EmuLinks:Linking Networks in Software

Soumya Tejaswi Vadlamani

Abstract

Developers struggle a lot to create a controlled environment for testing networking applications,leading to difficulties in replicating real-world network behaviors and conditions.This lack of emulation could result in unreliable and unpredictable outcomes during application development and testing phases.So,This project centers on developing a network emulator that replicates the complex networking behaviors within a controlled environment.The emulator leverages socket-based networking and imitates real-world network functionalities.The network model simulates broadcast Ethernet LANs employing bridges that transmit data frames among stations within LANs. A star-wired LAN topology connects stations directly to a central bridge, enabling frame forwarding. IP routers interconnect LANs for inter-station communication, employing a hop-by-hop datagram delivery model based on forwarding tables. The emulation design implements these concepts via TCP socket connections, where bridges act as servers and stations as clients, establishing connections for communication.This comprehensive endeavor aims to impart a robust understanding of network emulation and software-defined networking principles, making significant contributions to both practical networking knowledge and collaborative software development skills.

Index Terms:- Network Emulator,hop-by-hop datagram delivery model,LAN

Introduction

Developers encounter significant hurdles when endeavoring to construct controlled environments specifically tailored for the efficient testing of networking applications. The intricate nature of replicating genuine network behaviors and conditions within a testing framework presents a formidable challenge. This complexity often leads to struggles in accurately reproducing the diverse and dynamic scenarios encountered within real-world network environments. The absence of a robust emulation system compounds these challenges, impeding developers' ability to ensure that their testing environment adequately mirrors the complexities inherent in genuine networks. Consequently, the absence of emulation capabilities jeopardizes the reliability and predictability of outcomes during crucial phases of application development and testing.

The absence of emulation capabilities in testing environments introduces a slew of issues that impede comprehensive evaluation and validation of networking applications. When developers lack the ability to replicate genuine network behaviors, they have difficulty thoroughly examining and comprehending their applications' behavior under varying network conditions. Because of this limitation, it is difficult to precisely identify potential vulnerabilities or performance bottlenecks that may arise during deployment in real-world networking settings. As a result, the lack of a controlled emulation environment limits the thorough testing required for robust and secure networking applications, leaving them vulnerable to flaws or deficiencies that could compromise their reliability and functionality.

Furthermore, the inability to simulate real-world network scenarios limits developers' ability to address problems that may arise after deployment. The scope of comprehensive testing remains constrained due to the inability to accurately replicate the dynamic nature of networks, leaving potential flaws undetected until applications are operational in live environments. This lack of foresight prior to deployment

increases the likelihood of encountering unexpected challenges or malfunctions, potentially resulting in operational disruptions necessitating reactive measures to address these issues. post-deployment

To address these issues, the project focuses on creating an advanced network emulator that is specifically designed to replicate complex networking behaviors in a meticulously controlled environment. This emulator's core functionality is socket-based networking, which faithfully simulates the intricate functionalities inherent in real-world network operations. This emulator's network model intricately mimics the dynamics of broadcast Ethernet Local Area Networks (LANs) by utilizing interconnected bridges that facilitate data frame transmission among various stations within the LANs. This configuration is similar to a star-wired LAN topology in that it fosters direct and efficient connections between individual stations and a central bridge, allowing for seamless frame forwarding and communication within the network.Furthermore, the emulation incorporates IP routers' pivotal role in connecting different LANs, enabling inter-station communication via a sophisticated hop-by-hop datagram delivery model utilizing forwarding tables.

The meticulous design of the emulation, which was created to replicate real-world networking complexities, is based on the fundamental principles of TCP socket connections. Bridges, which act as servers, and stations, which act as clients, create a sophisticated communication framework that is critical to the emulation process. This comprehensive approach allows for the faithful replication of various network functions, such as data frame transmission within LANs via bridges and inter-LAN communication via IP routers. The project aims to provide an in-depth understanding of network emulation and software-defined networking principles through this comprehensive approach, not only enhancing practical networking expertise but also fostering collaborative software development skills, which are critical in addressing modern networking challenges.

