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
- 2.9 Building a Web server

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
- 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@somemail.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)
```

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

<table>
<thead>
<tr>
<th>SMTP</th>
<th>SMTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>user agent</td>
<td>user agent</td>
</tr>
<tr>
<td>sender's mail server</td>
<td>receiver's mail server</td>
</tr>
</tbody>
</table>

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 3501]
  · more features (more complex)
  · manipulation of stored messages on server
HTTP: 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

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
2.9 Building a Web server

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
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 to host, routers, name servers to communicate to resolve names

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

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 cannot resolve name root name server
- contacts authoritative name server if name mapping not known
- gets mapping returns mapping to local name server

DNS services
- Hostname to IP address translation
- IP address to Hostname translation
- Host aliasing
  - Canonical and alias names
  - Mail server aliasing
- Load distribution
  - Replicated Web servers:
    - set of IP addresses for one canonical name
    - But performs poorly - suppose Aol cached one IP address

Why not centralize DNS?
- single point of failure
- traffic volume
- distant centralized database maintenance
- doesn't scale!

Root DNS Servers

yahoo.com DNS servers
amazon.com DNS servers
com DNS servers
org DNS servers
edu DNS servers
poly.edu umass.edu DNS servers

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

84 root name servers worldwide
See www.root-servers.org
**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 and 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 a host makes a DNS query, query is sent to its local DNS server
- Acts as a proxy, forwards query into hierarchy

**Example**

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu
- Find gaia.cs.umass.edu
- Find gaia.cs.umass.edu
- List of IPs for TLD servers for edu
- Find gaia.cs.umass.edu
- IP for authoritative server for umass.edu
- Find gaia.cs.umass.edu
- IP for gaia.cs.umass.edu
- IP for gaia.cs.umass.edu

**Recursive queries**

- recursive query:
  - Name server finds answer
  - puts burden of name resolution on contacted name server
  - heavy load?

- iterated query:
  - contacted server replies with name of server to contact
  - "I don't know this name, but ask this server"
DNS: caching and updating records

When a name server learns a mapping, it **caches** the mapping.

- A server discards cached entries after a timeout (typically 2 days).
- TLD servers typically cached in local name servers.
- Thus root name servers queried infrequently.

**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 IP address of authoritative name server for this domain
- **Type=CNAME**
  - `name` is alias name for some "canonical" (the real) name
- **Type=MX**
  - `value` is canonical name
  - `value` is name of mailserver associated with `name`
Inserting records into DNS

Example: just created startup "Network Utopia"
Register name networkuptopia.com at a registrar (e.g., Network Solutions)
Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
Registrar inserts two RR into the com TLD server:

(networkuptopia.com, dns1.networkuptopia.com, NS)
(dns1.networkuptopia.com, 212.212.212.1, A)

Put in authoritative server Type A record for www.networkuptopia.com and Type MX record for networkuptopia.com

How do people get the IP address of your Web site?

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!

P2P: 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: problems with centralized directory

- Single point of failure
- Performance bottleneck
- Copyright infringement

file transfer is decentralized, but locating content is highly decentralized

Query flooding: Gnutella

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

overlay network: graph
- edge between peer X and Y if there's a TCP connection
- all active peers and edges is overlay net
- Edge is not a physical link
- Given peer will typically be connected with < 10 overlay neighbors

Gnutella: protocol

- Query message sent over existing TCP connections
- peers forward Query message
- QueryHit sent over reverse path

File transfer: HTTP

Scalability: limited scope flooding

Gnutella: Peer joining

1. Joining peer X must find some other peer in Gnutella network: use list of candidate peers
2. X sequentially attempts to make TCP with peers on list until connection setup with Y
3. X sends Ping message to Y; Y forwards Ping message.
4. All peers receiving Ping message respond with Pong message
5. X receives many Pong messages. It can then setup additional TCP connections

Peer leaving: see homework problem!
Questions about Gnutella, 1

What are 'little-endian' and 'big-endian'? Why does the protocol have to specify them?
Unique identifiers: How are unique Descriptor IDs and Servent Identifiers generated?
The spec says (p 3, para 2) "if a servent becomes out of synch with its input stream, it should drop the connection". How would it know?

Questions about Gnutella, 2

In the section 'Descriptor Routing' on page 5, the spec says "Pong descriptors may only be sent along the same path that carried the incoming Ping descriptor" and "Push descriptors may only be sent along the same path that carried the incoming QueryHit descriptor." How would this be implemented?
In the section 'Firewalled Servents' the spec says "A servent can request a file push by routing a Push request back to the servent that sent the QueryHit descriptor describing the target file." How is this possible? Isn't the latter servent behind a firewall?

Exploiting heterogeneity: KaZaA

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 the content in all its children.

KaZaA: Querying

Each file has a hash and a descriptor
Client sends keyword query to its group leader
Group leader responds with matches:
    For each match: metadata, hash, IP address
If group leader forwards query to other group leaders, they respond with matches
Client then selects files for downloading
    HTTP requests using hash as identifier sent to peers holding desired file
Kazaa tricks

- Limitations on simultaneous uploads
- Request queuing
- Incentive priorities
- Parallel downloading
Internet and Intranet Protocols and Applications

Lecture 3:
Application Layer 2:
Email, DNS and P2P

February 1, 2005

Arthur Goldberg
Computer Science Department
New York University
artg@cs.nyu.edu
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:
q 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!)
q 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 1996-2004
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 file sharing
• 2.7 Socket programming with TCP
• 2.8 Socket programming with UDP
• 2.9 Building a Web server
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

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
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

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

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 3501]
- more features (more complex)
- manipulation of stored messages on server

HTTP: 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**

---

C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

S: +OK POP3 server ready
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
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
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 services

Hostname to IP address translation

IP address to Hostname translation

Host aliasing
- Canonical and alias names

Mail server aliasing

Load distribution
- Replicated Web servers:
  set of IP addresses for one canonical name
- But performs poorly - suppose Aol cached one IP address

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

doesn't scale!
