Project 2b: Building a Simple Router: ARP Caching and Longest Prefix Match

Assigned: 2022-02-21

Due: 2022-03-07, 10:00 PM Pacific Time

Overview

This project is based on project 2a. To make the specification easier to read, the differences from project 2a are shown in bold. The main differences are ARP caching, longest prefix match (LPM) routing, and accommodating protocols other than ICMP on top of IP.

In this assignment you will write a simple router given a static network topology and routing table. Your router will receive raw Ethernet frames. It will process these packets just like a real router and then forward them to the correct outgoing interface. You are responsible for implementing the logic for handling the incoming Ethernet frames (forward them, generate ICMP messages, drop them and more).

Your router will route real packets from an emulated host (client) to two emulated application servers (http server 1/2) sitting behind your router. The application servers are each running an HTTP server. When you have finished the forwarding path of your router, you should be able to access these servers using regular client software. In addition, you should be able to ping and traceroute to and through a functioning router. A sample routing topology is shown below:

If the router is functioning correctly, you should be able to perform the following operations:
● Ping the router's interfaces (192.168.2.1, 172.64.3.1, 10.0.1.1) from the client.
● **Traceroute from the client to any of the router's interfaces.**
● Ping the servers (192.168.2.2, 172.64.3.10) from the client.
● **Traceroute from the client to any of the servers.**
● Download a file using HTTP from one of the servers.

Additional requirements are laid out in the [Required Functionality](#) section.

**Mininet**

This assignment runs on top of Mininet which was developed at Stanford. Mininet allows you to emulate a topology on a single machine. It provides the needed isolation between the emulated nodes so that your router node can process and forward real Ethernet frames between the hosts like a real router. You don't have to know how Mininet works to complete this assignment, but more information about Mininet (if you're curious) is available [here](#).

**Getting Started**

Just continue from where you left off Project 2a!

**Configuration Files**

Same as Project 2a!

**Test Connectivity of Your Emulated Topology**

Skip! You have already completed this for Project 2a!

**Starter Code**

You are just continuing from where you left off Project 2a!

**ARP Handling**

**ARP Request**

There can be two scenarios:
- Receive an ARP request: These will be sent to the router by the hosts. The router must check if the request is for one of its IPs. If it is, an ARP reply must be constructed.
- Send an ARP request: This scenario will arise when the router receives an IP packet that must be forwarded and it needs to know the MAC address corresponding to the next-hop IP of the packet.

ARP Reply

These will be sent by the hosts in response to the ARP request sent by the router. The router must send all the packets waiting on this ARP request towards its destination and store the next-hop IP → MAC address mapping in the ARP cache.

IP Forwarding/Handling

Packets to Other Hosts

Given a raw Ethernet frame, if the frame contains an IP packet that is not destined towards one of our router interfaces:

- Sanity-check the packet (meets version, minimum length and has correct checksum).
- Decrement the TTL by 1. If the TTL is 0 after decrementing its value, send an ICMP Time Exceeded message back to the source of the packet. Otherwise, recompute the packet checksum over the modified header and proceed to the next steps.
- Find an entry in the routing table that has the longest prefix match (LPM) with the destination IP address.
  - If an entry exists, check the ARP cache for the MAC address corresponding to the next-hop IP.
    - If the ARP cache has a MAC address entry for the next-hop IP, you no longer need to send an ARP request for the next-hop IP since you already know the MAC address associated with the next-hop IP. You can simply forward the packet after updating the necessary headers.
    - If the ARP cache does not have a MAC address entry for the next-hop IP, send an ARP request for the next-hop IP (if one hasn't already been sent out within the past second), and add the packet to the queue of packets waiting on this ARP request.
  - If the router gets an ARP reply within 5 attempts (one per second), send the packets waiting on this ARP request and store the next-hop IP → MAC address mapping in the ARP cache.
  - If an ARP reply is not received within the 5 attempts, send an ICMP Destination Host Unreachable message to the source of the packet.
  - If no matching entry is in the routing table, send an ICMP Destination Net Unreachable message back to the source of the packet.
Packets to the Router

Given a raw Ethernet frame, if the frame contains an IP packet that is destined towards one of our router interfaces:

- Sanity-check the packet (meets version, minimum length and has correct checksum).
- If the packet is an ICMP echo request and its checksum is valid, send an ICMP echo reply to the sending host.
  - **NOTE:** The data field of an ICMP echo request does not have a fixed length. Its length is determined by the total length field of the IP header. The router should copy the complete data field from an echo request to the corresponding echo reply.
- If the packet contains a TCP or UDP payload, send an ICMP Destination Port Unreachable message to the source of the packet.
- Otherwise, ignore the packet.

