Lecture 2: Application Layer Overview

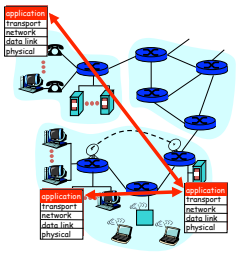
goals:

- conceptual + implementation aspects of network application protocols
  - client server paradigm
  - service models
- learn about protocols by examining popular application-level protocols
- specific protocols:
  - http
  - Networked Storage
- programming network applications
  - Bluetooth/ZigBee Applications

Applications and application-layer protocols

Application: communicating, distributed processes
- running in network hosts in "user space"
- exchange messages to implement app
- e.g., email, file transfer, the Web

Application-layer protocols
- one "piece" of an app
- define messages exchanged by apps and actions taken
- user services provided by lower layer protocols



Network applications: some jargon

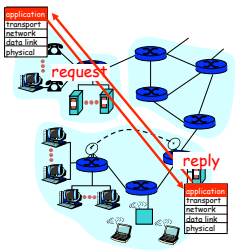
- A process is a program that is running within a host.
- Within the same host, two processes communicate with interprocess communication defined by the OS.
- Processes running in different hosts communicate with an application-layer protocol
- A user agent is an interface between the user and the network application.
  - Web:browser
  - E-mail: mail reader
  - streaming audio/video: media player

Client-server paradigm

Typical network app has two pieces: *client* and *server*

Client:
- initiates contact with server ("speaks first")
- typically requests service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader

Server:
- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail



Application-layer protocols (cont).

API: application programming interface
- defines interface between application and transport layer
- socket: Internet API
  - two processes communicate by sending data into socket, reading data out of socket

Q: how does a process "identify" the other process with which it wants to communicate?
- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered

… lots more on this later.

Services provided by Internet transport protocols

TCP service:
- *connection-oriented:* setup required between client, server
- *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?

The Web: some jargon

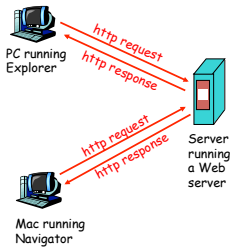
- Web page:
  - consists of "objects"
  - addressed by a URL
- Most Web pages consist of:
  - base HTML page, and
  - several referenced objects.
- URL has two components: host name and path name:
- User agent for Web is called a browser:
  - MS Internet Explorer
  - Netscape Communicator
- Server for Web is called Web server:
  - Apache (public domain)
  - MS Internet Information Server

`www.someSchool.edu/someDept/pic.gif`

The Web: the http protocol

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
- http1.0: RFC 1945
- http1.1: RFC 2616



The http protocol: more

http: TCP transport service:

- 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 example

Suppose user enters URL www.someSchool.edu/someDepartment/home.index (contains text, references to 10 jpeg images)

1a. http client initiates TCP connection to http server (process) at www.someSchool.edu. Port 80 is default for http server.

1b. http server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. http client sends http *request message* (containing URL) into TCP connection socket

3. http server receives request message, forms *response message* containing requested object (someDepartment/home.index), sends message into socket

time

http example (cont.)

4. http server closes TCP connection.

5. http client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

time

Non-persistent and persistent connections

Non-persistent

- HTTP/1.0
- server parses request, responds, and closes TCP connection
- 2 RTTs to fetch each object
- Each object transfer suffers from slow start

Persistent

- default for HTTP/1.1
- on same TCP connection: server, parses request, responds, parses new request,..
- Client sends requests for all referenced objects as soon as it receives base HTML.
- Fewer RTTs and less slow start.

But most 1.0 browsers use parallel TCP connections.

http message format: request

- two types of http messages: *request, response*
- http request message:
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

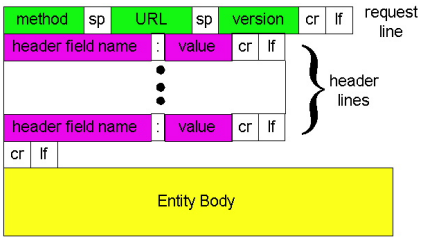
header lines

Carriage return, line feed indicates end of message

```
GET /somedir/page.html HTTP/1.0
Host: www.ele.uri.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr

(extra carriage return, line feed)
```


http request message: general format



http message format: respone

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 ...
```


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 www.eurecom.fr 80
```

Opens TCP connection to port 80 (default http server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

2. Type in a GET http request:

```
GET /~ross/index.html HTTP/1.0
```

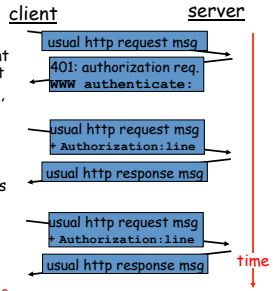
By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server

3. Look at response message sent by http server!

User-server interaction: authentication

Authentication goal: control access to server documents

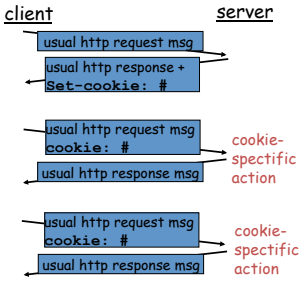
- stateless: client must present authorization in each request
- authorization: typically name, password
  - `authorization:` header line in request
  - if no authorization presented, server refuses access, sends `WWW authenticate:` header line in response



Browser caches name & password so that user does not have to repeatedly enter it.

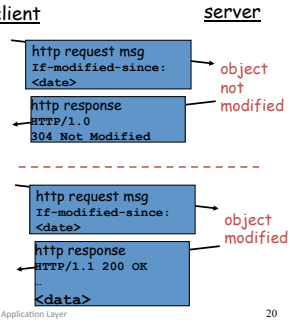
User-server interaction: cookies

- server sends "cookie" to client in response mst
  `Set-cookie: 1678453`
- client presents cookie in later requests
  `cookie: 1678453`
- server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices



User-server interaction: conditional GET

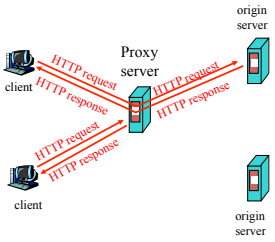
- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request
  `If-modified-since: <date>`
- server: response contains no object if cached copy up-to-date:
  `HTTP/1.0 304 Not Modified`



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



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

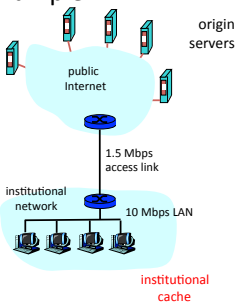
Assumptions

- 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

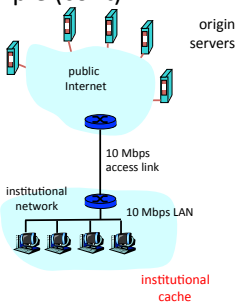
- increase bandwidth of access link to, say, 10 Mbps

Consequences

- 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)

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