



# SIMPLE ROUTER - project 2a



Roger Cheng



# MININET

---

- It is a network emulation orchestration system which runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code.
- Please use “#mininet> help” command or refer the documentation online at “<http://mininet.org/walkthrough/>” for more information.
- Mininet is also a simple way to create an SDN prototype to simulate a network topology. You can read more about SDN [here](#).

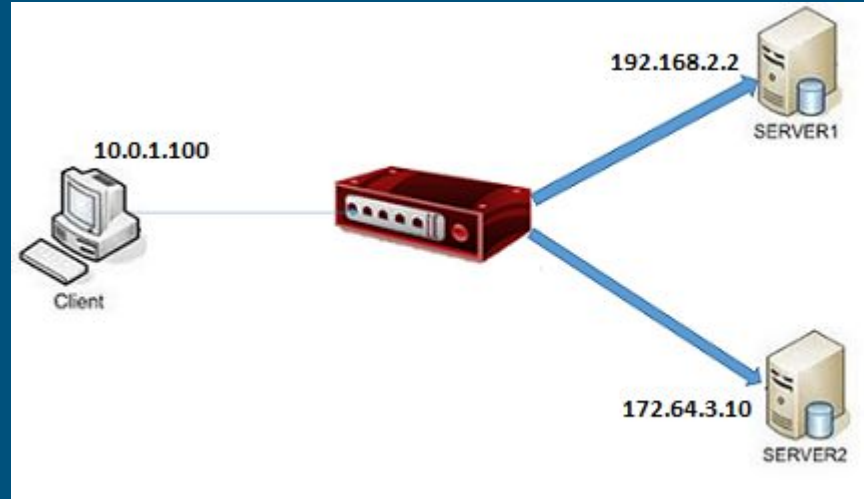
# GETTING STARTED

---

- Boot the VM image provided on Virtual Box/Vmware fusion/player and use mininet/mininet (username/password).
- You need 3 terminals, One each for mininet, pox controller and another one for running the code.
- In the third terminal, “cd” into the starter code directory, execute “make clean” -> “make” -> ./sr
- ~/cse123\_p2/sr\_solution is a reference solution for you to see what the expected behavior is.
- Execute ping/traceroute commands from first terminal where mininet is running (at the mininet prompt) ..

# TOPOLOGY

- Github repository has been setup with skeleton code for this project and a VM will be provided with suitable environment settings.
- Again, Coding will done in C.
- The topology is setup using IP\_CONFIG and rtable files. Please check them out.



Router interfaces:

```
eth3    HWaddr c2:e3:78:bf:71:f9
        inet addr 10.0.1.1
eth2    HWaddr 26:e4:1f:26:49:ab
        inet addr 172.64.3.1
eth1    HWaddr 56:01:ec:fb:34:09
        inet addr 192.168.2.1
```

# PROJECT OVERVIEW

---

- Route Ethernet frames between the client (10.0.1.100) and the HTTP servers (192.168.2.2 & 172.64.3.10).
- Router should handle two types of packets:
  - ARP packets: requests & replies
  - IP Packets: ICMP & TCP/UDP
- Where do you start coding ?
  - `sr_handlepacket()` method in `sr_router.c`

