CSC358H5: Principles of Computer Networking — Winter 2025

Programming Assignment 1: Network Interface and ARP

Due Date: Sunday, Feb 23, 11:59:59 PM

The main goal of PA0 was to setup the environment for programming assignments and also prepare you for modifying and testing C++ code during this. This assignment will use that environment to implement a simplified portion of the network stack: creating Ethernet packets from IP packets and sending them out. It is based on one of Stanford's CS 144 lab assignments.

Collaboration

All programming assignments should be done individually. All codes that you use in your submissions should be written by you. Do NOT simply copy-and-paste code from websites such as Stack Overflow, GitHub, or even AI generating agents. It is OK to see the sample code snippets on such places, but you should write the code yourself and give the credit to them by citing the source that you used (*e.g.*, by mentioning the page URL).

You should not share your code with others or see other student codes. It is OK to discuss the assignment or potential solutions for them with another student. However, in addition to writing the whole code yourself, you should also mention this (the discussion with other students in finding the answer to the assignments) in the submissions.

You can also ask your questions on the course discussion board publicly, as long as you do not reveal your code. If there is a specific question related to a part of your code that needs revealing it, either ask it in a tutorial session or an office hour from a TA or via a private question on the discussion website.

Using Git

You are welcome to store your code in a private repository on GitHub, GitLab, Bitbucket, etc., but please make sure your code is not publicly accessible.

0 Background

One of the key ideas of the networking stack, which has been explained in detail in the course, is to hide details of lower layers from higher levels and make it possible to have different upper layers that rely on the same lower layer. In this assignment, we focus on the Data Link layer which is responsible for creating and sending out frames and also receiving and processing incoming frames. In particular, we will focus on Ethernet frames. Your task is to complete a simplified network interface to perform **creating and sending Ethernet frames** and **receiving Ethernet frames and extracting the frame content**.

Before we explain the tasks in this assignments, let us briefly review how a packet traverse the network to reach its destination. Consider the network topology illustrated in Figure 1, in which Host A and B, together with Router's Interface 1 are in the same physical network, which means that they can communicate with each other at the data link layer (*i.e.*, send frames to each other) without requiring a router. In other words, communication happens through switching, not routing. Similarly, Router's Interface 2 and the Server are in the same physical network. Assume that Host A wants to send a packet to the Server.

Figure 2a illustrates the networking stack for this topology. Recall that packets get additional headers wrapped around them as they move down the stack, to lower layers. To send an IP packet, we first fill in its destination IP at Layer 3. Then, we pass that packet down to Layer 2, where we have to add a MAC address to send the packet along the link. For instance, consider the packet on wire at time t. Figure 2b illustrates the L2 and L3 address fields of this packet. What MAC address should Host A use for the destination MAC address field?

