Lecture 2: Application Layer

Acknowledgement: Materials presented in this lecture are predominantly based on slides from:

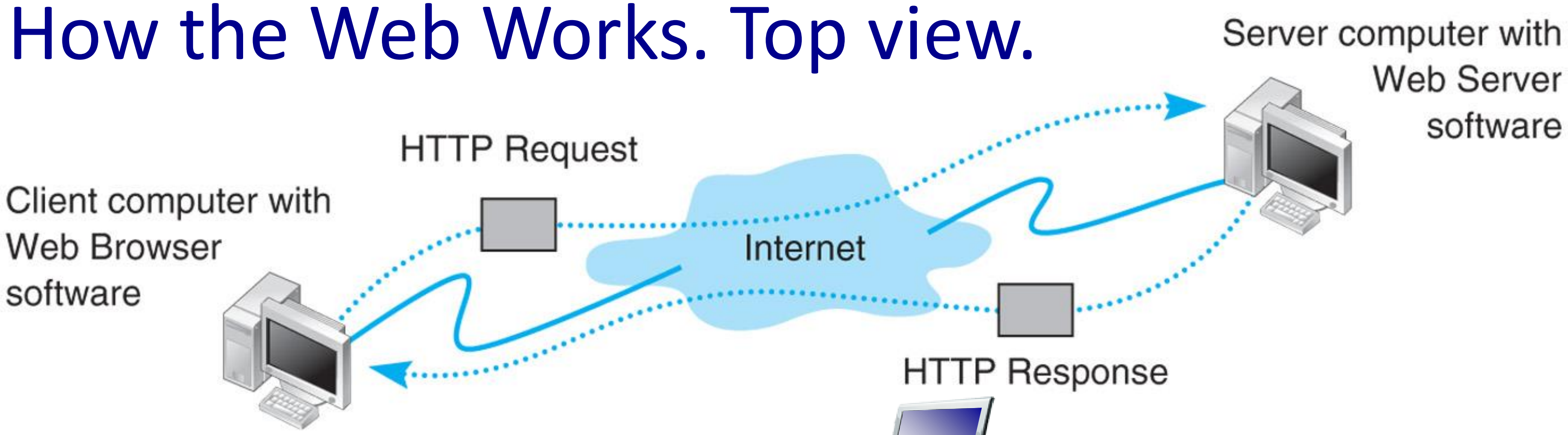
- J. FitzGerald, A. Dennis, A. Durcikova : Business Data Communications and Networking, 12th ed., 2014, John Wiley & Sons, Chapter 2
- J. F. Kurose, K. W. Ross: Computer Networking. A Top-down approach, 7th ed., 2017, Pearson, Chapter 2

(2018 update)

