Last Time:

- transport layer
  - UDP
  - TCP
This Time

- TCP
- some basic protocols for getting started
  - RARP
  - BOOTP
  - DHCP
- Domain Name Service
- sample service: mail

Getting Started

- how does a host get 'on the air' when starting up?
- needs to know certain things (in order):
  - its own IP address
  - its own address mask
  - a default gateway
- usually can be stored in local disk files
  - but what if there's no local disk?
Step By Step

- problem:
  - have hardware address (read from ROM on interface card)
  - need IP address
- similar problem seen before?
  - have IP address and need hardware address
  - used ARP to resolve
- this is reverse of earlier problem, so...

RARP

- Reverse Address Resolution Protocol: RARP [RFC903]
- given hardware address, return corresponding IP address
- packet format very similar to ARP
  - "op type" different values
    - 3 for RARP request
    - 4 for RARP response
- request normally broadcast, response normally unicast
RARP

- server receives request
  - e.g., Ethernet frame type 0x8035
- resolving binding usually requires server to consult a file
  - containing IP:hardware_address entries
- server sends unicast reply
  - to whom?

ICMP Address Mask request/reply

- recall ICMP has two main jobs:
  - carry error messages back to a sender
  - perform request/reply information acquisition
  - distinguished by type field in pkt
    - see slide 04.20
- host sends broadcast ICMP msg
  - op type 17: address mask request
- server replies with unicast ICMP msg
  - op type 18: address mask reply
ICMP Router Solicitation

- ICMP to the rescue again
  - msg type 10: router solicitation
  - msg type 9: router advertisement
- on startup, host sends 3 requests, 3 seconds apart
  - stops as soon as first router advertisement arrives
- routers use type 9 routinely to advertise routes
  - can advertise multiple routers per advertisement
  - gives each a 'preference' level from its p.o.v.

Or, All At Once

- a host can get all of this information (and more) in a more convenient, single-step operation
- uses a different protocol: Bootstrap Protocol BOOTP [RFC951]
- BOOTP can, in a single interaction, provide:
  - IP address
  - router IP address
  - name of bootstrap file to load (to get OS)
BOOTP Messages

<table>
<thead>
<tr>
<th>OP</th>
<th>HTYPE</th>
<th>HLEN</th>
<th>HOP_CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION NUMBER</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Seconds elapsed [16]:
set by client when sends request as
time since began attempt to boot

client hardware address [128]: set by client to its
interface hardware address

client IP address [32]: set to
0 if client doesn’t know its IP addr
IP address of client if it does know
### BOOTP Messages

<table>
<thead>
<tr>
<th>OP</th>
<th>HTYPE</th>
<th>HLEN</th>
<th>HOP_CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION NUMBER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECONDS ELAPSED</td>
<td>UNUSED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLIENT IP ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YOUR IP ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERVER IP ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROUTER IP ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLIENT HARDWARE ADDRESS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERVER HOSTNAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOOT FILE NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[vendor-specific area [512]:]

### BOOTP: how delivered?

- RARP, ICMP use IP datagrams directly
- BOOTP uses UDP (port 67, 68)

![Diagram showing IP HDR, UDP HDR, and BOOTP REQUEST/REPLY]

- client usually sends as link-layer broadcast
  - with IP address 255.255.255.255 (limited b'cast)
  - what's source IP address?
Chicken and Egg Problem

- booting client asks some server for info
  - uses broadcast
- server responds sending UDP unicast reply
  - since server knows IP address
- but: think about what the server does...

Chicken and Egg Problem

- but: think about what the server does...
  - server is a process running on some host
  - has IP address of target
  - will send a UDP unicast to the designated IP addr
  - lower-level wants to map target IP address to local hardware address
  - looks in ARP table but doesn’t find target IP addr
Chicken and Egg Problem

- but: think about what the server does...
  - server is a process running on some host
  - has IP address of target
  - will send a UDP unicast to the designated IP addr
  - lower-level wants to map target IP address to local hardware address
  - looks in ARP table but doesn’t find target IP addr
  - so sends ARP request for anyone to tell it
  - but nobody knows, least of all client who asked

Chicken and Egg Problem

- solutions:
  1. BOOTP server software should make entry in server host’s ARP table before sending reply
  2. use broadcast for reply (not highly recommended)
Getting on the Internet

- RARP
  - used when need IP address given MAC address
  - use in conjunction with ICMP requests to locate gateway(s) and obtain mask
- BOOTP
  - an “one-stop–shopping” protocol
  - who tells the servers what to say?

