish that, and the expected output.

```
commands.dat content:
--------------------------
7
MESSAGE C E #A few small hops for frames, but a giant leap for this message.#
SHOW_FRAME_INFO C out 3
SHOW_Q_INFO C out
SHOW_Q_INFO C in
SHOW_FRAME_INFO C in 5
SEND C
PRINT_LOG D
```

Explanation of the commands is given below.

### 4.1 Client Communication

Suppose we want to simulate a message exchange between two clients. `MESSAGE` command will accomplish the preparation of the message for transmission; i.e. its fragmentation into frames. The format of MESSAGE command is:

`MESSAGE<space>sender_ID<space>receiver_ID<space>#message#`

This command will trigger the preparation of the message (given between two hash tags which should not be included in the message) for transmission by fragmenting it if necessary, packing it into frames, and placing the frames onto the sender's outgoing queue. The message should not be actually sent until the command SEND is received.

The output of our sample MESSAGE command should display the information about the message and all frames that were prepared for its transmission in the format given below.

```
--------------------------------------------------------------------------------
Command: MESSAGE C E #A few small hops for frames, but a giant leap for this message.#
--------------------------------------------------------------------------------
Message to be sent: A few small hops for frames, but a giant leap for this message.

Frame #1
Sender MAC address: CCCCCCCCCC, Receiver MAC address: BBBBBBBBBB
Sender IP address: 8.8.8.8, Receiver IP address: 0.0.1.1
Sender port number: 0706, Receiver port number: 0607
Sender ID: C, Receiver ID: E
Message chunk carried: A few small hops for
--------
Frame #2
Sender MAC address: CCCCCCCCCC, Receiver MAC address: BBBBBBBBBB
Sender IP address: 8.8.8.8, Receiver IP address: 0.0.1.1
Sender port number: 0706, Receiver port number: 0607
Sender ID: C, Receiver ID: E
Message chunk carried:  frames, but a giant
--------
Frame #3
Sender MAC address: CCCCCCCCCC, Receiver MAC address: BBBBBBBBBB
Sender IP address: 8.8.8.8, Receiver IP address: 0.0.1.1
Sender port number: 0706, Receiver port number: 0607
Sender ID: C, Receiver ID: E
Message chunk carried:  leap for this messa
--------
Frame #4
Sender MAC address: CCCCCCCCCC, Receiver MAC address: BBBBBBBBBB
Sender IP address: 8.8.8.8, Receiver IP address: 0.0.1.1
Sender port number: 0706, Receiver port number: 0607
Sender ID: C, Receiver ID: E
Message chunk carried: ge.
--------
```

Note that the receiver information in the Data Link/Physical layer (MAC address) does not match the receiver IP address and ID. The reason for this is the fact that the intended receiver of this message is client E. However, client E is not a direct neighbor of client C, so C must send this message to one of its direct neighbors which can eventually deliver the message to its final destination. This information is provided in C's routing table. C must check its routing table to see who is on the other side of the link that can deliver the message to E. In our sample use case, that next hop neighbor is client B; therefore, the Data Link/Physical layer information must be adjusted accordingly. Even though the final destination for the message is client E, message needs to take a next hop in the network over the C-B link, and this layer does not care about IP addresses or clients' IDs. It only cares about the MAC addresses of clients on both ends of the link over which frames will travel at that point.

## 4.2 Incoming/Outgoing Queue Status

Suppose we requested to inspect the contents of the Frame #3 on the C's Outgoing Queue just as the communicating application placed the request for sending the message. Command `SHOW_FRAME_INFO` will accomplish that. Its format is:

```
SHOW_FRAME_INFO<space>client_ID<space>which_queue<space>frame_number
```

The parameter which_queue can be either "in" or "out" specifying the outgoing and incoming queue respectively. The information that needs to be displayed when this command is placed is as follows:

```
--------------------------------
Command: SHOW_FRAME_INFO C out 3
--------------------------------
Current Frame #3 on the outgoing queue of client C
Carried Message: " leap for this messa"
Layer 0 info: Sender ID: C, Receiver ID: E
Layer 1 info: Sender port number: 0706, Receiver port number: 0607
Layer 2 info: Sender IP address: 8.8.8.8, Receiver IP address: 0.0.1.1
Layer 3 info: Sender MAC address: CCCCCCCCCC, Receiver MAC address: BBBBBBBBBB
Number of hops so far: 0
```

