Chapter 2
Application Layer

A note on the use of these ppt slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:
- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we’d like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2007
J.F Kurose and K.W. Ross, All Rights Reserved
Chapter 2: Application layer

- 2.1 Principles of network applications
- 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 TCP
- 2.8 Socket programming with UDP
Chapter 2: Application Layer

Goals:
- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- programming network applications
  - socket API
Some network apps

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video clips
- voice over IP
- real-time video conferencing
- grid computing
Network applications: some terminologies

**Process:** program running within a host.

- within same host, two processes communicate using **inter-process communication** (defined by OS).
- processes in different hosts communicate by exchanging **messages governed by application-layer protocol**
Applications and application-layer protocols

Application: communicating, distributed processes
- e.g., e-mail, Web, P2P file sharing, instant messaging
- running in end systems (hosts)
- exchange messages to implement application

Application-layer protocols
- one “piece” of an app
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)
Creating a network app

write programs that
  ✓ run on (different) end systems
  ✓ communicate over network
  ✓ e.g., web server software communicates with browser software

little software written for devices in network core
  ✓ network core devices do not run user applications
  ✓ applications on end systems allows for rapid app development, propagation
Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P
Client-server paradigm

Typical network app has two pieces: **client** and **server**

**Server:**
- Provides service
- Answers and provides requested service to client
- E.g., Web server sends requested Web page, mail server delivers e-mail

**Client:**
- Initiates contact with server to request for service from server,
- Web: client implemented in browser; e-mail: in mail reader
Client-server architecture

server:
- always-on host
- permanent IP address
- server farms for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses
- example: Gnutella

Highly scalable but difficult to manage
Hybrid of client-server and P2P

Skype
- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging
- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies
Processes communicating

- **Client process**: process that initiates communication
- **Server process**: process that waits to be contacted

- Applications with P2P architectures have client processes & server processes
**Sockets**

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process

- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)
Addressing processes

- to receive messages, process must have **identifier**
- host device has unique 32-bit IP address
- **Q:** does IP address of host on which process runs suffice for identifying the process?
  - **A:** No, many processes can be running on same host

**identifier** includes both IP address and port numbers associated with process on host.

- Example port numbers:
  - HTTP server: 80
  - Mail server: 25

- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80

- more shortly...
App-layer protocol defines

- **Types of messages exchanged**, e.g., request & response messages
- **Syntax of message types**: what fields in messages & how fields are delineated
- **Semantics of the fields**, i.e., meaning of information in fields
- **Rules for when and how** processes send & respond to messages
App-layer protocol defines

Public-domain protocols:
- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:
- e.g., skype
What transport service does an app need?

Data loss
- some apps (e.g., audio) *can tolerate* some loss
- other apps (e.g., file transfer, telnet) require *100% reliable* data transfer

Timing (e2e delay)
- some apps (e.g., Internet telephony, interactive games) require *low* delay to be “effective”

Bandwidth
- some apps (e.g., multimedia) require *minimum* amount of bandwidth to be “effective”
- other apps (“*elastic* apps”) make use of whatever bandwidth they get
## Transport service requirements of common apps

<table>
<thead>
<tr>
<th>Application</th>
<th>Data loss</th>
<th>Bandwidth</th>
<th>Time Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>file transfer</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>e-mail</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>Web documents</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>real-time audio/video</td>
<td>loss-tolerant</td>
<td>audio: 5kbps-1Mbps video:10kbps-5Mbps</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td>stored audio/video</td>
<td>loss-tolerant</td>
<td>same as above</td>
<td>yes, few secs</td>
</tr>
<tr>
<td>interactive games</td>
<td>loss-tolerant</td>
<td>few kbps up</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td>instant messaging</td>
<td>no loss</td>
<td>elastic</td>
<td>yes and no</td>
</tr>
</tbody>
</table>
# Internet transport protocols services

## TCP service:
- **connection-oriented**: setup required between client and server processes
- **reliable transport** between sending and receiving process
- **flow control**: sender won’t overwhelm receiver
- **congestion control**: throttle sender when network overloaded
- **does not provide**: timing, minimum bandwidth guarantees

