Spring 2014
Computer Networks
CMPE323

Laboratory Experiment No. 4: Introduction to Internet Protocol (IP) and Class-less Inter-domain Routing (CIDR)

Aims and Objectives:

- Introduce students to a scalable best-effort end-to-end routing architecture.
- Introduce students to IP versions 4 and 6 as examples of such routing architecture.
- Allow students to manually craft their first IPv4 and IPv6 packets (according to IETF standards RFC791 and RFC2460 respectively).
- Practically verify the correct routing behavior according to the CIDR architecture (IETF standard RFC4632).

Materials Required:

- IP routers,
- Ethernet switches,
- PCs with Ethernet adapters,
- and straight-through/crossover/rollover cables.

Change Log:

- 11-3-2014: original document – mkhonji.
- 12-3-2014: fixed typos in fig1 + added `no shutdown` in footnote 4 – mkhonji.

1 Introduction

Before we discuss why Internet Protocol (IP) is useful, we will first:

- recap what we have learned so far with regards to Ethernet broadcast domains and why the scalability problem that introduce,
- then introduce how IP and CIDR solve the scalability issues of broadcast domains.
- Additionally, we will briefly touch the main differences between the two most popular version of IP, namely versions 4 and 6 respectively.

1.1 Why is Ethernet not enough?

In previous labs 2 and 3, we learned that:

- In order for Ethernet to function, it *sometimes* has to broadcast messages to all connected devices. This introduced the notion of a *broadcast domain*.
- If we connect too many nodes or PCs to a set of inter-connected Ethernet switches, we will create a huge broadcast domain.
- Huge broadcast domains are a bad idea as they introduce efficiency problems (e.g. loss of bandwidth), as well as the linear growth of the MAC address table as number of communicating nodes increase. Additionally, security issues (e.g. some attacks thrive in broadcast domains).
- To reduce the size of broadcast domains, we should segregate them into smaller broadcast domains by:
  - Connecting nodes/PCs to isolated physical Ethernet switches. This way a PC can only talk to other PCs that are connected to the *same* switch (but cannot talk to other PCs that are connected to *different* switches).
  - Or by simply using Virtual LANs (VLANs) to decouple the broadcast domain segregation from the underlying physical network layout. This way, we can set up any broadcast domain boundaries by simply configuring the Ethernet switches (e.g. no need to physically disconnect/connect cables; thanks to IEEE 802.1Q tags).

However, the PCs in the different broadcast domains remained isolated and had no means of communication between each other. If we stop here, then large networks such as the Internet will not be possible. In order to allow large networks to exist, we need to find a method to inter-connect isolated broadcast domains while also maintaining broadcast domain boundaries.

The question is: can we allow PCs from different broadcast domains to communicate with each other in a scalable manner (i.e. while maintaining small-sized broadcast domains)? The answer to the question is "yes" and the mechanism is introduced in the subsequent sections of this lab.

1.2 What are IP and CIDR? How do they address the scalability problems?

The primary objective of the IP protocol is to facilitate an addressing scheme that is aggregatable in order to allow hierarchal routing.

Simply put, Class-less Inter-domain Routing (CIDR) is basically a set of rules that govern how should equipments behave based on information that is provided in the IP packets.

Figure 1 depicts an IP example with focus on the routing table. Notice how the number of IP addresses in the IP routing table is independent of the number of communicating PC.

Additionally, routers (i.e. R1 . . . 5) do not forward broadcasts. For example, if the PC with the IP address `10.0.0.1` wishes to send a packet to `10.1.0.3`, it only needs to forward the packet to R3's `eth1` interface by encapsulating it in a MAC frame with a destination MAC address equivalent to the MAC address of R3's `eth1` interface.

