Computer Networks and Applications

COMP 3331/COMP 9331

Week 3

Application Layer (Email, DNS, Socket Programming)

Reading Guide: Chapter 2, Sections 2.4, 2.5, 2.7
Announcements

- Lab for Week 3 – Socket Programming
- Sample Problem Set
  - Did anyone attempt the first problem set? Please do. Solutions soon.
  - Set for application layer is now available
- Remember **mid-semester exam** in **Week 6**
- **First Programming Assignment** to be released this week
- Please participate on the forums
2. Application Layer: outline

2.1 principles of network applications
   ▪ app architectures
   ▪ app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
   ▪ SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP
Electronic mail

**Three major components:**

- user agents
- mail servers
- simple mail transfer protocol: SMTP

**User Agent**

- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server
Electronic mail: mail servers

mail servers:
- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - “server”: receiving mail server
Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII
Scenario: Alice sends message to Bob

1) Alice uses UA to compose message “to” 
   bob@someschool.edu
2) Alice’s UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob’s mail server
4) SMTP client sends Alice’s message over the TCP connection
5) Bob’s mail server places the message in Bob’s mailbox
6) Bob invokes his user agent to read message
Sample SMTP interaction

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
Try SMTP interaction for yourself:

- telnet servername 25
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

**Implication: one could send forged emailed**

Note: Many SMTP servers will not allow the above interaction without authentication. E.g: try the above with mail.unsw.edu.au @ port 587
How to tell a fake email?

Examine Long Headers or Raw Source

Further reading: http://www.millersmiles.co.uk/identitytheft/spoofemail-060603.htm
Phishing

- Spear phishing
  - Phishing attempts directed at specific individuals or companies
  - Attackers may gather personal information (social engineering) about their targets to increase their probability of success
  - Most popular and accounts for over 90% of attacks

- Clone phishing
  - A type of phishing attack whereby a legitimate, and previously delivered email containing an attachment or link has had its content and recipient address(es) taken and used to create an almost identical or cloned email.
  - The attachment or link within the email is replaced with a malicious version and then sent from an email address spoofed to appear to come from the original sender.
Securing E-mail

- STARTTLS: upgrade a plain text connection to TLS/SSL instead of using a separate port for encrypted communication
  - Can be used for SMTP/IMAP/POP

- PGP (later in the course)
SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF . CRLF to determine end of message

comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg
Mail message format

SMTP: protocol for exchanging email msgs
RFC 822: standard for text message format:
- header lines, e.g.,
  - To:
  - From:
  - Subject:
  - different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the “message”
  - ASCII characters only
Quiz: E-mail attachments?

- IF SMTP only allows 7-bit ASCII, how do we send pictures/videos/files via email?

  A: We use a different protocol instead of SMTP

  B: We encode these objects as 7-bit ASCII

  C: We’re really sending links to the objects, rather than the objects themselves

  D: We don’t !! You have been lied to !!
Mail access protocols

- **SMTP**: delivery/storage to receiver’s server
- **mail access protocol**: retrieval from server
  - **POP**: Post Office Protocol [RFC 1939]: authorization, download
  - **IMAP**: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
  - **HTTP(S)**: Gmail, Yahoo! Mail, etc.
**POP3 protocol**

**authorization phase**
- **client commands:**
  - `user`: declare username
  - `pass`: password
- **server responses**
  - `+OK`
  - `-ERR`

**transaction phase, client:**
- `list`: list message numbers
- `retr`: retrieve message by number
- `dele`: delete
- `quit`

---

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```
more about POP3

- previous example uses POP3 “download and delete” mode
  - Bob cannot re-read e-mail if he changes client
- POP3 “download-and-keep”: copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name
Quiz: HTTP vs SMTP

Which of the following is not true?