## UDP service:
- **unreliable data transfer** between sending and receiving process
- **does NOT provide**: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

**Q:** why bother? Why is there a UDP?
### Internet apps: application, transport protocols

<table>
<thead>
<tr>
<th>Application</th>
<th>Application layer protocol</th>
<th>Underlying transport protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-mail</td>
<td>SMTP [RFC 2821]</td>
<td>TCP</td>
</tr>
<tr>
<td>remote terminal access</td>
<td>Telnet [RFC 854]</td>
<td>TCP</td>
</tr>
<tr>
<td>Web</td>
<td>HTTP [RFC 2616]</td>
<td>TCP</td>
</tr>
<tr>
<td>file transfer</td>
<td>FTP [RFC 959]</td>
<td>TCP</td>
</tr>
<tr>
<td>streaming multimedia</td>
<td>proprietary (e.g. RealNetworks)</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>proprietary (e.g., Vonage, Dialpad)</td>
<td>typically UDP</td>
</tr>
</tbody>
</table>
Chapter 2: Application layer

- 2.1 Principles of network applications
  - app architectures
  - app requirements
- 2.2 Web and HTTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
Web and HTTP

First some jargon

- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file, ...
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

  www.someschool.edu/someDept/pic.gif

  host name                  path name
HTTP: HyperText Transfer Protocol

Web’s application layer protocol
- client/server model
  - client: browser that requests, receives, “displays” Web objects
  - server: Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068
**HTTP**

Uses TCP:
- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

**HTTP is “stateless”**
- server maintains NO information about past client requests

**Protocols that maintain “state” are complex!**
- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled
HTTP connections

Nonpersistent HTTP
- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP
- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode
Nonpersistent HTTP

Suppose user enters URL

www.someSchool.edu/someDepartment/home.index

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms response message containing requested object, and sends message into its socket
Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects
Response time modeling

Definition of RTT: time to send a small packet to travel from client to server and back.

Response time:
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+transmit time
Persistent HTTP

Nonpersistent HTTP issues:
- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- but browse HTTP: HyperText Transfer Protocol rs often open parallel TCP connections to fetch referenced objects

Persistent without pipelining:
- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent HTTP
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server are sent over connection

Persistent with pipelining:
- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects
HTTP request message

- two types of HTTP messages: request, response
- HTTP request message:
  - ASCII (human-readable format)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

(extra carriage return, line feed)
HTTP request message: general format

```
Method  URL  Version  
:      :      :     
Header :  : Header  
:      :  :     
```

Entity Body
Uploading “form” input

Post method:
- Web page often includes “form” input
- Input is uploaded to server in entity body

URL method:
- Uses GET method
- Input is uploaded in URL field of request line:

www.somesite.com/animalssearch?monkeys&banana
Method types

HTTP/1.0
- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1
- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field
HTTP response message

- status line (protocol, status code, status phrase):
  - HTTP/1.1 200 OK
  - Connection close
  - Date: Thu, 06 Aug 1998 12:00:15 GMT
  - Server: Apache/1.3.0 (Unix)
  - Last-Modified: Mon, 22 Jun 1998 ...
  - Content-Length: 6821
  - Content-Type: text/html

- header lines

- data, e.g., requested HTML file
  - data data data data data data data ...

2: Application Layer
HTTP response status codes

In first line in server->client response message.
A few sample codes:

200 OK
- request succeeded, requested object later in this message

301 Moved Permanently
- requested object moved, new location specified later in this message (Location:)

400 Bad Request
- request message not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported
Cookies: keeping “state”

Many major Web sites use cookies

Four components:
1) cookie header line in the HTTP response message
2) cookie header line in HTTP request message
3) cookie file kept on user’s host and managed by user’s browser
4) back-end database at Web site

Example:
- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates:
  - a unique ID and
  - an entry in backend database for ID
Cookies: keeping “state” (cont.)

client
Susan

server
Amazon

Cookie file
ebay: 8734

Cookie file
amazon: 1678
ebay: 8734

one week later:

Cookie file
amazon: 1678
ebay: 8734

usual http request msg
usual http response +
Set-cookie: 1678
server creates ID 1678 for user

usual http request msg
cookie: 1678

usual http response msg

usual http request msg
cookie: 1678

usual http response msg

cookie-specific action

backend database

entry in backend database

access

access
Cookies: keeping “state” (cont.)