Static or Dynamic?

- servers answering BOOTP or RARP requests look–up answers in tables
- tables are manually created and maintained
- good in relatively static configuration settings
  - an office
- bad when configuration changes often and rapidly
  - Internet café
Dynamic Setting

- hosts appear and disappear easily
  - want IP address fast when appear

- support many different hosts
  - but only ‘few’ at a time
Dynamic Setting

- hosts appear and disappear easily
  - want IP address fast when appear
- support many different hosts
  - but only ‘few’ at a time
- could re-use IP addresses
  - but need automated scheme to manage

---

Dynamic Host Configuration Protocol

- DHCP [rfc2131]
- provides for
  - permanent addresses
  - temporary, re–usable, addresses drawn from pool of available addresses
- follows client–server model
DHCP

- client sends DHCP request
  - broadcasts using UDP
  - request can cross ‘DHCP relays’
- server replies – offers – new address
  - offers for fixed time period: lease time
  - address reclaimed at end of lease
  - can be offered to another requesting node
  - client can negotiate to renew its lease

DHCP Messages

<table>
<thead>
<tr>
<th>OP</th>
<th>HTYPE</th>
<th>HLEN</th>
<th>HOP_CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION NUMBER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECONDS ELAPSED</td>
<td>FLAGS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLIENT IP ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YOUR IP ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERVER IP ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROUTER IP ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLIENT HARDWARE ADDRESS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERVER HOSTNAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOOT FILE NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTIONS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DHCP: Addresses, not Names

- no formal relationship between DHCP and DNS
  - so when a node's IP address changes, DHCP doesn’t do anything for name changing
  - name must be changed if IP address changed
Names and Numbers

- all protocols refer to nodes by their address
  - IP address in Internet
  - humans find “dotted–decimal” address inconvenient
- names are easier to use and remember

Domain Names

- names are strings separated by dots
- ordering reflects an organization scheme
  - more local as go leftward
  - e.g., somenode.netlab.gmu.edu

- names separated by dots are independent of numbers separated by dots in dotted–decimal addresses
Domain Names

- naming provides for natural hierarchy, e.g., somenode.netlab.gmu.edu appears as:

  - edu
    - gmu
      - netlab
        - somenode

- but can have other info in the tree...
Domain Names

- and have more than one such tree:

TLDs

- originally had 7 TLDs: .com .mil .edu .net .gov .org .int
- plus country domains
  - two letter abbreviations e.g., .us .uk
- extended list to include:
  .aero .info .pro .biz .museum .coop .name
**What’s In A Name**

- TLDs ‘controlled’ by ICANN [http://www.icann.org](http://www.icann.org)
  - designates registrars to oversee domains
- to get a domain name, must register it with designated registrar
- beyond the TLD, division of subtree is organization dependent
- each domain name is unique
- allocated FCFS
  - source of much legal contention
  - profitable opportunity?

---

**Names and Numbers**

- all protocols refer to nodes by their address
  - IP address in Internet
- humans find “dotted–decimal” address inconvenient
- names are easier to use and remember
- but names are hard for computers to work with
  - need for service to translate between IP addresses and names
Requirements of a Name Service

- initially, believed one server with daily updates would suffice
  - every node would contact the central server for the mapping between hostname and IP address
- this did not work
- lessons learned from this led to current service:

Requirements of a Name Service

- distributed database
  - no one location contains all information
Requirements of a Name Service

- distributed database
- hierarchical database
  - introduce levels (call them domains)
  - keep more detailed information in low levels, less detailed info in higher levels

- robustness and reliability
  - cannot ever be unable to resolve query because name service was unavailable
  - importance rises going up hierarchy
  - must have high enough performance to continue working well under high load
Requirements of a Name Service

- distributed database
- hierarchical database
- robustness and reliability
- autonomy
  - organizations can name local hosts without needing ‘central’ authorization

