Network Applications

Yan Wang
Applications and Application-layer Protocols

- Application: communicating, distributed processes
  - e.g., e-mail, Web, instant messaging
  - running in end systems (hosts)
  - exchange messages to implement application
- Application-layer protocols
  - one “piece” of an application
  - define messages exchanged by applications and actions taken
  - use communication services provided by lower layer protocols (TCP, UDP)
Application-layer Protocol Defines

- Types of messages exchanged, e.g., request & response messages
- Syntax of message types: what fields in messages & how fields are delineated
- Semantics of the fields, i.e., meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:
- allowing for interoperability
- e.g., HTTP, FTP

Proprietary protocols:
- e.g., Skype
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., web client implemented in browser
- **Server:**
  - provides requested service to client
  - e.g., web server sends requested web page
Web and HTTP
World Wide Web

- Server provides access to web pages, which are typically prepared off-line.
- Browsers request web pages from server using Hypertext Transfer Protocol (HTTP).
- Objects on the web are identified by Uniform Resource Locators (URL).
HTTP

• A simple client-server transaction protocol
• Web’s application layer protocol
• client/server model
  ▫ client: browser that requests, receives (using the HTTP protocol), and displays Web objects
  ▫ server: Web server sends (using the HTTP protocol) objects in response to requests

• HTTP 1.0: RFC 1945
• HTTP 1.1: RFC 2068
HTTP (2)

- **Uses TCP:**
  - client initiates TCP connection 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
Web and HTTP

- **Web page consists of objects**
- Object can be HTML file, JPEG image, Javascript, audio file, video 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
  ```

  - **host name**
  - **path name**
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 over single TCP connection between client and server.
  - HTTP/1.1 uses persistent connections in default mode
Non-persistent HTTP

Suppose user enters URL (contains text, references to 10 jpeg images)
www.someSchool.edu/someDepartment/index.html

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. Message indicates that client wants object /someDepartment/index.html

1b. HTTP server at host www.someSchool.edu on waiting for TCP connection at port 80. “accepts” connection, notifying client

3. HTTP server receives request message, forms response message containing requested object, and sends message into its socket

Steps 1-5 repeated for each of 10 jpeg objects.
Modeling Response Time

- **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 = 2RTT + transmit time**
Persistent HTTP

• Non-persistent HTTP issues:
  ▫ requires at least 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
Non-persistent HTTP vs. persistent HTTP

Non-persistent HTTP behavior

Persistent HTTP behavior
Persistent HTTP

- **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
HTTP Message: Request

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

```
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\nAccept-Language: en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7\r\nKeep-Alive: 115\r\nConnection: keep-alive\r\n```

carriage return character
line-feed character

request line
(GET, POST, HEAD commands)

header lines

carriage return, line feed at start of line indicates end of header lines
HTTP Request: General Format

- **Method**: A string that represents the action to be performed on the resource identified by the **URL** field.
- **URL**: The network location of the requested resource.
- **Version**: The version of the HTTP protocol being used.
- **Header lines**: Additional fields that may be included in the request.
- **Entity body**: Optional data that accompanies the request.

**Syntax**

```
method sp URL sp version
```

```
header field name: value
```

```
cr lf
```

**Notes**

- **sp**: space
- **cr**: carriage return (\r)
- **lf**: line feed (\n)
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**
    - identical to GET except that the server MUST NOT return a message-body in the 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 Message: Response

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\ndata data data data data data data ...

header lines

data, e.g., requested HTML file
HTTP Response Status Codes

Status codes appear in the 1st line in server to 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**
Trying Out HTTP

1. Telnet to a Web server:

```
telnet cs.binghamton.edu 80
```
Opens TCP connection to port 80 (default HTTP server port) at cs.binghamton.edu. Anything typed is sent to port 80 at cs.binghamton.edu.

