CSC458 PA1
Packaging IP in Ethernet

Ehsan Etesami
ehsan.etesami@utoronto.ca

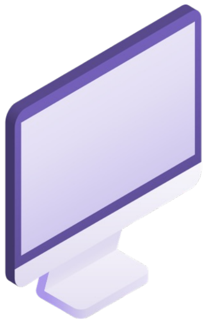
Thanks to Farid Zandi and Parsa Pazhooheshy

Winter 2025

Department of Computer Science
University of Toronto

Introduction

IP address can be changed



IP Address: 10.0.0.1
MAC Address: 00:1B:44:11:3A:B7

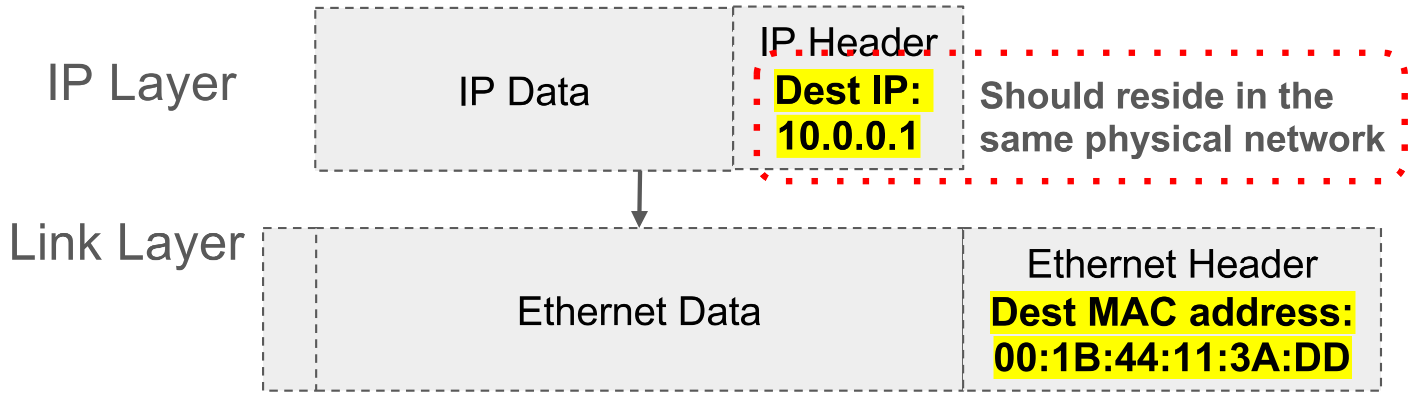
unique to every network card or interface

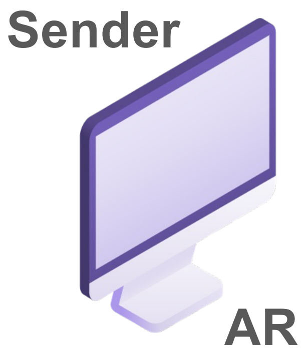
When data is sent over a network, it uses the IP address to find the correct destination.

Within the local network, it needs the MAC address to actually deliver the data to the right device.

Introduction

Assumption: We have the IP packet and the IP address of the next hop.





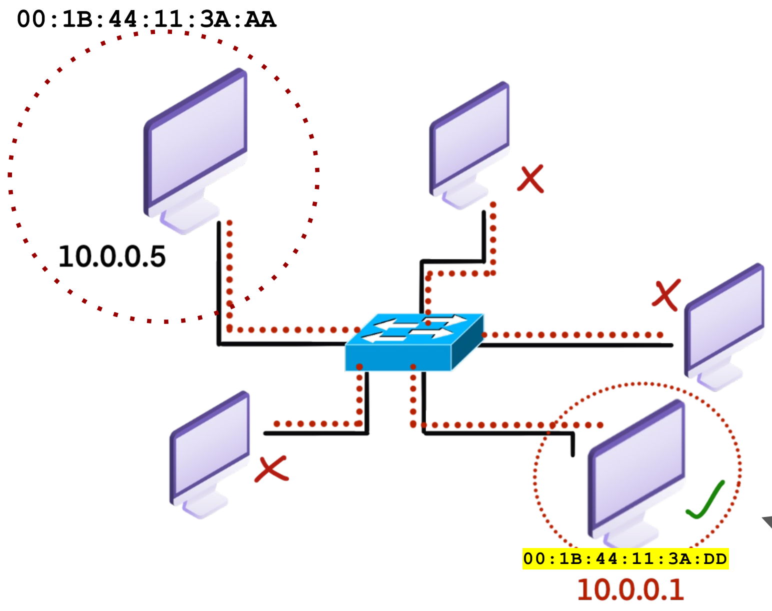
ARP Table

IP Address	MAC Address	TTL
10.0.0.1	00:1B:44:11:3A:DD	30000

It only caches each record for a limited time (30 seconds); *Computers can move between networks!*

