Building your own Internet Router

Due on Friday March 27, at 5:00pm

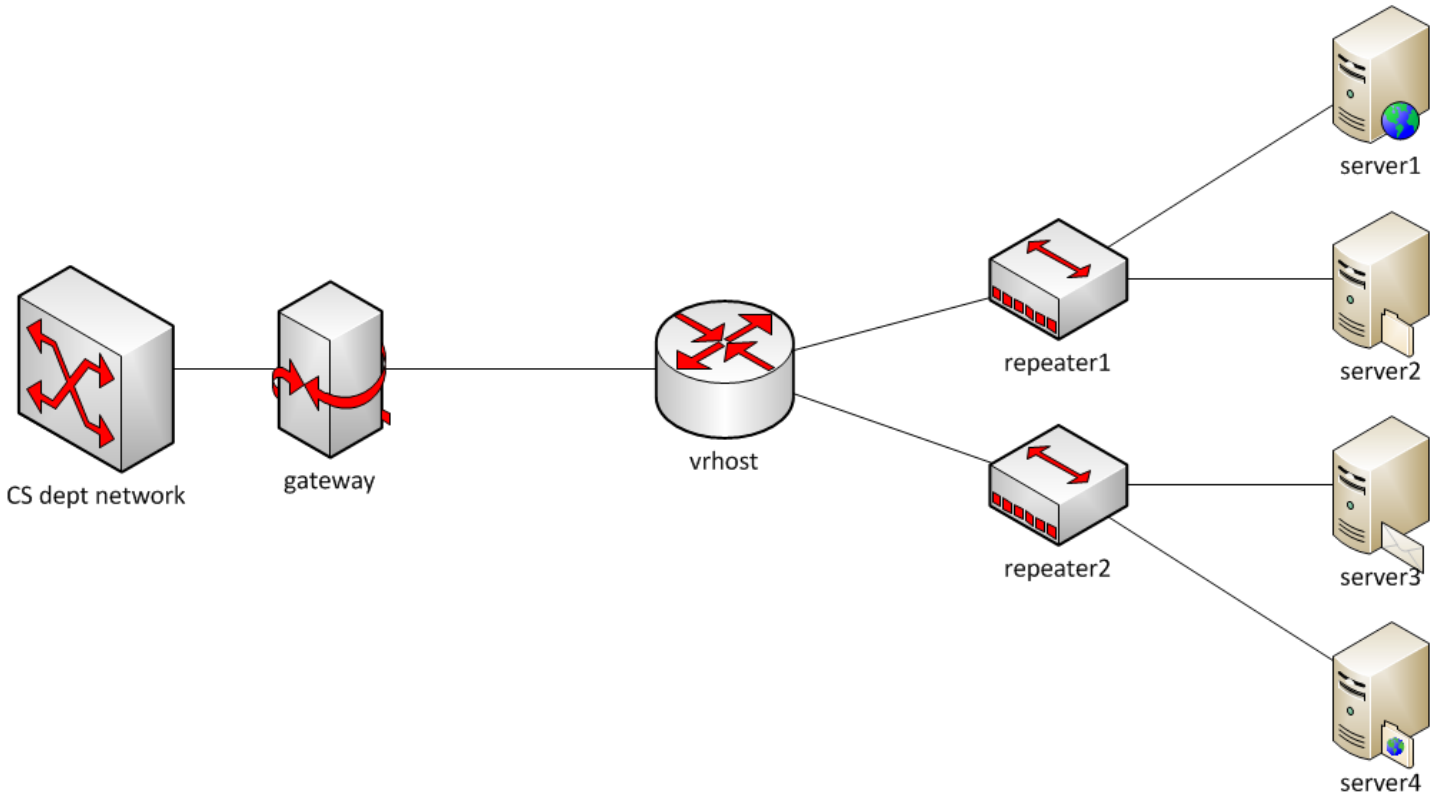
TA: Yifeng Li, yifengl@email.arizona.edu

Introduction

In this assignment you will implement a fully functional Internet router that routes real network traffic. The goal is to get hands-on experience on IP packet processing. Your router will run as a user process locally, and when finished will route real packets that are flow across the CS department network to application servers. We'll be giving you a skeleton, incomplete router (the "sr" or simple router) that you have to complete, and then demonstrate that it works by downloading some files from a web server via your router.