# HOW TO DIFFERENTIATE PACKETS?

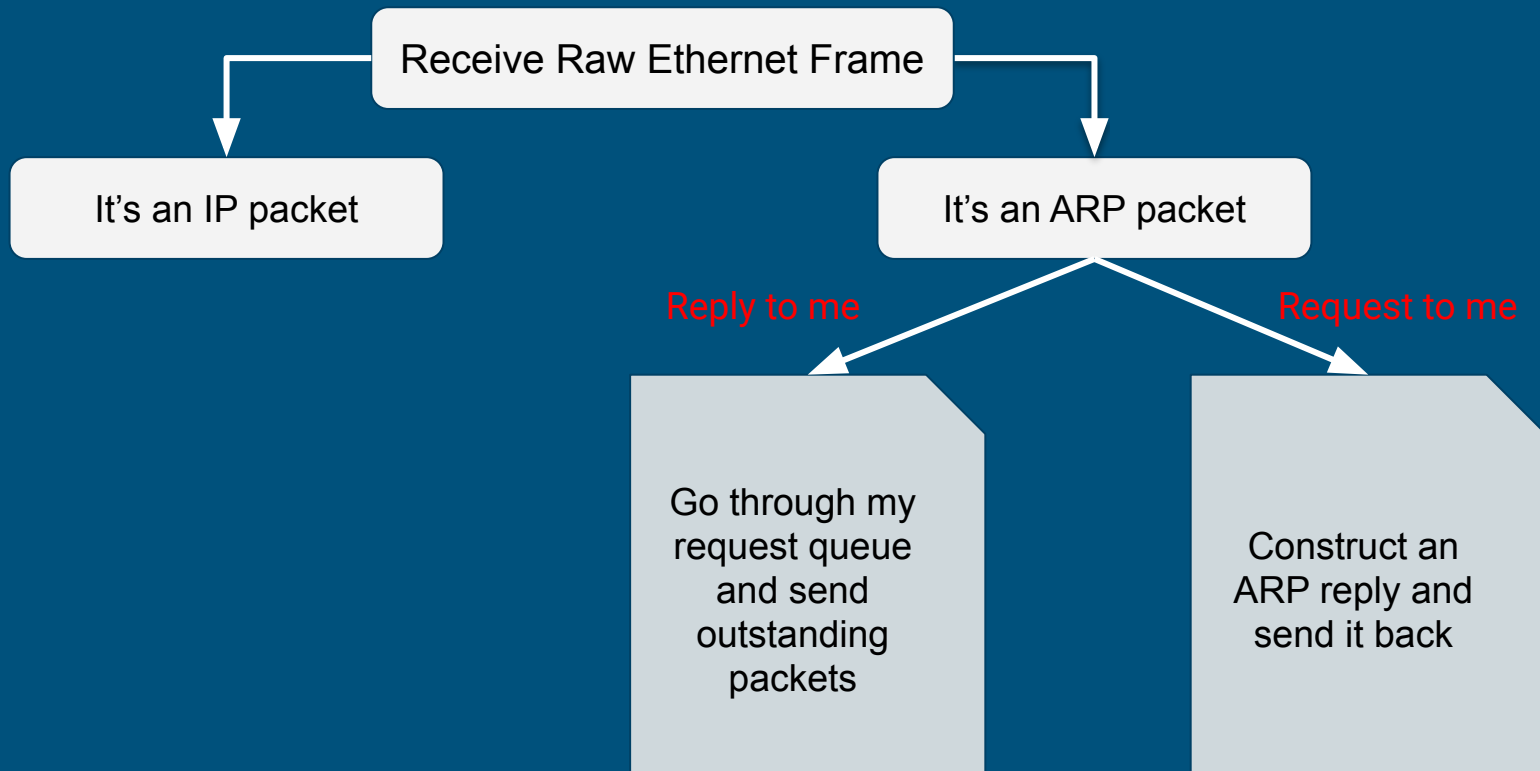
---

- “ether\_type” is your friend. Check the ‘packet’ received in sr\_handlepacket() using ethertype function defined in sr\_utils.c.
- You can find below definitions in sr\_protocol.h

```
enum sr_ethertype {  
    ethertype_arp = 0x0806,  
    ethertype_ip = 0x0800,  
};
```

```
struct sr_ethernet_hdr  
{  
    #ifndef ETHER_ADDR_LEN  
    #define ETHER_ADDR_LEN 6  
    #endif  
    uint8_t ether_dhost[ETHER_ADDR_LEN]; /* destination ethernet address */  
    uint8_t ether_shost[ETHER_ADDR_LEN]; /* source ethernet address */  
    uint16_t ether_type; /* packet type ID */  
} __attribute__((packed));  
typedef struct sr_ethernet_hdr sr_ethernet_hdr_t;
```

# ARP FLOW



# IF IT'S AN ARP PACKET

---

- Verify the length of the packet.
- Use `sr_get_interface` (in `sr_if.c`) to get the interface record. Get the mac address of the destination IP address/interface IPaddress from the record.
- Check the 'opcode' variable of the ARP header and see if it is `arp_op_request` or `arp_op_reply`.
- If it is an ARP Request update all the fields of the packet and use `sr_send_packet()` to send an ARP reply.
- If it is a Reply, send all the packets in the queue to the destination.