SHOW_Q_INFO command is used to inspect a client's queue in general. Its format is:

```
SHOW_Q_INFO<space>client_ID<space>which_queue
```

In our example, we want to inspect the C's outgoing queue status with the third command. The required output should be as follows:

```
---------------------------
Command: SHOW_Q_INFO C out
---------------------------
Client C Outgoing Queue Status
Current total number of frames: 4
```

Similarly, with the fourth command, the status of C's incoming queue would be displayed (note that C has not received anything at that point):

```
---------------------------
Command: SHOW_Q_INFO C in
---------------------------
Client C Incoming Queue Status
Current total number of frames: 0
```

With the fifth command we requested to inspect the contents of the Frame #5 on the C's incoming queue. The output should be:

```
--------------------------------
Command: SHOW_FRAME_INFO C in 5
--------------------------------
No such frame.
```

### 4.3 Sending Messages

The command SEND will cause the message that is currently placed on the sender's outgoing queue to be sent to it's intended receiver. Its format is:

```
SEND<space>client_ID
```

This command will trigger transmission and forwarding of the frames from the sender's outgoing queue until they reach their final destination, as explained in the sample use case scenario. The following network trace output should be produced:

```
----------------
Command: SEND C
----------------
A message received by client B, but intended for client E. Forwarding...
    Frame #1 MAC address change: New sender MAC BBBBBBBBBB, new receiver MAC DDDDDDDDDD
    Frame #2 MAC address change: New sender MAC BBBBBBBBBB, new receiver MAC DDDDDDDDDD
    Frame #3 MAC address change: New sender MAC BBBBBBBBBB, new receiver MAC DDDDDDDDDD
    Frame #4 MAC address change: New sender MAC BBBBBBBBBB, new receiver MAC DDDDDDDDDD
A message received by client D, but intended for client E. Forwarding...
    Frame #1 MAC address change: New sender MAC DDDDDDDDDD, new receiver MAC EEEEEEEEEE
    Frame #2 MAC address change: New sender MAC DDDDDDDDDD, new receiver MAC EEEEEEEEEE
    Frame #3 MAC address change: New sender MAC DDDDDDDDDD, new receiver MAC EEEEEEEEEE
    Frame #4 MAC address change: New sender MAC DDDDDDDDDD, new receiver MAC EEEEEEEEEE
A message received by client E from client C after a total of 3 hops.
Message: A few small hops for frames, but a giant leap for this message.
```

The most important thing in this step is setting the proper MAC addresses within the frames as they hop over each link. When the frames in our example use case started their journey from client C to client E, their first hop was over the C-B link. For this reason, the MAC addresses of the sender and receiver where CCCCCCCCCC and BBBBBBBBBB respectively. When B receives the frames, sees that they are actually intended for client E, and forwards them to client D, it first must unpack the frames, pop the the Data Link/Physical Layer from each frame, change the MAC address information to reflect the next hop over B-D link, by setting the MAC addresses of the sender and receiver of each frame to BBBBBBBBBB and DDDDDDDDDD respectively, and finally push the changed Data Link/Physical Layer back onto each frame.

Forwarding message to its intended destination may not be always possible. Suppose that in our example use case the routing table of D was as follows:

```
A B
B B
C B
E -
```

That is, when the message intended for client E reaches client D, D would not be able to find what the next hop should be. In that case an error message stating that the user is unreachable should also be displayed. The complete output of the SEND command in such case should be as follows:

```
----------------
Command: SEND C
----------------
A message received by client B, but intended for client E. Forwarding...
        Frame #1 MAC address change: New sender MAC BBBBBBBBBB, new receiver MAC DDDDDDDDDD
        Frame #2 MAC address change: New sender MAC BBBBBBBBBB, new receiver MAC DDDDDDDDDD
        Frame #3 MAC address change: New sender MAC BBBBBBBBBB, new receiver MAC DDDDDDDDDD
        Frame #4 MAC address change: New sender MAC BBBBBBBBBB, new receiver MAC DDDDDDDDDD
A message received by client D, but intended for client E. Forwarding...
Error: Unreachable destination. Packets are dropped after 2 hops!
```

The process of sending messages from the sender client until they reach the final destination (intended receiver) should be performed recursively. The message frames should travel through the network until they reach the intended client, or are dropped due to some unforeseen circumstances (e.g. unreachable user).

