Lab 3: Simple Router

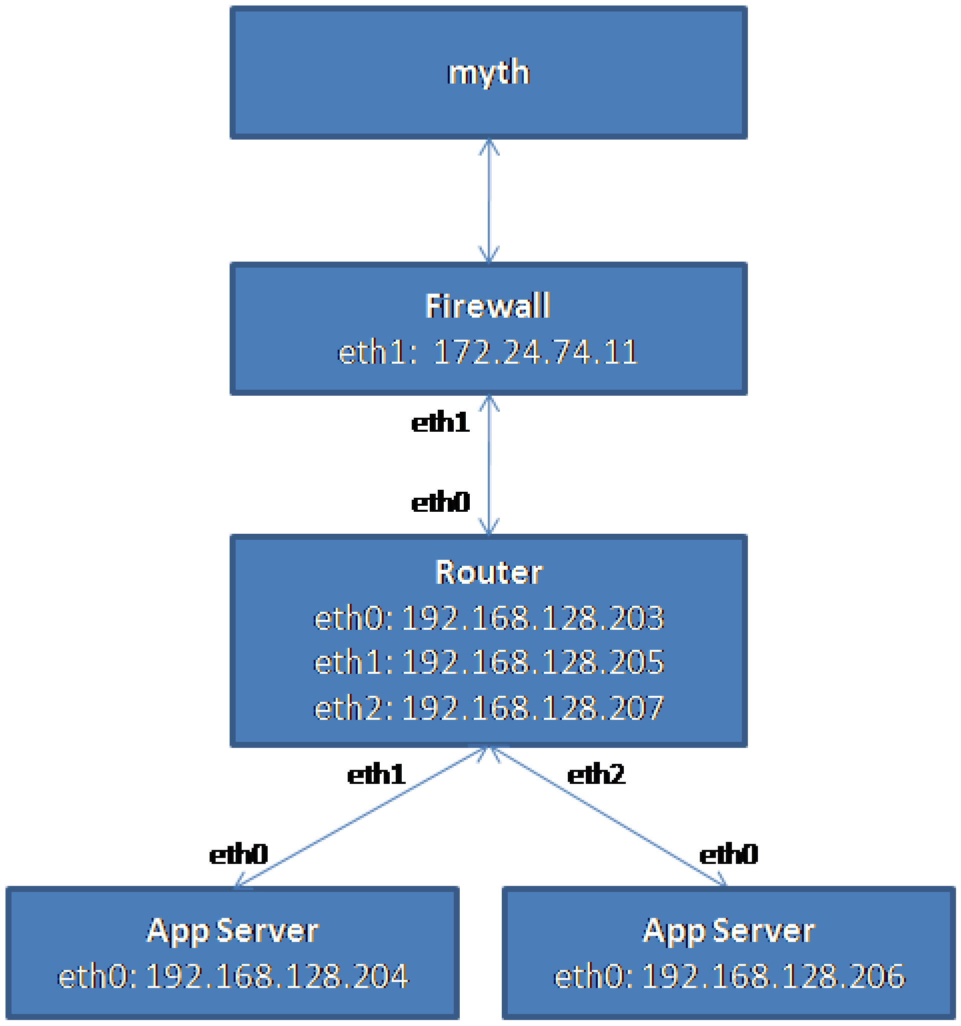
Due date: Thursday, October 28 @ the beginning of class.

Update 10.15.2010: The app servers are no longer running FTP servers. You no longer should be able to FTP to the app servers. You should still be able to run HTTP however.

Introduction

In this lab assignment you will be writing a simple router with a static routing table. You will be passed raw Ethernet frames and given a function that can send a raw Ethernet frame. It is up to you to come up with the forwarding logic between getting a frame and potentially sending it out over another interface.

