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 1996-2009
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

Our 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
Creating a network app

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

No need to write software for network-core devices
- 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 applications
- 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 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
Google Data Centers

- Estimated cost of data center: $600M
- Google spent $2.4B in 2007 on new data centers
Pure P2P architecture

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

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

**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**

**Client process:** process that initiates communication

**Server process:** process that waits to be contacted

- Note: 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
- \textbf{Q:} does IP address of host suffice for identifying the process?
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
- Message syntax:
  - what fields in messages & how fields are delineated
- Message semantics
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

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

Throughput
- some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- other apps (“elastic apps”) make use of whatever throughput they get

Security
- Encryption, data integrity, …
## Transport service requirements of common apps

<table>
<thead>
<tr>
<th>Application</th>
<th>Data loss</th>
<th>Throughput</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</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>video: 10kbps-5Mbps</td>
<td></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 throughput guarantees, security

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

**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>HTTP (eg Youtube), RTP [RFC 1889]</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>SIP, RTP, proprietary (e.g., Skype)</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 applications
- 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 overview

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 overview (continued)

Uses TCP:
- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol 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

Aside

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.

Persistent HTTP
- Multiple objects can be sent over a single TCP connection between client and server.
HTTP request message

- two types of HTTP messages: \textit{request, response}
- HTTP request message:
  - \textbullet{} ASCII (human-readable format)

\begin{itemize}
  \item \textbf{request line (GET, POST, HEAD commands)}
  \item \textbf{header lines}
  \item \textbf{Carriage return, line feed indicates end of message}
\end{itemize}

\texttt{GET /somedir/page.html HTTP/1.1}
\texttt{Host: \textit{www.someschool.edu}}
\texttt{User-agent: \textit{Mozilla/4.0}}
\texttt{Connection: close}
\texttt{Accept-language:fr}

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

```
method  sp  URL  sp  version  cr  lf
header field name : value  cr  lf
•
•
•
header field name : value  cr  lf
```
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/animalsearch?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 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
Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:
   
   telnet cis.poly.edu 80
   
   Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at cis.poly.edu

2. Type in a GET HTTP request:

   GET /~ross/ HTTP/1.1
   Host: cis.poly.edu

   By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!
User-server state: cookies

Many major Web sites use cookies

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

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

client

- ebay 8734
- cookie file
- ebay 8734
- amazon 1678

usual http request msg
Set-cookie: 1678

usual http response msg
cookie: 1678

server

Amazon server creates ID 1678 for user
create entry

cookie-specific action
access
access

one week later:

- ebay 8734
- amazon 1678

usual http request msg
cookie: 1678

usual http response msg
Cookies (continued)

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

How to keep “state”:
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

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

Diagram:
- Client
- Proxy server
- Origin server
- HTTP request
- HTTP response
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

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 (cont)

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

consequence
- 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
Caching example (cont)

possible solution: install cache
- suppose hit rate is 0.4

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 avg delay = Internet delay + access delay + LAN delay = 0.6*(2.01) secs + 0.4*milliseconds < 1.4 secs
Conditional GET

- **Goal**: don’t send object if cache has up-to-date cached version
- **cache**: 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>`
Content Distribution Networks

- Strategically deploy servers for performance, scalability, cost effectiveness.
- Top 3: Akamai, Limelight, and CDNetworks
- Mirror content, DNS redirect,
- P2P CDNs
Content distribution networks (CDNs)

- The content providers are the CDN customers.

**Content replication**

- CDN company installs hundreds of CDN servers throughout Internet
  - close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers
**CDN example**

1. Origin server
   - HTTP request for www.foo.com/sports/sports.html

2. DNS query for www.cdn.com
   - CDN company 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
- Static files on CDN
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

**Companies:**
Content distribution

- Web caching
- Content distribution networks (CDNs)
- Peer-to-peer file sharing

- Web caching helps user-side bottleneck
- CDN helps server-side bottleneck
Web applications

- Thin client vs. “thick” client
- Also referred to as Rich Internet Applications
- Benefits:
  - Little disk space, automatic upgrade, integrate with other web procedures, cross-platform compatibility, mobility friendly, business reasons
- Limits
  - Internet connectivity, inconsistency in browser implementations, (currently) limited functionalities
Examples

- Google Documents

- Gears (original Google Gears), a software platform for Ajax web applications development.
  - open-source, BSD-license, incremental
  - Offline capability (e.g., Google Reader, Google Docs)
  - Three components
    - LocalServer: to access the application offline
    - Database: to store large amounts of structured data
    - WorkerPool: to perform long sync when you reconnect (w.o. blocking the browser).