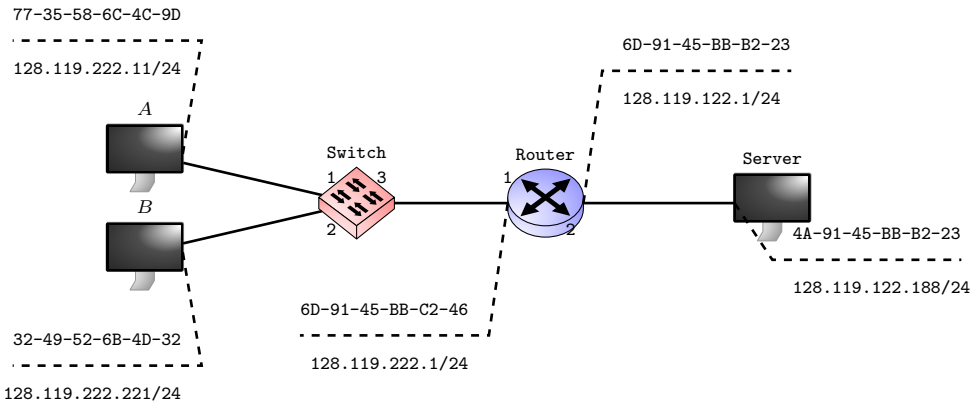
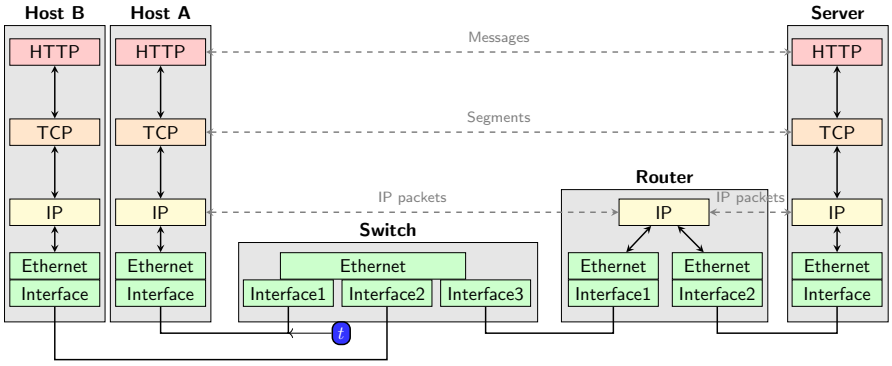
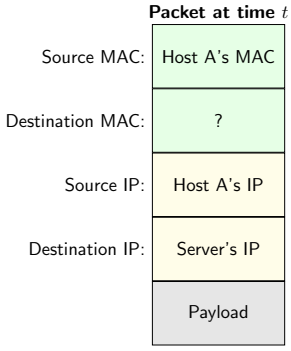


Figure 1: Host A and B and Router's Interface 1 are in the same physical network, and Router's Interface 2 and the Server are in another physical network. Host A sends one packet to the Server.



(a) Illustration of the networking stack for the topology depicted in Figure 1.



(b) Address fields in L2 and L3 headers of the packet at time t.

Figure 2: What MAC address should Host A use for the destination MAC address field?

0.1 Create and Send Ethernet Frame

The first task of the link layer that you will implement in this assignment is to create and send Ethernet frame. When the protocol in the upper layer (*i.e.*, IP) has a packet to send, it first must determine who the next hop is to send the packet to (*i.e.*, the next machine/router on the network that should process it). The next hop is a machine that resides in the same physical network, which means that we can directly send packets to it. The next hop is determined as part of the packet routing. For instance, Consider the example in Figure 1. When IP in Host A wants to send a packet to the Server, the next hop will be determined as the Router's Interface 1. In this assignment, we assume that the next hop has been given to us. In the next programming assignment, we study how the next hop is determined.[1]

Next, IP will invoke the link layer to send that packet to this next hop and provides the IP packet and the IP address of the next hop to the link layer. For instance, IP in Host A in Figure 1 will invoke the link layer to send the packet, which has Source IP address 128.119.222.11 and destination IP address 128.119.122.188 (*i.e.*, Server's IP address), to the next hop which has IP address 128.119.222.1 (*i.e.*, the IP address of the Router's

[1]As a short teaser, this is how the next hop is determined, which will be discussed in details in future: First, we need to check if the destination IP is a machine in our own local network, or a machine in a different local network. To determine this, the sender's forwarding table will have an entry indicating the range of local IP addresses, sometimes called our **subnet**. For example, the entry might say that 128.119.222.0/24 is direct, which means all addresses between 128.119.222.0 and 128.119.222.255 are on the same local network. The table also has a default route, saying that all other non-local destinations should be forwarded to the router.

Interface 1).

However, we should use MAC addresses in this layer to communicate with other machines. The network layer only provides the IP address of the next hop. If the destination IP is in our subnet, we need some way to translate between the destination IP address and that machine's corresponding MAC address. If the destination is outside our subnet, we need some way to translate the router's IP address (from the forwarding table) to its corresponding MAC address, so we can send the packet to the router. So the first job of the link layer is to figure out the corresponding MAC address of this IP address.

