CS 144 Lab 3

Section TAs: Tahir Azim, Kiran Isaac Abraham
Outline

- Protocol Layering
- IP forwarding, ARP and ICMP Review
- Lab 3 Description
- Wireshark Demo (Kiran)
TCP/IP Protocol Stack

A

App
TCP/UDP
IP
MAC

if0

Router

IP
MAC

if1

B

App
TCP/UDP
IP
MAC
TCP/IP Protocol Stack

A

App

TCP/UDP

IP

MAC

if0

Router

IP

MAC

ifl

B

App

TCP/UDP

IP

MAC
TCP/IP Protocol Stack

A

A

Router

B
TCP/IP Protocol Stack

A

Router

B

TCP/IP Protocol Stack

App

TCP/UDP

IP

MAC

if0

IP

MAC

ifl
TCP/IP Protocol Stack

A

Router

B

TCP/UDP

IP

MAC

App

TCP/UDP

IP

MAC

App

TCP/UDP

IP

MAC

if0

if1
TCP/IP Protocol Stack
TCP/IP Protocol Stack

A

App

TCP/UDP

IP

MAC

if0

Router

IP

MAC

if1

B

App

TCP/UDP

IP

MAC
TCP/IP Protocol Stack

A  \[\text{App} \rightarrow \text{TCP/UDP} \rightarrow \text{IP} \rightarrow \text{MAC}\]  if0

Router  \[\text{IP} \rightarrow \text{MAC}\]

B  \[\text{App} \rightarrow \text{TCP/UDP} \rightarrow \text{IP} \rightarrow \text{MAC}\]  if1
TCP/IP Protocol Stack

A

App

TCP/UDP

IP

MAC

if0

Router

IP

MAC

if1

B

App

TCP/UDP

IP

MAC
TCP/IP Protocol Stack

A

App

TCP/UDP

IP

MAC

if0

Router

IP

MAC

if1

B

App

TCP/UDP

IP

MAC
TCP/IP Protocol Stack

A

Router

B
Protocol Layer

- TCP/UDP Layer:
  - Application-to-application data transfer
- IP:
  - End host to end host data transfer
- MAC (Ethernet, WiFi etc):
  - One-hop data transfer
Protocol Encapsulation

Sending

Application data
Transport header
IP header
Link layer header

Receiving
Protocol Headers

• Each protocol’s header only relevant to that protocol
  • TCP header contains app address (port)
  • IP header contains end-host address (IP address)
  • MAC header contains one-hop MAC address
Implications

Diagram:

A

App
TCP/UDP
IP
MAC

Router

IP
MAC

B

App
TCP/UDP
IP
MAC
Implications

MAC layer needs to update src and next-hop MAC addresses, send on interface to next hop
Implications

MAC layer needs to update src and next-hop MAC addresses, send on interface to next hop

IP needs to decrement TTL, update checksum, find next hop

A

B
How to find the next hop?
How to find the next hop?

- IP addresses
How to find the next hop?

- IP addresses
  - Packets routed based on dest IP address
How to find the next hop?

- IP addresses
- Packets routed based on dest IP address
- Q: Does an Internet host have only one IP address?
How to find the next hop?

- IP addresses
  - Packets routed based on dest IP address
  - Q: Does an Internet host have only one IP address?
  - No, every interface on a host/router has a different IP address
How to find the next hop?

• Longest-prefix matching

• Find the most specific entry that matches the destination IP address
How to find the next hop?

- Longest-prefix matching
- Find the most specific entry that matches the destination IP address

```
How to find the next hop?

Prefix  Next-hop     Port
0.0.0.0/0  128.17.16.1  3
128.9/16   128.17.14.1  2
128.9.16/20 128.17.14.1  2
128.9.19/24 128.17.20.1  1
128.9.25/24 128.17.14.1  2
128.9.176/20 128.17.20.1  1
142.12/19   128.17.16.1  3
```

Forwarding/routing table

* Example from Nick McKeown's CS244A class*
How to find the next hop?