The creation of a network emulator represents a watershed moment in networking technology, aiming to replicate the intricate functionalities of real networks within a controlled virtual environment. This emulator makes use of socket-based networking capabilities to provide a foundation for communication and interaction among various components. It mimics the behaviors and functions of real-world network operations through meticulous emulation. Emulation entails simulating broadcast Ethernet Local Area Networks (LANs) with bridges, which are critical devices that allow for the seamless transmission of data frames among interconnected stations. This emulation, which uses a star-wired LAN topology, allows for efficient frame forwarding among network components. Furthermore, the emulation includes IP routers to connect different LANs, allowing inter-station communication through a meticulously designed hop-by-hop datagram delivery model based on forwarding tables.

TCP socket connections established between bridges and stations are central to this emulation framework, serving as the foundation for communication paradigms within the simulated network. This network emulator offers a comprehensive platform that has been painstakingly designed to replicate various network functions, ensuring a complete emulation experience. The project's ultimate goal is to provide aspiring technologists and engineers with a comprehensive understanding of network emulation and software-defined networking principles, thereby significantly contributing to the development of practical networking expertise and fostering collaborative software development skills.

Network Topology Overview

A network emulator is an important tool in the field of computer networking because it provides a controlled environment for simulating the behavior, functionalities, and conditions of computer networks. It works as a virtual platform that mimics the complexities and dynamics of real-world networks,

allowing researchers, developers, and network engineers to study, analyze, and test various networking applications, protocols, and systems without requiring physical hardware infrastructure. A network emulator creates an environment in which different networking components such as routers, switches, bridges, and stations (devices) interact and communicate as they would in a real network by simulating these components and their interactions.

The primary goal of a network emulator is to accurately replicate the behaviors and functionalities of real-world networks. It provides a controlled environment for researchers to investigate the performance, reliability, scalability, security, and robustness of networking applications and protocols under various network conditions. This emulation capability is particularly useful for testing applications or protocols in scenarios that are impossible or impractical to replicate in a physical network configuration, such as high network loads, failure conditions, or security vulnerabilities.

The ability of a network emulator to generate repeatable and customizable network environments is one of its most important features. Researchers can configure and manipulate network parameters such as bandwidth, latency, packet loss, and network topology to simulate various network conditions. This feature allows for the controlled execution of extensive experiments, allowing for the evaluation of application behavior in various networking scenarios. Network emulators also enable the isolation of specific network components or segments for in-depth analysis, making them useful tools for networking system debugging and performance tuning.

Furthermore, network emulators are critical in the development and validation of new networking technologies, protocols, and algorithms. Researchers and developers can test the feasibility and efficiency of novel networking concepts in a simulated environment before deploying them in real-world settings. This pre-deployment evaluation aids in the identification of potential flaws, the optimization of performance, and the assurance of the stability and compatibility of new networking solutions.

Network emulators are commonly used in research in many fields, including computer science, telecommunications, and engineering. They are critical platforms for conducting controlled and reproducible experiments, validating theoretical models, and analyzing networking system behavior. Researchers use network emulators to put new ideas to the test, conduct empirical studies, and contribute new insights and advancements to the field of computer networking.

Network emulators enable the isolation and analysis of specific network components or segments. This capability aids in-depth analysis, debugging, and performance tuning of networking systems by focusing on individual network elements.Network emulators are essential tools for testing and developing new networking technologies, protocols, and algorithms. They allow researchers and developers to evaluate the feasibility, efficiency, and compatibility of novel networking concepts before implementing them in real-world scenarios. This pre-deployment testing aids in the identification of potential flaws, performance optimization, and stability maintenance.

Network emulators are eventually useful tools for simulating network environments that mimic the functionalities and conditions of real-world networks. Their application spans networking application research, development, testing, and validation, providing a controlled and reproducible platform for experimentation and analysis in computer networking.

The Design

In this section,we briefly describe the few key aspects of the network model design.Broadcast Ethernet LANs were considered, in which LAN stations (and routers) are connected to a shared broadcast physical medium. A packet sent from any station in such a LAN network is broadcast and received by all other stations in the same LAN.