Your router will route real packets from the *myth* machines to application servers sitting behind your router. The application servers are running both an HTTP server ~~and a FTP server~~. When you have finished the forwarding path of your router, you should be able to access these servers using regular client software. In addition, you should be able to *ping* and *traceroute* to and through a functioning router. For example, an example routing topology is shown below:



If the router is functioning correctly, all of the following operations should work:

- Pinging to any of the router's interfaces (192.168.128.203, 192.168.128.205, 192.168.128.207)

- Tracerouting to any of the router's interfaces
- Pinging to any of the app servers (192.168.128.204, 192.168.128.206)
- Tracerouting to any of the app servers
- Downloading a file using HTTP from one of the app servers
- ~~Downloading a file using FTP from one of the app servers~~

Additional requirements are laid out in the Requirements section.

The Virtual Network System

This assignment runs on top of the Virtual Network System which was built at Stanford. VNS allows you to build virtual network topologies consisting of nodes that operate on actual Ethernet frames. This is why your router node can process and send real Ethernet frames and send them over the network like a real router.

You don't have to know how VNS works to complete this assignment, but more information about VNS (if you're curious) is available here.

Getting Started

Creating a Topology

The first thing you will need to do is to get a testing topology like the one shown in the diagram. We have provided you with testing topologies that are accessible from /usr/class/cs144/lab3_10au/student_auths/<SUNET_ID@stanford.edu>/auth_key

Please copy the auth_key into your assignment directory and folllow directions in INSTRUCTIONS to set up your routing table file, rtable.vrhost. **Note: You must specify rtable.vrhost the first time to get the routing table file for your topology**. You may specify other routing table files if you want your router to use a customized routing table file.

Downloading the Starter Code

Download and untar the assignment package

You should be able to build the router out of the box.

Understanding a Routing Table File

You'll notice that after running the starter code that you will have simple routing table file, rtable.vrhost. The rtable file has the format:

```
prefix next_hop netmask interface
```

The base rtable file only has a default route to the firewall. For example, the routing table for router in the sample topology in the introduction follows:

```
0.0.0.0 172.24.74.11 0.0.0.0 eth0
192.168.128.204 192.168.128.204 255.255.255.255 eth1
192.168.128.206 192.168.128.206 255.255.255.255 eth2
```

Building and Running

You should now have all the pieces needed to build and run the router:

- Your *auth_key* to access your VNS topology.
- A routing table file that corresponds to the routing table for the router node in that topology, which we'll call *rtable.vrhost*
- The starter code

You can then build and run the starter code:

```
> make
> ./sr -u <SUNET_ID> -T '1-router 2-server' -s vns-2.stanford.edu -
r rtable.vrhost
```

Note: Currently, you can only run your sr binary on myth machines. If you try to run sr on pod, you won't be able to connect to the VNS servers. In addition, you will have to ping and traceroute from Stanford machines. You can do so simply by ssh-ing into myth and running ping or traceroute from one of these machines.

From another terminal or another myth, try pinging one of the router's interfaces. Currently, all the starter code does is print that it gets a packet to the command line.

For example, when running the starter code on the topology and pinging an interface on the router from another terminal, we get the following output:

```
> ./sr -u bmistree -T '1-router 2-server' -s vns-2.stanford.edu -r
rtable.vrhost
Using VNS sr stub code revised 2009-10-14 (rev 0.20)
Client bmistree connecting to Server vns-2.stanford.edu:3250
Requesting topology template 1-router 2-server
successfully authenticated as bmistree
Connected to new instantiation of topology template 1-router 2-
server
Loading routing table
---------------------------------------------
Destination Gateway Mask Iface
0.0.0.0 172.24.74.11 0.0.0.0 eth0
192.168.128.204 192.168.128.204 255.255.255.255 eth1
192.168.128.206 192.168.128.206 255.255.255.255 eth2
---------------------------------------------
Router interfaces:
eth0 HWaddr70:00:17:db:00:02
inet addr 192.168.128.203
eth1 HWaddr70:00:17:db:00:04
inet addr 192.168.128.205
eth2 HWaddr70:00:17:db:00:06
inet addr 192.168.128.207
<-- Ready to process packets -->

# After running "ping 192.168.128.203" from another
# terminal or myth

*** -> Received packet of length 42
*** -> Received packet of length 42
*** -> Received packet of length 42
.
```

```
.
.
```

Logging Packets

You'll find it quite useful for debugging errors to use a packet inspection tool like Wireshark, tcpdump, or JEthereal. You can save packets to a packet log file using the -l option:

```
> ./sr -u bmistree -T '1-router 2-server' -s vns-2.stanford.edu -r
rtable.vrhost -l logname
```

Wireshark

From there, you can use Wireshark, a graphical packet inspection tool, to open the logfile. We will try to install a binary of Wireshark for you to use (pending).

