Chapter 2
Application Layer

A note on the use of these Powerpoint slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you see the animations; and 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) that you mention their source (after all, we’d like people to use our book!)
- If you post any slides 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 1996-2016
J.F Kurose and K.W. Ross, All Rights Reserved
CS 280 Lecture 3: Application Layer, Web and HTTP

John Magee
19 September 2016

Most slides adapted from Kurose and Ross, Computer Networking 7/e
Source material copyright 1996-2016
J.F Kurose and K.W. Ross
Chapter 2: outline

Today:
2.1 principles of network applications
2.2 Web and HTTP

Next Class:
2.3 electronic mail
   • SMTP, POP3, IMAP
2.4 DNS

2.5 P2P applications
2.6 video streaming and content distribution networks
2.7 socket programming with UDP and TCP
Chapter 2: application layer

our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
  - content distribution networks

- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS

- creating network applications
  - socket API
Some network apps

- 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 other side of door to deliver message to socket at receiving process

---

**Diagram:**

- **Application process** (controlled by app developer)
  - transport
  - network
  - link
  - physical

- **Internet**

- **Application process** (controlled by OS)
  - transport
  - network
  - link
  - physical
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

**timing**
- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

**security**
- encryption, data integrity, …
## 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></td>
</tr>
<tr>
<td>interactive games</td>
<td>loss-tolerant</td>
<td>few kbps up</td>
<td>yes, few secs</td>
</tr>
<tr>
<td>text messaging</td>
<td>no loss</td>
<td>elastic</td>
<td>yes, 100’s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>msec</td>
</tr>
<tr>
<td></td>
<td></td>
<td></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, that “talk” to TCP

SSL socket API

- cleartext passwords sent into socket traverse Internet encrypted
- see Chapter 8
Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
   • SMTP, POP3, IMAP
2.4 DNS
2.5 P2P applications
2.6 video streaming and content distribution networks
2.7 socket programming with UDP and TCP
Internet or WWW?

The Internet is like hardware...

The World Wide Web is like software...

The Internet is a prerequisite for the World Wide Web.
The World Wide Web

A system of interlinked hypertext documents and other resources (e.g. images) accessed via the Internet.

The WWW was conceived of and first implemented by Tim Berners-Lee, circa 1989-1991.
The First Web Page

World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give univers

Everything there is online about W3 is linked directly or indirectly to this document, including an executive Questions.

What's out there?
Pointers to the world's online information, subjects, W3 servers, etc.

Help
on the browser you are using

Software Products
A list of W3 project components and their current state. (e.g. Line Mode, X11 Viola, NeXTStep, S

Technical
Details of protocols, formats, program internals etc

Bibliography
Paper documentation on W3 and references.

People
A list of some people involved in the project.

History

Web Browser History

- **Line Mode Browser** - Feb 1992. This was also brought to us by Berners Lee. It was the first browser to support multiple platforms.
- **Viola WWW Browser** released - March 1992. This is widely suggested to be the world's first popular browser. It brought with it a stylesheet and scripting language, long before JavaScript and CSS.
- **Mosaic Browser** released - Jan 5th 1993. Mosaic was really highly rated when it first came out. It was developed at University of Illinois.
- **Cello Browser** released - June 8th, 1993. This was the first browser available for Windows.
- **Netscape Navigator 1.1** released - March 1995. This was the first browser to introduce tables to HTML.
- **Opera 1.0** released - April 1995. This was originally a research project for a Norwegian telephone company. The browser is still available today and is currently at version 12.
- **Internet Explorer 1.0** released - August 1995. Microsoft decided to get in on the act when its Windows operating system '95 was released. This was the browser that ran exclusively on that.

Source: https://webdesign.tutsplus.com/articles/a-brief-history-of-the-world-wide-web--webdesign-8710  
Application Layer 2-23
Browser History

Welcome to NCSA Mosaic, an Internet information browser and World Wide Web client. NCSA Mosaic was developed at the National Center for Supercomputing Applications at the University of Illinois in Urbana-Champaign. NCSA Mosaic software is copyrighted by The Board of Trustees of the University of Illinois (UI), and ownership remains with the UI.
Displaying a WWW Page

Figure 16.1 A browser retrieving a Web page

How do you “visit a website”?
Displaying a WWW Page

- Browser decodes URL to parse out host name and document location.
- Browser makes network connection to server.
- Client requests resource, and waits for the server to respond (using the hypertext transfer protocol).
- Browser parses the response, requests any embedded data, and formats/displays output.
Protocol

A protocol is a standard way of doing something.

