Midterm stats

Mean: 63.66 (out of 80)
Standard deviation: 11.34

Internet Protocol Stack

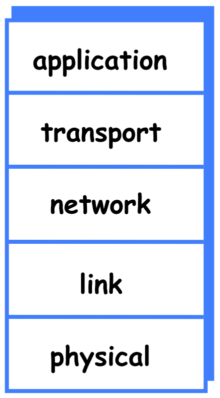
application: supporting network applications
- HTTP, SMTP, FTP, etc.

transport: endhost-endhost data transfer
- TCP, UDP

network: routing of datagrams from source to destination
- IP, routing protocols

link: data transfer between neighboring network elements
- Ethernet, WiFi

physical: bits "on the wire"



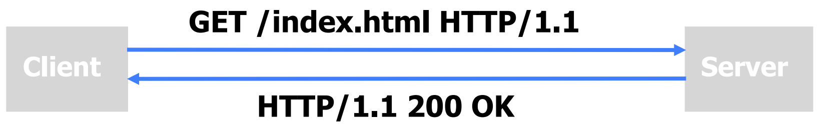
Principles of Network Applications

Our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- examine some popular application-level protocols
  - HTTP
  - SMTP / POP3 / IMAP

Application-Layer Protocols

- Messages exchanged between applications
  - syntax and semantics of the messages between end-hosts
  - tailored to the specific application (e.g., Web, e-mail)
  - Messages transferred over transport connection (e.g., TCP)
- Popular application-layer protocols
  - HTTP, SMTP, FTP, SSH, …

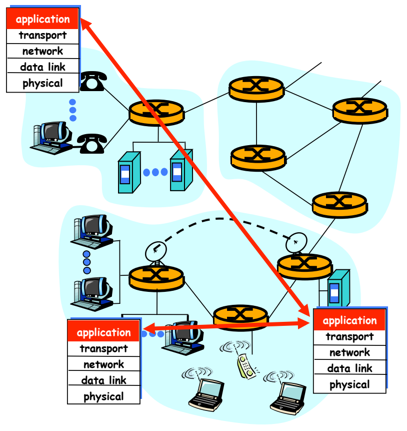


Some Network Apps (and Their Protocols)

- E-mail (SMTP)
- Web (HTTP)
- Instant messaging (IRC)
- Remote login (Telnet)
- P2P file sharing (Napster, Gnutella, KaZaa)
- Multi-user network games
- Streaming stored video clips (Adobe's RTMP)
- **Internet telephone (Skype)**
- **Real-time video conference (RTP)**
- **Massively parallel computing**

Creating a Network Application

- Write programs that
  - run on different end systems and
  - communicate over a network.
  - e.g., Web browser software communicates with browser server
- No app software written for devices in network core
  - Network core devices do not function at app layer
  - This design allows for rapid app development



Application Architectures

Peer-to-peer (p2p):

- hybrid of p2p and centralized server
- pure p2p
- hierarchical p2p
- end-host (p2p) multicast

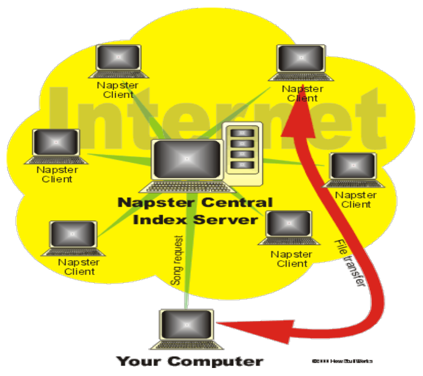
Client-server:

- DNS
- FTP
- SMTP
- HTTP
  - cookies
  - web caching
  - CDN
- Multi-player games

Hybrid of P2P and Centralized Server

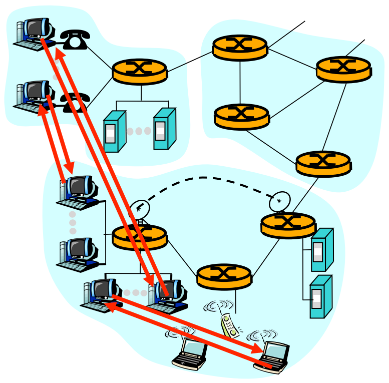
Napster:

