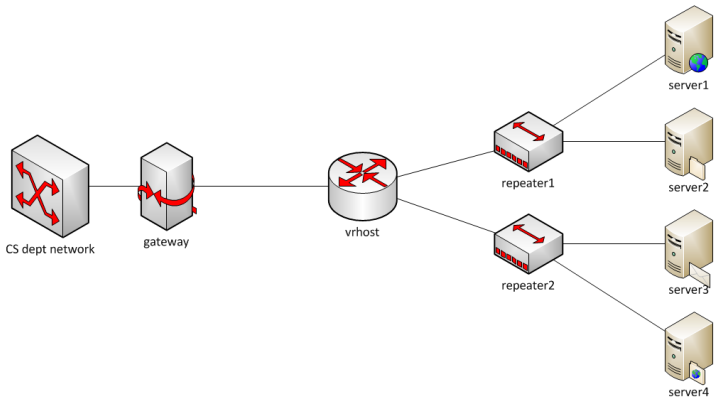
Building Your Own Router

In this assignment you will implement a functional IP router that is able to route real traffic. You will be given an incomplete router to start with. What you need to do is to implement the Address Resolution Protocol (ARP) and basic IP forwarding. A correctly implemented router should be able to forward traffic for any IP applications, including downloading files between your favorite web browser and an Apache server via your router.

Overview



This is the network topology, and *vrhost* is the router that you will work on. Each student will be assigned a topology with specific IP addresses and Ethernet addresses to the network interfaces. The router connects the CS department network to two internal subnets, each of which has two web servers in it. The goal of this project is to implement essential functionality in the router so that you can use a regular web browser in the CS network to access any of the four web servers to download files.

Note: This topology is only accessible from within the CS network.

In this assignment we provide you with the skeleton code for a Simple Router (sr), that can connect to vrhost and launch itself, but doesn't implement any packet processing or forwarding, which will be filled in by you.

The project is divided into two milestones. The first milestone implements half of the ARP protocol, and the second milestone implements the rest of ARP as well as 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 user package tarball which is attached to your assignment email.

To run the code, untar the code package *`tar xvf stub_sr_vnl.tar`* and the user package *`tar xvf vnltopo*.tar`*, move all the user package files into the ***stub_sr*** directory, and compile the code by *`make`*. Once compiled, you can start the router as follows:

```
./sr -t topID
```

Where topID is the topology ID assigned to you.

Another command-line option that may be useful is `-r routing_table_file`, which allows you to specify the routing table to load. By default, it loads the routing table from file *`rtable`*, which is specific to your topology and provided by the user package.

You can also use `-h` to print the list of acceptable command line options.

After the router is started, it will connect to `vrhost`, which will send some initialization information, including all the interfaces and their Ethernet and IP addresses. The stub code uses this information to build an interface list, and the head of the list is member *`if_list`* of *`struct sr_instance`*.

The routing table is read from the file `rtable`. Each line of the file has four fields: `Destination, gateway(i.e., nexthop), mask, and 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 address of the incoming packet.

On connection the interface information will be printed and looks like the following:

```
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 has received a packet. However, the `sr` will not do anything with the packet, so you will not see any reply for the ping.

Developing your router using the Stub Code

Important Data Structures

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`): The stub code creates a linked-list of interfaces, `if_list,` in the router instance. 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, `routing_table`, as a member of the router.

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 invoked each time a packet is received. The `*packet` points to the packet buffer which contains the full packet **including** the Ethernet header (but without Ethernet preamble and CRC). The length of the packet and the name of the receiving interface are also 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 allows you to send out an Ethernet packet of certain length ("len"), via the outgoing interface "iface".

Thus the stub code already implemented receiving and sending packets. What you need to do is to fill in sr_handlepacket( ) with packet processing that implements ARP and IP forwarding.

Downloading Files from Web Servers

Once you've correctly implemented the router, you can visit the web page located at `http://server-ip:16280/` by using GUI browser, text-based browser like lynx, or command-line tools such as curl and wget, from a CS department machine. The application servers serve some files via HTTP, FTP, and also host a simple UDP service. You will see how to access them when you get to the front web page.

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.com. The stub code itself provides some data structures in sr_protocols.h which you can 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 as well.

Inspecting Packets with tcpdump

`tcpdump` can server as an important debugging tool. As you work with the `sr` router, you will want to take a look at the packets that the router is sending and receiving on the wire. 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 topID -l logfile
```

As the router runs, it will record all the packets that it receives and sends (including headers) into the file named "`logfile`." After stopping the router, you can use tcpdump to display the contents of the logfile:

```
tcpdump -r logfile -e -vvv –x
```

The `-r` switch tells tcpdump to read logfile, `-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.

