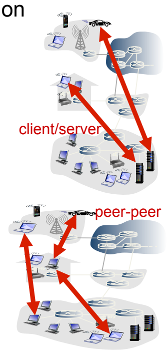
Application Layer

- Application layer protocols define
  - what messages are exchanged
  - the **syntax** of the messages
  - the **semantics** of the messages
  - **how/when** to exchange messages
  - note the **interaction** is defined, not the application itself!
- We will examine four network applications and their protocols
  - e-mail (SMTP - simple mail transport protocol)
  - the Web (HTTP - HyperText Transfer Protocol)
  - Domain Name Service (DNS)
  - P2P file sharing
- But first we' ll look at current application architectures...

Reading:
Kurose & Ross: Ch 2

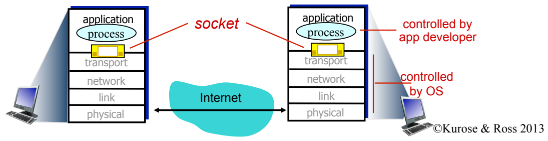
Models of Interaction

- Client - Server
  - central storage of information in always on server
  - distinction between client which receives service and server which provides service
  - note that it is possible for a host to act as both a client and as a server in different interactions.
  - Web, e-mail, FTP
- Peer to Peer
  - distributed storage of information
  - no clear distinction between clients and servers. Hosts share typically equal control of processing and data
  - Peers dynamically join and leave
  - Bit Torrent



How processes communicate

- **Sockets** provide the application programmers' interface (API) between a **process** and the transport layer.
- User application code runs on end-systems - not network core
- The application programmer needs to specify
  - which transport protocol to use
  - what host to send messages to (e.g. IP address or hostname)
  - what process on the destination host to send messages to (port number)



©Kurose & Ross 2013

Internet Transport Services

- What services do applications need?
  - Reliable data transfer
  - Minimum throughput guarantees
  - Bounded delays
  - Security
- What do the Internet protocols provide?
  - Reliable data transfer with transmission control protocol TCP
  - Minimal overhead, available bandwidth/delays, no delivery guarantee with user datagram protocol UDP
  - emerging protocols for providing timing and bandwidth guarantees
- Current choices in Internet are TCP or UDP. How does a network application designer decide?

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100' s msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100' s msec
text messaging	no loss	elastic	yes and no

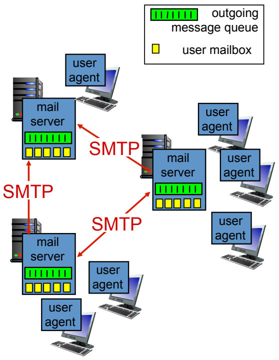
Electronic mail

Three major components:

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

User Agent

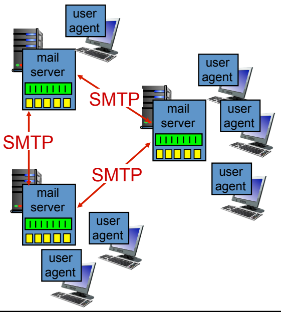
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server. TCP port 25



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
```

handshaking

msg

close

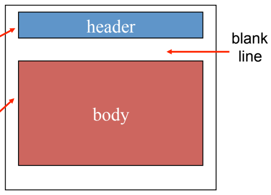
Mail message format

SMTP: protocol for exchanging email msgs

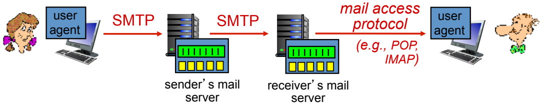
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



Mail access protocols

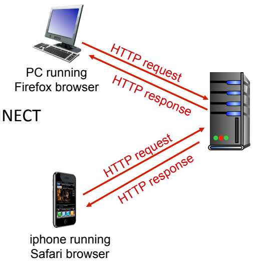


- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo!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

The World Wide Web

HTTP 1.0 - RFC 1945
HTTP 1.1 - RFC 2616

- Client-Server
- requests
  - GET (get a URL)
  - HEAD (meta data)
  - POST (send data to server)
  - PUT, OPTIONS,DELETE,TRACE,CONNECT
- responses
  - "nnn message <data>"
  - 1xx Informational
  - 2xx Success
  - 3xx Redirection
  - 4xx Client Error
  - 5xx Server Error



HTTP request format

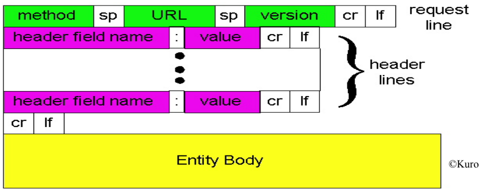
```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

request line (GET, etc commands)

header lines

Carriage return, line feed indicates end of message

(extra carriage return, line feed)



©Kurose & Ross 2001

HTTP response format

status line (protocol status code status phrase)

header lines

data, e.g., requested html file

```
HTTP/1.0 200 OK
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 ...
```

version | sp | status code | sp | phrase | cr | lf

header field name | : | value | cr | lf

⋮

header field name | : | value | cr | lf

cr | lf

Entity Body

Trying out network applications (client side) for yourself

1. Telnet to your favorite server (web server in this example):

```
telnet www.cs.adelaide.edu.au 80
```

Opens TCP connection to port 80 (default HTTP server port) at www.cs.adelaide.edu.au
Anything typed in sent to port 80 at www.cs.adelaide.edu.au

2. Type in the protocol request (a GET HTTP request):

```
GET /index.html HTTP/1.1
Host: www.cs.adelaide.edu.au
```

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 server!

You can do the same with a mail server on port 25.

Persistent HTTP

Nonpersistent HTTP issues:

- HTTP/1.0
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

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

Persistent ***without*** pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent ***with*** pipelining:

- default in HTTP/1.1
- Often not turned on in browsers
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

A few of the differences between HTTP 1.0 and 1.1

- HTTP 1.0
  - non-persistent connections
  - Methods
    - GET - get a web object
    - HEAD - get header information about web object
    - POST - send data to specified URL
  - Basic caching
- HTTP 1.1
  - both persistent and non-persistent connections
  - 1.0 methods plus
    - PUT - store data as URL
    - DELETE
    - TRACE - allows client to see what was received by server
    - CONNECT - reserved for tunneling
  - additional header fields
    - HOST - identifies host and port number where web object is located
  - reliable caching

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

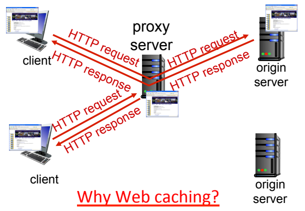
comparison with HTTP:

- 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

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
- 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)