2. Type in an HTTP GET request:

```
GET /~yanwang/CS428_528.html HTTP/1.1
Host: cs.binghamton.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 to look at captured HTTP request/response)
Client-Server Interaction: Authorization

- **Authorization**: control access to server content
  - authorization credentials: typically name, password
  - **stateless**: client must present authorization in each request
    - **authorization**: header line in each request
    - if **no authorization**: header, server refuses access, sends **WWW authenticate**: header line in response

```
client

usual http request msg

401: authorization req.

server

WWW authenticate:

usual http request msg
+ Authorization: <cred>

usual http response msg

usual http request msg
+ Authorization: <cred>

usual http response msg
```

```
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 accesses 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 (2)

one week later:

usual http request msg
cookie: 1678
usual http response msg

Amazon server creates ID 1678 for user
create entry
backend database
access

usual http request msg
cookie: 1678
usual http response msg

amazon server
client
server

usual http request msg
usual http response msg

cookie file
amazon 1678

create

access

specific action

specific action

specific action
Cookies: Keeping State (3)

- What cookies can be used for:
  - 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
Conditional GET: Client-Side Caching

**Goal:** don’t send object if the client (e.g., the browser) has an 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

HTTP request msg

If-modified-since: <date>

HTTP response

HTTP/1.0 304 Not Modified

---

**server:** response contains object modified after <date>

HTTP request msg

If-modified-since: <date>

HTTP response

HTTP/1.0 200 OK

<data>

---

object modified after <date>

---

object not modified before <date>
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
Web Caching

- Cache acts as both client and server
- Cache can do up-to-date check using `If-modified-since` HTTP header
  - Issue: should cache take risk and deliver cached object without checking?
  - Heuristics are used.
- 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
Caching Example

Assumptions
- average object size = 1 Mbits
- avg. request rate from institution’s browser to origin servers = 15/sec
- avg. data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server and back to router = 2 sec
- access link rate: 15 Mbps

Consequences
- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds
Caching Example: fatter access link

**Assumptions**
- average object size = 1Mbits
- avg. request rate from institution’s browser to origin servers = 15/sec
- avg. data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server and back to router = 2 sec
- access link rate: 15 Mbps

**Consequences**
- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds

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

**Install cache**
- suppose hit rate is 0.4

**Consequence**
- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 0.6*15 Mbps/15 Mbps = 60%, resulting in negligible delays (say 10 msec)
- total delay
  
  \[
  \text{total delay} = 0.6 \times (\text{delay from origin servers}) + 0.4 \times (\text{delay when satisfied at cache})
  \]
  
  \[
  = 0.6 \times (2.01) + 0.4 \times (~\text{msecs})
  \]
  
  \[
  = \sim 1.2 \text{ secs}
  \]
  
  less than with 100 Mbps link

*Cost:* web cache (cheap!)
FTP: File Transfer Protocol

- Transfer file to/from remote host
- Client/server model
- Client: side that initiates transfer (either to/from remote)
- Server: remote host
- FTP: RFC 959
- FTP server: port 21
• FTP client contacts FTP server at port 21, using TCP
• client authorized over control connection
• client browses remote directory, sends commands over control connection
• when server receives file transfer command, *server* opens 2\textsuperscript{nd} TCP data connection (for file) to client
• after transferring one file, server closes data connection

![Diagram of FTP connections](image)

• one *control connection* for each FTP session
• one *data connection* for each file transfer
• **Common commands:**
  - **ABOR**: abort data transfer
  - **LIST** *filelist*: list files or directories
  - **QUIT**: logoff
  - **RETR** *filename*: retrieve (get) a file
  - **STOR** *filename*: store (put) a file
  - **TYPE** *transfertype*: A for ASCII, I for binary
  - **USER** *username*: username on server
  - **PASS** *password*: password on server
  - **PORT** *a,b,c,d,e,f*: client endpoint (a.b.c.d, e*256 + f)
• Common Replies:
  ▫ 125: data connection open; transferring
  ▫ 331: username OK, password required
  ▫ 425: cannot open data connection
  ▫ 452: error writing file
  ▫ 500: unrecognized command

• Note that the user interface (graphic or text-based) is determined by the implementation, not the protocol.
Control Connection

- The server waits for connection requests on port 21
- The client sends connection requests to port 21 of the server
  - A group of well-known ports has been reserved for important applications
  - ports 20 and 21 are reserved for FTP
  - client port # is up to client OS
Data Connection

• To establish a data connection, the client chooses an unused port number and sends this number to the server via the control connection, using the PORT command.
• The server receives the client’s port number and establishes a TCP connection to the client; the server’s port number is always 20.
DNS
Domain Name System (DNS)

- On the Internet, hosts are identified by IP addresses.
- However, addresses are hard for users to remember.
- A user-friendly name is also typically assigned to each host in a network.
- Use DNS protocol to map between host names and IP addresses.
- DNS is a distributed database implemented in hierarchy of many name servers.
- It is an application-layer protocol that provides core Internet function: hosts, name servers communicate to resolve names (address/name translation)
DNS Service