Address Resolution Protocol (ARP) allows machines to translate an IP address into its corresponding MAC address. Link layer maintains a **ARP Table** which translates IP addresses to MAC addresses. This table contains the MAC addresses of different IP addresses that this machine is aware of.

If the IP address of the next hop is available in the table (*i.e.*, the machine knows the corresponding MAC address for it), it can easily send the packet out by creating an Ethernet frame with appropriate header sections (such as its own MAC address as the source and the MAC address of the next hop as destination) and include the given IP packet as the content of the frame.

If the next hop's IP address is not available in the table, the link layer will try to find the MAC address of this IP address first. To do so, Address Resolution Protocol (ARP) is used to find the MAC address of an IP address.

- To request a translation, a machine can broadcast a soliciation message: "I have MAC address 77-35-58-6C-4C-9D. What is the MAC address of the machine with IP 128.119.222.18?" In the example illustrated in Figure 1, this broadcast frame will be received by Host B and Router's Interface 1.
- All machines who are not this IP address ignore the message. The user who has this IP address unicasts a reply to the sender's MAC address, saying "I am 128.119.222.18, and my MAC address is 6D-91-45-BB-C2-46."[2]
- When the machine receives an IP-to-MAC mapping, it adds it to the local **ARP Table**, which caches these mappings for the future. The table also includes an expiry date for each entry, since IP addresses aren't permanently assigned to a computer. A different computer could get assigned the same IP address, or the same computer could change IP addresses. After expiry of an entry, it will be removed from the table. In this assignment, use 30 seconds as the expiry time.

Note that ARP runs directly on Layer 2, so all packets are sent and received over Ethernet, not IP.

If it could find the MAC address of this IP address, it sends the waiting packet out. However, if it could not find the MAC address of this IP address after the maximum wait time, it will assume that the next hop is not active anymore and drops the packet. In this assignment, use 5 seconds as the maximum wait time.

0.2 Receive Ethernet Frames and Extract Frames Content

The other task of the link layer that you must implement in this assignment is to process the incoming Ethernet packets. When it receives an Ethernet packet that is addressed to its machine, it will try to extract the content of the frame and pass it to the upper layer (*i.e.*, IP). It also has to process incoming ARP requests and answer anyone that relates to it.

1 Getting Started

In this assignment, you will implement the two tasks described in sections 0.1 and 0.2 to create a simple link layer stack for a network interface. You will continue to use the same starter code directory from pa0. The following commands remain the same from PA0, the only difference is that the target now is "pa1".

[2]Machines can also broadcast their own IP-to-MAC mapping to everybody, even if nobody asks.

Run the following command to create directory for compiling the code.

Linux Command: Configuring the build system and generating the build files in the "build" directory

```
cmake -S . -B build
```

Enter the build directory. You can build the source code by running the following command **from the build directory**:

Linux Command: Building the source code

```
cmake --build .
```

You are ready to start working on the source code to complete the assignment. Whenever you want to run build your solution and run tests on it, you can run:

Linux Command: PA1 automated tests

```
cmake --build . --target pa1
```

2 TODO List!

There are 4 functions inside *NetworkInterface* class (*network_interface.cc* file) that you need to complete:

1. `void NetworkInterface::send_datagram( const InternetDatagram& dgram, const Address& next_hop )`

   This is the function that will be called from IP layer to send an IP packet (dgram) to the next hop (with IP address next_hop). You should complete this function so that it performs the following tasks:

   - First, check to see if you know the MAC address of the next hop. You should look up in your caches ARP table for this. If you know the MAC address, you can create the proper Ethernet frame and put it in the ready-to-be-sent queue (which will be discussed later on). To do this, you should create an Ethernet frame, set the type as IPv4 packet (type = EthernetHeader::TYPE_IPv4), properly set the source and destination MAC addresses, and put the serialized version of the dgram in it.
   - If you do not know the MAC address of the next hop, you should try to find it. This is done by broadcasting an ARP request and asking who has that MAC address. Then you should wait for the response to that request. You should also queue this packet in a queue so that you can prepare it for sending when you receive a reply. There is one extra thing that you need to consider here. If you have sent an ARP request for the same IP address in the last 5 seconds, you should NOT send a new ARP request. Instead, you should append this packet to queue that holds previous packets that are waiting for that IP address.

