# Lab 1: Build Your Own Router

## 1. Introduction

In this lab I wrote code for a router configuration with a static routing table and a static network topology. The router receives raw Ethernet frames and processes the packets and forwards them to the correct outgoing interface. In the basic topology, there is a client which is directly connected to one interface of the router. There are two HTTP servers connected to two other interfaces of the router. Mininet is used to set up the topologies of the emulated router and process packets in them. There are two main parts of the Lab, first is to handle IP forwarding (achieved using longest prefix match) and second is to handle ARP packets. The key protocols to understand were:

a) Ethernet
b) Internet Protocol (IP)
c) Internet Control Messaging Protocol (ICMP)
d) Address Resolution Protocol (ARP).

Raw Ethernet frames were forwarded one hop by changing the destination MAC address of the forwarded packet to the MAC address of the next hop's incoming interface. For IP packets, the crucial part was to check if the received packet has a correct checksum and decrement the TTL and recomputing the checksum over the changed header before forwarding it to the next hop. ICMP control messages were a key part of the implementation. I sent out five different types of ICMP messages:

a) Echo Reply (type 0, code 0)
b) Destination Net Unreachable (type3, code 0)
c) Destination Host Unreachable (type 3, code 1)
d) Port Unreachable (type 3, code 3)
e) Time exceeded (type 11, code 0)

ARP was used to determine the next-hop MAC address corresponding to the next-hop IP address stored in the routing table.

## 2. High-Level Design

### A. File Details

a) sr_router.c – This file contains the core code for handling the incoming packets at any router interface.
b) sr_router.h – This file contains the declarations for the new functions added as part of the sr_router.c file.
c) sr_arpcache.c – This file contains the code for handling the ARP requests and managing the logic for the ARP cache functioning.
d) sr_arpcache.h – This file contains the declarations for the new functions added as part of the sr_arpcache.h file.

## B. Design Details

a) The moment the router encounters a packet at its interface, it checks whether the packet is an IP packet or a ARP packet. Accordingly, the respective functions to handle each packet type is called.
b) The function handing the IP packet will check if the packet is destined to its interface or not.
c) If the packet is destined to the router interface, it will make sure that it is an ICMP packet and send echo reply if the packet is ICMP echo request.
d) If the packet is not an ICMP packet, it either has a UDP or TCP payload and the ICMP port unreachable message is sent.
e) If the packet is not destined to one of the router interfaces, perform the longest prefix match to forward the packet via the correct router interface.
f) Also, using the next-hop IP address, lookup the MAC address of the next-hop in the ARP cache.
g) If the MAC address (ARP request) is found in the ARP cache, send the packet via the appropriate interface to the next-hop IP address and the next-hop MAC address.
h) If the MAC address (ARP request) is not found in the ARP cache, call the function to handle ARP requests and queue the ARP request if the number of ARP requests for that next-hop IP is not more than five requests.
i) If the ARP request has been sent more than five time and still not able to retrieve the next-hop MAC address, send ICMP destination unreachable response.
j) The function handing the ARP packet will check if it is a ARP request or a ARP reply.
k) If the incoming packet is an ARP request packet, it will check if the packet is destined to the router interface or not. If it is not destined to the router interface, the ARP request packet will be discarded.
l) If the ARP request is not destined to the router interface, build the ARP reply packet and send it.
m) If the incoming packet is an ARP reply packet, go through the ARP request waiting queue and send ARP reply accordingly.

## C. Tools Used

a) Mininet to perform the ping, traceroute, etc. tests
b) POX to check the incoming packets
c) GDB to identify memory corruption bugs and isolate bugs in code
d) Wireshark to go through the log of packet capture recorded in the logfile.pcap file and debug individual Ethernet, IP, ICMP and ARP header fields
e) Git and Github used as version control systems to continuously save the code progress on the local repository and the cloud repository respectively
f) Sublime Text editor to write and edit code

# 3. Implementation

## A. sr_router.c

a) sr_handlepacket()

This function will be invoked once a packet arrives at the router interface. It will first check if the packet is an IP packet or an ARP packet and call functions send_ip_packet() and send_arp_packet() respectively.

b) send_ip_packet()