Protocols to Understand

For help with understanding the different fields in the headers of the different protocols, please refer to the following links:

- **ARP:** http://www.networksorcery.com/enp/protocol/arp.htm
- **IP:** http://www.networksorcery.com/enp/protocol/ip.htm
- **ICMP:** http://www.networksorcery.com/enp/protocol/icmp.htm

Ethernet

You are given a raw Ethernet frame and you have to send raw Ethernet frames. You should understand the concept of source and destination MAC addresses and the idea that when we forward a packet one hop, we change the destination MAC address of the forwarded packet to the MAC address of the next hop’s incoming interface.

Internet Protocol (IP)

Before operating on an IP packet, you should verify its checksum (on the header only) and ensure that it meets the minimum length of an IP packet. Remember that you are only dealing with IPv4 packets in this assignment. If you determine that a datagram should be forwarded, you should correctly decrement the TTL field of the header and recompute the checksum over the changed header before forwarding it to the next hop. Remember that “For purposes of computing the checksum, the value of the checksum field is zero”.

Internet Control Message Protocol (ICMP)
ICMP is a simple protocol that can send control information to a host. In this assignment, your router will use ICMP to send messages back to a sending host. You will need to properly generate the following ICMP messages (including the ICMP header checksum) in response to the sending host under the following conditions:

- **Echo Reply (Type 0):** Sent in response to an echo request (ping) to one of the router's interfaces. (This is only for echo requests to any of the router's IPs. An echo request sent elsewhere should be forwarded to the next hop address as usual.)
- **Echo Request (Type 8):** Received as a ping message to either the interfaces of the router or forwarded to any other devices in the network. (This packet is not generated by our router.)
- **Destination Net Unreachable (Type 3, Code 0):** Sent if there is a non-existent route to the destination IP (No matching entry in the routing table when forwarding an IP packet).
- **Destination Host Unreachable (Type 3, Code 1):** Sent if 5 ARP requests were sent to the next-hop IP without an ARP reply.
- **Destination Port Unreachable (Type 3, Code 3):** Sent if an IP packet containing an UDP or a TCP payload is sent to one of the router's interfaces. (This is needed for traceroute to work)
- **Time Exceeded (Type 11, Code 0):** Sent if an IP packet is discarded during processing because the TTL field is 0. (This is also needed for traceroute to work) The source address of an ICMP message can be the address of any of the router's interfaces as specified in RFC792. As mentioned earlier, the only incoming ICMP messages destined to the router's IPs that you have to explicitly process are the ICMP echo requests. You can drop any other ICMP messages directed to the router.

**NOTE:** You have already been provided with a sr_icmp_t11_hdr_t struct in your starter code. You can use this struct for all the different types of ICMP messages. But if you want to create additional structs for ICMP messages for your convenience, make sure to use the packed attribute so that the compiler doesn't try to align the fields in the struct to word boundaries.

**NOTE:** You can use the -t flag to test your ICMP Time Exceeded test case. This will allow you to set the TTL on your packets.

```
mininet> client ping -t 1 172.64.3.10
```

**NOTE:** You can test the ICMP Destination Host Unreachable test case by adding a new entry to the routing table for a host which does not exist in the topology, say something like the following entry, and then ping that IP (172.64.3.100) from the client.

```
172.64.3.100   172.64.3.100    255.255.255.255    eth2
```

**NOTE:** You can test the ICMP Destination Net Unreachable test case by pinging an IP from the client which does not exist in the routing table.
NOTE: You can test the ICMP Destination Port Unreachable test case by ensuring that traceroute runs successfully.

Address Resolution Protocol (ARP)

ARP is needed to determine the next-hop MAC address that corresponds to the next-hop IP address stored in the routing table. Without the ability to generate an ARP request and process ARP replies, your router will not be able to fill out the destination MAC address field of the raw Ethernet frame you are sending over the outgoing interface. Analogously, without the ability to process ARP requests and generate ARP replies, no other router could send your router Ethernet frames. Therefore, your router must generate and process ARP requests and replies.

