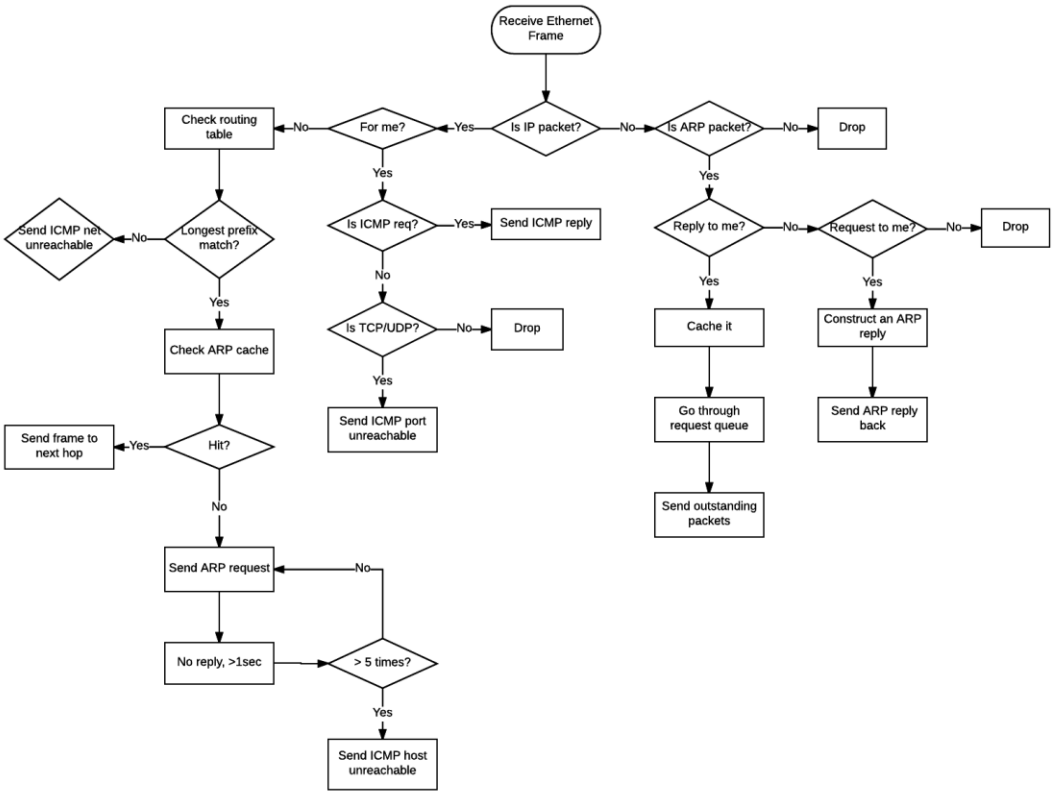
Report of Lab 1

Yifan Wu

1. High-level design of the router structure



2. Detailed implementation

Mainly added functions:
sr_arpcache.c:

/* Tries to find ip address in arp cache. If found, send ethernet frame, else, add packet to arp queue */
void sr_attempt_send(struct sr_instance *, uint32_t , uint8_t *, unsigned int , char *);

/* Sends an ARP looking for target IP */
void sr_send_arp(struct sr_instance *, enum sr_arp_opcode , char *, unsigned char *, uint32_t);

/* Check a request to see if another ARP needs to be sent or whether we should give up and send an ICMP host unreachable back to source */
void sr_handle_arpreq(struct sr_instance *, struct sr_arpreq *);

```
/* Send ICMP messages to all the packets waiting on this request */
void sr_send_icmp_to_waiting(struct sr_instance *, struct sr_arpreq *);

/* Send arp request to target ip address */
void sr_send_arp_req(struct sr_instance *, uint32_t );

/* Send an ethernet frame;
 * Take in the data and the MAC address destination and the interface through which to send it */
void sr_send_eth(struct sr_instance *, uint8_t *, unsigned int , uint8_t *, char *, enum sr_ethertype );

/* Handle receving an ARP;
 * Insert IP-MAC mapping of reply into the ARP cache,
 * then check if any packet can now be sent as a result of this mapping, if so, sends it */
void sr_recv_arp(struct sr_instance *, struct sr_arp_hdr *);
```

sr_router.c:

```
/* Handles an IP packet.
 * First, check whether packet is for me. if is, call sr_handle_my_ip_packet;
 * if not, decrement TTL and recompute checksum, then forward it. */
void sr_handle_ip_packet(struct sr_instance* , uint8_t * , unsigned int , char* );

/* Handle ARP packets. Call recv_arp to store ARP info into the cache.
    * Then, if it is an ARP request, sends reply, else, return */
void sr_handle_arp_packet(struct sr_instance* , struct sr_arp_hdr * , unsigned int , char* );

/* check if packet is for me, if is, return interface, if not return null*/
struct sr_if * sr_check_packet(struct sr_instance *, uint32_t );

/* Handle an IP packet for me/router;
   if it's ICMP echo, send reply, if it's TCP or UDP, send port unreachable to sender
*/
void sr_handle_my_ip_packet(struct sr_instance * , struct sr_ip_hdr * );

/* Sends an ICMP packet of type 3 */
void sr_send_icmp3(struct sr_instance *, enum sr_icmp_type , enum sr_icmp_code, uint32_t , uint32_t , uint8_t *, unsigned int );

/* Sends an ICMP packet not type 3 */
void sr_send_icmp(struct sr_instance *, enum sr_icmp_type , enum sr_icmp_code, uint32_t , uint32_t , uint8_t *, unsigned int );
```

/* Send an IP packet */
void sr_send_ip(struct sr_instance *, enum sr_ip_protocol , uint32_t , uint32_t , uint8_t *, unsigned int);

Other additions:
For some header files, such as sr_arpcache.h, sr_router.h, sr_protocol.h, etc. some declaration and enum types are added, to make the code structure and some attributes representations clearer.

3. Testing cases

All cases are tested in mininet.

Firstly, test given cases, including client ping/traceroute ip addresses of router's interfaces and HTTP server, and client use wget to download file from HTTP servers. All testing cases are passed.

Secondly, ping unreachable(non-exist) addresses, such as 10.0.1.2, 192.168.2.10, 137.32.2.100, fail, all sent packets are lost.

Thirdly, client ping/traceroute client itself, such as client ping client, succeed.

Fourthly, ping/traceroute client, router interface and another server from one of the servers, such as server1 ping client, server2 traceroute server1, succeed.

Fifthly, one of the servers use wget to download file from another server, such as server1 wget server2, succeed.