Client (Susan)

Amazon Server

usual http request msg

usual http response msg

Set-cookie: 1678

usual http request msg

cookie: 1678

usual http response msg

cookie: 1678

usual http response msg

usual http request msg

usual http response msg

one week later:

Amazon server creates ID 1678 for user

create entry

cookie-specific action

access

cookie-specific action

access

backend database
Cookies (continued)

What cookies can bring:
- authorization
- shopping carts
- Recommendations (personalization)
- user session state (Web e-mail)

Cookies and privacy:
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

How to keep “state”:
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state
Web caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client
More about Web caching

- cache acts as both client and server
- typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?
- reduce response time for client request
- reduce traffic on an institution’s access link.
- Internet dense with caches: enables “poor” content providers to effectively deliver content (but so does P2P file sharing)
Caching example (1)

Assumptions
- average object size = 100,000 bits
- avg. request rate from institution’s browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences
- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds
Caching example (2)

Possible solution
- increase bandwidth of access link to, say, 10 Mbps

Consequences
- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay
  = 2 sec + msecs + msecs
- often a costly upgrade
Install cache

- suppose hit rate is .4

Consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total delay = Internet delay + access delay + LAN delay
  = .6*2 sec + .6*.01 secs + milliseconds < 1.3 secs
Conditional GET: client-side caching

- **Goal**: DON’T send object if client has up-to-date cached version

  - **client**: specify DATE of cached copy in HTTP request
    - `If-modified-since: <date>`

  - **server**: response contains NO object if cached copy is up-to-date:
    - HTTP/1.0 304 Not Modified

- **client**: specify DATE of cached copy in HTTP request
  - `If-modified-since: <date>`

- **server**: response contains NO object if cached copy is up-to-date:
  - HTTP/1.0 304 Not Modified

- **server**: response contains object if cached copy is modified:
  - HTTP/1.0 200 OK
  - `<data>`
Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
FTP: the file transfer protocol

- transfer file to/from remote host
- client/server model
  - *client*: side that initiates transfer (either to/from remote)
  - *server*: remote host
- ftp: RFC 959
- ftp server: port 21
FTP: separate control, data connections

- FTP client contacts FTP server at port 21, specifying TCP as transport protocol.
- Client obtains authorization over control connection.
- Client browses remote directory (e.g., list, dir) by sending commands over control connection.
- When server receives a command for a file transfer (e.g., get, put).
- Server opens a second TCP data connection to transfer another file.
- Control connection: “out of band”.
- FTP server maintains “state”: current directory, earlier authentication.
FTP commands, responses

Sample commands:
- sent as ASCII text over control channel
- USER username
- PASS password
- LIST return list of file in current directory
- RETR filename retrieves (gets) file
- STOR filename stores (puts) file onto remote host

Sample return codes
- status code and phrase (as in HTTP)
- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can’t open data connection
- 452 Error writing file
Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
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., Eudora, Outlook, elm, Netscape Messenger
- 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
Scenario: Alice sends message to Bob

1) Alice uses UA to compose message and “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
Client MTA     Server MTA (Mail Transfer Agent)

SYN, destination port=25  
ACK                                SYN, ACK
Authentication [SYN]  
Authentication [RST]  

TCP three-way handshaking (IP of Client MTA)

Different ports (don’t have the function of authentication now)

220: service ready
HELO <SP><domain><CRLF>
Client MTA use it to identify itself
250 <Server MTA domain>

MAIL FROM: reversing path (domain of relaying MTA, sender’s mail account)
RCPT To:
RECP TO: forwarding path (receiver’s mail account)

Continued...
The receiver treats the lines following the “DATA” packet as mail data from the sender.

354: Start mail input; end with .

