Application Layer

Goals:

- Conceptual aspects of network application protocols
  - Client server paradigm
  - Service models
- Review protocols by examining popular application-level protocols
  - HTTP
  - DNS
Applications and Application-Layer Protocols

Application: communicating, distributed processes
- Running in network hosts in “user space”
- Exchange messages to implement app
- E.g., email, file transfer, the Web

Application-layer protocols
- One “piece” of an app
- Define messages exchanged by apps and actions taken
- User services provided by lower layer protocols
Client-Server Paradigm

Typical network app has two pieces: **client** and **server**

**Client:**
- Initiates contact with server ("speaks first")
- Typically requests service from server,
- E.g., request WWW page, send email

**Server:**
- Provides requested service to client
- E.g., sends requested WWW page, receives/stores received email
Services Provided by Internet Transport Protocols

TCP service:
- **Connection-oriented**: setup required between client, server
- **Reliable transport** between sending and receiving process
- **Flow control**: sender won’t overwhelm receiver
- **Congestion control**: throttle sender when network overloaded
- **Does not providing**: timing, minimum bandwidth guarantees

UDP service:
- Unreliable data transfer between sending and receiving process
- Does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee
The HTTP Protocol

HTTP: Hypertext transfer protocol

- Application layer protocol for the Web
- Client/server model
  - **Client**: browser that requests, receives, “displays” web objects
  - **Server**: Web server sends objects in response to requests

- Also used as part of many other application layer protocol
HTTP Development Timeline

- Mar 1990  CERN labs document proposing Web
- Jan 1992  HTTP/0.9 specification
- Dec 1992  Proposal to add MIME to HTTP
- Feb 1993  UDI (Universal Document Identifier) Network
- Mar 1993  HTTP/1.0 first draft
- Jun 1993  HTML (1.0 Specification)
- Oct 1993  URL specification
- Nov 1993  HTTP/1.0 second draft
- Mar 1994  URI in WWW
- May 1996  HTTP/1.0 Informational, RFC 1945
- Jan 1997  HTTP/1.1 Proposed Standard, RFC 2068
- Jun 1999  HTTP/1.1 Draft Standard, RFC 2616
- 2001      HTTP/1.1 Formal Standard
- ...ongoing HTTP/2 Drafts and Standardization
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

The HTTP Protocol: Basics

HTTP: TCP transport service
- 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
HTTP Message Format: Request

- Two types of http messages: *Request, response*
- http request message:
  - ASCII (human-readable format)

```
GET /somedir/page.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr
```

(extra carriage return, line feed)

Carriage return, line feed indicates end of message
HTTP Request Methods

- Methods
  - GET
  - HEAD
  - POST
  - PUT
  - Delete
HTTP Message Format: Response

- **status line**
  - (protocol
  - status code
  - status phrase)

- **header lines**
  - **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 data ...
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
The HTTP Protocol: Connections

- **Non-persistent connection:**
  One object in each TCP connection
  - Some browsers create multiple TCP connections *simultaneously* — one per object

- **Persistent connection:**
  Multiple objects transferred within one TCP connection

- **Pipelined persistent connections:**
  Multiple requests issued without waiting for response
  Responses are in order of the requests

- **Persistent connection with out-of-order delivery (SPDY):**
  Multiple requests issued. Server can return the objects in arbitrary order. Typically according to server priority. In addition the server may send additional objects.
User-Server Interaction: Authentication

Authentication goal: Control access to server documents

- Stateless: Client must present authorization in each request
- Authorization: Typically name, password
  - Authorization: header line in request
  - If no authorization, server refuses access, sends WWW authenticate: header line in response

client

usual http request msg

401: authorization req.

WWW authenticate:

usual http request msg

+ Authorization: line

usual http response msg

server

usual http request msg

usual http response msg

usual http request msg

+ Authorization: line

usual http response msg

usual http response msg

time
User-Server interaction: Conditional GET

- **Goal:** Don’t send object if client has up-to-date stored (cached) version
  
- **Client:** Specify date of cached copy in http request
  
  If-modified-since: <date>

- **Server:** Response contains no object if cached copy up-to-date:
  
  HTTP/1.0 304 Not Modified

  If-modified-since: <date>

  HTTP/1.0 200 OK
  ...
  <data>

  object modified

  object not modified
User-Side State: Cookies

Most 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 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

Client

- usual http request msg

- usual http response +
  Set-cookie: 1678

Server

- server creates ID 1678 for user

Cookie file
- ebay: 8734

One week later:

- usual http request msg

- cookie: 1678

- usual http response msg

Cookie file
- amazon: 1678
- ebay: 8734
Cookies: Keeping State (2)

What cookies can bring:
- Authorization
- Shopping carts
- Recommendations
- User session state (e.g. for 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 even more
- Advertising companies obtain info across sites

- Users can even be tracked if cookies are turned off
Web Caches (Proxy Servers)

Goal: Satisfy client request without involving origin server

- User sets browser: WWW accesses via web cache
- Client sends all http requests to web cache
  - If object at web cache, web cache immediately returns object in http response
  - Else requests object from origin server, then returns http response to client
Why Web Caching?