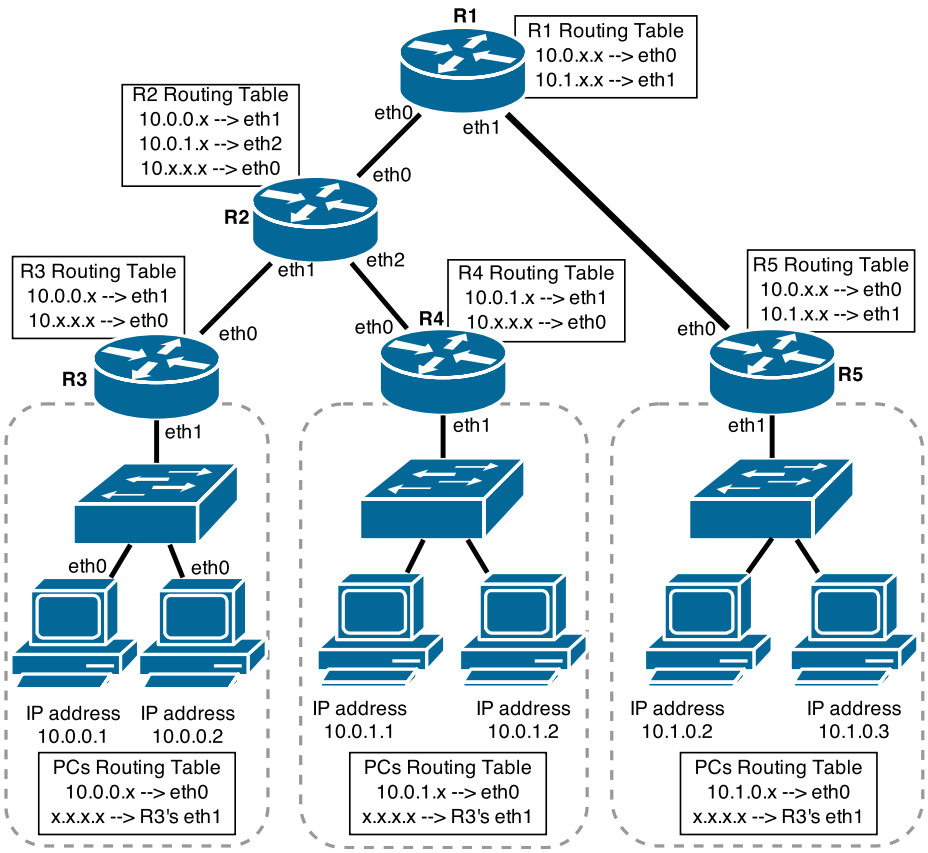

Figure 1: An IPv4 network with focus on the IPv4 routing table. **Note:** there are many other possible valid configurations, and that the figure shows interface IDs in the routing table, which is not always the case; in Ethernet networks (or any broadcast network), the next-hop is the IP address of the next-hop interface. However, interface IDs are shown for space efficiency.

Notes:

- This lab will focus on IP packets over Ethernet, but it doesn't have to

be this way. IP packet can be carried over a variety of layer 2 protocols such as FrameRelay, HDLC, TokenRing, etc.

- In theory, and in order for IP routing to function, intermediate devices that facilitate such routing (i.e. IP routers) don't need to have IP addresses for their own interfaces (e.g. R3's `eth1` interface doesn't need to have an IP address). All routers need is a routing table, connectivity with next hops at layer 2, and the ability to process IP packets accordingly (e.g. decrementing IP header's TTL, updating IP header's checksum, etc). The routing table needs to basically map destination IP-network addresses against a layer-2 next-hope layer 2 address (as in case of broadcast domains) or router's exit point interface (in case of point-to-point networks).

- However, configuring MAC addresses as next-hope addresses will make the IP routing table configuration dependent on the underlying layer 2 protocol. It is therefore desirable to follow a simpler approach were layer 2 protocol details are abstracted. This is also a problem if associated Ethernet network adapters are updated as their MAC addresses will change and therefore all routing entries need to be updated accordingly (which adds administrative overhead).

- Such layer 2 abstraction is achieved as follows:

  - Instead of populating routing tables with entries that map IP-network addresses against next-hop MAC addresses, next-hop IP addresses are used instead of next-hop MAC addresses.
  - In order to resolve the next-hop IP addresses into next-hop MAC addresses, the ARP protocol is used automatically by routers and nodes alike.

