Application Layer: Web & HTTP

Instructor: Carey Williamson
Office: ICT 740
Email: carey@cpsc.ucalgary.ca

Class Location: ICT 122
Lectures: MWF 12:00 - 12:50


Slides are adapted from the book's companion Web site, with changes by Anirban Mahanti and Carey Williamson.
Outline

Introduction to App-Layer Protocols
Brief History of WWW
Architecture
HTTP Connections
HTTP Format
Web Performance
Cookies
Network applications: some jargon

Process: program running within a host.
within same host, two processes communicate using inter-process communication (IPC, defined by OS).
processes running on different hosts communicate with an application-layer protocol

user agent: interfaces with user “above” and network “below”.
implements user interface & application-level protocol

Web: browser
E-mail: mail reader
streaming audio/video: media player
Applications and application-layer protocols

Application: communicating, distributed processes
- e.g., e-mail, Web, P2P file sharing, instant messaging
- running in end systems (hosts)
- exchange messages to implement application

Application-layer protocols
- one “piece” of an app
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)
App-layer protocol defines:

**Types** of messages exchanged (e.g., reqs & response messages)

**Syntax** of message types: what fields in messages & how fields are delineated

**Semantics** of the fields (i.e., the meaning of information in fields)

**Rules** for when and how processes send & respond to messages

Public-domain protocols:
- defined in RFCs (Requests for Comments)
- allows for interoperability
  - eg, HTTP, SMTP

Proprietary protocols:
- eg, KaZaA
Client-server paradigm

Typical network app has two pieces: client and server

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

Server:
- provides requested service to client
  - e.g., Web server sends requested Web page; mail server delivers e-mail
Processes communicating across network

process sends/receives messages to/from its socket

socket analogous to door

sending process shoves message out door

sending process assumes transport infrastructure on other side of door which brings message to socket at receiving process

API allows: (1) choice of transport protocol (TCP/UDP); (2) ability to set several parameters (e.g., MSS)

host or server

process

socket

TCP with buffers, variables

host or server

process

socket

TCP with buffers, variables

Internet

controlled by OS

controlled by app developer
Addressing processes:

For a process to receive messages, it must have an identifier. Every host has a unique 32-bit IP address.

**Q:** does the IP address of the host on which the process runs suffice for identifying the process?

**Answer:** No, many processes can be running on the same host.

Identifier includes both the IP address and **port numbers** associated with the process on the host.

Example port numbers:
- HTTP server: 80
- Mail server: 25

More on this later
What transport service does an app need?

Data loss

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

Timing

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

Bandwidth

most apps (“elastic apps”) make use of whatever bandwidth they get
other apps (e.g., multimedia) require minimum amount of bandwidth to be “effective”
## Transport service requirements of common apps

<table>
<thead>
<tr>
<th>Application</th>
<th>Data loss</th>
<th>Bandwidth</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></td>
<td></td>
<td>video:10kbps-5Mbps</td>
<td></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 msec</td>
</tr>
<tr>
<td>instant messaging</td>
<td>no loss</td>
<td>elastic</td>
<td>yes</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
- **not provided**: timing, minimum bandwidth guarantees

UDP service:

- unreliable data transfer between sending and receiving process
- not provided: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

**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>proprietary</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td></td>
<td>(e.g. RealNetworks)</td>
<td></td>
</tr>
<tr>
<td>Internet telephony</td>
<td>proprietary</td>
<td>typically UDP</td>
</tr>
<tr>
<td></td>
<td>(e.g., Dialpad, skype)</td>
<td></td>
</tr>
</tbody>
</table>
Outline

Introduction to App Layer Protocols
Brief History of WWW
Architecture
HTTP Connections
HTTP Format
Web Performance
Cookies
History of the Web


Tim Berners-Lee at CERN in 1991
  Demonstrated prototype at a conf. in ’91
  Text-based

Marc Andreessen developed the first graphical Web browser in 1993: Mosaic

Andreessen founds Netscape Communications

Browser war starts around 1995-96

America Online buys Netscape in 1998
Some “Web” Terminology

Web page may contain links to other pages (sometimes also called Web Objects)
Object can be HTML file, JPEG image, Java applet, audio file, ...
Web pages are “Hypertexts”
One page points to another
Proposed by Prof. Vannevar Bush in 1945!
Each object is addressable by a URL:

http://www.someschool.edu/someDept/pic.gif

protocol host name path name
Outline

Introduction to App Layer Protocols
Brief History of WWW
Architecture
HTTP Connections
HTTP Format
Web Performance
Cookies
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 1.0: RFC 1945
HTTP 1.1: RFC 2616
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
Outline

Introduction to App Layer Protocols
Brief History of WWW
Architecture
HTTP Connections
HTTP Format
Web Performance
Cookies
HTTP connections

Non-persistent HTTP

At most one object is sent over a TCP connection.

HTTP/1.0 uses non-persistent HTTP

Persistent HTTP

Multiple objects can be sent (one at a time) over a single TCP connection between client and server.