Assume: Cache is “close” to client (e.g., in same network)

- Smaller response time: cache “closer” to client
- Decrease traffic to distant servers
  - Link out of institutional/local ISP network often bottleneck
Web 2.0: E.g., AJAX Enabled Apps

- E.g.: Google Maps: A canonical AJAX application
Content Distribution Networks (CDNs)

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

![CDN diagram]

- Origin server in North America
- CDN distribution node
- CDN server in S. America
- CDN server in Europe
- CDN server in Asia
**CDN Example**

1. **Origin server**
   - www.foo.com
   - Distributes HTML
   - Replaces: http://www.foo.com/sports.ruth.gif

2. **CDNs authoritative DNS server**

3. **Nearby CDN server**

**HTTP request for**
www.foo.com/sports/sports.html

**DNS query for**
www.cdn.com

**HTTP request for**

**CDN company**
- cdn.com
- Distributes gif files
- Uses its authoritative DNS server to route redirect requests
More about CDNs

Routing requests
- CDN creates a “map”, indicating distances from leaf ISPs and CDN nodes
- When query arrives at authoritative DNS server
  - Server determines ISP from which query originates
  - Uses “map” to determine best CDN server

Not just Web pages
- Streaming stored audio/video
- Streaming real-time audio/video
  - CDN nodes create application-layer overlay network
DNS: Domain Name System

People: many identifiers:
- SSN, name, Passport #

Internet hosts, routers:
- IP address (32/128 bit) – used for addressing datagrams
- "name", e.g., gaia.cs.umass.edu – used by humans

Q: How to map between IP addresses and name?
- Domain Name System (DNS)
  RFC1034/RFC1035, RFC3007
DNS: Domain Name System

Domain Name System:

- *Distributed database:* Implemented in hierarchy of many *name servers*

- *Application-layer protocol:* Host, routers, name servers communicate to *resolve* names (address/name translation)
  - Core Internet function implemented as application-layer protocol
  - Complexity at network’s “edge”
DNS Name Servers

Why not centralize DNS?

- Single point of failure
- Traffic volume
- Distant centralized database
- Maintenance

Does not scale!
DNS Name Servers (2)

No server has all name-to-IP address mappings

Local name servers:
- Each ISP, company has *local (default) name server*
- Host DNS query first goes to local name server

Authoritative name server:
- For a host: stores that host’s IP address, name
- Can perform name/address translation for that host’s name
DNS: Hierarchical Naming Tree
Distributed, Hierarchical Database

Client wants IP for www.amazon.com; 1\textsuperscript{st} approx:
- Client queries a root server to find com DNS server
- Client queries com DNS server to get amazon.com DNS server
- Client queries amazon.com DNS server to get IP address for www.amazon.com
DNS: Root Name Servers

- Contacted by local name server that cannot resolve name.
- Root name server:
  - Contacts authoritative name server if name mapping not known.
  - Gets mapping.
  - Returns mapping to local name server.
  - Most use anycast addressing.

13 root name servers worldwide:

- a NSI Herndon, VA
- b USC-ISI Marina del Rey, CA
- c PSInet Herndon, VA
- d U Maryland College Park, MD
- e NASA Mt View, CA
- f Internet Software C. Palo Alto, CA
- g DISA Vienna, VA
- h ARL Aberdeen, MD
- i NORDUnet Stockholm
- j NSI (TBD) Herndon, VA
- k RIPE London
- m WIDE Tokyo
- l ICANN Marina del Rey, CA
- n ICANN Marina del Rey, CA
TLD and authoritative servers

- **Top-level domain (TLD) servers:**
  Responsible for com, org, net, edu, etc, and all top-level country domains like uk, fr, ca, jp...
  - Verisign, Inc. maintains servers for .com TLD
  - DENIC maintains servers for .de TLD

- **Authoritative DNS servers:**
  Organization’s DNS servers, providing authoritative hostname to IP mappings for organization’s servers (e.g., Web and mail).
  - Can be maintained by organization or service provider
Local Name Server (Recurser)

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has at least one.
  - Also called “default name server”
- When a host makes a DNS query, query is sent to its local DNS server
  - Acts as a proxy, forwards query into hierarchy.
DNS Records

**DNS**: Distributed db storing resource records (RR)

**RR format**: (name, value, type, ttl)

- **Type=A**
  - *name* is hostname
  - *value* is IP address

- **Type=NS**
  - *name* is domain (e.g., foo.com)
  - *value* is IP address of authoritative name server for this domain

- **Type=CNAME**
  - for alias

- **Type=MX**
  - for mail

- **Type=AAAA**
  - for IPv6
Example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu
Recursive queries

Recursive query:
- Puts burden of name resolution on contacted name server
- Heavy load?

Iterated query:
- Contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”
DNS: Iterative Queries

Recursive query:
- Puts burden of name resolution on contacted name server
- Heavy load?

Iterated query:
- Contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”

Diagram:
- Local name server: dns.eurecom.fr
- Intermediate name server: dns.umass.edu
- Authoritative name server: dns.cs.umass.edu
- Requesting host: surf.eurecom.fr
- Iterated query path:
  1. Requesting host to Local name server
  2. Local name server to Intermediate name server
  3. Intermediate name server to Authoritative name server
  4. Authoritative name server to Intermediate name server
  5. Intermediate name server to Local name server
  6. Local name server to Intermediate name server
  7. Intermediate name server to Local name server
  8. Local name server to Requesting host
Mapping IP Address to Names


23.220.149.130.in-addr.arpa.
1.a.c.6.d.c.5.8.0.5.6.8.0.6.d.c.1.0.0.0.9.b.6.9.0.7.4.0.1.0.0.2.ip6.arpa.