The Application Layer: HTTP, SMTP

Lecture 5

http://www.cs.rutgers.edu/~sn624/352-S22

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Quick recap of concepts

Domain Name System

128.45.10?.??

Distributed database
of name to IP addr
mappings.

Entry types in the database:
resource records. A, NS, MX, AAAA

HyperText Transfer Protocol (HTTP)

URL: a resource or process

$\underbrace{\texttt{mail.google.com}}_{\text{Host name}}\underbrace{\texttt{/inbox}}_{\text{Path name}}$

HTTP is a client/server application

Methods: GET/POST/…

Headers
User-agent/server/…

Response codes: 200, 404, etc.

Web Server

This lecture: more about HTTP!

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

HTTP Persistence

HTTP connections

Non-persistent HTTP

- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent 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

TCP is a kind of reliable communication service provided by the transport layer. It requires some resources for the connection to be set up at the endpoints before data communication.

Non-persistent HTTP (HTTP/1.0)

Web Server

Suppose user visits a page with text and 10 images.

1a. HTTP client initiates TCP connection to HTTP server

1b. HTTP server at host “accepts” connection, notifying client

2. HTTP client sends HTTP request message

3. HTTP server receives request message, replies with response message containing requested object

time

Non-persistent HTTP (HTTP/1.0)

Web Server

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

time

6. Steps 1-5 repeated for each of 10 jpeg objects

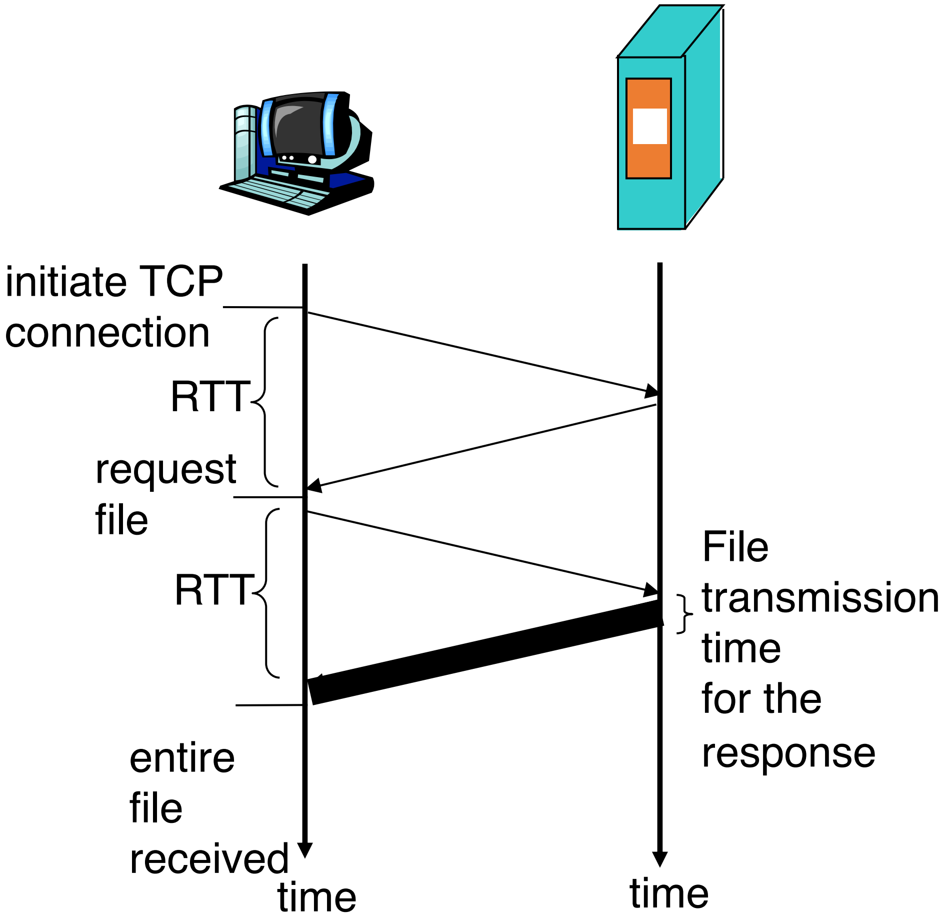
Single connection per object

Useful at a time when web pages contained 1 object: the base HTML file.

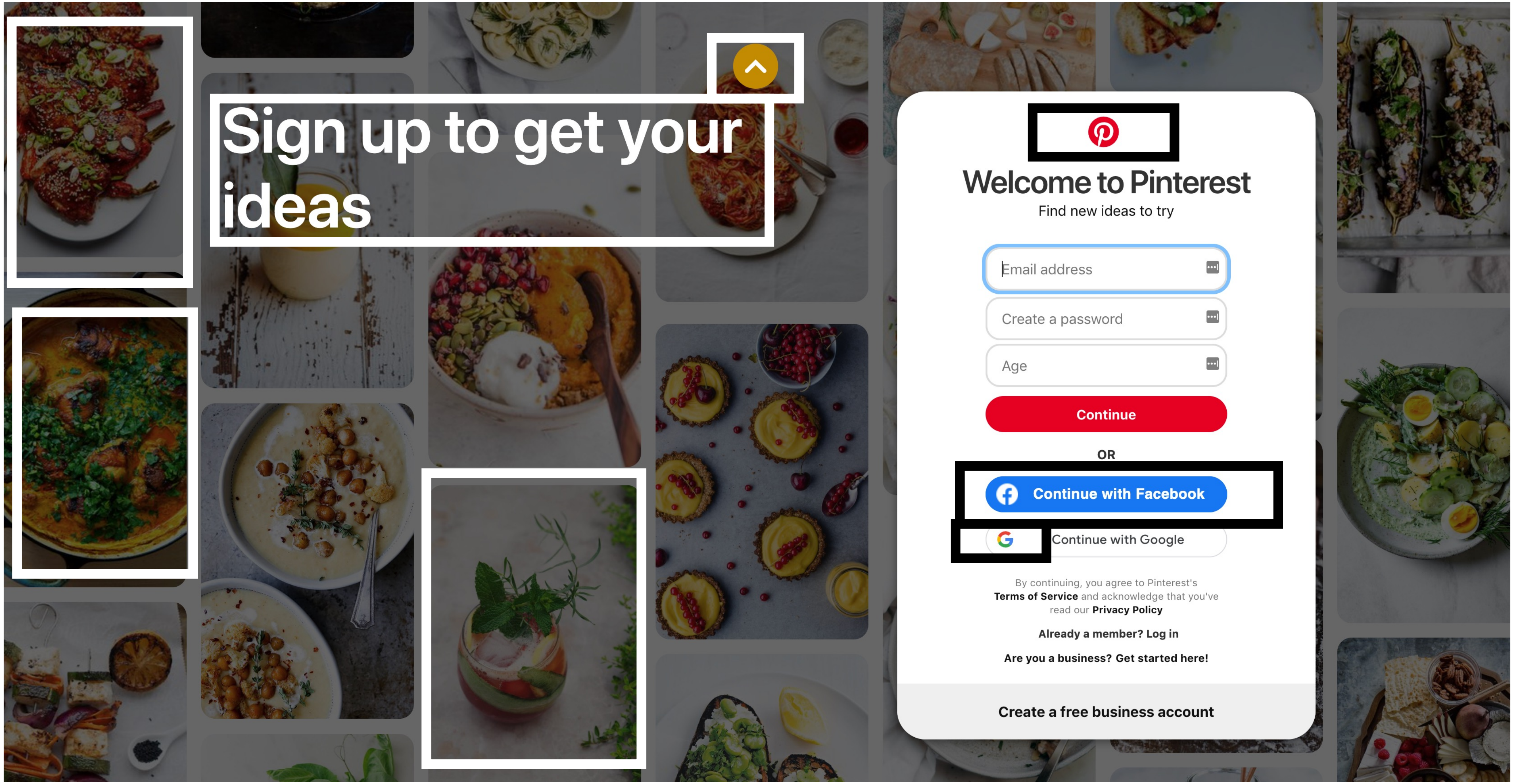
How long does it take to transfer an object with non-persistent HTTP?
i.e.: before your browser can load the (entire) object?

