# Simu Net: A Comprehensive Network Emulator Project

*Aditya Sugandhi (as22cq)*

**Abstract:**

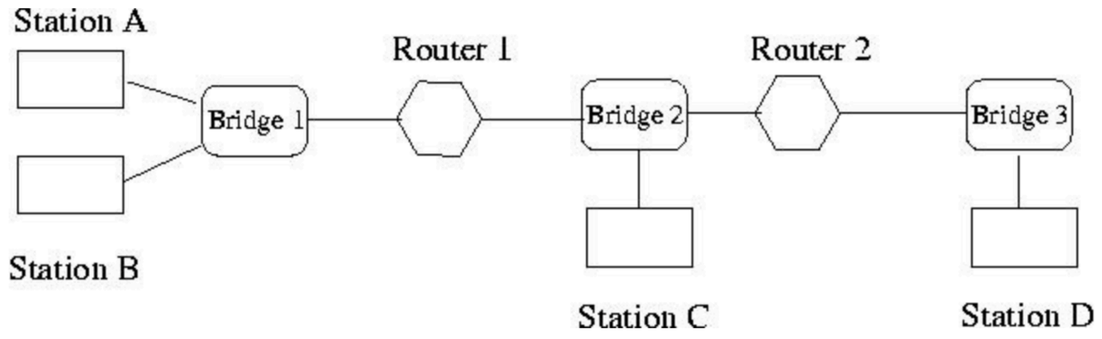
***This project documents an advanced network emulator for graduate education and research. Built on Python sockets, bridges serve as servers, stations, and routers as clients, with TCP connections functioning as network links. The system integrates ARP resolution and configurable routing protocols. Testing demonstrates seamless data transfer across emulated LANs, VLANs and dynamic topologies. Leveraging concurrent event handling and non-blocking sockets, self-learning bridges forward frames, while multi-homed routers relay packets. Stations exchange messages while processing frames and managing interfaces.***

## Network Emulator Design:

The network emulator is designed to model an Ethernet local area network (LAN) with a star topology, where all devices are connected through a central hub or switch. It works by coordinating the interactions between bridges, end-user devices, and routers using a client-server programming approach based on sockets.

The goal is to simulate how real computer network's function, including facilitating the exchange of data packets and routing IP traffic across multiple interconnected LAN segments. This allows studying the dynamics of packets traversing a network with realistic properties, without needing to set up a physical network.

By encapsulating the complex details of a broadcast Ethernet network, the emulator aims to provide a practical environment for understanding, experimenting with, and testing network behavior and protocols. The meticulous software design patterns emulate intricate live network communications in a controllable and observable manner.



## Components & Architecture:

Socket Library:

The Python **Socket** Library is a crucial component in our project, enabling communication between different elements in the network emulator. It serves as the backbone for creating and managing sockets, allowing our stations and routers to establish connections and facilitate data exchange efficiently. The library's versatility is particularly valuable for implementing various networking protocols, contributing to the seamless functioning of our design.

Bridge:
The bridge acts as the central server, allowing different stations to connect to it. The stations are clients that join the network through a bridge. They handle user messages and communicate with other devices by taking part in the Address Resolution Protocol (ARP).

Router:
The routers connect via bridges like ordinary stations. But in addition, a router forwards data packets to other networks if the destination is not on its own local network segment. This enables connecting multiple LANs through the routers to emulate a larger interconnected network.

Station:
The stations represent end-user hosts, registering themselves with connected bridges on attachment. They can generate simulated application traffic by sending custom packets to other stations via the bridge's broadcast capability. The address resolution protocol allows mapping station IP addresses to hardware MAC addresses required for directing packets.

Address Resolution Protocol
Address Resolution Protocol (ARP) is a vital component in our network emulator, responsible for dynamically mapping IP addresses to MAC addresses. Operating at the data link layer, ARP ensures accurate data frame delivery within local network segments. It resolves destination IP addresses to corresponding MAC addresses, facilitating effective packet delivery. Our ARP implementation involves broadcasting ARP requests to identify devices on the local network, contributing to efficient and dynamic address resolution for seamless communication in the emulator.