Lecture 2: Application Layer. Overview

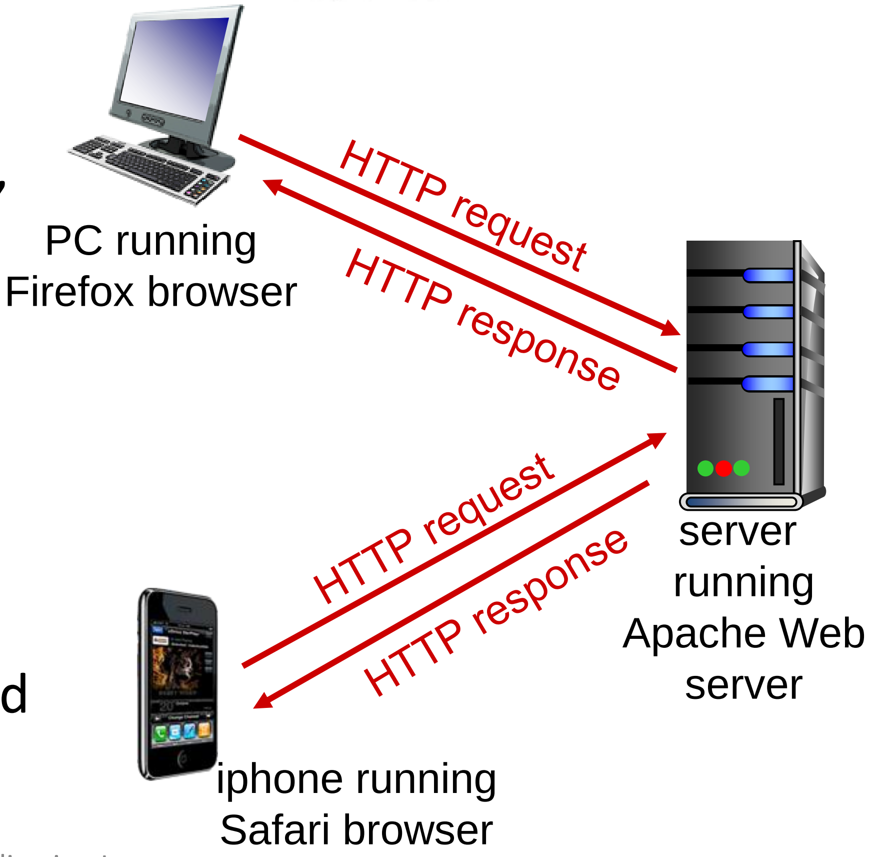
- World Wide Web: The HTTP protocol
  - How the Web Works
  - The HTTP Request and Response.
  - The Socket Address
  - Persistent and non-persistent HTTP connection
  - Cookies
  - Web caches
- Electronic Mail
  - SMTP, IMAP, POP
  - Two-tear e-mail Architecture
  - Web-mail Architecture
  - Details of SMTP, MIME, IMAP
- Instant messages

How the Web Works. Top view.


Server computer with Web Server software
HTTP Request
Client computer with Web Browser software
Internet
HTTP Response

Client-Server model:

- *client:* **browser** that requests, receives, and displays/renders Web (HTML) objects using the **HTTP protocol**
- *server:* **Web server** sends (using HTTP protocol) objects in response to requests
- Multiple request-response cycles to exchange HTTP packets with embedded HTML objects (text, images, video, …)


PC running Firefox browser
HTTP request
HTTP response
server running Apache Web server
HTTP request
HTTP response
iphone running Safari browser

HTTP – HyperText Transfer Protocol

- HTTP Protocol is an Internet standard maintained by the **Internet Engineering Task Force** (IETF).
- RFC 7230 : **Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing**, June 2014, describes the current most commonly used version

HTTP is

- a **stateless** (server maintains **no** information about past client requests)
- application level
- request/response protocol
- running on top of a TCP connection
- for distributed, collaborative, hypertext information systems.

What in HTTP/1.1 documents

RFC 7230

- provides an overview of HTTP **architecture** and its associated terminology,
- defines the "http" and "https" **Uniform Resource Identifier** (URI) schemes,
- defines the **HTTP/1.1 message syntax** and parsing requirements,
- describes related **security** concerns for implementations.

Related documents

- ➢ RFC 7231: **Semantics** and Contents
- ➢ RFC 7232: Conditional Requests
- ➢ RFC 7233: Range Requests
- ➢ RFC 7234: Caching
- ➢ RFC 7235: Authentication

HyperText Transfer Protocol Version 2

RFC 7540 published in May 2015 describes **HTTP/2**

HTTP/2:

- describes an optimized expression of the semantics of the HTTP,
- enables a more efficient use of network resources and
- a **reduced perception of latency** by
  - introducing **header field compression** and
  - allowing **multiple concurrent** exchanges on the same TCP connection.
- HTTP's **existing semantics** as described in HTTP/1.1 remain unchanged.

