

CSE123 Computer Networks
Winter 2022 Project 2a
DESIGN A SIMPLE ROUTER

Project 2 TA - Aritra Basu

Acknowledgement: Roger Chen

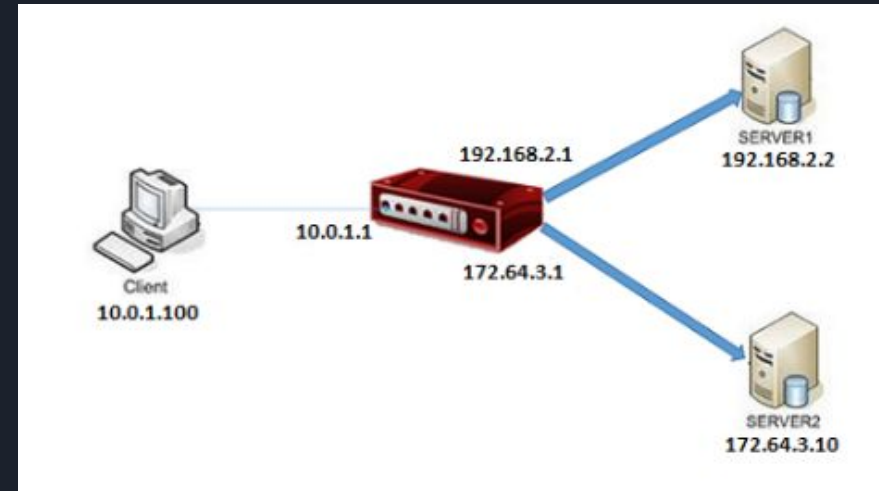


Project Setup

- Get the invitation to access the skeleton code on Canvas.
- Once you accept the invitation, you will get a private repo with the starter code.
- You will be working on a VM for this project. So remember to push your code frequently to GitHub because if somehow the VM image gets corrupted, you cannot recover your code.
- Although you will be submitting your code on Gradescope, there is no autograder configured. We will be testing your code in the VM environment. A sample solution binary has been provided to you so that you can check if your code's output is matching with the expected output.
- All the coding will be done in C.

Overview

- In this assignment you will write a simple router for a given static network topology and routing table.
- Your router will receive raw Ethernet frames and need to process these packets to forward them on the correct outgoing interface.
- You are responsible for implementing the logic for handling the incoming Ethernet frames.
- A VM will also be provided to you with all the suitable environment settings.





Mininet

- This assignment runs on top of Mininet, which was developed at Stanford.
- Mininet is a network emulation tool which allows you to emulate a network topology on a single Linux kernel by providing the necessary isolation between the emulated hosts and routers.
- Mininet uses lightweight virtualization to make a single system look like a complete network running on the same kernel.
- You don't have to know how Mininet works to complete this assignment, but more information about Mininet (if you are curious) is available [here](#).



Getting Started

- Follow the instructions provided in the Project 2a Specs document to install VirtualBox.
- Download the virtual machine image `ubuntu1404.ova`.
- Configure VirtualBox as explained in the instructions and boot the VM image.
- Run the VM and login using username “mininet” and password “mininet”.
- After logging in, remember to run the command “`sudo dhclient eth1`” to get an IP address on the host-only network. You must do this every time you shut down your VM and log back in because without this IP, you cannot SSH into your VM.
- You can shut down your VM when you are not working on your project.



Getting Started

- You might find it more convenient to SSH into your VM instead of working inside of it.
- To do this, run “ifconfig”.
- Look for the IP address of the “eth1” interface and note it down.
- Logout of your vm by typing “exit”. This should take you back to the login screen.
- Minimize your VM window. Do not shutdown the machine.
- In a terminal, run “ssh mininet@<the ip address you noted earlier>”. Enter “mininet” when prompted for password.



Getting Started

- You will need 3 terminals for this project, one for each of the following (please follow the instructions in the Project 2a Specs document)
 - POX
 - Mininet
 - Router (for running your code)
- You can use [screen](#) to run multiple shell sessions on the same window.
- If you are not comfortable with screen, simply launch multiple SSH sessions from different terminals (you can use a SSH client like MobaXterm or iTerm for this).



Getting Started

- In the third terminal, “cd” into the starter code directory.
- Execute “make clean”, followed by “make”.
- `sr_solution2a` is a reference solution that has been provided to you to see what the expected behavior of your code should be. Run this binary file.
- Execute ping commands from the terminal where mininet is running (using the instructions in the Project 2a Specs document). You will be able to view the expected output.
- Next, run the binary from the starter code “sr” (please follow the instructions in the document). Since the starter code cannot handle ping commands, you will get unexpected results.
- Now, you are ready to start coding!