• Longest-prefix matching

• Find the **most specific** table entry that matches the destination IP address

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next-hop</th>
<th>Port</th>
<th>Address</th>
<th>Netmask</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0/0</td>
<td>128.17.16.1</td>
<td>3</td>
<td>0.0.0.0</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>128.9/16</td>
<td>128.17.14.1</td>
<td>2</td>
<td>128.9.0.0</td>
<td>255.255.0.0</td>
</tr>
<tr>
<td>128.9.16/20</td>
<td>128.17.14.1</td>
<td>2</td>
<td>128.9.16.0</td>
<td>255.255.240.0</td>
</tr>
<tr>
<td>128.9.19/24</td>
<td>128.17.20.1</td>
<td>1</td>
<td>128.9.19.0</td>
<td>255.255.255.0</td>
</tr>
<tr>
<td>128.9.25/24</td>
<td>128.17.14.1</td>
<td>2</td>
<td>128.9.25.0</td>
<td>255.255.255.0</td>
</tr>
<tr>
<td>128.9.176/20</td>
<td>128.17.20.1</td>
<td>1</td>
<td>128.9.176.0</td>
<td>255.255.240.0</td>
</tr>
<tr>
<td>142.12/19</td>
<td>128.17.16.1</td>
<td>3</td>
<td>142.12.0.0</td>
<td>255.255.224.0</td>
</tr>
</tbody>
</table>

* Example from Nick McKeown's CS244A class
How to find the next hop?

- Longest-Prefix Matching

  - For entry, i, in routing table, compute:
    - $\text{Subnet}_i = (\text{Address}_i) \& (\text{Netmask}_i)$

  - For entry, i, in routing table, compute:
    - $X_i = (\text{Dest\_IP}) \& (\text{Netmask}_i)$

  - Forward along interface with largest $X_i$ such that $X_i =\equiv \text{Subnet}_i$
How to get the MAC address of the next-hop?

- Address resolution protocol (ARP)
  - Request contains an IP address, get MAC address in reply
  - Broadcast (dest MAC: ff-ff-ff-ff-ff-ff) over the chosen interface
  - If IP address matches an interface, reply with MAC addr
How to get the MAC address of the next-hop?

- Address resolution protocol (ARP)
  - Request contains an IP address, get MAC address in reply
  - Broadcast (dest MAC: ff-ff-ff-ff-ff-ff-ff) over the chosen interface
  - If IP address matches an interface, reply with MAC addr
What if TTL falls to zero, or ARP reply not received?

- Internet Control Message Protocol (ICMP)
- Runs on top of IP
Lab 3 (Static Router)

- Lab3 setup
- Lab3 requirements
- Lab 3 hints
VNS/Lab 3 setup
VNS/Lab 3 setup

- Download and untar the assignment package:
  - http://www.scs.stanford.edu/llau-cs144/lab/router/router.tar.gz
VNS/Lab 3 setup

• Download and untar the assignment package:
  • \texttt{http://www.scs.stanford.edu/11au-CS144/lab/router/router.tar.gz}

• Copy your auth\_key to your lab directory
  • \texttt{/usr/class/cs144/lab3_11au/student_auths/<SUNET\_ID@stanford.edu>/auth\_key}
VNS/Lab 3 setup

• Download and untar the assignment package:
  • http://www.scs.stanford.edu/llau-cs144/lab/router/router.tar.gz

• Copy your auth_key to your lab directory
  • /usr/class/cs144/lab3_11au/student_auths/<SUNET_ID@stanford.edu>/auth_key

• If you don’t see your auth_key there... blame the TA!
Lab 3 setup

• *make* and then run:
  
  • `./sr -u <SUNET_ID> -T '1-router 2-server' -s vns-2.stanford.edu -r rtable.vrhost`

• This will download your routing table into rtable.vrhost

• Run the above command again, then ping one of the IP addresses
  
  • “Received packet of length 42”
Lab 3 setup

• rtable.vrhost file: your static routing table

0.0.0.0 172.24.74.17 0.0.0.0 eth0
10.3.0.219 10.3.0.219 255.255.255.254 eth1
10.3.0.217 10.3.0.217 255.255.255.254 eth2
Lab 3 setup

