Application Protocols

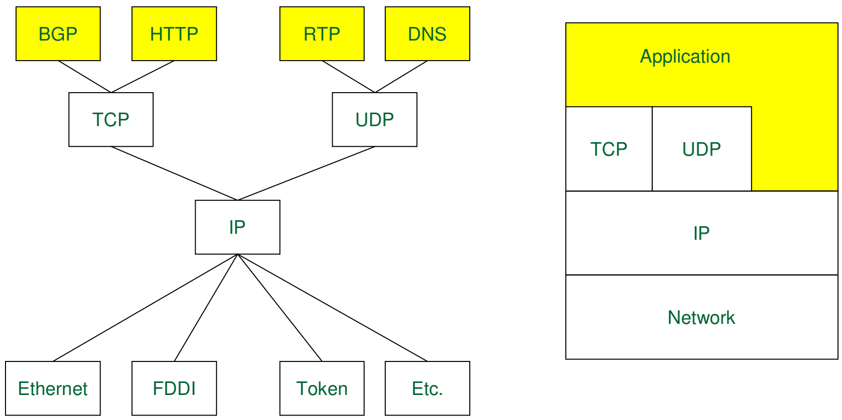
EECS 122: Lecture 6

Department of Electrical Engineering and Computer Sciences
University of California
Berkeley

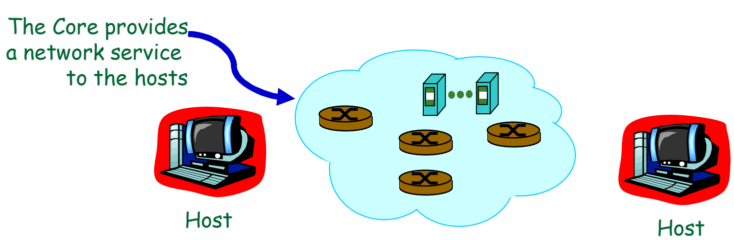
Today

- Adminstrivia
- The last two lectures have exposed you to building programs and simulations of networks
- Today we focus on specific applications and protocols
  - DNS
  - HTTP
  - SMTP
- Lots of details but focus on the concepts…

Where do Application Protocols Run?



Where do Application Protocols Run?



- Host-Host:
  - HTTP, SMTP
- Host-Network:
  - DNS
- Network-Network:
  - Routing Protocols (e.g. OSPF)

Internet transport protocols services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* 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

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Vonage,Dialpad)	typically UDP

Domain Name Service

- Resolves a host name names into an IP address
- Why host names?
  - To organize machines
    - Eg. robotics.eecs.berkeley.edu
    - This conveys more information to humans than 128.32.48.234
- Why IP addresss?
  - The network needs an address to route
- Host Names yield information to people and IP addresses yield information to routers

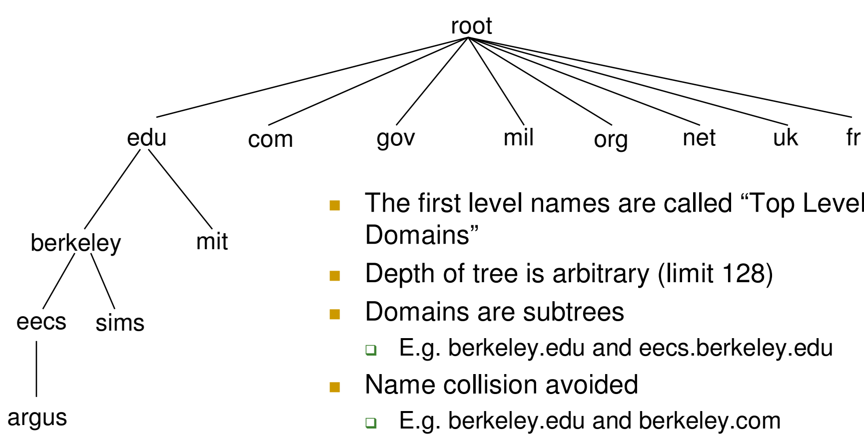
DNS: History

- Initially all host-addess mappings were in a file called hosts.txt (in /etc/hosts)
  - Changes were submitted to SRI by email
  - New versions of hosts.txt were ftp'd periodically from SRI
  - An administrator could pick names at their discretion
- As the internet grew this system broke down because
  - SRI couldn't handled the load
  - The system was unreliable since there was a single point of contact
  - Names were not unique
  - Many hosts had inaccurate copies of hosts.txt
- Internet growth was threatened!

