

Application Layer Overview and Web/HTTP

2: Application Layer 1

Some network apps

- ❑ E-mail
- ❑ Web
- ❑ Instant messaging
- ❑ Remote login
- ❑ P2P file sharing
- ❑ Multi-user network games
- ❑ Streaming stored video clips
- ❑ Internet telephone
- ❑ Real-time video conference
- ❑ Massive parallel computing

2: Application Layer 2

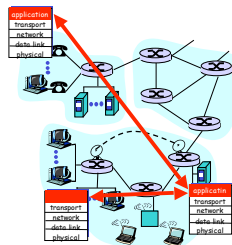
Creating a network app

Write programs that

- run on different end systems and
- communicate over a network.
- e.g., Web: Web server software communicates with browser software

No software written for devices in network core

- Network core devices do not function at app layer
- This design allows for rapid app development



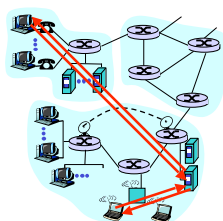
2: Application Layer 3

Application architectures

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

2: Application Layer 4

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

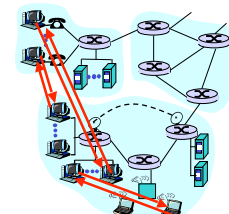
Examples?

2: Application Layer 5

Pure P2P architecture

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

Highly scalable
But difficult to manage



2: Application Layer 6

Hybrid of client-server and P2P

Napster

- File transfer P2P
- File search centralized:
 - Peers register content at central server
 - Peers query same central server to locate content

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
 - User registers its IP address with central server when it comes online
 - User contacts central server to find IP addresses of buddies

2: Application Layer 7

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**

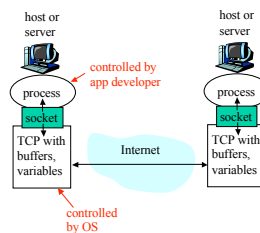
Client process: process that initiates communication
Server process: process that waits to be contacted

- Note: applications with P2P architectures have client processes & server processes

2: Application Layer 8

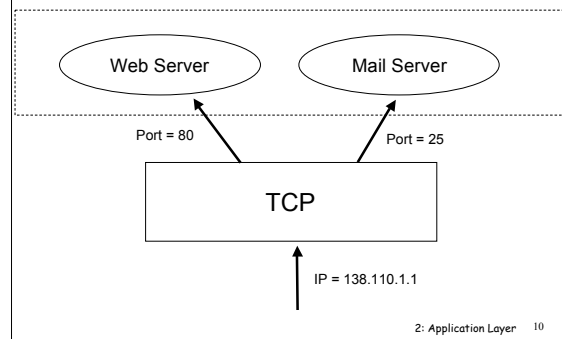
Sockets

- 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)



2: Application Layer 9

Port Numbers



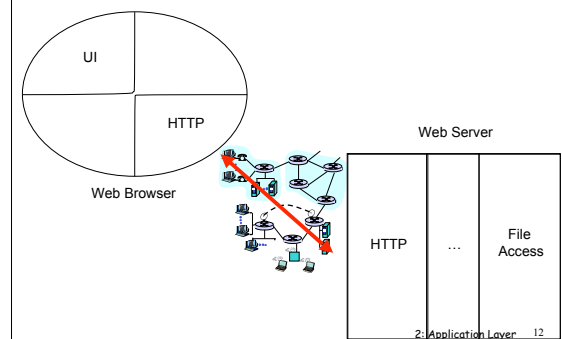
2: Application Layer 10

App-layer protocol defines

- Types of messages exchanged, eg, request & response messages
 - Syntax of message types: what fields in messages & how fields are delineated
 - Semantics of the fields, ie, meaning of information in fields
 - Rules for when and how processes send & respond to messages
- Public-domain protocols:**
- defined in RFCs
 - allows for interoperability
 - eg, HTTP, SMTP
- Proprietary protocols:**
- eg, KaZaA

2: Application Layer 11

Applications and App-Layer Protocols



2: Application Layer 12

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"

Bandwidth

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

2: Application Layer 13

Transport service requirements of common apps

Application	Data loss	Bandwidth	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
instant messaging	no loss	elastic	yes and no

2: Application Layer 14

Internet transport protocols services

TCP service:

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

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: why bother? Why is there a UDP?

2: Application Layer 15

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

2: Application Layer 16

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

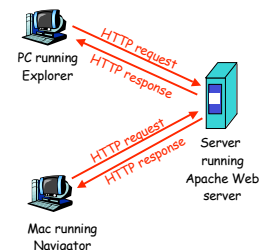
host name
path name

2: Application Layer 17

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 1.0: RFC 1945
- HTTP 1.1: RFC 2068



2: Application Layer 18

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"

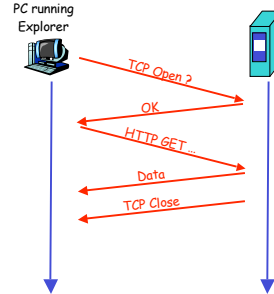
- server maintains no information about past client requests

