COSC 465: Computer Networking
Spring 2014
Project 5: IP router 3: ICMP responder and generator
Due: 14 March 2014, 11:59:59 pm

Overview

This project is the second in a series of projects that have the ultimate goal of creating the "brains" for a Internet IPv4 router. The basic functions of an Internet router are to:

1. Respond to ARP (address resolution protocol) requests for addresses that are assigned to interfaces on the router. (Remember that the purpose of ARP is to obtain the Ethernet MAC address associated with an IP address so that an Ethernet frame can be sent to another host over the link layer.)
2. Receive and forward packets that arrive on links and are destined to other hosts. Part of the forwarding process is to perform address lookups ("longest prefix match" lookups) in the forwarding table. We will just use "static" routing in our router rather than implement a dynamic routing protocol like RIP or OSPF.
3. Make ARP requests for IP addresses that have no known Ethernet MAC address. A router will often have to send packets to other hosts, and needs Ethernet MAC addresses to do so.
4. Respond to ICMP messages like echo requests ("pings").
5. Generate ICMP error messages when necessary, such as when an IP packet's TTL (time to live) value has been decremented to zero.

The goal of the third router project is to accomplish items 4 and 5 above. When you're done with this project, you will have a fully functioning Internet router!

What you need to do

Setup

The github repository for the IP router projects (https://github.com/jsommers/cosc465_iprouter) contains a new test case file for this stage of the router and a new `setup3.sh` to include those tests. If you cloned a repo directly from mine, you can simply run "git pull" to retrieve these updated and new files. If you forked my repo and cloned from there, you will need to pull from "upstream" to retrieve the changes (or simply clone my repo, copy the new files over, and go from there).

There is also a new forwarding table for this project, named forwarding_table3.txt. You will need to use this forwarding table instead of the one you used for stage 2 of the router project.

Create a copy of your myrouter2.py file as myrouter3.py. The test script assumes your program is named myrouter3.py, so just to simplify things, start with a new file name and add it to git.

Responding to ICMP echo requests

The first key task for this project is for the router to respond to ICMP echo request ("pings") sent to an address assigned to one of its interfaces.

Prior to making a forwarding decision for an incoming IP packet (i.e., a forwarding table lookup), you should first check whether the IP destination address is the same as one of the addresses assigned to one of the router's interfaces. If the packet is also an ICMP echo request, then you should construct an ICMP echo reply and send it back to the original host that sent the ping. To do that, you should:

- Construct an ICMP header + echo reply, correctly populating the fields in the header. The POX documentation has code examples for doing this.
- Construct an IP header, which should have the destination IP address set as the source address of the incoming ICMP echo request, and the IP source address set as the router's interface address. The "payload" for the IP header should be the ICMP header that you created.
- Send (forward) the packet you constructed. You should already have code from P4 to do forwarding table lookups and ARP requests to handle this part.

Generating ICMP error messages

There are 4 situations in which you'll need to generate ICMP error messages. To this point, we have either explicitly ignored these error cases, or simply haven't considered them. The following table describes the specific cases, and the ICMP error message you'll need to generate in response to them:

1. When attempting to match the destination address of an IP packet with entries in the forwarding table, no matching entries are found (i.e., the router doesn't know where to forward the packet).	In this case, an I**CMP destination network unreachable** error should be sent back to the host referred to by the source address in the IP packet.
2. After decrementing an IP packet's TTL value as part of the forwarding process, the TTL becomes zero.	In this case, an **ICMP time exceeded error** message should be sent back to the host referred to by the source address in the IP packet.
3. ARP Failure. During the forwarding process, the router often has to make ARP requests to obtain the Ethernet address of the next hop or the destination host. If there is no host that "owns" a particular IP address, the router will never receive an ARP reply.	If after 5 retransmission of an ARP request the router does not receive an ARP reply, the router should send an **ICMP destination host unreachable** back to the host referred to by the source address in the IP packet.
4. An incoming packet is destined to an IP addresses assigned to one of the router's interfaces, but the packet is not an ICMP	The only packets destined for the router itself that it knows how to handle are ICMP echo requests. Any other packets should cause

echo request	the router to send an **ICMP destination port unreachable** error message back to the source address in the IP packet.