A. HTTP is pull-based, SMTP is push-based

B. HTTP uses a separate header for each object, SMTP uses a multipart message format

C. SMTP uses persistent connections

D. HTTP uses client-server communication but SMTP does not
2. Application Layer: outline

2.1 principles of network applications
  - app architectures
  - app requirements
2.2 Web and HTTP
2.3 FTP
2.4 electronic mail
  - SMTP, POP3, IMAP
2.5 DNS

2.6 P2P applications
2.7 socket programming with UDP and TCP
Note: 6th Edition uses Python. Lecture notes show examples of both Java and Python
Socket programming

**goal:** learn how to build client/server applications that communicate using sockets

**socket:** door between application process and end-end-transport protocol
Socket programming

Two socket types for two transport services:
- **UDP**: unreliable datagram
- **TCP**: reliable, byte stream-oriented

Application Example:
1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.
Socket programming with UDP

UDP: no “connection” between client & server
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- rcvr extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:
- UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server
**Client/server socket interaction: UDP (Java)**

**Server (running on **hostid**)**

- Create socket, port=x, for incoming request: `serverSocket = DatagramSocket()`
- Read request from `serverSocket`
- Write reply to `serverSocket` specifying client host address, port number

**Client**

- Create socket, `clientSocket = DatagramSocket()`
- Create, address (hostid, port=x), send datagram request using `clientSocket`
- Read reply from `clientSocket`
- Close `clientSocket`
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
    }
}
Example: Java client (UDP), cont.

```java
DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
    new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);

clientSocket.close();
```

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

Self Study

26
Example: Java server (UDP)

```java
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        while(true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            System.out.println("Packet received: "+receivePacket.getData().toString());
        }
    }
}
```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length, IPAddress, port);

serverSocket.send(sendPacket);

End of while loop, loop back and wait for another datagram
Client/server socket interaction: UDP (Python)

**server (running on serverIP)**

create socket, port=x:
serverSocket = socket(AF_INET,SOCK_DGRAM)

read datagram from serverSocket
write reply to serverSocket specifying client address, port number

**client**

create socket:
clientSocket = socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and port=x; send datagram via clientSocket

read datagram from clientSocket
close clientSocket
Example app: UDP client

Python UDPClient

from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
Example app: UDP server

Python UDPServer

from socket import *
serverPort = 12000

serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))

print "The server is ready to receive"

while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)

create UDP socket
bind socket to local port number 12000
loop forever
Read from UDP socket into message, getting client’s address (client IP and port)
send upper case string back to this client

Self Study
Socket programming with TCP

- **client must contact server**
  - server process must first be running
  - server must have created socket (door) that welcomes client’s contact

- **client contacts server by:**
  - Creating TCP socket, specifying IP address, port number of server process
  - *when client creates socket:* client TCP establishes connection to server TCP

- **when contacted by client,** server TCP creates new socket for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

**application viewpoint:**
TCP provides reliable, in-order byte-stream transfer (“pipe”) between client and server
TCP Sockets

Client process

Client socket

Three-way handshake

bytes

Server process

Welcoming socket

Connection socket
Client/server socket interaction: TCP (in Java)

Server (running on hostid)

- create socket, port=x, for incoming request:
  - welcomeSocket = ServerSocket()
  - wait for incoming connection request
    - connectionSocket = welcomeSocket.accept()
  - read request from connectionSocket
  - write reply to connectionSocket
  - close connectionSocket

Client

- create socket, connect to hostid, port=x
  - clientSocket = Socket()
  - send request using clientSocket
  - read reply from clientSocket
  - close clientSocket

TCP connection setup
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
    }
Example: Java client (TCP), cont.

BufferedReader inFromServer =
    new BufferedReader(new
    InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();
import java.io.*;
import java.net.*;

class TCPServer {

    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;

        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {
            Socket connectionSocket = welcomeSocket.accept();

            BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
        }
    }
}
Example: Java server (TCP), cont

Create output stream, attached to socket

Read in line from socket

Write out line to socket

DataOutputStream outToClient =
new DataOutputStream(connectionSocket.getOutputStream());

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';

outToClient.writeBytes(capitalizedSentence);

} { End of while loop, loop back and wait for another client connection

Self Study
Example app: TCP client

Python TCPClient

from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence: ')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
**Example app: TCP server**

**Python TCPServer**

```python
tfom socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
connectionSocket.close()
```