What Apps Want

- an application may be asked to access a resource on a remote machine
  - remote target referenced by name
- app needs to resolve name into address
  - use resolver
- resolver consults name service asking for address
  - asks its nameserver
  - in UNIX systems, look in /etc/resolv.conf
Resolving Names

- nameserver receives request from client
- nameserver either knows or doesn’t
- if doesn’t know:
  - it asks root server
  - i.e., it becomes a client asking for resolution
- root server either:
  - has answer, or
  - name of another nameserver who should have answer
Resolving Names

- some nameserver is the authoritative nameserver for the domain in question
  - provides authoritative answer: either address or indication that the sought name doesn't exist
- UNIX provides command-line resolvers
- e.g., dig osf1.gmu.edu replies with:

```plaintext
;; ANSWER SECTION:
osf1.gmu.edu. 30139 IN A 129.174.1.13
;; AUTHORITY SECTION:
gmu.edu. 85749 IN NS portal.gmu.edu.
gmu.edu. 85749 IN NS sargon.gmu.edu.
```

- UNIX provides resolver for use by programs as set of library functions:
  - `struct hostent *gethostbyname()`
  - `struct hostent *gethostbyaddr()`

```c
struct hostent
{
  char *h_name; /* Official name of host. */
  char **h_aliases; /* Alias list. */
  int h_addrtype; /* Host address type. */
  int h_length; /* Length of address. */
  char **h_addr_list; /* List of addresses from name server. */
};
```
Sample DNS Record

- db in nameserver is a text file containing resource records
  - syntax: `<domain_name><ttl><class><type><value>`
  - example entry: (from Tanenbaum⁴, fig. 7–3)
    - `flits.cs.vu.nl 86400 IN HINFO Sun Unix`
    - `flits.cs.vu.nl 86400 IN A 192.31.231.165`
    - `flits.cs.vu.nl 86400 IN MX 1 flits.cs.vu.nl`
    - `flits.cs.vu.nl 86400 IN MX 2 zephyr.cs.vu.nl`
    - `www.cs.vu.nl 86400 IN CNAME star.cs.vu.nl`
    - `ftp.cs.vu.nl 86400 IN CNAME zephyr.cs.vu.nl`

---

Sample Resource Record

- `flits.cs.vu.nl 86400 IN HINFO Sun Unix`
- `flits.cs.vu.nl 86400 IN A 192.31.231.165`
- `flits.cs.vu.nl 86400 IN MX 1 flits.cs.vu.nl`
- `flits.cs.vu.nl 86400 IN MX 2 zephyr.cs.vu.nl`
- `www.cs.vu.nl 86400 IN CNAME star.cs.vu.nl`
- `ftp.cs.vu.nl 86400 IN CNAME zephyr.cs.vu.nl`

- **Class:** IN for Internet
- **Canonical Name:** establish aliases for a host
- **Address:** IP address
- **Host Info:** CPU OS
- **Mail eXchange:** ≥ 1 name to which email goes
DNS Messages

Identification [16]: value assigned by client, returned by server, to let client match requests with answers

OP Code [4]: 0 ⇒ normal query
TC[1]: 0 if not, 1 if truncated answer (was > 512 bytes)
QR [1]: 0 if query, 1 if response
AA[1]: 0 if not, 1 if is authoritative answer
RA[1]: 0 if not, 1 if is recursion available
RD[1]: 0 if not, 1 if recursion desired

Return Code[4]: 0 if no error, 3 if name error
DNS Messages

- sent via UDP
A Familiar Content Service

- electronic mail service

![Diagram of electronic mail service]

**Figure 28.1 Outline of Internet electronic mail.**

MTA to MTA

- mail transfer agent (MTA) moves messages to their destination
  - from a queue to some (recipient) MTA
  - to a user agent from some (sender) MTA
- MTAs speak simple mail transfer protocol (SMTP) to each other
  - originally in RFC821
  - currently RFC2821
- SMTP is a TCP service
  - runs on port 25
MTA's view: sending mail

220 something.org Sendmail 8.12 ready
HELO mynode.subdomain.com
250 OK
MAIL From:<scarter@mynode.subdomain.com>
250 <scarter@mynode.subdomain.com> OK
RCPT To: <joneil@nodex.sgc.mil>
250 <joneil@nodex.sgc.mil> OK
DATA
354 Enter mail, end with \r\n\n  mail message inserted here
.
250 OK mail accepted
QUIT
221 something.org closing connection