Non-persistent HTTP transfer time

- Total delay = propagation + queueing + transmission
- Response time for user
  - = total round-trip delay
  - = sum of forward and backward total delays
- Total round-trip delay for a "small" packet called a Round Trip Time (RTT)
  - Round-trip delays with zero transmission delay
- Assumptions:
  - Small packets: TCP initiation packet, response, HTTP request are all small
  - No processing delays at the server
  - RTT stable over time
- 2RTT + file transmission time *per object*



Per-object overheads quickly add up



Modern web pages have 100s of objects in them.

Persistent HTTP (HTTP/1.1)

Suppose user visits a page with text and 10 images.

Web Server

1a. HTTP client initiates TCP connection to HTTP server

1b. HTTP server at host "accepts" connection, notifying client

2. HTTP client sends HTTP request message

3. HTTP server receives request message, replies with response message containing requested object

`Connection: keep-alive` HTTP header introduced in HTTP/1.1

time

Persistent HTTP (HTTP/1.1)

Web Server

4. HTTP server sends a response.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

Server keeps the TCP connection alive.

time

The 10 objects can be requested over the same TCP connection.

i.e., save an RTT per object (otherwise spent opening a new TCP connection in HTTP/1.0)

Persistence vs. # of connections

- Persistence is distinct from the number of concurrent connections made by a client
- Your browser has the choice to open multiple connections to a server!
  - Bounded by the HTTP specification to a small number (e.g., 5)
- Further, a single connection can have multiple object (HTTP) requests in flight with persistent HTTP

Remembering Users On the Web

HTTP: Remembering users

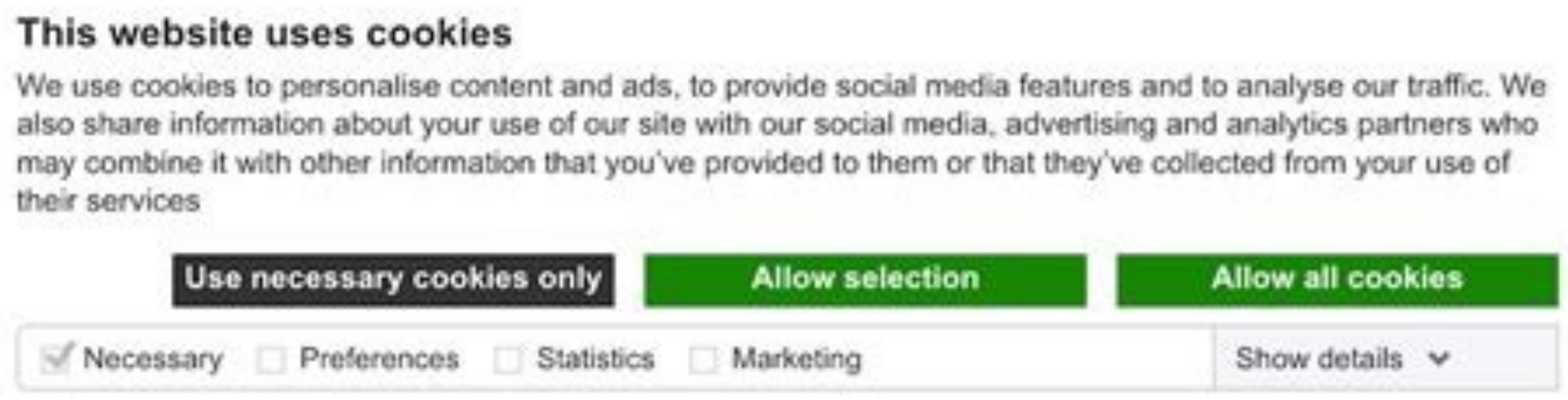
So far, HTTP mechanisms considered stateless

- Each request processed independently at the server
- The server maintains no memory about past client requests

However, state, i.e., memory, about the user at the server, is very useful!

- User authentication (e.g., gmail)
- Shopping carts (e.g., Amazon)
- Video recommendations (e.g., Netflix)
- Any user session state in general

Familiar with these?



Cookies: Keeping user memory

client

server

Cookie is typically opaque to client.

Cookie file
Netflix: 436
Amazon: 1678

http request msg + auth

server creates ID 1678 for user

entry in backend database

http response +
Set-cookie: 1678

Cookie file
Netflix: 436
Amazon: 1678

http request (no auth)
cookie: 1678

cookie-specific action

access

Personalized http response

one week later:

Cookie file
Netflix: 436
Amazon: 1678

http request (no auth)
cookie: 1678

cookie-specific action

access

Personalized http response

How cookies work

Collaboration between client and server to track user 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 endpoint, managed by user's browser
4. back-end database maps cookie to user data at Web endpoint

Cookies come with an expiration date (yet another HTTP header!)