- create TCP welcoming socket
- server begins listening for incoming TCP requests
- loop forever
- server waits on accept() for incoming requests, new socket created on return
- read bytes from socket (but not address as in UDP)
- close connection to this client (but not welcoming socket)
2. Application Layer: outline

2.1 principles of network applications
  ▪ app architectures
  ▪ app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
  ▪ SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

A nice overview: https://webhostinggeeks.com/guides/dns/
DNS: domain name system

**people:** many identifiers:
- TFN, name, passport #

**Internet hosts, routers:**
- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., www.yahoo.com - used by humans

**Q:** how to map between IP address and name, and vice versa?

**Domain Name System:**
- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol*: hosts, name servers communicate to resolve names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network’s “edge”
DNS: History

- Initially all host-address mappings were in a hosts.txt file (in /etc/hosts):
  - Maintained by the Stanford Research Institute (SRI)
  - Changes were submitted to SRI by email
  - New versions of hosts.txt periodically FTP’d from SRI
  - An administrator could pick names at their discretion

- As the Internet grew this system broke down:
  - SRI couldn’t handle the load; names were not unique; hosts had inaccurate copies of hosts.txt

- The Domain Name System (DNS) was invented to fix this

http://www.wired.com/2012/10/joe-postel/
Example use of DNS

- You type www.cse.unsw.edu.au in the URL window of your web browser
- Your browser must establish a TCP connection with the CSE web server
- To do this, your browser needs to know the IP address of the CSE web server
- How does it obtain the IP address??
  - The browser passes on the hostname to the client side of the DNS application running on your machine (gethostbyname() function in UNIX)
  - The DNS client sends out a query for mapping the hostname to an IP address into the DNS hierarchy black box over UDP (destination port number: 53)
  - The DNS client receives a reply with the IP address for www.cse.unsw.edu.au
  - The browser can now initiate a TCP connection with the HTTP server process located at that IP address
DNS: services, structure

**DNS services**
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name
  - Content Distribution Networks: use IP address of requesting host to find best suitable server
    - Example: closest, least-loaded, etc

**why not centralize DNS?**
- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: *doesn’t scale!*
Goals

- No naming conflicts (uniqueness)
- Scalable
  - many names
  - (secondary) frequent updates
- Distributed, autonomous administration
  - Ability to update my own (machines’) names
  - Don’t have to track everybody’s updates
- Highly available
- Lookups should be fast
Key idea: Hierarchy

Three intertwined hierarchies

- Hierarchical namespace
  - As opposed to original flat namespace

- Hierarchically administered
  - As opposed to centralised

- (Distributed) hierarchy of servers
  - As opposed to centralised storage
Hierarchical Namespace

- “Top Level Domains” are at the top
- Domains are sub-trees
  - E.g: .edu, berkeley.edu, eecs.berkeley.edu
- Name is leaf-to-root path
  - instr.eecs.berkeley.edu
- Depth of tree is arbitrary (limit 128)
- Name collisions trivially avoided
  - each domain is responsible
A zone corresponds to an administrative authority that is responsible for that portion of the hierarchy.

- E.g., UCB controls names: *.berkeley.edu and *.sims.berkeley.edu

- E.g., EECS controls names: *.eecs.berkeley.edu
Server Hierarchy

- Top of hierarchy: Root servers
  - Location hardwired into other servers

- Next Level: Top-level domain (TLD) servers
  - .com, .edu, etc.
  - Managed professionally

- Bottom Level: Authoritative DNS servers
  - Actually store the name-to-address mapping
  - Maintained by the corresponding administrative authority
Server Hierarchy

- Each server stores a (small!) subset of the total DNS database
- An authoritative DNS server stores “resource records” for all DNS names in the domain that it has authority for
- Each server needs to know other servers that are responsible for the other portions of the hierarchy
  - Every server knows the root
  - Root server knows about all top-level domains
DNS Root

- Located in Virginia, USA
- How do we make the root scale?