DNS Features

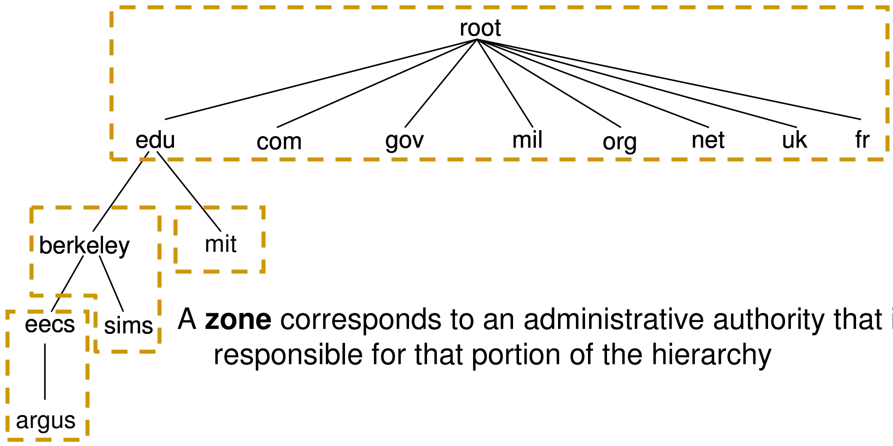
- Hierarchical Namespace
- Distributed architecture for storing names
  - Nameservers assigned zones of the hierarchical namespace
  - Backup servers available for redundancy
- Administration divided along the same hierarchy
  - DNS client is simple: Resolver
- Client server interaction on UDP Port 53 (but can use TCP if desired)

Hierarchical Namespace



- The first level names are called "Top Level Domains"
- Depth of tree is arbitrary (limit 128)
- Domains are subtrees
  - E.g. berkeley.edu and eecs.berkeley.edu
- Name collision avoided
  - E.g. berkeley.edu and berkeley.com

Hierarchical Administration



A **zone** corresponds to an administrative authority that is responsible for that portion of the hierarchy

Hierarchical Server Organization

- Each server has authority over a portion of the hierarchy
  - A server maintains only a subset of all names
- Each server contains all the records for the hosts in its zone
- Each server needs to know other servers that are responsible for the other portions of the hierarchy
  - Every server knows the root
  - Root server knows about all top-level domains

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

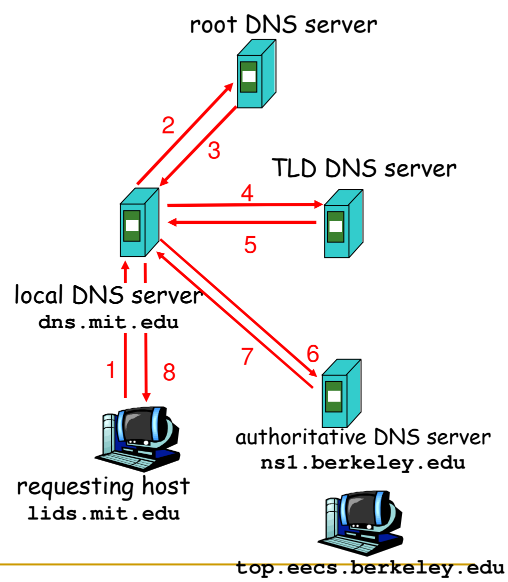
- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
  - Also called "default name server"
- When a host makes a DNS query, query is sent to its local DNS server
  - Acts as a proxy, forwards query into hierarchy.