ARP: Address Resolution Protocol

Introduction



Who has 10.0.0.1? Tell 10.0.0.5?

This is a broadcast message, the dst MAC address is FF:FF:FF:FF:FF:FF

What is the corresponding MAC address of the destination IP address?

ARP Table at 10.0.0.5

IP Address	MAC Address	TTL
10.0.0.1		

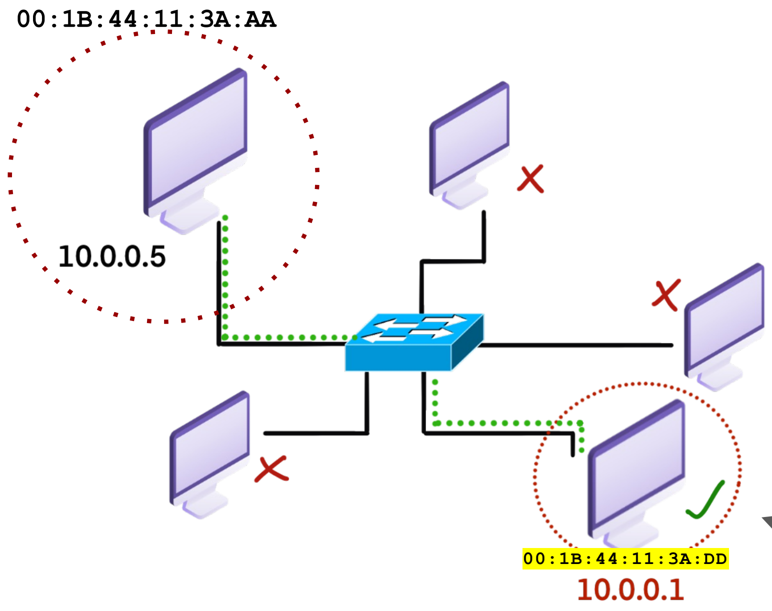
No response after 5 seconds?

Assume the next hop is not active anymore.

ARP Table at 10.0.0.1

IP Address	MAC Address	TTL
10.0.0.5	00:1B:44:11:3A:AA	30000

Introduction



What is the corresponding MAC address of the destination IP address?

ARP Table at 10.0.0.5

IP Address	MAC Address	TTL
10.0.0.1	00:1B:44:11:3A:DD	

No response after 5 seconds?
Assume the next hop is not active anymore.

ARP Table at 10.0.0.1

IP Address	MAC Address	TTL
10.0.0.5	00:1B:44:11:3A:AA	30000

Who has 10.0.0.1? Tell 10.0.0.5?