### 4.4 Client Logs

With the last command PRINT_LOG we requested to print the log of Client D upon the successful completion of communicating the message between C and E. The format of this command is:

```
PRINT_LOG<space>client_ID
```

Until that point, D has received and forwarded one message, so the output should be:

```
---------------------
Command: PRINT_LOG D
---------------------
Client D Logs:
--------------
Log Entry #1:
Timestamp: 2018-9-2 15:59:16
Message: A few small hops for frames, but a giant leap for this message.
Number of frames: 4
Number of hops: 2
Sender ID: C
Receiver ID: E
Activity: Message Received
Success: Yes
--------------
Log Entry #2:
Timestamp: 2018-9-2 15:59:17
Message: A few small hops for frames, but a giant leap for this message.
Number of frames: 4
Number of hops: 2
Sender ID: C
Receiver ID: E
Activity: Message Forwarded
Success: Yes
```

In the case where the frames were dropped because the next hop could not be determined due to the lack of routing information, the second log entry of client D would be:

```
Log Entry #2:
Timestamp: 2018-9-7 11:57:22
Message: A few small hops for frames, but a giant leap for this message.
Number of frames: 4
Number of hops: 2
Sender ID: C
Receiver ID: E
Activity: Message Forwarded
Success: No
```

In case the client does not have any log entries yet, nothing should be printed.

### 4.5 Invalid Commands

In case an invalid command is provided, your program should output the following message:

```
----------------------
Command: ORDER pizza
----------------------
Invalid command.
```

# 5 Report

You are expected to write a report about your program. For report format refer to:
ftp://ftp.cs.hacettepe.edu.tr/pub/dersler/genel/FormatForLabReports.doc

# 6 Grading Policy

Your work will be graded over a maximum of 100 points according to the grading policy given in the table below.

| Task | Points |
|---|---|
| Report | 10 |
| Command MESSAGE gives proper output | 20 |
| Command SHOW_FRAME_INFO gives proper output | 10 |
| Command SHOW_Q_INFO gives proper output | 5 |
| Command SEND gives proper output | 30 |
| Command PRINT_LOG gives proper output | 20 |
| Invalid Command output | 5 |
| **Total** | **100** |

# 7 Important Notes

**Implementing stack and queue structures:** Frames must be implemented as stacks (pop and push methods must be used to add layer information to and remove it from frames). Similarly, all queues must be implemented as queue data structures (insert and remove methods must be used for adding and removing frames to and from queues). Your code will be checked thoroughly to ensure this. Works that lack these implementations will be graded with **0 points** regardless of the output (check the grading policy).

**Usage of dynamic memory allocation:** Send and receive buffer capacities, as well as client logs, will have to be dynamically allocated and managed. Failing to use dynamic memory allocation in your code in order to avoid wasting memory resources will automatically lower your grade by **50 points**.

**How your work will be compiled:** Your code will be compiled on dev using gcc version 4.8.5. Hence, you are highly advised to compile and test your code on dev before you submit it. Furthermore, You must include a makefile with your source codes. Your code will be compiled and run using the following commands on dev:

```
$ make
$.\HUBBMNET clients.dat routing.dat commands.dat max_msg_size outgoing_port incoming_port
```

That said, use option -o in your makefile to specify the output file name for the executable which must be HUBBMNET.

Failing to include a makefile and making me compile your code manually will automatically lower your grade by **50 points** (check the grading policy).

**A valuable suggestion:** Use Valgrind (click here) to check for memory leaks and illegal reads/writes. This tool can help you a lot with memory related problems in your code. On dev you can type the following command:

```
$ valgrind --leak-check=yes --leak-check=full --show-leak-kinds=all ./HUBBMNET
```

**Submission format:** You must submit your work with the file hierarchy stated below:

<StudentID.zip>
→ <src.zip>
→ <report.pdf>

**Submitting:** You will use ouronline submission system to submit your work: https://submit.cs.hacettepe.edu.tr/. The deadline will end at 23:59 pm on the specified deadline date. No other submission method (email etc.) will be accepted.

**Other important notes:**

- Write readable code with understandable variable names.
- Save all your work until the assignment is graded.
- The assignment must be original, INDIVIDUAL work. Duplicate or very similar assignments are both going to be punished. General discussion of the problem is allowed, but DO NOT SHARE answers, algorithms or source codes.
- You can ask questions through the course's piazza page and you are supposed to be aware of everything discussed in the page.