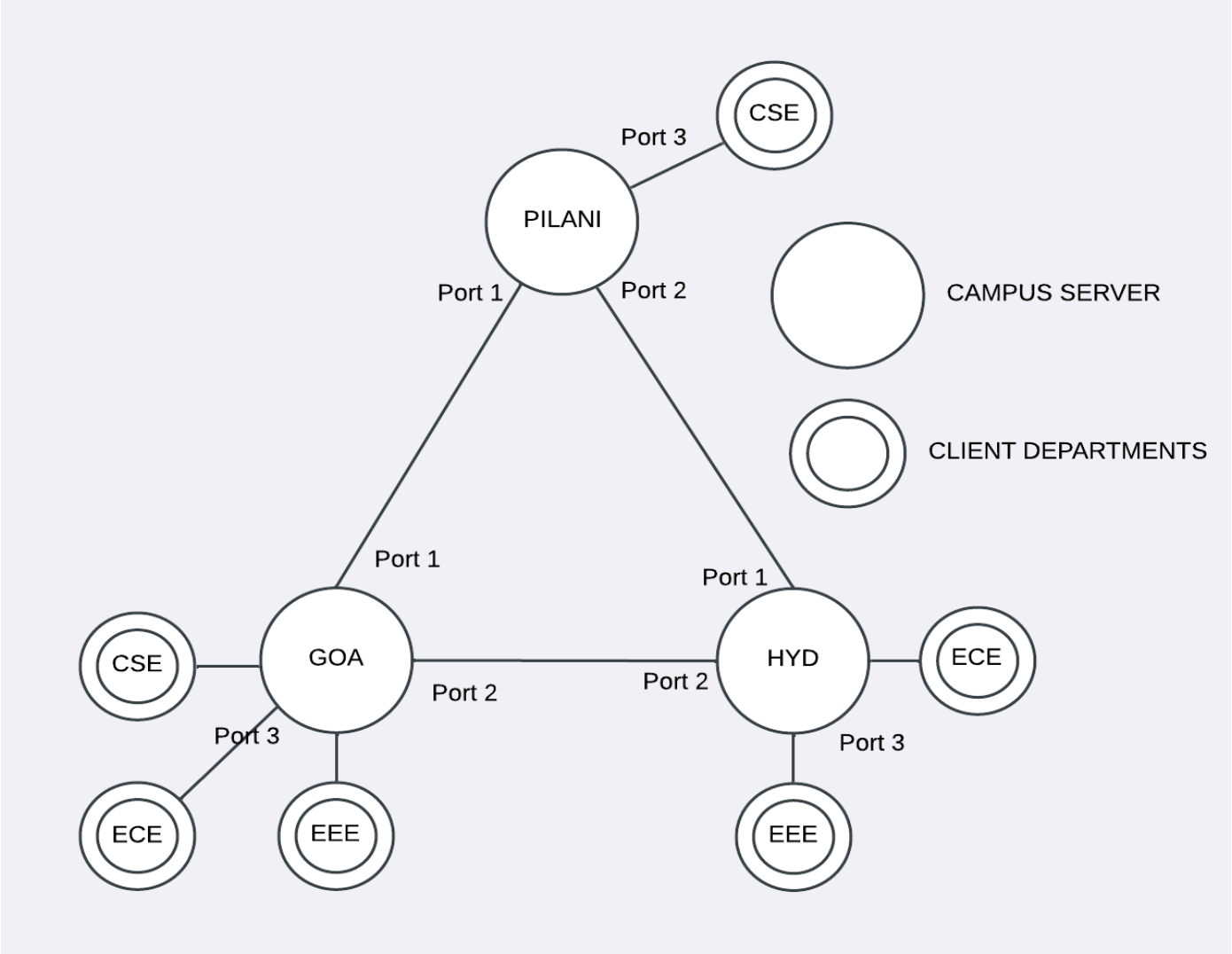
Lab 7 - Take home

In this lab, we will optimize the message sending process from clients/servers to other servers/clients. We will also consider a multi-server scenario with the following network architecture.