10.0.0.1 is at 00:1B:44:11:3A:DD

PA1: Deliverables

network_interface.hh

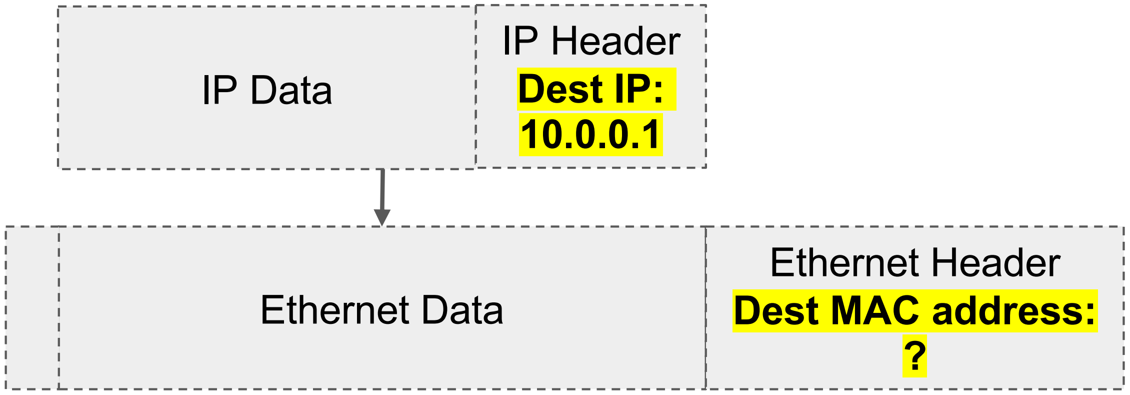
network_interface.cpp

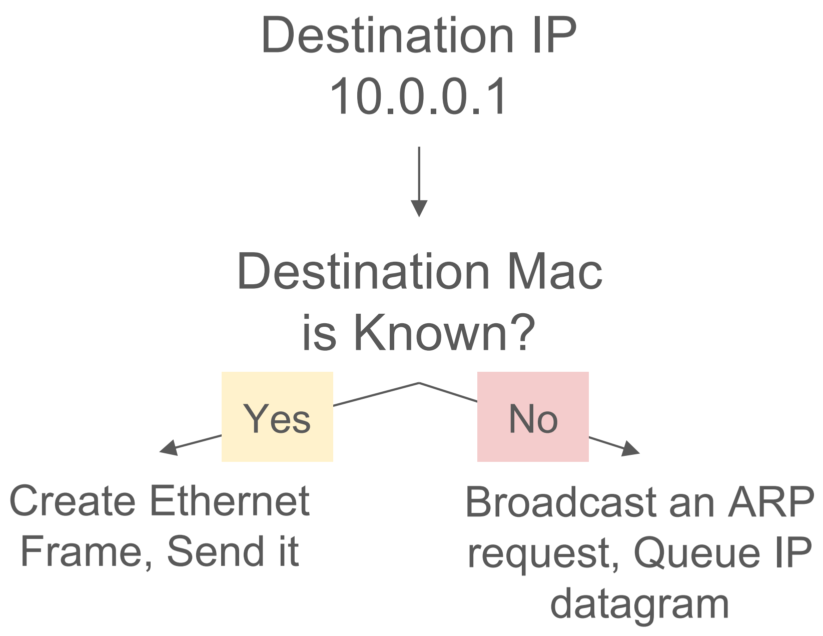
pa1.md

```
class NetworkInterface {
  private:
    EthernetAddress ethernet_address_;
    Address ip_address_;

  Public:
     NetworkInterface(
          const EthernetAddress& ethernet_address,
          const Address& ip_address);

     std::optional<EthernetFrame> maybe_send();

     void send_datagram(
          const InternetDatagram& dgram,
          const Address& next_hop);

     std::optional<InternetDatagram> recv_frame(
          const EthernetFrame& frame);

     void tick(size_t ms_since_last_tick);
};
```