Introductory example from RFC7230

The following example illustrates a typical message exchange for a GET request on the URI "http://www.example.com/hello.txt":

Client request:

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
```

Server response:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain

Hello World! My payload includes a trailing CRLF.
```

HTTP **request** message

request-line = method SP request-target SP HTTP-version CRLF

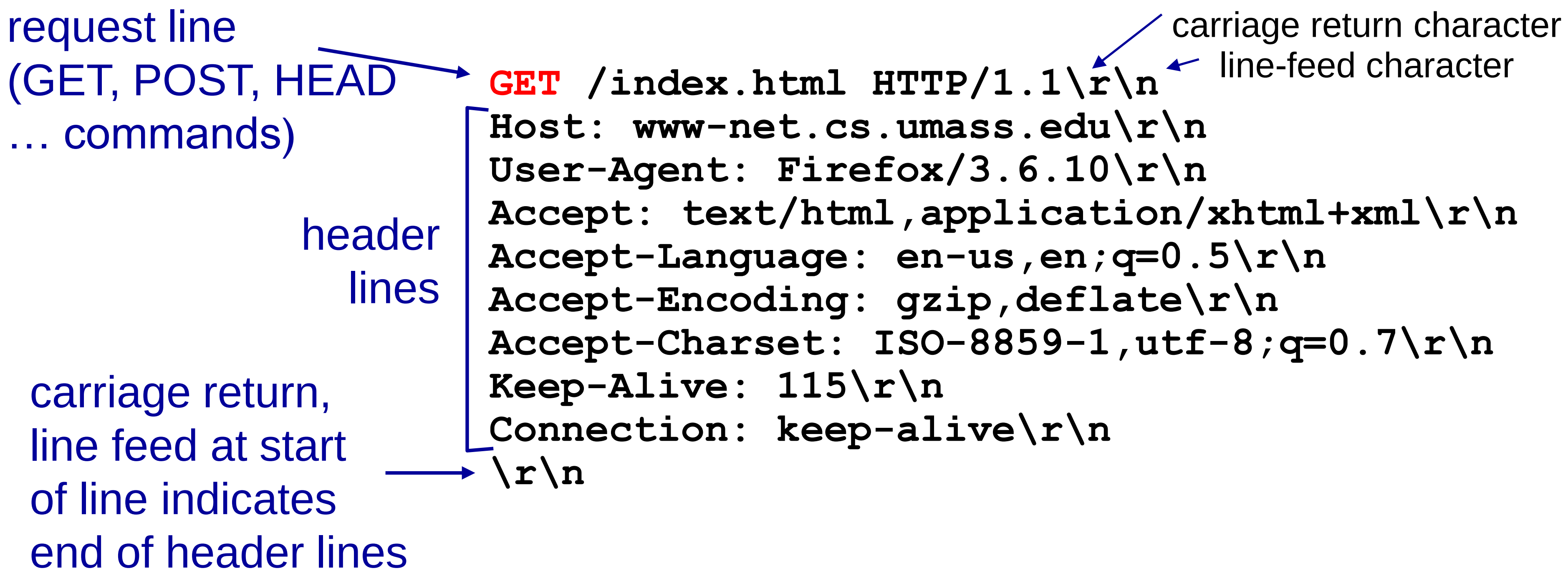
e.g. GET /hello.txt HTTP/1.1

header-field = field-name ":" OWS field-value OWS (optional white spaces)

e.g. Host: cnn.com

User-Agent: Mozilla/5.0 (Windows NT 6.1; ...

Referrer: http://www.rexswain.com/httpview.html


request line (GET, POST, HEAD … commands)
carriage return character
line-feed character
header lines
carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Request Methods

- GET Transfer a current representation of the target resource
- HEAD Same as GET, but only transfer the status line and header section
- POST Perform resource-specific processing on the request payload
- PUT Replace all current representations of the target resource with the request payload
- DELETE Remove all current representations of the target resource
- CONNECT Establish a tunnel to the server identified by the target resource.
- OPTIONS Describe the communication options for the target resource
- TRACE Perform a message loop-back test along the path to the target resource.