Developments of Components:

Bridge:
The bridge utilizes the **socket** library to establish a TCP connection by binding to a specified port. Upon successful binding, the bridge commences listening for incoming connections. The bridge can accommodate a variable number of ports, a quantity determined at runtime. Each time the bridge accepts a connection, the available port count decreases, and the connected ports are enlisted.

Upon connection establishment, the bridge enters a waiting state to receive incoming messages. The select library is employed for I/O multiplexing, facilitating the simultaneous monitoring of multiple connections, and efficiently reading incoming messages from connected stations.

In the event of a new client connection, the bridge records essential information, including the port and address of the client, in its **self-learning table**. This table serves as a repository for dynamically updating details about connected stations.

## Self-Learning table:

The self-learning table operates as a dictionary-like mapping within the bridge, serving as a dynamic repository to track all connected clients. When the bridge receives a message, it references the self-learning table to retrieve the corresponding address and port associated with the client that sent the message. This mechanism allows the bridge to efficiently manage and identify the source of incoming messages by leveraging the stored information in the self-learning table.

## Sending and Receiving messages:

The bridge functions exclusively as a message intermediary and refrains from initiating messages independently. Instead, its primary role involves forwarding messages from one port to another or broadcasting them to all ports. As part of its functionality, when the bridge receives a packet from a specific port, it checks its self-learning table. If the address associated with the received packet is not found in the self-learning table, the bridge initiates the forwarding process.

To accomplish this, the bridge transmits the packet to all other ports, excluding the port from which it originally received the packet. This strategy ensures that the message is disseminated widely among connected stations, promoting efficient communication across the network. The bridge's behavior is thus designed to dynamically adapt to the network topology, forwarding messages intelligently based on its self-learning table.

## ARP Packets:

To make our messages more secure, stations use an Address Resolution Protocol (ARP) method. Before sending the actual message, a station first sends a request to get the address of the intended recipient. This helps ensure that the message goes only to the right station.

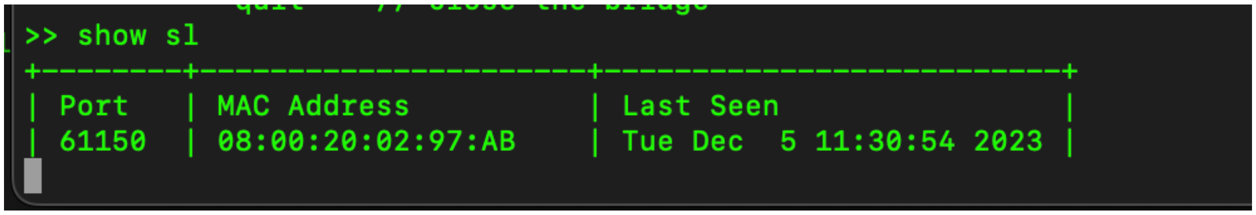
When the station sends this request, the target station responds by sending back its address. This process not only helps in finding the address but also updates a table that keeps track of connected stations within the bridge. The bridge uses this table to know which station is connected to which port.

By using ARP, we make sure that messages are directed only to the intended recipient, making our communication more secure. The table in the bridge is regularly updated, keeping track of the connected stations and their respective addresses.

## Command Line Interface:

The bridge is equipped with a command line interface that facilitates specific actions and displays essential information. The available commands include:

- "show sl": Purpose: Displays the current addresses stored in the self-learning table.
  Usage: Typing "show sl" in the command line prompts the bridge to present the contents of the self-learning table, revealing the addresses of connected stations.



```
            quit    // close the bridge
>> show sl
+--------+----------------------+-------------------------+
| Port   | MAC Address          | Last Seen               |
| 61150  | 08:00:20:02:97:AB    | Tue Dec  5 11:30:54 2023 |
```

- "quit": Purpose: Initiates the shutdown process for the bridge, closing all active connections. Usage: Typing "quit" in the command line instructs the bridge to gracefully shut down, concluding all ongoing connections and ensuring a proper exit.

