IT441 Network Servers & Infrastructures

This Time

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

TCP Segment

TCP segment format [RFC 793]

DATA...

TCP Segment

source port number [16]

dest port number [16]

sequence number [32]

(outbound): number of first byte of data in this segment - used by receiver to re-sequence segment data received out-of-order
TCP Segment

- SRC PORT
- DEST PORT
- SEQUENCE NUMBER
- ACK NUMBER
- URG
- RSV
- SYN
- FIN
- PSH
- ADR
- ECE
- CUR
- FOC
- DATA
- urgent pointer [16]
- offset beyond current seq num where urgent data ends

TCP Handshaking

- TCP uses "3-way handshake" both to set-up and shutdown a virtual connection

- active open
- SYN 32173Vx0024
- FIN, RST, URG
- FIN
- close

- passive open
- SYN 27145/131024
- ACK 27214/131024
- RST

- FIN, RST, URG
- FIN
- close

- but what if only one side is done?

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

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 (4:20)
- 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

- OP (8): 1-request
  2-reply
- HTYPE (8): network hardware type
  1=Ethernet
- HLEN (8): hardware address length (bytes)
  6 for Ethernet
- HOPS (8): number of servers that forwarded this request
  client sets to 0
BOOTP: how delivered?

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

![Diagram of IP HDR, UDP HDR, 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

- 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 cafe

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

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

- OP
- HTYPE
- HLEN
- HOPS
- XID
- Transaction ID
- SECONDS
- ENDS
- CLIENT IP ADDRS
- SERVER IP ADDRS
- ROUTER IP ADDRS
- TELNET IP ADDRS
- CLIENT HARDWARE ADDRESS
- SERVER HOSTNAME
- BOOT FILE NAME
- OPTIONS

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

DHCP: Addresses, not Names

- node X gets IP address / from DHCP server
- node X’s lease expires without renewal
- node Y gets IP address / from DHCP server
- traffic to somename.com goes to Y not X

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
- 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:
**Domain Names**

- but can have other info in the 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
    - 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

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 csf1.gmu.edu replies with:
    
    ; ANSWER SECTION:
    csf1.gmu.edu. 30119 IN A 129.174.1.13
    ; AUTHORITY SECTION:
    gmu.edu. 85749 IN NS portal.gmu.edu.
    gmu.edu. 85749 IN NS sarpon.gmu.edu.

Resolving Names

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

```
struct hostent {
    char *h_name; /* official host name */
    char **h_aliases; /* alias list */
    int h_addrtype; /* host address type */
    int h_length; /* length of address */
    char **h_addr_list; /* list of addresses */
}
```

Sample DNS Record

- db in nameserver is a text file containing resource records
- syntax: <domain_name><ttl><class><type><value>
  
  example entry: (from Tanenbaum, fig. 7.3)
  
<table>
<thead>
<tr>
<th>Server</th>
<th>Class</th>
<th>Type</th>
<th>TTL</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>filts.cs.uu.nl</td>
<td>IN</td>
<td>A</td>
<td>60400</td>
<td>129.31.231.168</td>
</tr>
<tr>
<td>filts.cs.uu.nl</td>
<td>IN</td>
<td>1</td>
<td>60400</td>
<td>1 filts.cs.uu.nl</td>
</tr>
<tr>
<td>filts.cs.uu.nl</td>
<td>IN</td>
<td>2</td>
<td>60400</td>
<td>2 sephyr.cs.uu.nl</td>
</tr>
<tr>
<td><a href="http://www.cs.uu.nl">www.cs.uu.nl</a></td>
<td>IN</td>
<td>A</td>
<td>60400</td>
<td>star.cs.uu.nl</td>
</tr>
<tr>
<td>ftp.cs.uu.nl</td>
<td>IN</td>
<td>NAME</td>
<td>60400</td>
<td>sephyr.cs.uu.nl</td>
</tr>
</tbody>
</table>

Sample Resource Record

- Class for Internet
- Canonical Name: establish aliases for a host
- Mail/Exchange: if FQDN name to which email goes

DNS Messages

- Identification (16): value assigned by client echoed by server, to aid client's matching requests with answers
DNS Messages

- sent via UDP

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 RFC2821
  - 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: <jonall@nodox.spc.mil> OK
250 <jonall@nodox.spc.mil> OK
DATA
354 Enter mail, end with \r\n.\r\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>Code</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>

- 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

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

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
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
8. HELP
   ▶ causes server to send helpful information to client
9. NOOP
   ▶ has no effect on server other than to cause it to respond with OK
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
    - addresssee, sender, date, etc.
  - 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
      0x7E except 0x0A
    - 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
    - optional

Mail Headers
- Message-ID: <ident>
  - not intended to be human-readable
  - e.g., <2004032220205.270885@netlab.qmu.ac.uk>
  - 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 ≤ 1020 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="--0028831543-0334000708"
    - Content-Type: text/plain; charset=ISO-8859-1
    - Content-Transfer-Encoding: 7bit
    - Content-Type: image/jpeg;
      name="PIC_000.jpg"
    - Content-Transfer-Encoding: base4
    - Content-Disposition: inline;
      filename="PIC_000.jpg"
    - Content-Type: image/jpeg;
      name="PIC_000.jpg"
    - Content-Transfer-Encoding: base64
      boundary="--0028831543-0334000708--"
Non-ASCII in Header

- headers may also contain non-ASCII chars
- Introduced using syntax:

```plaintext
charset encoding encoded-text
```}

Transfer Encodings

- ways to represent non-ASCII data as ASCII:
  - quoted-printable:
    - ASCII chars with codes 0x21 to 0x7E (except 0x30)
      appears “as is”
    - all line breaks explicitly appear as \n
Quoted Printable Example

From: <ISO-8859-1@Patrick_P@kth.se> <paf@kth.se>
translates into:

example taken from RFC1522

Base-64 Example

From: <ISO-ASCIITQ@Keith_Morse@<nore@cs.utk.edu>
To: <mal@cs.utk.edu>
CC: <mal@cs.utk.edu>
Subject: If you can read this you understand the example.

which a user agent renders as:

From: Keith Morse <nore@cs.utk.edu>
To: Feld Jern Simonsen <felid@kog.dk>
CC: André Picard <FIRAND@Phil.uig.ac.be>
Subject: If you can read this you understand the example.