- rtable.vrhost file: your static routing table

```
0.0.0.0  172.24.74.17  0.0.0.0  eth0
10.3.0.219  10.3.0.219  255.255.255.254  eth1
10.3.0.217  10.3.0.217  255.255.255.254  eth2
```

Address/Prefix
## Lab 3 setup

- **rtable.vrhost file: your static routing table**

<table>
<thead>
<tr>
<th>Address</th>
<th>Prefix</th>
<th>Netmask</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td></td>
<td>172.24.74.17</td>
<td>eth0</td>
</tr>
<tr>
<td>10.3.0.219</td>
<td>10.3.0.219</td>
<td>255.255.255.254</td>
<td>eth1</td>
</tr>
<tr>
<td>10.3.0.217</td>
<td>10.3.0.217</td>
<td>255.255.255.254</td>
<td>eth2</td>
</tr>
</tbody>
</table>

Address/Prefix | Netmask
Lab 3 setup

• rtable.vrhost file: your static routing table

<table>
<thead>
<tr>
<th>Address/Prefix</th>
<th>Netmask</th>
<th>Next hop</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td>172.24.74.17</td>
<td>0.0.0.0</td>
<td>eth0</td>
</tr>
<tr>
<td>10.3.0.219</td>
<td>10.3.0.219</td>
<td>255.255.255.254</td>
<td>eth1</td>
</tr>
<tr>
<td>10.3.0.217</td>
<td>10.3.0.217</td>
<td>255.255.255.254</td>
<td>eth2</td>
</tr>
</tbody>
</table>
Lab 3 setup

- `rtable.vrhost file: your static routing table`

<table>
<thead>
<tr>
<th>Address/Prefix</th>
<th>Next hop</th>
<th>Netmask</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td>172.24.74.17</td>
<td>0.0.0.0</td>
<td>eth0</td>
</tr>
<tr>
<td>10.3.0.219</td>
<td>10.3.0.219</td>
<td>255.255.255.254</td>
<td>eth1</td>
</tr>
<tr>
<td>10.3.0.217</td>
<td>10.3.0.217</td>
<td>255.255.255.254</td>
<td>eth2</td>
</tr>
</tbody>
</table>
Lab 3 topology

What you will implement
Lab 3 requirements
Lab 3 requirements

• Implement a simple router
Lab 3 requirements

- Implement a simple router
  - Receive raw, valid Ethernet frame
Lab 3 requirements

- Implement a simple router
  - Receive raw, valid Ethernet frame
  - Sanity check (has minimum length, IP checksum correct) -- shouldn’t crash router
Lab 3 requirements

• Implement a simple router
  • Receive raw, valid Ethernet frame
  • Sanity check (has minimum length, IP checksum correct) -- shouldn’t crash router
  • Check if it’s meant for one of the router’s own interfaces
Lab 3 requirements

- Implement a simple router
  - Receive raw, valid Ethernet frame
  - Sanity check (has minimum length, IP checksum correct) -- shouldn’t crash router
  - Check if it’s meant for one of the router’s own interfaces
  - If not, decrement TTL, check if > 0, then update checksum
Lab 3 requirements

- Implement a simple router
  - Receive raw, valid Ethernet frame
  - Sanity check (has minimum length, IP checksum correct) -- shouldn’t crash router
  - Check if it’s meant for one of the router’s own interfaces
  - If not, decrement TTL, check if > 0, then update checksum
  - Find which entry in routing table has longest prefix match with packet destination
Lab 3 requirements

- Implement a simple router
  - Receive raw, valid Ethernet frame
  - Sanity check (has minimum length, IP checksum correct) -- shouldn’t crash router
  - Check if it’s meant for one of the router’s own interfaces
  - If not, decrement TTL, check if > 0, then update checksum
  - Find which entry in routing table has longest prefix match with packet destination
  - Check the ARP cache to find MAC for that next hop. If it’s there, send the packet. Else, enqueue packet, issue ARP request on that interface and send packet on ARP reply.
Required Functionality