Few Common Errors

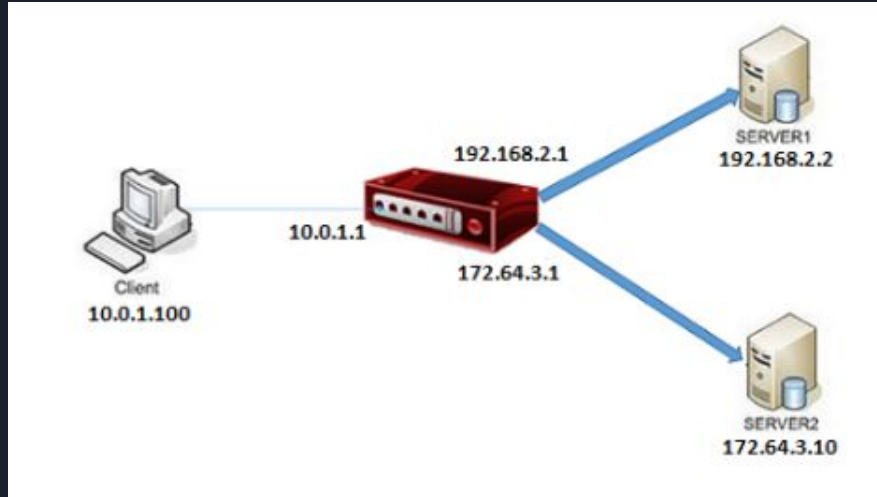
- Routing table not consistent with hardware

If you see this error, you need to restart mininet. Stop your router binary, mininet and the POX controller. Then run them once again in the correct order (POX → mininet → router binary).

- RTNETLINK answers: File exists

This is not exactly an error. You may see this when you run "sudo dhclient eth1". It means that an IP has already been associated with your “eth1” interface. You can run “ifconfig” to check the IP.

Topology



- The topology is setup using two files:
 - IP_CONFIG
 - rtable

- **> cat ~/cse123-p2/IP_CONFIG**

```
server1 192.168.2.2
server2 172.64.3.10
client 10.0.1.100
sw0-eth1 192.168.2.1
sw0-eth2 172.64.3.1
sw0-eth3 10.0.1.1
```

- **> cat ~/cse123-p2/router/rtable**

```
192.168.2.2 192.168.2.2 255.255.255.255 eth1
172.64.3.10 172.64.3.10 255.255.255.255 eth2
10.0.1.100 10.0.1.100 255.255.255.255 eth3
```



What should your router support?

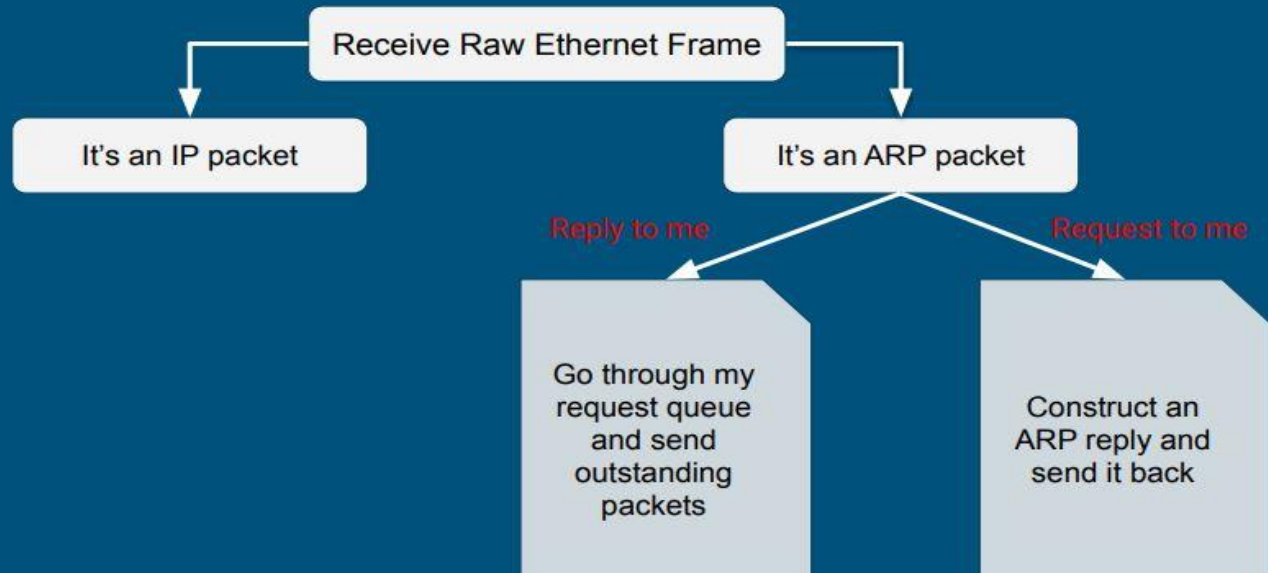
- **Route Ethernet frames between the client (10.0.1.100) and the two HTTP servers (192.168.2.2 & 172.64.3.10).**
- **Your router should be able to handle two types of packets:**
 - **ARP Packets: Requests & Replies**
 - **IP Packets: ICMP**