How does a name get resolved

- Query "walks" its way up and down the hierarchy
  - Iterated query
    - I don't know, but here's who to ask next
  - Recursive query
    - I don't know right now, but I'll get back to you…

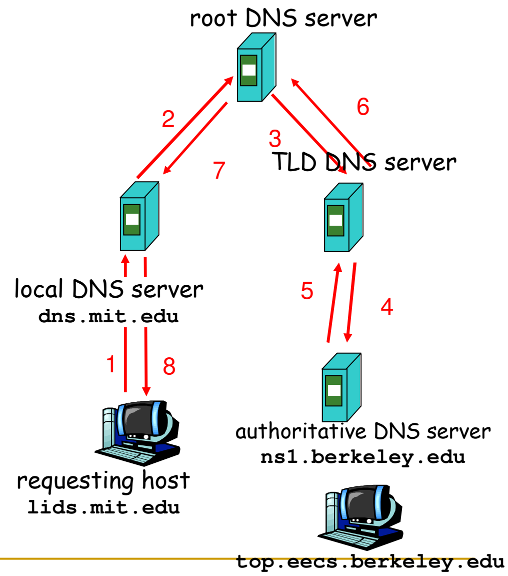
Iterated Query

- Host at lids.mit.edu wants IP address for `top.eecs.berkeley.edu`.
- "I don't know, but here's who to ask next"



Recursive Query

- Host at lids.mit.edu wants IP address for `top.eecs.berkeley.edu`.
- I don't know right now, but I'll get back to you…



DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- update/notify mechanisms under design by IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

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 hostname of authoritative name server for this domain
- Type=CNAME
  - **`name`** is alias name for some "canonical" (the real) name
    `www.ibm.com` is really `servereast.backup2.ibm.com`
  - **`value`** is canonical name
- Type=MX
  - **`value`** is name of mailserver associated with **`name`**

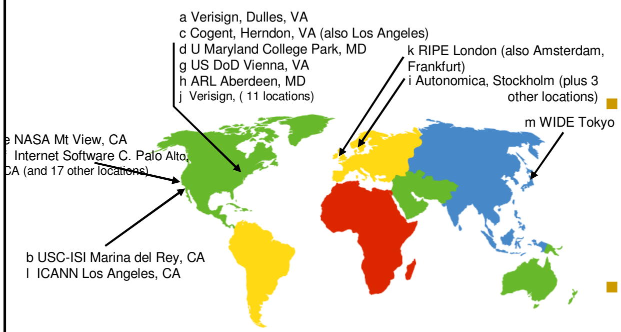
Inserting records into DNS

- Example: just created startup "Network Utopia"
- Register name networkuptopia.com at a registrar (e.g., Network Solutions)
  - Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts two RRs into the com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- Put in authoritative server Type A record for www.networkuptopia.com and Type MX record for networkutopia.com

Robustness and Security



{A,..,M}.Root-Servers.Net

- For non-root severs multiple servers are common as well
- Caching provides another form of redundancy and quicker response time
- DOS attack in October 2002
- Secure DNS

DNS and Virtual IP addresses

- DNS records don't have to store the real IP address of the host
- All hosts in the acme.com may have the same IP address
  - A firewall at this IP address decides whether to "admit" a transport level connection (firewall) to the host x.acme.com
  - A load balancer decides to forward the connection to one of several identical servers
  - In both cases, the gateway must use a local lookup to decide which end host to direct the connection
- Redirection to be to anywhere! Even another country.
- Allows for distributed caching architectures
- Makes tracking the geographic location of a name very difficult

Example: www.akamai.com

- From Berkeley

```
C:\>ping www.akamai.com
Pinging a1440.g.akamai.net [64.164.108.148] with 32 bytes of data:
Reply from 64.164.108.148: bytes=32 time=10ms TTL=249
Reply from 64.164.108.148: bytes=32 time=10ms TTL=249
Reply from 64.164.108.148: bytes=32 time=10ms TTL=249
Reply from 64.164.108.148: bytes=32 time=20ms TTL=249

Ping statistics for 64.164.108.148:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 10ms, Maximum =  20ms, Average =  12ms
```