Alternatively, you can install Wireshark locally and run it from your own machine to inspect the packet log.

JEthereal

JEthereal is another option. You can find it here: http://yuba.stanford.edu/JEthereal/ JEthereal is convenient to use as an applet from the browser.

tcpdump

You can also choose to use tcpdump from the command line to inspect the packet log:

```
> tcpdump -r logname -e -vvv -x
```

The -r switch tells tcpdump where to look for the logfile. -e tells tcpdump to print the headers of the packets, not just their payload. -vvv makes the output very verbose, and -x puts the packets in a hex format that is usually easier to read than ASCII. You may want to specify the -xx option instead of -x to print the link-level (Ethernet) header in hex as well.

General Forwarding Logic

To get you started, an outline of the forwarding logic for a router follows, although it does not contain all the details. There are two main parts to this assignment: *Handling ARP and IP forwarding*

IP Forwarding

Given a raw Ethernet frame, if the frame contains an IP packet that is not destined towards one of our interfaces:

1. Sanity-check the packet (meets minimum length and has correct checksum).
2. Decrement the TTL by 1, and recompute the packet checksum over the modified header.
3. Find out which entry in the routing table has the longest prefix match with the destination IP address.
4. Check the ARP cache for the next-hop MAC address corresponding to the next-hop IP. If it's there, send it. Otherwise, send an ARP request for the next-hop IP (if one hasn't been sent within the last second), and add the packet to the queue of packets waiting on this ARP request.

Obviously, this is a very simplified version of the forwarding process, and the low-level details follow. For example, if an error occurs in any of the above steps, you will have to send an ICMP message back to the sender notifying them of an error. You may also get an ARP request or reply, which has to interact with the ARP cache correctly.

Protocols to Understand

Ethernet

You are given a raw Ethernet frame and have to send raw Ethernet frames. You should understand source and destination MAC addresses and the idea that we forward a packet one hop by changing the destination MAC address of the forwarded packet to the MAC address of the next hop's incoming interface.

Internet Protocol

Before operating on an IP packet, you should verify its checksum and make sure it meets the minimum length of an IP packet. You should understand how to find the longest prefix match of a destination IP address in the routing table, as described in Section 2. If you determine that a datagram should be forwarded, you should correctly decrement the TTL field of the header and recompute the checksum over the changed header before forwarding it to the next hop.

Internet Control Message Protocol

ICMP is a simple protocol that can send control information to a host. In this assignment, your router will use ICMP to send messages back to a sending host. You will need to properly generate the following ICMP messages (including the ICMP header checksum) in response to the sending host under the following conditions:

- Echo reply (type 0)
  Sent in response to an echo request (ping) to one of the router's interfaces. (This is only for echo requests to any of the router's IPs. An echo request sent elsewhere should be forwarded to the next hop address as usual.)
- Destination unreachable (type 3, code 0)
  Sent if there is a non-existent route to the destination IP (no matching entry in routing table when forwarding an IP packet).
- Destination host unreachable (type 3, code 1)
  Sent if five ARP requests were sent to the next-hop IP without a response.
- Port unreachable (type 3, code 3)
  Sent if an IP packet containing a UDP or TCP payload is sent to one of the router's interfaces. This is needed for traceroute to work.
- Time exceeded (type 11, code 0)
  Sent if an IP packet is discarded during processing because the TTL field is 0. This is also needed for traceroute to work.

