CIS 457 Project 2: Virtual Router

Objective: Learn about the function of a router though implementing a simplified version in software.

Deliverables: You must turn in your code on blackboard on the part two due date. Additionally, you must turn in your documentation (hardcopy) by the next lecture after the part two due date. You must demo your client and server meeting part 1 requirements on the part 1 due date in lab. You must arrange a time to demo the complete project after the part two due date. Lab time may be available for this demo, but if there is not sufficient time, you are responsible for arranging an alternate time.

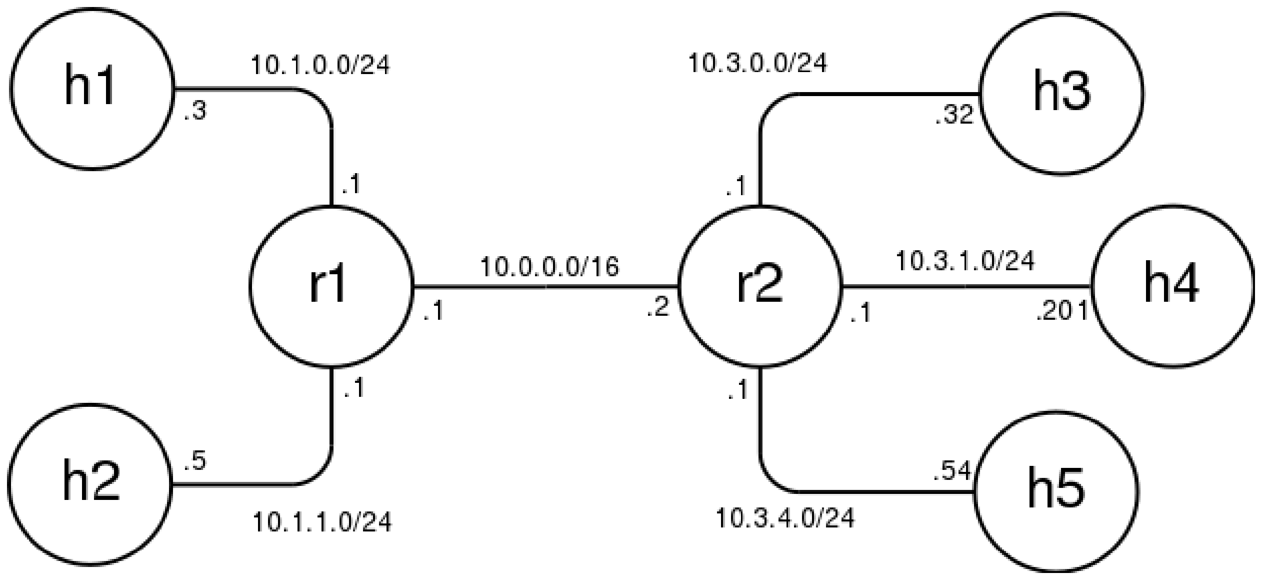
Groups: This project may be done in groups of 1 - 3 students.

Grading: This project is worth 100 points (35 for part 1, 55 for part 2, and 10 for documentation), as described below.

Lab environment

For this lab we will be using the Mininet virtual network.

The virtual network, in the configuration needed for this lab, can be started by running the script prj2-net.py (available on blackboard) (this script needs to run as root, so use `sudo` to run it. You should end up at the mininet prompt. Two virtual routers, `r1` and `r2` will be created, along with five virtual hosts `h1 - h5`. The are connected as shown in the diagram. You can start terminals on any of the hosts (or routers) with the `xterm` command from the mininet prompt. Without xterms, you can run commands on any host by typing the host name, followed by the command you would like to run.



Each of the end hosts and routers acts as a linux machine and can be interacted with in nearly the same way as any linux machine. However, these machines do share some resources with each other. Most importantly, they share a file system with each other and with the mininet VM itself. In most other ways relevant to networking, they act as separate machines. You can (and probably should during testing) run wireshark on these machines.

Sample code

route.c, available on blackboard, contains code to get you started on this project, including how to open a packet socket in C or C++. Note that this code must be run with root privileges, and should only be run in mininet. Do not run this code directly on the datacomm or eos machines. This code can easily be translated to python (packet sockets are supported, the parameters to the socket system call are similar to those in C++). However, python does not have getifaddrs to get the list of interfaces. You may use the netifaces library in python to do so.

Program Specifications

You must write the (simplified) functionality of the routers in software. Each of the routers must run the same program (although different configuration files, user input, or command line parameters may be specified). The routers are simplified in the sense that they use a small statically defined routing table; they do not participate in any actual routing protocols. Specifically, by the time you are done, the following functionality must work

- The router must respond to a ping to any of its interfaces.
- The router must be visible in traceroutes between end hosts attached to different ports

- The router must correctly route all IPv4 traffic to end hosts.
- When the router recieves a packet it can not route, it must send the appropriate ICMP message back to the sender.