HTTP/1.1 uses persistent connections in default mode

Pipelined
Non-pipelined
Response time modeling

Definition of RTT: time to send 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 = 2*RTT + transmit time
http://www.somewhere.com/index.html

Persistent HTTP

Nonpersistent HTTP issues:
requires 2 RTTs per object
OS must work and allocate
host resources for each TCP
connection
but 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
are sent over connection

Persistent without pipelining:
client issues new request
only when previous
response has been received
one RTT for each
referenced object

Persistent with pipelining:
default in HTTP/1.1
client sends requests as
soon as it encounters a
referenced object
as little as one RTT for all
the referenced objects
Outline

Introduction to App Layer Protocols
Brief History of WWW
Architecture
HTTP Connections
HTTP Format
Web Performance
Cookies
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
```

(request line)

(GET, POST, HEAD commands)

(header lines)

Carriage return, line feed indicates end of message

(extra carriage return, line feed)
HTTP request message: general format

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

request line

- header field name**: value**
- **cr**
- **lf**

header lines

- header field name**: value**
- **cr**
- **lf**

Entity Body
HTTP Methods

GET: retrieve a file (95% of requests)
HEAD: just get meta-data (e.g., mod time)
POST: submitting a form to a server
PUT: store enclosed document as URI
DELETE: removed named resource
LINK/UNLINK: in 1.0, gone in 1.1
TRACE: http “echo” for debugging (added in 1.1)
CONNECT: used by proxies for tunneling (1.1)
OPTIONS: request for server/proxy options (1.1)
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 data data data data data ...

data, e.g., requested HTML file

header lines

status line
(protocol status code status phrase)
HTTP Response Status Codes

1XX: Informational (def’d in 1.0, used in 1.1)
   100 Continue, 101 Switching Protocols

2XX: Success
   200 OK, 206 Partial Content

3XX: Redirection
   301 Moved Permanently, 304 Not Modified

4XX: Client error
   400 Bad Request, 403 Forbidden, 404 Not Found

5XX: Server error
   500 Internal Server Error, 503 Service Unavailable, 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!
Outline

Introduction to App Layer Protocols
Brief History of WWW
Architecture
HTTP Connections
HTTP Format
Web Performance
Cookies
Web Proxy Caching

Objective: satisfy client request without involving origin server resulting in reduced server & network load, low latency to response

user sets browser: Web accesses via cache
browser sends all HTTP requests to cache
object in cache => cache hit: returns object
else cache requests object from origin server, then returns object to client

Cache acts as both client and server
Web Caching Hierarchy

- Client
- Local proxy cache (e.g., local ISP, University)
- Regional proxy cache
- National/international proxy cache
Why Cache?

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.
Some Issues

Not all objects can be cached
  E.g., dynamic objects, copyrighted material
Cache consistency
  strong
  weak
Cache Replacement Policies
  Variable size objects
  Varying cost of not finding an object (a “miss”) in the cache
Prefetch?
  A large fraction of the requests are one-timers
Weak Consistency

Each cached copy has a TTL beyond which it must be validated with the origin server.

\[
\text{TTL} = \text{freshness life time} - \text{age}
\]

freshness life time: often heuristically calculated; sometimes based on MAX_AGE or EXPIRES headers

age = current time (at client) - timestamp on object (time at which server generated response)

Age Penalty?
Conditional GET: client-side caching

**Goal:** don't send object if client has up-to-date cached version

client: 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

---

client

HTTP request msg

If-modified-since: <date>

server

HTTP response

HTTP/1.0
304 Not Modified

---

client

HTTP request msg

If-modified-since: <date>

server

HTTP response

HTTP/1.0
200 OK
<data>

---
Content distribution networks (CDNs)

The content providers are the CDN customers.

Content replication
CDN company installs hundreds of CDN servers throughout Internet in lower-tier ISPs, close to users.
CDN replicates its customers’ content in CDN servers. When provider updates content, CDN updates servers.
Cookies: keeping “state”

Many major Web sites use cookies

Four components:
1) cookie header line in the HTTP response message
2) cookie header line in HTTP request message
3) cookie file kept on user’s host and managed by user’s browser
4) back-end database at Web site

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

**Client**
- **Cookie file**
  - ebay: 8734

**Server**
- creates ID 1678 for user

**One week later:**
- **Cookie file**
  - amazon: 1678
ebay: 8734

**Diagram:**
- usual http request msg
  - usual http response +
    - Set-cookie: 1678
  - usual http request msg
    - cookie: 1678
  - cookie-specific action
  - cookie-specific action

**Diagram Elements:**
- entry in backend database
- access
- access

**Cookie File:**
- Amazon: 1678
- Ebay: 8734
Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state
  (Web e-mail)

Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

aside
The major application on the Internet
A large fraction of traffic is HTTP

Client/server model:
Clients make requests, servers respond to them
Done mostly in ASCII text (helps debugging!)

Various headers and commands

Web Caching & Performance

Content Distribution Networks