There are 3 campus servers (Goa, Pilani and Hyd) and multiple departments (say CS, EEE and so on) connect to each of the servers depending on the campus. The campus servers are connected to each other and can communicate directly. If any department wants to communicate with any other department, it has to go through 1 or more servers depending on which campus the destination department is located. (Say if CS Goa wants to send a message to EEE Hyd, then CS Goa client sends the message to Goa server, the Goa server sends the message to Hyd server and the Hyd server sends the message to EEEHyd client. In case CS

Goa wants to send a message to EEE Goa, then CS Goa client sends the message to Goa server, the Goa server sends the message to EEE Goa client).

For this network and subsequent communication, let's define a new protocol called **RaSh Protocol** (named after its creators).

The RaSh protocol has the following header field and format (every row represents 16 bits) :

Header for inter-campus communication

Version	Header length	Total length		
SRC Dept	DST Dept	CheckSum		
Hops	Type	ACK	SRC Campus	DST Campus

Header for intra-campus communication

Version	Header length	Total length
SRC Dept	DST Dept	CheckSum
Hops	Type	ACK

The struct for the packet has been provided in the header file.

The Header contains the following fields:

1. **Version(4 bits):** Version (current version is 2, bit value for same will be 0b0010)
2. **Header Length(4 bits):** Contains the total size of the header(in bytes). Header Length 6 bytes represents inter-campus communication, and 5 bytes represents intra campus communication.
3. **Total Length(1 bytes):** Contains the total size of the packet(in bytes)
4. **SRC Department(3 bits):** Represents the source department.
5. **DST Department(3 bits):** Represents the destination department. If this field is 0b000, it means that the packet is intended for that server and does not require forwarding. Positive value corresponds to some destination department.
6. **CheckSum(10 bits):** Contains the checksum similar to the internet checksum used in TCP and UDP
7. **Hops(3 bits):** Represents the number of hops so far. Incremented by the server(s).
8. **Type(3 bits):** If the DST Department is 0b000, i.e. packet is intended for the server, this field will represent what type of control packet it is:
    a. 0 = Name
    b. 1 = EXIT
    c. 2 = LIST
9. **ACK Field(2 bits):** Every packet from the client receives an ACK/NACK from the server. If the ACK field is 1, it's a success packet and payload will be empty. If the ACK field is

0, it is a NACK packet. The packet had to be dropped by the server for some reason, which will be mentioned in the payload. Any other value in ACK Field implies it is a normal packet containing a payload.

10. **SRC Campus(4 bits):** Represents the source campus. This field is present only when the communication required is inter-campus.
11. **DST Campus(4 bits):** Represents the destination campus. This field is present only when the communication required is inter-campus.

Types of packets:

1. **Unicast Packets:** If the DST Campus(if field present) and DST Dept represent a valid combination of a department in a specific campus, the packet has to be sent to the DST Dept of DST Campus. If the DST Campus field is not present, the packet has to be sent to the DST Dept of the same campus. If there is no valid DST Campus or DST Department, the packet is dropped by the server and a NACK containing the error message(payload) “Invalid Destination” is sent back to the department.

2. **Broadcast Packets:**
    a. University Level Broadcast: If the DST Campus field is 1111 (binary), the packet is broadcasted to all departments of all campuses irrespective of the DST Department field.
    b. Campus Level Broadcast: If the DST Campus field does not exist or is not 1111, and DST department is 111, then the packet will be sent to all departments of the same campus or DST campus respectively.

3. **Control Packets:** Packets that originate from a server or are meant for the server. In such cases the destination department field will be set as 0000. These kinds of packets will be used by the client to indicate to the server that it is disconnecting so that the server can reuse its 3-bit code. It will also be used for providing updated lists of client names and their respective codes to other servers as well as departments. When this list is sent from one server to another both the dest and src department fields will be set to 000. On the other hand when a server sends this information to a client only the src department field will be set to 0000.

The packet structs have to be forcefully converted into a byte stream using the serialize function before sending it. The argument to the send() function must be of type unsigned char*.

To be implemented:
1. Complete the serialize and deserialize functions for the packets. You can run/edit the tester.c file to check the implementation of those 2 functions. Naturally there will be different test cases for evaluation. Please note deserialize function won’t be evaluated until serialize code passes the test.

