SMTP
Electronic Mail

Three Components:

1. **User Agents**
   - a.k.a. “mail reader”
   - e.g., gmail, Outlook, yahoo

2. **Mail Servers**
   - *mailbox* contains incoming messages for user
   - *message queue* of outgoing (to be sent) mail messages
   - *Sender mail server* makes connection to *Receiver mail server* (IP address, port 25)

3. **SMTP protocol**
   - Used to send messages
   - *Client*: sending user agent or sending mail server
   - *server*: receiving mail server
Scenario: Alice sends message to Bob

1) Alice uses UA to compose message and “to” bob@someschool.edu
2) Alice’s UA sends message to her mail server; message placed in message queue
3) Client side of SMTP opens TCP connection with Bob’s mail server
4) SMTP client sends Alice’s message over the TCP connection
5) Bob’s mail server places the message in Bob’s mailbox
6) Bob invokes his user agent to read message
Sample SMTP interaction

220 hill.com SMTP service ready

HELO town.com
250 hill.com Hello town.com, pleased to meet you

MAIL FROM: <jack@town.com>
250 <jack@town.com>… Sender ok

RCPT TO: <jill@hill.com>
250 <jill@hill.com>… Recipient ok

DATA
354 Enter mail, end with “.” on a line by itself

Jill, I’m not feeling up to hiking today. Will you please fetch me a pail of water?
.

250 message accepted

QUIT
221 hill.com closing connection
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Positive Completion Reply</strong></td>
<td></td>
</tr>
<tr>
<td>211</td>
<td>System status or help reply</td>
</tr>
<tr>
<td>214</td>
<td>Help message</td>
</tr>
<tr>
<td>220</td>
<td>Service ready</td>
</tr>
<tr>
<td>221</td>
<td>Service closing transmission channel</td>
</tr>
<tr>
<td>250</td>
<td>Request command completed</td>
</tr>
<tr>
<td>251</td>
<td>User not local; the message will be forwarded</td>
</tr>
<tr>
<td><strong>Positive Intermediate Reply</strong></td>
<td></td>
</tr>
<tr>
<td>354</td>
<td>Start mail input</td>
</tr>
<tr>
<td><strong>Transient Negative Completion Reply</strong></td>
<td></td>
</tr>
<tr>
<td>421</td>
<td>Service not available</td>
</tr>
<tr>
<td>450</td>
<td>Mailbox not available</td>
</tr>
<tr>
<td>451</td>
<td>Command aborted: local error</td>
</tr>
<tr>
<td>452</td>
<td>Command aborted; insufficient storage</td>
</tr>
<tr>
<td><strong>Permanent Negative Completion Reply</strong></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>Syntax error; unrecognized command</td>
</tr>
<tr>
<td>501</td>
<td>Syntax error in parameters or arguments</td>
</tr>
<tr>
<td>502</td>
<td>Command not implemented</td>
</tr>
<tr>
<td>503</td>
<td>Bad sequence of commands</td>
</tr>
<tr>
<td>504</td>
<td>Command temporarily not implemented</td>
</tr>
<tr>
<td>550</td>
<td>Command is not executed; mailbox unavailable</td>
</tr>
<tr>
<td>551</td>
<td>User not local</td>
</tr>
<tr>
<td>552</td>
<td>Requested action aborted; exceeded storage location</td>
</tr>
<tr>
<td>553</td>
<td>Requested action not taken; mailbox name not allowed</td>
</tr>
<tr>
<td>554</td>
<td>Transaction failed</td>
</tr>
</tbody>
</table>
Mail access protocols

- **SMTP**: delivery/storage to receiver's server
- **Mail access protocol**: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
  - IMAP: Internet Mail Access Protocol [RFC 1730]
  - HTTP: Hotmail, Yahoo! Mail, etc.
Mail message (stored on server) format

SMTP: protocol for exchanging email msgs
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