Verisign, Dulles, VA
DNS Root Servers

- 13 root servers (labeled A-M; see http://www.root-servers.org/)

- A Verisign, Dulles, VA
- C Cogent, Herndon, VA
- D U Maryland College Park, MD
- G US DoD Vienna, VA
- H ARL Aberdeen, MD
- J Verisign
- K RIPE London
- I Autonomica, Stockholm
- M WIDE Tokyo
- B USC-ISI Marina del Rey, CA
- L ICANN Los Angeles, CA
- E NASA Mt View, CA
- F Internet Software Consortium, Palo Alto, CA
DNS Root Servers

- 13 root servers (labeled A-M; see http://www.root-servers.org/)
- Replicated via any-casting

A Verisign, Dulles, VA
C Cogent, Herndon, VA (also Los Angeles, NY, Chicago)
D U Maryland College Park, MD
G US DoD Vienna, VA
H ARL Aberdeen, MD
J Verisign (21 locations)
K RIPE London (plus 16 other locations)
I Autonomica, Stockholm (plus 29 other locations)
E NASA Mt View, CA
F Internet Software
  Consortium,
  Palo Alto, CA
  (and 37 other locations)
B USC-ISI Marina del Rey, CA
L ICANN Los Angeles, CA
M WIDE Tokyo
  plus Seoul, Paris, San Francisco

Root Server health: https://www.ultratools.com/tools/dnsRootServerSpeed
**Anycast in a nutshell**

- Routing finds shortest paths to destination

- If several locations are given the same address, then the network will deliver the packet to the closest location with that address

- This is called “anycast”
  - Very robust
  - Requires no modification to routing algorithms
TLD, authoritative servers

**top-level domain (TLD) servers:**
- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

**authoritative DNS servers:**
- organization’s own DNS server(s), providing authoritative hostname to IP mappings for organization’s named hosts
- can be maintained by organization or service provider
**Local DNS name server**

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called “default name server”
- Hosts configured with local DNS server address (e.g., `/etc/resolv.conf`) or learn server via a host configuration protocol (e.g., DHCP)
- Client application
  - Obtain DNS name (e.g., from URL)
  - Do `gethostbyname()` to trigger DNS request to its local DNS server
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy
DNS name resolution example

- host at wagner.cse.unsw.edu.au wants IP address for gaia.cs.umass.edu

**iterated query:**
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”
**DNS name resolution example**

**recursive query:**
- puts burden of name resolution on contacted name server

Quiz: Which one would you use? Why?

Iterated queries:
- Requesting host: wagner.cse.unsw.edu.au
- Local DNS server: cse.unsw.edu.au
- TLD DNS server
- Authoritative DNS server: dns.cs.umass.edu

Recursive queries:
- Requesting host: wagner.cse.unsw.edu.au
- Local DNS server: cse.unsw.edu.au
- Root DNS server
- TLD DNS server
- Authoritative DNS server: dns.cs.umass.edu
- Requesting host: gaia.cs.umass.edu

Iterated queries vs Recursive queries:
- Iterated queries follow the local DNS server to the authoritative DNS server.
- Recursive queries involve queries to the root DNS server as well.
DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited

- Subsequent requests need not burden DNS
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
Quiz: DNS Record TTL

- The TTL value should be:

  A. Short, to make sure that changes are accurately reflected

  B. Long to avoid re-queries of higher level DNS servers

  C. Something else
DNS records

DNS: distributed db storing resource records (RR)

RR format: \((\text{name}, \text{value}, \text{type}, \text{ttl})\)

**type=A**
- name is hostname
- value is IP address

**type=NS**
- name is domain (e.g., foo.com)
- value is hostname of authoritative name server for this domain

**type=CNAME**
- name is alias name for some “canonical” (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- value is canonical name

**type=MX**
- value is name of mailserver associated with name
query and reply messages, both with same message format

msg header
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative

<table>
<thead>
<tr>
<th></th>
<th>identification</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td># questions</td>
<td># answer RRs</td>
<td></td>
</tr>
<tr>
<td># authority RRs</td>
<td># additional RRs</td>
<td></td>
</tr>
<tr>
<td>questions</td>
<td>answers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>authority</td>
<td></td>
</tr>
<tr>
<td></td>
<td>additional info</td>
<td></td>
</tr>
</tbody>
</table>
**DNS protocol, messages**