Cookies have many uses

- The good: Awesome user-facing functionality
  - Shopping carts, auth, … very challenging or impossible without it
- The bad: Unnecessary recording of your activities on the site
  - First-party cookies: performance statistics, user engagement, …
- The ugly: Tracking your activities across the Internet
  - Third-party cookies (played by ad and tracking networks) to track your activities *across the Internet.*
  - Potentially *personally identifiable information (PII)*
  - Ad networks target users with ads, may sell this info
  - Scammers can target you too!

PSA: Cookies and Privacy

- Disable and delete unnecessary cookies by default

- Suggested privacy-conscious browsers, websites, tools:

- DuckDuckGo (search)
- Brave (browser)
- AdBlock Plus (extension)
- ToR (distract targeting)
- … assuming it doesn't break the functions of the site.

https://gdpr.eu/cookies/

Caching in HTTP

Web caches

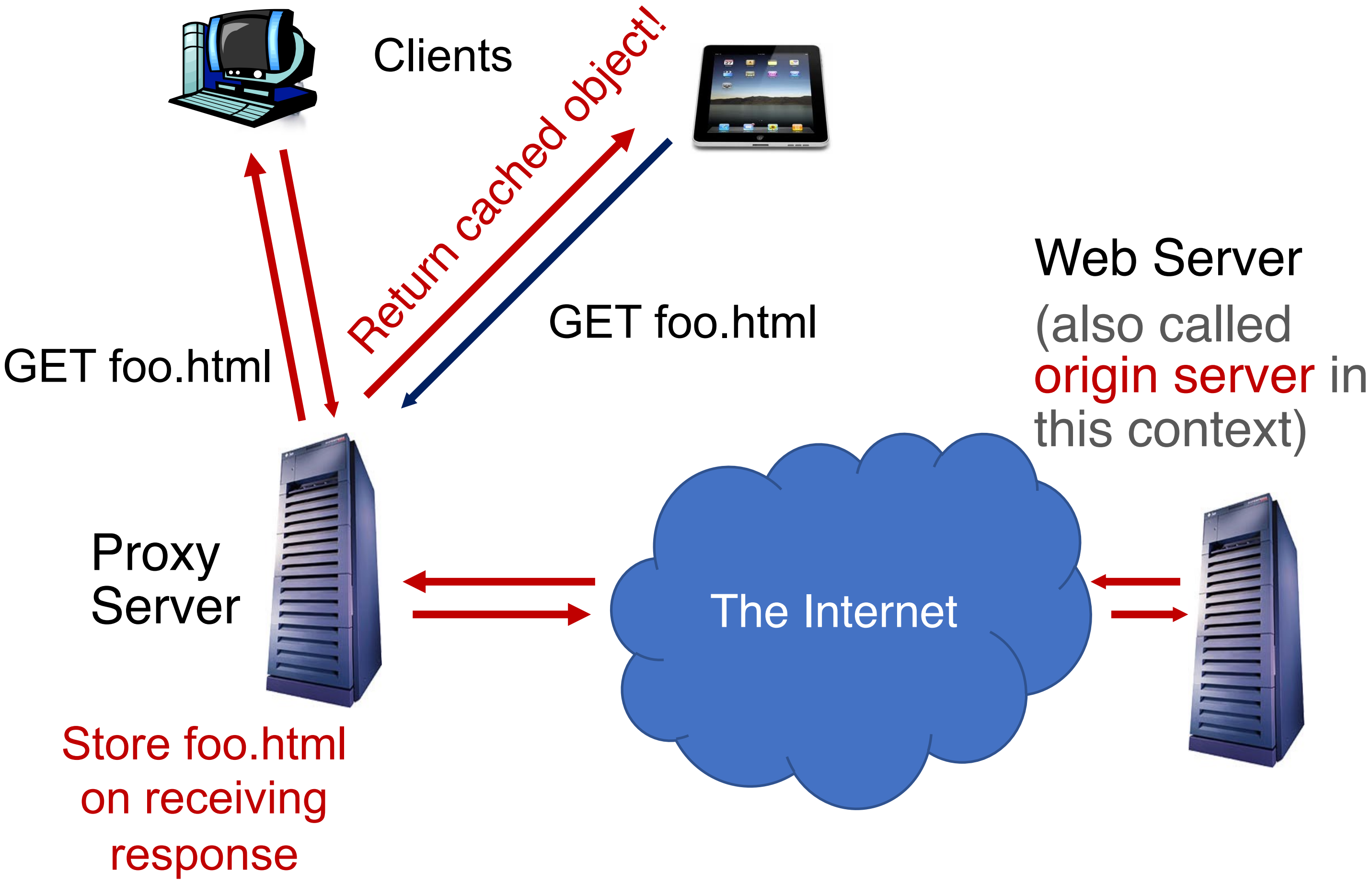
Web caches: Machines that remember web responses for a network

Why cache web responses?

- Reduce response time for client requests
- Reduce traffic on an institution’s access link

