Simple Router

This project is a localized version of Stanford's Simple Router project. You can refer to the original version if you want: http://www.scs.stanford.edu/09au-cs144/lab/router.html

A router is a network device to forward incoming packets based on the routing table and underlying protocols. In this project, you will develop a simple virtual router with a static routing table (i.e. all connected machines are fixed with their IP and MAC address). Your job is to create the 1) Packet forwarding logic and 2) Link/Internet layer protocols so that received packets must be directed to the correct interface.

Important Notice
The deadline for this project is **June 1st, Thursday, 11:55 PM**. The submission link will be automatically closed after the deadline. Note that any kinds of late submissions are not allowed.

Submission link: https://forms.gle/SbXnqnESEzVsFNu68

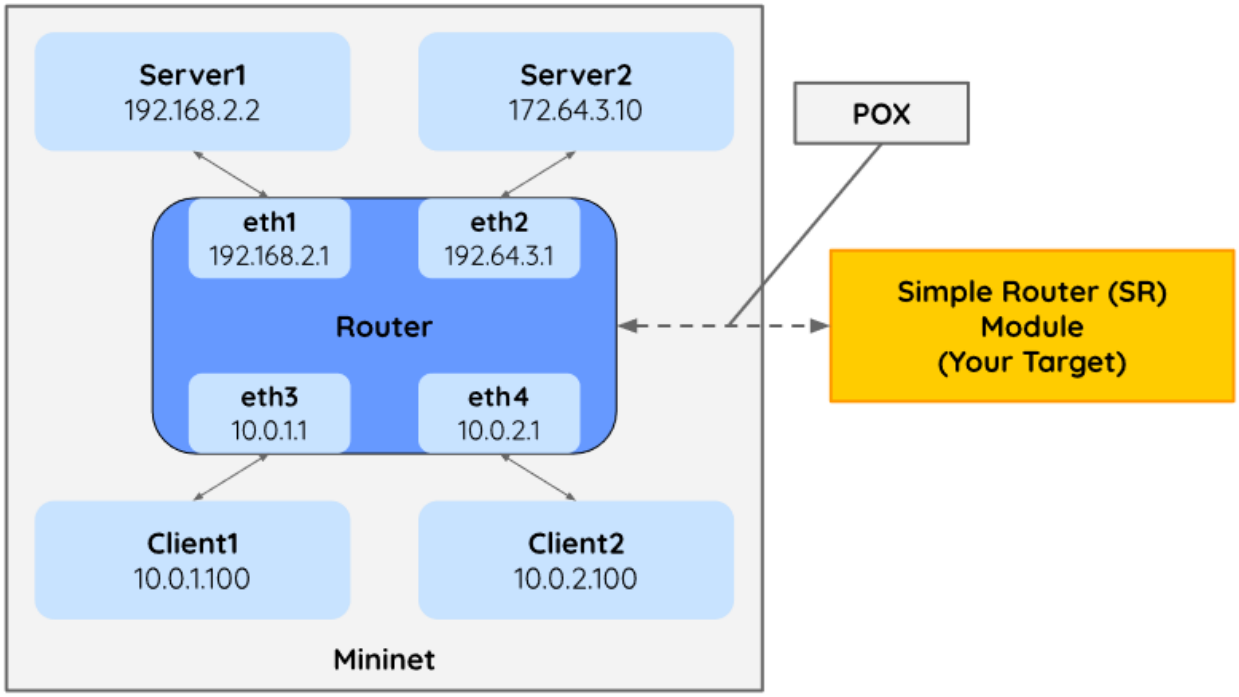

Figure 1: Virtual Network Environment for the project

Virtual Network Environment
You are not utilizing the real Internet, but using a virtual network that we provide. There are two clients (client 1, 2) and two servers (server 1, 2). Packets will be first generated from one of the clients, and servers or a router may reply to the packet. Be careful that the router's IP is

assigned for each interface. Check lecture materials to understand how routers are related with Ethernet and IP.

- All packet transmissions are emulated on **Mininet**, which was also built at Stanford. Mininet provides the needed isolation between the emulated nodes so that your router node can process and forward real Ethernet frames between the hosts just like a real router. You don't have to know how Mininet works to complete this project.
- Your Simple Router codes will be attached to the simple router running on Mininet. This attachment is done by an internal communication environment called **POX**, and they are already set up and you don't need to care about them. Just be sure that you only have to care about building the router functionalities.
- Assume that all the clients and servers are working properly on any protocol. You just need to focus on implementing the router, and be careful of some errors (ex. Checksum error) that may lead to the packet drop on clients/servers.