2. Implement server.c to be run as each server. The server should be able to serve multiple departments simultaneously using a multithreaded solution. Each instance of server.c will take 5 command line arguments. They are:

- Arg1 - IP address which will be common for all servers
- Arg2 - Port Number 1 (Port Explanations given below as part of server setup)
- Arg3 - Port Number 2
- Arg4 - Port Number 3
- Arg5 - Campus Identification. For all purposes in the code, as well as populating campus fields in the packets, the following mapping should be used.
    - ID of Pilani Campus = 1
    - ID of Goa campus = 2
    - ID of Hyderabad campus = 3
    - The binary representation of campus ID will be used to populate the SRC and DST campus fields in the header.

2. Server Setup: For setting up the connections between the different campus servers please follow the instructions given below. Here Port 1 and Port 2 represent the arguments mentioned above **for the specific process**. The values will vary for different processes. Ensure you follow the sequence of events exactly as mentioned below.

- The Pilani server will first start executing and listen on its Port 1 for Goa Server and Port 2 for Hyd Server.
- The Goa server will then start executing and connect to the Pilani server through Port 1 and listen on Port 2 for Hyd Server to join.
- The Hyd server will start executing only after the other 2 servers start and connect to Pilani through Port 1 and Goa through Port 2.
- Note: Value wise, Port 1 argument of Pilani Server will match Port 1 argument of Goa Server, Port 2 argument of Pilani server will match Port 1 argument of Hyd Server and Port 2 argument of Goa server will match Port 2 argument of Hyd Server.
- Port 3 will be used for connecting with their individual departments.
- Ensure connections are established properly by sending packets. Exact format specified in the third point of Detailed Example. **(1 Mark)**

3. Implement department.c for the clients. The clients should be able to send and receive data concurrently. This file takes 4 arguments.

- Arg1 - Ip of the server
- Arg2 - port number of the server(Should match with Port 3 of whatever campus they want to connect with)
- Arg3 - Campus identification (1,2,3 mentioned above) of the server
- Arg4 - Name of the client department for example “CSE”

4. When a new client connects to a server it has to send its department name to the server it connects to. Specifically, it will send a control packet with the payload as "<department_name>\n" which is taken as a command line argument.

- Since it has not yet been assigned a department label yet, it will populate **SRC Dept as 111**.
- The server internally stores the information about the name and assigns the first available 3-bit number starting from 001 to 110 to represent the client, similar to how client id was assigned in the previous lab.
- This number will be used to populate the SRC and DST Departments fields in the header in all subsequent interactions with the department.
- For example, if a computer from the CSE department connects to the Goa server, it first sends a "CSE\n" as the payload to the server, and the server internally maps it to 0b001 (1 in binary).
- If a new client joins, the server will map it to the code 0b010 (binary).
- Meanwhile, if the CSE department computer disconnects from the server and a new department joins, the lowest available code (001 binary) will be assigned again to this new department.
- The client gets to know the assigned department number from the DST Dept field in the subsequent ACK it receives from the server. After the ACK is sent, the server sends a LIST type control packet to the client which will contain the list of all departments connected to all servers.
- The format of the the payload when server sends the LIST command is “<campus_name1>;<dept_name1>=<dept_id1>;<dept_name2>=<dept_id2>;...|<campus_name2>;<dept_name1>....|<campus_name3>;.....”
- After the ACK is sent and the list is sent to the new client, the server proceeds to send a copy of its updated list of clients to all other clients (except the new client) and the other servers which will forward it to their clients using a LIST control packet (this updated list will only contain the clients associated with the server instead of clients associated with all servers).
- At any time only 1 message can be sent by the client and they have to wait until they get the ACK before sending the next message.

For example in the above case when a new department-lets say MECH- joins the Goa server it sends a control packet with payload “MECH\n” to the Goa server which responds with an ACK packet followed by a LIST control packet which has its payload set to “G;CSE=1;ECE=2;EEE=3;MECH=4;|P;CSE=1;EEE=2;|H;ECE=1;EEE=2;”.

After this it sends another LIST control packet containing “CSE=1;ECE=2;EEE=3;MECH=4;” as its payload is sent to the rest of the clients and the other servers.