- From the NY Area
  - 63.240.15.146
- From the UK
  - *194.82.174.224*

Examples

DNS Summary

- DNS is a crucial part of the internet
- Namespace is hierarchical
- Administration is distributed
- It is vulnerable in various ways but no more than other parts of the internet infrastructure
- Its performance is enhanced by caching
- DNS “Hacks” can enable many interesting things

The WWW

- A distributed database of URLs
- Core components:
  - Servers which store files and execute remote commands
  - Browsers retrieve and display “pages” of content linked by hypertext
    - Each link is a URL
- Can build arbitrarily complex applications, all of which share a uniform client!
- Need a protocol to transfer information between clients and servers
  - HTTP

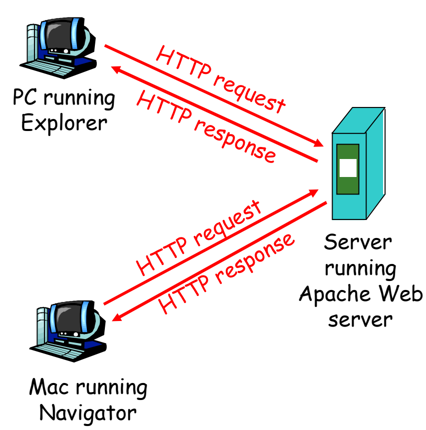
Uniform Record Locator

- `protocol://host-name:port/directory-path/resource`
- Extend the idea of hierarchical namespaces to include anything in a file system
  - ftp://www.eecs.berkeley.edu/122/Lecture6/presentation.ppt
- Extend to program executions as well…
  - http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b
  - Server side processing can be incorporated in the name

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



HTTP overview (continued)

Uses TCP:

- 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

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

FTP

HTTP request message

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

request line (GET, POST, HEAD commands)

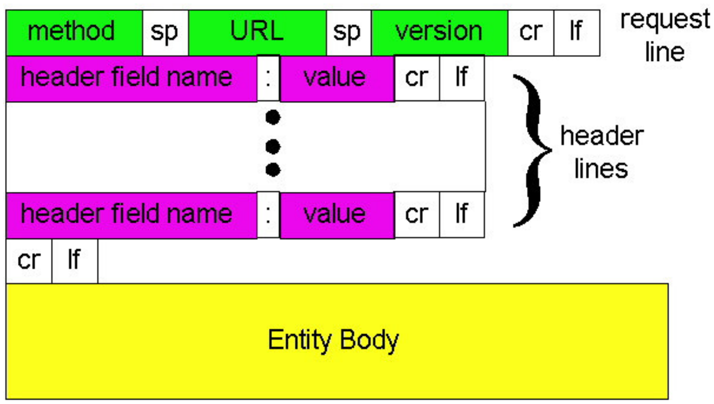
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

HTTP request message: general format



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
  - asks server to leave requested object out of 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 response message

status line (protocol status code status phrase)

```
HTTP/1.1 200 OK
```

header lines

```
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 ...
```

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`

Persistence

- A web page typically contains many objects
  - E.g. Images
  - Each object must be requested with a separate http "Get" command
  - Non Persistent Connection:
    - Different __TCP__ connection for each object request.
    - HTTP 1.0
  - Persistent Connection
    - Reuse the same TCP connection for each object request
    - HTTP 1.1

HTTP/1.0 Performance

- Create a new TCP connection for each resource
  - Large number of embedded objects in a web page
  - Many short lived connections
- Requires 2 RTTs per object
- TCP transfer
  - Too slow for small object
  - May never exit slow-start phase
- Connections may be set up in parallel (5 is default in most browsers)
- OS overhead for *each* TCP connection

