Module 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 Tracking you on the web

Network applications:
- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- ...
- ...

Creating a network app

write programs that:
- run on (different) end systems
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices
- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation
Application architectures

possible structure of applications:
- client-server
- peer-to-peer (P2P)

Client-server architecture

server:
- always-on host
- permanent IP address
- data centers for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - self scalability – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management

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

**clients, servers**

**client process:** process that initiates communication

**server process:** process that waits to be contacted

aside: applications with P2P architectures have client processes & server processes
**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 the other side of door to deliver message to socket at receiving process.

**Addressing processes**

- To receive messages, process must have **identifier**.
- Host device has unique 32-bit IP address.
- **Q:** Does IP address of host on which process runs suffice for identifying the process?
  - **A:** No, many processes can be running on same host.
- **Identifier** includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- To send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80
- More shortly…
App-layer protocol defines

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:
- defined in RFCs
- allows for interoperability
  - e.g., HTTP, SMTP
proprietary protocols:
- e.g., Skype

What transport service does an app need?

data integrity
- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

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

security
- encryption, data integrity, ...

timing
- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”
Transport service requirements: common apps

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

Internet transport protocols services

TCP service:
- 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 throughput guarantee, security
- connection-oriented: setup required between client and server processes

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

Q: why bother? Why is there a UDP?
Internet apps: application, transport protocols

<table>
<thead>
<tr>
<th>Application</th>
<th>Application layer protocol</th>
<th>Underlying transport protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-mail</td>
<td>SMTP [RFC 2821]</td>
<td>TCP</td>
</tr>
<tr>
<td>remote terminal access</td>
<td>Telnet [RFC 854]</td>
<td>TCP</td>
</tr>
<tr>
<td>Web</td>
<td>HTTP [RFC 2616]</td>
<td>TCP</td>
</tr>
<tr>
<td>file transfer</td>
<td>FTP [RFC 959]</td>
<td>TCP</td>
</tr>
<tr>
<td>streaming multimedia</td>
<td>HTTP (e.g., YouTube), RTP [RFC 1889]</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>SIP, RTP, proprietary (e.g., Skype)</td>
<td>TCP or UDP</td>
</tr>
</tbody>
</table>

Securing TCP

TCP & UDP
- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL
- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer
- Apps use SSL libraries, which “talk” to TCP
SSL socket API
- cleartext passwds sent into socket traverse Internet encrypted
Module 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 Tracking you on the web

Web and HTTP

First, a review…
- 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, e.g.,

```url
www.someschool.edu/someDept/pic.gif
```

| host name | path name |
HTTP overview

HTTP: hypertext transfer protocol
- Web’s application layer protocol
- client/server model
  - client: browser that requests, receives, (using HTTP protocol) and “displays” Web objects
  - server: Web server sends (using HTTP protocol) objects in response to requests

... (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
What does a web page look like?

- web page content is formatted in HTML (hypertext markup language)
  - text, images, links to other pages
- try: http://gaia.cs.umass.edu/cs290/example.html

```html
<html>
<p>
This image is stored on this server:
<img src="shutterstock_90734176.jpg" height="80" width="80">
</p>

<p>
This image is stored on the NY Times web server:
<img src="http://i1.nyt.com/images/misc/nytlogo379x64.gif">
</p>
</html>
```
Retrieving/displaying a web page

HTTP connections

*non-persistent HTTP*
- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

*persistent HTTP*
- multiple objects can be sent over single TCP connection between client, server
HTTP request message

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

```
GET /index.html HTTP/1.1
Host: www-net.cs.umass.edu
User-Agent: Firefox/3.6.10
Accept: text/html,application/xhtml+xml
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1, utf-8; q=0.7
Keep-Alive: 115
Connection: keep-alive

```

HTTP GET request message:

```
GET sp URL sp version cr if
header field name | value cr if

header field name | value cr if

cr if

entity body
```

Application Layer 2-23
HTTP response message

status line (protocol status code status phrase)
HTTP/1.1 200 OK
Date: Sun, 26 Sep 2010 20:09:20 GMT
Server: Apache/2.0.52 (CentOS)
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT
ETag: "17dc6-a5c-bf716880"
Accept-Ranges: bytes
Content-Length: 2652
Keep-Alive: timeout=10, max=100
Connection: Keep-Alive
Content-Type: text/html;
charset=ISO-8859-1

header lines

data, e.g., requested HTML file
data data data data data ...

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

  200 OK
  - request succeeded, requested object later in this msg

  301 Moved Permanently
  - requested object moved, new location specified later in this msg
    (Location:)

  400 Bad Request
  - request msg 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 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!
   (or use Wireshark to look at captured HTTP request/response)

User-server state: cookies

many Web sites use cookies  
four components:  

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

example:

- Susan always access Internet  
  from PC  
- visits specific e-commerce  
  site for first time  
- when initial HTTP requests  
  arrives at site, site creates:  
  - unique ID  
  - entry in backend  
    database for ID
Cookies: keeping “state” (cont.)

Cookies (continued)

what cookies can be used for:
- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

how to keep “state”:
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

aside
cookies and privacy:
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
Conditional GET

- **Goal**: don’t send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- **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

- object not modified before `<date>`
- object modified after `<date>`

HTTP request msg
If-modified-since: `<date>`

HTTP response
HTTP/1.0
304 Not Modified

HTTP request msg
If-modified-since: `<date>`

HTTP response
HTTP/1.0 200 OK
`<data>`