# Arp Reply example

Router Interface Request  
Received

eth3

HWaddr 11:22:33:44:55:66

inet addr 10.0.3.1

## Incoming Message

### **ETHERNET header:**

destination: FF:FF:FF:FF:FF:FF

source: AA:AA:AA:11:11:11

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 1

sender hardware address:

AA:AA:AA:11:11:11

sender ip address: 10.0.3.2

target hardware address:

00:00:00:00:00:00

target ip address: 10.0.3.1

## Reply Message

### **ETHERNET header:**

destination: AA:AA:AA:11:11:11

source: 11:22:33:44:55:66

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 2

sender hardware address:

11:22:33:44:55:66

sender ip address: 10.0.3.1

target hardware address:

AA:AA:AA:11:11:11

target ip address: 10.0.3.2

# Arp Reply example

Router Interface Request  
Received

eth3

HWaddr 11:22:33:44:55:66

inet addr 10.0.3.1

## Incoming Message

### **ETHERNET header:**

destination: FF:FF:FF:FF:FF:FF

source: AA:AA:AA:11:11:11

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 1

sender hardware address:

AA:AA:AA:11:11:11

sender ip address: 10.0.3.2

target hardware address:

00:00:00:00:00:00

target ip address: 10.0.3.1

## Reply Message

### **ETHERNET header:**

destination: AA:AA:AA:11:11:11

source: 11:22:33:44:55:66

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 2

sender hardware address:

11:22:33:44:55:66

sender ip address: 10.0.3.1

target hardware address:

AA:AA:AA:11:11:11

target ip address: 10.0.3.2

# Arp Reply example

Router Interface Request  
Received

eth3

HWaddr 11:22:33:44:55:66

inet addr 10.0.3.1

## Incoming Message

### **ETHERNET header:**

destination: FF:FF:FF:FF:FF:FF

source: AA:AA:AA:11:11:11

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 1

sender hardware address:

AA:AA:AA:11:11:11

sender ip address: 10.0.3.2

target hardware address:

00:00:00:00:00:00

target ip address: 10.0.3.1

## Reply Message

### **ETHERNET header:**

destination: AA:AA:AA:11:11:11

source: 11:22:33:44:55:66

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 2

sender hardware address:

11:22:33:44:55:66

sender ip address: 10.0.3.1

target hardware address:

AA:AA:AA:11:11:11

target ip address: 10.0.3.2

# Arp Reply example

Router Interface Request  
Received

eth3

HWaddr 11:22:33:44:55:66

inet addr 10.0.3.1

## Incoming Message

### **ETHERNET header:**

destination: FF:FF:FF:FF:FF:FF

source: AA:AA:AA:11:11:11

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 1

sender hardware address:

AA:AA:AA:11:11:11

sender ip address: 10.0.3.2

target hardware address:

00:00:00:00:00:00

target ip address: 10.0.3.1

## Reply Message

### **ETHERNET header:**

destination: AA:AA:AA:11:11:11

source: 11:22:33:44:55:66

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 2

sender hardware address:

11:22:33:44:55:66

sender ip address: 10.0.3.1

target hardware address:

AA:AA:AA:11:11:11

target ip address: 10.0.3.2

# Arp Reply example

Router Interface Request  
Received

eth3

HWaddr 11:22:33:44:55:66

inet addr 10.0.3.1

## Incoming Message

### **ETHERNET header:**

destination: FF:FF:FF:FF:FF:FF

source: AA:AA:AA:11:11:11

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 1

sender hardware address:

AA:AA:AA:11:11:11

sender ip address: 10.0.3.2

target hardware address:

00:00:00:00:00:00

target ip address: 10.0.3.1

## Reply Message

### **ETHERNET header:**

destination: AA:AA:AA:11:11:11

source: 11:22:33:44:55:66

type: 2054

### **ARP header :**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 2

sender hardware address:

11:22:33:44:55:66

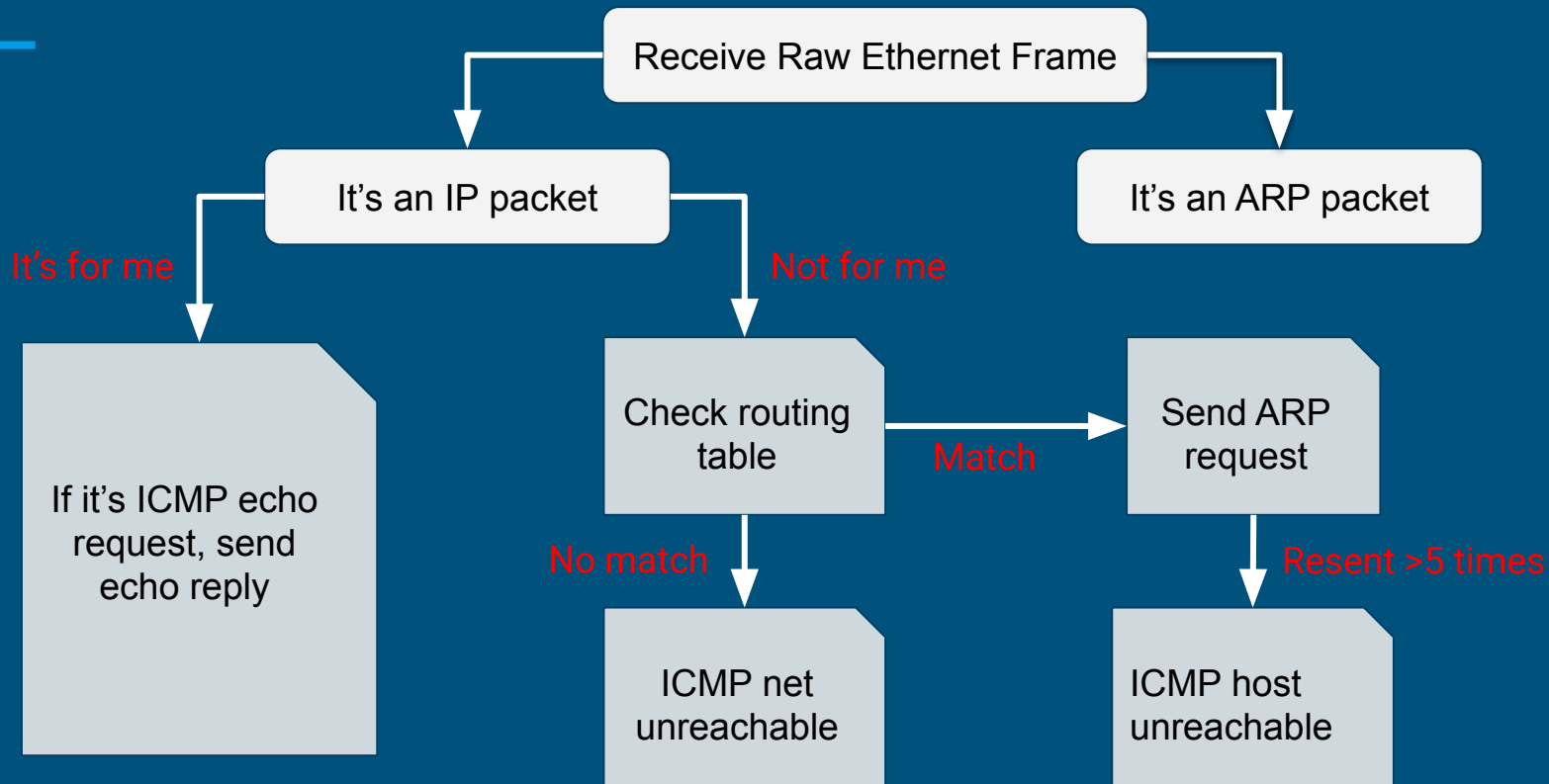
sender ip address: 10.0.3.1

target hardware address:

AA:AA:AA:11:11:11

target ip address: 10.0.3.2

# IP FLOW



# IF IT'S AN IP PACKET

---

- Check the length.
- Validate the IP header.
  - Should not be IPv6
  - Check ip\_hl
  - Check ip\_len
  - Checksum

# IF IT'S AN IP PACKET - continue

---

- Check if it is destined to you, the router.
  - Check ip\_p.
  - If it is ICMP echo request (type 8), then generate ICMP echo reply (type 0).