1.3 Different versions of IP

There are two version of IP in today's production networks, namely IPv4 and IPv6.

1.3.1 IPv4

The primary service that the IPv4 provides is an aggregatable source and destination IP addresses, which is a 32bit number. However, the IPv4 header provides additional options that are beyond the scope of this lab, which are (see Figure 2):

- 4 bits version — `0100` in binary to indicate that the subsequent bits to be interpreted as IPv4.

- 4 bits header length — the unit is in 32bits, and the minimal header length is 5×32 bits. For example, for the smallest header (i.e. one without optional fields), the value is `0101` in binary.

- 8 bits for type of service — this is used for Quality of Service (QoS) related topics which is beyond the scope of this lab. For the purpose of this lab, we will assume a value of 0 for this field.
- 16 bits total length — total number of octets for the IPv4 header as well as its payload.
- 16 bits packet identification — this is used for fragmentation purposes which is beyond the scope of this lab. For the purpose of this lab, we will assume a value of 0.
- 3 bits flags — this is used for fragmentation purposes which is beyond the scope of this lab. For the purpose of this lab, we will assume a value of 0.
- 13 bits fragment offsets — this is used for fragmentation purposes which is beyond the scope of this lab. For the purpose of this lab, we will assume a value of 0.
- 8 bits time to live — this is a number that indicates maximum number of routes this packet can cross before its death, which helps in avoiding eternally looping packets. Each router in the path decrements this value by 1, and once it reaches 0 the packet will be discarded.
- 8 bits next protocol — this is a number that identifies the protocol ID of the data in the payload of this IP packet.
- 16 bits header checksum — this allows routers and hosts to detect errors in packets and discard them if errors exist. The header checksum algorithm is simple and is as follows:

  > "The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero." — Source RFC791.

  For the purpose of verifying the checksum, the 16 bit one's complement sum of all 16 bit words is computed, including the header's checksum. If the answer is 0, the header is deemed valid.
- 32bit source IP address.
- 32bit destination IP address.
- Variable length options — this beyond the scope of this lab, and we will not include them in our headers.

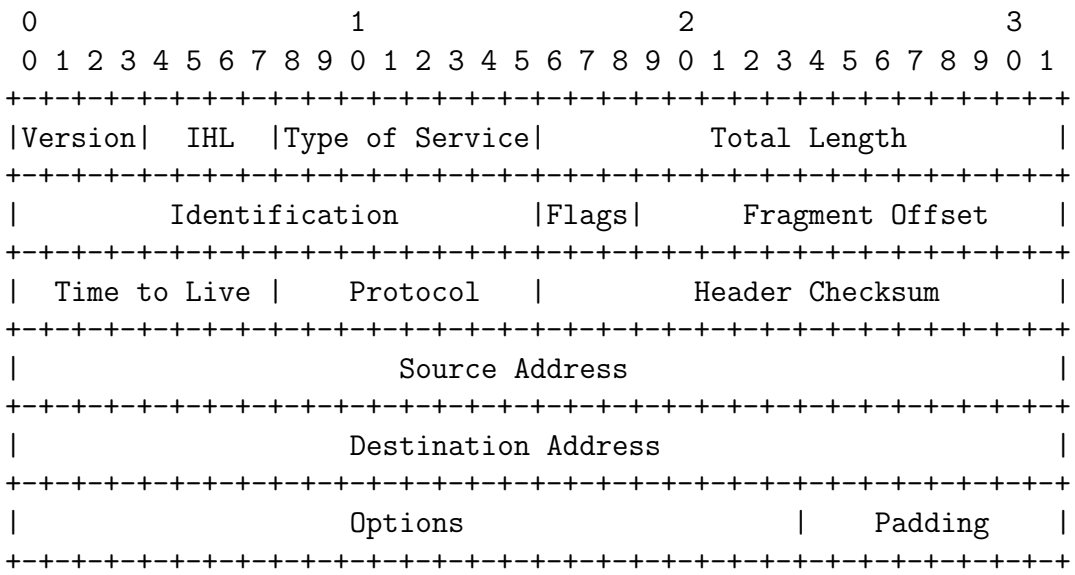

Figure 2: Internet Protocol (IP) version 4 header format — Source RFC791.