- file transfer P2P
- file search centralized:
  - peers register IP address and content at central index server
  - peers query central index server to locate content



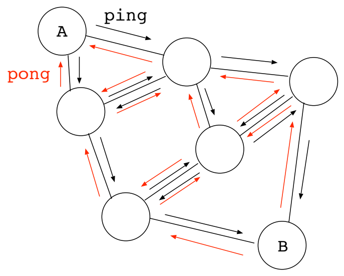
Pure P2P Architecture

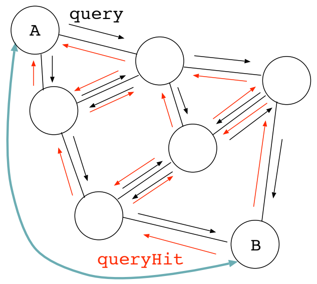
- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses
- example: Gnutella
- highly scalable (why?)
- but difficult to manage
  - how to find peer?
  - how to find content?



Gnutella

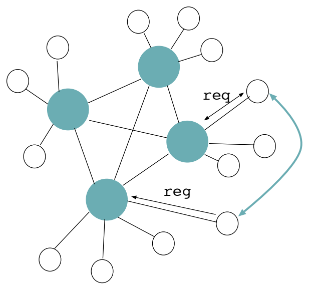
- no centralized index server
- network discovery using **ping** and **pong** messages
- file discovery using **query** and **queryHit** messages
- both ping and query messages are forwarded using the **flooding** algorithm: forward on all links except incoming one
- previously seen messages are not further forwarded
- new version of gnutella uses KaZaA-like supernodes





Hierarchical P2P

- FastTrack used by KaZaA, Groskster, iMesh, Morpheus
- hierarchical architecture
  - peers divided into **supernodes** and ordinary nodes
  - each supernode keeps an index of all its children's files
  - requests are sent to supernodes
  - supernodes query each other for files not in their local indices



- ordinary nodes are "promoted" to supernodes if they have enough resources and have stayed on network long enough
- parallel download of files

- eDonkey/eMule also builds a hierarchical network, but the "supernodes" are dedicated servers, not just more equal peers

Freenet: Anonymous P2P

- no index server
- requester doesn't connect directly to content provider
- instead, content passed in a bucket-brigade fashion from provider to requester
- the next time content is requested, it is provided from the nearest cache
- requester cannot differentiate provider from a cache holder or a forwarding peer (allow for anonymity)

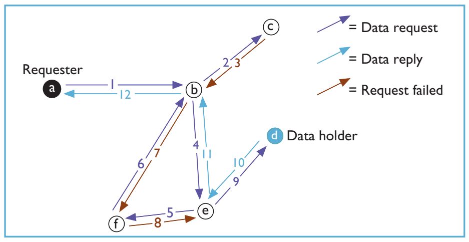
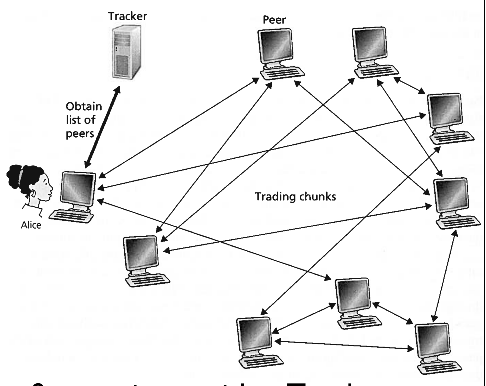


Figure 1. Typical request sequence. The request moves through the network from node to node, backing out of a dead-end (step 3) and a loop (step 7) before locating the desired file.