HTTP **response** message

status-line = HTTP-version SP status-code SP reason-phrase CRLF

e.g. HTTP/1.1 301 Moved Permanently CRLF

header-fields = field-name ":" OWS field-value OWS

[message-body]

status line (protocol status code status phrase)

header lines

data, e.g., requested **HTML** file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

`200 OK`

- request succeeded, requested object later in this msg

`301 Moved Permanently`

- requested object moved, new location specified later in this msg (Location:)

`400 Bad Request`

- request msg not understood by server

`404 Not Found`

- requested document not found on this server

`505 HTTP Version Not Supported`

Response Status Codes

- Status codes are defined in RFC 7231: Semantics and Content, section 6.
- Divided into five groups:

Informational 1xx

- 100 Continue

  request has been received and not rejected yet.
- 101 Switching Protocols

  used to migrate to newer protocols, e.g. HTTP\2.0

Successful 2xx

- 200 OK
  The request succeeded
- 201 Created
  The request resulted in creation of new resources
- 202 Accepted
  Request accepted for processing
- 203 Non-Authoritative Information
  The enclosed payload has been notified.
- 204 No Content
  Request has been fulfilled. No additional info from the server.
- 205 Reset Content
- 206 Partial Content

Redirection 3xx

- 300 Multiple Choices
- 301 Moved Permanently

  Target resource has been assigned a new permanent URI and any future references to this resource ought to use one of the enclosed URIs.
- 302 Found
- 303 See Other
- 304 Not Modified
- 305 Use Proxy
- 307 Temporary Redirect

Client Error 4xx

- 400 | Bad Request
- 401 | Unauthorized
- 402 | Payment Required
- 403 | Forbidden
- 404 | Not Found
- 405 | Method Not Allowed
- 406 | Not Acceptable
- 407 | Proxy Authentication Required
- 408 | Request Timeout
- 409 | Conflict
- 410 | Gone
- 411 | Length Required
- 412 | Precondition Failed
- 413 | Payload Too Large
- 414 | URI Too Long
- 415 | Unsupported Media Type
- 416 | Range Not Satisfiable
- 417 | Expectation Failed
- 426 | Upgrade Required

Server Error 5xx

- 500 | Internal Server Error
- 501 | Not Implemented
- 502 | Bad Gateway
- 503 | Service Unavailable
- 504 | Gateway Timeout
- 505 | HTTP Version Not Supported

HTTP connections

non-persistent HTTP

- at most one object sent over a TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over a single TCP connection between client and server

Non-persistent HTTP

suppose user enters URI:

`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at **www.someSchool.edu** on port 80

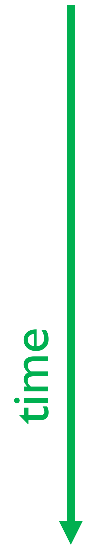
1b. HTTP server at host **www.someSchool.edu** waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URI) into TCP connection socket. Message indicates that client wants object **someDepartment/home.index**

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

time

Non-persistent HTTP (cont.)



