The Application Layer

Chapter 2 from the Kurose and Ross textbook

PART 1 – Sections 2.1 to 2.3
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

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

**client process**: process that initiates communication

**server process**: process that waits to be contacted

Note: 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

The sending process must specify

- The name or address of the host (hostname or IP address*), and
- The process on that host that will handle the received message (port number).
- For example, HTTP typically uses port 80, while SMTP uses port 25.

*More when we get to the network layer.
What else is defined?

<table>
<thead>
<tr>
<th>types of messages exchanged</th>
<th>e.g., request, response</th>
</tr>
</thead>
<tbody>
<tr>
<td>message syntax</td>
<td>what fields in messages</td>
</tr>
<tr>
<td></td>
<td>how fields are delineated</td>
</tr>
<tr>
<td>message semantics</td>
<td>meaning of information in fields</td>
</tr>
<tr>
<td>rules</td>
<td>for when and how processes send &amp; respond to messages</td>
</tr>
</tbody>
</table>

Two forms of definitions:
- Open protocols are defined in RFCs, and allow for interoperability. Such as HTTP, SMTP
- Proprietary protocols, such as Skype

Well, what does the application need?

<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>Email</td>
<td>No loss</td>
<td>Elastic</td>
<td>No</td>
</tr>
<tr>
<td>Web documents</td>
<td>No loss</td>
<td>Elastic (few kbps)</td>
<td>No</td>
</tr>
<tr>
<td>Real-Time audio/video</td>
<td>Loss-tolerant</td>
<td>Audio: few kbps–1Mbps Video: 10 kbps–5 Mbps</td>
<td>Yes: 100s of msec</td>
</tr>
<tr>
<td>Stored audio/video</td>
<td>Loss-tolerant</td>
<td>Same as above</td>
<td>Yes: few seconds</td>
</tr>
<tr>
<td>Interactive games</td>
<td>Loss-tolerant</td>
<td>Few kbps–10 kbps</td>
<td>Yes: 100s of msec</td>
</tr>
<tr>
<td>Instant messaging</td>
<td>No loss</td>
<td>Elastic</td>
<td>Yes and no</td>
</tr>
</tbody>
</table>
TCP services

• Provides a connection-oriented service.
  – Handshakes set up a full-duplex TCP connection between sockets prior to the exchange of information.

• Provides reliable transport service.
  – The communication processes can rely on TCP to deliver all data sent without error and in the proper order.
  – Includes a congestion-control mechanism.*

*But at a cost and TCP does not guarantee a minimum transmission rate.

UDP provides

• No frills delivery
  – unreliable data transfer service.

• No congestion-control
  – the socket is always open.
### 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>

### The WWW and HTTP

The Web's application-layer protocol. Relatively simple and illustrates many key principles.
The World Wide Web


- He wrote the first web client and server in 1990. His specifications of URIs, HTTP and HTML were refined as Web technology spread.

Hyper-Text Transfer Protocol*

- HTTP is the Web’s client/server protocol; uses TCP as its underlying transport protocol.
- User agent (browser) implements the client side of HTTP.
- Web pages generally consist of a base HTML file which references other objects (JPEG, GIF, Java applet, audio clips).
HTTP nonpersistent connection

1a. Client initiates a TCP connection to www.someSchool.edu on port 80.
1b. Server at host www.someSchool.edu accepts connection and acknowledges.
2. Client sends HTTP request for /someDepartment/home.index to TCP socket set up in 1.
3. Server receives message through socket, finds, encapsulates, and sends object in HTTP response.

Then . . .

5. Client receives response message. TCP connection terminates. Client extracts file, examines HTML, and requests other objects.
4. Server tells TCP to close TCP connection (but TCP waits to hear from client first).
6. First four steps are repeated for each referenced object.
Timing of a nonpersistent connection*

*HTTP/1.0 can only issue nonpersistent connections.

Persistent HTTP

Non-persistent HTTP
- 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

GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/5.0
Connection: close
Accept-language: fr

*The Host line specifies the host on which the object resides. Seems kinda dumb since aren’t we already there?

HTTP request message: general format

<table>
<thead>
<tr>
<th>method</th>
<th>sp</th>
<th>URL</th>
<th>sp</th>
<th>version</th>
<th>cr</th>
<th>lf</th>
</tr>
</thead>
<tbody>
<tr>
<td>header field name</td>
<td>value</td>
<td>cr</td>
<td>lf</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>header field name</th>
<th>value</th>
<th>cr</th>
<th>lf</th>
</tr>
</thead>
<tbody>
<tr>
<td>cr</td>
<td>lf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

entity body
HTTP response message

- **status line**
  - (protocol status code status phrase)
  - HTTP/1.1 200 OK
  - Connection: close
  - Date: Sat, 07 Jul 2007 12:00:15 GMT
  - Server: Apache/1.3.0 (Unix)
  - Last-Modified: Sun, 6 May 2007 ...
  - Content-Length: 6821
  - Content-Type: text/html

- **header lines**
  - data data data data data ...