Client MTA sends the content of the mail object.
Server MTA replies with “ACK” packet

(IP of relaying MTAs)
(IP of original host)

Client MTA sends the end-of-mail command ( . )
250: Requested mail action okay, completed

2 cases:
• Client MTA has other mails to send, go back to “MAIL FROM”
• Client MTA has no mail to send anymore, sends “QUIT” packet
• Server MTA replies with 221 and closes the connection
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)
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 commands!
- body
  - the “message”, ASCII characters only
Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type

From: alice@crepes.fr  
To: bob@hamburger.edu  
Subject: Picture of yummy crepe.  
MIME-Version: 1.0  
Content-Transfer-Encoding: base64  
Content-Type: image/jpeg  

base64 encoded data ......  
...........................  
......base64 encoded data
Mail access protocols

- **SMTP**: delivery/storage to receiver’s server
- **Mail access protocol**: retrieval from server
  - **POP**: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - **IMAP**: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.
Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
DNS: Domain Name System

People: many identifiers:
- SSN, name, passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., gaia.cs.umass.edu - used by humans

Q: map between IP addresses and name?

Domain Name System:
- distributed database implemented in hierarchy of many name servers
- application-layer protocol - to resolve names (address/name translation)
DNS servers

Domain A

Root Name Servers

Domain B

Internet

Domain C

Domain D
DNS: Domain Name System

DNS services
- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?
- single point of failure
- traffic volume
- distant centralized database
- maintenance
doesn’t scale!
Distributed, Hierarchical Database

Client wants IP for www.amazon.com; 1st approx:
- client queries a root server to find com DNS server
- client queries com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com
**DNS: Root name servers**

- contacted by **local name server** that can not resolve name
- **root name server:**
  - contacts **authoritative name server** if name mapping not known
  - gets mapping
  - returns mapping to local name server

13 root name servers worldwide

- a Verisign, Dulles, VA
- c Cogent, Herndon, VA (also LA)
- d U Maryland College Park, MD
- g US DoD Vienna, VA
- h ARL Aberdeen, MD
- j Verisign, (21 locations)
- b USC-ISI Marina del Rey, CA
- f Internet Software C. Palo Alto, CA (and 36 other locations)
- e NASA Mt View, CA
- k RIPE London (also 16 other locations)
- i Autonomica, Stockholm (plus 28 other locations)
- m WIDE Tokyo (also Seoul, Paris, SF)
- l ICANN Los Angeles, CA
- n US DoD Vienna, VA
- o ARL Aberdeen, MD
TLD and Authoritative Servers

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

- **Authoritative DNS servers:**
  - organization’s DNS servers, providing authoritative hostname to IP mappings for organization’s servers (e.g., Web, mail).
  - can be maintained by organization or service provider
Local Name Server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
  - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy
DNS name resolution example

- Host at cis.poly.edu 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
- heavy load?

1. requesting host: cis.poly.edu
2. local DNS server: dns.poly.edu
3. TLD DNS server: dns.cs.umass.edu
4. authoritative DNS server: gaia.cs.umass.edu
5. root DNS server
Domain Name Service (DNS) (cont’d)

- DNS protocol runs over UDP and uses port 53.

- Used by other application-layer protocols - including HTTP, SMTP and FTP for name translation

- Name translation adds an additional delay - sometimes substantial -- to the Internet applications that use DNS
DNS: caching and updating records

- once name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- update/notify mechanisms under design by IETF
  - RFC 2136
DNS records

DNS: distributed db storing resource records (RR)

RR format: (name, value, type, 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
DNS protocol, messages

DNS protocol: 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>identification</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of questions</td>
<td>number of answer RRs</td>
</tr>
<tr>
<td>number of authority RRs</td>
<td>number of additional RRs</td>
</tr>
<tr>
<td>questions (variable number of questions)</td>
<td></td>
</tr>
<tr>
<td>answers (variable number of resource records)</td>
<td></td>
</tr>
<tr>
<td>authority (variable number of resource records)</td>
<td></td>
</tr>
<tr>
<td>additional information (variable number of resource records)</td>
<td></td>
</tr>
</tbody>
</table>
DNS protocol, messages

Name, type fields for a query

RRs in response to query

Records for authoritative servers

Additional "helpful" info that may be used

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

questions (variable number of questions)

answers (variable number of resource records)

authority (variable number of resource records)

additional information (variable number of resource records)
**Host Aliasing**

- A host with a complicated hostname can have *one or more alias names*, e.g.,
  - a hostname `relay1.west-coast.enterprise.com` could have, say, two aliases such as `enterprise.com` and `www.enterprise.com`
  - the hostname `relay1.west-coast.enterprise.com` is said to be *canonical hostname*
Host Aliasing (cont’d)

- Alias hostnames, when present, are typically more mnemonic than a canonical hostname.