Functions to Complete

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

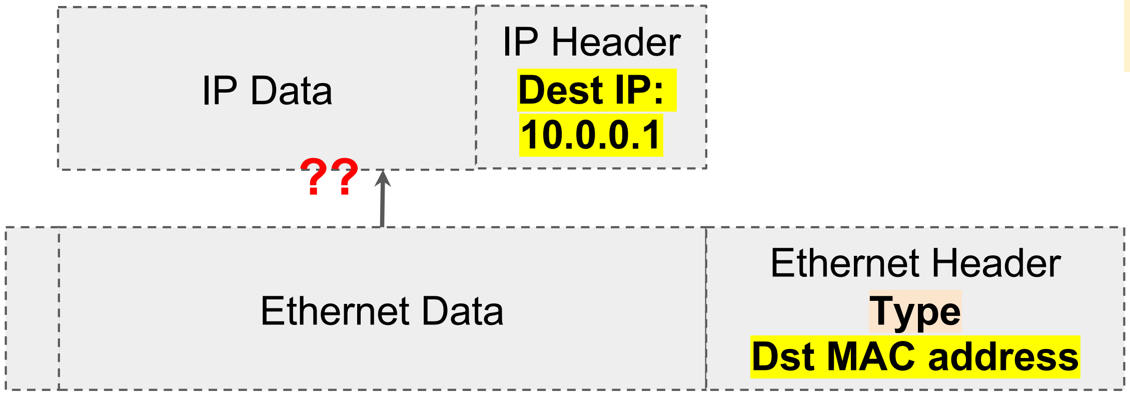




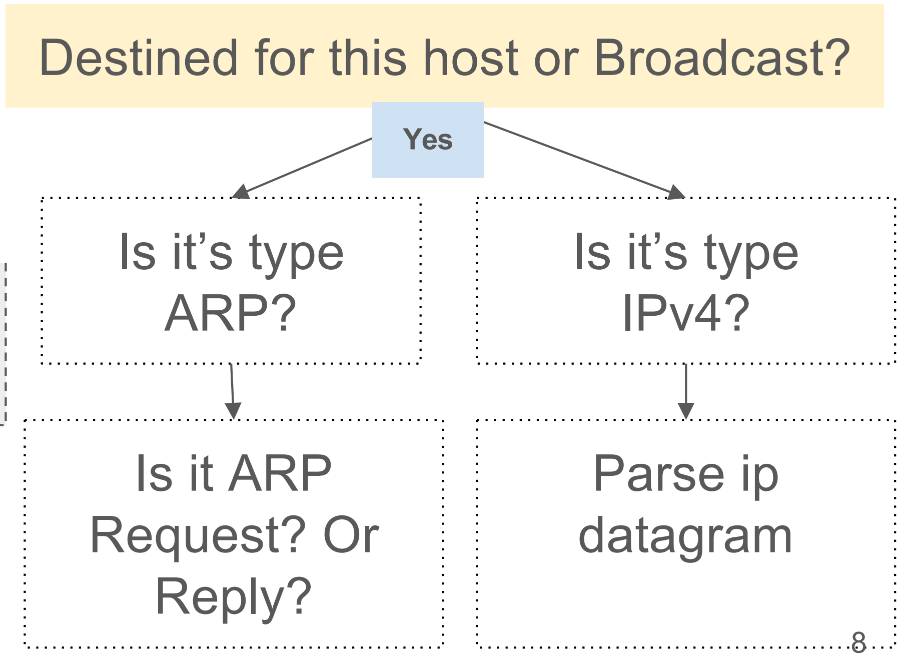
Note: Don’t flood the network with ARP requests:
If you have sent an ARP request for the same IP address in the last 5 seconds, you should NOT send a new ARP request.

Functions to Complete

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



Each time an ARP reply/request is received for an existing ARP entry you need to **update entry's data including the TTL field**.


Destined for this host or Broadcast?
Yes
Is it's type ARP?
Is it's type IPv4?
Is it ARP Request? Or Reply?
Parse ip datagram

Functions to Complete

```
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.

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

This is the callback function that informs you about the passage of time. You need to check:

- 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.

PA1: Understanding the Tick Function

- To simulate the passage of time, we will call the **tick function** with a given value of X milliseconds.
- This means that X milliseconds have passed **since the last time** the tick function was called.
- Inside the function, you should check the Time-To-Live (TTL) of all entries and ensure that **none** of them have expired.

PA1: Notes

You can modify both **network_interface.cc** and **network_interface.hh** files.

You will only submit **these two files** along with **writeups/pa1.md**

We will use the **MarkUs** submission system.

We will

- Check your submission with automated tests **(90%)**
    - **50%** is dedicated to public tests
    - 40% is dedicated to private tests
- Read your source code and asses its coding style **(5%)**
- Check completeness of the pa1.md **(5%)**

PA1: Environment Setup

On Intel/AMD computers:

- Use a VM image in VirtualBox

On ARM MacBooks and Macs:

- Install the UTM virtual machine software
- Use the provided ARM64 GNU/Linux virtual machine image

Using your own GNU/Linux installation

- Require **C++20 compiler** (**GCC** 13 or later, **clang** 16 or later)
- Ubuntu 23.04+

https://stanford.edu/class/cs144/vm_howto/

PA1: Walkthrough: Environment Setup

1. Download the VM image
2. Connect to your VM
3. Clone the repository
4. Use VSCode to connect to your VM through SSH

Let’s put it into practice.

PA1: Writeups

The "**Credit/Thank**" Section:

- Acknowledge anyone who assisted you in completing the assignment.

The "**Collaborate**" Section:

- Any kind of equal collaboration. We just want to make sure we understand the nature of any potential collaboration.

The "**Program Structure and Design**" Section:

- Explain the key design decisions you made to help reviewers understand your code. Start with a high-level overview of your approach, highlighting the most important choices. This section is not a substitute for inline comments but should be read before the reviewer examines your code in detail.