Hyper Text Transfer Protocol (HTTP) specifies how to request and deliver content (e.g. web pages).
Hyper Text Transfer Protocol

HTTP is a protocol which specifies requests and responses between clients and servers. It assumes/builds upon:

- The Internet exists/computer is connected
- Reliable transport of data
- Web servers are waiting to service clients

HTTP is not limited to web pages -- it can be used to transfer any kind of data.
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.,

  www.clarku.edu/~jmagee/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

PC running Firefox browser
server running Apache Web server

iPhone running Safari browser
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

**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
Non-persistent HTTP

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 on port 80

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, and sends message into its socket.
Non-persistent 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

RTT (definition): time for a small packet to travel from client to server and back

HTTP 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
- non-persistent HTTP response time = 2RTT + file transmission 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)
    - Request line
      - GET, POST, HEAD commands
    - Header lines
      - Carriage return, line feed at start of line indicates end of header lines

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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
HTTP request message: general format

- method
- sp
- URL
- sp
- version
- cr
- lf

- header field name
- value
- cr
- lf

- header field name
- value
- cr
- lf

- cr
- lf

- entity body
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
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

data data data data data data data data ...

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
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 or Netcat to your favorite Web server:

   telnet www.cs.clarku.edu 80
   or
   nc www.cs.clarku.edu 80

   opens TCP connection to port 80
   (default HTTP server port) at www.cs.clarku.edu.
   anything typed in sent
   to port 80 at www.cs.clarku.edu

2. type in a GET HTTP request:

   GET /~jmagee/test.html HTTP/1.1
   Host: www.cs.clarku.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!)
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.)

client

- ebay 8734
  - cookie file

server

- usual http request msg

Amazon server creates ID 1678 for user

- usual http response msg
  - set-cookie: 1678

backend database

- cookie: 1678

one week later:

- usual http request msg
  - cookie: 1678

- usual http response msg

ebay 8734
amazon 1678

- usual http request msg
  - cookie: 1678

- usual http response msg

ebay 8734
amazon 1678
Cookies (continued)

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

aside

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

how to keep “state”:
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state
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
  - server for original requesting client
  - client to origin 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 (so too does P2P file sharing)
Caching example:

**assumptions:**
- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

**consequences:**
- LAN utilization: 15%  **problem!**
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs
Caching example: fatter access link

assumptions:
- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:
- LAN utilization: 15%
- access link utilization: 99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs
  msecs

Cost: increased access link speed (not cheap!)
Caching example: install local cache

**assumptions:**
- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

**consequences:**
- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

*How to compute link utilization, delay?*

**Cost:** web cache (cheap!)
Calculating access link utilization, delay with cache:

- Suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin

- Access link utilization:
  - 60% of requests use access link

- Data rate to browsers over access link
  \[ = 0.6 \times 1.50 \text{ Mbps} = 0.9 \text{ Mbps} \]
  - Utilization = \( \frac{0.9}{1.54} = 0.58 \)

- Total delay
  \[ = 0.6 \times (\text{delay from origin servers}) + 0.4 \times (\text{delay when satisfied at cache}) \]
  \[ = 0.6 (2.01) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs} \]
  - Less than with 154 Mbps link (and cheaper too!)
**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`
Chapter 2: Summary

so far…

- application architectures
  - client-server
  - P2P
  - hybrid

- application service requirements:
  - reliability, bandwidth, delay

- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, Skype
Student To Dos

• **Readings:**
  • Kurose-Ross: 2.1 - 2.6 (This week)

• **Written HW2**
  • Due Tonight (9/19)
    - Ch. 1 Questions
    - Explore with Linux tools

• **Wireshark Lab 2 (HTTP)**
  • Due Next Monday (?)