- DNS can be invoked by an application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.
Mail Server Aliasing

- It is highly desirable that email addresses be mnemonic, e.g.,
  - Bob has an account with Hotmail - *bob@hotmail.com*
  - the hostname of the Hotmail mail server is more complicated and much less mnemonic than simply hotmail.com (e.g., *relay1.west-coast.hotmail.com*).
Mail Server Aliasing (cont’d)

- DNS can be invoked by a mail application to obtain the *canonical hostname* for a supplied alias hostname as well as the *IP address* of the host.

- DNS permits a company's mail server and Web server to have *identical* (aliased) hostnames, e.g., enterprise.com.
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
- How do people get IP address of your Web site?
Load Distribution

- DNS is used to perform load distribution among **replicated servers**, e.g., web servers.
  - **Busy sites**, such as cnn.com, are replicated over multiple servers, with each server running on a different end system, and having a different IP address.

- A set of IP addresses is associated with one canonical hostname.
Load Distribution (cont’d)

- The DNS database contains *the set of IP addresses for each hostname*.
- When clients make a DNS query for a name mapped to a set of addresses,
  - the server responds with the *entire* set of IP addresses,
  - but *rotates* the ordering of the addresses within each reply.
Load Distribution (cont’d)

- A client *typically* sends its HTTP request message to the **first** IP address listed in the set.
- **DNS rotation** distributes the traffic among all the replicated servers.
- DNS rotation also used for *email* with multiple mail servers having the same alias name.
References

- http://www.dns.net/dnsrd/docs/
  - a nice collection of documents pertaining to DNS
References (cont’d)

- The Internet Software Consortium provides many resources for BIND, a popular public-domain name server for Unix machines
- Paul Albitz and Cricket Liu, “DNS and BIND,” O'Reilly & Associates, 1993
Discussions

- DNS is *not* an application with which a user directly interacts.
- Instead, the DNS provides a core Internet name-to-address translation function for user applications and other Internet software.
- Much of the "complexity" in the Internet architecture is located at the "edges" of the network.
Server Farm and Web Switch
Introduction

- The Internet, in particular the World Wide Web, has experienced explosive growth and continues to extend at an amazing pace.

- Cluster-based server architecture is a successful and cost effective alternate to build a scalable, reliable, and high-performance Internet server system
Server Environment

- Multiple Servers per application clustering
  - Scalability
  - Availability

- Manageability needs
  - Service portability
  - Transparent to users

- No Service Disruption
Introduction (cont’d)

☐ An important issue is “how to dispatch and route incoming requests to the server best suited to respond?”

☐ Issues ignored by Existing Routing Schemes
  - Session Integrity
  - Sophisticated Load Balancing
  - Differentiated Services
  - Content Deployment
Content-aware Request Distribution

Advantages

- Increase performance due to improved hit rates
- Able to partition the server’s database over the different back-end nodes
- Specialized for certain types of requests
Content-aware Request Distribution (cont’d)

- Issue of “Content Inspection”
  - “read” the content of the requests
  - a TCP connection must be established with the client prior to assigning the request to a back-end node.
Web Switch

- To support virtual IP for web service
  - network address translation (VIP -> realIP)
- To support content-based switching
  - URL switching
  - Cookie switching - stateful
- To support server farm
  - load dispatching
  - Health check
HTTP Request Examples

GET /sports/baseball/index.shtml HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.08 [en] (X11; I; FreeBSD 4.1-Release i386)
Host: www.kimo.com.tw
Accept: image/gif, image/x-xbitmap, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,* utf-8
Cookie: SESSIONID=123456789
URL Switching, Content Publishing/Web Hosting

- Server load balancing based on requested content
- Reduce content replication and management overhead
- URL switching is on a per VIP basis
- High performance URL parser