Caches can be implemented in the form of a proxy server

Web caching using a proxy server

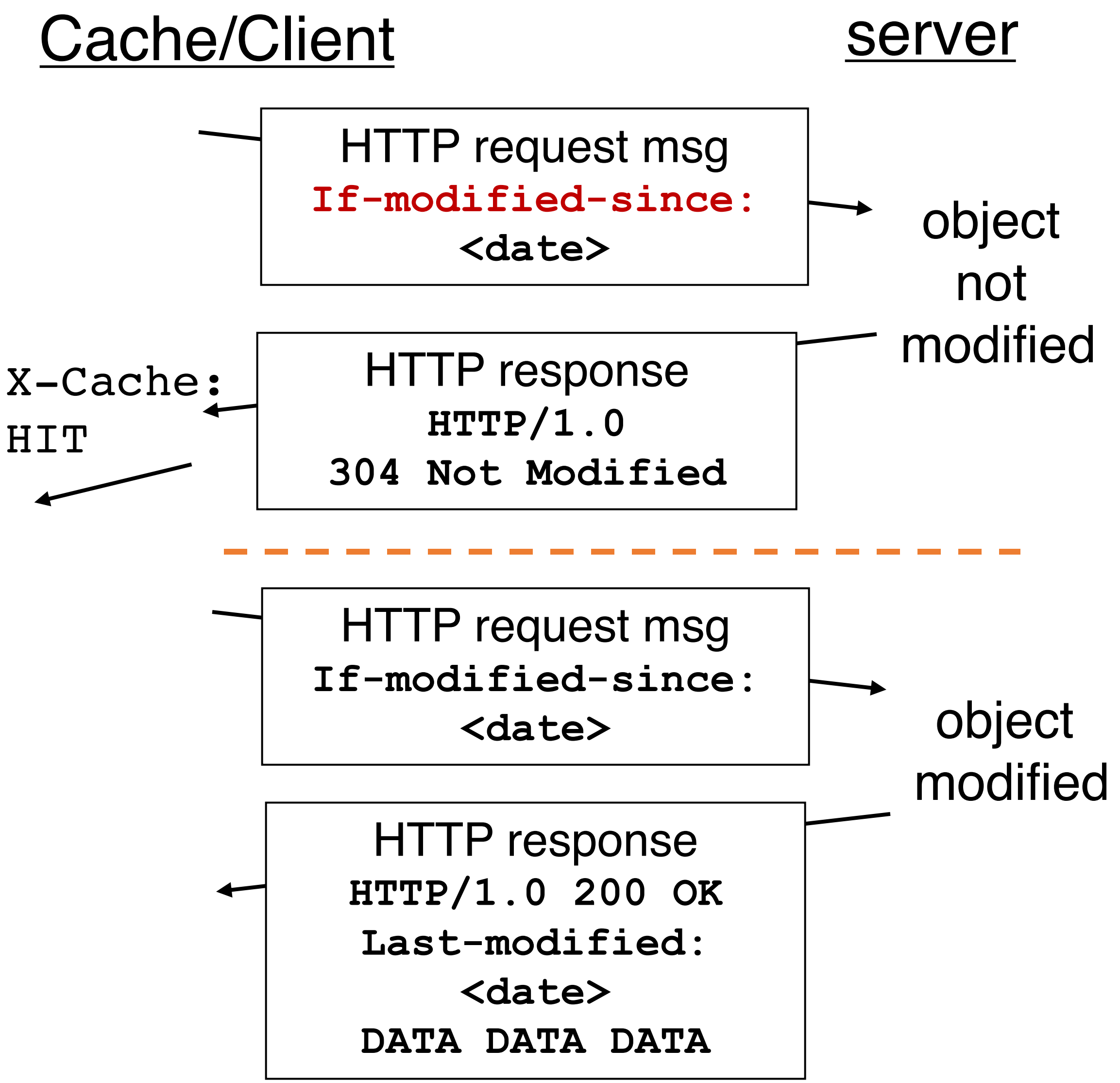


- You can configure a HTTP proxy on your laptop's network settings.
- If you do, your browser sends all HTTP requests to the proxy (cache).
- Hit: cache returns object
- Miss: obtain object from originating web server (origin server) and return to client
  - Also cache the object locally

Caching in the HTTP protocol

- Conditional GET guarantees cache content is up-to-date while still saves traffic and response time whenever possible

- Date in the cache's request is the last time the server provided in its response header Last-Modified



Content Distribution Networks (CDNs)

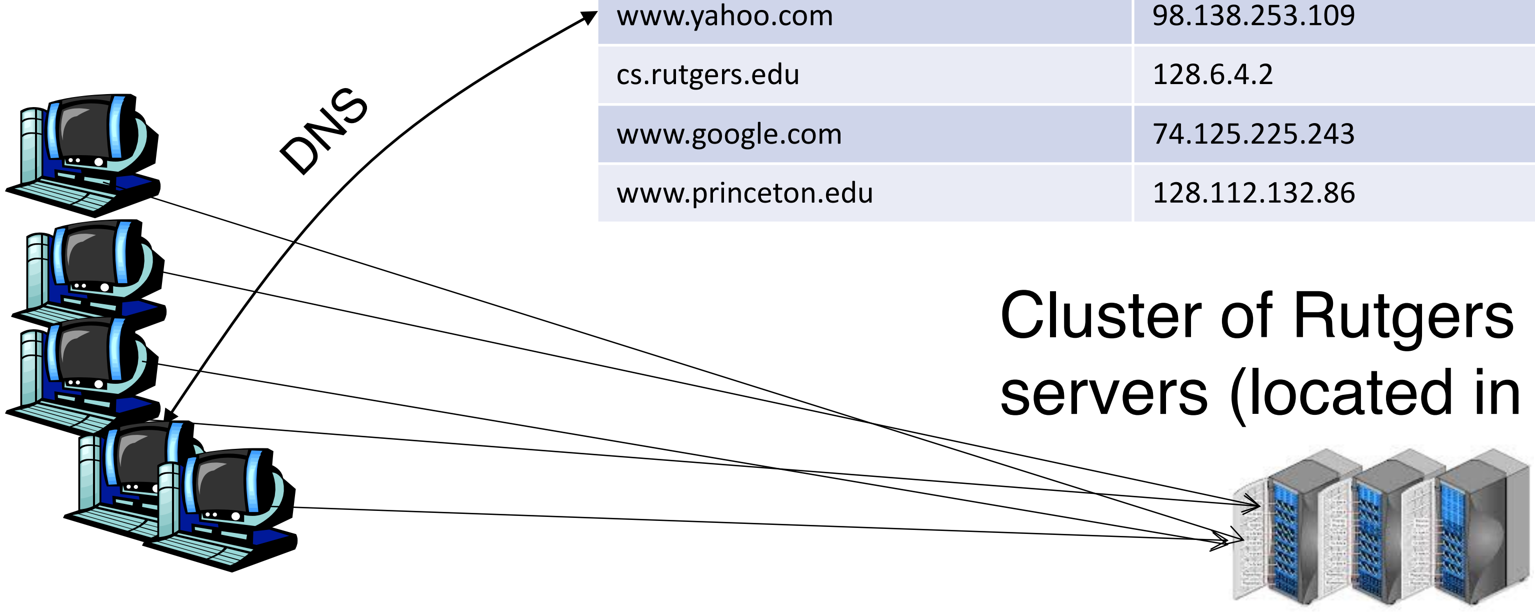
A global network of web caches
- Provisioned by ISPs and network operators
- Or content providers, like Netflix, Google, etc.