4. HTTP server closes TCP connection.

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

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

Non-persistent HTTP: response time

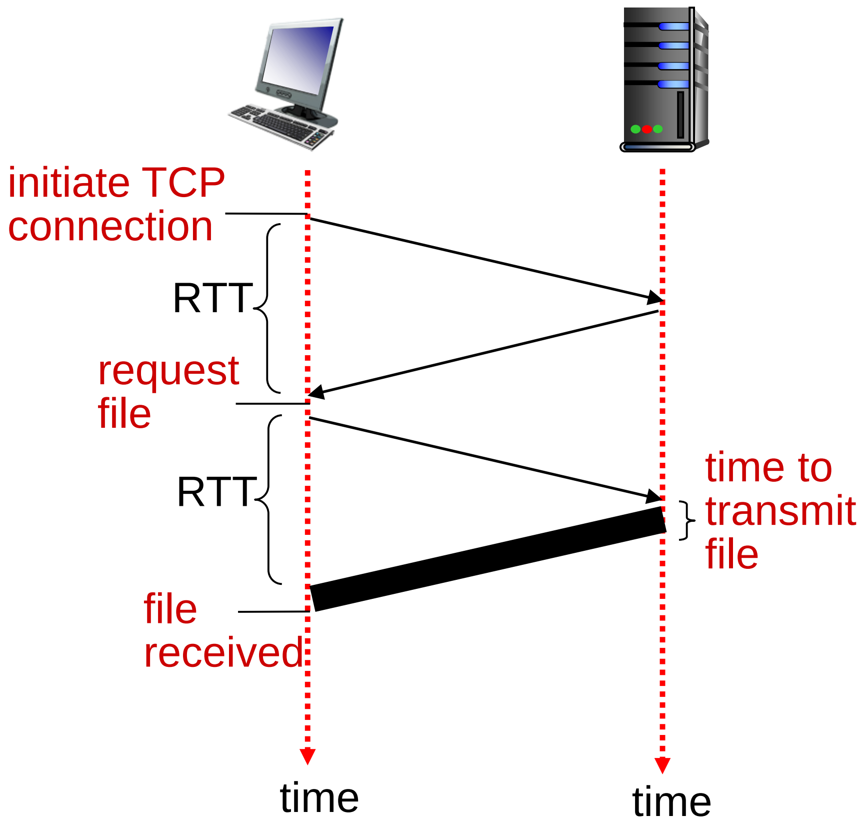
RTT– Round-Trip Time

(definition): time for a small packet to travel from client to server and back

HTTP 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
- **non-persistent HTTP response time:**

  = 2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

- 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 TCP 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 State Management Mechanism

- HTTP is a **stateless** protocol: the 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

- HTTP state management mechanism is based on **cookies** stored on a client computer.
- Cookies are defined in RFC 6265
- Are managed by the **user's browser**.