1.3.2 IPv6

One of the main problems with IPv4 is that its address space was too small as its designers did not expect IPv4 to gain such massive popularity. In the early 1990s, it was known that IPv4's 32bit address space is not enough.

IPv6 solves this problem by using 128 bit address space which provides a extremely larger address space.

Additionally, IPv6 is a much simpler protocol than IPv4 and fixes a number other issues with the IPv4 which is as follows (see Figure 3):

- Header length field is removed. As far as routers are concerned, the IPv6 header is fixed length. Optional fields are provided as extensions (beyond the scope).
- Removes the checksum field as it slows down routing (as routers would need to recompute the checksum every time they decrement the TTL in IPv4). The checksum is also redundant as they exist in layer 4 headers (e.g. TCP and UDP).
- Time to live is renamed into *hop limit* which is a more accurate terminology.

Because IPv6 addresses are long, they are represented in hex in groups of 16 bits. For example, the following is an example of an IPv6 address: `FD00:0000:0000:0000:0000:0000:0000:0001`. Additionally, and since sequences of 0s are common in IPv6 addresses, a sequence of consecutive 0s can be abbreviated by `::` as follows: `FD00::1`.

1.4 CIDR

In order to aggregate groups of IP addresses, subnet masks are used. Quite simply, a subnet mask is either:

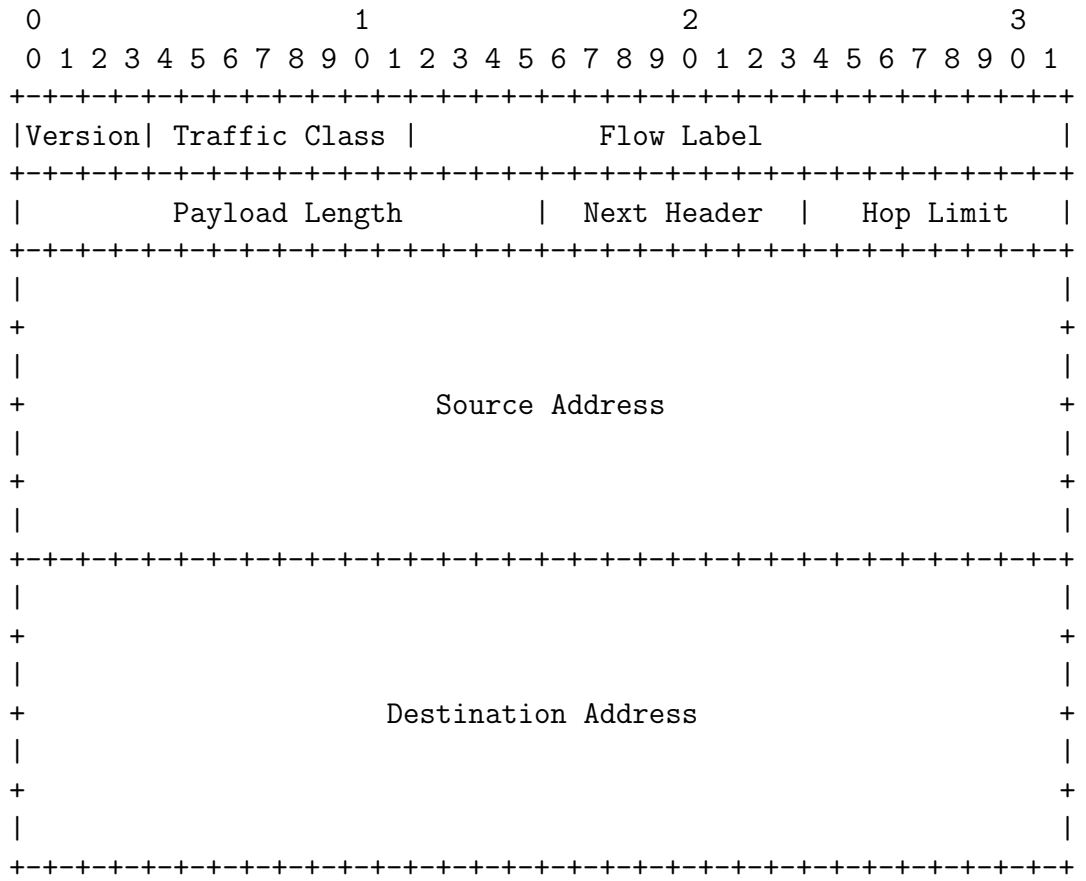

Figure 3: Internet Protocol (IP) version 6 header format — Source RFC2460.