Uses (overlaps with uses of web caching in general)
- Reduce traffic on a network's Internet connection, e.g., Rutgers
- Improve response time for users: CDN nodes are closer to most users than origin servers
- Reduce bandwidth requirements on content provider
- Reduce $$ to maintain origin servers

Without CDN

DOMAIN NAME	IP ADDRESS
www.yahoo.com	98.138.253.109
cs.rutgers.edu	128.6.4.2
www.google.com	74.125.225.243
www.princeton.edu	128.112.132.86

Clients distributed all over the world



Cluster of Rutgers CS origin servers (located in NJ, USA)

128.6.4.2

- Problems:
- Huge bandwidth requirements for Rutgers
- Large propagation delays to reach users

Where the CDN comes in

- Distribute content of the origin server over geographically distributed CDN servers
- But how will users get to these CDN servers?
- Use DNS!
  - DNS provides an additional layer of indirection
  - Instead of domain -> IP addr, use domain -> DNS server (NS record!)
- The CDN runs its own DNS servers (CDN name servers) to perform this redirection
  - Send users to the "closest" CDN web server for a given domain

With CDN

DOMAIN NAME	IP ADDRESS
www.yahoo.com	98.138.253.109
cs.rutgers.edu	124.8.9.8 (NS record pointing to CDN name server)
www.google.com	74.125.225.243

DNS reply

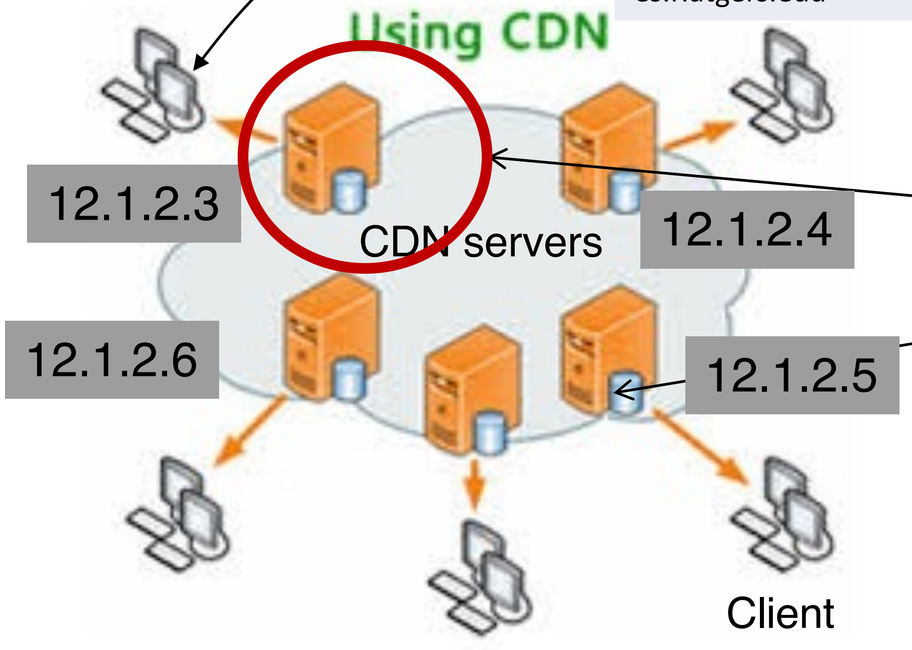
NS record delegates the choice of IP address to the CDN name server.

CDN Name Server (124.8.9.8)

DOMAIN NAME	IP ADDRESS
Cs.Rutgers.edu	12.1.2.3
Cs.Rutgers.edu	12.1.2.4
Cs.Rutgers.edu	12.1.2.5
Cs.Rutgers.edu	12.1.2.6

Custom logic to map ONE domain name to one of many IP addresses!

Popular CDNs:
CloudFlare
Akamai
Level3
…



128.6.4.2
Origin server

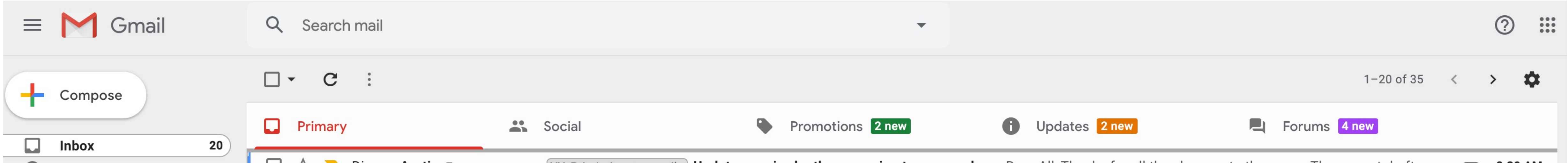
Most requests go to CDN servers (caches).
CDN servers may request object from origin
Few client requests go directly to origin server

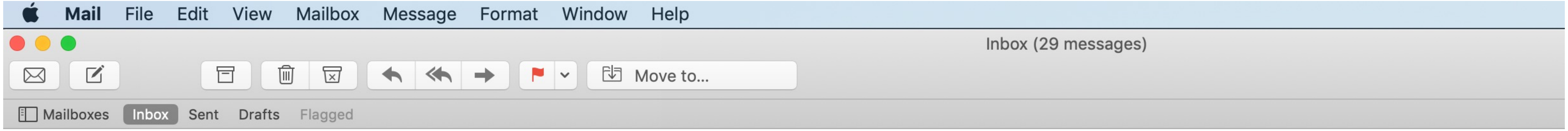
Summary of HTTP

- Request/response protocol
- ASCII-based human-readable message structures
- Improve performance using connection persistence, caching, and CDN
- Enhanced stateful functionality using cookies
- Simple, highly-customizable protocol (just add headers)
- Protocol that forms of the basis of the web we enjoy today!

Simple Mail Transfer Protocol

We’re all familiar with email.
How does it work?