- Google browser: Chrome
- Ajax, Adobe Flash

- ASP (Application service provider)
  - Provide web access to software (for monthly or yearly fees)

- Software as a service
- Computing as a service
Web 2.0

- Origin: Web 2.0 Conference in 2004
- "embrace the strength of the web and use it as a platform", -- Tim O'Reilly
- Enabling technologies: Ajax, Flex, etc.

- Rich user experience, user participation, dynamic content, meta data, web standards and scalability.
- Interactive, more cooperation, distributed
- Craigslist, Flickr, del.icio.us, Wikipedia, Adsense, Ebay, Twitter, YouTube
- Blog, podcast, tagging, etc.
- Critics
Long Tail

- In statistics, refers to distributions that are probability distributions whose tails are not exponentially bounded.
- The niche strategy of businesses, such as Amazon.com or Netflix, that sell a large number of unique items, each in relatively small quantities.
- Large stores vs. small businesses.
- An example of “embracing the Internet”, instead of fighting it.
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
- 2.9 Building a Web server
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, TCP is transport protocol
- Client authorized over control connection
- Client browses remote directory by sending commands over control connection.
- When server receives file transfer command, server opens 2nd TCP connection (for file) to client
- After transferring one file, server closes data connection.
- Server opens another 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
[liu@shannon ~]$ ftp -v ftp.uu.net
Connected to ftp.uu.net.
220 FTP server ready.
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Name (ftp.uu.net:liu): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230-
230- Welcome to the UUNET archive.
230- A service of UUNET Technologies Inc, Falls Church, Virginia
230- For information about UUNET, call +1 703 206 5600, or see the files
230- in /uunet-info
....
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (192,48,96,9,118,43)
150 Opening ASCII mode data connection for /bin/ls.
total 20088
drwxr-sr-x 2 1 512 Jun 29 2001 .forward
-rw-r--r-- 1 11 0 Jun 29 2001 .hushlogin
-rw-r--r-- 1 100 59 Jun 29 2001 .kermrc
226 Transfer complete.

ftp> ls
227 Entering Passive Mode (192,48,96,9,125,39)
150 Opening ASCII mode data connection for /bin/ls.
total 20088
drwxr-sr-x 2 1 512 Jun 29 2001 .forward
-rw-r--r-- 1 11 0 Jun 29 2001 .hushlogin
-rw-r--r-- 1 100 59 Jun 29 2001 .kermrc
-rw-r--r-- 1 100 0 Jun 29 2001 .notar
226 Transfer complete.

ftp> bi
200 Type set to I.

ftp> quit
221-You have transferred 0 bytes in 0 files.
221-Total traffic for this session was 7739 bytes in 2 transfers.
221-Thank you for using the FTP service on neo-ftp.uu.net.
221 Goodbye.

Note: server does the active open/close of the data connection.
Connection close=file complete
SFTP

