IT441: Network Servers & Infrastructure

CLASS 5 : 03 Oct 2005
13:30 - 16:15

Last Time

- IPv4 options
- ICMP
- bridges & routers
- IPv4 addressing: classful, CIDR
- NAT
- IP: a delivery service
- UDP: a transport service
This Time

- TCP
- boot-time issues
- IP address: how allocated to host
- DNS

Transport Protocol

- need a transport protocol:
  - can identify individual end–point (process)
  - both source process and dest process
  - a.k.a. “end–to–end” protocol
- implemented as a separate layer in IP:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION</td>
<td>5</td>
</tr>
<tr>
<td>TRANSPORT</td>
<td>4</td>
</tr>
<tr>
<td>INTERNET</td>
<td>3</td>
</tr>
<tr>
<td>NETWORK INTERFACE</td>
<td>2</td>
</tr>
<tr>
<td>PHYSICAL</td>
<td>1</td>
</tr>
</tbody>
</table>
**UDP: User Datagram Protocol**

- simple, cheap, fast
- connectionless
- best–effort
- message–oriented
- no control msgs
- \(n:m\) interaction; \(n,m \geq 1\)
- message oriented
- no fragmenting or reassembly

**UDP Ports**

- port numbers identify a service
  - whose implementation is platform specific
  - but that meets protocol definition of the service
- port numbers are assigned:
  - 0 – 1023 are “well-known” port numbers
  - originally listed in RFC 1700
  - now found online at:
    - [http://www.iana.org/assignments/port-numbers](http://www.iana.org/assignments/port-numbers)
- see `/etc/services` on UNIX platforms for list, e.g.,
  - `daytime 13/udp`
A Better Model...

- applications want
  - to focus on the nature of the app
  - not to deal with handling communications reliability problems
- need a transport protocol able to deliver end-to-end reliable transport:
  - no duplicate packets
  - no out of order packets
  - no missing packets
  - no errors in packets

...Hard to Build

- easy to ensure such reliability across single LAN
- hard to ensure across internet with
  - multiple networks of different capabilities
  - routers with different capabilities
  - nodes and routers that can fail at random moments
Transport Control Protocol: TCP

- TCP is IP's other transport protocol
- *uses IP for delivery*, yet provides:
  - connection-oriented service
  - point-to-point interaction (1:1)
  - complete reliability
    - startup
    - data exchange
    - shutdown
  - fdx communication
  - stream data interface

TCP: Connection-Oriented

- *uses virtual connection between src and dest connection*:
  - is set–up by src and dest before data exchanged
  - used for data exchange
  - is shutdown when done
- need for control messages of some kind
Reliability

- TCP provides reliability by:
  - handling re-transmits of data believed not received with adaptive timers
  - using sequence numbers in packets (segments)
  - performing flow control with windows
  - performing congestion control

ACKs, Timers, Re-Transmits

- each received TCP pkt is ACKnowledged
  - so sender knows it has been received
- each TCP pkt sent starts a timer
  - if timer counts down to 0 before ACK received, sender sends same pkt again
- this handles data corruption errors as well
TCP Retransmission Timer in Action

One Time Fits All?

- what should time-out timer be set to?
One Time Fits All?

- what should time-out timer be set to?
  - on Ethernet LAN can be small ms. range
  - on satellite link needs to be hundreds of ms

TCP’s timer

- what should time-out timer be set to?
  - on Ethernet LAN can be small ms. range
  - on satellite link needs to be hundreds of ms
- TCP uses *adaptive* timers based on measured round trip time (RTT)
  - measures time from send to ACK for a segment
  - keeps weighted average
  - uses statistical formula to adjust timeout value to follow conditions and avoid excess re-xmits
Flow Control

- TCP manages flow of data using a window
  - size of memory buffer to hold received data
  - initialized (both sides) during connection set-up
  - window = amount of buffer space currently remaining
  - src/dest use window advertisement to inform each other of buffer availability
  - if receive 0-window, stop sending until receive non-zero window
Congestion Control

- network is congested when traffic load is too high
- congested networks increase delay in segment delivery
- can lead to increase in re-xmits