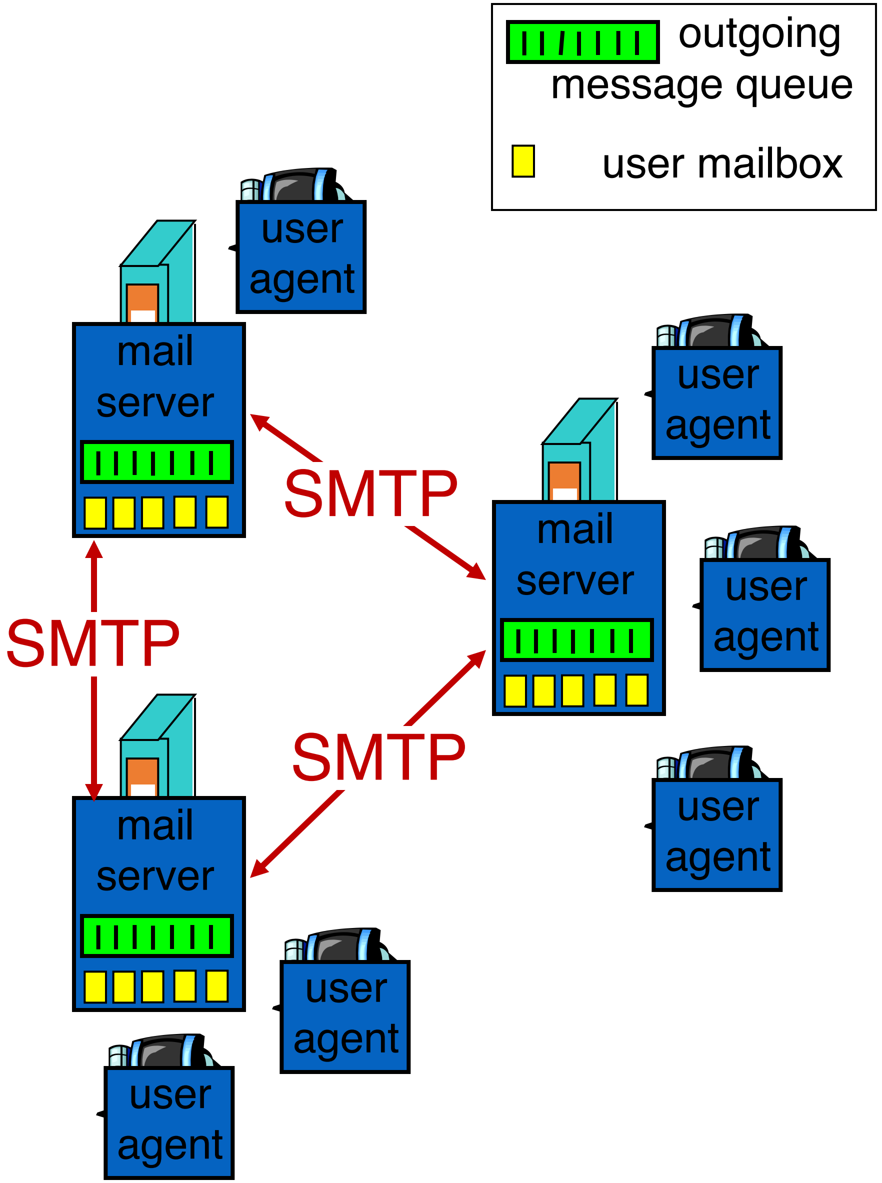




Electronic Mail

Three major components:

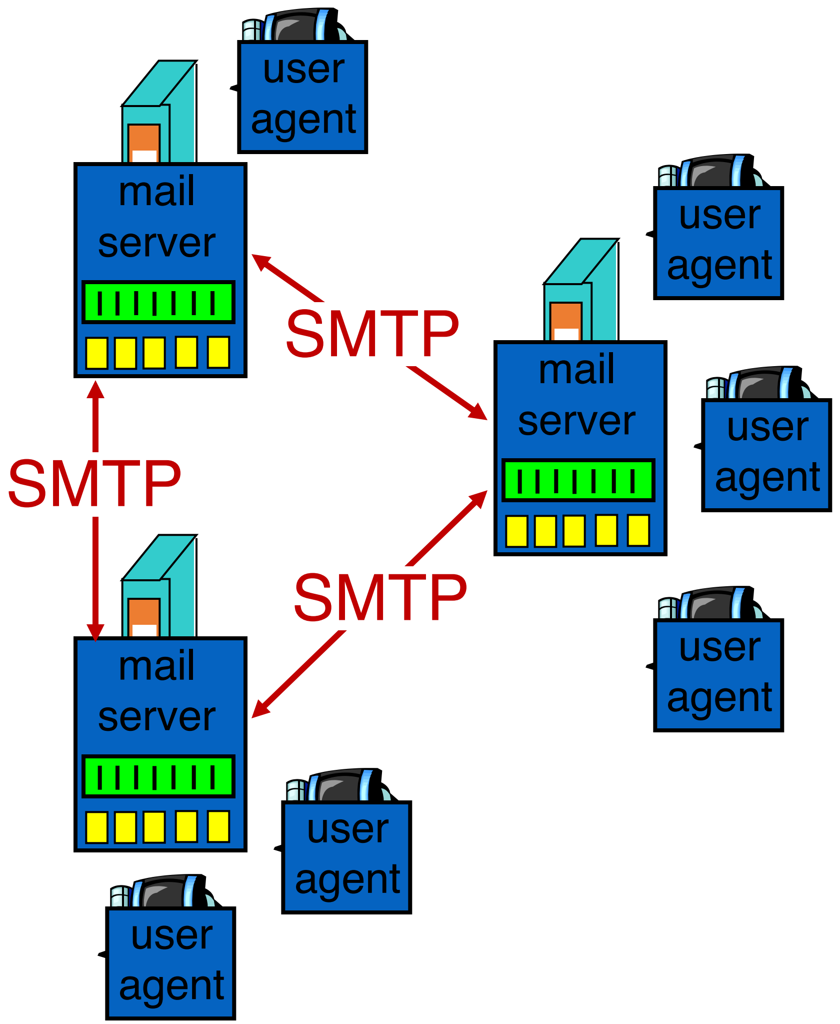
1. User agents
   - a.k.a. “mail reader”
   - e.g., Applemail, Outlook
   - Web-based user agents (ex: gmail)



Electronic Mail: Mail servers

2. Mail Servers

- Mailbox contains incoming messages for user
- Message queue of outgoing (to be sent) mail messages
- Sender's mail server makes connection to Receiver's mail server
    - IP address, port 25

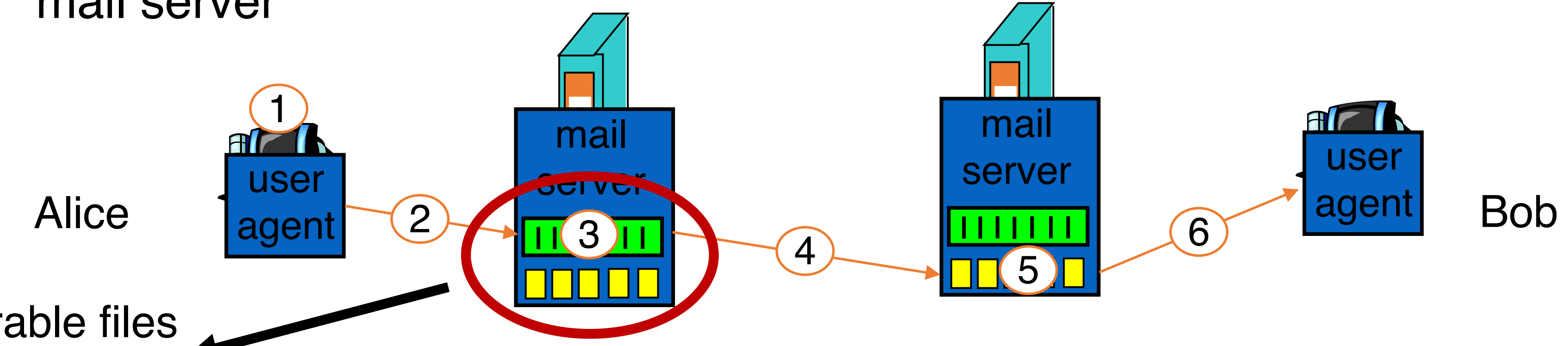


3. SMTP protocol: client/server protocol

- Used to send messages
- Client: sending user agent or sending mail server
- server: receiving mail server

Scenario: Alice sends message to Bob

1) Alice (`alice@rutgers.edu`) uses UA to compose message to `bob@nyu.edu`
2) Alice's UA sends message to her mail server; message placed in outgoing 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 incoming mailbox
6) Sometime later, Bob invokes his user agent to read message



A set of durable files on the machine. Persisted on disk.