- 32 bit number in IPv4,
- or 128 bit number in IPv6.

The subnet mask is a sequence of bits that help routers understand which portions of a given IP address to be used for the routing lookup (network part), and which parts to be ignored (host part).

For example, the following IPv4 addresses are represented in binary as follows:

- 10.0.0.1 is represented in binary as: `00001010.00000000.00000000.00000001`.
- 10.0.0.2 is represented in binary as: `00001010.00000000.00000000.00000010`.
- 10.0.254 is represented in binary as: `00001010.00000000.00000000.11111110`.

And if we wish to configure the router to aggregate all of the above addresses into a single routing table entry, we will need the following routing entry:

- Network IPv4 address: `00001010.00000000.00000000.00000000`.
- Subnet mask: `11111111.11111111.11111111.00000000`
- Exit interface or next hop address.

- Metric (the lower the more preferred the routing table entry).

The IPv4 router then, generally, performs the routing task upon the reception of every IPv4 packet:

- Extract the IPv4 destination address from the packet.
- Perform a bit-wise `AND` operation against longest subnet mask. This will result in returning the network IP address (where host bits are 0s).
- Compare the resultant network IP address against the network IP address of the routing entry that is associated to the applied subnet mask.
- If the resultant network IP address matches the routing table entry's network IP address, then execute it by forwarding the received packet towards the exit interface or the next-hop address.
- In case multiple routing entries are matched, execute the one with the lowest metric.
- In case there are multiple routing table entries with the *same* metric, then routers sometimes load balance across them (i.e. sometimes execute one, while some other times execute the other one). This is called equal-cost load balancing. Unequal-cost load balancing exists but it's beyond the lab's scope.

Historically (early 1980s), IPv4 addresses were aggregated using subnet masks that were derived from the leading bits of the subject IPv4 address. For example, IPv4 addresses that begin with the binary sequence `0` were considered to have a subnet mask of 255.0.0.0 (also known as class *A* addresses), and IPv4 addresses with the the binary sequence `10` were considered to have a subnet mask of 255.255.0.0 (also known as class *B* addresses), ... Similar procedure applies for classes *C, D, E*.

However, such class-full architecture was unsuccessful as it did not allow granular IPv4 address allocation, which was an apparent problem due to the IPv4 address space shortage. For this reason, the class-full architecture architecture was obsoleted in the favour of the class-less inter-domain architecture (CIDR).

With CIDR, routers and nodes are manually given subnet masks for all subject network IP addresses irrespective of their class. Therefore there is no need to determine the class of an IP. In fact, more recent RFCs (e.g. those published in late 1990s+) shy away from the term *class*. For example, RFC1918 refers to the private IP address range `10.0.0.0/8` as *24-bit* address blocks instead of obsoleted terminology *class A* address blocks which was found in older and obsoleted standards such as RFC1627.

As for IPv6, the same applies, except that IPv6 addresses are 128 bit numbers as opposed to IPv4's 32 bit addresses. Thus, subnet masks in IPv6 are 128 bits.

2 Lab Preparation

1. Connect a PC to the routers console port using a rollover cable[1].
2. Erase the configuration of the routers[2].
3. Connect a PC to the switches console ports using rollover cables.
4. Erase the configuration of the switch[3].
5. Run Wireshark on all involved lab PCs as depicted in Figure 4.
6. Physically connect the lab as depicted in Figure 4.