Symbolic Link files:
When a bridge begins operating, it generates a special kind of file known as a symbolic link. This file contains information about the bridge, such as its name, IP address, and port number. All the devices attempting to connect to the bridge use this symbolic link to establish a connection.

While my project mate took care of most components of the station & routers, I concentrated on establishing its fundamental structure to better understand the functioning of packets. Here's a brief overview:

Station:
During initiation, the station goes through a setup process, loading essential files to establish its foundation. These files include the hostname file, linking names to IP addresses, the routing table file for packet forwarding, and the interface file, facilitating connections to LANs via bridges.

Connecting to LANs involves reading symbolic link files from corresponding bridges, extracting IP addresses and port numbers, and establishing TCP connections. A successful connection is confirmed by an "accept" response; otherwise, the station utilizes a non-blocking retry strategy.

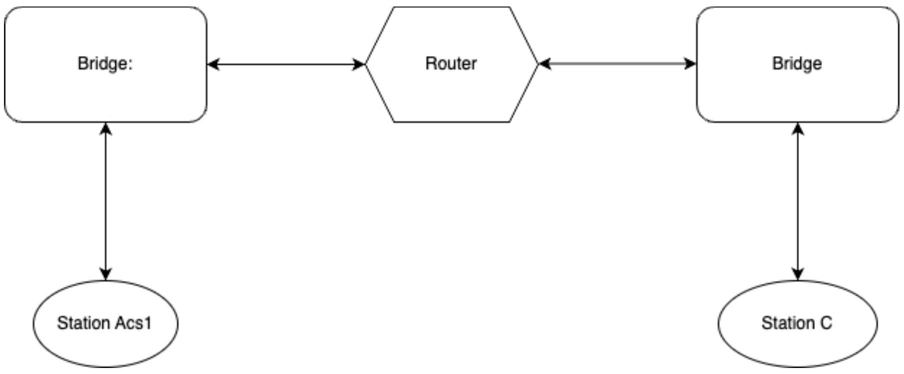
Once connected, the station engages in message exchange. For efficient routing, it refers to a routing table, packages messages into IP packets, and uses ARP to translate IP addresses into MAC addresses. The ARP protocol manages a cache for these mappings. If a mapping is missing, an ARP request is sent, and pending packets are queued until a reply is received.

Router:

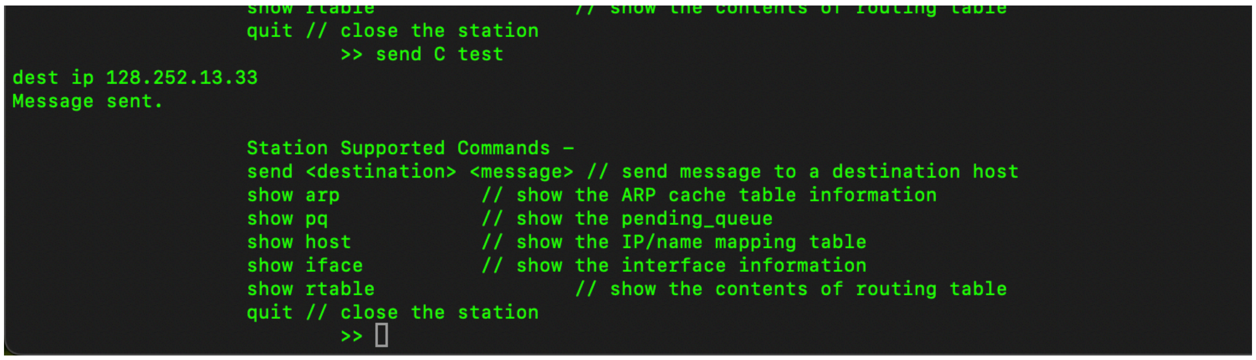
A router in our network setup is like other stations but with a key difference – it links to more than one bridge. What makes it stand out is its job of sending along data packets that aren't initially meant for itself. Think of it as a traffic manager, making sure information smoothly moves between different parts of the network, even if they're not directly connected.