BitTorrent



Content distribution:

- content is divided into N pieces of 16KB each and sent to N peers

Content download:

- to download a file, a peer must first register with a Tracker
- Tracker returns a random list of peers who have the file
- peer opens about 5 TCP connections to the provided peers
- a peer will only upload to peers from whom it can also download (“tit-for-tat”)

Challenges for P2P Networks

1. NAT and firewall:
   - cannot peer with a host you can’t address

   Solutions:
   - Gnutella:
     - querier sends PUSH message to responder over the p2p network
     - responder opens a TCP connection to querier and send over the file
     - no luck if both are behind firewalls
   - KaZaA, eDonkey, Skype:
     - supernodes act as proxy if both peers are behind firewalls
   - Standards to circumvent NAT (and firewall!): UPnP, STUN

2. Download/upload bandwidth asymmetry

   ⇒ needs bandwidth subsidy by content provider or CDN,
   or suffer long download time

Application Architectures

Peer-to-peer (p2p):

- hybrid of p2p and centralized server
- pure p2p
- hierarchical p2p
- end-host (p2p) multicast

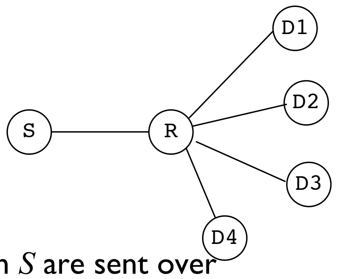
Client-server:

- DNS
- FTP
- SMTP
- HTTP
  - cookies
  - web caching
  - CDN
- Multi-player games

Modes of Delivery

Unicast, broadcast, multicast

Assuming a video conference involving *S*, *D2*, and *D3*



- unicasting: two copies of packets from *S* are sent over the *SR* link
- broadcasting: one copy of packet sent from *S* to all destinations, but packet sent to *D1* and *D4* unnecessarily
- multicasting: one copy of packets from *S* is sent over the *SR* link, *R* then sends one copy each to *D2* and *D3*

Multicast Delivery

Uses of multicasting:

- video conferencing, distance learning, distributed computation, p2p delivery, multi-player gaming, etc.

Multicast design goals:

- can support millions of receivers per multicast group
- receivers can join and leave any group at any time
- senders don't have to know all receivers
- senders don't have to be members of a group to send
- there could be more than one sender per group

Multicast Group Management

Issues in multicast group management:

1. how to advertise/discover a multicast group?
2. how to join a multicast group?
3. delivering multicast packets to the group

IPv4 multicast:

- use multicast (Class-D) addresses as anonymous rendezvous point
- create a well-known multicast group (address) to advertise/discover multicast groups
- multicast data is sent using UDP
  - sender `sendto()` the multicast address
  - receiver `recvfrom()` the multicast address
- not uniformly deployed throughout the Internet

End-host Multicast

Issues in multicast group management:

1. how to advertise/discover a multicast group?
2. how to join a multicast group?
3. delivering multicast packets to the group

End-host (p2p) multicast:

- use a well-known, centralized rendezvous server
- each peer must register with rendezvous server
- rendezvous server returns a (random) list of peers
- each peer can support only a limited number of peers
- avoid sending duplicate messages and looping:
  - if single source, construct a shortest-path tree rooted at source
  - or use flood-and-prune algorithm
- prefer peers in same subnet

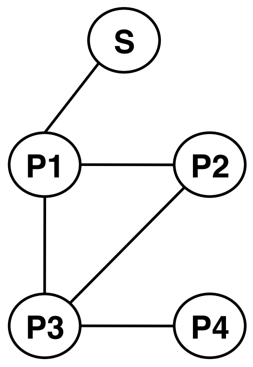
Flood and Prune

How to ensure that only one copy of packet from *S* is forwarded by *P3* to *P4*?

- keep track of sequence number
- only forward packet that comes from shortest path from (to) source

How to ensure that only one copy of packet from *S* reaches *P3*?

- only forward if self is on neighbor's shortest path from (to) source
- prune (*P3* telling *P2* not to forward pkts from *S*)
  - must be done per source if there are multiple sources, each source forming its own multicast group and (logically) its own multicast tree
  - must periodically flood in case of membership change



Application Architectures

Peer-to-peer (p2p):

- hybrid of p2p and centralized server
- pure p2p
- hierarchical p2p
- end-host (p2p) multicast

Client-server:

- DNS
- FTP
- SMTP
- HTTP
  - cookies
  - web caching
  - CDN
- Multi-player games

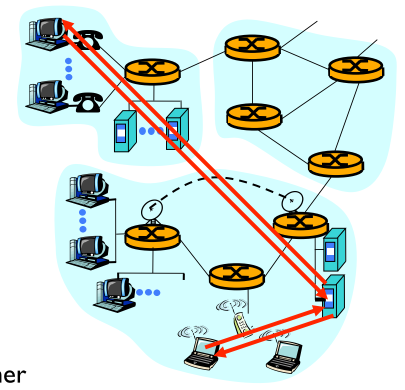
Client-Server Computing

server:

- a process that manages access to a resource
- usually has a permanent IP address
- waits for connection
- server farms for scaling
  - how do server farms maintain a single IP address externally?

client:

- a process that needs access to a resource
- initiates connection with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

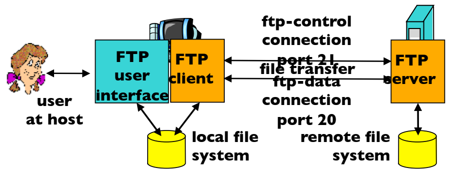


Process vs. machine

File Transfer Protocol (FTP, RFC959)

Transfer file to and/or from remote host

- client: the side that initiates transfer (either to/from remote)
- server: remote host
  - server maintains "state": current directory, earlier authentication



- separate control, data connections
  - server listens on port 21 for control connection from client
  - client obtains authorization and send commands over control connection
  - when server receives a command for a directory listing or file transfer, the server opens a separate TCP data connection to client
    - client listens for connection on an ephemeral port
    - client sends ephemeral port number to server
    - server uses port 20 for the ftp-data connection to client's ephemeral port
  - after each directory/file transfer, server closes ftp-data connection

What's the advantage of an out-of-band control channel?

FTP Commands

Sent as ASCII text over control channel

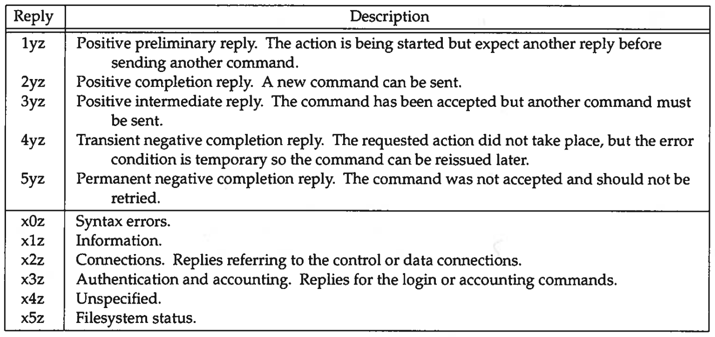
Sample commands:

Command	Description
ABOR	abort previous FTP command and any data transfer
LIST *filelist*	list files or directories
PASS *password*	password on server
PORT *n1,n2,n3,n4,n5,n6*	client IP address (*n1.n2.n3.n4*) and port (*n5* × 256 + *n6*)
QUIT	logoff from server
RETR *filename*	retrieve (get) a file
STOR *filename*	store (put) a file
SYST	server returns system type
TYPE *type*	specify file type: A for ASCII, I for image
USER *username*	username on server

Stevens

FTP Reply Codes

Meanings of the first and second digits of the reply code:



Reply	Description
1yz	Positive preliminary reply. The action is being started but expect another reply before sending another command.
2yz	Positive completion reply. A new command can be sent.
3yz	Positive intermediate reply. The command has been accepted but another command must be sent.
4yz	Transient negative completion reply. The requested action did not take place, but the error condition is temporary so the command can be reissued later.
5yz	Permanent negative completion reply. The command was not accepted and should not be retried.
x0z	Syntax errors.
x1z	Information.
x2z	Connections. Replies referring to the control or data connections.
x3z	Authentication and accounting. Replies for the login or accounting commands.
x4z	Unspecified.
x5z	Filesystem status.

Stevens

Samp

```
33
12                                                  ing
425 Can’t open data connection
452 Error writing file
```

Fixed Header vs. ASCII Commands

What are the advantages and disadvantages of using ASCII commands over fixed header as with IP or TCP?

Why limit ASCII commands to four letter words?

Why use numeric code in reply?

Why bother with ASCII message in reply?

