Project 2: A Network Emulator

NOTE: You can form a team of up to 2 students to work on the project.

Project Objectives

1. Mastering socket based networking application development
2. Understanding functionalities of various networking devices such as bridges/switches and routers
3. Understanding how packets are forwarded in networks
4. Experiencing team software development

Project Description

In this project, you need to implement a network emulator. In the following, we will first highlight the computer network model used in this project, we then describe in detail how the components of the computer network should be implemented in software.

1. Network model overview
2. Network emulation
3. Network components
4. Project executable code and template
5. Grading policy

Section I: Network model overview

We consider broadcast Ethernet LANs, where stations (and routers)in a LAN are attached to a shared broadcast physical medium. In such a LAN network, a packet sent from any one station is broadcasted to and received by all the other stations in the same LAN

We consider a star-wired LAN topology where each station is directly connected to a common central node, called a *transparent bridge* or simply *bridge* (or switch). Each station is connected to the bridge through one of the bridge's ports, thus a bridge with *n* ports can connect *n* stations in total. When a bridge is first plugged, it broadcasts an Ethernet data frame it receives from one station (via one of its ports) to all the other stations attached to it, i.e., by re-transmitting the frame through all the ports except the one from which it receives the frame. This enables every station to hear the transmission from any other station connected to the bridge. Each frame has a frame header that includes the source and destination (next-hop) MAC addresses. Through **self learning**, the bridge gradually learns the location of the stations on the LAN and only forwards the frame to the necessary segment of the LAN. A station (and router) should accept a data frame sent by another station that is addressed to it (i.e., the destination MAC address is the address of the station or it is a broadcast MAC address). A station discards any data frames that are not addressed to it. In this project, bridges do not need to support the spanning tree algorithm.

IP routers interconnect LANs to form an IP internetwork. Any pair of stations in this IP network can communicate with each other using the Internet Protocol. Stations (or more precisely, the interfaces through which they are attached to the LANs) are assigned globally unique IP addresses. A *station*

can transmit an IP packet to another station. Before it transmits the packet, it needs to add an IP packet header. The two most important fields in the header are source IP address and destination IP address. IP uses a table-driven, hop-by-hop datagram delivery model. Routers (and in the very first hop, stations) are responsible for delivering IP packets to their next-hops towards the destination. This hop-by-hop packet forwarding is based on some forwarding table at the stations and routers. A station should accept an IP packet sent by another station that is addressed to it (i.e., the destination IP address is the address of the station). A station discards any other packets that are not addressed to it. In contrast, a router should forward to the next hop an IP packet whose destination IP address is not an IP address associated with the router.

Figure 1 presents a simple IP network, where we have three LANs, realized through three bridges, Bridge 1, 2, 3, respectively. Router 1 interconnects LAN 1 and LAN 2, while Router 2 interconnects LAN 2 and LAN 3. On LAN 1, we have two stations, Station A and Station B. LAN 2 and LAN 3 have a single station, Station C and Station D, respectively. In this configuration, Station A and Station B can communicate with each other directly through Bridge 1. When Station A sends a message to Station C, Router 1 will forward this message from LAN1 (Bridge 1) to LAN2 (Bridge 2) by consulting its routing table. Similarly, when Station A tries to send a message to Station D, Router 1 will forward it from LAN1 to LAN2, and Router 2 will forward it from LAN2 to LAN3 based on the packet destination IP address, i.e., the IP address of Station D.