Rutgers mail server

NYU mail server

Observations on these exchanges

- Mail servers are useful “always on” endpoints
  - Receiving the email on behalf of Bob, should Bob’s machine be turned off
  - Retrying the delivery of the email to Bob on behalf of Alice, should Bob’s mail server be unavailable in the first attempt
- The same machine can act as client or server based on context
  - Rutgers’s mail server is the server when Alice sends the mail
  - It is the client when it sends mail to Bob’s mail server
- SMTP is push-based: info is pushed from client to server
  - Contrast to HTTP or DNS where info is pulled from the server

Sample SMTP interaction

- A small demo

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

MAIL command response codes

Table 23.2 *Responses*

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

220: Service ready
250: Request command complete
354: Start mail input
421: Service not available

Mail message (stored on server) format

SMTP: protocol for exchanging email msgs

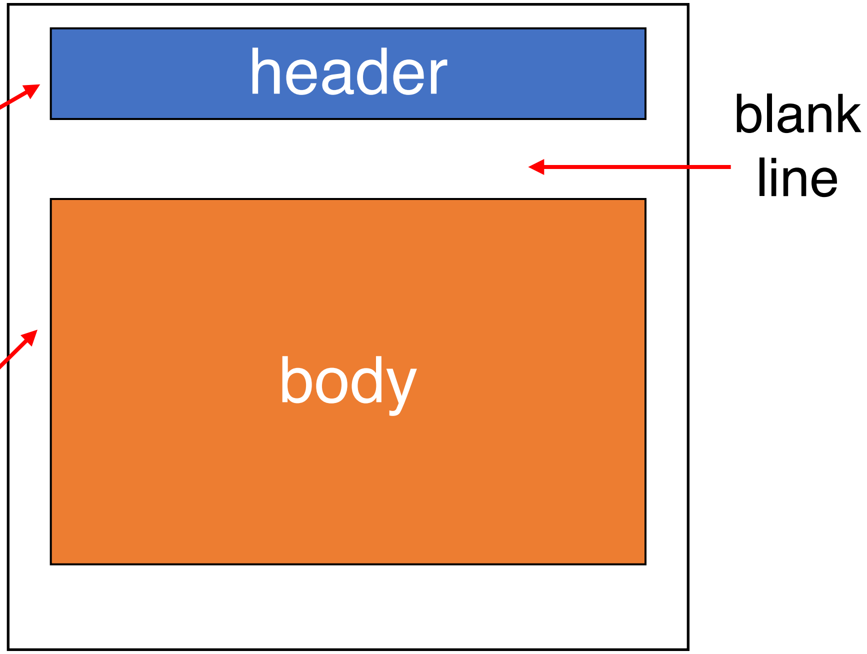
RFC 822: standard for text message format:

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

  different from SMTP commands!

  (these would still be under "DATA")

- 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 typ

MIME version

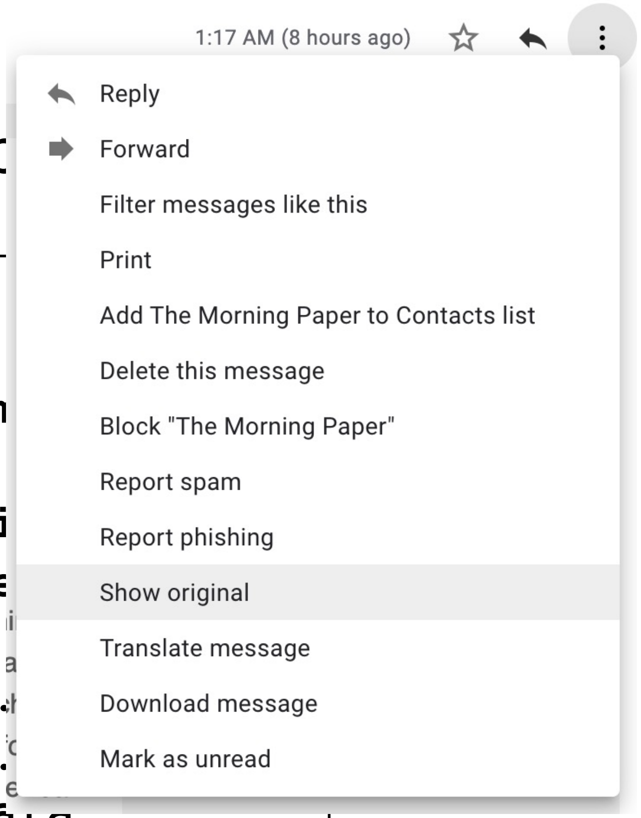
method used to encode data

multimedia data type, subtype, parameter declaration

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yum
MIME-Version: 1.0
Content-Transfer-Encodi
Content-Type: image/jpe