How do you differentiate packets?

- “ether_type” to your rescue!
- Check the type of the packet received inside the “sr_handlepacket()” function using “ethertype” defined in the sr_utils.c file.
- **TIP:** You will also find a number of structs defined for you in the sr_protocol.h file. Please review this file properly. You will be using this extensively.
 - sr_ethernet_hdr_t
 - sr_arp_hdr_t
 - sr_ip_hdr_t
 - sr_icmp_t11_hdr_t

ARP FLOW





If it is an ARP packet

- Verify the length of the packet.
- Check the “ar_op” (opcode) variable of the ARP header to check if it is an “arp_op_request” or an “arp_op_reply”.
- If it is an ARP Request for the router, construct an ARP reply. You can reuse the same packet buffer by updating the necessary fields. Then call “sr_send_packet()” to send the ARP reply.
- If it is an ARP Reply, send all the packets in the request queue that were waiting for that ARP reply to their next-hop (this point will become clearer after the next few slides).
- TIP: “sr_get_interface()” defined in the sr_if.c file can be used to get the details of an interface (the MAC address and the IP address associated with an interface) - you will find this useful!

ARP Request → ARP Reply

ARP Request

- > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
- > Source: 42:a5:f3:5b:de:9b (42:a5:f3:5b:de:9b)
- Type: ARP (0x0806)

Address Resolution Protocol (request)

Hardware type: Ethernet (1)
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: request (1)
Sender MAC address: 42:a5:f3:5b:de:9b (42:a5:f3:5b:de:9b)
Sender IP address: 10.0.1.100
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
Target IP address: 10.0.1.1