Background

Before starting the project, you should fully understand 1) how the router works, and 2) each underlying protocol: Ethernet, IP, ICMP, and ARP. Here we provide a brief explanation, but you must check what fields you should fill in each protocol.

1. General Forwarding Logic

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

1. Sanity-check the packet (meets minimum length and has correct checksum).
2. Check the IP. If an IP is on the blacklist, drop (ignore) the packet.
3. Decrement the TTL by 1, and recompute the packet checksum over the modified header.
4. Find out which entry in the routing table has the longest prefix match (LPM) with the destination IP address.
5. Check the ARP cache for the next-hop MAC address corresponding to the next-hop IP. If it's there, send it. Otherwise, send an ARP request for the next-hop IP, and add the packet to the queue of packets waiting on this ARP request.

Obviously, this is a very simplified version of the forwarding process, and the low-level details follow. For example, if an error occurs in any of the above steps, you will have to send an ICMP message back to the sender notifying them of an error. You may also get an ARP request or reply, which has to interact with the ARP cache correctly.

2. Protocols to Understand

A. Ethernet (Reference)

You are given a raw Ethernet frame and have to send raw Ethernet frames. You should understand the source and destination MAC addresses and the idea that we forward a packet one hop by changing the destination MAC address of the forwarded packet to the MAC address of the next hop's incoming interface.

B. Internet Protocol version 4 (IPv4) (Reference)

We will use the term IP as IPv4 for the entire document. We do not consider other IP versions. Before operating on an IP packet, you should verify its checksum and make sure it meets the minimum length of an IP packet. You should understand how to find the longest prefix match of a destination IP address in the routing table, as described above. 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.

C. Address Resolution Protocol (ARP) (Reference)
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 would 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, you are required to cache ARP replies. Cache entries should time out after 15 seconds to minimize staleness. The provided ARP cache class already times the entries out for you.
When forwarding a packet to a next-hop IP address, the router should first check the ARP cache for the corresponding MAC address before sending an ARP request. In the case of a cache miss, an ARP request should be sent to a target IP address about once every second until a reply comes in. If the ARP request is sent five times with no reply, an ICMP destination host unreachable is sent back to the source IP as stated above. The provided ARP request queue will help you manage the request queue.
In the case of an ARP request, you should only send an ARP reply if the target IP address is one of your 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 that 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.

D. Internet Control Message Protocol (ICMP) (Reference)
An IP packet may contain an ICMP message. ICMP is used to send control information back to the 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 (you don’t need to implement other types of messages in this project):

- **Echo reply (type 0, code 0)**
  Send 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.)
- **Destination unreachable (type 3, code 0)**
  Send 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)**
  Send if five ARP requests were sent to the next-hop IP without a response.

- **Port unreachable (type 3, code 3)**
  Send if an IP packet containing a UDP or TCP payload is sent to one of the router's interfaces. This is needed for the 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 source address of any of the incoming interfaces, as specified in RFC 792. For type 3 and type 11 responses, don't forget to add the original IP header + 8bytes (28bytes) to the data field.
As mentioned above, the only incoming ICMP message destined towards the router's IPs that you have to explicitly process are ICMP echo requests.

Get Started
1. Introduction
We provide a Virtual Machine (VM) image that all required programs and environments are set for the project. This means that you will run a Ubuntu operating system on top of the existing operating system. This is available by installing a VM software: Virtualbox. You can either install at your 1) allocated computing machines for previous projects or 2) your own computer. **Please make sure that the project perfectly runs on the provided VMs before submission. You can set up your own environment rather than using the provided VM image on your own responsibility; we will grade on the same environment as the VM image and we will not accept any excuses for different running environments.**

2. Install VirtualBox
Install the latest version of VirtualBox (download).

3. Download and import VM image
Download a VirtualBox VM image (download). Import the image file with VirtualBox.You may change some settings (Memory, CPU, Network), but please remind that your works will be evaluated on TA's machines. Specifications are as below:

- Ubuntu 18.04 **(Do not upgrade your operating system)**
- ID: ee323, PW: ee323
- Root PW: root

4. Check out the starter code
You will be able to check the **ee323_sr** directory at the home directory (`~/ee323_sr`). You can check out the reference binary (program) **sr_solution**. In order to execute it, you should first set POX and Mininet in each terminal. **Be careful that each program should be executed in the exact order, with waiting sufficient time for each program to launch**.