- **requested HTML file**

Some HTTP response status codes

- **200 OK**
  - request succeeded, requested object in this message

- **301 Moved Permanently**
  - requested object moved, new location specified 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**
Try it!

Type:  
telnet cs.wellesley.edu 80

Opens TCP connection to port 80

Type:  
GET /~rshull/index.html HTTP/1.1
Host: cs.wellesley.edu

Issues a GET request to HTTP server (don’t forget to hit carriage return twice at the end).

HTTP is stateless

- The HTTP protocol only knows only what it sees in the current request. This simplifies the server design and permitted the development of high-performance Web servers.

- Sometimes, however, HTTP’s lack of memory can be a big disadvantage.

- We need ...
Cookies: keeping “state”

Cookies and privacy
The World Wide Wait

- Low-speed links result in large bottleneck link delays (large L/R).
- Congested links cause long queuing delays and dropped packets.
- The referenced web server is overworked and underpaid.

Web caching

- Web caches (proxy server)* attempt to satisfy HTTP requests on behalf of the origin servers.
- The content provider is not involved; content is replicated on demand.

*Similar to your browser caching recently viewed materials.
1. Browser establishes TCP connection to Web cache & sends request.
2. If web cache has a copy, it sends it.
3. If web cache does not have a copy, it opens a TCP connection to www.someschool.edu to get one.
4. Web cache receives object, stores a copy and forwards a copy to client.

LAN no problem

- Average size of requested objects is 100,000 bits.
- Average request rate to origin servers is 15 requests per second.
- Average Internet delay is 2 seconds.
- Total delay
  \[ d_{total} = d_{LAN} + d_{access} + d_{Internet} \]
**Bottleneck**

- Average size of requested objects is 100,000 bits.
- Average request rate to origin servers is 15 requests per second.
- Average Internet delay is 2 seconds.
- Total delay
  \[ d_{total} = d_{LAN} + d_{access} + d_{Internet} \]

**A solution**

- Traffic intensity on the 1.5 Mbps institution to Internet access link results in unacceptably large delays.
- One possible solution is to upgrade to 10 Mbps access link.
- Does this help?
  \[ d_{total} = d_{LAN} + d_{access} + d_{Internet} \]
An alternative solution

- Web cache hit rates range from 0.2 to 0.7. Suppose a study suggests a hit rate of 0.4 for this institution.

- Thus, 40% of the time requests are satisfied within 10 msecs.

- The remaining 60% of the request still go over the 1.5 Mbps access link.

- At traffic intensity 0.6, delays drop to the millisecond range and Internet access ~ 2.01 secs.

The conditional GET

- Although caching can reduce user-perceived response time, there is a potential snag.

- The object housed in the Web server may have been modified since the copy was cached.

- HTTP provides a mechanism that allows a cache to verify its objects are up to date.
### Conditional GET in action

**Browser**

GET /fruit/kiwi.gif HTTP/1.1  
Host: www.exotiquecuisine.com

**Web Cache**

GET /fruit/kiwi.gif HTTP/1.1  
Host: www.exotiquecuisine.com

**Server**

HTTP/1.1 200 OK  
Date: Sat, 7 Jul 2007 15:39:29  
Server: Apache/1.3.0 (Unix)  
Context-Type: image/gif  
(data data data ...)

---

### Three weeks later

**Browser**

GET /fruit/kiwi.gif HTTP/1.1  
Host: www.exotiquecuisine.com

**Web Cache**

GET /fruit/kiwi.gif HTTP/1.1  
Host: www.exotiquecuisine.com  
If-modified-since: Wed, 4 Jul 2007 09:23:24

**Server**

HTTP/1.1 304 Not Modified  
Date: Sat, 28 Jul 2007 15:39:29  
Server: Apache/1.3.0 (Unix)  
(empty entity body)
The File Transfer Protocol

FTP History 101

- The original specification for the File Transfer Protocol was written by Abhay Bhushan (RFC 114, April 1971).

- It ran on NCP until 1980* when it was replaced by a TCP/IP version (RFC 114).

*Flag day.
File Transfer Protocol (FTP)*

Like HTTP, FTP is a file transfer protocol and shares many common characteristics. However, there are some important differences.

*Fetch on the Mac and Winsock on the PC.

FTP and HTTP

- FTP, like HTTP, runs on top of TCP.
- However, unlike HTTP, FTP uses two parallel TCP connections to transfer a file, a control connection and a data connection. We say FTP sends its control information out-of-band.

- Also unlike HTTP, FTP maintains state. In particular, FTP remembers the current directory and earlier authentication.
Control and data connection

- When a user starts an FTP session, the client side of FTP initiates a control TCP connection with the server.
- When the server receives a command for a file transfer, it initiates a TCP data connection to the client side.
- The control connect stays open. The data connection stays open for one file only.