To lessen the number of ARP requests sent out by the router, you are required to cache the ARP replies. The cached entries should time out after every 15 seconds in order to minimize staleness. The provided ARP cache class already times the entries out for you! When sending a packet to the next-hop IP address, the router should first check the ARP cache for the MAC address corresponding to the next-hop IP address before sending an ARP request. In the case of a cache miss, an ARP request should be sent to the next-hop IP address. If the ARP request is sent 5 times (once per second) without getting a response, an ICMP Destination Host Unreachable message must be sent back to the source IP. But, in the case of a cache hit, there is no longer a need to queue the IP packet since the MAC address associated with the next-hop IP is already known from the cache. So, you do not have to send an ARP request for every IP packet.

When handling ARP requests, you should only send an ARP reply if the target IP address is one of the router's IP addresses. In the case of an ARP reply, you should only cache the entry if the target IP address is one of your router's IP addresses.

NOTE: ARP requests are sent to the broadcast MAC address (ff-ff-ff-ff-ff-ff). ARP replies are sent directly to the requester's MAC address.

Code Overview

Basic Functions

Your router receives a raw Ethernet frame and sends raw Ethernet frames when sending a reply to the sending host or forwarding the frame to the next hop. The basic functions to handle these functions are:
void sr_handlepacket(struct sr_instance* sr, uint8_t * packet, unsigned int len, char* interface)

This method, located in sr_router.c, is called by the router each time a packet is received. The packet argument points to the packet buffer which contains the full packet including the ethernet header. The name of the receiving interface is passed into the method as well.

int sr_send_packet(struct sr_instance* sr, uint8_t* buf, unsigned int len, const char* iface)

This method, located in sr_vns_comm.c, will send an arbitrary packet of length len to the network out of the interface specified by iface.

**NOTE:** You should not free the buffer given to you in sr_handlepacket (this is why the buffer is labeled as being "lent" to you in the comments). You are responsible for doing correct memory management on the buffers that sr_send_packet borrows from you (i.e., sr_send_packet will not call free on the buffers that you pass to it). However, not cleaning up the memory you allocated (unless you have allocated absurd numbers) will not result in any penalty in this assignment. Do get your code working first before you get into the intricacies of memory cleanup.

void sr_arpcache_sweepreqs(struct sr_instance *sr)

The assignment requires you to send an ARP request about once every second until an ARP reply is received or your router has already sent 5 requests without a response. This function, defined in sr_arpcache.c, is called every second, and you should add the code here that iterates through the ARP request queue and re-sends any outstanding ARP requests that haven’t been sent in the past second. If an ARP request has already been sent 5 times with no response, an ICMP Destination Host Unreachable message should be sent back to all the senders of packets that were waiting on a reply to this ARP request.

**Data Structures**

**Router (sr_router.c/h):**

The full context of the router is housed in the struct sr_instance (sr_router.h). sr_instance contains information about the topology that the router is routing for as well as the routing table and the list of interfaces.

**Interfaces (sr_if.c/h):**

The stub code uses this to create a linked list of interfaces in the router instance at member if_list. Utility methods for handling the interface list can be found at sr_if.h.
Routing Table (sr_rt.c/h):

The routing table in the stub code is read from a file (rtable) and stored in a linked list of routing entries in the current routing instance at member routing_table.

ARP Cache & ARP Request Queue (sr_arpcache.c/h):

You will need to queue the packets waiting on an ARP request. When an ARP reply arrives, you will have to add the next-hop IP → MAC address mapping to the ARP cache and send all the packets that were waiting on that ARP request. Pseudocodes for these operations have been provided in sr_arpcache.h. The cached entries should time out after every 15 seconds in order to minimize staleness. The base code already creates a thread that times out the ARP cache entries every 15 seconds! You must fill out the sr_arpcache_sweeprqs() function in the sr_arpcache.c file that gets called every second to iterate through the ARP request queue and re-send ARP requests if necessary.