Persistent HTTP

- server leaves connection open after sending response
  - TCP overhead minimized
- subsequent HTTP messages between same client/server sent over open connection

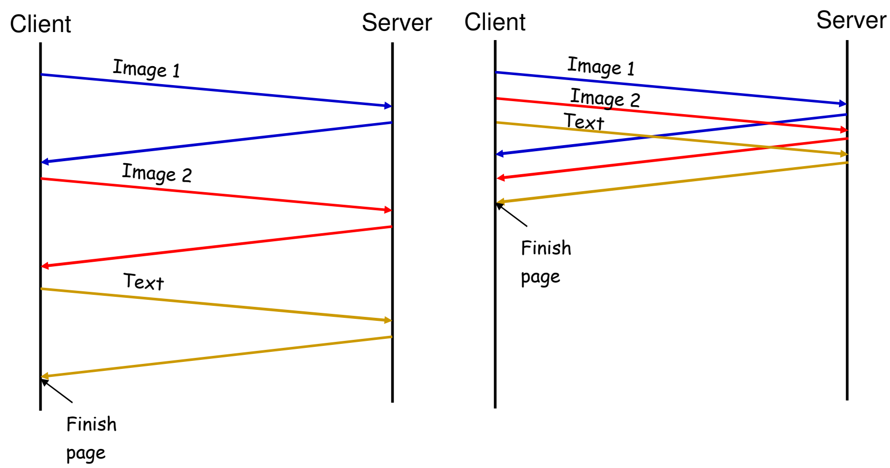
No pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Pipelining:

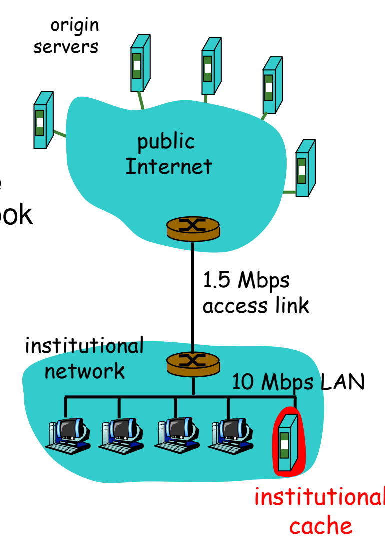
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects
- default in HTTP/1.1

The Advantage of Pipelining



Caching

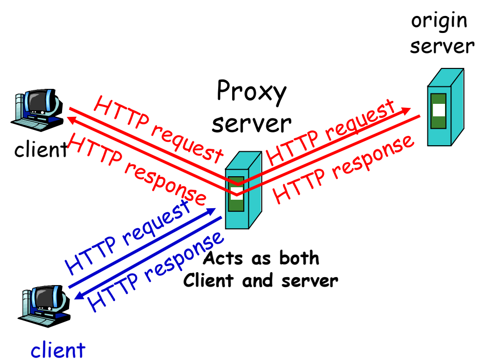
- Store frequently referenced objects closer to the clients
  - Saves Time: No need to go all the way to the server (access could look "instantaneous")
  - Saves Access Bandwidth
  - Saves Web Server Resources
- Limitations?
  - Frequently changing objects
  - Hit counts
  - Privacy



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

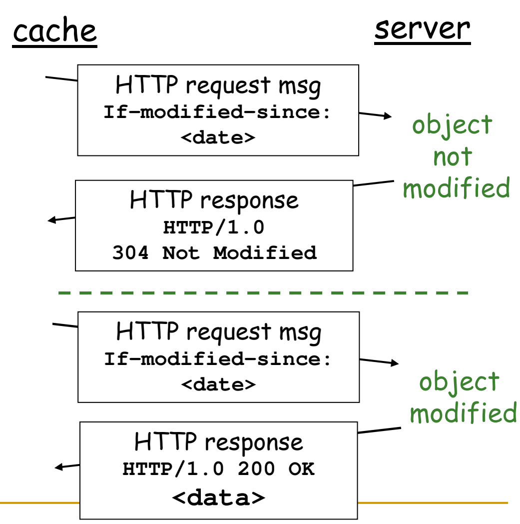


Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- cache: 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