First a check is performed to verify the length of the IP packet received and will return error if length of IP packet is not valid. Then we check the IP checksum to verify that the data has been reliably received and return error if there is a mismatch. Now, we will check if the interface that the packet was received on is at the router interface or not using the function fetch_interface_using_ip(). If the packet is destined to the router interface, check if the packet is an ICMP packet or not. If it is an ICMP packet, verify the length of the ICMP packet and the checksum. If the ICMP packet is not an echo request, return error at that point itself. If the ICMP packet is an echo request, send an echo reply using function send_icmp_packet_0(). Now, if the packet is not an ICMP packet, it will have either a UDP or TCP payload and send an ICMP port unreachable message using function send_icmp_packet_3_11(). If the packet is not to one of the router interfaces, check if the TTL is less than or equal to 1 and send ICMP Time exceeded message using function send_icmp_packet_3_11() if the TTL value is <= 1. If the TTL is valid, perform a longest prefix match on the routing table and if no longest prefix match exists, send the ICMP destination net unreachable message using function send_icmp_packet_3_11(). If the longest prefix match exists, decrement the TTL and recalculate the checksum and the packet is ready to be forwarded. Now we do a lookup of the next-hop MAC address as per the next-hop IP address in the ARP cache. If an entry is found in the ARP cache, forward the packet along that interface after building the correct ethernet frame. If the ARP cache entry is not found, queue the ARP request and call the function handle_arpreq().

c) send_arp_packet()

First a check is performed to verify the length of the ARP packet and will return error if the length of the ARP packet is not valid. Then we check the ARP packet received is an ARP request or an ARP reply. If the packet received is an ARP request and is not destined to one of the router interfaces, discard the packet. If it is destined to the router interface, construct an ARP reply with Ethernet and ARP header field populated and send the ARP reply via the router interface. Now, if the ARP packet received is an ARP reply, check if an ARP request exists in the ARP cache queue. If the ARP request entry exists, forward the packet via that router interface and destroy the corresponding ARP cache entry in the queue.

d) send_icmp_packet_0()
This function is to send an echo reply in response to an echo request. All destination addresses of the echo reply packet will be the source addresses of the echo request packet. Build the Ethernet, ICMP and IP headers and append the existing IP data which came with the echo request as there should be no intermittent data loss.

e) send_icmp_packet_3_11()
This function is to send destination net unreachable, destination host unreachable, destination port unreachable and time exceeded messages. All destination addresses of the echo reply packet will be the source addresses of the echo request packet. Build the Ethernet, ICMP and IP headers and append the existing IP data which came with the echo request as there should be no intermittent data loss.

f) fetch_interface_using_ip()
This function identifies the corresponding destination interface of the router as per the given 32-bit IP address.

## B. sr_router.h

This header file contains the declarations for the below functions:

a) sr_handlepacket()
b) send_ip_packet()
c) send_arp_packet()
d) send_icmp_packet_0()
e) send_icmp_packet_3_11()

## C. sr_arpcache.c

a) sr_arpcache_sweepreqs()
This function gets called every second. For each request that it send it, it will keep checking whether it should resend a request or not, or whether it should destroy the ARP request. I have made sure to keep a record of the pointer of the next request in the ARP cache queue, so that it does not get lost when the handle_arpreq() function is called.

b) handle_arpreq()
First check if more than one request was sent within the second. If the ARP request was sent out more than five times, then send out the destination host unreachable message and destroy the ARP request packet. If the ARP request was send out less than five times, construct a new ARP request packet by filling the ethernet and ARP header field and later this packet will be queues into the ARP requests queue.

c) fetch_interface_using_ethernet()
This function identifies the corresponding destination interface of the router as per the given 32-bit IP address.

### D. sr_arpcache.h

This header file contains the declarations for the below functions:

a) sr_arpcache_sweepreqs()
b) handle_arpreq()
c) fetch_interface_using_ethernet()

## 4. Results

My router is successfully able to perform the following operations:

a) Ping any of the router interfaces from the VM.
b) Traceroute to any of the router's interface IP address.
c) Ping any of the HTTP servers from the VM.
d) Traceroute to any of the HTTP server IP addresses.
e) Download a file using HTTP from one of the HTTP servers.