Missing pieces
+
Putting the pieces together

CS 168, Fall 2014
Sylvia Ratnasamy

Material thanks to Ion Stoica, Scott Shenker, Jennifer Rexford, Nick McKeown, and many other colleagues
Today

• Switched Ethernet (wrap-up)
  – Frames and framing
  – MAC addresses
  – Routing
  – Forwarding

• Missing pieces and putting the pieces together
Last Time

• Switched Ethernet
  – Frame formats
  – MAC addresses
  – Spanning Tree approach
    • Take arbitrary topology
    • Pick subset of links that form a spanning tree

• Today: how do we forward over the spanning tree?
Flooding on a Spanning Tree

• Switches flood using the following rule:
  – (Ignoring all ports not on spanning tree!)
    – Originating switch sends packet out all ports
    – When a packet arrives on one incoming port, send it out all ports other than the incoming port
Flooding on Spanning Tree
Flooding on Spanning Tree
But isn’t flooding wasteful?

• Yes, but we can use it to bootstrap more efficient forwarding

• Idea: watch the packets going by, and learn from them
  – If node A sees a packet from node B come in on a particular port, it knows what port to use to reach B!
  – Works because there’s only one path to B
Nodes can “learn” routes

• Switch learns how to reach nodes by remembering where flooding packets came from
  – If flood packet from Node A entered switch on port 4, then switch uses port 4 to send to Node A
General Approach

• Flood first packet to node you are trying to reach

• All switches learn where you are

• When destination responds, some switches learn where it is...
  – Only some switches, because packet to you follows direct path, and is not flooded
Once a node has sent a flood message, all other switches know how to reach it...
When a node responds, **some** of the switches learn where it is
Ethernet switches are “self learning”

When a packet arrives:
• Inspect source MAC address, associate with incoming port
• Store mapping in the switch table
• Use time-to-live field to eventually forget mapping

Packet tells switch how to reach A.
Self Learning: Handling Misses

When packet arrives with unfamiliar destination

- Forward packet out all other ports
- Response may teach switch about that destination
Hence: Forwarding Rule

When switch receives a packet:
index the switch table using destination MAC
if entry found for destination {
  if dest on port from which packet arrived
    then *drop* packet
  else *forward* packet on port indicated
}
else *flood*

forward on all but the port on which the frame arrived

*Why do this?*
Summary of Learning Approach

• Avoids loop by restricting to spanning tree
• This makes flooding possible
• Flooding allows packet to reach destination
• And in the process switches learn how to reach source of flood
• No route “computation”
• Forwarding entries a consequence of traffic pattern
Contrast

• IP
  – Packets forwarded on all links
  – Aggregatable addresses
  – Routing protocol computes loop-free paths
  – Forwarding table computed by routing protocol

• Ethernet
  – Packets forwarded on subset of links (spanning tree)
  – Flat addresses
  – “Routing” protocol computes loop-free topology
  – Forwarding table derived from data packets (+SPT for floods)
Strengths of Ethernet’s Approach?

- Plug-n-Play: zero-configuration / self-*
- Simple
- Cheap?
Weaknesses of This Approach?

- Much of the network bandwidth goes unused
  - Forwarding is only over the spanning tree
- Delay in reestablishing spanning tree
  - Network is “down” until spanning tree rebuilt
  - And rebuilt spanning tree may be quite different
- Slow to react to host movement
  - Entries must time out
- Poor predictability
  - Location of root and traffic pattern determines forwarding efficiency
Today

• Switched Ethernet (wrap-up)
  – Frames and framing
  – MAC addresses
  – Routing
  – Forwarding

• Missing pieces and putting the pieces together
Discovery

• A host is “born” knowing only its MAC address

• Must discover lots of information before it can communicate with a remote host B
  – what is my IP address?
  – what is B’s IP address? (remote)
  – what is B’s MAC address? (if B is local)
  – what is my first-hop router’s address? (if B is not local)
  – ...

ARP and DHCP

• Link layer discovery protocols
  – ARP → Address Resolution Protocol
  – DHCP → Dynamic Host Configuration Protocol
  – confined to a single local-area network (LAN)
  – rely on broadcast capability
ARP and DHCP

• Link layer discovery protocols
• Serve two functions
  – Discovery of local end-hosts
    • for communication between hosts on the same LAN
ARP and DHCP

• Link layer discovery protocols

• Serve two functions
  – Discovery of local end-hosts
  – Bootstrap communication with remote hosts
    • what’s my IP address?
    • who/where is my local DNS server?
    • who/where is my first hop router?
DHCP

• “Dynamic Host Configuration Protocol”
  – defined in RFC 2131
• A host uses DHCP to discover
  – its own IP address
  – its netmask
  – IP address(es) for its local DNS name server(s)
  – IP address(es) for its first-hop “default” router(s)
DHCP: operation

1. One or more local DHCP servers maintain required information
   – IP address pool, netmask, DNS servers, etc.
   – application that listens on UDP port 67
DHCP: operation