Other Web Proxy Functions

- Filter requests/responses
- Modify requests/responses
  - Change http requests to ftp requests
  - Change response content, e.g., transcoding to display data efficiently on a Palm Pilot
- Provide better privacy

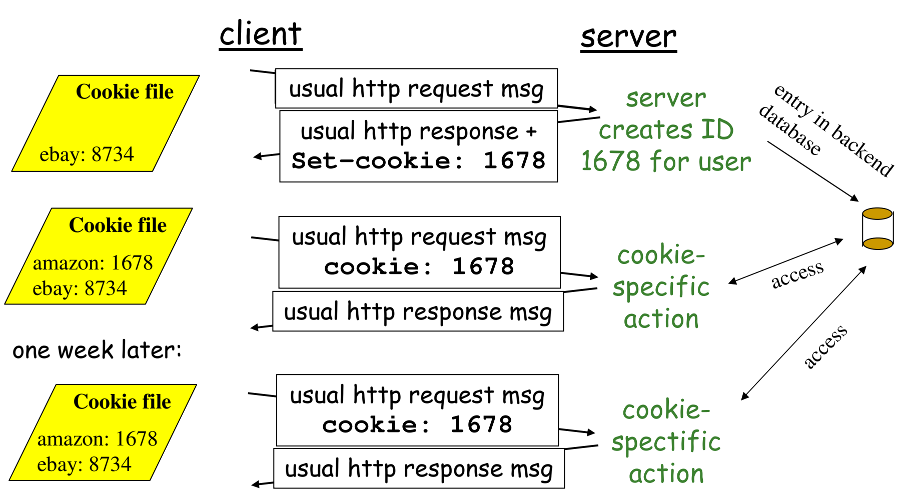
Cookies: An Example of How Applications Add State

- When you are shopping at website
  - How does the merchant to track what you are browsing?
  - When you were at the site last?
  - Suppose you don't login…
- An ad network wants to make sure it doesn't keep showing you the same ad on each site that you visit
- Browsers help implement these functions by allowing a webserver to maintain state on your computer
  - This state is called a Cookie!

Cookies

- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID
- 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
- Additional Cookie Functions
  - authorization
  - shopping carts
  - recommendations
  - user session state (Web e-mail)

Cookies: keeping "state" (cont.)



Other interesting state creating examples

- Annoying
  - Spyware
  - Viruses
- Potentially useful
  - Client-side scripting
    - E.g. Ajax: **A**synchronous **J**avaScript **A**nd **X**ML

HTTP and DNS

- Both
  - are client – server applications
  - have decentralized management
  - enable access to vast amounts of distributed information
  - are based on open protocols
  - are distributed databases
- But
  - Http runs on TCP and DNS on UDP
  - Http runs between two end hosts, whereas DNS is part of the network infrastructure

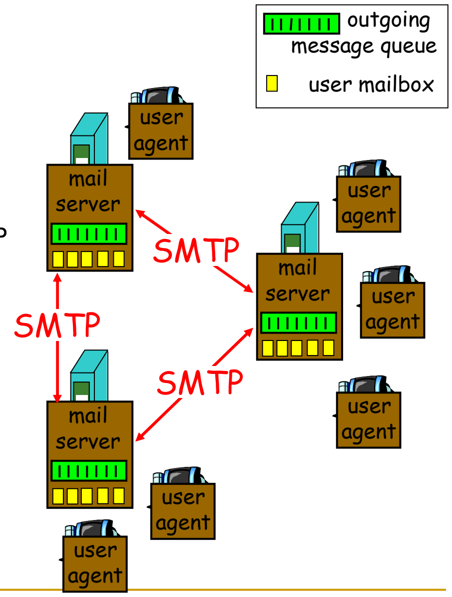
Electronic Mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