Note the order in which the campuses and their departments do not matter.

5. The structure of the header is described above. You can generate a packet using the given packet generator function or by your own means. The header is followed by the payload, similar to what you have learnt in this course for TCP packets, UDP packets, and IP packets.

6. As specified previously, there are 3 types of packets: unicast, broadcast, and control packets.

7. User Inputs:

- For unicast packets, the client has to enter the name of the destination campus server, destination department name, and the payload to be sent in the format ("1.<dest_campus>:<dest_department>:<payload>\n" ("\n" automatically added when enter is pressed). If the packet has to be sent within the same campus, the format is "2.<dest_department>:<payload>\n". Examples of this packet type are provided in the detailed example section.

- For broadcast packets, type "3.<payload>\n" for campus level broadcasting, where the packet is sent to all the clients connected to the same campus server. Type "4.<payload>\n" for university level broadcasting, where the packet is sent to all departments across all campuses.

- Control packets:
  - Type "5.EXIT" for exiting the client. The client must be properly terminated. The payload will be empty in this case.
  - "LIST" is not a valid command which can be entered by the user to get names on demand. It is only used by servers when sending updated user lists and must be automatically generated by them. The list will be sent in the format: "<dept_name1>=<dept_id1>;<dept_name2>=<dept_id2>;..." and will be sent as the payload of a packet (headers must be added by you)
  - For example when EEE client connects to the GOA server (refer to the above figure) the server sends a packet with payload "CSE=1;ECE=2;EEE=3;" to all connected clients, and other servers. See #8 for details.

- **Do not send these directly to the server. Extract the relevant data, populate a new packet, serialize the packet into a byte stream and send the byte stream.**

- Note the number in the beginning must be used to identify the format of the input and then use it to create the packet containing the header and the payload to the server.

8. Whenever a new client joins or an existing client disconnects from a server, the server broadcasts the list of department names and their mappings to all of its clients (except the newly joined client as it already has a list of all clients of all servers that was sent to it in the ACK for name control packet)and all the other servers, which in turn forward it to their clients(This can be sent 1 second after the ACK is sent for that particular client). The client is responsible and updating the list of clients connected to each server and their client IDs

9. Every communication will receive an ACK/NACK with the ACK Field set/reset respectively. All other packets will have some other value.

10. In each case, the header fields will have different configurations based on the packet type and communication requirements.

Note:

1. The connections between the servers are single channel, i.e., only one packet can be sent at a time between two campus servers. Naturally for the server thread servicing a particular department, it may receive multiple packets either from its own campus/other campus. To overcome this issue, the ideal solution will be to implement queues so that all packets are serviced properly. However, for the sake of simplicity, assume **there will be a discernible time delay between all packets sent through the network**.
2. **All processes must print every packet they receive to the terminal in the specified format.**

Functions provided:

1. struct packet* generatePacket(...): You can use this function to generate a packet with the required parameters. Check the function signature in the code for the actual order of parameters. If you want to write a more customized function which automatically calculates total length based on the data or whatnot, please feel free to add such an implementation. Just make sure to use a different name for the function.
2. void printPacket(struct packet* p): This will print a packet 'p' to the terminal. **Whenever any output is expected to be printed to the terminal or to a file, this format is expected to be followed.**

Functions to implement and Mark Distribution:

1. unsigned char* serialize(struct packet* p) : Function for serializing the packet **(0.5 mark)**
2. struct packet *deserialize(unsigned char* buffer): Function for deserializing the packet **(0.5 mark)**
3. struct packet* generateUnicastPacket(char* input): Whenever the user enters an input in the form of "1:" or "2:"(i.e., starting with 1 or 2), this function is expected to be called with the user input as the argument. The user input should be parsed through and the relevant data extracted. A packet should be generated by populating the fields with the extracted data and returned. **(1 mark)**
4. struct packet* generateBroadcastPacket(char* input): Whenever the user enters an input starting with "3." or "4.". Same explanation as above. **(0.5 mark)**
5. struct packet* generateControlPacket(char* input): Whenever the user enters an input starting with "5.". Same explanation as above. **(0.5 mark)**
6. Server setup(Part 3 of Processes Setup in Detailed Example): **(1 Mark)**
7. The terminal output of packets received by the processes: **(2 Marks)**

Detailed example:

Processes Setup:

1. Run the command `./server 127.0.0.1 1111 2222 3333 1` to start the Pilani Server. 1111 will be the port with which connection to Goa would be established, 2222 to Hyderabad and 3333 for the departments of Pilani itself.