<table>
<thead>
<tr>
<th>identification</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td># questions</td>
<td># answer RRs</td>
</tr>
<tr>
<td># authority RRs</td>
<td># additional RRs</td>
</tr>
</tbody>
</table>

- **name, type fields for a query**
- **RRs in response to query**
- **records for authoritative servers**
- **additional “helpful” info that may be used**

2 bytes

2 bytes

questions (variable # of questions)

answers (variable # of RRs)

authority (variable # of RRs)

additional info (variable # of RRs)
bash-3.2$ dig www.cse.unsw.edu.au
>;<<>> DiG 9.6-ESV-R4 <<>> www.cse.unsw.edu.au
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26306
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 4

;; QUESTION SECTION:
;www.cse.unsw.edu.au. IN A

;;ANSWER SECTION:
www.cse.unsw.edu.au. 300 IN CNAME albeniz.orchestra.cse.unsw.edu.au.
albeniz.orchestra.cse.unsw.edu.au. 86400 IN A 129.94.242.51

;;AUTHORITY SECTION:
orchestra.cse.unsw.edu.au. 86400 IN NS beethoven.orchestra.cse.unsw.edu.au.
orchestra.cse.unsw.edu.au. 86400 IN NS maestro.orchestra.cse.unsw.edu.au.

;;ADDITIONAL SECTION:
maestro.orchestra.cse.unsw.edu.au. 86400 IN A 129.94.242.33
beethoven.orchestra.cse.unsw.edu.au. 86400 IN A 129.94.208.3
beethoven.orchestra.cse.unsw.edu.au. 86400 IN A 129.94.242.2
beethoven.orchestra.cse.unsw.edu.au. 86400 IN A 129.94.172.11

;; Query time: 1 msec
;; SERVER: 129.94.242.2#53(129.94.242.2)
;; WHEN: Mon Mar 18 12:36:28 2013
;; MSG SIZE  rcvd: 195
Inserting records into DNS

- example: new startup “Network Utopia”
- register name networkuptopia.com at DNS registrar (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into .com TLD server:
    (networkutopia.com, dns1.networkutopia.com, NS)
    (dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server type A record for www.networkuptopia.com; type MX record for networkutopia.com
- Q: Where do you insert these type A and type MX records?
Reliability

- DNS servers are replicated (primary/secondary)
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- Usually, UDP used for queries
  - Need reliability: must implement this on top of UDP
  - Spec supports TCP too, but not always implemented
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don’t care which server responds
DNS provides Indirection

- Addresses can change underneath
  - Move www.cnn.com to 4.125.91.21
  - Humans/Apps should be unaffected

- Name could map to multiple IP addresses
  - Enables
    - Load-balancing
    - Reducing latency by picking nearby servers

- Multiple names for the same address
  - E.g., many services (mail, www, ftp) on same machine
  - E.g., aliases like www.cnn.com and cnn.com

- But, this flexibility applies only within domain!
DNS Load Balancing

- (Authoritative) DNS server monitors the load of the multiple web servers
- Replies back with the IP address for the least loaded server
- Can also consider the location of the client (e.g., a user from Sydney directed to Australian server)
Reverse DNS

- IP address -> domain name
- Special PTR record type to store reverse DNS entries

Where is reverse DNS used?
- Troubleshooting tools such as traceroute and ping
- “Received” trace header field in SMTP e-mail
- SMTP servers for validating IP addresses of originating servers
- Internet forums tracking users
- System logging or monitoring tools
- Used in load balancing servers/content distribution to determine location of requester
Do you trust your DNS server?