The source address of an ICMP message can be the source address of any of the incoming interfaces, as specified in RFC 792.

As mentioned above, the only incoming ICMP message destined towards the router's IPs that you have to explicitly process are ICMP echo requests.

You may want to create additional structs for ICMP messages for convenience, but make sure to use the packed attribute so that the compiler doesn't try to align the fields in the struct to word boundaries:

Address Resolution Protocol

ARP is needed to determine the next-hop MAC address that corresponds to the next-hop IP address stored in the routing table. Without the ability to generate an ARP request and process ARP replies, your router would not be able to fill out the destination MAC address field of the raw Ethernet frame you are sending over the outgoing interface. Analogously, without the ability to process ARP requests and generate ARP replies, no other router could send your router Ethernet frames. Therefore, your router must generate and process ARP requests and replies.

To lessen the number of ARP requests sent out, you are required to cache ARP replies. Cache entries should time out after 15 seconds to minimize staleness. The provided ARP cache class already times the entries out for you.

When forwarding a packet to a next-hop IP address, the router should first check the ARP cache for the corresponding MAC address before sending an ARP request. In the case of a cache miss, an ARP request should be sent to a target IP address about once every second until a reply comes in. If the ARP request is sent five times with no reply, an ICMP destination host unreachable is sent back to the source IP as stated above. The provided ARP request queue will help you manage the request queue.

In the case of an ARP request, you should only send an ARP reply if the target IP address is one of your router's IP addresses. In the case of an ARP reply, you should only cache the entry if the target IP address is one of your router's IP addresses.

Note that ARP requests are sent to the broadcast MAC address (ff-ff-ff-ff-ff-ff). ARP replies are sent directly to the requester's MAC address.

IP Packet Destinations

An incoming IP packet may be destined for one of your router's IP addresses, or it may be destined elsewhere. If it is sent to one of your router's IP addresses, you should take the following actions, consistent with the section on protocols above:

- If the packet is an ICMP echo request and its checksum is valid, send an ICMP echo reply to the sending host.
- If the packet contains a TCP or UDP payload, send an ICMP port unreachable to the sending host.
- Otherwise, ignore the packet.

Packets destined elsewhere should be forwarded using your normal forwarding logic.

Code Overview

Basic Functions

Your router receives a raw Ethernet frame and sends raw Ethernet frames when sending a reply to the sending host or forwarding the frame to the next hop. The basic functions to handle these functions are:

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

This method, located in sr_router.c, is called by the router each time a packet is received. The "packet" argument 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.

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

This method, located in sr_vns_comm.c, will send an arbitrary packet of length, len, to the network out of the interface specified by iface.

You should not free the buffer given to you in sr_handlepacket (this is why the buffer is labeled as being "lent" to you in the comments). You are responsible for doing correct memory management on the buffers that sr_send_packet borrows from you (that is, sr_send_packet will not call free on the buffers that you pass it).

```
    void sr_arpcache_sweepreqs(struct sr_instance *sr)
```

The assignment requires you to send an ARP request about once a second until a reply comes back or we have sent five requests. This function is defined in sr_arpcache.c and called every second, and you should add code that iterates through the ARP request queue and re-sends any outstanding ARP requests that haven't been sent in the past second. If an ARP request has been sent 5 times with no response, a destination host unreachable should go back to all the sender of packets that were waiting on a reply to this ARP request.

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/h):

After connecting, the server will send the client the hardware information for that host. The stub code uses this to create a linked list of interfaces in the router instance at member if_list. Utility methods for handling the interface list can be found at sr_if.c/h.

The Routing Table (sr_rt.c/h):

The routing table in the stub code is read on 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 ARP Cache and ARP Request Queue (sr_arpcache.c/h):

You will need to add ARP requests and packets waiting on responses to those ARP requests to the ARP request queue. When an ARP response arrives, you will have to remove the ARP request from the queue and place it onto the ARP cache, forwarding any packets that were waiting on that ARP request. Pseudocode for these operations is provided in sr_arpcache.h.