Application Architectures

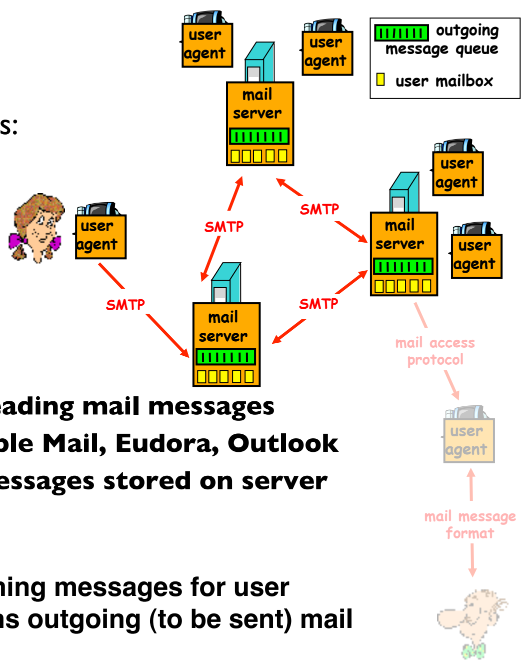
Peer-to-peer (p2p):

- hybrid of p2p and centralized server
- pure p2p
- hierarchical p2p
- end-host (p2p) multicast

Client-server:

- DNS
- FTP
- SMTP
- HTTP
  - cookies
  - web caching
  - CDN
- Multi-player games

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., Thunderbird, Apple Mail, Eudora, Outlook**
- **outgoing, incoming messages stored on server**

Mail servers :

- **mailbox contains incoming messages for user**
- **message queue contains outgoing (to be sent) mail messages**

Electronic Mail: SMTP (RFC 2821)

SMTP protocol runs between mail servers to send email messages

- **client: sending mail server**
- **server: receiving mail server**

Uses TCP to reliably transfer email message from client to server, port 25

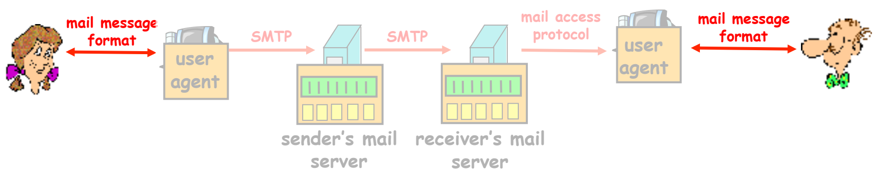
- **direct transfer: sending server to receiving server**
- **three phases of transfer**
  - **handshaking (greeting)**
  - **transfer of messages**
  - **closure**
- **command/response interaction**
  - **commands: ASCII text**
  - **response: status code and phrase**
- **messages must be in 7-bit ASCII**

Sample SMTP Interaction

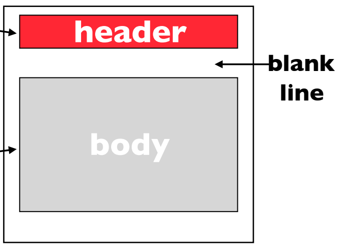
```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like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

Try it out! `% telnet crepes.fr 25`
(Real programmers send email by . . . not)

Mail Message Format (RFC 822)

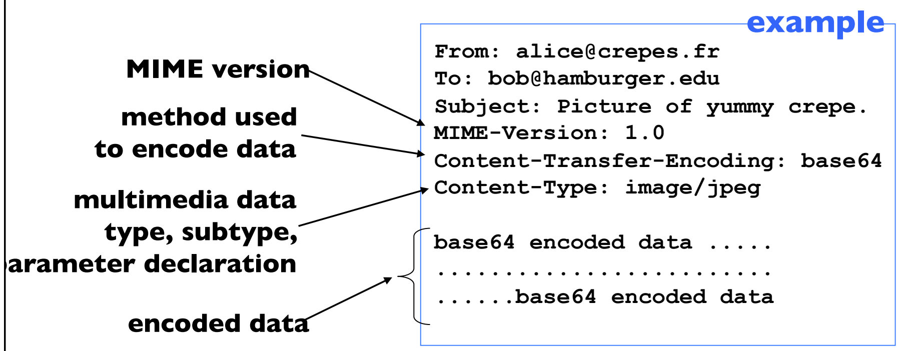


- SMTP: protocol for exchanging email messages
- RFC 822: standard for **text only** message format:
  - header lines, e.g.,
    - To:
    - From:
    - Subject:
    - different from SMTP commands!
  - body
    - the "message", ASCII characters only



Mail Message Format: MIME

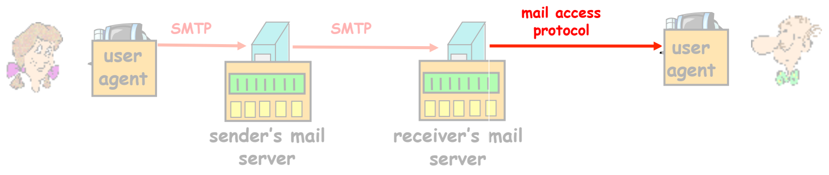
- MIME: MultIMEdia mail extension (RFC 2045, 2056)
- additional lines in msg header declare MIME content type


example
MIME version
method used to encode data
multimedia data type, subtype, arameter 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)
  - IMAP: Internet Mail Access Protocol (RFC 1730)
  - HTTP: Yahoo! Mail, Gmail, Hotmail, etc.

POP3 and IMAP

POP3

- simple authorization (agent ⇔ server) and download
- POP3 is stateless across sessions
- two modes:

1. "download and delete" mode
   - user cannot re-read e-mail if he changes client
2. "download-and-keep" mode
   - copies of messages on different clients

IMAP

- **more features (more complex)**
- **IMAP keeps user state across sessions**
- **all messages are kept at the server**
- **manipulation of stored messages on server**
- **allows user to organize messages in folders**
  - **names of folders and mappings between message IDs and folder name**

POP3 Dnload and Delete Example

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
```