# IF IP PACKET IS NOT DESTINED TO THE ROUTER

---

- Check `ip_ttl`. If  $TTL \leq 1$ , send ICMP time exceeded (type 11, code 0).
- Look up next-hop address by doing exact comparisons to the ip addresses in the routing table using the packet's destination address. If it does not exist, send ICMP net unreachable (type 3, code 0).
- If it does exist, then reduce ttl and update checksum.
- send ARP request to determine MAC address
- Encapsulate IP datagram in Ethernet packet
- Forward packet to outgoing interface

# Arp Request example

Outgoing Router Interface

eth1

HWaddr AA:BB:CC:DD:EE:FF

inet addr 192.168.5.1

## Incoming IP Packet

### **ETHERNET header:**

destination: 11:22:33:44:55:66

source: AA:AA:AA:11:11:11

type: 2048

### **IP header:**

version: 4

header length: 5

type of service: 0

length: 84

id: 13429

fragment flag: DF fragment

offset: 0

TTL: 63

protocol: 1

checksum: 9785

source: 10.0.3.32

destination: 192.168.5.40

ICMP header: type: 8

code: 0

checksum: 6971

## Outgoing ARP Request

### **ETHERNET header:**

destination: FFF:FF:F:FF:FF:FF

source: AA:BB:CC:DD:EE:FF

type: 2054

### **ARP header:**

hardware type: 1

protocol type: 2048

hardware address length: 6

protocol address length: 4

opcode: 1

sender hardware address:

AA:BB:CC:DD:EE:FF

sender ip address: 192.168.5.1

target hardware address:

00:00:00:00:00:00

target ip address: 192.168.5.40

# TIPS & TRICKS

---

- GDB and Wireshark – your best buddies for next few weeks.
- Use the Print functions available in `sr_utils.c` for printing out network header information from your packets.
- Don't get mixed up with endianness: Linux - little endian, network - big endian. Take a look at the print functions to get a clearer picture
- Make sure to push your code to your Git repo. VM image could get corrupted.
- Set milestones so that you can keep track of how much work is pending

# Main Functions and Structures

---

- In `sr_router.h`
  - `struct sr_instance` is the context of the router
- In `sr_router.c`
  - `sr_handlepacket` is called for every packet that goes through the router—you have to fill it out
- `sr_protocol.h` contains convenience structs for accessing fields in packets
- `sr_if.h` contains methods for getting information about the router's interfaces
- ARP Cache in `sr_arpcache.h`

# FAQ

---

- How to verify the header length?
  - Compare the len passed in from sr\_handlepacket to the size of
  - IP packet: `sr_ethernet_hdr_t + sr_ip_hdr_t`
  - ARP Packet: `sr_ethernet_hdr_t + sr_arp_hdr_t`

# FAQ

---

- How to test the code?

- Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface] [-m mark] [-M pmtudisc\_option] [-l preload] [-p pattern] [-Q tos] [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp\_option] [-w deadline] [-W timeout] [hop1 ...] destination

```
Usage:
  traceroute [ -46dFITnreAUDV ] [ -f first_ttl ] [ -g gate,... ] [ -i device ] [ -m max_ttl ] [ -N squeries ] [ -p port ] [ -t tos ] [ -l flow_label ] [ -w waittime ] [ -q nqueries ] [ -s src_addr ] [ -z sendwait ] [ --fwmark=num ] host [ packetlen ]
```

[RFC 1393](#)

# FAQ

---

What to do when “Routing table not consistent with hardware” appear?

Restart mininet, and run ./sr or ./sr\_solution

# FAQ

---

What are the Endianness for Networks Protocols vs Linux?

Linux is using little endian

Network uses big endian

Convert the endianness using ntohs/ntohl

# FAQ

---

How to check the packet is destined to the router?

Check if the destination IP address of the packet is equal to the IP addresses of the router interface

# FAQ

---

What should be in the data field in the ICMP reply?

- ICMP type 3 and 11

- 28 bytes starting from the IP header of the original packet  
which the router received

- ICMP type 0

- Reuse the data section from the request

# Questions?

---