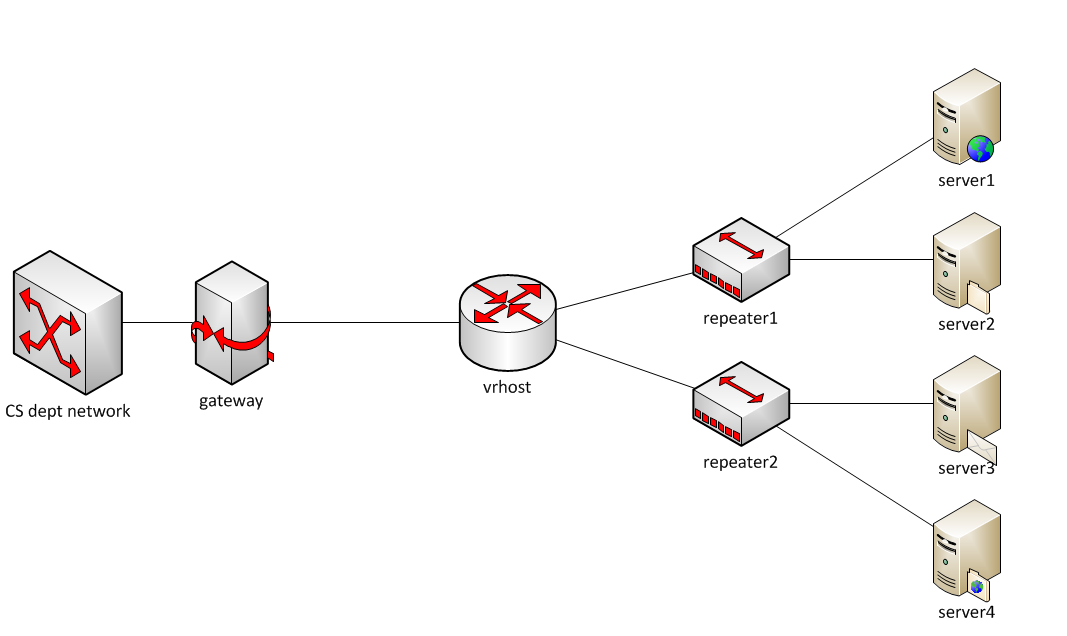
Project: Building Your Own Router

In this project 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), the basic IP forwarding, and ICMP ping. 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 his/her own topology with specific IP addresses and Ethernet addresses to the network interfaces. The *vrhost* 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 department network. You need to run your router on lectura or one of the machines in CS labs.***

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.

# Test Driving the Router Stub Code

The first thing is to get familiar with the sr stub code. 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 sr 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 save this information in a linked 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 the nexthop address 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 access one of the web servers by running the following command from a CS machine:

*wget* [*http://ServerIP:16280*](http://ServerIP:16280)

where ServerIP is the IP address of one of the servers in your topology. 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 and wget will time out.

# 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). It contains information about 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 in sr\_if.h, sr\_if.c. Note that IP addresses are stored in network order, so you shouldn’t apply htonl() when copying an address from the interface list to a packet header.

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, *struct sr\_rt \* routing\_table*, as a member of the router instance.

**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". Remember that the packet buffer needs to start with an Ethernet header.

***Thus the stub code already implemented receiving and sending packets. What you need to do is to fill in sr\_handlepacket( ) with packet processing logic that implements ARP, IP forwarding and ICMP.***

**Downloading Files from Web Servers**

Once you've correctly implemented the router, you can visit the web page located at <http://ServerIP:16280/> by using GUI browser, text-based browser like lynx, or command-line tools such as curl and wget, from a CS department machine. “ServerIP” is the IP address of one of your servers. 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](http://www.networksorcery.com/enp/Protocol.htm). The stub code itself provides data structures in sr\_protocols.h for IP, ARP, and Ethernet headers, which you can use.

With a pointer to a packet (uint8\_t \*), you can cast it to an Ethernet header pointer (struct sr\_ethernet\_hdr \*) and access the header fields. Then move the pointer pass the Ethernet header (offset the pointer by sizeof(struct sr\_ethernet\_hdr)) and cast it to a pointer to ARP header or IP header, and so on. This is how you access different protocol headers.

**Inspecting Packets with tcpdump**

tcpdump can serve 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 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 –xx

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 -xx displays the content in hex, including the link-level (Ethernet) header.

***Learn to read the hexadecimal output from tcpdump. It shows you the packet content including the Ethernet header. You can see how a correctly formatted ARP request (coming from the gateway) looks like, and check where your packet might have problem.***

**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

# Required Functionalities

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. You need to send a correctly formatted ARP reply in order to receive the next packet from the gateway.

You need to implement ARP request, ARP reply, ARP cache, IP packet processing, routing table lookup, and packet forwarding in order to get the web access working. You also need to implement ICMP echo request and echo reply for ping to work.

**(For students who took CS425 before, you also need to support traceroute by implementing ICMP time exceeded messages.)**

The specific functionalities that are required in this project 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 forwards a packet to the nexthop but doesn’t know the nexthop’s Ethernet address, it sends an ARP request and parse the returned ARP reply to get the Ethernet address.
2. The router maintains an ARP cache: once it learns the Ethernet address for a given IP address, it remembers the mapping, and reuses it next time when sending packets to the same IP.
3. The router can successfully forward packets between the gateway and the application servers.
   1. If the destination IP is the router itself, and the packet is a TCP or UDP packet, the router should drop the packet.
      1. *For students who took CS425, your router should send back an ICMP Port Unreachable message.*
   2. Decrement the TTL by 1. If the result is 0, discard the packet. Otherwise, update the checksum field.
      1. *For students who took CS425, your router should send an ICMP Time Exceeded message back to the source.*
   3. Look up the routing table to find out the IP address of the nexthop.
   4. Check ARP cache for the Ethernet address of the nexthop. If needed, it should send an ARP request to get the Ethernet address.
   5. While the router is waiting for ARP reply, it should buffer incoming packets that go to the same nexthop.
   6. After the ARP reply is received, save the information in ARP cache, and send out all the packets that are waiting for the ARP reply.
   7. Using the router, you can download large files from the servers within reasonable amount of time.
   8. When receiving an ICMP echo request destined to the router itself, the router should respond with an ICMP echo reply, so that pinging the router would work.
4. 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. Remember to zero out the checksum field when you feed the packet to your checksum calculation. If the checksum is wrong, tcpdump will complain.

# 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 in the following ways:

1. Access the web server from a CS machine. E.g.,

wget <http://ServerIP:16280> (this retrieves the web front page from the server.)

wget <http://ServerIP:16280/64MB.bin> (this retrieves a 64MB file.)

1. Ping the router and servers.
2. *(For students who took CS425): Traceroute to the router and servers.*
3. Log packets and analyze the router’s behavior. E.g., verify TTL decrement, checksum, and the ARP request/reply behavior.

**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.

# 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.

# Deadline

**Monday Oc 14, 2019, at 11:59pm.**