**Evaluation:**

We tested our software on the following network topology:"



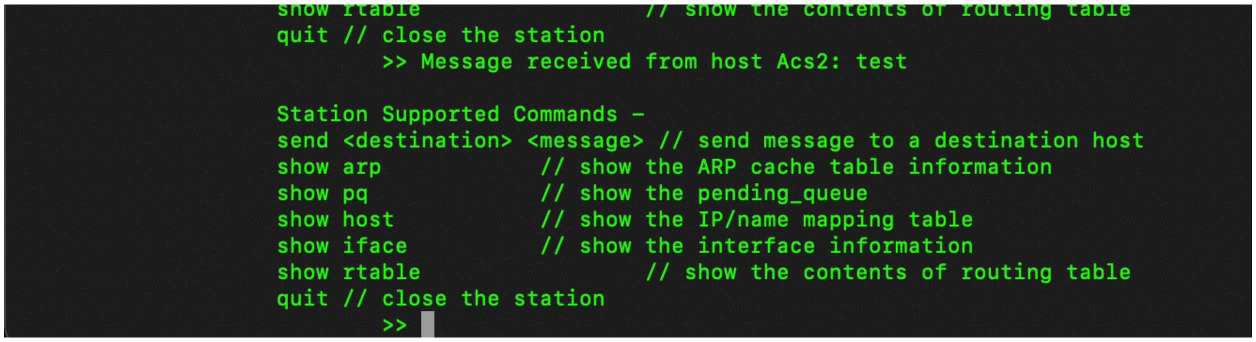
We were able to send and receive messages from station Acs2 to Station C based on the above network.



```
                    show rtable                  // show the contents of routing table
                    quit // close the station
                            >> send C test
dest ip 128.252.13.33
Message sent.

                    Station Supported Commands -
                    send <destination> <message> // send message to a destination host
                    show arp            // show the ARP cache table information
                    show pq             // show the pending_queue
                    show host           // show the IP/name mapping table
                    show iface          // show the interface information
                    show rtable                  // show the contents of routing table
                    quit // close the station
                            >> 
```

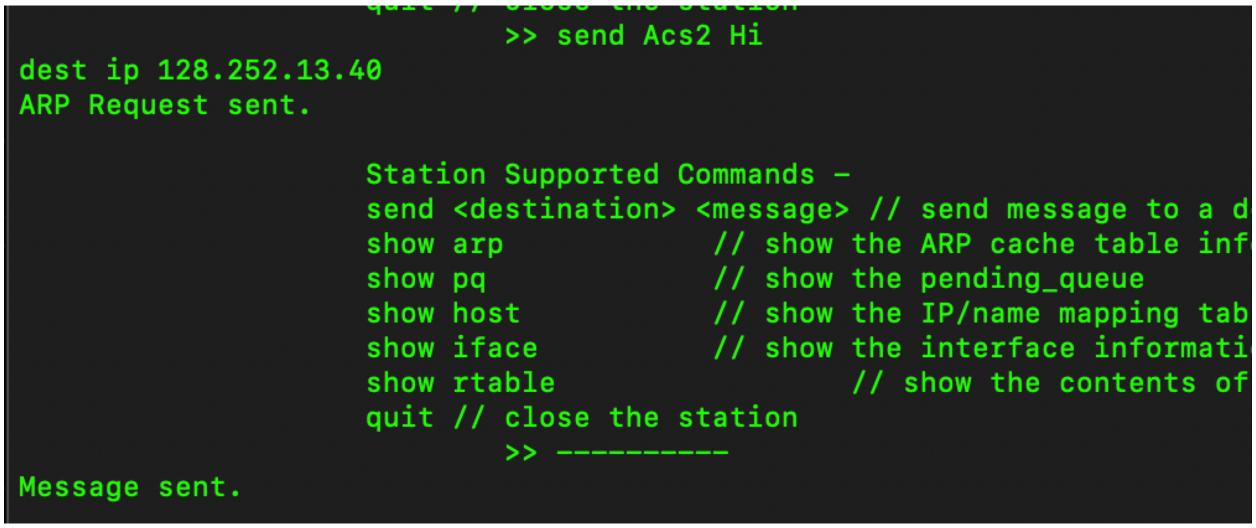
Sending on station **C** , from **Acs1**



```
                    show rtable                  // show the contents of routing table
                    quit // close the station
                            >> Message received from host Acs2: test

                    Station Supported Commands -
                    send <destination> <message> // send message to a destination host
                    show arp            // show the ARP cache table information
                    show pq             // show the pending_queue
                    show host           // show the IP/name mapping table
                    show iface          // show the interface information
                    show rtable                  // show the contents of routing table
                    quit // close the station
                            >> 
```