A star-wired LAN topology is the one which is considered where each station is directly connected to a common central node known as a transparent bridge or simply bridge (or switch). The bridge is connected to each station via one of its ports, allowing a bridge with n ports to connect n stations in total. When a bridge is first connected, an Ethernet data frame received from one station (via one of its ports) is broadcasted to all other stations connected to it, i.e., by re-transmitting the frame through all ports except the one from which it receives the frame. This allows every station to hear transmissions from any other station connected to the bridge. Each frame has a frame header with the source and destination MAC addresses.We did not consider the spanning tree algorithm for supporting the bridges.

IP routers connect LANs to form an IP internetwork, allowing communication between any pair of stations on this IP network using the Internet Protocol. Stations (or, more precisely, the interfaces through which they are connected to LANs) are given globally unique IP addresses. After adding an IP packet header, a station can send an IP packet to another station. The source IP address and destination IP address are the two most important fields in the header. IP employs a table-driven, hop-by-hop datagram delivery model in which routers (and, in the first hop, stations) are responsible for forwarding IP packets to their next-hops towards the destination using a forwarding table at the stations and routers.

IP routers connect LANs to form an IP internetwork, allowing communication between any pair of stations on this IP network using the Internet Protocol. Stations (or, more precisely, the interfaces through which they are connected to LANs) are given globally unique IP addresses. After adding an IP packet header, a station can send an IP packet to another station. The source IP address and destination IP address are the two most important fields in the header. IP employs a table-driven, hop-by-hop datagram delivery model in which routers (and, in the first hop, stations) are responsible for forwarding IP packets to their next-hops towards the destination using a forwarding table at the stations and routers.An IP packet addressed to a station (the destination IP address is the station's address) should be accepted. Other packets that are not addressed to a station are discarded. If the destination IP address is not associated with the router, the router should forward the IP packet to the next hop.

Figure 1 depicts a simple IP network with three LANs realized by three bridges: Bridges 1, 2, and 3, respectively. Router 1 connects LAN 1 and LAN 2, while Router 2 connects LAN 2 and LAN 3. LAN 1 has two stations, Station A and Station B, whereas LAN 2 and LAN 3 each have a single station, Station C and Station D. Communication between Station A and Station B is direct in this configuration via Bridge 1. When Station A sends a message to Station C, Router 1 will use its routing table to forward the message from LAN 1 (Bridge 1) to LAN 2 (Bridge 2).Similarly, when Station A attempts to send a message to Station D, Router 1 forwards it from LAN 1 to LAN 2, and Router 2 forwards it from LAN 2 to LAN 3 based on the packet's destination IP address, which is Station D's IP address.

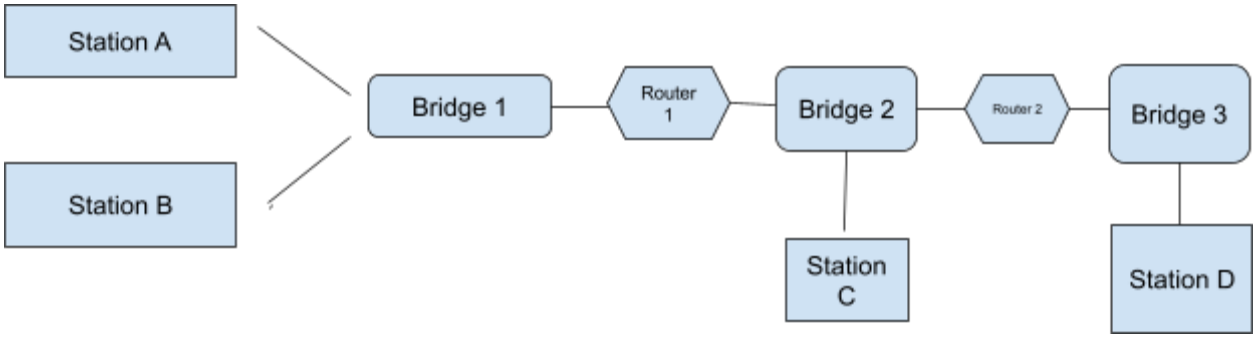


Figure 1:The Design

The Implementation

The client-server socket programming paradigm was used to create the network emulator. A bridge is used as a server and stations, or routers, connected to it, act as clients in the network emulator. A connection-oriented TCP socket connection will simulate the physical links that exist between the bridge and a station. In the client-server model, in order to connect, a client must know the server's address. Each bridge had a command line argument for this reason, which was the LAN name, lan-name. Since there is only one bridge per LAN, lan-name uniquely identifies both the LAN and the bridge. In addition, the lan-name is included in a command line argument that each station and router has, which lets them know which bridge to connect to.

