Lab1-Report

Qiushi Xu

qiushixu@usc.edu

In this lab, I extended the existing forwarding logic of a router by implementing two primary functionalities: **ARP Handling** and **IP Forwarding**.

1. **ARP Handling**

The core task is to ensure proper handling of ARP requests within the router's cache and adequately respond or manage these requests based on their state. The architecture primarily revolves around two primary functions:

A. **ARP Request Sweeper (sr_arpcache_sweepreqs)**

This function iterates through each ARP request in the cache at regular intervals. It assesses the state of each request and decides the course of action based on the duration since the last request was sent and the number of times the request has been sent.

Pseudo Code:

```
void sr_arpcache_sweepreqs(struct sr_instance *sr):
    for each request in sr->cache.requests:
        handle_arpreq(request)
```

Implement Code:

The function iteratively checks each ARP request in the router’s cache. For each of these requests, it calls the ARP request handler function `handle_arpreq`.

B. **ARP Request Handler**

Pseudo Code:

```
function handle_arpreq(req):
    if the difference between now and the last time the request was sent >= 1.0:
        if request has been sent >= 5 times:
            send icmp host unreachable to source addr of all packets waiting on this request
            destroy the arp request
        else:
            resend the arp request
            update the sent timestamp of the request
            increment the times_sent counter of the request
```

Implement Code:

The logic is consistent with the pseudo-code. The function first checks if the request has been sent more than five times. If so, it sends an ICMP host unreachable message to the source address of all packets waiting on the ARP request. If the request hasn't been sent five times, it

resends the ARP request.

Additionally, I incorporated the initialization and set-up of ICMP, IP, and Ethernet headers for sending the ICMP messages and the ARP requests, as well as the necessary checksum calculations for packet integrity.

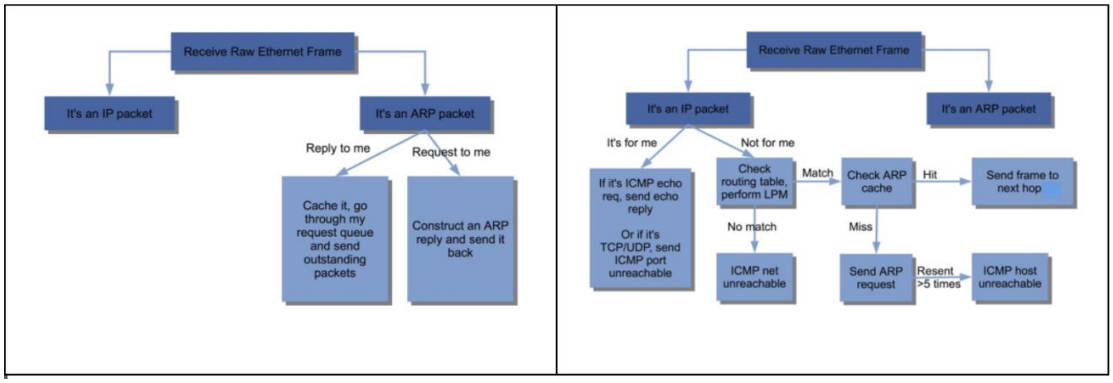
2. **IP Forwarding**

In order to benefit the construction of the forwarding process, I wrote the helper function of Longest Prefix Match Algorithm:

```
function longest_prefix_match(ip):
    best_match = NULL
    longest_mask = 0
    for each routing_entry in routing_table:
        if ip AND cur_entry.mask == cur_entry.dst_ip AND cur_entry.mask:
            if best_match is NULL or cur_entry.mask > longest_mask:
                longest_mask = cur_entry.mask
                best_match = cur_entry
```

The implementation of LPM Algorithm basically Iterate over each entry in the routing table, comparing the target IP ANDed with the current entry's mask to the entry's destination IP ANDed with the mask. If there are matches, pick the one which has the longest mask.

And finally, I implemented the function sr_handlepacket(sr, packet, len, interface), which is called each time the router receives a packet on the interface and contains the complete logic of how the router should deal with an incoming packet. The high level design can be demonstrated in the flowchart shown below.



Here's a breakdown of what it does:

A. **Handling ARP Packets**: When an ARP packet arrives, the router checks whether it's a request targeted at one of its interfaces. If it is, the router responds with an ARP reply. If it's an ARP reply to a previously sent request, the router processes the reply to determine which waiting packets can now be forwarded.

B. **Handling IP Packets**: For incoming IP packets, first the router will verify the checksum of ip_packet. If it’s not correct, it drops the packet. And the router checks if the destination IP address belongs to one of its interfaces. If it does:

(1) For ICMP echo requests, the router replies with an ICMP echo reply.

(2) For other types of packets (like TCP/UDP, possibly Traceroute), the router replies with an ICMP Port Unreachable message.

If the packet is not destined for one of its interfaces:

(1) The router checks if the Time-to-Live (TTL) has expired. If it has, the router sends an ICMP Time Exceeded message.

(2) If the TTL is still valid, the router would find the next-hop IP address using the longest prefix match, then consult the ARP cache to find the next-hop's MAC address, and finally send out the packet. If the ARP cache doesn't have an entry for the next-hop IP, the router should initiate an ARP request. If there’s matched next-hop, it will instead send back an ICMP Net Unreachable message.

Limitation

Code Redundancy: When setting up those ICMP packets and ARP packets, I realize that there are many lines of codes are repetitive and redundant, which could be encapsulated into functions that can be reused.

General Robustness: Despite the success of running all test cases, we might want to add more logic to handle other scenarios (e.g., handling ARP requests not targeted at the router but to be forwarded, other ICMP message types, etc.).