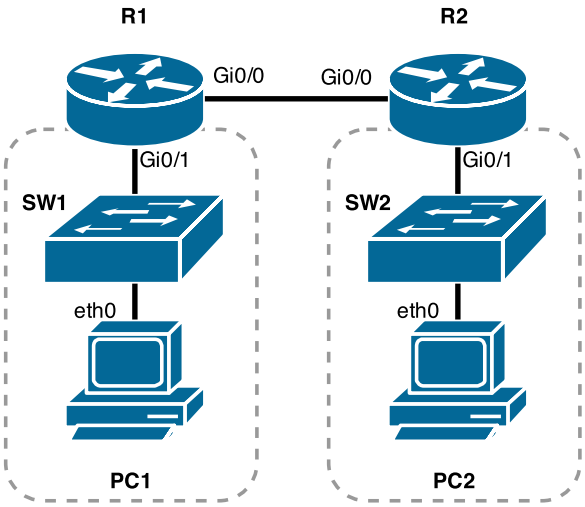

Figure 4: Physical lab topology.

3 Lab experiments

3.1 Basic IPv4 routing workflow

[50 points]

Note: When done, show your work to the lab engineer for grading purposes.

Perform the following tasks:

- Configure[4] `R1`'s interfaces as follows:

[1] If using Linux: `screen /dev/ttySx` were `x` is the serial interfaces ID that is connected to the console cable. If using Windows: Use Hyperterminal or Putty to connect to `COMx` ports.

[2] `enable`, `erase startup-config`, `reload`, and make sure to answer `no` to all yes/no questions while hitting *enter* for all `confirm` prompts.

[3] `enable`, `erase startup-config`, `delete vlan.dat`, `reload`, and answer `no` to all yes/no questions while hitting *enter* for all `confirm` prompts.

[4] `enable`, `configure terminal`, `interface Gi0/0`, `no shutdown`, `ip address 10.0.0.1 255.255.255.0`.

- GigabitEthernet 0/0: IPv4 address 10.0.12.1, subnet mask 255.255.255.0.
- GigabitEthernet 0/1: IPv4 address 10.0.1.1, subnet mask 255.255.255.0.

- Configure `R2`'s interfaces as follows:
  - GigabitEthernet 0/0: IPv4 address 10.0.12.2, subnet mask 255.255.255.0.
  - GigabitEthernet 0/1: IPv4 address 10.0.2.1, subnet mask 255.255.255.0.
- Configure[5] `PC1`'s interfaces as follows:
  - eth0: IPv4 address 10.0.1.2, subnet mask 255.255.255.0.
- Configure `PC2`'s interfaces as follows:
  - eth0: IPv4 address 10.0.2.2, subnet mask 255.255.255.0.

While monitoring Wireshark instances across the PCs, answer the following questions:

1. **Q:** View the routing table of `R1`[6], `R2`, `PC1`[7] and `PC2`. What are the routing table entries?
2. Send[8] ICMP Echo messages from `PC1` to `PC2`'s IP address 10.0.2.2.
   (a) **Q:** Did the `ping` command succeed?
   (b) **Q:** Did you see ARP messages? If you saw, whose MAC address was attempted to be resolved?
   (c) **Q:** Did you see ICMP messages crossing the network? What paths did they follow?
   (d) **Q:** If you saw ICMP messages, what are the source and destination MAC address?
   (e) **Q:** View[9] the MAC address table of switches `SW1`, do you see `PC2`'s MAC address?
   (f) **Q:** View the MAC address table of switches `SW2`, do you see `PC1`'s MAC address?
   (g) **Q:** What do you conclude from this observation?
3. Send ICMP Echo messages from `PC2` to `PC1`'s IP address 10.0.1.2.
   (a) **Q:** Did the `ping` command succeed?
   (b) **Q:** Did you see ARP messages? If you saw, whose MAC address was attempted to be resolved?
   (c) **Q:** Did you see ICMP messages crossing the network? What paths did they follow?

[5] `ifconfig eth0 10.0.1.2/24`
[6] `show ip route`
[7] `route -n`
[8] `ping 10.0.2.2`
[9] `show mac address-table`

(d) **Q:** If you saw ICMP messages, what are the source and destination MAC address?

(e) **Q:** View the MAC address table of switches `SW1`, do you see `PC2`'s MAC address?

