Lecture 9.

Direct Datagram Forwarding:

Address Resolution Protocol (ARP)
Routing decision for packet X has two possible outcomes:

- You are arrived to the final network: go to host X
- You are not arrived to the final network: go through router interface Y

In both cases we have an IP address on THIS network. How can we send data to the interface?

Need to use physical network facilities!
Reaching a physical host

- IP addresses only make sense to TCPIP protocol suite
- Physical networks have their own hardware address
  - e.g. 48 bits Ethernet address, 16 or 48 bits Token Ring, 16 or 48 bit FDDI, ...
  - Datalink layers may provide the basis for several network layers, not only IP!

Address Resolution Protocol
RFC 826
Here described for Ethernet, but more general: designed for any datalink with broadcast capabilities
Manual mapping

→ A possibility, indeed!!
  ⇒ Nothing contrary, in principle
    ⇒ actually done in X.25, ISDN (do not support broadcast)
  ⇒ Simply keep in every host a mapping between IP address and hardware address for every IP device connected to the considered network

→ drawbacks
  ⇒ tedious
  ⇒ error prone
  ⇒ requires manual updating
    ⇒ e.g. when attaching a new PC, must touch all others...
ARP

- **Dynamic mapping**
  - not a concern for application & user
  - not a concern for system administrator!

- **Any network layer protocol**
  - not IP-specific

- **Supported protocol in datalink layer**
  - not a datalink layer protocol !!!!

- **Need datalink with broadcasting capability**
  - e.g. ethernet shared bus
ARP idea

Who has IP address 131.175.15.124 ?

→ Send broadcast request

→ receive unicast response

It’s me! I have 00:00:a2:32:5a:3

131.175.15.124

131.175.15.12

131.175.15.8

Not me!
ARP cache

→ Avoids arp request for every IP datagram!
   ⇒ Entry lifetime defaults to 20min
   → deleted if not used in this time
   → 3 minutes for “incomplete” cache entries (i.e. arp requests to non existent host)
   → it may be changed in some implementations
      » in particularly stable (or dynamic) environments

⇒ **arp -a** to display all cache entries (arp –d to delete)

*try a traceroute or ping to check ARP caching!*
   ⇒ First packet generally delays more
   ⇒ includes an ARP request/reply!
ARP request/reply
Incapsulation in Ethernet Frame

- **Ethernet Destination Address**
  - ff:ff:ff:ff:ff:ff (broadcast) for ARP request

- **Ethernet Source Address**
  - of ARP requester

- **Frame Type**
  - ARP request/reply: 0x0806
  - RARP request/reply: 0x8035
  - IP datagram: 0x0800

Protocol demultiplexing codes!
**ARP request/reply format**

<table>
<thead>
<tr>
<th>0</th>
<th>7</th>
<th>8</th>
<th>15</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Type</td>
<td>Protocol Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware len</td>
<td>Protocol len</td>
<td>ARP operation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sender MAC address (bytes 0-3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sender MAC address (bytes 4-5)</td>
<td>Sender IP address (bytes 0-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sender IP address (bytes 2-3)</td>
<td>Dest MAC address (bytes 0-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dest MAC address (bytes 2-5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dest IP address (bytes 0-3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hardware type:** 1 for ethernet  
**Protocol type:** 0x0800 for IP (0000.1000.0000.0000)  
⇒ *the same of Ethernet header field carrying IP datagram!*

**Hardware len** = length in bytes of hardware addresses (6 bytes for ethernet)  
**Protocol len** = length in bytes of logical addresses (4 bytes for IP)  
**ARP operation:** 1=request; 2=reply; 3/4=RARP req/reply
Sample ARP request/reply

Ethernet Packet: ARP REQUEST

<table>
<thead>
<tr>
<th>FF:FF:FF:FF:FF:FF:FF</th>
<th>dest MAC</th>
<th>00:00:8c:3d:54:01</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0806</td>
<td>src MAC</td>
<td>0x0800</td>
</tr>
<tr>
<td>0x0001</td>
<td></td>
<td>0x0001</td>
</tr>
<tr>
<td>0x06  0x04  0x0001</td>
<td></td>
<td>0x0001</td>
</tr>
<tr>
<td>00:00:8c:3d:54:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>131.175.15.8</td>
<td>src IP</td>
<td>131.175.15.24</td>
</tr>
<tr>
<td>00:00:00:00:00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>131.175.15.24</td>
<td>dest MAC</td>
<td>00:00:8c:3d:54:01</td>
</tr>
<tr>
<td>checksum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ethernet Packet: ARP reply