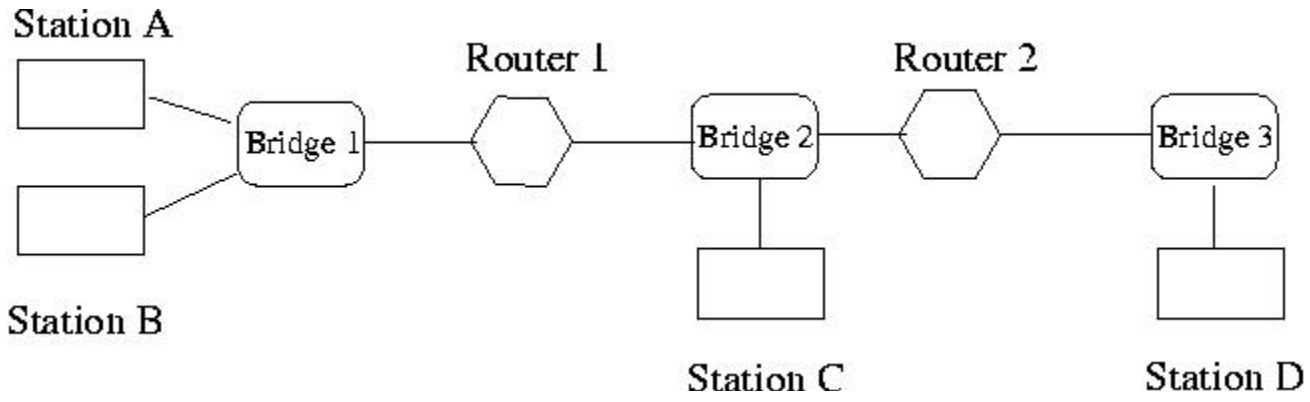

Figure 1: A simple network topology

Section II: Computer network emulation

Now let us see how we can implement the above networking concepts, elements, and functionalities in software, including physical links, stations, bridges, and routers. Simply put, we will use TCP socket connection to emulate physical links, and all the stations, bridges, and routers are implemented in software.

The network emulator will be implemented using the client-server socket programming paradigm. In the network emulator, a bridge is implemented as a server and stations (routers) attached to it as clients. The physical links between the bridge and a station will be emulated by a connection-oriented TCP socket connection. In the client-server paradigm, a client needs to know the address of the server for establishing a connection. For this purpose, each bridge has a command line argument: the name of the LAN *lan-name*. Each LAN has one single bridge, so *lan-name* uniquely identifies a LAN, and consequently, the bridge. Each station and router also has a command line argument that includes the *lan-name*, from which the station and router know to which bridge they should connect (see details in the next section).

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. Note that this is just one way of sharing information between a server and a client; you can implement this in other ways.

In this project, the forwarding table in a station or router is populated from some hard-coded configuration files.

Section III: Network components

As discussed above, our network model consists of three components, bridge, station and router. The program interface and the basic functionalities for each of these components are described below.

Bridge

- Program Interface

  A bridge takes two command line arguments *lan-name*and *num-ports*. We have already explained the role of *lan-name*. A bridge has to make sure that there is no other bridge already existing for the same *lan-name*. The argument *num-ports* specifies the maximum number of stations (and routers) that can be attached to it. So a bridge has to keep track of the number of stations currently connected to it and accept a connect request only if that number is less than *num-ports*.

  usage : **bridge lan-name num-ports**

- Functionalities

  When a bridge first starts, it needs to create a TCP socket, and 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. Note that, the IP address stored in the symbolic file is the *real* IP address of the machine on which the bridge is running. Then the bridge waits for connection set-up request from a station (or router). A separate socket connection needs to be established for each connection setup request (i.e., for each station). If a connection can be established, the bridge should return the string "accept". If the connection cannot be established because, for example, there are already *num-ports* connections, the string "reject" should be returned, and the TCP socket connection should be closed.

  When a bridge receives a data frame on one port, it needs to broadcast the frame to all the other ports, if it does not know which port the corresponding destination MAC address in the data frame is associated with. When the bridge receives a data frame from a port, it also needs to check if the mapping between the (incoming) port and the corresponding source MAC address is already in one of its database (some data structure). If the mapping is not in the database yet, the bridge stores the mapping. This is the so-called self-learning process. Through this process, the bridge gradually learns which port is associated with which MAC address (station). The mapping should be removed from the database if the corresponding station/router is not active for a fixed period of time.

Moreover, a station may need to attach to the bridge at any time (as long as there is a port available at the bridge). A bridge must wait for possible connection request from a client in addition to performing the data frame forwarding function. In other words, the bridge must monitor several events (connection setup request and data frame arrival on established ports) that can occur at the same time. In order to achieve this I/O multiplexing, you may want to use the select() function.