Overview of the Virtual Network Lab (VNL)

Virtual Network Lab (VNL) is an educational platform where students can gain hands-on experience on network protocols by programming routers and hosts. It is inspired by Stanford VNS. VNL is comprised of two components: (1) The VNL services which run in a set of virtual machines on postino.cs.arizona.edu, and (2) VNL soft-hosts such as your router. The service intercepts packets from the network, forwards these packets to the soft-host for processing, receives processed packets from the soft-host and injects them back into the network. A soft-host runs locally by the students as a regular user process and connects to the service via ssh tunnels. The soft-host can manipulate the packets in any way they want, generate responses based on the packets, or make routing decisions for those packets and send the replies back to the service to place back onto the network.



1-router 4-server topology

For example, in the above topology, the VNL service on `vrhost` may receive a TCP packet from the CS department network destined for server 1. The VNL service sends the packet to the a soft-host which will receive the packet on interface eth0, decrement the TTL, recalculate the header checksum, consult the routing table and send the packet back to the service with directions to inject it back onto the network out of interface eth1. What will the destination ethernet address be for the re-injected packet? What if the soft-host doesn't know the ethernet address for server 1?

In this assignment we provide you with the skeleton code for a basic VNL soft-host (called `sr` or Simple Router) that can connect and talk to the VNL service. Therefore, you don't need to be concerned about the interaction between VNL service and soft-host, or how packets flow in the physical topology. You can just focus

on the virtual topology and your own router. Your job is to make the router fully functional by implementing packet processing and forwarding within the skeleton code. More specifically, you'll need to implement ARP and basic IP forwarding.

Test Driving the `sr` Stub Code

Before starting the development, you should first get familiar with the `sr` stub code and some of the functionality it provides. Download the Stub Code Tarball from D2L and save it locally. You also need a VNL user package as a separate download (the download link is emailed to you). As described before, it implements all of the mechanics required for connecting and communicating with the service. To run the code, untar the code package `tar xvf stub_sr_vnl.tar` and the VNL user package `tar xvf vnltopo*.tar`, move the user packet files into the stub_sr directory, and compile it by `make`. Once compiled, you can connect to the VNL services as follows:

```
./sr -t <topo-id>
```

For example, connecting to the service on topology 87 would look like:

```
./sr -t 87
```

Other options that are useful include `-r routing_table_file`. You can use `./sr -h` to print the list of acceptable command line options.

NOTE: the VNL service is only accessible from within the CS department network.

After you connect successfully, the service will send your router/soft-host some initialization information, including all the interfaces and their IP addresses. The stub code uses this to build the interface list in the router (the head of the list is member `if_list` of `struct sr_instance`). The routing table is read from the file `rtable`. The format of the file is as follows:

```
destination gateway mask interface
```

A valid `rtable` file may look as follows:

```
0.0.0.0 172.24.74.17 0.0.0.0  eth0
172.24.74.64 0.0.0.0 255.255.255.248  eth1
172.24.74.80 0.0.0.0 255.255.255.248  eth2
```

Note: 0.0.0.0 as the destination means that this is the default route; 0.0.0.0 as the gateway means that it is the same as the destination of the incoming packet.

The VNL service, on connection should return the IP addresses associated with each one of the interfaces. The output for each interface should look like:

```
Router interfaces:
eth0    HWaddrc6:31:9f:bb:4b:6e
    inet addr 172.29.0.9
eth1    HWaddrcb:6c:4f:12:a5:2d
    inet addr 172.29.0.10
eth2    HWaddr85:e4:4d:99:e1:2c
    inet addr 172.29.0.12
```

To test if the router is actually receiving packets try pinging the IP address of eth0 of the router. The `sr` should print out that it received a packet. What type of packet do you think this is?

What should your router do on receipt of an ARP request packet?