Server Response Codes

- first defined for SMTP in RFC822
- now widely used in other services
- based on 3-digit xyz values:

<table>
<thead>
<tr>
<th>xyz</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1yz</td>
<td>positive preliminary reply</td>
</tr>
<tr>
<td>2yz</td>
<td>positive completion reply</td>
</tr>
<tr>
<td>3yz</td>
<td>positive intermediate reply</td>
</tr>
<tr>
<td>4yz</td>
<td>transient negative completion reply</td>
</tr>
<tr>
<td>5yz</td>
<td>permanent completion reply</td>
</tr>
</tbody>
</table>
Server Response Codes

<table>
<thead>
<tr>
<th>x0z</th>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1z</td>
<td>informational</td>
</tr>
<tr>
<td>x2z</td>
<td>connections</td>
</tr>
<tr>
<td>x3z</td>
<td>unspecified</td>
</tr>
<tr>
<td>x4z</td>
<td>unspecified</td>
</tr>
<tr>
<td>x5z</td>
<td>Mail system</td>
</tr>
</tbody>
</table>

- and z is context-specific code providing more detail

Server Commands

1. HELO, EHLO:
   - identify the SMTP client to the SMTP server
   - client sends its fully-qualified domain name
   - HELO used by older clients
   - all servers support HELO
   - newer ones support extended services, hence EHLO
Server Commands

1. HELO, EHLO:
2. MAIL
   - start mail transaction where mail is sent to server
3. RCPT
   - identifies a single recipient of the mail message
   - use multiple instances of RCPT for multiple recipients
Server Commands

1. HELO, EHLO:
2. MAIL
3. RCPT
4. DATA
   - server replies with 354 line then copies everything sent to it by client up to the ‘.’ line
   - data should consist only of 7–bit ASCII characters
     - and avoid ctrl chars other than SP, HT, CR, and LF

Server Commands

1. HELO, EHLO:
2. MAIL
3. RCPT
4. DATA
5. RSET
   - aborts current mail operation in progress
   - any data received as part of this operation is deleted from server
   - has no effect if appears immediately after EHLO
Server Commands

1. HELO, EHLO:
2. MAIL
3. RCPT
4. DATA
5. RSET
6. VRFY
   - asks receiver to confirm that argument identifies a user or mailbox

7. EXPN
   - asks receiver to confirm that argument identifies a mailing list and, if so, return list membership
Server Commands

1. HELO, EHLO:
2. MAIL
3. RCPT
4. DATA
5. RSET
6. VRFY
7. EXPN
8. HELP
   - causes server to send helpful information to client

Server Commands

1. HELO, EHLO:
2. MAIL
3. RCPT
4. DATA
5. RSET
6. VRFY
7. EXPN
8. HELP
9. NOOP
   - has no effect on server other than to cause it to respond with OK
Server Commands

1. HELO, EHLO:
2. MAIL
3. RCPT
4. DATA
5. RSET
6. VRFY
7. EXPN
8. HELP
9. NOOP
10. QUIT

- server must send OK and close connection

email messages

- simple structure —
  - envelope: used by MTAs for delivery
    - consists of the 2 SMTP commands MAIL and RCPT
  - header: contains non-message information like addressee, sender, date, etc.
    - has a standard syntax
  - body: contains actual content of message
    - can be ‘anything’
Mail Headers

- used by user agents
- simple syntax: attribute:value
  - attributes may contain ASCII chars with codes 0x21 to 0xFE except 0x3A
  - values may contain any ASCII chars except CR and LF
  - e.g., "From: user@somplace.com"
  - some start with X- are user-defined
  - e.g., X-Charset, X-Mailer

Mail Headers

- **From**: mailbox-list
  - specifies the author(s) of the message, that is, the mailbox(es) of the person(s) or system(s) responsible for the writing of the message
  - must appear
- **Sender**: mailbox
  - specifies the mailbox of the agent responsible for the actual transmission of the message
  - can appear, but must appear if mailbox-list in From: line has ≥ 1 mbox
- **Reply-To**: address-list
  - list to which replies to a message should be sent
  - is optional
Mail Headers