- Censorship

https://wikileaks.org/wiki/Alternative_DNS

- Logging
  - IP address, websites visited, geolocation data and more
  - E.g., Google DNS:
    https://developers.google.com/speed/public-dns/privacy
Attacking DNS

DDoS attacks
- Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server to be bypassed
- Bombard TLD servers
  - Potentially more dangerous

Redirect attacks
- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus replies to DNS server, which caches

Exploit DNS for DDoS
- Send queries with spoofed source address: target IP
- Requires amplification

Want to dig deeper?
http://www.networkworld.com/article/2886283/security0/top-10-dns-attacks-likely-to-infiltrate-your-network.html
DNS Cache Poisoning

- Suppose you are a bad guy and you control the name server for drevil.com. Your name server receives a request to resolve www.drevil.com. and you respond as follows:

```plaintext
;; QUESTION SECTION:
;www.drevil.com. IN A

;; ANSWER SECTION:
www.drevil.com 300 IN A 129.45.212.42

;; AUTHORITY SECTION:
drevil.com 86400 IN NS dns1.drevil.com.
drevil.com 86400 IN NS google.com

;; ADDITIONAL SECTION:
google.com 600 IN A 129.45.212.222
```

Solution: Do not allow DNS servers to cache IP address mappings unless they are from authoritative name servers.

A drevil.com machine, not google.com
Dig deeper?

DNS Cache Poisoning Test
https://www.grc.com/dns/dns.htm

DNSSEC: DNS Security Extensions,
http://www.dnssec.net
Content distribution networks

- **challenge**: how to stream content (e.g. millions of videos) to hundreds of thousands of simultaneous users?

- **option 1**: single, large “mega-server”
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

  ....quite simply: this solution *doesn’t scale*
Content distribution networks

- **challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- **option 2:** store/serve multiple copies of videos at multiple geographically distributed sites (CDN)
  - **enter deep:** push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - **bring home:** smaller number (10’s) of larger clusters in POPs near (but not within) access networks
    - used by Limelight
**CDN: “simple” content access scenario**

Bob (client) requests video http://netcinema.com/6Y7B23V
- video stored in CDN at http://KingCDN.com/NetC6y&B23V

2. Resolve http://netcinema.com/6Y7B23V via Bob’s local DNS
3. netcinema’s authoritative DNS returns URL http://KingCDN.com/NetC6y&B23V
4. Resolve http://KingCDN.com/NetC6y&B23 via KingCDN’s authoritative DNS, which returns IP address of KingCDN server with video
5. Resolve URL http://KingCDN.com/NetC6y&B23V with video
6. Request video from KINGCDN server, streamed via HTTP
Challenge: how does CDN DNS select “good” CDN node to stream to client

- pick CDN node geographically closest to client
- pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping access ISPs, reporting results to CDN DNS)
- IP anycast

Alternative: let client decide - give client a list of several CDN servers

- client pings servers, picks “best”
- Netflix approach
Akamai CDN using DNS redirection

0: cnn.com distributes images to Akamai server
1: An end user types in www.cnn.com in their browser
2: index page contains link to the a73.g.akamai.net
3: local DNS server asks root DNS server to resolve a73.g.akamai.net
4: root DNS replies with IP address of high-level Akamai DNS server, akamai.net
5: local DNS server queries this nameserver to resolve a73.g.akamai.net
6: high-level DNS server replies with IP address of the DNS server for g.akamai.net
7: local DNS queries this nameserver to resolve a73.g.akamai.net
8: low level DNS server takes into consideration the client’s location and replies with the IP address of the closest Akamai web server that hosts the image
9: end user’s machine sends a GET request for the image to this closest Akamai web server
10: Akamai webserver responds with the image
Case study: Netflix

- 37% downstream traffic in North America in 2015
- owns very little infrastructure, uses 3rd party services:
  - own registration, payment servers
  - Amazon (3rd party) cloud services:
    - Netflix uploads studio master to Amazon cloud
    - create multiple version of movie (different encodings) in cloud
    - upload versions from cloud to CDNs
    - Cloud hosts Netflix web pages for user browsing
  - three 3rd party CDNs host/stream Netflix content: Akamai, Limelight, Level-3
Case study: Netflix

1. Bob manages Netflix account

2. Bob browses Netflix video

3. Manifest file returned for requested video

4. DASH streaming

Netflix registration, accounting servers

Amazon cloud

upload copies of multiple versions of video to CDNs

Akamai CDN

Limelight CDN

Level-3 CDN

Self Study