Troubleshooting of the topology

You can view the status of your topology nodes: (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), or notify the TA.

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

Milestone 1: Answering ARP Requests

When the router is running and you initiate web access to one of the servers, the very first packet that the router receives will be an ARP request, sent by the gateway node to the router asking the Ethernet address of the router.

In the first milestone, you need to

- Get familiar with the stub code
- Process the ARP request
- Send back a correct ARP reply
- Receive the next packet, which should be a TCP SYN packet going to one of the web servers.

The correct behavior can be verified by logging the packets and viewing them with tcpdump.

Milestone 2: Implementing a working router

In the 2nd milestone, you'll need to implement the rest of ARP and the basic IP forwarding. More specifically the required functionalities are listed as follows:

1. The router correctly handles ARP requests and replies. When it receives an ARP request, it can send back a correctly formatted ARP reply. When it wants to send an IP to the nexthop and doesn't know the nexthop's Ethernet address, it can send

an ARP request and parse the returned ARP reply to get the Ethernet address.

2. The router maintains an ARP cache whose entries should be refreshed every time a matching packet passes, and should be removed after 15s of no activity
3. The router can successfully forward packets between the gateway and the application servers. When an IP packet arrives, the router should do the following:
    a. Check if the destination address is itself. If yes, discard the packet.
    b. Decrement the TTL by 1. If the result is 0, discard the packet. Otherwise, update the checksum field. Refer the textbook for the IP Checksum algorithm.
    c. Look up the routing table to find out the IP address of the nexthop.
    d. Check ARP cache for the Ethernet address of the nexthop. If needed, it should send an ARP request to get the Ethernet address.
    e. Make the packet and send it to the nexthop.
4. Using the router, you can download files from the servers.
5. The IP checksum algorithm as well as sample source code is in Peterson & Davie, page 95. A great way to test if your checksum function is working is to run it on an arriving packet. If you get the same checksum that is already contained in the packet, then your function is working. Also, if the checksum is wrong, tcpdump will complain.

Testing and Grading

The project will be graded on a different topology. Don't hardcode anything about your topology in the source code.

You can work by yourself or in a group of two students. No project group should have more than two students.

Your code must be written in C and use the Stub Code.

We will test your code by two ways:

1. Access the web server from a CS machine. E.g.,
   wget http://server-IP:16280 (this retrieves the web front page from the server.)
   wget http://server-IP:16280/64MB.bin (this retrieves a 64MB file.)

2. Log packets and analyze the router's behavior. E.g., Retrieve a web page, then wait for 20s, then retrieve the page again. From the traffic log, we should see ARP request/reply at the beginning of each retrieval, but not during the retrieval because of ARP cache. We will also use this way to verify TTL decrement and checksum.

Grading is based on functionality, i.e., what works and what doesn't, **not the source code**, i.e., what has been written. For example, when a required functionality doesn't work, its credit will be deducted, regardless of whether it's caused by a trivial oversight in the code vs. a serious design flaw.

The total score of the project is 100 points with the distribution of Milestone 1 (30 pts), and Milestone 2 (70 pts).

Submission

Only one submission per group.

1. Name your working directory "topXX", where XX is the topology ID in your assignment.

2. Make sure this directory has all the source files and the Makefile. Include a README.txt file listing the names and emails of group members, and anything you want us to know about your router, especially when it only partially works.

3. Create a tarball

```
cd topXX
make clean
cd ..
tar -zcf topXX.tgz topXX
```

4. Upload topXX.tgz onto D2L dropbox.

Deadlines

Milestone 1: Tuesday April 19, at 11:59pm.
Milestone 2: Monday May 9, at 5:00pm.