- Hostname to IP address translation
- Host aliasing
  - canonical, alias names
- Mail server aliasing
- Load distribution
  - replicated Web servers: many IP addresses correspond to one name
DNS: a Distributed, Hierarchical Database

- Client wants IP for www.amazon.com
  - client queries 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
• There are 13 root NS around the world, maintaining 13 identical databases of top-level domain NS.
• Every root NS knows all .com NS, .edu NS, .net NS, .org NS, ...
• Each .com NS is also “complete”, it knows the NS of all 2nd-level .com domains.
  ▫ It knows the NS of amazon.com, google.com, etc.
• The same applies to every .net NS, .edu NS, .jp NS, and so on.

"Root-current" by No machine-readable author provided. Matthäus Wander assumed (based on copyright claims). - No machine-readable source provided. Own work assumed (based on copyright claims). Licensed under CC BY 2.5 via Commons - https://commons.wikimedia.org/wiki/File:Root-current.svg#/media/File:Root-current.svg
• **Top-level domain (TLD) servers:**
  - responsible for **com, org, net, edu, aero, jobs, museum**, and all top-level country domains, e.g., **uk, fr, ca, jp**

• **Authoritative DNS servers:**
  - organization’s own DNS server(s)
  - provide authoritative hostname to IP mappings for organization’s named hosts
  - can be maintained by organization or service provider
• It is the lower-level NS that actually maintain machine addresses.
  ▫ An amazon NS knows the exact IP address of www.amazon.com
• A BU NS knows the exact IP address of cs.binghamton.edu
• Each low-level NS *knows* all machines in its domain.
• Every NS in the world has the list of root NS.
• Each host is configured with the IP addresses of one or two local NS.
Having the IP addresses of Amazon server and BU host.
Local Name Server

• One per ISP (residential ISP, company, university)
• When host makes DNS query, query is sent to its local DNS server.
  ▫ Acts as a proxy, forwards query into hierarchy
  ▫ Reduces lookup latency for commonly searched hostnames
• Hosts learn local name server via:
  ▫ DHCP (same protocol that tells host its IP address)
  ▫ Static configuration (e.g., can use Google’s “local” name server at 8.8.8.8 or 8.8.4.4)
Application’s Use of DNS

- Extract server name (e.g., from the URL)
- Do \texttt{getaddrinfo()} to trigger resolver code, sending messages to the local name server
DNS Name Resolution Example

- Host at cs.binghamton.edu wants IP address for gaia.cs.umass.edu

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

- Once (any) name server learns mapping, it caches mapping
  - Cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
  - Thus, root name servers not often visited
Content Distribution Networks (CDNs)

- **Challenge**: how to serve content to hundreds of thousands of simultaneous users?

- **Option 1**: single, large “mega-server”
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

....quite simple, but, this solution *doesn’t scale*
Content Distribution Networks (CDNs)

- **Challenge**: how to serve content to hundreds of thousands of simultaneous users?

- **Option 2**: store/serve multiple copies of videos at multiple geographically distributed sites (CDN)
  - *enter deep*: push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home*: smaller number (10’s) of larger clusters in points of presence (POPs) near (but not within) access networks
    - used by Limelight
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
A (Simple) Example

- Bob (client) requests video http://video.netcinema.com/6Y7B23V
- video stored in CDN at http://a1105.kingCDN.com/6Y7B23V

Bob gets URL for video http://video.netcinema.com/6Y7B23V from netcinema.com web page

1. 6. request video from kingCDN server, returned via HTTP
2. resolve video.netcinema.com via Bob’s local DNS
3. netcinema’s DNS returns a1105.kingcdn.com

KingCDN.com

4. netcinema’s authoritative DNS

KingCDN authoritative DNS
Try it out yourself

- The UNIX utility **dig** can be used for DNS lookups. Read the **man** page for **dig**
- Use **dig** to find out which CDN service is bestbuy using and, how images on www.bestbuy.com are served.
CDN Server Selection Strategy

- How does CDN DNS select “good” CDN node to serve clients
  - pick CDN node geographically closest to client
  - pick CDN node with shortest delay (or min # hops) to client
    (CDN nodes periodically ping access ISPs, reporting results to CDN DNS)

- Alternative: let the clients decide
  - give client a list of several CDN servers
  - client pings servers, picks “best”
  - Netflix approach – available bandwidth based?
Reading

- Sections 9.1.2, 9.3.1, 9.4.2, 9.4.3