User-server state: cookies

many Web sites use cookies

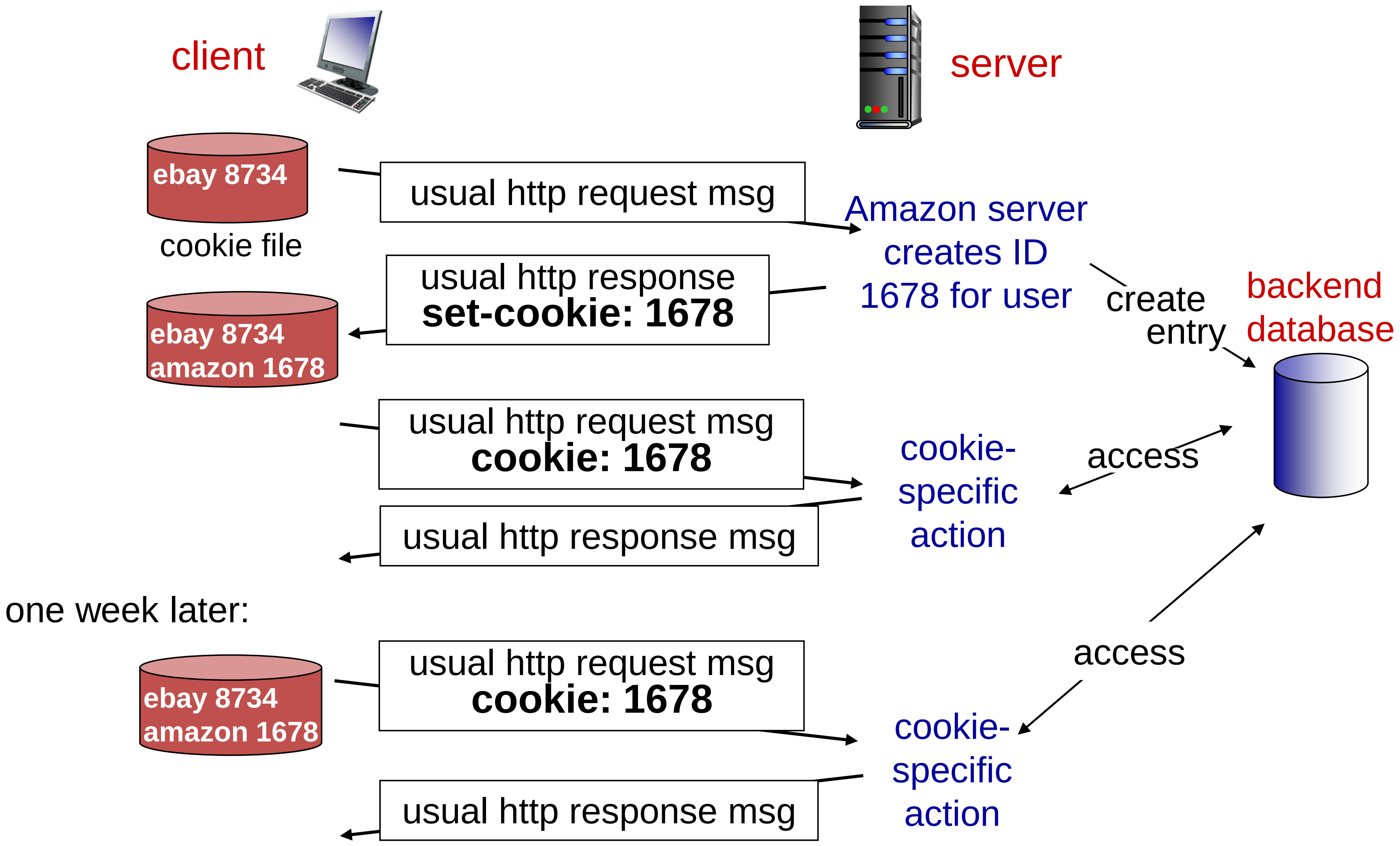
four components:

1. cookie header line of HTTP *response* message
2. cookie header line in next HTTP *request* message
3. cookie file kept on user's host, managed by user's browser
4. back-end database at Web site

example:

- Susan always access Internet from PC
- visits http://www.taobao.com/ site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

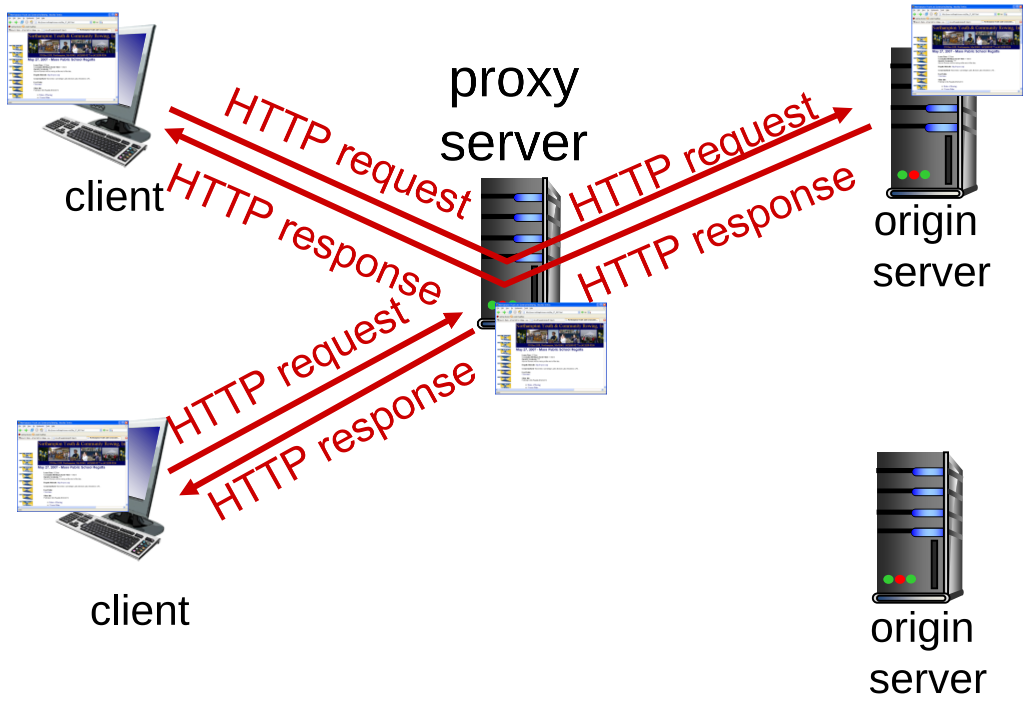
how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

