Project 2 - Part 1
Simple Router
- Aravind Kumar
• The skeleton code is in your github repositories and dependencies for the project have been setup in a VM image for your convenience.

• You write the router in C.

• VM has the network topology that allows your router implementation to talk with the emulated hosts.

• The network topology is configured through two files: IP_CONFIG, rtable
- Emulates network topology
- Provides isolation between emulated nodes for the SR to forward packets between them.
- More information at http://mininet.org/walkthrough/
Getting Started

- Boot your Vmdisk image on Vmware and login [Username/Password: mininet]
- Start Mininet in one terminal, and POX controller in another terminal.
  
  Note: Execute byobu-enable F2 to open new terminal and F3 to switch between terminals.

- In the third terminal, fetch the router skeleton code, execute make clean followed by make in the router directory. Run ./sr

- ~/cse123_p2/sr_solution is a reference solution to see expected behavior

- Execute ping/traceroute commands from first terminal where mininet is running.
Goal: 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 and replies
- **IP Packets**: ICMP, TCP/UDP

Where do you start coding?

- sr_handlepacket() method in sr_router.c
How to check if the packet is ARP or IP Packet?

- Check `ether_type` of ‘packet’ received in `sr_handlepacket()` using `ethertype` function defined in `sr_utils.c`.

```c
struct sr_ethernet_hdr {
    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;
```
Receive Raw Ethernet Frame

It's an IP packet
- Cache it, go through my request queue and send outstanding packets

It's an ARP packet
- Reply to me
- Request to me
- Construct an ARP reply and send it back

Courtesy: Stanford project slides
IP Flow Chart

1. Receive Raw Ethernet Frame

   - It's an IP packet
     - It's for me
       - If it's ICMP echo req, send echo reply
         - Or if it's TCP/UDP, send ICMP port unreachable
     - Not for me
       - Check routing table, perform LPM
         - Match
         - Check ARP cache
           - Hit
             - Send frame to next hope
           - Miss
             - ICMP net unreachable
       - No match
         - Send ARP request
           - Resent >5 times
           - ICMP host unreachable
   - It's an ARP packet
     - Send frame to next hope
- IP (Echo Reply packet) -> Frame header + IP header (20 bytes) + ICMP segment + Payload

- ARP -> Frame header (with 3 fields) + ARP request/reply

Sample captured packets in Wireshark

ARP request/reply

- Type: ARP (0x0806)

Address Resolution Protocol (reply)

- Hardware type: Ethernet (1)
- Protocol type: IP (0x0800)
- Hardware size: 6
- Protocol size: 4
- Opcode: reply (2)
- Source IP address: 10.0.1.1 (10.0.1.1)
- Target IP address: 10.0.1.100 (10.0.1.100)
ICMP Packets

- **Echo reply (type 0):** Sent in response to an echo request (ping) to one of the router's interfaces.

- **Destination net unreachable (type 3, code 0):** Sent when there is no matching entry in routing table when forwarding an IP packet.

- **Time exceeded (type 11, code 0):** Sent when the ttl value of the received packet is 1 (client ping -c 1 -t 1 192.168.2.2)

- **Destination host unreachable (type 3, code 1):** Sent when the router receives no arp response for the request (5 attempts) which was sent by the router.

- **Port unreachable (type 3, code 3):** Sent if an IP packet containing a UDP or TCP payload is sent to one of the router's interfaces. This is needed for traceroute to work.
If packet is ARP Type, then

- Verify the length of packet, and verify that it is destined to you, i.e. the router.
  
  E.g., this can be done by first using `sr_get_interface()` [defined in `sr_if.c`] to get the interface through which the frame arrived. Compare the ip address of this interface to the target ip address in the packet. If they are the same, then it means it was indeed destined to you, the router.

- Check 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 ARP reply.

- If it is an **ARP Reply**, insert entry in ARP cache and send all the outstanding packets in the ARP queue.

- You get an ARP reply only if you had sent an ARP request earlier to get the mac address of the packet destination.
If packet is IP type, then

- Check the packet length.
- Validate the IP header.
  - Version should be ipv4
  - Validate Checksum (cksum function in sr_utils.c)
  - etc.
- Check to see if it is destined to you i.e. the router or to some other host.
If the IP packet is destined to you, then

- Check ip_p to see if it is icmp.

  Router should not handle non-ICMP packets (tcp or udp). If it is tcp or udp, generate ICMP port unreachable (type 3, code 3).

- If it is icmp, check to see it is type 8. If it is ICMP echo request (type 8), then generate ICMP echo reply (type 0).
If the IP packet is not destined to you, then

- Check ip_ttl. If TTL <= 1, send ICMP time exceeded (type 11, code 0).
- Look up next-hop address by doing a LPM on the routing table using the packet’s destination address. If it does not exist, send ICMP host unreachable (type 3, code 0).
- If it does exist, then reduce ttl and update checksum.
- From next-hop address, determine outgoing interface and next-hop MAC address
- If necessary, send ARP request to determine MAC address
- Encapsulate IP datagram in Ethernet packet
- Forward packet to outgoing interface
Tips and Tricks

- Look at the working binary shared in your VM (./sr_solution) using wireshark, to understand what packets are being sent and how they look like.
- Don’t forget - gdb and wireshark are your friends.
- 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.
- Start by setting milestones. For example:
  - Milestone #1: implement functionality to support pinging the router’s interfaces from client
  - Milestone #2: support situations where LPM leads to a “no match”
  - Milestone #3: supports LPM and forwarding.
START EARLY!