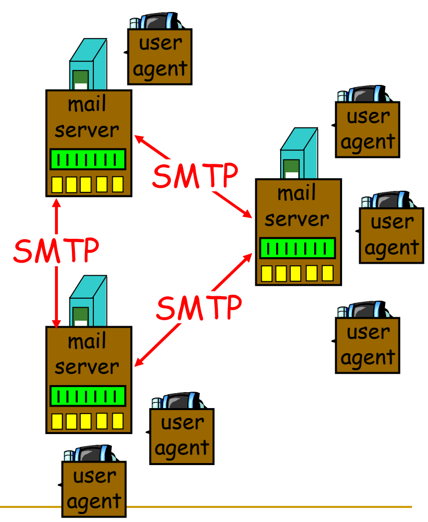
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Netscape Messenger
- outgoing, incoming messages stored on server



Electronic Mail: mail servers

Mail Servers

- mailbox contains incoming messages for user
- message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
  - client: sending mail server
  - “server”: receiving mail server

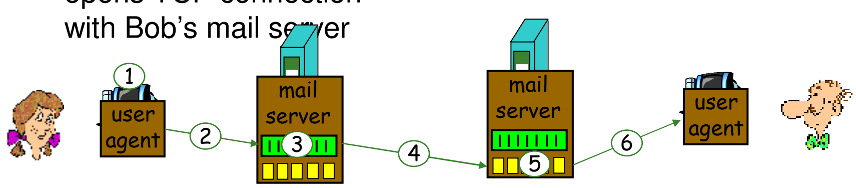


Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII

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

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```


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)

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message

Comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

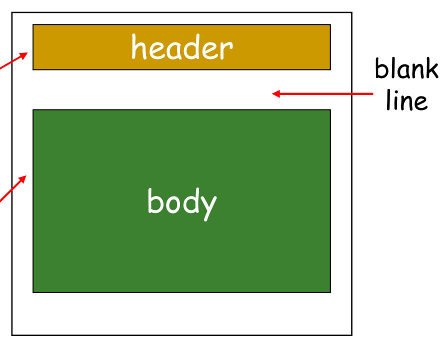
Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

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

  different from SMTP commands!
- 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

MIME version

method used to encode data

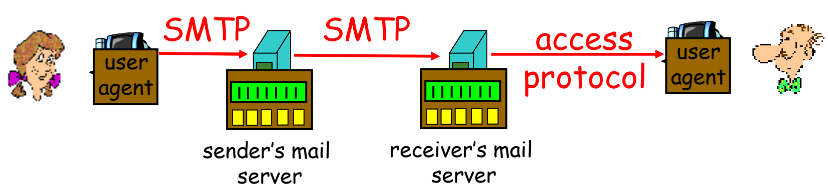
multimedia data type, subtype, parameter declaration

encoded data

```
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
```


Mail access protocols

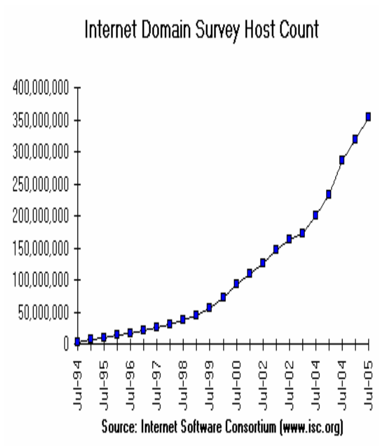


- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: Hotmail , Yahoo! Mail, etc.

What did we learn today?

- Application Protocols utilize the transport protocol to make the internet useful
- Examples: DNS, HTTP, SMTP
- Some concepts:
  - Connection Persistence
  - Caching
  - How applications add client state via the browser
- Remember: The goal in this class is not master any one application protocol but to understand the concepts that make them ultra scalable and useful

Conclusion



- The applications we discussed today are not complex but they have had huge global impact
- Simplicity, trust in distributed control and open standards helped make this happen.