Once a bridge is ready to listen as a TCP socket server, it stores its IP address and the port number by creating two files (or symbolic links) in its directory (assumed that both bridge and station are invoked from the same directory which shared across all the machines in the emulation) namely *.lan-name*.addr and *.lan-name*.port. Given a *lan-name*, a station, by reading these two files, knows the complete socket address of the bridge.

The following sections describe the program interface and the basic functionalities of each of these components.

BRIDGE:

Program Interface:
A bridge was given two command line arguments: lan-name and num-ports. A bridge must ensure that no other bridge with the same name exists on that LAN. It can connect to a maximum number of stations (and routers), as indicated by the num-ports argument. In other words, a bridge must keep track of the number of stations to which it is currently connected and accept connection requests only if the number is less than num-ports.

SYNTAX:bridge lan-name num-ports

Functionalities:

When a bridge first starts up, it creates a TCP socket as well as two symbolic link files to store the IP address of the machine on which it is running and the port number associated with the socket. It should be noted that the IP address stored in the symbolic file corresponds to the machine that is running the bridge. The bridge then awaits a request for a connection setup from a station (or router). Each request for connection setup (i.e., each station) necessitates the establishment of a new socket connection. The bridge returns the string "accept" if a connection is possible. If the connection cannot be established because there are already num-ports connections in use, the string "reject" is returned, and the TCP connection is terminated.

When a bridge receives a data frame on one of its ports but is unsure which port is associated with the destination MAC address, it distributes the frame to all other ports. Concurrently, the bridge examines its database to see if the mapping between the incoming port and the source MAC address is stored. If this mapping is not present, the bridge initiates a self-learning process that involves storing the new mapping. Through repeated data frame receptions and mappings, the bridge gradually learns which port corresponds to which MAC address or station. Furthermore, the bridge automatically removes outdated mappings corresponding to inactive stations or routers after a predetermined period to keep the database up to date and efficient. The self-learning ability of the bridge ensures efficient and optimized packet forwarding within the network, improving overall efficiency and performance.

Furthermore, as long as there is an available port on the bridge, a station can connect to it at any time. As a result, the bridge is expected to handle potential client connection requests while also forwarding data frames. This necessitates the bridge monitoring multiple concurrent events, such as connection setup requests and the arrival of data frames on established ports. To effectively manage this simultaneous monitoring of multiple events, I/O multiplexing was used. Using the select() function allows for I/O multiplexing, allowing the bridge to handle multiple events at the same time, ensuring smooth network operations and optimal resource utilization.

STATION:

Program Interface:

The station accepts three file names as arguments: interface, routing table, and hostname. These files contain critical information for the station's network functionality. The hostname file, similar to a DNS name lookup service, serves as a repository for mapping hostnames (interfaces) to their respective IP addresses. A line in the hostname file, such as "A 128.252.1.254," for example, indicates that station A's IP address is 128.252.1.254. All network stations and routers use this standardized hostname file.

The interface file contains information about the NIC card interfaces on the station. An interface file example line, such as "A 128.252.1.254 255.255.255.0 00:00:0C:04:52:27 cs1," specifies the interface name (A), the associated IP address (128.252.1.254), the network mask (255.255.255.0), the Ethernet address of the NIC card (00:00:0C:04:52:27), and the name of the LAN (bridge) to which the station interface is connected.

The forwarding table that the station uses when sending messages to other stations is stored in the routing table file. This file contains entries for each destination network prefix, next hop IP addresses, network masks, and network interfaces that can be used to reach the next hop. If the value in the second column is 0.0.0.0, the packet is intended for a machine on the same LAN. The first column, denoted by 0.0.0.0, specifies the default router. The default router in the example is 128.252.13.38.

SYNTAX:station -no interface routing table hostname

Here,the "-no" flag denotes that the program is a station (as opposed to a router).