- **To**: address-list
  - contains the address(es) of the primary recipient(s) of the message
- **CC**: address-list
  - contains the addresses of others who are to receive the message
- **BCC**: address-list
  - contains addresses of recipients of the message whose addresses are not to be revealed to other recipients of the message

---

Mail Headers

- **Message-ID**: <ident>
  - not intended to be human-readable
  - e.g., 200403262028.12qKs2C00185620@netlab.gmu.edu
  - provides unique identifier referring to particular version of particular message; uniqueness guaranteed by host generating it
- **In-Reply-To**: <ident>
  - lists message-id of current msg (to which it is a reply)
- **References**: <ident>
  - provides id's appearing in current msg's references field
Mail Headers

- informational fields, human readable:
  - Subject:
  - Comments:
  - Keywords:

Mail Headers

- user defined fields:
  - X-Charset: identify character set used to represent message
  - X-Mailer: user agent used to send message
  - X-Sender: duplicate of From:
email messages

- originally were only text
  - using 7-bit ASCII
- protocols designed around text-only original version
  - still support 7-bit ASCII
  - some may support 8-bit character data (see RFC1652)
  - described in RFC822 (still often cited)
  - current version is RFC2822
- lines must be $\leq 1000$ bytes long
- but now want more than just text...

Different Content Types

- need mechanism to support:
  - different kinds of content to appear in a msg
  - multiple different kinds of content within one msg
**Different Content Types**

- need mechanism to support:
  - different kinds of content to appear in a msg
  - multiple different kinds of content within one msg
- use Multipurpose Internet Mail Extensions (MIME)
  RFC1521 (now: RFCs: 2045–2049)
  - provides way to encode binary data using only printable ASCII characters
  - inflates size of data

**MIME**

- used in email, adds lines to headers:
- for standard text messages:
  - MIME-Version: 1.0
  - Content-type: text/plain;
    charset=US-ASCII
- text of message follows
- message ends at end-of-text
MIME

- used in email, adds lines to headers:

- for multi-part messages:

  Content-Type: multipart/mixed;
  boundary="--------------090405080304070600000708"
  
  Content-Type: text/plain; charset=ISO-8859-1;
  format=flowed
  Content-Transfer-Encoding: 7bit

  First part of message (text)
  
  Content-Type: image/jpeg;
  name="IMG_0980.jpg"
  Content-Transfer-Encoding: base64
  Content-Disposition: inline;
  filename="IMG_0980.jpg"

  Second part of message (jpeg image)

  

Non-ASCII in Header

- headers may also contain non-ASCII chars

- introduced using syntax:

```plaintext
=? charset ? encoding ? encoded-text ?=
```

start delimiter

identify character set:

- `e.g., ISO-8859-1`

end delimiter

encoding scheme:

- `Q` for quoted printable
- `B` for base-64
Transfer Encodings

ways to represent non-ASCII data as ASCII:

2. quoted-printable:
   - ASCII chars with codes 0x21 to 0x7E (except 0x3D) appears “as is”
   - all line breaks expressly appear as \r\n
   - space appears as “_” unless at end of line “==20”
   - lines longer than 76 bytes have inserted line breaks
   - everything else represented as sequence =xx
     - e.g., ‘é’ appears as “=E9”

Quoted Printable Example

From: =?ISO-8859-1?q?Patrik_F=AEltstrom=F6m?= <paf@nada.kth.se>
translates into:

Patrik Fältström

eample taken from RFC1522
Transfer Encodings

ways to represent non-ASCII data as ASCII:

2. base64 encoding
   - replace every 6 bits of binary data with a single ASCII character from special charset table
   - table contains:
     - A..Z
     - a..z
     - 0..9
     - + / =
   - send string so generated

Base64 Example

From: =?US-ASCII?q?Keith_Moore?= <moore@cs.utk.edu>
To: =?ISO 8859-1?q?Keld_J=88rn_Simonsen?= <keld@dkuug.dk>

which a user agent renders as:

From: Keith Moore <moore@cs.utk.edu>
To: Keld Jorn Simonsen <keld@dkuug.dk>
CC: André Picard <PIRARD@vlm1.ulg.ac.be>
Subject: If you can read this you understand the example.