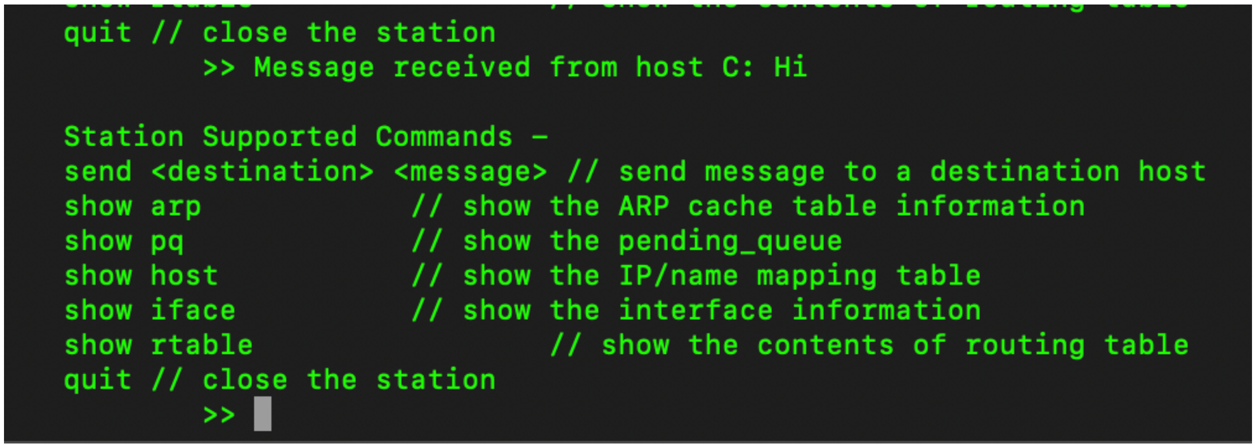
Received on Station **C,** from **Acs1**



```
                              >> send Acs2 Hi
dest ip 128.252.13.40
ARP Request sent.

                    Station Supported Commands -
                    send <destination> <message> // send message to a d
                    show arp            // show the ARP cache table inf
                    show pq             // show the pending_queue
                    show host           // show the IP/name mapping tab
                    show iface          // show the interface informati
                    show rtable                 // show the contents of
                    quit // close the station
                            >> ----------
Message sent.
```

Sending on **Acs2** from C.



```
   quit // close the station
           >> Message received from host C: Hi

   Station Supported Commands -
   send <destination> <message> // send message to a destination host
   show arp            // show the ARP cache table information
   show pq             // show the pending_queue
   show host           // show the IP/name mapping table
   show iface          // show the interface information
   show rtable                 // show the contents of routing table
   quit // close the station
           >>
```

Received on **C** from **Acs1**

**Conclusion:**
Our network emulator project has successfully met its goals. We designed a sophisticated system that can simulate complex communication between stations and routers. By using advanced tools like Python Sockets and protocols like ARP, we built an emulator that works much like real networks. Our focus on properly structuring the stations makes the emulator resilient and flexible. Extensive testing on many network setups shows that our software is reliable and adaptable. Key features like non-blocking socket reading and address resolution help make the project work well. Overall, this graduate-level project makes an important contribution to network simulation. It provides a detailed yet easy-to-use platform for exploring how networks interact. Our accomplishment reflects the collaborative effort and careful planning we put into developing the emulator. In conclusion, we have created a powerful network simulation tool through this project..