Although stations and routers are implemented in the same source file in the demo code due to their similar functionalities, routers forward packets not intended for them, while stations do not engage in this forwarding behavior.

Functionalities Implemented:

When a station first boots up, the three files must be loaded and the data structures that will hold the information must be initialized. It must load the hostname file in particular and save the name and IP address mapping in a data structure. This mapping table is consulted when end users send a message to another station (using the station name) to determine the corresponding destination IP address. It must open the routing table file and save it as a data structure. This routing table will be consulted when a packet needs to be forwarded. It must load the interface file and store the data in a data structure.

After loading the interface file, the station is connected to all of the LANs (i.e., the corresponding bridges) specified in it. It should be noted that a station can be connected to more than one LAN. The station first reads the corresponding symbolic link files to determine the IP addresses and port numbers associated with a bridge. The station then connects to the corresponding bridge via a TCP socket. Following the establishment of the TCP socket connection, the return string from the server (the bridge) must be read. As previously stated, if the return string is "accept", the station has successfully connected to the bridge. Otherwise, the connection was rejected by the bridge (there were no ports left on the bridge).

It is critical to note that you should not keep the station waiting indefinitely for a response from the bridge. Instead, non-blocking reading on the socket connection should be used. A station should make a predetermined number of attempts to read from the socket connection (for example, 5). Following the failure of all attempts, the station declares that the bridge has rejected the connection (similar to the effect of receiving the return string "reject"). The interval between retries is also set (for example, 2 seconds). Use the system call fcntl() to change the properties of the socket (blocking, nonblocking, etc.). You should save the socket's old properties before switching to non-blocking mode. You must later restore the socket's original properties.

A station completes the three tasks listed after successfully establishing the connection. (1) It sends messages from the keyboard typed in by end users, (2) it accepts messages addressed to this station by displaying the messages received as well as the name of the station sending the messages, and (3) it handles control messages such as ARP. Before arriving at its destination, a packet must travel through several hops. We determine which router (or station) is the next hop at each hop (either a station or a router) based on the destination IP address and the local routing table.

In this network emulator, a station must support two layers: IP layer and MAC layer. When a station sends a message typed in by an end user, the message is encapsulated in an IP packet (via the addition of an IP packet header). The IP layer consults the forwarding table to determine the next-hop IP address. The IP layer determines the MAC address of the next-hop router (or the final destination if both are on the same LAN) before passing the packet to the MAC layer. Remember that the MAC layer (or, more specifically, the Ethernet card/interface) does not understand IP addresses. IP addresses are mapped to MAC addresses using the Address Resolution Protocol (ARP).

The ARP protocol is used to determine the MAC address that corresponds to an IP address. Each station (router) maintains an ARP cache with the mapping between the MAC address and the IP address. When the station needs to send an IP packet, it looks up the MAC address of the corresponding next-hop IP address in this table. If the mapping is in the table, the IP layer passes the MAC address to the MAC layer, along with the IP packet and the corresponding socket id (recall that the forwarding table tells us which interface the packet should be sent through, and the interface tells us the corresponding socket id).The MAC layer creates the data frame by appending the MAC header to the IP Packet and then send out the data frame through the corresponding socket id.

The station must first send an ARP request message if the mapping between an IP address and its corresponding MAC address is not already in the ARP cache. The IP packet will then be placed in a pending queue before being returned to the main loop to await any events (such as the end user typing something or receiving an Ethernet frame). When it receives an Ethernet frame, it must process it according to its type: IP packet or ARP packet. If the contained message is an IP packet destined for the local station, it is displayed. The packet is dropped if it is not intended for the station. If the frame is an ARP packet, the packet type is determined by the ARP protocol.

ARP packets are divided into two types: requests and replies. The ARP protocol saves the mapping between the source IP address and the corresponding MAC address if the packet is an ARP request and the destination IP address is a station local address. Furthermore, an ARP reply message is sent back to the source (the requester) containing the mapping between the IP addresses and the MAC address of the local station. The ARP protocol saves the mapping between the source IP address (the sender) and the corresponding MAC address if the packet is an ARP reply packet. However, the IP packet pending queue must be checked to see if any of the IP packets can be sent now. (at least one IP packet must be sent now).