1. One or more local DHCP servers maintain required information

2. Client **broadcasts** a DHCP discovery message
   - L2 broadcast, to MAC address FF:FF:FF:FF:FF:FF
DHCP: operation

1. One or more local DHCP servers maintain required information
2. Client broadcasts a DHCP discovery message
3. One or more DHCP servers responds with a DHCP “offer” message
   – proposed IP address for client, lease time
   – other parameters
DHCP: operation

1. One or more local DHCP servers maintain required information
2. Client broadcasts a DHCP discovery message
3. One or more DHCP servers responds with a DHCP “offer” message
4. Client broadcasts a DHCP request message
   – specifies which offer it wants
   – echoes accepted parameters
   – other DHCP servers learn they were not chosen
DHCP: operation

1. One or more local DHCP servers maintain required information
2. Client **broadcasts** a DHCP discovery message
3. One or more DHCP servers responds with a DHCP “offer” message
4. Client **broadcasts** a DHCP request message
5. Selected DHCP server responds with an ACK
DHCP: operation

1. One or more local DHCP servers maintain required information
2. Client broadcasts a DHCP discovery message
3. One or more DHCP servers responds with a DHCP “offer” message
4. Client broadcasts a DHCP request message
5. Selected DHCP server responds with an ACK

(DHCP “relay agents” used when the DHCP server isn’t on the same broadcast domain -- see text)
DHCP uses “soft state”

• Soft state: if not refreshed, state is forgotten
  – hard state: allocation is deliberately returned/withdrawn
  – e.g., used to track address allocation in DHCP

• Implementation:
  – address allocations are associated with a lease period
  – server: sets a timer associated with the record of allocation
  – client: must request a refresh before lease period expires
  – server: resets timer when a refresh arrives; sends ACK
  – server: reclaims allocated address when timer expires

• Simple, yet robust under failure
  – state always fixes itself in (small constant of) lease time
Soft state under failure

- What happens when host XYZ fails?
  - refreshes from XYZ stop
  - server reclaims \textit{a.b.c.d} after \textit{O}(lease period)
Soft state under failure

- What happens when server fails?
  - ACKs from server stop
  - XYZ releases address after O(lease period); send new request
  - A new DHCP server can come up from a `cold start’ and we’re back on track in ~lease time

\[
a.b.c.d \text{ is mine from (now, now+lease)}
\]
What happens if the network fails?
- refreshes and ACKs don’t get through
- XYZ release address; DHCP server reclaims it
Are we there yet?

What I learnt from DHCP
my IP: 1.2.3.48
netmask: 1.2.3.0/24 (255.255.255.0)
Local DNS: 1.2.3.156
router: 1.2.3.19
Sending Packets Over Link-Layer

- Link layer only understands MAC addresses
  - Translate the destination IP address to MAC address
  - Encapsulate the IP packet in a link-level (Ethernet) frame
ARP: Address Resolution Protocol

• Every host maintains an **ARP** table
  – list of (IP address $\rightarrow$ MAC address) pairs

• Consult the table when sending a packet
  – Map destination IP address to destination MAC address
  – Encapsulate the (IP) data packet with MAC header; transmit

• But: what if IP address **not** in the table?
  – Sender broadcasts: “**Who has IP address 1.2.3.156?**”
  – Receiver responds: “**MAC address 58-23-D7-FA-20-B0**”
  – Sender caches result in its ARP table
What if the destination is remote?

- Look up the MAC address of the first hop router
  - 1.2.3.48 uses ARP to find MAC address for first-hop router rather than ultimate destination IP address
- How does the red host know the destination is not local?
  - Uses netmask (discovered via DHCP)
- How does the red host know about 1.2.3.19?
  - Also DHCP
ARP header

- =1 for Ethernet
- =6 for Ethernet
- =4 for IPv4
- =0x0800 for IPv4
- 1=request; 2=reply

Hardware Type

Protocol Type

Hardware Address Length

Protocol Address Length

Opcode

Sender Hardware Address

Sender Protocol Address (bytes 1-2)

Sender Protocol Address (bytes 3-4)

Target Hardware Address

Target Protocol Address
Key Ideas in Both ARP and DHCP

• **Broadcasting**: Can use broadcast to make contact
  – Scalable because of limited size

• **Caching**: remember the past for a while
  – Store the information you learn to reduce overhead

• **Soft state**: eventually forget the past
  – Associate a time-to-live field with the information
  – … and either refresh or discard the information
  – Key for robustness in the face of unpredictable change
# Taking Stock: Naming

