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
WWW: The HTTP protocol

HTTP: hypertext transfer protocol
- WWW’s application layer protocol
- Client/server model
  - **Client:** browser that requests, receives, “displays” WWW objects
  - **Server:** WWW server sends objects in response to requests
HTTP - 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
- ...

...
The HTTP protocol: More

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 WWW server (http server)
- TCP connection closed

HTTP is “stateless”
- Server maintains no information about past client requests

Aside
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 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)
HTTP message format: Reply

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 ...

data, e.g., requested html file

header lines
HTTP reply 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
HTTP request methods

Methods
- GET
- HEAD
- POST
- PUT
- Delete
The HTTP protocol: Even more

- **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
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

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>usual HTTP request msg</td>
<td>401: authorization req. WWW authenticate:</td>
</tr>
<tr>
<td>usual HTTP request msg + Authorization:line</td>
<td>usual HTTP response msg</td>
</tr>
<tr>
<td>usual HTTP request msg + Authorization:line</td>
<td>usual HTTP response msg</td>
</tr>
<tr>
<td>usual HTTP request msg</td>
<td>usual HTTP response msg</td>
</tr>
</tbody>
</table>
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>
  ```
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

Cookie file
amazon: 1678
ebay: 8734

Usual HTTP request msg
Usual HTTP response +
Set-cookie: 1678

Cookie file
amazon: 1678
ebay: 8734

Usual HTTP request msg
Cookie: 1678
Usual HTTP response msg

One week later:

Cookie file
amazon: 1678
ebay: 8734

Usual HTTP request msg
Cookie: 1678
Usual HTTP response msg

Server

Server creates ID 1678 for user

Cookie-specific action

Entry in backend database

Access

Access
Cookies: Keeping “state” (2)

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
Web caches (proxy server)

**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 already in web cache, web cache immediately returns the object in http response
  - Else requests object from origin server, then returns http response to client
Why WWW 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 example**

1. **Origin server**
   - www.foo.com
   - Distributes HTML

2. **CDNs authoritative DNS server**
   - DNS query for www.cdn.com

3. **Nearby CDN server**

---

**Origin server**
- www.foo.com
- Distributes HTML

**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 bit) – used for addressing datagrams
- “name”, e.g., gaia.cs.umass.edu – used by humans

Q: Map between IP addresses and name?

- Secure Domain Name System (DNS)
  Dynamic Update: RFC 3007
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; 1st 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
  - Some use anycast

13 root name servers worldwide
TLD and authoritative servers

- **Top-level domain (TLD) servers**: Responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Network solutions maintains servers for com TLD
  - Educause for edu 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

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has 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
  - name is hostname (e.g., foo.com)
  - value is canonical hostname (e.g., relay1.bar.foo.com)

- **Type=MX**, for mail
  - name is alias mail server name (e.g., foo.com)
  - value is canonical mail servername (e.g., mail.bar.foo.com)
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”
Mapping IP address to names

- Special domain: ARPA

68.49.149.130.in-addr.arpa.