Protocol Headers (sr_protocol.c/h):

Within the router framework you will be dealing directly with raw Ethernet packets. The stub code itself provides some data structures in sr_protocols.h which you may use to manipulate headers easily. There are a number of resources which describe the protocol headers in detail. Network Sorcery’s RFC Sourcebook provides a condensed reference to the packet formats that you’ll be dealing with (the links have been shared earlier in the Protocols to Understand section). For the actual specifications, you can also refer to the RFCs:


Debugging Functions:

We have provided you with some basic debugging functions in sr_utils.h. Feel free to use them to print out network header information from your packets. Below are some functions you may find useful:

- print_hdr(uint8_t *buf, uint32_t length)- Prints out all possible headers starting from the Ethernet header in the packet.
- print_addr_ip_int(uint32_t ip)- Prints out a formatted IP address from a uint32_t. Make sure you are passing the IP address in the correct byte ordering.

Required Functionality

- The router must respond correctly to ICMP echo requests to its interfaces.
The router must correctly handle TCP/UDP packets to its interfaces. In this case, the router should respond with an ICMP Destination Port Unreachable message.

The router must successfully route ICMP/TCP/UDP packets between the client and the application servers.

The router must correctly handle ARP requests and replies.

The router must correctly handle traceroutes through it (where it is not the end host) and to it (where it is the end host).

The router must maintain an ARP cache whose entries are invalidated after a timeout period of 15 seconds (the provided ARP cache class already times the entries out for you!)

While forwarding an IP packet, the router must first check the ARP cache for the MAC address corresponding to the next-hop IP before sending an ARP request.

The router must queue all packets waiting for outstanding ARP replies. If a host does not respond to an ARP request within 5 seconds, the queued packets are dropped and an ICMP Destination Host Unreachable message is sent back to the source of the queued packets (one for each queued packet).

The router must not needlessly drop packets (such as when waiting for an ARP reply).

The router must enforce guarantees on timeouts, i.e., if an ARP request is not responded to within a fixed period of time, the ICMP Destination Host Unreachable message is generated even if no more packets arrive at the router. (You can guarantee this by implementing the `sr_arpcache_sweepreqs()` function in `sr_arpcache.c` correctly.)

Submission Instructions

1. Add your name and PID into the README file.
2. Add a short description/overview of your implementation in the README file.
3. Once you have modified the files, commit the changes to your private GitHub repo using:
   a. `git add <files you have edited>`
   b. `git commit -m <your commit message>`
4. Push/sync the changes up to your private GitHub repo using:
   a. `git push origin master`
5. After you are done with your solution code, submit your repository to Gradescope using either of the following two options:
   a. Option 1 - Submit a zip:
      i. Zip your project folder.
      ii. On Gradescope, choose “Upload” as the submission method.
iii. **Upload your zip.**

b. **Option 2 - Submit through GitHub:**

   i. Push your completed code to your GitHub repository.

   ii. On Gradescope, choose “GitHub” as the submission method.

   iii. **Authorize Gradescope to access your GitHub repository by clicking “Connect to GitHub”**.

   iv. **Repository Menu: Select your Project 2 repository.**
v. **Branch Menu:** Select the *master* branch (unless you are certain that you pushed your code to some other branch that you created).

### Additional Examples

These examples show some expected traceroute and HTTP download output using the `sr_solution` binary:

```
mininet> client traceroute 192.168.2.2
traceroute to 192.168.2.2 (192.168.2.2), 30 hops max, 60 byte packets
   1  10.0.1.1 (10.0.1.1)  149.135 ms  147.469 ms  147.474 ms
   2  192.168.2.2 (192.168.2.2)  217.474 ms *  217.488 ms
mininet> client traceroute 172.64.3.10
traceroute to 172.64.3.10 (172.64.3.10), 30 hops max, 60 byte packets
   1  10.0.1.1 (10.0.1.1)  176.872 ms  176.724 ms  176.689 ms
   2  * * *
   3  * 172.64.3.10 (172.64.3.10)  176.748 ms  216.648 ms
```

**NOTE:** The extra *s appearing on line 2 are acceptable as long as you are not getting more than 2-3 lines like these “* * *” in your output on the mininet terminal.

```
mininet> client wget -O- http://192.168.2.2
Connecting to 192.168.2.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 161 [text/html]
Saving to: 'STDOUT'

0% [ ] 0 --.-K/s
<html>
<head><title> This is Server1 </title></head>
<body>
Congratulations! <br/>
Your router successfully route your packets to server1. <br/>
</body>
</html>
100%[======================================>] 161 --.-K/s in 0s

2020-05-26 13:50:38 (17.6 MB/s) - written to stdout [161/161]
```