<table>
<thead>
<tr>
<th>Layer</th>
<th>Examples</th>
<th>Structure</th>
<th>Configuration</th>
<th>Resolution Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>App. Layer</td>
<td><a href="http://www.cs.berkeley.edu">www.cs.berkeley.edu</a></td>
<td>organizational hierarchy</td>
<td>~ manual</td>
<td>DNS</td>
</tr>
<tr>
<td>Network Layer</td>
<td>123.45.6.78</td>
<td>topological hierarchy</td>
<td>DHCP</td>
<td></td>
</tr>
<tr>
<td>Link layer</td>
<td>45-CC-4E-12-F0-97</td>
<td>vendor (flat)</td>
<td>hard-coded</td>
<td>ARP</td>
</tr>
</tbody>
</table>
Discovery mechanisms

We’ve see two approaches

– Broadcast (ARP, DHCP)
  • flooding doesn’t scale
  • no centralized point of failure
  • zero configuration

– Directory service (DNS)
  • no flooding / scalable
  • root of the directory is vulnerable (caching is key)
  • needs configuration to bootstrap (local, root servers, etc.)

Open: can we get Internet-scale yet zero config?
Steps in end-to-end communication

What do hosts need to know? And how do they find out?
Steps in reaching a destination

• First look up destination’s IP address
  – Need to know where my local DNS server is
    • DHCP

  – Also needs to know my own IP address
    • DHCP
Sending a Packet

• On same subnet:
  – Use MAC address of destination
  – ARP

• On some other subnet:
  – Use MAC address of first-hop router
  – DHCP + ARP

• And how can a host tell whether destination is on same or other subnet?
  – Use the netmask
  – DHCP
Example: A sending a packet to B

How does host A send an IP packet to host B?
Example: A sending a packet to B

1. A sends packet to R.
2. R sends packet to B.
A sends packet through router R

- Host A constructs an IP packet to send to B
  - Source 111.111.111.111, destination 222.222.222.222
- Host A has a gateway router R
  - Used to reach destinations outside of 111.111.111.0/24
  - Address 111.111.111.110 for R learned via DHCP
A sends packet through router R

- Host A learns the MAC address of R’s interface
  - ARP request: broadcast request for 111.111.111.110
  - ARP response: R responds with E6-E9-00-17-BB-4B
- Host A encapsulates the IP packet for B, and sends to R
Two points:

- IP routing table points to this port
- Destination address is within mask of port’s address (i.e., local)

- Router $R$ consults its forwarding table
  - Packet matches $222.222.222.0/24$ via other adapter (port)
**R sends packet to B**

- Router R’s learns the MAC address of host B
  - ARP request: broadcast request for 222.222.222.222
  - ARP response: B responds with 49-BD-D2-C7-56-2A
- Router R encapsulates the packet and sends to B
• Are we there yet?
  – Yes!
Putting the pieces together

Walk through the steps required to download [www.google.com/index.html](http://www.google.com/index.html) from your laptop

- Assume: ‘cold start’ -- nothing cached anywhere
- Assume: yourDNS on a different subnet from yourDHCP
- Ignore intra- and interdomain routing protocols
Step 1: Self discovery

• You use DHCP to discover bootstrap parameters
  – your IP addr (u.u.u.u)
  – your DNS server’s IP (u.dns.ip.addr)
  – R’s IP address (r.r.r.r)
  – ...

• Exchange between you and yourDHCP

<table>
<thead>
<tr>
<th>Ethernet</th>
<th>IP</th>
<th>UDP</th>
<th>DHCP</th>
</tr>
</thead>
</table>

• Protocol count = 4
• You are ready to contact www.google.com
  → need an IP address for www.google.com
  → need to ask google’s DNS server
  → need to ask my DNS server to ask google’s DNS...
  → I know my DNS server’s IP addr is u.dns.ip.addr
  → create a packet to send...

<table>
<thead>
<tr>
<th>Ethernet</th>
<th>IP</th>
<th>UDP</th>
<th>DNS</th>
</tr>
</thead>
</table>

source: u.u.u..u
dst: u.dns.ip.addr

destination MAC?
Step 2: Getting out the door

- **You** use ARP to discover the MAC address of R
- Exchange between you and R
  - Ethernet
  - ARP
- Protocol count = 5
Step 3: Send a DNS request

- Exchange between you and yourDNS
- Now ready to send that packet

<table>
<thead>
<tr>
<th>Ethernet</th>
<th>IP</th>
<th>UDP</th>
<th>DNS</th>
</tr>
</thead>
</table>

R’s MAC

source: u.u.u..u
dst: u.dns.ip.addr

Protocol count = 6
Step 4: yourDNS does its thing

- yourDNS resolves www.google.com

- Protocol count = 6
Step 5: Getting the content (at last)

- Exchange between you and google’s server at g.g.g.g

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>u.u.u.u</td>
<td>g.g.g.g</td>
</tr>
<tr>
<td>IP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Protocol count = 8
Recap: Name discovery/resolution

• MAC addresses?
  – my own: hardcoded
  – others: ARP (given IP address)

• IP addresses?
  – my own: DHCP
  – others: DNS (given domain name)
    • how do I bootstrap DNS communication? (DHCP)

• Domain names?
  – search engines