- body
  - the “message”, ASCII characters only
Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data ..... 
........................
......base64 encoded data
```
SMTP: final words

Comparison with HTTP:

- HTTP: pull
- SMTP: push

- both have **ASCII command/response** interaction, status codes

- HTTP: each object included in its own response msg
- SMTP: multiple objects sent in multipart msg
Review for App Layer Protocols
App-Layer Protocols
App-layer protocol defines

- Types of messages exchanged:
  - e.g., request, response

- Message format:
  - Syntax: what fields in messages
  - Semantics: meaning of information in fields

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

Public-domain protocols:
- defined in RFCs
- DNS, HTTP, FTP, SMTP

Proprietary protocols:
- e.g., Skype, Hangout
Network Application

- To communicate, 2 hosts need to identify each other.

- Computer network: IP address
  - IPv4 (32 bits)
  - IPv6 (128 bits)

- More than one program on a host: Port #

- A network connection is a 4-tuple:
  - $IP_S$, $Port_S$, $IP_D$, $Port_D$
Client-server architecture (CS)

Server:
- always-on host
- permanent IP address
- server farms for scaling

Clients:
- communicate with server
- may not be always connected
- may have dynamic IP addresses
- do not communicate directly with each other
App-layer protocols

- DNS: Domain Name Service
- HTTP: HyperText Transfer Protocol
- FTP: File Transfer Protocol
- SMTP: Simple Mail Transfer Protocol
DNS
Domain Name System (DNS)

- For any networked application, we need to know the IP address of a given host name.

- Problem:
  - On average, IP addresses have 12 digits
  - We need an easier way to remember IP addresses

- Solution:
  - Use names to refer to hosts
  - Add a service (DNS) to map between host names and IP addresses
  - We call this Address Resolution
Simple DNS

Centralize DNS?
- single point of failure
- traffic volume
- Distant centralized database
- maintenance
doesn't scale!
Distributed, Tree-based Database

Root DNS Servers

- com DNS servers
  - yahoo.com DNS servers
  - amazon.com DNS servers

- org DNS servers
  - pbs.org DNS servers

- edu DNS servers
  - rutgers.edu DNS servers
  - umass.edu DNS servers
  - cs.rutgers.edu DNS server

RFC 1034
DNS Protocol

- When client wants to know an IP address for a host name
  - Client *sends* a DNS query to the primary name server in its zone

- If name server *contains the mapping*, it returns the IP address to the client

- Otherwise, the name server *forwards the request* to the root name server

- The request *works its way down the tree* toward the host until it *reaches a name server* with the correct mapping
2 DNS Query Types

**Iterated Query**
- Requesting host: cis.poly.edu
- Local DNS server: dns.poly.edu
- cs.umass.edu DNS server
- .edu DNS server
- Root DNS server
- Flow:
  1. DNS request from cis.poly.edu to dns.poly.edu
  2. dns.poly.edu sends query to .edu DNS server
  3. .edu DNS server sends query to cs.umass.edu DNS server
  4. cs.umass.edu DNS server sends query to root DNS server
  5. Root DNS server sends response to cs.umass.edu DNS server
  6. cs.umass.edu DNS server sends response to .edu DNS server
  7. .edu DNS server sends response to local DNS server
  8. Local DNS server sends response to requesting host, cis.poly.edu

**Recursive Query**
- Requesting host: cis.poly.edu
- Local DNS server: dns.poly.edu
- cs.umass.edu DNS server
- .edu DNS server
- Root DNS server
- Flow:
  1. DNS request from cis.poly.edu to dns.poly.edu
  2. dns.poly.edu sends query to .edu DNS server
  3. .edu DNS server sends query to cs.umass.edu DNS server
  4. cs.umass.edu DNS server sends query to root DNS server
  5. Root DNS server sends response to cs.umass.edu DNS server
  6. cs.umass.edu DNS server sends response to .edu DNS server
  7. .edu DNS server sends response to local DNS server
  8. Local DNS server sends response to requesting host, cis.poly.edu
Once (any) name server learns mapping, it \textit{caches} mapping
\begin{itemize}
\item cache entries \textit{timeout} (disappear) after some time
\item TLD (Top Level Domain) servers typically \textit{cached} in local name servers
  \begin{itemize}
  \item Thus root name servers not often \textit{visited}
  \end{itemize}
\end{itemize}

How does a host \textit{contact} the name server if all it has is the name and no IP address?
\begin{itemize}
\item IP address of at least 1 nameserver must be given in advance or with another protocol (DHCP, bootp)
\end{itemize}
HTTP
Web and HTTP

- **Web page consists of objects**