Distributed, Hierarchical Database

Root DNS Servers

- com DNS servers
  - yahoo.com DNS servers
  - amazon.com DNS servers
- org DNS servers
  - pbs.org DNS servers
- edu DNS servers
  - poly.edu DNS servers
  - umass.edu DNS servers

**Client wants IP for www.amazon.com; 1st approx:**

1. Client queries a root server to find com DNS server
2. Client queries com DNS server to get amazon.com DNS server
3. Client queries amazon.com DNS server to get IP address for www.amazon.com
DNS: Root name servers

contacted by local name server that cannot resolve name root name server:

contacts authoritative name server if name mapping not known

gets mapping

returns mapping to local name server

84 root name servers worldwide

See www.root-servers.org
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 and 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 a host makes a DNS query, query is sent to its local DNS server
  Acts as a proxy, forwards query into hierarchy
Example

Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

2. Find gaia.cs.umass.edu
3. Find gaia.cs.umass.edu
4. List of IPs for TLD servers for edu
5. Find gaia.cs.umass.edu
6. IP for authoritative server for umass.edu
7. Find gaia.cs.umass.edu
8. IP for gaia.cs.umass.edu
9. IP for gaia.cs.umass.edu
Recursive queries

**recursive query:**
Name server finds answer
puts burden of name resolution on contacted name server
heavy load?

**iterated query:**
contacted server replies with name of server to contact
"I don't know this name, but ask this server"
DNS: caching and updating records

When a name server learns a mapping, it caches the mapping.

A server discards cached entries after a timeout (typically 2 days).

TLD servers typically cached in local name servers.
- Thus root name servers queried infrequently.

Update/notify mechanisms under design by IETF.

RFC 2136

www.ietf.org/html.charters/dnsind-charter.html
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 IP address of authoritative name server for this domain

- **Type=CNAME**
  - name is alias name for some “cannonical” (the real) name
  - value is cannonical name

- **Type=MX**
  - value is name of mailserver associated with name

www.ibm.com is really servereast.backup2.ibm.com
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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(variable number of questions)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>(variable number of resource records)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>(variable number of resource records)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>(variable number of resource records)</td>
</tr>
</tbody>
</table>

2: Application Layer  27
**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)

12 bytes
Inserting records into DNS

Example: just created startup “Network Utopia”
Register name networkuptopia.com at a registrar (e.g., Network Solutions)

Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
Registrar inserts two RRs into the com TLD server:

(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)

Put in authoritative server Type A record for www.networkuptopia.com and Type MX record for networkutopia.com

How do people get the IP address of your Web site?
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
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!
P2P: 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: problems with centralized directory

Single point of failure
Performance bottleneck
Copyright infringement

file transfer is decentralized, but locating content is highly decentralized
Query flooding: Gnutella

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

overlay network: graph
edge between peer X
and Y if there's a TCP
connection
all active peers and
edges is overlay net
Edge is not a physical
link
Given peer will
typically be connected
with < 10 overlay
neighbors
Gnutella: protocol

Query message sent over existing TCP connections
peers forward
Query message
QueryHit sent over reverse path

Scalability: limited scope flooding

File transfer: HTTP
Query
QueryHit
Query
Query
QueryHit
Query
QueryHit
Query

2: Application Layer
Gnutella: Peer joining

1. Joining peer X must find some other peer in Gnutella network: use list of candidate peers
2. X sequentially attempts to make TCP with peers on list until connection setup with Y
3. X sends Ping message to Y; Y forwards Ping message.
4. All peers receiving Ping message respond with Pong message
5. X receives many Pong messages. It can then setup additional TCP connections

Peer leaving: see homework problem!
Questions about Gnutella, 1

What are ‘little-endian’ and ‘big-endian’? Why does the protocol have to specify them?

Unique identifiers: How are unique Descriptor IDs and Servent Identifiers generated?

The spec says (p 3, para 2) “if a servent becomes out of synch with its input stream, it should drop the connection”. How would it know?
Questions about Gnutella, 2

In the section ‘Descriptor Routing’ on page 5, the spec says “Pong descriptors may only be sent along the same path that carried the incoming Ping descriptor” and “Push descriptors may only be sent along the same path that carried the incoming QueryHit descriptor.” How would this be implemented?

In the section ‘Firewalled Servents’ the spec says “A servent can request a file push by routing a Push request back to the servent that sent the QueryHit descriptor describing the target file.” How is this possible? Isn’t the latter servent behind a firewall?
Exploiting heterogeneity: KaZaA

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 the content in all its children.
KaZaA: Querying

Each file has a hash and a descriptor
Client sends keyword query to its group leader
Group leader responds with matches:
   For each match: metadata, hash, IP address
If group leader forwards query to other group leaders, they respond with matches
Client then selects files for downloading
   HTTP requests using hash as identifier sent to peers holding desired file
**Kazaa tricks**

Limitations on simultaneous uploads
Request queuing
Incentive priorities
Parallel downloading