ROUTINE:

Program Interface:

SYNTAX: station -route interface routingtable hostname

Functionalities Implemented:

A router is similar to other stations in that it must be connected to more than one bridge and forwards packets that are not intended for it.

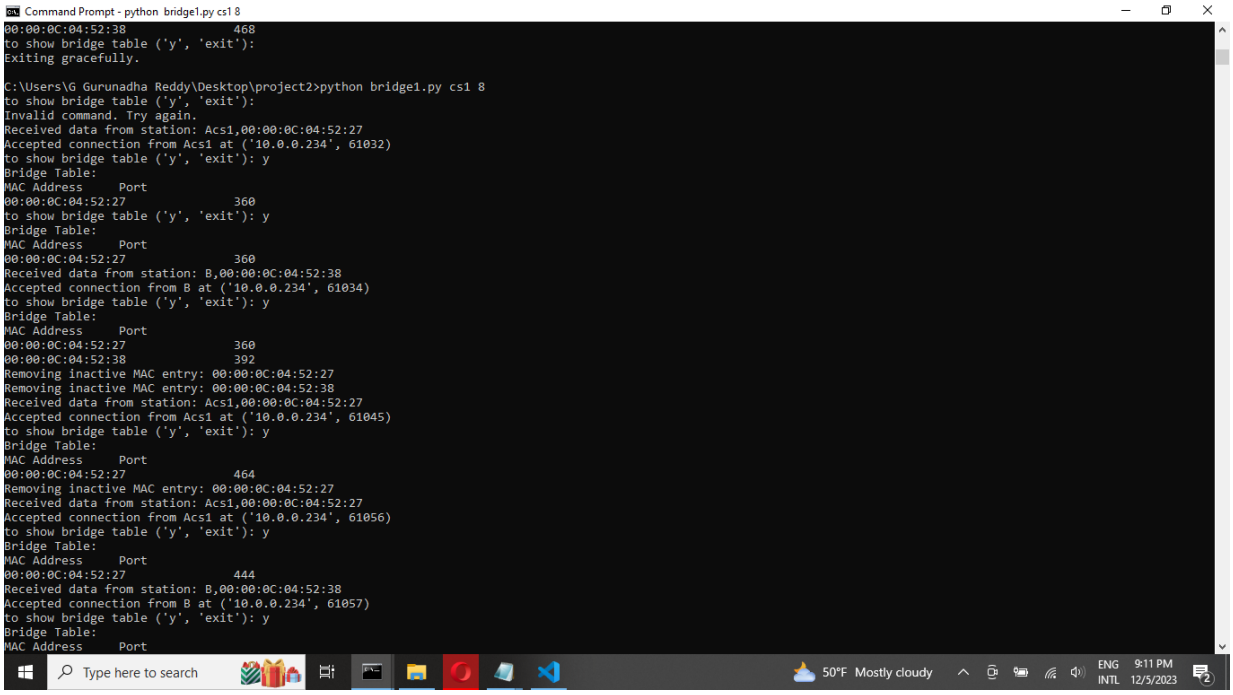
Observations

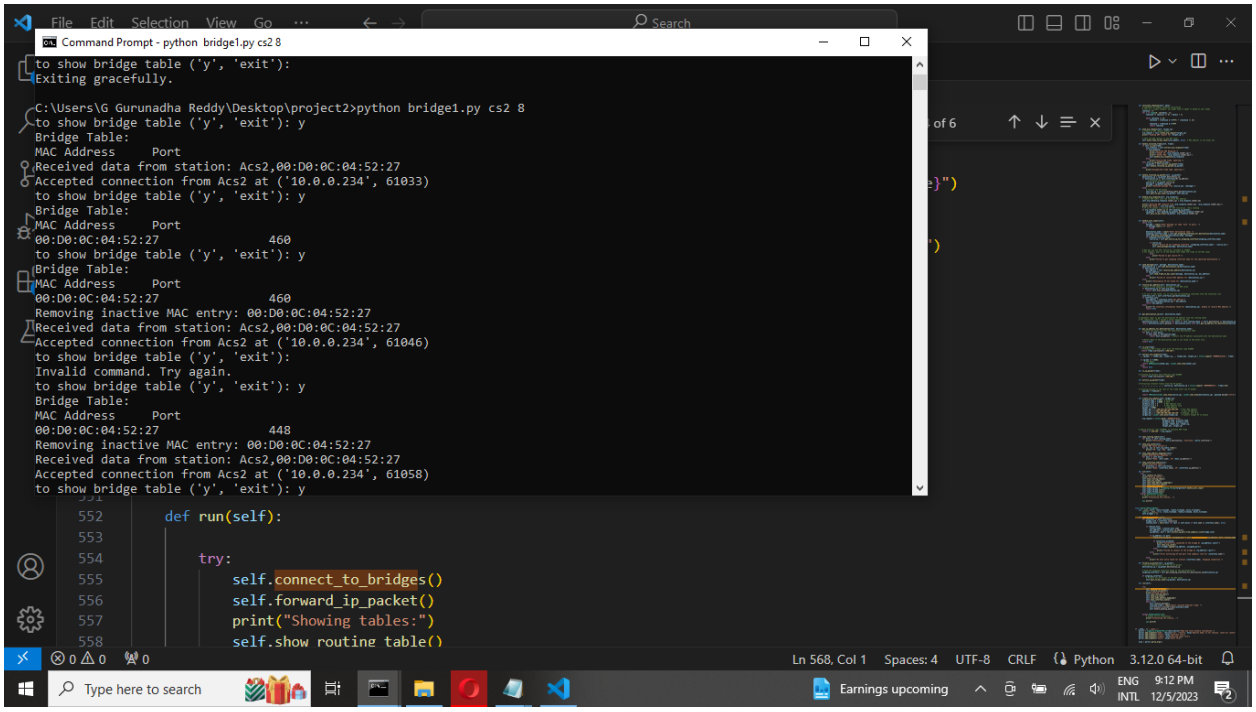
The following observations were deduced :

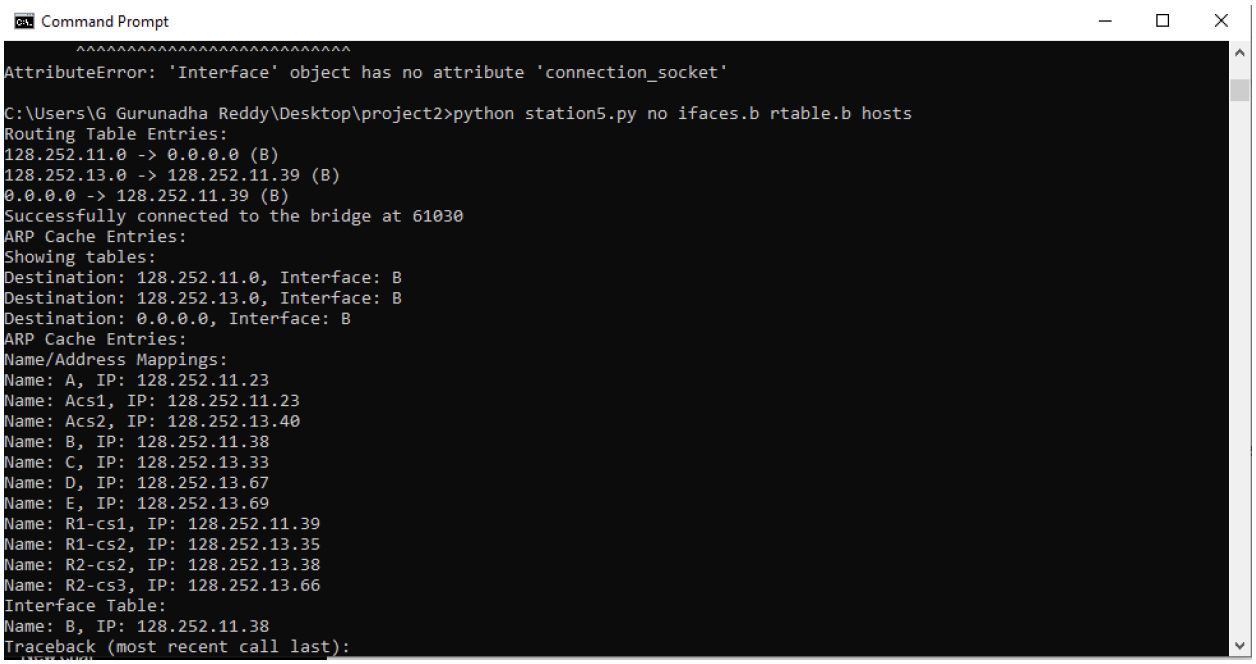
1.stations load all the configuration files and initialize the tables
2.stations send/receive Ethernet data frames (payload encapsulated in DLL header)
3.data frame broadcasting function of bridges
4.self-learning function of bridges (including entry timeout)
5.ARP protocol (ARP request, ARP reply, and ARP cache and entry timeout)
6.stations send/receive IP packets (payload encapsulated in network layer header)
7.IP packet forwarding (at a router)
8.Stations handle sending/receiving data frames and user input concurrently
9.connecting/disconnecting stations/routers at any time
10.clean up when a station/bridge gets killed (close TCP socket, clear allocated memories, etc)