base64 encoded data ...
.......................
......base64 encoded data
```



CS 352
Mail: Access Protocols

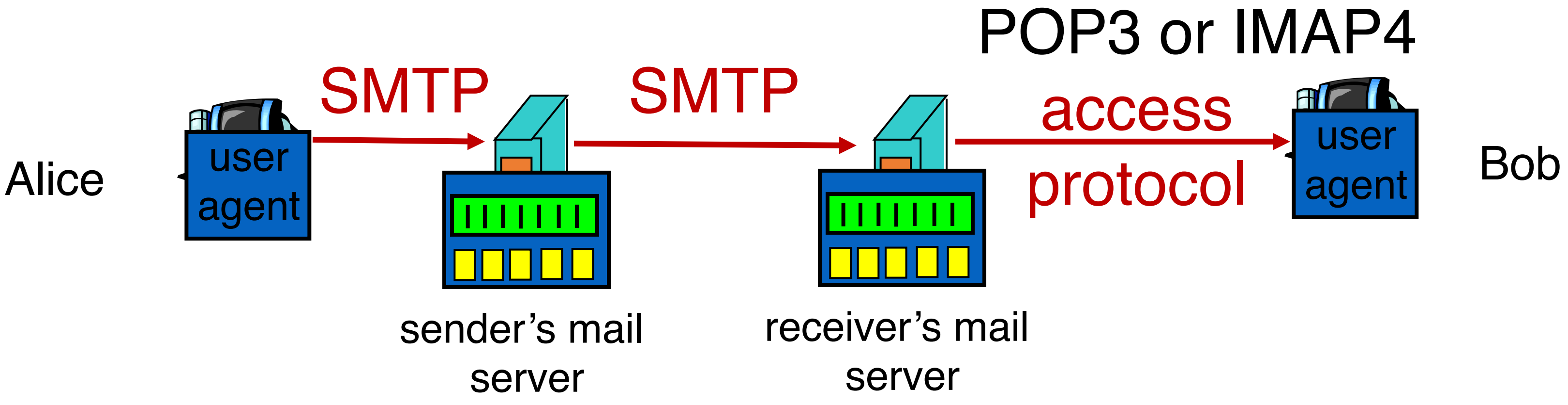
CS 352, Lecture 5.2

http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - Client connects to POP3 server on TCP port 110
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - Client connects to TCP port 143
  - HTTP: gmail, outlook, etc.

POP vs IMAP

- POP3
- Stateless server
- UA-heavy processing
- UA retrieves email from server, then typically deleted from server
- Latest changes are at the UA
- Simple protocol (list, retr, del within a POP session)

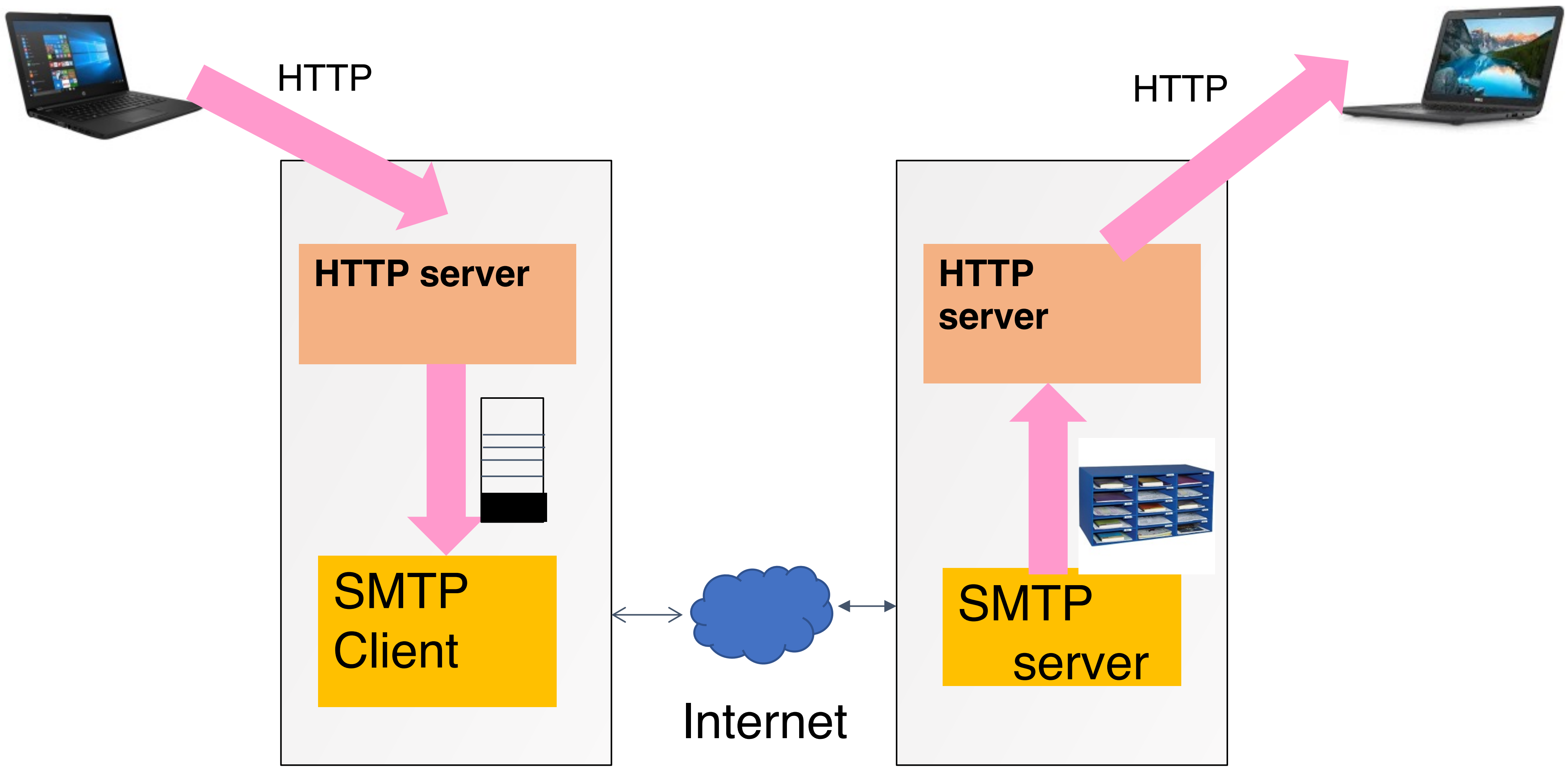
- IMAP4
- Stateful server
- UA and server processing
- Server sees folders, etc. which are visible to UAs
- Changes visible at the server
- Complex protocol

What about web-based email?

- Connect to mail servers via web browser
  - Ex: gmail, outlook, etc.

- Browsers speak HTTP
- Email servers speak SMTP
- Need a bridge to retrieve email using HTTP

Web based email



Comparing SMTP 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

- HTTP: can put non-ASCII data directly in response
- SMTP: need ASCII-based encoding

More themes from app-layer protocols

- Separation of concerns. Examples:
  - Content rendering for users (browser, UA) separate from protocol operations (mail server)
  - Reliable mail sending and receiving: mail UA doesn't need to be "always on" to send or receive email reliably
- In-band vs. out-of-band control:
  - In-band: headers determine the actions of all the parties of the protocol
  - There are protocols with out-of-band control, e.g., FTP
- Keep it simple until you really need complexity
  - ASCII-based design; stateless servers. Then introduce:
  - Cookies for HTTP state
  - IMAP for email organization
  - Security extensions (e.g., TLS)
  - Different methods to set up and use underlying connections (e.g., persistence)