Station

- Program interface

  A station takes three arguments, interface, routingtable, and hostname. All of them are file names. The file hostname contains the mapping between host (interface) names and IP addresses. For example, the following is an example line taken from a hostname file,

  A 128.252.1.254

  which states that station A's IP address is 128.252.1.254. This provides some similar functionality of DNS name lookup service. All stations and routers use a common hostname file.

  The file interface contains the (NIC card) interface information of the station. The following is an example line taken from an interface file,

  A 128.252.1.254 255.255.255.0 00:00:0C:04:52:27 cs1

  which states that, the (station) interface name is A, its corresponding IP address is 128.252.1.254, and the network mask is 255.255.255.0. The fourth field is the Ethernet address of the NIC card. The last field is the name of the LAN (bridge) that the (station) interface is connected to.

  The file routingtable contains the forwarding table that the station will use when it sends a message to another station. One example of this file's content is,

  128.252.11.0 0.0.0.0 255.255.255.0 A
  0.0.0.0 128.252.13.38 0.0.0.0 A

  The first column is the destination network prefix, the second one is the next hop IP address, the third the network mask, and the last is the network interface through which the next hop can be reached. If the second column is 0.0.0.0, the packet is destined to a machine on the same LAN. If the first column is 0.0.0.0, the corresponding row is the default entry and the corresponding second column is the default router. In the example, 128.252.13.38 is the default router.

  usage: **station -no interface routingtable hostname**

  where -no specifies that this is a station instead of a router. (Stations and routers are implemented in the same source file in the demo code, as they have very similar functionalities, except that routers forward packets that are not destined to them, while stations do not.)

- Functionalities

  When a station first starts, it needs to load in the three files and initialize the corresponding data structures to hold the information. In particular, it needs to load the *hostname* file and store the name and IP address mapping in a data structure. Later when end users wants to send a message to another station (using station name), this mapping table will be consulted to find out the corresponding destination IP address. It needs to load the *routingtable* file and store the routing table in a data structure. This routing table will be consulted when a packet needs to be forwarded. It needs to load the interface file and store the interface information in a data structure.

  After loading in the interface file, the station needs to connect to all the LANs (i.e., the corresponding bridges) specified in the interface file. Note that a station can connect to multiple LANs. The station first needs to read the corresponding symbolic link files to find out the IP addresses and the port number associated with a bridge. Then the station initializes a TCP socket connection to the corresponding bridge. After the TCP socket connection is established, it needs to read the return string from the server (the bridge). As mentioned above, if the return string is "accept", the station knows that the station has been successfully plugged to the bridge. Otherwise, the connection was rejected by the bridge (no ports left on the bridge).

  Note that you should not let the station wait for the reply from the bridge forever. Instead, you should use non-blocking reading on the socket connection. A station should try to read from the socket connection for a preset number of times (for example, 5 times). After all tries fail, the station declares that the connection is rejected by the bridge (similar to the effect of having received the return string "reject"). The waiting time between retries is also preset (for example, 2 seconds). You can use the system call fcntl () to change the properties of the socket (blocking, nonblocking etc). You should store the old properties of the socket before you change to nonblocking mode. Later you need to restore the old properties of the socket.

  After the connection has been successfully established, a station needs to perform the following three tasks. (1) It sends out messages typed in by end users from keyboard, (2) it accepts messages addressed to this station by displaying the messages it receives and the name of the station sending the messages, and (3) it handles the control messages such as ARP. A packet may traverse many hops before reaching the destination. At each hop (either a station or router), we need to determine which router (or station) is the next hop based on the destination IP address and the local routing table.

  In this network emulator, a station needs to support two layers: IP layer and MAC layer. In particular, when a station sends out a message typed in by end user, it needs to encapsulate the message in an IP packet (adding an IP packet header). The IP layer needs to consult the forwarding table to determine the next-hop IP address. Before the IP layer passes the packet to the MAC layer, the IP layer needs to identify the MAC address of the next-hop router (or the final destination if on the same LAN). Recall that the MAC layer (or rather the Ethernet card/interface) does not understand IP addresses. Mapping from IP address to MAC address is done using the Address Resolution Protocol (ARP). Below we describe ARP briefly. Please refer to the textbook and online materials for a detailed description of ARP.