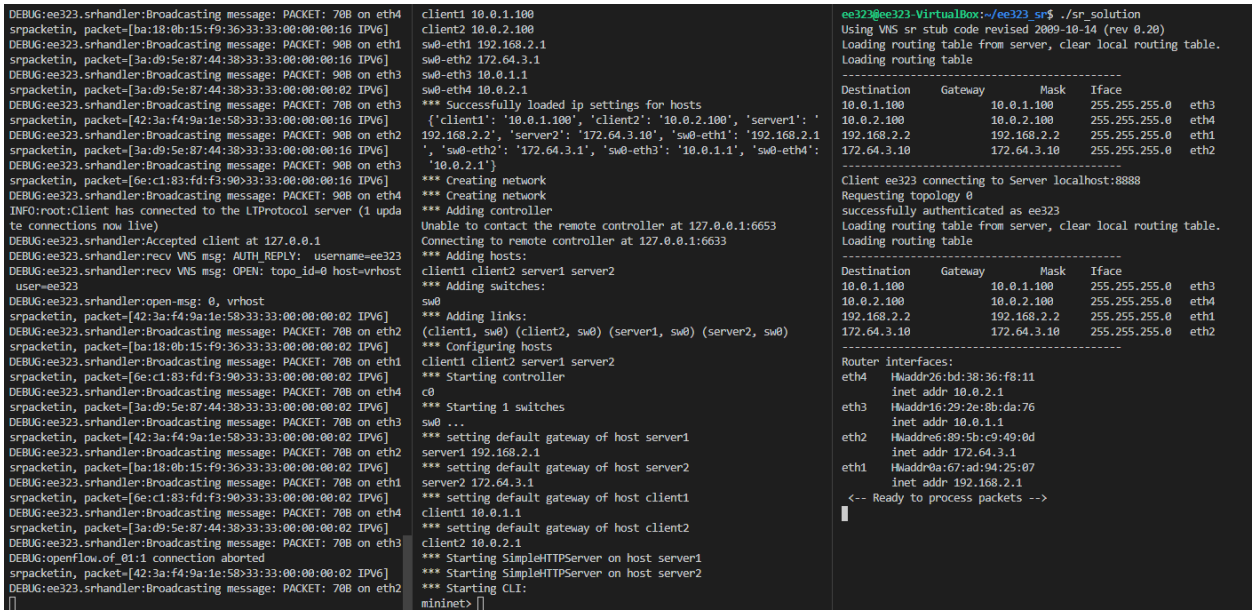

Figure 2: Executing starter code in three terminals

<At terminal 1>

```
~/ee323_sr$ ./run_pox.sh
```

<At terminal 2>

```
~/ee323_sr$ ./run_mininet.sh
```

<At terminal 3>

```
~/ee323_sr$ ./sr_solution
```

```
mininet> client1 ping -c 3 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=63 time=106 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=63 time=43.0 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=63 time=77.5 ms

--- 192.168.2.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 43.047/75.799/106.769/26.046 ms
mininet>
```

Figure 3: Executing ping request from client1 to server1.

Also, you can test your Simple Router on the Mininet (terminal 2). For example,

```
mininet> client1 ping -c 3 192.168.2.2
```

Will generate a ping (ICMP type0) message from client1 to the destination 192.168.2.2 (server1) with three times (-c option). As an example, you can generate packets in order of

```
mininet> [source] [command] [options] [destination]
```

Where commands can be one of (`ping, traceroute, wget`). You can check what options are available for each command by typing a command without `options` and `destination`.

You can close each program with `CTRL+C` (for POX and Simple Router) or `CTRL+D` (for Mininet). If you fail the connectivity (due to server down, Internet connection problems, etc.), close all existing terminals and kill processes using the command:

```
~/ee323_sr$ ./killall.sh
```

You can build and execute your own Simple Router in the **router** directory with:

```
~/ee323_sr/router$ make
~/ee323_sr/router$ ./sr
```

Also, you can generate PCAP files by setting command-line arguments to log packets (usually for debugging purposes).

```
~/ee323_sr/router$ ./sr -l filename.pcap
```

To analyze PCAP files, install Wireshark (download). Installing Wireshark in the local machine is not required, and you do not submit PCAP files, but we will generate PCAP files and use them for grading implementation integrity.