<table>
<thead>
<tr>
<th>00:00:8c:3d:54:01</th>
<th>src MAC</th>
<th>0x0800</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td></td>
<td>0x0001</td>
</tr>
<tr>
<td>0x06  0x04  0x0002</td>
<td></td>
<td>0x0002</td>
</tr>
<tr>
<td>00:4f:33:3:ee:67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>131.175.15.24</td>
<td>src IP</td>
<td>131.175.15.15.24</td>
</tr>
<tr>
<td>00:00:8c:3d:54:01</td>
<td>dest MAC</td>
<td>00:4f:33:3:ee:67</td>
</tr>
<tr>
<td>131.175.15.8</td>
<td>dest IP</td>
<td>131.175.15.15.24</td>
</tr>
<tr>
<td>checksum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ARP frame type
Ethernet / IP
MAC=6 / IP=4 / rq=1, rpl=2
checksum
ARP cache updating

- ARP requests carry requestor IP/MAC pair
- ARP requests are broadcast
  ⇨ thus, they MUST be read by everyone
- Therefore, it comes for free, for every computer, to update its cache with requestor pair

- Cannot do this with ARP reply, as it is unicast!
Proxy ARP

- Device that responds to an ARP request on behalf of some other machine
  - allows having ONE logical (IP) network composed of more physical networks
  - especially important when different technologies used (e.g. 100 PC ethernet + 2 PC dialup SLIP)

ARP request for 131.175.15.24

ARP reply on behalf of 131.175.15.24

*returns router MAC address! Then router will forward packets to remote host*
Gratuitous ARP

→ ARP request issued by an IP address and addressed to the same IP address!!
   → Clearly nobody else than ME can answer!
   → WHY asking the network which MAC address do I have???

→ Two main reasons:
   → determine if another host is configured with the same IP address
     → in this case respond occurs, and MAC address of duplicated IP address is known.
   → Use gratuitous ARP when just changed hardware address
     → all other hosts update their cache entries!
     → A problem is that, despite specified in RFC, not all ARP cache implementations operate as described….
ARP: not only *this* mechanism!

- Described mechanism for broadcast networks (e.g. based on shared media)
- Non applicable for non broadcast networks
  - In this case OTHER ARP protocols are used
    - e.g. distributed ARP servers
    - e.g. algorithms to map IP address in network address
Getting an IP address:

Reverse Address Resolution Protocol (RARP)
The problem

- Bootstrapping a diskless terminal
  - this was the original problem in the 70s and 80s
- Reverse ARP [RFC903]
  - a way to obtain an IP address starting from MAC address
- Today problem: dynamic IP address assignment
  - limited pool of addresses assigned only when needed
- RARP not sufficiently general for modern usage
  - BOOTP (Bootstrap Protocol - RFC 951): significant changes to RARP (a different approach)
  - DHCP (Dynamic Host Configuration Protocol - RFC 1541): extends and replaces BOOTP
**RARP packet format**

almost identical to ARP. Differences:

<table>
<thead>
<tr>
<th>Field</th>
<th>Original ARP</th>
<th>RARP Request</th>
<th>RARP Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>6 bytes</td>
<td>6 bytes</td>
<td>28 bytes</td>
</tr>
<tr>
<td>Source address</td>
<td>6 bytes</td>
<td>6 bytes</td>
<td>28 bytes</td>
</tr>
<tr>
<td>Destination address</td>
<td>2 bytes</td>
<td>2B</td>
<td></td>
</tr>
<tr>
<td>Type:</td>
<td>Dest MAC addr (bytes 0-1)</td>
<td>Dest IP addr (bytes 0-3)</td>
<td>Dest MAC addr (bytes 2-5)</td>
</tr>
<tr>
<td>Hardware Type</td>
<td>3 bytes</td>
<td>3 bytes</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Hardware len</td>
<td>2 bytes</td>
<td>2 bytes</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Protocol len</td>
<td>2 bytes</td>
<td>2 bytes</td>
<td>4 bytes</td>
</tr>
<tr>
<td>oper:</td>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

- **ftyp:** 0x8035
- **Sender MAC address:** (bytes 0-3)
- **Sender MAC address:** (bytes 4-5)
- **Sender IP address:** (bytes 0-1)
- **Sender IP address:** (bytes 2-3)
- **Dest MAC address:** (bytes 0-1)
- **Dest MAC address:** (bytes 2-5)
- **Dest IP address:** (bytes 0-3)
IP = ????
MAC = 0:0:8c:3d:54:1

My MAC address is 0:0:8c:3d:54:1. What is my IP address?