Simply put, the ARP protocol is used to identify the MAC address associated with an IP address. Each station (router) maintains an ARP cache, which contains the mapping between MAC address and IP address. When the station needs to send an IP packet, it consults this table to look for the MAC address of the corresponding next-hop IP address. If the mapping is in the table, the MAC address is taken and passed to the MAC layer by the IP layer, together with the IP packet and the corresponding socket id (recall that from the forwarding table we know through which interface the packet should be sent, and from the interface, we know the corresponding socket id). The MAC layer forms the data frame by adding the MAC header to the IP packet and then send out the data frame through the corresponding socket id.

However, if the mapping between an IP address and the corresponding MAC address is not in the ARP cache yet, the station first needs to send an ARP request message. Then it will store the IP packet in a pending queue and return to the main loop to wait for any events (end user types anything or receiving an Ethernet frame). After receiving an Ethernet frame, it needs to process the frame according to the type of the frame: IP packet or ARP packet. If it is an IP packet and destined to the local station, the contained message is displayed. The packet is dropped if the packet is not destined to the station. If the frame is an ARP packet, the ARP protocol checks the type of the packet. ARP packet can be of two types: ARP request or ARP reply. If it is an ARP request packet and the destination IP address is a local address of the station, the ARP protocol stores the mapping between the source IP address and the corresponding MAC address. Moreover, an ARP reply message is sent back to the source (the requester), which contains the mapping between the IP addresses and the MAC address of the local station. If the packet is an ARP reply packet, the ARP protocol stores the mapping between the source (the sender) IP address and the corresponding MAC address. Moreover, the pending queue of the IP packets needs to be checked to see if any of the IP packets can be sent now (actually, there must be at least one IP packet that can be sent now).

Router

- Program interface

  usage: station -route interface routingtable hostname

- Functionalities

  A router is like any other stations, except that it must be connected to more than one bridge and they forward packet that is not destined to it.

An example session to start the emulation

In different xterm windows:

- bridge cs1 8 (starting bridge 1 with name "cs1". It has 8 ports)
- bridge cs2 8 (second bridge)
- bridge cs3 8 (third bridge)
- station -route ifaces.r1 rtable.r1 hosts (first router. The interface information of the router is in file ifaces.r1. Routing table in rtable.r1. IP/hostname mapping in hosts)
- station -route ifaces.r2 rtable.r2 hosts (second router)

- station -no ifaces.a rtable.a hosts (first host; Host A)
- station -no ifaces.b rtable.b hosts (second host. Host B)
- station -no ifaces.c rtable.c hosts (third host. Host C)
- station -no ifaces.d rtable.d hosts (last host. Host D)

Section IV: Provided code of the project

(read the README.txt file first after untaring)

- Executable code of the project
- Project template code

Section V: Grading policy

We only grade the part of the code that works. We also use Berkeley MOSS to detect potential plagiarism. Your code should at least implement the following functionalities (10 points each).

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)
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 stations, and MAC address/port mappings on bridges
12. code readability/report

At the beginning of all your source code files, you must include the following information:

Course: (course ID and name)
Semester: Spring 2016
Names: (names of your team)

Hints

Given that the second project is a relatively large project, you need to design carefully how you should proceed with the project. For example, you may want to work on the project in two phases.

In phase one, you only work on a single LAN, with a single bridge connecting multiple stations. You only need to worry about the data link layer, you do not worry about the network layer (IP layer). (However, to simplify your later development of the IP layer), you may want to have an empty IP layer (an empty function) so that later you can add the network layer functions.) This is in principle similar to the first project. You need to pay attention to two things: packet encapulation (that is, you need to have a data link layer header and payload in a dataframe you send out). Second, the bridge needs to support the self-learning function and maintian the bridge table. For this purpose, you can

add the MAC address into the hostname file so that you can map a host name to the corresponding MAC address.

In the second phase, you add the support of IP and ARP. When you start to work on the ARP part, remember to remove the MAC addresses you add into the hostname file.

Please check the provided template, in particular the ether.h and ip.h files. They provide the data structure for ethernet data frames and ip packets. Note that you do not need to use the exact the format; you can define your own data frame and ip packet format.