Application Architectures

Peer-to-peer (p2p):

- hybrid of p2p and centralized server
- pure p2p
- hierarchical p2p
- end-host (p2p) multicast

Client-server:

- DNS
- FTP
- SMTP
- HTTP
  - cookies
  - web caching
  - CDN
- Multi-player games

Web and HTTP

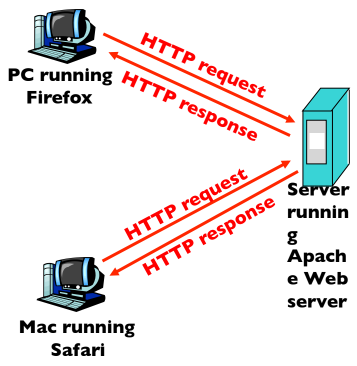
- A web page consists of objects
- An object can be an HTML file, a JPEG image, a Java applet, an audio file, a flash video …
- A web page comprises a base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

$$\underbrace{\texttt{http}}_{\textbf{protocol}}\texttt{://}\underbrace{\texttt{www.someschool.edu}}_{\textbf{host name}}\texttt{/}\underbrace{\texttt{someDept/pic.gif}}_{\textbf{path name}}$$

HTTP Overview

HTTP: HyperText Transfer Protocol

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



HTTP Overview

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!

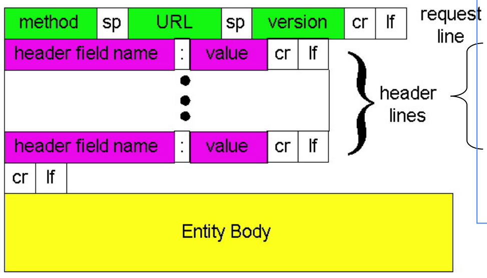
- **past history (state) must be maintained**
- **if server/client crashes, their views of "state" may be inconsistent, must be reconciled**

HTTP Request Message

Two types of HTTP messages: request, response

HTTP request message:

- in ASCII (human-readable format)
- general format:



example

```
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)

Carriage return, line feed indicates end of message

Method Types (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**

Uploading form input alternatives

1. `POST` **method:**
   - **web pages often include form input**
   - **input is uploaded to server in entity body**
2. **as parameter to `GET` URL method:**
   - **input is uploaded in URL field of request line:**
     $\texttt{www.somesite.com/animalsearch}\underbrace{\texttt{?monkeys\&banana}}_{\textbf{input parameters}}$

HTTP Response Message

first line: status line (protocol status code, status phrase)

header lines

example

```
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

data data data data data ...
```

data, e.g
request
HTML fi

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`