**URL Directory**
- www.domain.com/news real server 1
- www.domain.com/weather real server 1
- www.domain.com/travel real server 2
- www.domain.com/shopping real server 2
Server Load Balancing

- Users connect to a Virtual IP Address but actually are served by multiple Physical Servers

- Local Load share algorithm -
  - Round-Robin
  - Least Load first (session count)
  - Least traffic first (bytes count)
  - Least weighted load (weight + session count)
  - Ping to find the most responsive host

HTTP traffic to Servers 1 & 2
FTP traffic to Servers 2 & 3
Email to Server 4
Virtual IP

- Back-end nodes in the cluster share a common IP address, called VIP.
- Each node has its own unique IP (public or private) and MAC address.
- Client always sees VIP.
Server Farms (Cluster-based Server Architecture)

- Server Farm looks like one Server with one IP Address to the world
- Service portability: Service can be managed independent of Servers
- Mix & Match different Servers: Capacity, O/S
- Maximum Scalability: Each Server is utilized to its potential
Content Delivery Network (CDN)

- Deliver Web-based content from geographically dispersed servers that sit on the edge of various networks.
- Deliver content according to the proximity of the Web surfer.
- Example
  - A Web surfer viewing a Web site on a computer in California most likely will get content delivered from servers on the West Coast;
  - a Boston viewer would get images from a server on the East Coast.
Content distribution networks (CDNs)

- The content providers are the CDN customers.

**Content replication**
- CDN company installs hundreds of CDN servers throughout Internet
  - in lower-tier ISPs, close to users
- CDN replicates its customers’ content in CDN servers. When provider updates content, CDN updates servers
CDN example

Origin server

1. HTTP request for www.foo.com/sports/sports.html

2. DNS query for www.cdn.com

CDNs authoritative DNS server


Nearby CDN server

origin server

- www.foo.com
- distributes HTML
- Replaces:
  - http://www.foo.com/sports/ruth.gif
  - with

CDN company

- cdn.com
- distributes gif files
- uses its authoritative DNS server to route redirect requests
More about CDNs

routing requests

- CDN creates a “map”, indicating distances from leaf ISPs and CDN nodes
- when query arrives at authoritative DNS server:
  - server determines ISP from which query originates
  - uses “map” to determine best CDN server

not just Web pages

- streaming stored audio/video
- streaming real-time audio/video
  - CDN nodes create application-layer overlay network
Chapter 2: Application layer

- 2.1 Principles of network applications
  - app architectures
  - app requirements
- 2.2 Web and HTTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server
More about CDNs

Routing requests
- CDN creates a “map”, indicating distances from leaf ISPs and CDN nodes
- When query arrives at authoritative DNS server:
  - Server determines ISP from which query originates
  - Uses “map” to determine best CDN server

Not just Web pages
- Streaming stored audio/video
- Streaming real-time audio/video
  - CDN nodes create application-layer overlay network
Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
Searching for Information in a P2P Community

- Information index - mapping of information to locations of hosts that have the information
- Peers - a large number of participating peers
- File sharing
  - Dynamically track the files that the peers are making available for sharing
  - Peers come and go
  - Peers obtain new and delete copies.
- Instant messaging, cellular telephone, Internet telephony system
  - Presence tracking
P2P file sharing

**Example**

- Alice runs **P2P client application** on her notebook computer.
- Intermittently connects to Internet; gets **new IP address** for each connection.
- Asks for “Hey Jude”
- Application displays other peers that have copy of Hey Jude.

- Alice chooses **one of the peers**, Bob.
- File is copied from Bob’s PC to Alice’s notebook: **HTTP**
- While Alice downloads, other users uploading from Alice.
- Alice’s peer is both a Web client and a transient Web server.