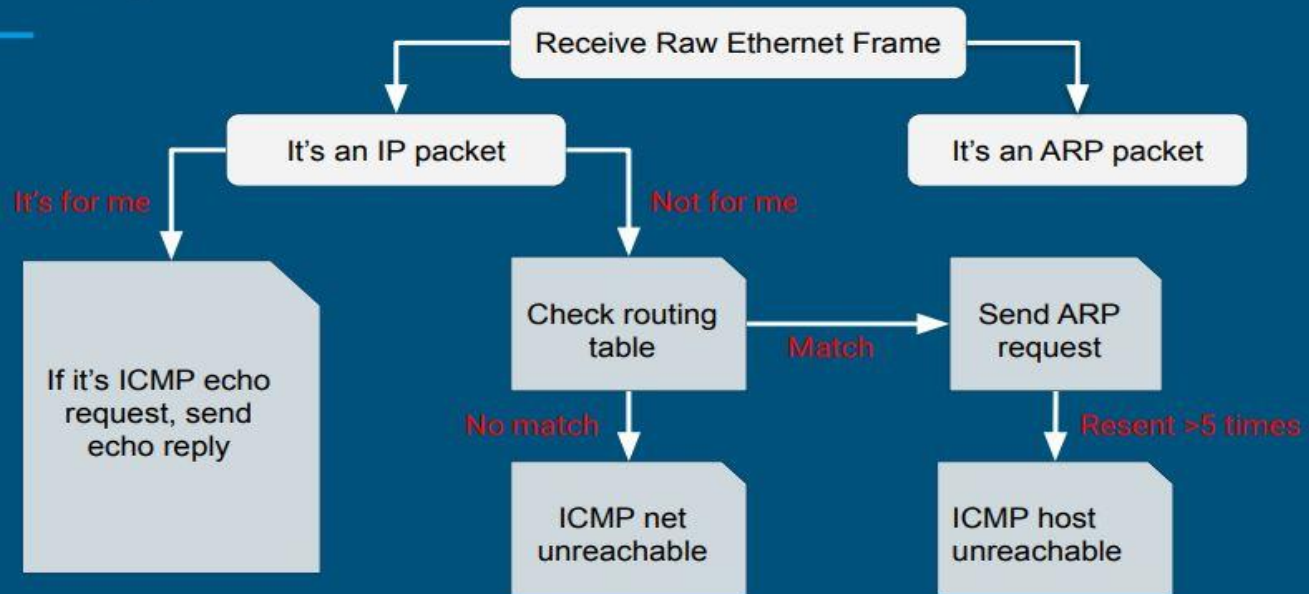
ARP Reply

- > Destination: 42:a5:f3:5b:de:9b (42:a5:f3:5b:de:9b)
- > Source: 8e:b2:d1:d0:f6:a4 (8e:b2:d1:d0:f6:a4)
- Type: ARP (0x0806)

Address Resolution Protocol (reply)

Hardware type: Ethernet (1)
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: reply (2)
Sender MAC address: 8e:b2:d1:d0:f6:a4 (8e:b2:d1:d0:f6:a4)
Sender IP address: 10.0.1.1
Target MAC address: 42:a5:f3:5b:de:9b (42:a5:f3:5b:de:9b)
Target IP address: 10.0.1.100

IP FLOW





If it is an IP packet

- Sanity check the packet by validating the IP header:
 - Verify that it is a IPv4 packet.
 - Verify the IP header length - “ip_hl” (must be 20 bytes - refer to RFC791).
 - Verify IP packet length - “ip_len” (must be greater than the IP header length).
 - Verify the checksum.
- TIP: While verifying the checksum, it is very important that you set the “ip_sum” (checksum) field to 0 before calling the cksum() function defined in sr_utils.c (as per RFC791).
- TIP: Remember that the IP checksum is only computed on the IP header. However, the ICMP checksum is computed over the ICMP header + ICMP data.



If it is an IP packet (destined for the router)

- Check the IP protocol “ip_p”.
- If it is ICMP echo request (type 8), then generate ICMP echo reply (type 0).
- Otherwise, ignore the packet.
- **TIP:** The data field of an ICMP echo request does not have a fixed length. Its length is determined from the total length field in the IP header. The router should copy the complete data field from an echo request to the corresponding echo reply. Easy solution is to use the same packet buffer for the reply and modify the packet headers accordingly.
- **TIP:** Remember that any time you make modifications to an IP / ICMP header, you need to recompute the checksum. So, the sequence will be like: 1) Update the IP / ICMP header → 2) Set the checksum field to 0 → 3) Calculate the new checksum and set it to the checksum field.

> Destination: 8e:b2:d1:d0:f6:a4 (8e:b2:d1:d0:f6:a4)
> Source: 42:a5:f3:5b:de:9b (42:a5:f3:5b:de:9b)
Type: IPv4 (0x0800)

▼ Internet Protocol Version 4, Src: 10.0.1.100, Dst: 10.0.1.1
0100 = Version: 4
.... 0101 = Header Length: 20 bytes (5)

> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 84
Identification: 0x4415 (17429)
> Flags: 0x40, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0

Time to Live: 64

Protocol: ICMP (1)

Header Checksum: 0xe02f [validation disabled]

[Header checksum status: Unverified]

Source Address: 10.0.1.100

Destination Address: 10.0.1.1

▼ Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0

Checksum: 0x68eb [correct]

[Checksum Status: Good]

Identifier (BE): 4135 (0x1027)

Identifier (LE): 10000 (0x2710)

Sequence Number (BE): 1 (0x0001)

Sequence Number (LE): 256 (0x0100)

[Response frame: 4]

Timestamp from icmp data: Feb 3, 2022 18:17:12.000000000 Pacific Standard Time

[Timestamp from icmp data (relative): 0.117039000 seconds]

> Data (48 bytes)

ICMP Echo Request

> Destination: 42:a5:f3:5b:de:9b (42:a5:f3:5b:de:9b)
> Source: 8e:b2:d1:d0:f6:a4 (8e:b2:d1:d0:f6:a4)
Type: IPv4 (0x0800)

▼ Internet Protocol Version 4, Src: 10.0.1.1, Dst: 10.0.1.100
0100 = Version: 4
.... 0101 = Header Length: 20 bytes (5)

> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 84
Identification: 0x4415 (17429)
> Flags: 0x40, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0

Time to Live: 255

Protocol: ICMP (1)

Header Checksum: 0x212f [validation disabled]

[Header checksum status: Unverified]

Source Address: 10.0.1.1

Destination Address: 10.0.1.100

▼ Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
Code: 0

Checksum: 0x70eb [correct]

[Checksum Status: Good]

Identifier (BE): 4135 (0x1027)

Identifier (LE): 10000 (0x2710)

Sequence Number (BE): 1 (0x0001)

Sequence Number (LE): 256 (0x0100)

[Request frame: 3]

[Response time: 0.064 ms]

Timestamp from icmp data: Feb 3, 2022 18:17:12.000000000 Pacific Standard Time

[Timestamp from icmp data (relative): 0.117103000 seconds]

> Data (48 bytes)

ICMP Echo Reply



If it is an IP packet (not destined for the router)

- Check the TTL value - “ip_ttl”. If $TTL \leq 1$, send ICMP Time Exceeded message (type 11, code 0).
- Look up the next-hop IP address by doing exact comparison of the address in the routing table and the packet’s destination IP (do not perform longest-prefix matching in Project 2a).
- If an entry does not exist, send ICMP Destination Net Unreachable message (type 3, code 0).
- Otherwise, decrement TTL by 1 and recompute the packet checksum over the modified header.
- Store the packet in the request queue and send ARP request to the next-hop IP to determine the MAC.
- If no response is received, you must retransmit the ARP request 5 times (one per second). After 5 seconds, if still there is no response, send ICMP Destination Host Unreachable message (type 3, code 1).
- Once the ARP reply is received, update the Ethernet header of all the packets in the request queue that were waiting on that ARP reply from their next-hop IP and forward them using the outgoing interface, i.e., the interface on which the ARP reply was received. (do not cache the ARP reply in Project 2a)



ICMP Error Messages

- **You are dealing with 3 type of ICMP error messages in Project 2a:**
 - **ICMP Destination Net Unreachable - Type: 3, Code 0**
 - **ICMP Destination Host Unreachable - Type: 3, Code 1**
 - **ICMP Time Exceeded - Type: 11, Code 0**
- **Whenever you create an ICMP error message, please keep these points in mind:**
 - **The length of the data portion of the ICMP error message is 28.**
 - **The data portion of the ICMP error message must contain the IP header of the original packet (20 bytes) + “the first 8 bytes of the original datagram's data” (as per RFC792).**
 - **What this means is that you start from the beginning of the IP header of the original packet that triggered the ICMP error and then just copy over the next 28 bytes to the data field of the ICMP error packet.**



Tips

- Where do you start coding?
 - `sr_handlepacket()` method in the `sr_router.c` file
- What are the functions that you have to fill in?
 - `sr_handlepacket()` method in the `sr_router.c` file.
Note: This method gets called for every packet that goes through the router.
 - `sr_arpcache_sweepreqs()` method in the `sr_arpcache.c` file.
Note: This method gets called every 1 second. (you can implement your ARP request retransmission here)
- Take advantage of the print functions available in the “`sr_utils.c`” file for printing out the network header information from your packets. In fact, these functions can give you some ideas about how to access and manipulate the packet headers in your code.



Tips

- In the file “sr_router.h”, struct “sr_instance” defines the context of the router.
- In the file “sr_protocol.h”, you will find all the convenient structs for accessing the packet headers.
- In the file “sr_if.h”, you will find the method for getting information about the router’s interfaces. (the method name was already mentioned in an earlier slide!)
- The file “sr_rt.h” contains information about the struct that stores the rtable information (you will require this when you need to find the matching entry in the routing table for a packet).
- The file “sr_utils.c” has quite a few utility functions, such as cksum(), ethertype() and ip_protocol() that you will find to be very useful.



Tips

- **What are the Endianness for Networks Protocols vs Linux?**
 - **Linux uses Little Endian.**
 - **Network Protocols use Big Endian.**
 - **Convert the endianness using `ntohl()` / `htonl()`.**
- **What is the order in which I should write the code?**
 - **Handle ARP requests**
 - **Handle IP packets to the router's interface**
→ At this point, ping to the router's interface should work.
 - **Handle IP packets that the router must forward**
 - **Handle ARP replies (this goes hand-in-hand with forwarding IP packets)**
→ At this point, ping from the client to the servers should work.