(f) **Q:** View the MAC address table of switches `SW2`, do you see `PC1`'s MAC address?

(g) **Q:** What do you conclude from this observation?

4. Add the default routes to `PC1` and `PC1` as follows:

   - `PC1`: `route add -net 0.0.0.0/0 gw 10.0.1.1`
   - `PC2`: `route add -net 0.0.0.0/0 gw 10.0.2.1`

5. **Q:** View the routing table of `PC1` and `PC2`. Do you see any new added routes?

6. Send ICMP Echo messages from `PC1` to `PC2`'s IP address 10.0.2.2 again.

   (a) **Q:** Did the `ping` command succeed?

   (b) **Q:** Did you see ARP messages? If you saw, whose MAC address was attempted to be resolved?

   (c) **Q:** Did you see ICMP messages crossing the network? What paths did they follow?

   (d) **Q:** If you saw ICMP messages, what are the source and destination MAC address?

   (e) **Q:** View the MAC address table of switches `SW1`, do you see `PC2`'s MAC address?

   (f) **Q:** View the MAC address table of switches `SW2`, do you see `PC1`'s MAC address?

   (g) **Q:** What do you conclude from this observation?

7. Send ICMP Echo messages from `PC2` to `PC1`'s IP address 10.0.1.2 again.

   (a) **Q:** Did the `ping` command succeed?

   (b) **Q:** Did you see ARP messages? If you saw, whose MAC address was attempted to be resolved?

   (c) **Q:** Did you see ICMP messages crossing the network? What paths did they follow?

   (d) **Q:** If you saw ICMP messages, what are the source and destination MAC address?

   (e) **Q:** View the MAC address table of switches `SW1`, do you see `PC2`'s MAC address?

   (f) **Q:** View the MAC address table of switches `SW2`, do you see `PC1`'s MAC address?

   (g) **Q:** What do you conclude from this observation?

8. Using Cisco IOS commands[10], add the following routes to R1's and R2's routing tables:

   - R1:
     - Network IP: 10.0.2.0.
     - Netmask: 255.255.255.0
     - Nexthop IP address: 10.0.12.2
   - R2:
     - Network IP: 10.0.1.0.
     - Netmask: 255.255.255.0
     - Nexthop IP address: 10.0.12.1

9. **Q:** View the routing table of `R1` and `R2`. Do you see any new added routes?

10. Send ICMP Echo messages from `PC1` to `PC2`'s IP address 10.0.2.2 again.

    (a) **Q:** Did the `ping` command succeed?

    (b) **Q:** Did you see ARP messages? If you saw, whose MAC address was attempted to be resolved?

    (c) **Q:** Did you see ICMP messages crossing the network? What paths did they follow?

    (d) **Q:** If you saw ICMP messages, what are the source and destination MAC address?

    (e) **Q:** View the MAC address table of switches `SW1`, do you see `PC2`'s MAC address?

    (f) **Q:** View the MAC address table of switches `SW2`, do you see `PC1`'s MAC address?

    (g) **Q:** What do you conclude from this observation?

11. Send ICMP Echo messages from `PC2` to `PC1`'s IP address 10.0.1.2 again.

    (a) **Q:** Did the `ping` command succeed?

    (b) **Q:** Did you see ARP messages? If you saw, whose MAC address was attempted to be resolved?

    (c) **Q:** Did you see ICMP messages crossing the network? What paths did they follow?

    (d) **Q:** If you saw ICMP messages, what are the source and destination MAC address?

    (e) **Q:** View the MAC address table of switches `SW1`, do you see `PC2`'s MAC address?

    (f) **Q:** View the MAC address table of switches `SW2`, do you see `PC1`'s MAC address?

    (g) **Q:** What do you conclude from this observation?

[10] `enable`, `configure terminal`, `ip route 10.0.2.0 255.255.255.0 10.0.12.2`

3.2 A closer look into the IPv4 packet

[50 points]

Note: When done, show your work to the lab engineer for grading purposes.