  - HTML file, JPEG image, Java applet, audio file, ...

- Web page consists of **a base HTML-file** which includes several referenced objects

- Each object is addressable by a **URL** (Uniform Resource Locator)

- Example URL:
  - `www.cs.rutgers.edu/undergraduate/pic.gif`
    - **host name**: `www.cs.rutgers.edu`
    - **path name**: `/undergraduate/pic.gif`
HTTP overview

- Web page consists of a base HTML-file which includes several referenced objects addressable by a URL
- Client/Server model
  - **client**: browser that requests, receives, “displays” Web objects
  - **server**: Web server sends objects in response to requests
- Request Message
- Response Message
Additional Topics about HTTP

- Persistent vs. Nonpersistent HTTP connections
- Cookies (User-server state)
- Web caches
# HTTP connections

## Nonpersistent HTTP
- At most one object is sent over a single TCP connection.

## Persistent HTTP
- Multiple objects can be sent over a single TCP connection between client and server.

### Nonpersistent HTTP issues:
- Requires 2 RTTs per object
  - TCP Connection and HTTP Request
- Browsers can open parallel TCP connections to fetch referenced objects

### Persistent HTTP
- Server leaves TCP connection open after sending response
- Subsequent HTTP messages sent over open connection
Cookie: User-server State

HTTP is “stateless”

- server maintains no information about past client requests

- What state can bring:
  - Authorization, shopping carts, recommendations, user session state

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
Web caches (proxy server)

- Reduce **response time** for client request.
- Reduce **traffic** on an institution’s access link.

- **browser sends all HTTP requests to cache**
  - Miss: cache **requests** object from origin server, then returns object to client
  - Hit: cache **returns** object

- **guarantees cache content is up-to-date**

- **saves** traffic and response time whenever possible
CDN
Content Distribution Networks (CDN)

- Reduce bandwidth Requirement & Traffic of content provider
- Reduce $$ of maintaining Servers
- Improve response time to user

Without CDN

Huge B/W requirements & Does not scale

With CDN

Using CDN
FTP
FTP: the file transfer protocol

- transfer file to/from remote host
- client/server model
  - client: side that initiates transfer (either to/from remote)
  - server: remote host
- “out of band” control
  - Control connection port 21 & Data connection port 20
- Active connection: data connection initiated form server
- Passive connection: data connection initiated form client
- Key Drawback: Sends passwords in plain ASCII text
- Replaced with scp, sftp instead
SMTP
Electronic Mail

Three Components:

1. User Agents

2. Mail Servers
   - mailbox contains incoming messages for user
   - message queue of outgoing (to be sent) mail messages

3. SMTP protocol
   - Used to send messages
   - Client: sending user agent or sending mail server
   - server: receiving mail server

Mail access protocol: retrieval from server
   - POP: Post Office Protocol
   - IMAP: Internet Mail Access Protocol
   - HTTP: Hotmail, Yahoo! Mail, etc.
Questions?
Summary: App-layer protocol defines

- Types of messages exchanged,
  - e.g., request, response

- Message format:
  - Syntax: what fields in messages
  - Semantics: meaning of information in fields

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

Public-domain protocols:
- defined in RFCs
- allows for interoperability
- DNS, HTTP, SMTP

Proprietary protocols:
- e.g., Skype, Hangout
Client-server architecture (CS)

server:
- always-on host
- permanent IP address
- server farms for scaling

clients:
- communicate with server
- may not be always connected
- may have dynamic IP addresses
- do not communicate directly with each other
Network Application

- 2 applications on 2 hosts connected by a network

- In order to communicate, need to identify each host

- Computer network: IP address
  - IPv4 (32 bits) 128.6.24.78

- In addition to host address, we need port #
- More than one program executing on a host
App-layer protocols

- DNS: Domain Name Service
- HTTP: HyperText Transfer Protocol
- FTP: File Transfer Protocol
- SMTP: Simple Mail Transfer Protocol
Themes

- Request/response nature of these protocols

- How Messages are structured
  - HTTP, SMTP, FTP - simple ASCII protocols

- Caching: Reducing Request/Response Msgs

- Name Lookup
  - Division of concerns (e.g. zones)
  - Hierarchy structure
Services provided by lower layers
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)