Router one should use the routing table r1-table.txt, and router two should use the routing table r2-table.txt, both available on blackboard. The table has three columns: a network prefix, an IP address of a next hop device if applicable, and an interface. Packets matching the prefix on a line in this table should be forwarded on the indicated interface. The next hop IP address indicates which device the packet should be forwarded to. If there is no next hop IP address for a prefix, the packet should be forwarded to the destination IP address specified in the packet's IP header.

Your program may be written in C, C++, Python. It is highly recommended that you complete this project in C++. This program requires the use of packet sockets. These are sockets that provide access to the entire packet (all headers). Starter code is provided only in C (and should compile fine as C++). If using python, it is up to you to translate the starter code. The netifaces library will be needed to get a list of network interfaces. Java does not provide any access to packet sockets. Therefore, this project can not be completed in Java. You must open one socket per interface on the router, and be able to process packets arriving on any of these interfaces.

Part One

Due: November 1 in lab

For part one, your router must correctly deal with arp requests and ICMP echo requests. When your router gets an ARP request for one of its IP addresses, it must send out an ARP response on the same socket, back to the sender of the ARP request. The MAC address indicated in the ARP reply should be one for the same interface as the IP address indicated in the request. ARP packets consist of an Ethernet header and an ARP header. No IP header is used.

Additionally for part one, your router must correctly respond to ICMP echo request packets with any of its own IP addresses as the destination. The correct action to take when receiving an ICMP echo request is to send an ICMP echo reply with the same ID, sequence number, and data as the response. You must correctly construct the Ethernet, IP, and ICMP headers.

Once these two steps are completed, a host should be able to successfully ping the router interface it is connected to. If ARP is working but not ICMP, ping on the host should be sending ICMP echo request, and they should be seen on the router. If ARP is not working, the ICMP echo requests will not be sent. If your ARP implementation is not yet correctly working, you may test ICMP by re-enabling the operating system's ARP responses which we have disabled for this project.

Part 1 is worth 30 points, divided as follows:

Criteria	Points
Router accepts packets on packet sockets	5
Builds correct ARP response (including ethernet header)	10
Builds correct ICMP echo reply (including ethernet and ip headers)	10
Correctly uses packet socket to send responses as appropriate	5
All of the above work on all router interfaces	5

Part Two

Due: November 14th at 11:59pm

In part two we will add forwarding and error checking functionality to our routers. Upon receiving an IPv4 packet where the router itself is not the destination, the following steps should be taken:

1. Verify the IP checksum in the received packet. If incorrect, drop the packet.
2. Decrement the TTL. If the TTL becomes zero, due to this operation, send back a ICMP time exceeded (TTL exceeded) message and drop the original packet. Otherwise, you must recompute the IP checksum due to the changed TTL.
3. Find an entry in the routing table, with prefix matching the destination IP address in the packet. If no such entry exists, send back an ICMP destination unreachable (network unreachable) message.
4. Using the routing table entry found, determine the interface and next hop IP address. The next hop IP address is only used in ARP, we do not put this address into the packet being forwarded in any way.
5. Construct an ARP request, to find the ethernet address corresponding to the next hop IP address. Send this request out on the correct interface from the previous step, and receive the reply. You may use a cache to bypass this step, although you are not required to. If there is no ARP response, send back and ICMP destination unreachable (host unreachable) message.
6. Using the ethernet address found through arp as the destination, and the ethernet address of the interface you are sending on as the source, construct a new ethernet header for the packet being forwarded.
7. Finally, send out the packet on the appropriate interface (packet socket).

After completing these steps correctly, all operations listed in the specifications above should work. Therefore, you should be able to test using traceroute, ping -t, and netcat between hosts.

For the forwarding table lookup, you must match the destination address in your packet against the entries in the table. In this project we will only use very simple forwarding tables, where there is at most one match for any address. All prefixes in the table will be either 16 or 24 bits. Although these are not generally true for real forwarding tables, your code for this project may assume these to be true. You are not required to implement the lookup in an efficient manner.

After finding the next hop IP address, you must construct the ARP packet to find the MAC address corresponding to the next hop IP address, and send it. Send this request out only on the correct interface from the previous step, and receive the reply. You may use a cache to bypass this step, although you are not required to.

For ease in demoing part 2, please print the next hop IP and MAC addresses that you obtained from the routing table and from ARP.

Part 2 is worth 30 points, divided as follows:

Criteria	Points
Routing table lookup	10
ARP request properly sent	10
Interpretation of ARP response	5
Construct new ethernet header and forward	10
IP checksum calculations, TTL update	5
ICMP Error messages	15

Documentation

Documentation of your program is worth 10 points. This should be a 1-3 page document describing the design of your program. This should not be a line by line explanation of your code, but should explain the overall structure and logic of the program, as well as what major challenges you encountered and how you solved them.