*All peers are servers = highly scalable!*
Organizing and Searching for Index in a P2P community: three approaches

- Centralized index
- Query flooding
- Hierarchical overlay
P2P: #1 Centralized directory

original “Napster” design

1) when peer connects, it informs central server:
   - IP address
   - content

2) Alice queries for “Hey Jude”

3) Alice requests file from Bob
P2P: #2 Query flooding: Gnutella

- fully distributed
  - no central server
- public domain protocol
- many Gnutella clients implementing this protocol

Overlay network: graph

- edge between peer X and Y if there’s a TCP connection
- all active peers and edges form overlay net
- edge: virtual (not physical) link
- given peer typically connected with < 10 overlay neighbors
Gnutella: protocol

Query flooding process
- Query message sent over existing TCP connections
- Peers forward Query message
- QueryHit message sent over reverse path

- Simple
- Scalability: limited scope flooding (peer-count)

File transfer: HTTP
Gnutella: Peer joining

1. joining peer Alice must find another peer in Gnutella network: use list of candidate (bootstrap) peers
2. Alice sequentially attempts TCP connections with candidate peers until one connection setup with Bob (use this node to learn about others)
3. Flooding: Alice sends Ping message to Bob; Bob forwards Ping message to his overlay neighbors (neighbors forward query....)
   - peers receiving Ping message respond to Alice with Pong message
4. Alice receives many Pong messages, and can then setup additional TCP connections
P2P: #3 Hierarchical Overlay

- between centralized index, query flooding approaches

- each peer is either a group leader or assigned to a group leader.
  - TCP connection between peer and its group leader.
  - TCP connections between some pairs of group leaders.

- group leader tracks content in its children
Comparing Client-server, P2P architectures

**Question**: How much time distribute file initially at one server to $N$ other computers?

$u_s$: server upload bandwidth

$u_i$: client/peer i upload bandwidth

$d_i$: client/peer i download bandwidth
Client-server: file distribution time

- server sequentially sends N copies:
  - $NF/u_s$ time
- client i takes $F/d_i$ time to download

Time to distribute $F$ to $N$ clients using client/server approach:

$$d_{cs} = \max \{ NF/u_s, F/\min(d_i) \}$$

increases linearly in $N$ (for large $N$)
P2P: file distribution time

- server must send one copy: $\frac{F}{u_s}$ time
- client $i$ takes $\frac{F}{d_i}$ time to download
- $NF$ bits must be downloaded (aggregate)
  - fastest possible upload rate (assuming all nodes sending file chunks to same peer): $u_s + \sum_{i=1,N} u_i$

$$d_{P2P} = \max \left\{ \frac{F}{u_s}, \frac{F}{\min(d_i)}, \frac{NF}{u_s + \sum_{i=1,N} u_i} \right\}$$
Comparing Client-server, P2P architectures

![Graph comparing minimum distribution time for P2P and Client-Server architectures](image-url)
P2P Case Study: BitTorrent

- P2P file distribution

**tracker**: tracks peers participating in torrent

**torrent**: group (100+~1000+) of peers exchanging chunks of a file

obtain a sublist of peers in the torrent

trading chunks

peer
BitTorrent (1)

- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain
BitTorrent (2)

Pulling Chunks
- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice issues requests for her missing chunks
  - rarest first (fewest repeated copies among her neighbors; attempt to equalize the number of copies of each chunk in the torrent)

Sending Chunks: tit-for-tat (equivalent retaliation)
- Alice sends chunks to four neighbors that send her chunks at the highest rate (continuous measurement)
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
BitTorrent (3)

- If the two peers are satisfied with the trading, they will put each other in their top lists until one of them finds a better partner.
- The effect: peers are capable of uploading at compatible rates tend to find each other.
- A trading algorithm
P2P Case study: Skype

- P2P (pc-to-pc, pc-to-phone, phone-to-pc) Voice-Over-IP (VoIP) application
  - also IM
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay
Skype: making a call

- User starts Skype
- SC registers with SN
  - list of bootstrap SNs
- SC logs in
  (authenticate)
- Call: SC contacts SN for callee ID
  - SN contacts other SNs
    (unknown protocol, maybe flooding) to find addr of callee; returns addr to SC
- SC directly contacts callee, over TCP
Chapter 2: Summary

- application architectures
  - client-server
  - P2P
  - hybrid

- application service requirements:
  - reliability, bandwidth, delay

- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, Skype

- socket programming
homework

Pages 208 - 212: #7, #9, #15, #16 and #21.