Lab 4: Simple Router

CS144 Lab 4 Screencast
May 2, 2008
Ben Nham

Based on slides by Clay Collier and Martin Casado
Assignment Overview

• You create a virtual network topology, using a topology creation tool
  – Covered in intro screencast
• You write a router in C
• Your router will route real IP packets over the Internet from standard clients (i.e. ping, traceroute, ftp, Firefox...)
• Due Wednesday, May 14th before class
  – Extended to Saturday, May 17th @ 5:00 PM if you attend class
• Get started early!
Getting Started

• Copy the assignment code from: 
  /usr/class/cs144/src/router

• Watch the intro screencast to:
  – Create a topology
  – Create a routing table for the topology
  – Compile and run the stub code with a routing table for your topology
Protocols You Need to Handle

- Ethernet
  - All packets given to you are raw Ethernet frames
- IP
- ARP
  - Needed to resolve IP addresses to MAC addresses
- ICMP requests/replies
  - Used by some programs to send requests (ping)
  - Needed to send control messages back to host
- See sr_protocol.h and Network Sorcery to deal with the raw bits
- Make sure you understand your pointer arithmetic!
Router Basics

- **eth0**: 192.168.128.6
- **eth1**: 192.168.128.50
- **eth2**: 192.168.129.106
- **eth1**: 192.168.128.51
- **eth2**: 192.168.129.107
- **ftp 192.168.128.51**

Network:
192.168.128.51

Connection Points:
- eth0: 192.168.128.51
- eth1: 192.168.128.51
- eth2: 192.168.129.106
Routing Table

- Static routing table in this assignment
- Make sure you understand LPM (Section 3)
- Loaded for you from the command line into the router context of type `struct sr_instance` (`sr_router.h`)
- Entries are a linked list of type `struct sr_rt` (`sr_rt.h`)
- Walk over the routing table linearly to do a longest prefix match on it

<table>
<thead>
<tr>
<th>IP address</th>
<th>Next-hop</th>
<th>Network Mask</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.128.51</td>
<td>192.168.128.51</td>
<td>255.255.255.255</td>
<td>eth1</td>
</tr>
<tr>
<td>192.168.129.106</td>
<td>192.168.129.106</td>
<td>255.255.255.255</td>
<td>eth2</td>
</tr>
<tr>
<td>0.0.0.0</td>
<td>172.24.74.17</td>
<td>0.0.0.0</td>
<td>eth0</td>
</tr>
</tbody>
</table>
Basic Forwarding Principles

• Remove IP datagram from Ethernet frame
  – Could also be an ARP payload in Ethernet frame
• Inspect the packet’s DA. Assuming the packet is destined for someone else:
  – Look up next-hop address by doing a LPM on the routing table using the packet’s DA
    • If it does not exist, send ICMP host unreachable
  – Decrement TTL, update header checksum
    • If TTL <= 1, send ICMP time exceeded
    • If checksum invalid, drop
  – From next-hop address, determine outgoing interface and next-hop MAC address
    • If necessary, send ARP request to determine MAC address
  – Encapsulate IP datagram in Ethernet packet
  – Forward packet to outgoing interface
Packets Destined for the Router

• If the packet’s DA is destined towards one of our interfaces:
  – If it’s an ICMP echo request, generate an ICMP echo reply
  – Otherwise if it’s a TCP or UDP packet, generate an ICMP port unreachable (needed for traceroute to work)
Checksums

• IP checksum
  – Need to check for all IP packet headers; drop packet if checksum is bad

• ICMP checksum
  – Need to validate for incoming packets destined for us
  – Need to calculate for outgoing packets
  – Ignore if forwarding

• TCP/UDP checksum
  – End-to-end checksum, ignore

• Use in_cksum from Lab 3 to calculate checksums
Handling ARP

• Routing table contains next-hop IPs, but you need both a next-hop IP and and a next-hop MAC address
• What you have to do:
  – Generate ARP requests and parse ARP replies
  – Listen to ARP requests and send ARP replies
  – Don’t send a request for each packet; instead, use an ARP cache
  – Requests should time out after 5 tries of about 1 second each
  – ARP cache entries should time out after about 15 seconds

<table>
<thead>
<tr>
<th>IP address</th>
<th>Ethernet MAC Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>172.24.74.130</td>
<td>00:e0:81:04:08:9b</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
ARP Cache Class

• We’ve given you an ARP cache class containing
  – ARP request queue
  – ARP cache

• ARP cache entries time out automatically after 15 seconds

• ARP request queue written so that it is easy consolidate ARP requests across multiple packets, and enforce the ARP request timeout

• See pseudocode in sr_arpcache.h
Required Functionality

• Forwarding packets should work
• Handles and generates ARP requests/replies correctly
• You can download a file using http and ftp from one of the app servers behind the router
• You can traceroute to and through the router
• You can ping to and through the router
• ICMP host and port unreachable messages are generated correctly
• ICMP time exceeded messages are generated correctly
• No shortcuts taken (don’t forward a packet to every interface)
• Should not crash, even with a malformed packet
Main Functions and Structures

• In sr_router.h:
  – struct sr_instance is the context of the router
• In sr_router.c:
  – sr_handlepacket is called for every packet that goes through the router—you have to fill it out
• sr_protocol.h contains convenience structs for accessing fields in packets
• sr_if.h contains methods for getting information about the router’s interfaces
• ARP Cache in sr_arpcache.h
Suggestions

• Think before coding—there’s a good amount of code to write
• Don’t put everything in one file, if you want to be able to navigate your code
• Read through the Network Sorcery pages to understand the protocols
• Read the FAQ
• Post and check the newsgroup
• Start early
Demo

- Look at working solution