Refer to the POX packet library documentation for details on parsing and constructing Ethernet, ARP, IP, and ICMP packet headers. There are examples for how to construct Ethernet and ARP headers, and details on what's included in each header. There are also examples for how to generate the various ICMP error messages, and how to stitch together multiple headers to form complete packets.

When you're done, you should just submit your `myrouter3.py` file to Moodle.

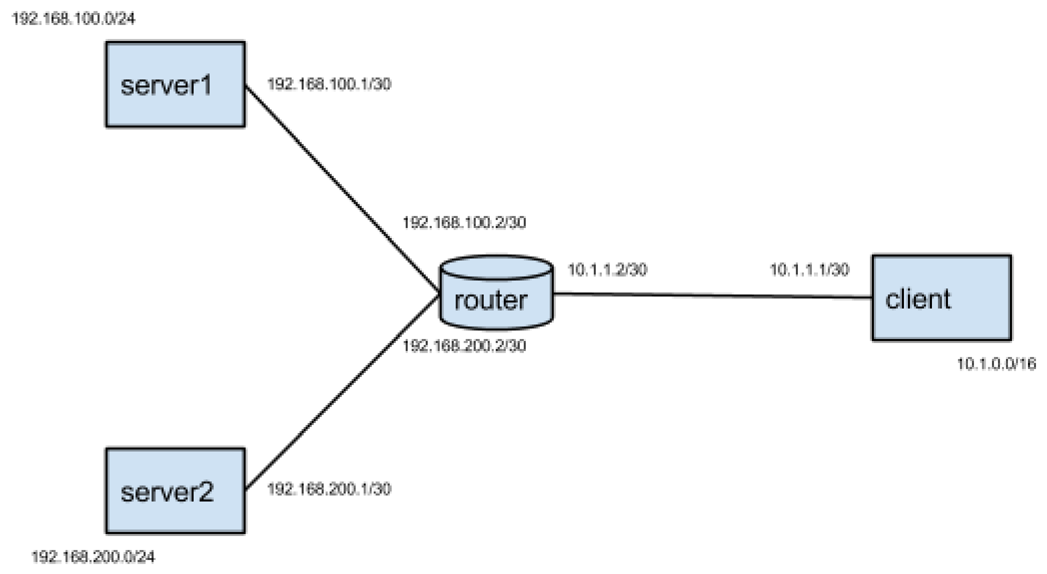
Testing your code

SRPY testing

For initial testing and debugging of your code, you can run the *runtests.sh* script that is built as a side-effect of running *setup.sh*. The `runtests3.sh` script uses the test scenario in `routertests3.srpy`, which includes several test cases.

Mininet ("live") testing

Once the SRPY tests pass, you should test your router in Mininet. There is a `start_mininet.py` script in the project git repo for building the following network topology:



(Note that the above topology is *not* the same as the one implied by the SRPY tests.)

To test each of the new router functionalities in Mininet, you can open up a terminal on the virtual machine, and cd (if necessary) to the folder where your project files are located (or transfer them into the virtual machine). Then type the following to get Mininet started:

```
$ sudo python start_mininet.py
```

Once Mininet is running, open a terminal on the router node (`xterm router`) and get the router running (`./runreal3.sh`).

Next, open a terminal on the client node (`xterm client`). Now, you should be able to:

- Use the ping tool to send an ICMP echo request to an IP address configured on one of the router's interfaces. Ping should successfully report that it is receiving replies to the echo requests.
- You can also use the ping tool and specifically set the initial TTL in the ICMP packets to be 1, so that when your router receives them, it will decrement the TTL to zero and generate an ICMP time exceeded error. The -t flag to ping allows you to explicitly set the TTL. For example:

  ```
  client# ping -c 3 -t 1 192.168.200.1
  ```
- You can send a ping from the client to an address that doesn't have a match in the router's forwarding table. There is a route set up on the client to forward traffic destined to 172.16.0.0/16 to the router, but the router doesn't have any forwarding table entry for this subnet. So the following ping should result in an ICMP destination net unreachable message sent back to the client:

  ```
  client# ping -c 172.16.1.1
  ```