2. `optional<InternetDatagram> NetworkInterface::recv_frame( const EthernetFrame& frame )`

   This is the function that is called when the system receives an Ethernet frame. It job is to process it as follows:

   - If this packet is not destined for this machine, discard it. A packet is destined for this machine either if its destination MAC address is this machine, or it is broadcasted to the whole network.
   - If this packet is destined to this machine and its payload an IPv4 packet, then try to parse the payload as an *InternetDatagram*. If the parse was successful, it should be returned (so that the system can pass it to the higher IP layer in the network stack).
   - If this packet is destined to this machine and its payload is an ARP packet, process the ARP packet. To do this, it should first try to parse the payload as an ARPMessage. If it could be parsed properly, then it should learn the mapping between the packet's Sender IP address and its MAC address and cache this in the ARP cache table. This information should be cached for 30 seconds.

Furthermore, if it is an ARP request that asks for our IP address, reply back to it. To do this, you should create an ARP reply packet that is destined to the sender and contains proper information (including our IP and MAC address). The package this ARP message in an Ethernet frame and place it in the ready-to-be-sent queue.

3. `optional<EthernetFrame> NetworkInterface::maybe_send()`

   Whenever the physical layer of the network is ready to send out a packet, it will call this function to check if there is any packet ready to be sent. This is where you should check your ready-to-be-sent queue to see if there is any packet in it. If there any packet, you should remove the first packet waiting in the queue (the oldest packet) from it and return it. Otherwise, if there is no packet to be sent, simply return nothing.

   Note that there could be different Ethernet packets in the queue: datagrams that are passed by IP layer, ARP requests to learn the MAC address of a next hop, and ARP replies to requests that are sent to us about our IP addresses.

4. `void NetworkInterface::tick( const size_t ms_since_last_tick )`

   This is the callback function that informs you about the passage of time. When this function is called, it means that *ms_since_last_tick* milliseconds are passed from the last time that it was called. You should keep track of time and perform the following two tasks:

   - Expire any entry in ARP cache table that was learnt more than 30 seconds ago.
   - Remove the pending ARP reply wait for any next hop IP that was sent more than 5 seconds ago. Furthermore, you should also empty any packets waiting for that IP address from the queue.

   You are only allowed to modify `network_interface.cc` and `network_interface.hh` files. Note that any extra helper function and any extra state information (*e.g.*, the ARP cache table) must be added to the `NetworkInterface` class. **Do not modify any other file**, because you will only submit these two files along with `writeups/pa1.md`.

3 Submitting The Assignment

You can access the MarkUs submission website of the course at:
`https://markus.teach.cs.toronto.edu/utm-2025-01/main/login_remote_auth`.

We prepared the submission assignment for Programming Assignment 1 there (PA1). You should be able to login to the MarkUs with your UTORid credential. For this assignment, you should submit three files:

- `network_interface.cc`
- `network_interface.hh`
- `pa1.md`

`pa1.md` includes a few questions that you should answer. Please submit all three files.

4 Grading

Your submission will be graded by the tests that were given to you in the starter code. PA1 is graded with a total of 25 marks and it contributes 10% towards your course grade. The first test (*i.e.*, Test #1 - compile with bug-checkers) is worth 1 mark and each of the remaining six tests (*i.e.*, Test #3-8) is worth 4 marks.

The writeup file is for us to receive feedback from you and to make sure that the submitted work is the student's own work. It's main functionality is to detect plagiarism and cheating, which would be reported the department and further actions will be taken by them.