Chapter 2: Application Layer

A note on the use of these ppt slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

♦ If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we’d like people to use our book!)
♦ If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2010
J.F Kurose and K.W. Ross, All Rights Reserved

Chapter 2: Application layer

2.1 Principles of network applications
2.2 Web and HTTP
2.3 FTP
2.4 Electronic Mail
   ▪ SMTP, POP3, IMAP
2.5 DNS
2.6 P2P applications
2.7 Socket programming with TCP
2.8 Socket programming with UDP
Chapter 2: Application Layer

Our goals:
- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- programming network applications
  - socket API

Some network apps
- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube)
- voice over IP
- real-time video conferencing
- cloud computing
- ...
- ...
- ...
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

Chapter 2: Application layer

2.1 Principles of network applications
2.2 Web and HTTP
2.3 FTP
2.4 Electronic Mail
   SMTP, POP3, IMAP
2.5 DNS
2.6 P2P applications
2.7 Socket programming with TCP
2.8 Socket programming with UDP
Application architectures

- client-server
- peer-to-peer (P2P)
- hybrid of client-server and P2P

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
Pure 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)

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

**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 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)
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

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

proprietary protocols:
- e.g., Skype

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”

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, ...
Transport service requirements of 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 msec</td>
</tr>
<tr>
<td>stored audio/video</td>
<td>loss-tolerant</td>
<td>video:10kbps-5Mbps</td>
<td></td>
</tr>
<tr>
<td>interactive games</td>
<td>loss-tolerant</td>
<td>same as above</td>
<td>yes, few secs</td>
</tr>
<tr>
<td>instant messaging</td>
<td>no loss</td>
<td>elastic</td>
<td>yes, 100’s msec</td>
</tr>
</tbody>
</table>

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 throughput guarantees, security

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

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>typically UDP</td>
</tr>
</tbody>
</table>

---

## Chapter 2: Application layer

### 2.1 Principles of network applications
- app architectures
- app requirements

### 2.2 Web and HTTP

### 2.3 FTP

### 2.4 Electronic Mail
- SMTP, POP3, IMAP

### 2.5 DNS

### 2.6 P2P applications

### 2.7 Socket programming with TCP

### 2.8 Socket programming with UDP
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
- example URL:
  
  www.someschool.edu/someDept/pic.gif

Where did this started?

- As We May Think, The Atl. monthly V. Bush, 1945 (reaction to Hiroshima)
- Memex (hypertext)
- "Wholly new forms of encyclopedias will appear, ready-made with a mesh of associative trails running through them, ready to be dropped into the memex and there amplified."
Other origins of the web

- Serbian origin, inventor of AC
- N. Tesla, wireless communication
- L. Kleinrock “A Brief History of the Internet and its Dynamic Future”

- And more humourous ones:
- D. Sedaris, nutcracker.com, in *Me Talk Pretty One Day*

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 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

protostats 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

non-persistent HTTP
- at most one object sent over TCP connection.

persistent HTTP
- multiple objects can be sent over single TCP connection between client, 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

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

non-persistent 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 /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 request message: general format
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

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\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS)\r\nLast-Modified: Tue, 30 Oct 2007 17:00:02 GMT \r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-1\r\n\r\ndata data data data data data ...

HTTP response status codes

- status code appears in 1st line in server->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!)