Announcements/Reminders

• Homework 1 posted.
  – Due January 29th.
  – Submission via eCommons.
Last Class…

• Layers.
• Internet protocol stack.
• Encapsulation.
• De-encapsulation.
• Network performance:
  • Delay.
  • Loss.
  • Throughput.

• Brief Internet history.
• Hierarchical Internet structure.
Losses

• Due to:
  – Congestion.
    • Queue overflow.
  – Transmission errors.
Delay or Latency

\[ d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}} \]
Throughput

- **throughput**: rate (bits/time unit) at which bits transferred between sender/receiver
  - *instantaneous*: rate at given point in time
  - *average*: rate over longer period of time
Application Layer
Application layer

• Application types
• Application layer examples
  – HTTP (Hypertext Transfer Protocol)
  – FTP (File Transfer Protocol)
  – SMTP (Simple Message Transfer Protocol)
• The Domain Name System (DNS)
Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P
Client-server architecture
What is a server?

- A process that listens for and responds to requests from clients by providing a certain service.
Client-server architecture

server:
- always-on host
- permanent IP address
- server farms for scaling

Clients:
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
P2P architecture

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

Highly scalable but difficult to manage
Hybrid of client-server and P2P

Skype

- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)
Hybrid of client-server and P2P

Instant messaging
- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies
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 based on P2P architectures typically have client & server processes running on same machine.
Application layer operation

• Network applications are usually composed of:
  – Application layer protocol.
  – User interface (application program).
Application layer operation

• Users typically access application layer protocol through an application program (typically with a GUI)
  – Web browsers use HTTP, and may use others such as FTP
  – Mail applications use SMTP
  – Remote login applications use Telnet or SSH (Secure Shell)
Application layer operation

• Application Layer uses the services of a Transport Protocol:
  – TCP (Transmission Control Protocol)
  – UDP (User Datagram Protocol)
• These have similar interfaces to the Application Layer, but
  – TCP provides a reliable delivery service
  – UDP is unreliable
• Transport Protocols are usually implemented within the Operating System
  – Windows, Linux, etc. all implement “protocol stacks”
• Application Layer is implemented in an application program (e.g., the Firefox browser).
• Operating system vendors provide interfaces (called Application Program Interfaces or APIs) to use the Transport Layer services).
Application process

• **Application Process:** Application program running within a host.
  – Application processes in different hosts communicate by exchanging messages
  – Messages sent using Transport Protocol (TCP or UDP)

• **Question:** How does an Application Process determine the address of a remote application process it wants to communicate with?
Addressing Applications

• Needs two components of address:
  – IP address identifies the remote host
  – “Port” number identifies the application on the remote host

• Similar to a zip code and house number
TCP and UDP ports

• A “port” is like a door to enter the transport layer.
  – Some ports are designated to be “listening ports” for standard applications.
    • Example: Port 80 is attached to HTTP, and is the listening port for a Web server application
    • Much like the phone numbers 411, 911, etc.
  – Clients may pick any available port number to start communication
  – An application-layer conversation is defined by 4 parameters:
    • Source IP address
    • Destination IP address
    • Source port number
    • Destination port number
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

![Diagram of sockets and network communication](image-url)

- controlled by OS
- controlled by app developer

CMPE 150 Winter’13 24
Well-known ports

• All common applications have pre-assigned listening ports:
  – Range between 0 and 1023
  – HTTP: Port 80
  – FTP: Port 21
  – Telnet: Port 23
  – SMTP: Port 25

• Complete list of well-known ports at Wikipedia.
Addressing Example

• An HTTP request from a UCSC host to Yahoo:
  – Destination IP address: 209.131.32.145 (host within yahoo.com)
  – Destination port number: 80
  – Source IP address: 128.114.48.62 (sundance.cse.ucsc.edu)
  – Source TCP port number: 49152 (selected from a list of available port)
Application-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

Public-domain protocols:
- defined in RFCs
- allows for interoperability
  - e.g., HTTP, SMTP, BitTorrent

Proprietary protocols:
- e.g., Skype
### 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 (eg 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>typically UDP</td>
</tr>
</tbody>
</table>
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
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

CMPE 150 Winter’13
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

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 connections

Nonpersistent HTTP
• At most one object is sent over a TCP connection.

Persistent HTTP
• Multiple objects can be sent over single TCP connection between client and server.
Nonpersistent HTTP

Suppose user enters URL www.someSchool.edu/someDepartment/home.index

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

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. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms response message containing requested object, and sends message into its socket
Nonpersistent HTTP (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
Non-Persistent HTTP: Response time

Definition of RTT: time for 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
Persistent HTTP

Nonpersistent HTTP issues:
• 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
• client sends requests as soon as it encounters a referenced object
• as little as one RTT for all the referenced objects
HTTP request message

- two types of HTTP messages: request, response
- HTTP request message:
  - ASCII (human-readable format)
    - 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/1.0**
- 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
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

data, e.g., requested HTML file

data data data data data data data ...

CMPE 150 Winter’13 39
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 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!
User-server state: cookies

Many major Web sites use cookies

Four components:
1) cookie header line of HTTP response message
2) cookie header line in 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 always 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.)

Client

- eBay 8734
- Cookie file
- Amazon 8734

Cookie file

One week later:

- eBay 8734
- Amazon 1678

Server

- Amazon server creates ID 1678 for user
- Create entry

- Backend database

Usual HTTP request msg

Usual HTTP response msg

Set-cookie: 1678

Usual HTTP request msg

Cookie: 1678

Usual HTTP response msg

Cookie: 1678

Usual HTTP response msg
Cookies (continued)

What cookies can bring:
• 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
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 = 1,000,000 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
possible solution
• increase bandwidth of access link to, say, 100 Mbps

consequence
• 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
possible solution: install cache
• suppose hit rate is 0.4

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 + .4*milliseconds < 1.4 secs
Conditional GET

- **Goal:** don’t send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request
  
  \[
  \text{If-modified-since: } <\text{date}> 
  \]

- server: response contains no object if cached copy is up-to-date:

\[
\text{HTTP/1.0 304 Not Modified}
\]

- server: response contains object if cached copy is modified:

\[
\text{HTTP/1.0 200 OK}
\]

\[
<\text{data}> 
\]