- SFTP: SSH file transfer protocol or Secure File transfer protocol
- Main motivation: encrypts both commands and data.
- Graphical and command line SFTP client.
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
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, Mozilla Thunderbird
- 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
  - 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 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
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
jadzia:~ % mail -v xinliu@shay.ecn.purdue.edu
Subject: hello
test
.
EOT
xinliu@shay.ecn.purdue.edu... Connecting to shay.ecn.purdue.edu. via esmtp...
220 shay.ecn.purdue.edu ESMTP Sendmail 8.12.10/8.12.10; Wed, 19 Nov 2003 15:35:00 -0500 (EST)
>>> EHLO jadzia.ifp.uiuc.edu
250-shay.ecn.purdue.edu Hello jadzia.ifp.uiuc.edu [130.126.122.22], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-STARTTLS
250-DELIVERBY
250 HELP
>>> MAIL From:<xinliu@ifp.uiuc.edu> SIZE=70
250 2.1.0 <xinliu@ifp.uiuc.edu>... Sender ok
>>> RCPT To:<xinliu@shay.ecn.purdue.edu>
>>> DATA
250 2.1.5 <xinliu@shay.ecn.purdue.edu>... Recipient ok
354 Enter mail, end with "." on a line by itself
>>>.
250 2.0.0 hAJKZ06i019357 Message accepted for delivery
xinliu@shay.ecn.purdue.edu... Sent (hAJKZ06i019357 Message accepted for delivery)
Closing connection to shay.ecn.purdue.edu.
>>> QUIT
221 2.0.0 shay.ecn.purdue.edu closing connection
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
```

- MIME version
- method used to encode data
- multimedia data type, subtype, parameter declaration
- encoded data
**MIME types**

Content-Type: type/subtype; parameters

**Text**
- example subtypes: plain, html

**Image**
- example subtypes: jpeg, gif

**Audio**
- example subtypes: basic (8-bit mu-law encoded), 32kadpcm (32 kbps coding)

**Video**
- example subtypes: mpeg, quicktime

**Application**
- other data that must be processed by reader before “viewable”
- example subtypes: msword, octet-stream

**Multipart**
Message format: multimedia extensions

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

From srikant@ifp.uiuc.edu Fri Oct 18 14:59:04 2002
Received: from dosai.csl.uiuc.edu (dosai.csl.uiuc.edu [130.126.137.172])
    by jadzia.ifp.uiuc.edu (8.10.1/8.10.1) with ESMTP id g9IjvwM27934
    for <xinliu@ifp.uiuc.edu>; Fri, 18 Oct 2002 14:58:57 -0500 (CDT)
Received: from localhost (srikant@localhost)
    by dosai.csl.uiuc.edu (8.10.0/8.10.0) with ESMTP id g9Ijx2923097
    for <xinliu@dosai.csl.uiuc.edu>; Fri, 18 Oct 2002 14:59:02 -0500 (CDT)
Date: Fri, 18 Oct 2002 14:59:01 -0500 (CDT)
From: Rayadurgam Srikant <srikant@ifp.uiuc.edu>
To: Xin Liu <xinliu@ifp.uiuc.edu>
Subject: CV
Message-ID: <Pine.GSO.4.44.0210181458520.23095-101000@dosai.csl.uiuc.edu>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII
Content-Length: 274
Status: RO
X-Status:
X-Keywords:
X-UID: 3
This message is in MIME format. The first part should be readable text, while the remaining parts are likely unreadable without MIME-aware tools. Send mail to mime@docserver.cac.washington.edu for more info.

---559023410-851401618-1034971141=:23095
Content-Type: TEXT/PLAIN; charset=US-ASCII

--------
R. Srikant                                      1308 W. Main Street
Associate Professor                             Urbana, IL 61801
Coordinated Science Lab. and                    (217) 333-2457 (Phone)
Department of General Engineering               (217) 244-1642 (Fax)
University of Illinois                          rsrikant@uiuc.edu
http://comm.csl.uiuc.edu/~srikant

---559023410-851401618-1034971141=:23095
Content-Type: APPLICATION/PostScript; name="my_bio.ps"
Content-Transfer-Encoding: BASE64
Content-Description:
Content-Disposition: attachment; filename="my_bio.ps"
4.5.2. TRANSPARENCY

The mail data may contain any of the 128 ASCII characters. All characters are to be delivered to the recipient's mailbox including format effectors and other control characters. If the transmission channel provides an 8-bit byte (octets) data stream, the 7-bit ASCII codes are transmitted right justified in the octets with the high order bits cleared to zero.

In some systems it may be necessary to transform the data as it is received and stored. This may be necessary for hosts that use a different character set than ASCII as their local character set, or that store data in records rather than strings. If such transforms are necessary, they must be reversible -- especially if such transforms are applied to mail being relayed.
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.
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
POP3 (more) and IMAP

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

IMAP
- Keep all messages in one place: the server
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name
Social and Economic Impacts

- Social impact
- Economic impact
- Managing emails
- Etiquette
- Spam
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
- 2.9 Building a Web server
DNS: Domain Name System

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

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., www.yahoo.com - 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 host, routers, name servers to communicate to resolve names (address/name translation)

- note: core Internet function, implemented as application-layer protocol
- complexity at network’s “edge”
DNS

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

2: Application Layer
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?
DNS: caching and updating records

- once (any) 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:** \((\text{name, value, type, ttl})\)

- **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></th>
<th>identification</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of questions</td>
<td>number of answer RRss</td>
<td></td>
</tr>
<tr>
<td>number of authority RRss</td>
<td>number of additional RRss</td>
<td></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)