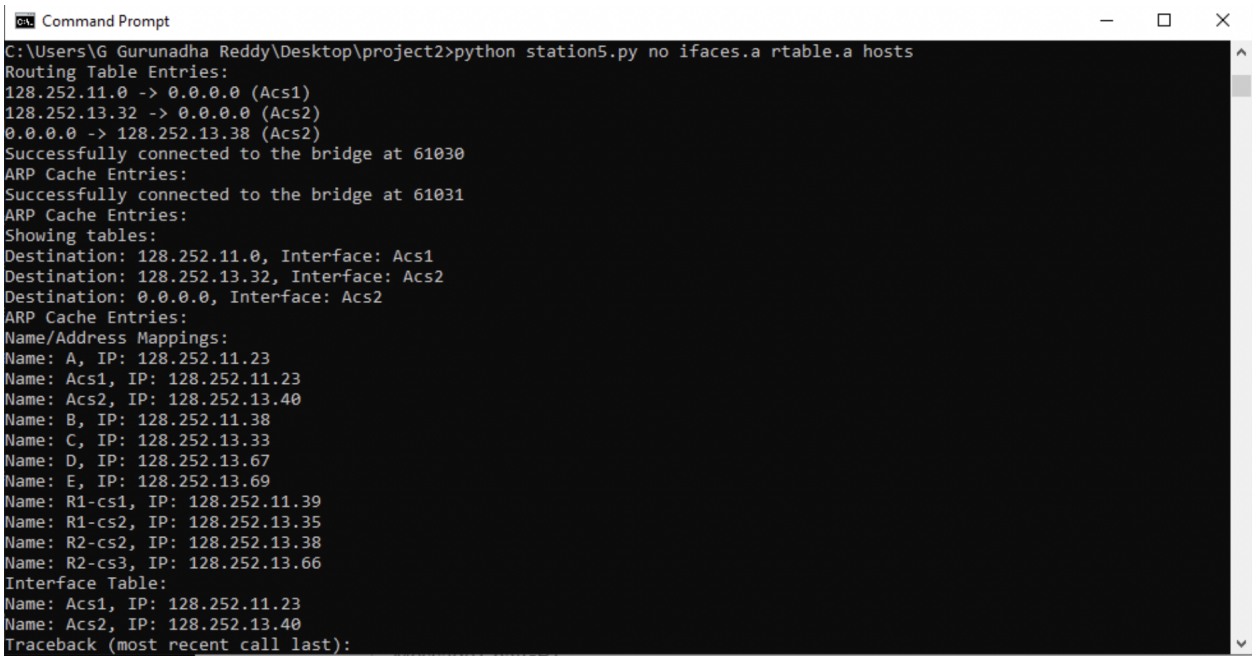
11.commands to show routing tables, ARP cache, name/address mappings, interface tables on 12.stations, and MAC address/port mappings on bridges

The Snaps of the routing tables,host interfaces are attached below.









Conclusion

We designed a network emulator using a star-lined network topology and added different functionalities to it that aims to impart a solid understanding of network emulation and software-defined networking principles, making significant contributions to both practical networking knowledge and collaborative software development skills.