Broadcast request

Your IP is 131.175.21.53

Unicast reply
RARP problems

Network traffic

- for reliability, multiple RARP servers need to be configured on the same Ethernet
  - to allow bootstrap of terminals even when one server is down
- But this implies that ALL servers simultaneously respond to RARP request
  - contention on the Ethernet occurs

RARP requests not forwarded by routers

- being hardware level broadcasts...
RARP fundamental limit

➤ Allows only to retrieve the IP address information
  ➞ and what about all the remaining full set of TCPIP configuration parameters???
    ➤ Netmask?
    ➤ name of servers, proxies, etc?
    ➤ other proprietary/vendor/ISP-specific info?

➤ This is the main reason that has driven to engineer and use BOOTP and DHCP
BOOTP/DHCP approach

- Requests/replies encapsulated in UDP datagrams
  - may cross routers
  - no more dependent on physical medium

- request addressing:
  - destination IP = 255.255.255.255
  - source IP = 0.0.0.0
  - destination port (BOOTP): 67
  - source port (BOOTP): 68

- router crossing:
  - router configured as BOOTP relay agent
  - forwards broadcast UDP requests with destination port 67
BOOTP parameters exchange

Many more parameters

- Client IP address (when static IP is assigned)
- Your IP address (when dynamic server assignment)
- Gateway IP address (bootp relay agent - router - IP)
- Server hostname
- Boot filename

Fundamental: vendor-specific information field (64 bytes)

- Seems a lot of space: not true!
- DHCP uses a 312 vendor-specific field!
Vendor specific information format allows general information exchange

<table>
<thead>
<tr>
<th>Tag (1 byte)</th>
<th>Len (1 byte)</th>
<th>Parameter exchanged</th>
</tr>
</thead>
</table>

- **E.g.: subnet mask:**
  - tag=1, len=4, parameter=32 bit subnet mask

- **E.g.: time offset:**
  - tag=2, len=4, parameter=time
    - (seconds after midnight, Jan 1 1900 UTC)

- **E.g. gateway (variable item):**
  - tag=3, len=N, list of gateway IPaddr (first preferred)

- **E.g. DNS server (tag 6)**
Software Defined Networking
Outline

» What is SDN?
  > Background
  > An OS for networks

» What is OpenFlow?
  > How it helps SDN

» The current status & the future of SDN

» Conclusions
Limitations of Current Networks

http://www.excitingip.net/27/a-basic-enterprise-lan-network-architecture-block-diagram-and-components/
Limitations of Current Networks

» Enterprise networks are difficult to manage

» “New control requirements have arisen”:
  > Greater scale
  > Migration of VMS

» How to easily configure huge networks?
Limitations of Current Networks

» Old ways to configure a network
Limitations of Current Networks

- Million of lines of source code
- Billions of gates
- Many complex functions baked into infrastructure
  - OSPF, BGP, multicast, differentiated services, Traffic Engineering, NAT, firewalls, ...

Cannot dynamically change according to network conditions
Limitations of Current Networks

» No control plane abstraction for the whole network!

» It’s like old times – when there was no OS...

Wilkes with the EDSAC, 1949
Idea: An OS for Networks

Closed

Operating System
Specialized Packet Forwarding Hardware
App
App
App
App

Operating System
Specialized Packet Forwarding Hardware
App
App
App
App

Operating System
Specialized Packet Forwarding Hardware
App
App
App
App

Operating System
Specialized Packet Forwarding Hardware
App
App
App
App

OpenFlow/SDN tutorial, Srini Seetharaman, Deutsche Telekom, Silicon Valley Innovation Center
Idea: An OS for Networks

Control Programs

Network Operating System

- Operating System
- Specialized Packet Forwarding Hardware
- App

OpenFlow/SDN tutorial, Srini Seetharaman, Deutsche Telekom, Silicon Valley Innovation Center
Idea: An OS for Networks

Control Programs

Network Operating System

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

OpenFlow/SDN tutorial, Srini Seetharaman, Deutsche Telekom, Silicon Valley Innovation Center
Idea: An OS for Networks