### DNS protocol, messages

<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>

- **Name, type fields** for a query
- **RRs** in response to query
- **Records** for authoritative servers
- Additional “helpful” info that may be used
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?
- Domain name value
Transport Layer

- UDP or TCP

- “Queries are messages which may be sent to a name server to provoke a response. In the Internet, queries are carried in UDP datagrams or over TCP connections.” RFC 1034

- “An update transaction may be carried in a UDP datagram, if the request fits, or in a TCP connection (at the discretion of the requestor). When TCP is used, the message is in the format described in [RFC1035 4.2.2].” RFC 2136, DNS update
DNS related RFCs

- Over 100 DNS-related RFC, about 20 obsolete ones.

- [http://www.dns.net/dnsrd/rfc/](http://www.dns.net/dnsrd/rfc/)
DNS attacks

- DNS cache poisoning attacks
- ...

2: Application Layer
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 applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
Pure P2P architecture

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

**Three topics:**
- File distribution
- Searching for information
- Case Study: Skype
**File Distribution: Server-Client vs P2P**

**Question**: How much time to distribute file from one server to \( N \) peers?

- \( u_s \): server upload bandwidth
- \( u_i \): peer i upload bandwidth
- \( d_i \): peer i download bandwidth

Diagram:
- Server
- File, size \( F \)
- \( u_s \) to peers
- \( d_N \) from peer
- Network (with abundant bandwidth)
File distribution time: server-client

- 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:

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

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

- server must send one copy: $F/u_s$ time
- client $i$ takes $F/d_i$ time to download
- $NF$ bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \sum u_i$

$$d_{P2P} = \max \{ F/u_s, F/\min(d_i), NF/(u_s + \sum u_i) \}$$
Server-client vs. P2P: example

Client upload rate = u, \( \frac{F}{u} = 1 \) hour, \( u_s = 10u \), \( d_{\text{min}} \geq u_s \)
File distribution: BitTorrent

- P2P file distribution

**tracker**: tracks peers participating in torrent

**torrent**: group of peers exchanging chunks of a file

![Diagram showing peer-to-peer file distribution with a tracker and peers trading chunks of a file.](image)
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 sends requests for her missing chunks
  - rarest first

**Sending Chunks: tit-for-tat**
- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - “optimistically unchoke”
BitTorrent: Tit-for-tat

(1) Alice “optimistically unchokes” Bob
(2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
(3) Bob becomes one of Alice’s top-four providers

With higher upload rate, can find better trading partners & get file faster!
P2P: searching for information

Index in P2P system: maps information to peer location (location = IP address & port number)

File sharing (eg e-mule)
- Index dynamically tracks the locations of files that peers share.
- Peers need to tell index what they have.
- Peers search index to determine where files can be found

Instant messaging
- Index maps user names to locations.
- When user starts IM application, it needs to inform index of its location
- Peers search index to determine IP address of user.
P2P: centralized index

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: problems with centralized directory

- single point of failure
- performance bottleneck
- copyright infringement: “target” of lawsuit is obvious

file transfer is decentralized, but locating content is highly centralized
Query flooding

- fully distributed
  - no central server
- used by Gnutella
- Each peer indexes the files it makes available for sharing (and no other files)

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
Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers
DHT Identifiers

- Assign integer identifier to each peer in range $[0, 2^n-1]$.  
  - Each identifier can be represented by $n$ bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
  - eg, key = h("Led Zeppelin IV")
  - This is why they call it a distributed “hash” table
How to assign keys to peers?

- **Central issue:**
  - Assigning (key, value) pairs to peers.

- **Rule:** assign key to the peer that has the closest ID.

- **Convention in lecture:** closest is the immediate successor of the key.

- **Ex:** n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1
Circular DHT (1)

- Each peer only aware of immediate successor and predecessor.
- “Overlay network”
Circle DHT (2)

O(N) messages on avg to resolve query, when there are N peers

Define closest as closest successor

Who’s resp for key 1110?
Circular DHT with Shortcuts

- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Reduced from 6 to 2 messages.
- Possible to design shortcuts so $O(\log N)$ neighbors, $O(\log N)$ messages in query.

Who's resp for key 1110?
Peer Churn

- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- What if peer 13 wants to join?

- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.
P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs
Peers as relays

- Problem when both Alice and Bob are behind “NATs”.
  - NAT prevents an outside peer from initiating a call to insider peer

- Solution:
  - Using Alice’s and Bob’s SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay
P2P

- Legality issue
- Network operator p2p filtering
- Network neutrality
Chapter 2: Summary

our study of network apps now complete!

- 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
Chapter 2: Summary

Most importantly: learned about protocols

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code

- message formats:
  - headers: fields giving info about data
  - data: info being communicated

Important themes:
- control vs. data msgs
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- “complexity at network edge”
Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

- Winston Churchill