Data Structures

1. The Router (sr_router.h)

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

2. Interfaces (sr_if.c/h)

After connecting, the server will send the client the hardware information for that host. 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.c/h.

3. The Routing Table (sr_rt.c/h)

The routing table in the stub code is read on from a file (default filename "rtable", can be set with command-line option -r) and stored in a linked list of routing entries in the current routing instance (member routing_table).

4. The ARP Cache and ARP Request Queue (sr_arpcache.c/h)
You will need to add ARP requests and packets waiting on responses to those ARP requests to the ARP request queue. When an ARP response arrives, you will have to remove the ARP request from the queue and place it onto the ARP cache, forwarding any packets that were waiting on that ARP request. Pseudocode for these operations is provided in sr_arpcache.h. The base code already creates a thread that times out ARP cache entries 15 seconds after they are added for you. You must fill out the sr_arpcache_sweepreqs function in sr_arpcache.c that gets called every second to iterate through the ARP request queue and re-send ARP requests if necessary. Pseudocode for this is provided in sr_arpcache.h.

What Should I Do?
Your job is to implement three functions on two C files in the **router** directory.

- sr_router.c: `sr_handlepacket()`, `ip_black_list()`
- sr_arpcache.c: `sr_arpcache_handle_arpreq()`

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.

`1. void sr_handlepacket(struct sr_instance* sr, uint8_t * packet, unsigned int len, char* interface)`
This function 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.
An incoming IP packet may be destined for one of your router's IP addresses, or it may be destined elsewhere. If it is sent to one of your router's IP addresses, you should take the following actions, consistent with the section on protocols above:

- If the packet is an ICMP echo request and its checksum is valid, send an ICMP echo reply to the sending host.
- If the packet contains a TCP or UDP payload, send an ICMP port unreachable to the sending host.
- Otherwise, ignore the packet.

Packets destined elsewhere should be forwarded using your normal forwarding logic.

`2. int ip_black_list(struct sr_ip_hdr *iph)`
This function is called by sr_handlepacket() to filter out the packets that contain blacklisted IP, working like a firewall. You should block both inbound and outbound packets. Target IP and the corresponding mask are given in the code. If you successfully block, print the message: “`[IP blocked] : %s\n`” (does not contain “”) where `%s` contains the blocked target IP address in the Dotted-Decimal Notation (notation example: 172.0.0.1).

`3. void sr_arpcache_sweepreqs(struct sr_instance *sr)`

The project requires you to send an ARP request about once a second until a reply comes back or we have sent five requests. This function is defined in sr_arpcache.c and called every second, and you should add code 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 been sent 5 times with no response, a destination host unreachable should go back to all the senders of held packets that were waiting on a reply to this ARP request.

4. **Cautions & Tips (Not in particular order)**

- In this assignment, when you create a packet, always set IP Time-To-Live to `INIT_TTL` (255).
- 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). However, when you call `sr_send_packet()` for sending a packet, you are responsible for doing correct memory management on the buffers (that is, `sr_send_packet()` will not call free on the buffers that you pass it).
- Double-check if your program does not make meaningless outputs; print only blacklist messages (and check again if your blacklist message is correct!).
- ARP caches should be properly handled; their creation, recalls, and destruction.
- Implement in order of: `sr_handlepacket()` → `sr_arpcache_sweepreqs()` → `ip_black_list()`. Implement ARP-related stuff first to make other protocols work.
- Understand what is different between inbound and outbound packets.

How to debug?

1. Debugging functions

We provide you with some basic debugging functions in sr_utils.c/h. Feel free to use them to print out network header information from your packets. **Do not leave any functions that print messages out to STDOUT when you submit.** Below are some functions that you may find useful:

- `print_hdrs(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

2. GDB

You may have experienced a lot of `malloc()`, `calloc()`, and `free()` functions during previous projects and courses. However, dealing with them is always difficult - SEGFAULT will make you angry (if not, why are you reading this? :D). Use GDB to check where did the segmentation fault occur:

```
~/ee323_sr/router$ gdb ./sr
(gdb) run
```

And make some inputs to the Mininet that invoke SEGFAULT.