Control Panel

Links to your control panel will be emailed to you. You will be able to see your virtual topology on the control panel. You will also find the link to download your user specific package at the bottom of the control panel.

Developing Your Very Own Router Using the SR Stub Code

Data Structures You Should Know About

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 topology the router is routing for as well as the routing table and the list of interfaces.

Interfaces (`sr_if.c, sr_if.h`): After connecting, the service will send the soft-host the hardware information for that host. The stub code uses this to create a linked list of interfaces in the router instance, `if_list`. Utility methods for handling the interface list can be found at `sr_if.h, sr_if.c`.

The Routing Table (`sr_rt.c, sr_rt.h`): The routing table in the stub code is read 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`).

The First Methods to Get Acquainted With

The two most important methods for developers to get familiar with are:

in `sr_router.c`

```
void sr_handlepacket(struct sr_instance* sr,
        uint8_t * packet/* lent */,
        unsigned int len,
        char* interface/* lent */)
```

This method is called by the router each time a packet is received. The "packet" pointer 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.

in `sr_vns_comm.c`

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

This method will send a packet of certain length ("len"), to the network out of the interface "iface".

Dealing with Protocol Headers

Within the `sr` framework you will be dealing directly with raw Ethernet packets, which includes Ethernet header and IP header. There are a number of online resources which describe the protocol headers in detail. For example, find IP, ARP, and Ethernet on www.networksorcery.comThe stub code itself provides some data structures in sr_protocols.h which you may use to manipulate headers, but it's not required. You can choose to write your own or use standard system header files found in /usr/include/net and /usr/include/netinet.

Inspecting Packets with tcpdump

As you work with the `sr` router, you will want to take a look at the packets that the router is sending and receiving. The easiest way to do this is by logging packets to a file and then displaying them using a program called tcpdump.

First, tell your router to log packets to a file in the tcpdump format:

```
./sr -t <topo-id> -l <logfile>
```

As the router runs, it will log the packets that it receives and sends (including headers). After stopping the router, you can use tcpdump to display the packets:

```
tcpdump -r <logfile> -e -vvv -x
```

The `-r` switch tells tcpdump the logfile to read, `-e` tells tcpdump to print the headers of the packets, not just the payload, `-vvv` makes the output very verbose, and `-x` displays the content in hex. You can also use `-xx` instead of `-x` to print the link-level (Ethernet) header in hex as well.

The Application Server

Once you've correctly implemented the router, you can visit the web page located at `http://<server-ip>:16280/`. The application servers serve some files via HTTP, FTP, and also hosts a simple UDP service. You will see how to access them when you get to the front web page.

Remember: The VNL service and the application servers are only accessible from within CS department network.

Troubleshooting

You can view the status of VNL services: (substitute 87 with your topology id)

```
./vnltopo87.sh gateway status
./vnltopo87.sh vrhost status
./vnltopo87.sh server1 status
./vnltopo87.sh server2 status
```

If your topology does not work correctly, you can attempt to reset it: (substitute 87 with your topology id)

```
./vnltopo87.sh gateway run
./vnltopo87.sh server1 run
./vnltopo87.sh server2 run
```

Required Functionality

In this project, you are asked to implement ARP and basic IP packet forwarding. The following functionality will be tested on your router:

1. The router correctly handles ARP requests and replies.
2. The router can successfully forward packets between the gateway and the application servers. For example, downloading a large file from the application server via http should work well.
3. The router maintains an ARP cache whose entries should be refreshed everytime a packet passes, and should be removed after a period of no activity, in the

order of 15 seconds.

4. The router does not needlessly drop packets. For example, when waiting for ARP reply, it should queue incoming data packets instead of dropping them.
5. After sending an ARP request, the router will wait for 1s. If no reply, it will re-send the request and wait for another second. This repeats until a total of 5 ARP requests have not been answered, then the router declares the nexthop is unreachable and drop the packets that are going to this nexthop. This negative ARP result should also be recorded in ARP cache.
6. Your router will be tested on a department linux machine using a topology different from your assignment. So make sure NOT to hardcode anything about your topology in your code.