FTP commands*, responses**

Sample commands:
sent as ASCII text over control channel
- USER username
- PASS password
- LIST return list of file in current directory
- RETR filename retrieves (gets) file
- STOR filename stores (puts) file onto remote host

Sample return codes
status code and phrase (as in HTTP)
- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can’t open data connection
- 452 Error writing file

*Four uppercase 7-bit ASCII characters followed by some optional arguments.
**Replies are three-digit numbers with optional message.
Web browser support

• Most web browser can retrieve files hosted on FTP servers. FTP URL syntax is described in RFC1738:
  ftp://[<user>[:<password>[@]<host>[:<port>[/]<url-path>]

• For example,

Innocence abroad

• A host that provides an FTP service may provide anonymous FTP access.

• Users log into the service with an ‘anonymous’ account with prompted for a user name.

• Users may be asked to send their email address instead of a password, but no verification is performed.
Danger …

- FTP was not designed to be a secure protocol and it isn’t.
- It is not able to encrypt its traffic; all transmissions are in clear text, and usernames, passwords, commands and data can be easily read by anyone able to perform packet capture.
- Exploits include: Offline password cracking, spoof attacks; and bounce attacks.

Getting by in a dangerous world

- Explicit SFTP is an extension to the FTP standard that allows clients to request that the FTP session be encrypted. This is done by sending the "AUTH TLS" command.
- SFTP, SSH* File Transfer Protocol, is not related to FTP except that it also transfers files and has a similar command set for users.
Email

How much is there to say about Email?

email: The killer app of the seventies

• In 1971 Ray Tomlinson sent the first successful email from one computer, to another sitting right next to it – through ARPANET.

• He used the @ sign to delineate what host the user receiving the message was on.

• Tomlinson can’t remember his first message but stated it was “QWERTYUIOP or something similar.”
You’ve got mail*

• Like ordinary mail (and unlike either the HTTP or FTP), e-mail is asynchronous.

• It has three major components:
  – user agents
  – mail servers
  – Simple Mail Transfer Protocol (SMTP)

* Please tell me you get that reference!

Electronic mail in the Internet

User agents allow users read, reply, forward, save, and compose messages.

*Currently, our browser acts as a user agent. Formerly, we used a proprietary agent from FirstClass.
Servers form the core

Each recipient has a mailbox located in one of the mail servers.

SMTP*

SMTP (Simple Mail Transfer Protocol, is the principal application-layer protocol for Internet electronic mail.

Like HTTP, SMTP has a client and server side. Both sides run on every server.

*Something of a legacy technology (RFC 1982), SMTP requires body of all mail messages to be in simple seven-bit ASCII.
Alice sends Bob a message

Alice invokes her user agent, provides Bob's address, composes message, and instructs user agent to send.
Alice sends Bob a message

1. Alice’s user agent sends the message to her mail server, where it is placed in message queue.

2. Client side of SMTP, running on Alice’s server sees message, opens TCP connection to SMTP server on Bob’s mail server.

3. Bob’s mail server receives the message.

4. Bob’s agent retrieves the message from the mail server.

Key:
- Message queue
- User mailbox
Alice sends Bob a message

After initial SMTP handshaking, SMTP client sends Alice's message into TCP connection.

At Bob's mail server, the server side of SMTP receives message and puts it into Bob's mailbox.
Alice sends Bob a message

Try SMTP interaction for yourself!

- `telnet servername 25`
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)
Comparing HTTP & SMTP

• Both transfer files using persistent connections on top of TCP.

• HTTP is mainly a pull protocol, while SMTP is a push protocol.

• SMTP requires each message to be 7-bit ASCII.

• HTTP encapsulates each object in its own HTTP message. SMTP places all objects in one message.

Mail message format

• SMTP: protocol for exchanging email msgs

• RFC 822: standard for text message format:

• Header lines, e.g.,
  – To:
  – From:
  – Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!

• Body: the “message”
  – ASCII characters only
Problem: How do we send attachments?

Unanswered questions

- Alice user agent uses SMTP to push e-mail to her mail server.
- Mail server relays e-mail to Bob. Why the two step?
- How does Bob’s user agent obtain his message from Alice?
Mail access protocols

• Since SMTL is a PUSH protocol and Bob wants to PULL his messages, he needs another application-layer protocol.

• There are several to choose from: Post Office Protocol (POP3); Internet Mail Access Protocol (IMAP); and HTTP.

POP3 protocol

• POP3 is an extremely simple mail access protocol with rather limited functionality, ...

• ... but it does get the job done.
**POP3 protocol**

- **authorization phase**
  - client commands:
    - **user**: declare username
    - **pass**: password
  - server responses
    - +OK
    - -ERR

- **transaction phase**, client:
  - **list**: list message numbers
  - **retr**: retrieve message by number
  - **dele**: delete
  - **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

**IMAP and HTTP**

- POP3 is a no frills access protocol.
- IMAP is somewhat more luxurious.
  - keeps all messages on server
  - allows users to organize messages into folders
  - keeps user state across sessions (like FC).
- Alternatively, we could build an e-mail application on top of HTTP (Gmail, FC, ...).