Using the knowledge acquired from the previous sections, send an IP packet from `PC1` to `PC2` (or `PC2` to `PC1` depending on your team's orientation) with the following specifications:

- Craft a MAC frame accordingly (i.e. choose the approperiate source MAC address, destination MAC address, type (0x0800 for IP).
- Inside the payload of the MAC frame, send the following IP packet:
  - Version: 4,
  - IHL: 5 32bit words (i.e. minimal header length),
  - Type of Service: set to 0 (off-topic),
  - Total length: calculate it according to the header length + payload length,
  - Identification: set to 0 (off-topic),
  - Flags: set to 0 (off-topic),
  - Fragment offset: set to 0 (off-topic),
  - Time to Live: choose the right TTL value such that it's not too small to cause death of packets while being routed,
  - Protocol: for this lab, set it to any arbitrary number (but in later labs we will set it according to the next-layer's protocol),
  - Header cehcksum: calculate it according to the provided algorithm from RFC791 as quoted in earlier sections,
  - Source address: set it accordingly,
  - Destination address: set it accordingly.
  - Payload: "Hello" (without double quotes; using the ASCII table in hex, it would be `0x48, 0x65, 0x6C, 0x6C, 0x6F`).

When done, hit `enter` if using `scapy`, or compile it if using `macpacketsender.c` using `gcc macpacketsender.c -o macpacketsender`, then execute it by `./macpacketsender`.

Upon the successful execution of this task, the receiver PC should be able to receive the message. Make sure to share your findings to the lab engineer upon completion.

3.3 Extra questions: IPv6

In this section, you will be able to use your laptop to inject MAC packets into various VLANs.

Note: when done, show your answers to the lab engineer for feedback. If the lab time is not enough, take your time and submit it in a later time. Although not graded, it can

3.3.1 Basic IPv6 routing workflow

1. Enable[11] IPv6 routing on routers `R1` and `R2`.
2. Assign IPv6 IP addresses to the router interfaces as follows:
   - `R1`'s `Gi0/0`: `FD00:12::1/64`[12]
   - `R1`'s `Gi0/1`: `FD00:1::1/64`
   - `R2`'s `Gi0/0`: `FD00:12::2/64`
   - `R2`'s `Gi0/1`: `FD00:2::1/64`
   - `PC1`'s `eth0`: `FD00:1::2/64`[13]
   - `PC2`'s `eth0`: `FD00:2::2/64`
3. Add the routing table entries as follows:
   - `R1`'s routing table:
     - IPv6 Network IP: `FD00:2::`
     - IPv6 Netmask: 64 bits
     - Nexthop: `FD00:12::2`
   - `R2`'s routing table:
     - IPv6 Network IP: `FD00:1::`
     - IPv6 Netmask: 64 bits
     - Nexthop: `FD00:12::1`
   - `PC1`'s routing table (a default route entry):
     - IPv6 Network IP: `::`
     - IPv6 Netmask: 0 bits
     - Nexthop: `FD00:1::1`
   - `PC2`'s routing table (a default route entry):
     - IPv6 Network IP: `::`
     - IPv6 Netmask: 0 bits
     - Nexthop: `FD00:2::1`
4. Verify the configurations:
   - `R1`'s routing table[14].
   - `R2`'s routing table.
   - `PC1`'s routing table[15].
   - `PC2`'s routing table.
5. Send an ICMP Echo from `PC1` to `PC2`'s IPv6 address[16]

You should then able to successfully receive IPv6 ICMP Echo and Echo-Reply messages across `PC1` and `PC2`.

[11] `enable`, `config terminal`, `ipv6 unicast-routing`
[12] `enable`, `config terminal`, `interface gi0/0`, `ipv6 address FD00:12::1/64`
[13] `ifconfig eth0 inet6 add FD00:1::2/64`
[14] `enable`, `show ipv6 route`
[15] `enable`, `route -A inet6 -n`
[16] `ping6 FD00:2::2`

3.3.2 A closer look into IPv6 packets

This is an easy task as the IPv6 header contains a lot less fields, and particularly no Checksum field.

Repeat the task from Section 3.2 except for applying IPv6's packet format as depicted in Figure 3