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

2: Application Layer 19

HTTP



2: Application Layer 20

HTTP connections

Nonpersistent HTTP

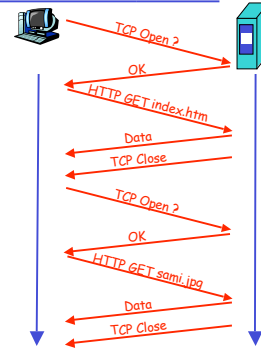
- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP

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

2: Application Layer 21

Nonpersistent HTTP



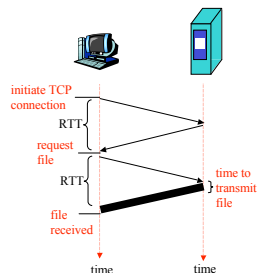
2: Application Layer 22

Response time modeling

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

Response time:

- one RTT to initiate TCP connection
 - one RTT for HTTP request and first few bytes of HTTP response to return
 - file transmission time
- total = 2RTT + transmit time**



2: Application Layer 23

Persistent HTTP

Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- but 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 are sent over connection

Persistent without pipelining:

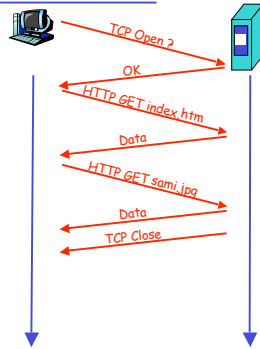
- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent with pipelining:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

2: Application Layer 24

Persistent HTTP



2: Application Layer 25

HTTP request message

- two types of HTTP messages: *request*, *response*

- HTTP request message:

- ASCII (human-readable format)

request line
(GET, POST, HEAD commands)

```
GET /somedir/page.html HTTP/1.1
```

header lines

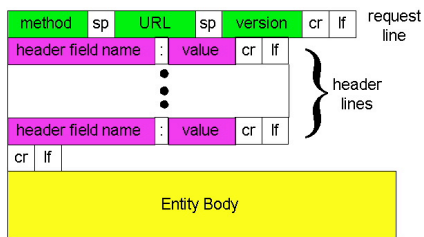
```
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

Carriage return, line feed
indicates end of message

(extra carriage return, line feed)

2: Application Layer 26

HTTP request message: general format



2: Application Layer 27

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

2: Application Layer 28

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

2: Application Layer 29

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 ...
```

2: Application Layer 30

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

2: Application Layer 31

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!

2: Application Layer 32

User-server state: cookies

Many major Web sites use cookies

Four components:

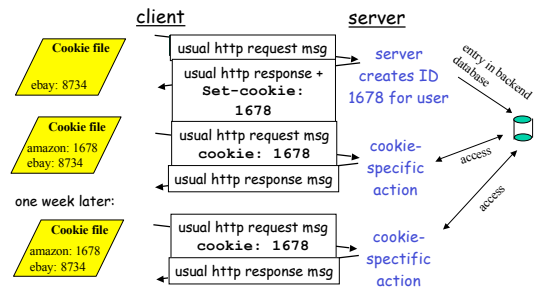
- 1) cookie header line in the HTTP response message
- 2) cookie header line in HTTP request message
- 3) cookie file kept on user's host and managed by user's browser
- 4) back-end database at Web site

Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

2: Application Layer 33

Cookies: keeping "state" (cont.)



2: Application Layer 34

Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Cookies and privacy:

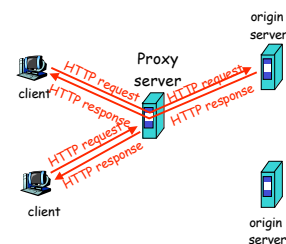
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

2: Application Layer 35

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



2: Application Layer 36

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)

2: Application Layer 37

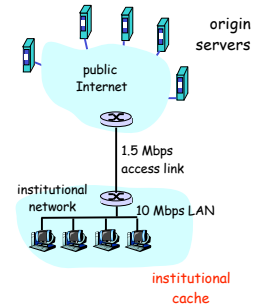
Caching example

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



2: Application Layer 38

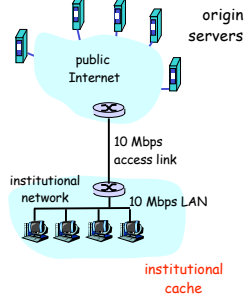
Caching example (cont)

Possible solution

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

Consequences

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay = 2 sec + msec + msec
- often a costly upgrade



2: Application Layer 39

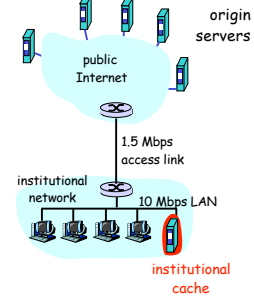
Caching example (cont)

Install cache

- suppose hit rate is .4

Consequence

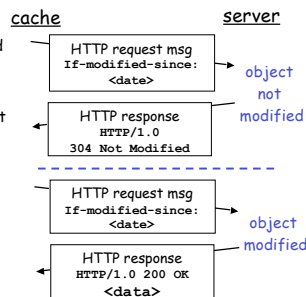
- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = .6*(2.01) secs + milliseconds < 1.4 secs



2: Application Layer 40

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



2: Application Layer 41