- Route HTTP packets to and from app servers
- Handle ARP requests and replies
- Correctly handle traceroutes and pings
- Handle TCP/UDP packets to one of its interfaces, replying with ICMP unreachable
- For more, see lab webpage
Where to start
Where to start

• sr_router.c

• sr_handlepacket(): Receives incoming Ethernet frames; may be ICMP, UDP, ARP, TCP packets
Where to start

• sr_router.c

• sr_handlepacket(): Receives incoming Ethernet frames; may be ICMP, UDP, ARP, TCP packets

• sr_protocols.h

• Contains structures for all supported packet types
Where to start

• sr_router.c

• sr_handlepacket(): Receives incoming Ethernet frames; may be ICMP, UDP, ARP, TCP packets

• sr_protocols.h

• Contains structures for all supported packet types

• sr_send_packet(): Send an Ethernet frame
Where to start

• sr_router.h:

• Defines sr_instance struct which contains list of router’s interfaces, the routing table and the ARP cache (sr_if, sr_rt and sr_arpcache)
Where to start?

- `sr_arpcache.h/.c`
  - Maintains a cache of IP->MAC mappings
  - Maintains linked lists of packets waiting on ARP replies
  - Provides a function that expires old cache entries
  - Provides a stub function to retransmit ARP requests (`sr_arpcache_sweepreqs`)
    - Called roughly every second
  - You have to fill this out
    - Retransmit ARP request once per second up to 5 times
ARP: what you have to do...

- Broadcast ARP requests if IP->MAC mapping not found (sr_arpcache_lookup)
- Enqueue packets if IP->MAC mapping not found (sr_arpcache_queuereq)
- Handle ARP responses (sr_arpcache_insert and sr_arpreq->packets)
- Write sr_arpcache_sweepreqs
- Send ICMP messages if ARP reply not received after 5 retries
- Read comments in sr_arpcache.h
Lab 3 Hints

- Beware: \texttt{sr\_if}, \texttt{sr\_rt} and \texttt{sr\_arpcache} store IP addresses in network byte order
Lab 3 Hints

- Beware: sr_if, sr_rt and sr_arpcache store IP addresses in network byte order
- Quite a bit of coding, so start early!
Lab 3 Hints

- Beware: sr_if, sr_rt and sr_arpcache store IP addresses in network byte order
- Quite a bit of coding, so start early!
- sr_protocols.h
Lab 3 Hints

- Beware: sr_if, sr_rt and sr_arpcache store IP addresses in network byte order
- Quite a bit of coding, so start early!
- sr_protocols.h
- networksorcery.com protocol pages extremely helpful
Lab 3 Hints

• Beware: sr_if, sr_rt and sr_arpcache store IP addresses in network byte order
• Quite a bit of coding, so start early!
• sr_protocols.h
• networksorcery.com protocol pages extremely helpful
• Cleaner code simplifies debugging
Lab 3 Hints

- Beware: sr_if, sr_rt and sr_arpcache store IP addresses in network byte order
- Quite a bit of coding, so start early!
- sr_protocols.h
- networksorcery.com protocol pages extremely helpful
- Cleaner code simplifies debugging
- Use Wireshark for debugging!
Lab 3 Hints

- Beware: sr_if, sr_rt and sr_arpcache store IP addresses in network byte order
- Quite a bit of coding, so start early!
- sr_protocols.h
- networksorcery.com protocol pages extremely helpful
- Cleaner code simplifies debugging
- Use Wireshark for debugging!
  - “-l logname” option saves all sent and received packets in Wireshark format
Filters

- New feature this year
- Allow you to filter out packets based on source IP address
- Just specify the filter in the “filters” file
  - For example, “171.64.15.55/32” means only packets from 171.64.15.55 are passed to sr_handlepacket()
Kiran: Wireshark