3. Generating PCAP Logs
As mentioned above, PCAP files are useful for debugging. They parse packet logs and show details of each protocol. You can also examine the checksums (FYI, you should be aware that wrong checksums may lead to no response from clients/servers).

Useful Functions
1. Handling Endianness (Reference)
You should be familiar with handling endianness: between network byte order and host byte order. Refer to materials on Project #1 if you want some recall.

- `ntohl(uint32_t hostlong)`: converts the unsigned integer from host byte order to network byte order.
- `ntohl(uint32_t netlong)`: converts the unsigned integer from network byte order to host byte order.
- `htons(uint16_t hostlong)`: converts the unsigned short from network byte order to host byte order.
- `ntohs(uint16_t netlong)`: converts the unsigned short from network byte order to host byte order.

2. Handling IP Address (Reference)
You can manually convert IP addresses between network byte order (unsigned integer) and dotted-decimal formation (ex. “192.168.0.1”). You can utilize following functions:

- `inet_ntoa(struct in_addr in)`: converts the IP address to dotted-decimal format.
- `inet_aton(const char *cp, struct in_addr *inp)`: converts the dotted-decimal format to IP address.

3. Calculating Checksum
You do not need to implement checksum calculation. Checksum can be easily calculated by `cksum()` function, but be careful to clear the checksum field before calculating it. You can find some examples utilizing checksum calculation with existing codes at `sr_handlepacket()`.

What are the grading criteria?
This project is worth **9%** of your total grade. The grading will be done as below.

- (20%) ARP: All subsequent test cases should work properly (i.e. if you fail ARP, then you may get zero point).
- (35%) ICMP
- (25%) IP
- (20%) Firewall
- (0%) Report: Submit the report that contains the captured image of execution results for each grading criteria. The report will not be explicitly graded to the numerical value, but we will use it when there are any issues with grading.
- **‘Make’ error will fail with a zero score. Test with**

```
~/ee323_sr/router$ make dist-clean; make
```

to be sure that the clean build is working.
- **Any violations (wrong file name, the report is not PDF, etc.) will result in a penalty.**
- **If your Simple Router binary has STDOUT output problems (such as adding any unnecessary messages), we will deduct 20% of your grade that you will get when the formatting problem is solved.**
- We will run your submission 5 times and will give you the highest score.
- We will set a 30 seconds limit per test case first, and re-run those with a 3 minutes limit that failed due to the timeout on the first trial. (There will be no penalty for failing a 30 seconds test but passing a 3 minutes test.) If your code fails to run in 3 minutes, you will get no credits for that test.

How to submit?
You need to submit the following items.
- sr_router.c
- sr_arpcahce.c
- report.pdf

Compress the above items into one zip file. You can use the command (Filename example: 20219876_JohnDoe_project4.zip):

```
~/ee323_sr/router$ zip -r {studentID}_{name(in English)}_project4.zip
                      sr_router.c sr_arpcache.c
```

After the compression, download it, add your report pdf file into the zip file, and submit.

Submission link: https://forms.gle/SbXnqnESEzVsFNu68

Where can I ask questions?
We encourage students to actively participate in the course. Please share your experiences and questions in the course Campuswire group. However, if you think your question is not appropriate for sharing for some reason (such as it contains private information), send it to ee323@nmsl.kaist.ac.kr. However, TAs may share your question on Campuswire.

Q&A
Q. Can I modify the skeleton code?
A. You can add some variables to three target functions, but **do not modify any skeleton code**.

Q. Can I change VirtualBox settings?
A. Yes you can, but on your own responsibility; although changing memory size or CPU core numbers would not affect much.

Q. How can I connect to the VM from outside?

A. In VirtualBox, configure the port-forward setting or use Bridge to Adaptors options to make VM connectable from outside.

Q. How can I edit the code (what editors can I use)?
A. You can use any editor that you want. It can be Vim or Emacs in VM, or you can remotely connect other editors such as Visual Studio Code.

Q. How can I download the file from VM?
A. The easiest way is using drag-and-drop from VirtualBox to your operating system. If you can't, the next easiest way would be setting a shared folder at VirtualBox to connect to your operating system. If you still have an issue, visit Campuswire :D

Q. There are some IPv6 packets shown in the POX shell. What are they?
A. You do not need to be afraid of them. They are just used for connecting and maintaining the connection between Mininet and your Simple Router so you can ignore them. Just use other methods rather than POX shell outputs for debugging.