Congestion Control

- TCP uses packet loss as metric of congestion
  - if congestion, reduce rate of (re)transmission
- to reduce rate:
  - make window artificially smaller
- recovering from congestion:
  - send small segment (< window size)
  - if ACKed without further loss, send two small segments
  - continue exponential ramp up until:
    - loss encountered
    - reached 50% window size, switch to linear
## Congestion Control on Start-up

- how fast to send when first starting?
  - don't have any notions about state of congestion
- use *TCP slow-start*:
  - send 1 segment, wait for ACK
  - send 2 segments, wait for their ACKs
  - send 4 segments...
  - like recovery from congestion on previous page
  - Stallings, fig. 7.8

## TCP Segment

- TCP segment format [RFC 793]
TCP Segment

**Source Port number [16]**

**Destination Port number [16]**

TCP Segment

**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

<table>
<thead>
<tr>
<th>SRC PORT</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SEQUENCE NUMBER

DATA...

Acknowledgement number [32]

(inbound): next expected sequence number, i.e., number of first byte of data in next expected segment;

TCP Segment

<table>
<thead>
<tr>
<th>SRC PORT</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SEQUENCE NUMBER

ACK NUMBER

DATA

Window [16]

(inbound): how much receiver buffer space is left for data starting at indicated ack number
**TCP Segment**

<table>
<thead>
<tr>
<th>SRC PORT</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQUENCE NUMBER</td>
<td></td>
</tr>
<tr>
<td>ACK NUMBER</td>
<td></td>
</tr>
<tr>
<td>WINDOW</td>
<td></td>
</tr>
<tr>
<td>DATA...</td>
<td></td>
</tr>
<tr>
<td>header length [4]</td>
<td></td>
</tr>
</tbody>
</table>

**TCP Segment**

<table>
<thead>
<tr>
<th>SRC PORT</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQUENCE NUMBER</td>
<td></td>
</tr>
<tr>
<td>ACK NUMBER</td>
<td></td>
</tr>
<tr>
<td>HLEN</td>
<td></td>
</tr>
<tr>
<td>WINDOW</td>
<td></td>
</tr>
<tr>
<td>DATA...</td>
<td></td>
</tr>
<tr>
<td>RFU [6]</td>
<td></td>
</tr>
</tbody>
</table>
TCP Segment

**ACK NUM is valid**

**PSH: rcvr push to app as soon as possible**

**SYN: sync seq nums to start connection**

**FIN: sender has finished**

**URG pointer is valid**

**RST: reset connection**

---

**TCP Segment**

<table>
<thead>
<tr>
<th>SRC PORT</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE PORT</td>
<td>DEST PORT</td>
</tr>
<tr>
<td>USE PORT</td>
<td>DEST PORT</td>
</tr>
<tr>
<td>USE PORT</td>
<td>DEST PORT</td>
</tr>
</tbody>
</table>

**SEQUENCE NUMBER**

**ACK NUMBER**

<table>
<thead>
<tr>
<th>HLEN</th>
<th>RFU</th>
<th>CODE BITS</th>
<th>WINDOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE PORT</td>
<td>USE PORT</td>
<td>USE PORT</td>
<td>USE PORT</td>
</tr>
</tbody>
</table>

**DATA...**

---

**Checksum [16]**

*computed on all of TCP header and data payload*
**TCP Segment**

<table>
<thead>
<tr>
<th>SRC PORT</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENCE NUMBER</td>
<td></td>
</tr>
<tr>
<td>ACK NUMBER</td>
<td></td>
</tr>
<tr>
<td>HLEN</td>
<td>RFU</td>
</tr>
<tr>
<td>CHECKSUM</td>
<td></td>
</tr>
<tr>
<td>DATA...</td>
<td></td>
</tr>
</tbody>
</table>

- **urgent pointer [16]**: offset beyond current seq num where urgent data ends

**TCP Segment**

<table>
<thead>
<tr>
<th>SRC PORT</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENCE NUMBER</td>
<td></td>
</tr>
<tr>
<td>ACK NUMBER</td>
<td></td>
</tr>
<tr>
<td>HLEN</td>
<td>RFU</td>
</tr>
<tr>
<td>CHECKSUM</td>
<td></td>
</tr>
<tr>
<td>DATA...</td>
<td></td>
</tr>
</tbody>
</table>