The base code already creates a thread that times out ARP cache entries 15 seconds after they are added for you.

You must fill out the `sr_arpcache_sweepreqs` function in sr_arpcache.c that gets called every second to iterate through the ARP request queue and re-send ARP requests if necessary. Psuedocode for this is provided in sr_arpcache.h.

Protocol Headers (sr_protocol.h)

Within the router framework you will be dealing directly with raw Ethernet packets. The stub code itself provides some data structures in sr_protocols.h which you may use to manipulate headers easily.

There are a number of resources which describe the protocol headers in detail. Network Sorcery's RFC Sourcebook provides a condensed reference to the packet formats you'll be dealing with:

- Ethernet
- IP
- ICMP
- ARP

For the actual specifications, there are also the RFC's for ARP (RFC826), IP (RFC791), and ICMP (RFC792).

Debugging Functions

We have provided you with some basic debugging functions in sr_utils.h, sr_utils.c. Feel free to use them to print out network header information from your packets.

Below are some functions you may find useful:

- `printAllHeaders(uint8_t *buf, uint32_t length)` - Prints out all possible headers starting from the Ethernet header in the packet
- `printIPFromInt(uint32_t ip)` - Prints out a formatted IP address from a uint32_t. Make sure you are passing the IP address in the correct byte ordering.

FAQ

This assignment has been used at several universities as well as CS244A, and as a result a long FAQ page exists here:

Simple Router FAQ

Please consult the FAQ before posting a question to the newsgroup.

Length of Assignment

This assignment is considerably harder than the first 2 labs, so get started early.

In our reference solution, we added 520 lines of C, including whitespace and comments:

```
> wc -l /usr/class/cs144/src/router/*.c | tail -1
 1585 total
> wc -l ~/cs144/router/solution/*.c | tail -1
 2104 total
```

Of course, your solution may need fewer or more lines of code, but this gives you an idea a rough idea of the size of the assignment to a first approximation.

Reference Binary

To help you debug your topologies and understand the required behavior we provide a reference binary here:

```
/usr/class/cs144/bin/sr_ref
```

Here is an example of running the reference router on a topology that uses the routing table file "rtable.vrhost" in the current working directory:

```
/usr/class/cs144/bin/sr_ref -u bmistree -T '1-router 2-server' -s vns-2.stanford.edu -r rtable.vrhost
```

Substitute the correct values for your SUNET ID when running the reference router. You might want to run it to help you:

- Test your routing tables
- Learn how the sr command line options work with VNS
- Understand the required functionality

Required Functionality

- The router can successfully route packets between the firewall and the application servers.
- The router correctly handles ARP requests and replies.
- The router correctly handles traceroutes through it (where it is not the end host) and to it (where it is the end host).
- The router responds correctly to ICMP echo requests.
- The router handles TCP/UDP packets sent to one of its interfaces. In this case the router should respond with an ICMP port unreachable.
- The router maintains an ARP cache whose entries are invalidated after a timeout period (timeouts should be on the order of 15 seconds).
- The router queues all packets waiting for outstanding ARP replies. If a host does not respond to 5 ARP requests, the queued packet is dropped and an ICMP host unreachable message is sent back to the source of the queued packet.
- The router does not needlessly drop packets (for example when waiting for an ARP reply)
- The router enforces guarantees on timeouts--that is, if an ARP request is not responded to within a fixed period of time, the ICMP host unreachable message is generated even if no more packets arrive at the router. You can guarantee this by implementing the sr_arpcache_sweepreqs function in sr_arpcache.c correctly.

Submitting

Please create a README file with your SUNET ID (network login) and any description/overview of your implementation that you feel is relevant.

To submit, run

```
make submit
```

from your project directory and submit the resulting tarball on the submission page.

Collaboration policy

You must write all the code you hand in for the programming assignments, except for code that we give you as part of the assignment. You are not allowed to look at anyone else's solution (and you're not allowed to look at solutions from previous years). You may discuss the assignments with other students, but you may not look at or copy each others' code.