"NOX: Towards an Operating System for Networks"

Software-Defined Networking (SDN)

Control Programs

Global Network View

Network Operating System

Control via forwarding interface

Protocols

Protocols
Software Defined Networking

- No longer designing distributed control protocols
  - Much easier to write, verify, maintain, ...
    > An interface for programming
  - NOS serves as fundamental control block
    > With a global view of network
Software Defined Networking

» Examples

> Ethane: network-wide access-control

> Power Management
Software Defined Networking

» Questions:

> How to obtain global information?
> What are the configurations?
> How to implement?
> How is the scalability?
> How does it really work?
Outline

» What is SDN?
  > Limitations of current networks
  > The idea of Network OS

» What is OpenFlow?
  > How it helps SDN

» The current status & the future of SDN

» Conclusions
OpenFlow

» “OpenFlow: Enabling Innovation in Campus Networks”

» Like hardware drivers
  – interface between switches and Network OS
OpenFlow

Control Path (Software)

Data Path (Hardware)
OpenFlow

OpenFlow Controller

OpenFlow Protocol (SSL/TCP)

Control Path

OpenFlow

Data Path (Hardware)
OpenFlow Switching

OpenFlow Client

<table>
<thead>
<tr>
<th>MAC src</th>
<th>MAC dst</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>*</td>
<td>port 1</td>
</tr>
</tbody>
</table>

## OpenFlow Table Entry

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch Port</td>
<td>MAC src</td>
<td>MAC dst</td>
</tr>
<tr>
<td>+ mask</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline
5. …

Packet + byte counters

## OpenFlow Examples

### Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>00:1f:..</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Routing

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>*</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Firewall

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>22</td>
<td>drop</td>
</tr>
</tbody>
</table>
Alice’s code:
- Simple learning switch
- Per Flow switching
- Network access control/firewall
- Static "VLANs"
- Her own new routing protocol: unicast, multicast, multipath
- Home network manager
- Packet processor (in controller)
- IPvAlice
OpenFlow

» Standard way to control flow-tables in commercial switches and routers

» Just need to update firmware

» Essential to the implementation of SDN
Centralized/Distributed Control

» “Onix: A Distributed Control Platform for Large-scale Production Networks”
Outline

» What is SDN?
  > Limitations of current networks
  > The idea of Network OS

» What is OpenFlow?
  > How it helps SDN

» The current status & the future of SDN

» Conclusions
Current status of SDN

» Hardware support

- Juniper MX-series
- NEC IP8800
- WiMax (NEC)
- HP Procurve 5400
- Netgear 7324
- PC Engines
- Pronto 3240/3290
- Ciena Coredirector

More coming soon...
Current status of SDN

» Industry support

> Google built hardware and software based on the OpenFlow protocol
> VMware purchased Nicira for $1.26 billion in 2012
> IBM, HP, NEC, Cisco and Juniper also are offering SDNs that may incorporate OpenFlow, but also have other elements that are specific to that vendor and their gear.

http://gigaom.com/2012/03/19/are-vendors-closing-openflow/
http://gigaom.com/2012/12/17/2012-the-year-software-defined-networking-sold-out/
http://www.extremetech.com/internet/140459-networking-is-getting-better-and-thats-partly-thanks-to-google
Future Focuses of SDN

» Research focuses

  > SIGCOMM HotSDN 2012
  > Mostly implementations of newly proposed systems, frameworks, or applications
Future Focuses of SDN

» New policies for security
» Programmable WLANs
» The placement of controllers (amount; location; centralized/distributed)
» Debugger for SDN

* All references are listed in the last slide.
Conclusions

» What is SDN?
   > A system-layered abstraction
   > Programmable, flexible, and extensible

» What is OpenFlow?
   > Interface between switches and controllers
   > Enabling SDN

» Future SDN
   > Enabling innovation
Thanks!

songchun.fan@duke.edu
References

» Natasha Gude et al., “NOX: Towards an Operating System for Networks”
» Nick McKeown et al., “OpenFlow: Enabling Innovation in Campus Networks”
» D Levin et. al., “Logically centralized?: state distribution trade-offs in software defined networks”, HotSDN 2012
» Brandon Heller et al., “The controller placement problem”
» N Handigol et. al., “Where is the debugger for my Software-Defined Network?”, HotSDN 2012
» L Suresh et. al., “Towards programmable enterprise WLANS with Odin”, HotSDN 2012