- **data (if any)**
- **options (if any)**
TCP Handshaking

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

```
active open  
SYN 522173/0/1024
SYN 82441/0/1024
ACK 522174//1024
ACK 82442
passive open
```

Duplicate Control

- duplicates are identified by sequence number
- may arrive:
  - within a message, before connection closes
  - outside of original message, after connection closed
- may be:
  - duplicate of sender's data
  - duplicate ACK from receiving end
Duplicate Detection Failure

Stallings, Fig. 6.5

TCP Handshaking

- shutdown is similar:
  
  | FIN_WAIT_1 | FIN | CLOSE_WAIT |
  | FIN_WAIT_2 | ACK of FIN | LAST_ACK |
  | TIME_WAIT | FIN | CLOSED |
  | ACK of FIN |

- 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

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 pov.
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
BOOTP Messages

<table>
<thead>
<tr>
<th>OP</th>
<th>HTYPE</th>
<th>HLEN</th>
<th>HOP CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Transaction Identifier [32]:
client sets to x; matches response to request

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

unused [16]
# BOOTP Messages

<table>
<thead>
<tr>
<th>OP</th>
<th>HTYPE</th>
<th>HLEN</th>
<th>HOP CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>TRANSACTION NUMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SECONDS ELAPSED</td>
</tr>
</tbody>
</table>

**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 address or 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></td>
<td></td>
<td></td>
<td>TRANSACTION NUMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SECONDS ELAPSED</td>
</tr>
</tbody>
</table>

**router IP address [32]:** set by server to default router’s IP address

**your IP address [32]:** set by server to IP address of requesting client
### 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>server host name [512]</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>ROUTER IP ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLIENT HARDWARE ADDRESS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**server IP address [32]:**

---

### 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]:**

---

*IT441 Network Servers & Infrastructure* 03 Oct 2005 © C. M. Snow 2005 05.49
**BOOTP: how delivered?**

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

![Diagram](https://via.placeholder.com/150)

- 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 addrs
  - lower-level wants to map target IP address to local hardware address
  - looks in ARP table but doesn't find target IP addrs

  - 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
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></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
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 otherwise:
  - node X gets IP address \( j \) from DHCP server
  - name server entry binds \( j \).somename.com
  - node X’s lease expires without renewal
  - node Y gets IP address \( j \) from DHCP server
  - node X gets IP address \( k \) from DHCP server

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

  more local
  Top Level Domain (TLD)

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

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

- 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

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

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 osfl.gmu.edu replies with:

```bash
;; ANSWER SECTION:
osfl.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.
```
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 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 Tanenbaum, 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 *INFO* 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

Host Info: CPU OS

Address: IP address

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
DNS Messages

- **OP Code [4]:** 0 ⇒ normal query
- **TC[1]:** 0 if not, 1 if is 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

**Identification**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Identification number</td>
</tr>
<tr>
<td>flags</td>
<td>Flags</td>
</tr>
<tr>
<td>QR</td>
<td>Query/Response</td>
</tr>
<tr>
<td>AA</td>
<td>Authoritative Answer</td>
</tr>
<tr>
<td>RD</td>
<td>Recursion Desired</td>
</tr>
<tr>
<td>RA</td>
<td>Recursion Available</td>
</tr>
</tbody>
</table>

**Flags**

- **MBZ:** Masked Bit Zone
- **RCODE:** Return Code [4]: 0 if no error, 3 if name error

**Questions** (var length)

- Question Domain Name
- Question Type
- Question Class
- Question TTL

**Answers** (var length)

- Answer Type
- Answer Class
- Answer TTL
- Answer Data

**Authority** (var length)

- Authority Domain Name
- Authority Class

**Additional** (var length)

- Additional Domain Name
- Additional Class
- Additional TTL
- Additional Data

**Number of Questions [16]**

**Number of Answer RRs [16]**

**Number of Authority RRs [16]**

**Number of Additional RRs [16]**
DNS Messages

- sent via UDP