2. Then start the Goa and Hyderabad servers with `./server 127.0.0.1 1111 4444 5555 2` and `./server 127.0.0.1 2222 4444 6666 3` respectively. Notice how the required ports match.

3. At this point, the connection between the servers must be properly initialized. Ensure that by sending 1 packet between all the servers. Pilani sends a packet to Goa and Hyd, Goa and Hyd respond to the above packet by each sending a packet to Pilani. Goa sends a packet to Hyd, Hyd responds with a packet. The SRC and DST Campuses must be properly populated. All packets must be printed to the terminal only by the receiving server.

4. Now, similar to previous labs, start instances of the client to be connected to each server. `./client 127.0.0.1 3333 1 CSE` should start a CS department process to connect to the Pilani server and so on. The servers should naturally handle this using multithreading as in previous labs.

5. The clients should send messages based on user inputs in the format mentioned in Part 7 of “To be implemented”. All receiving servers/clients should print the packet in the mentioned format. As mentioned in the note above, packets will be sent with significant delay so the printing order won’t be affected.

Client Connections:

Let's say we have three departments connected to the Pilani campus server: CSE (001), ECE (010), and EEE (011). On the Goa campus server, we have two departments: MECH (001) and CHEM (010).

1. Unicast packet:
If the CSE department in Pilani wants to send a message "Hello from CSE Pilani!" to the MECH department in Goa, the user input would be: "1.G:MECH:Hello from CSE Pilani!\n"

The client will create a packet with:
- DST Campus: 010 (Goa)
- DST Department: 001 (MECH)
- Other header fields populated as explained previously
- Payload: "Hello from CSE Pilani!"

The Pilani server will forward this to the Goa server, which will deliver it to the MECH client. The MECH client will print: "P:CSE:Hello from CSE Pilani!"

If the CSE department in Pilani wants to send a message “Hello from CSE!” to the ECE department in Pilani, the user would input: “2.ECE:Hello from CSE!\n”

The client will create a packet with:
- DST Campus: 001 (Pilani)
- DST Department: 010 (ECE)
- Other header fields populated as explained previously
- Payload “Hello from CSE!”

2. Campus Broadcast:
If ECE in Pilani wants to broadcast "Greetings from ECE!" to all departments in Pilani, the input would be: "3.Greetings from ECE!\n"

The client will create a packet with:
- DST Department: 111 (campus broadcast)
- Payload: "Greetings from ECE!"

The Pilani server will send this to CSE and EEE. They will print "P:ECE:Greetings from ECE!"

3. University Broadcast:
If CHEM in Goa wants to broadcast "Hello all from CHEM Goa!" to the entire university, the input would be: "4.Hello all from CHEM Goa!\n"

The client will create a packet with:
- DST Campus: 1111 (university broadcast)
- Payload: "Hello all from CHEM Goa!"

The Goa server will send this to Pilani and Hyderabad servers, which will forward to all their connected departments. Every department will print "G:CHEM:Hello all from CHEM Goa!"

4. Control Packets:
When a new client EEE connects to Pilani, it sends a control packet with payload "EEE\n". The Pilani server assigns its department ID 011 and sends an ACK to the EEE client followed by a LIST control packet with payload “P;CSE=1;ECE=2;EEE=3;|G;MECH=1;CHEM=2;” (as Hyderabad doesn't have any clients connected it is omitted). Then the Pilani server sends an updated list "CSE=1;ECE=2;EEE=3;" to all Pilani departments and other campus servers. The Goa and Hyderabad servers forward this list to their departments.

When the EEE client wants to disconnect, it sends "5.EXIT" to the Pilani server. The server removes EEE from its list and sends an updated "CSE=1;ECE=2;" to all Pilani departments and other campus servers to forward.