***Submission Requirements:***

At the end of the semester, EACH student must submit a ZIP file containing

1. A project report (in Word format) covering

a) a description of the program,

b) the challenges that you and your team had  and how did you or the team overcome them,

c) what you have learned by doing the project,

d) a discussion about algorithms and techniques used in the program,

e) contributions of each team member,

and f) any suggestions you may have (optional).

2. A short video clip demonstrating the router in action. (If the video is too big you can post it on a website, e.g. youTube, and submit the link).

3. All code that are needed to run your program.

4. A document describing your team's implementation of the router.

Obviously items 2. 3 and 4. are the same for all team members but item 1. must be prepared by each student individually (but some sections of the report, e.g. a) and d), can be shared).

Let me know if there is any questions or comments.

Teams should get together as soon as possible to discuss the requirements and decide on how to break down the task to assign to each team member.

Happy coding!

|  |
| --- |
| void sr\_handle\_arpreq(struct sr\_instance \*sr, struct sr\_arpreq \*req, |
|  |

struct sr\_if \*out\_iface)

Our part of this method involves sending the ICMP host unreachable to the source address of all packets waiting on this request. It does this by using memset/memcopy to find the original source address and making an ICMP packet using that information. We then use the sr\_send\_packet to send the ICMP packet off, and free the memory allocation.

|  |
| --- |
| void sr\_handlepacket\_arp(struct sr\_instance \*sr, uint8\_t \*pkt, |
|  |

unsigned int len, struct sr\_if \*src\_iface)

Our part of this method forwards all packets on the linked list queue. It does this via a sending while loop for the packets that are still there to be sent, and using memset/memcpy to populate the Ethernet header. Next, call sr\_send\_packet to send the packets. It then frees up the packet via free, and then releases the ARP request via sr\_arpreq\_destory.

|  |
| --- |
| void sr\_handlepacket(struct sr\_instance\* sr, |
|  |

|  |
| --- |
| uint8\_t \* packet/\* lent \*/, |
|  |

|  |
| --- |
| unsigned int len, |
|  |

char\* interface/\* lent \*/)

First we determine if ARP or IP or Misc via checking the ether\_type of the packet. If it matches 0806 it is sent to the provided sr\_handlepacket\_arp to be handled as it is an ARP packet.

If it is an IP (0800) we first verify length, then we verify the checksum with memcopy and calling up the checksum algorithm to compare the value provided with a new copy of the provided packet that is re-called with a checksum of 0. If the length is not valid, or the checksums do not match, we drop packet. Then we check to see if the address matches one of the router’s IP addresses, via looping through the IP address list and comparing to see if they match. Then we see if the IP packet contains an ICMP request, and if so if it is an ICMP echo request. If true we then call send\_icmp to send an ICMP Echo Reply. Else we send\_icmp port unreachable.

If the IP isn’t for us, we decrement the TTL and is now 0, we call send\_icmp for a time exceeded error.

Next we check the router table for IP and run our shortest prefix matching algorithm, and if an address is not valid we call send\_icmp for Destination Net Unreachable. If an address is valid we call sr\_arpcache\_lookup for an address. If we find an address we call sr\_send\_packet to send the packet. Else we call sr\_waitforarp to add the address to the list or if that fails to handle the fail case.

If the packet is not ARP or IP, we drop the packet.

|  |
| --- |
| void send\_icmp(struct sr\_instance\* sr, |
|  |

|  |
| --- |
| uint8\_t \* packet, // lent |
|  |

|  |
| --- |
| unsigned int len, |
|  |

|  |
| --- |
| char\* interface, // lent |
|  |

int type, int code)

This is our send\_icmp method that handles our ICMP packets to be sent out. It allocates a portion of memory to hold the data, then calls sr\_if to find the addr for the sending interface of the router for the sender IP address. It then populates the Ethernet header via memset and memcpy. We then create the packet with the header values being hardcoded to the ICMP standards for those that don’t change, and the error code #’s are passed via the call. It then uses sr\_send\_packet to send the ICMP packet, and free to free it up.