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
  - If object is in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- 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

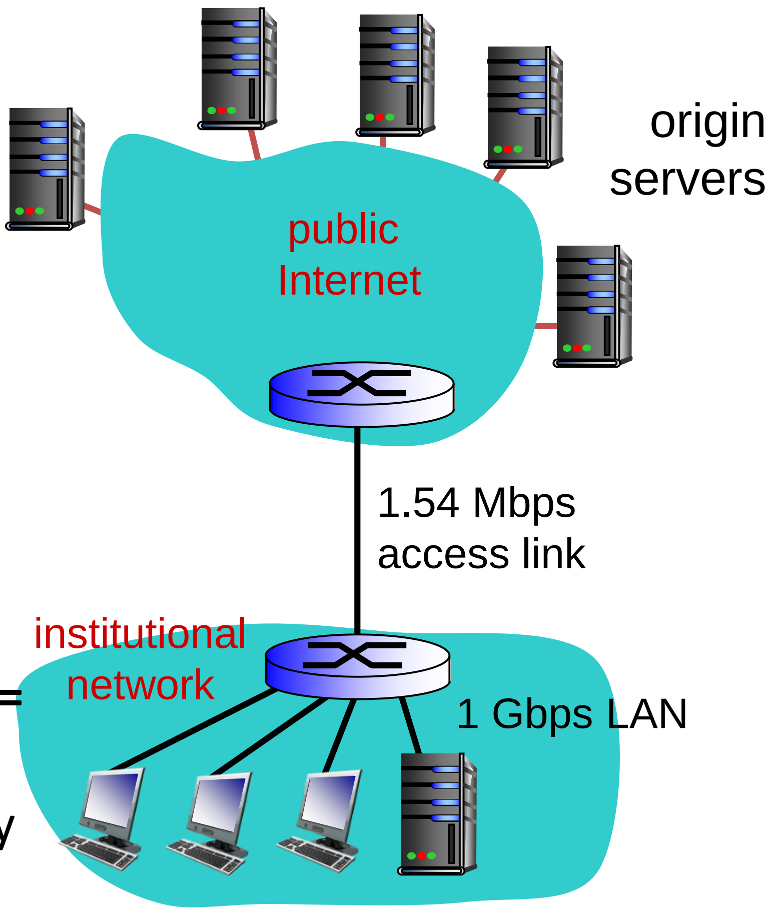
Timing example: no cache, slow access link

assumptions:

- Average object size: 100K bits
- Avg. request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 100K*15/sec = **1.50 Mbps** =
- RTT from institutional router to any origin server: 2 sec
- access link rate: **1.54 Mbps**

consequences:

- LAN utilization: 0.15% = 1.5Mbps/1Gbps
- **Access link utilization =** 1.5Mbps/1.54Mbp **= 99%** *problem!*
- **total delay** = Internet delay + access delay + LAN delay = 2 sec + minutes(?) + μsecs



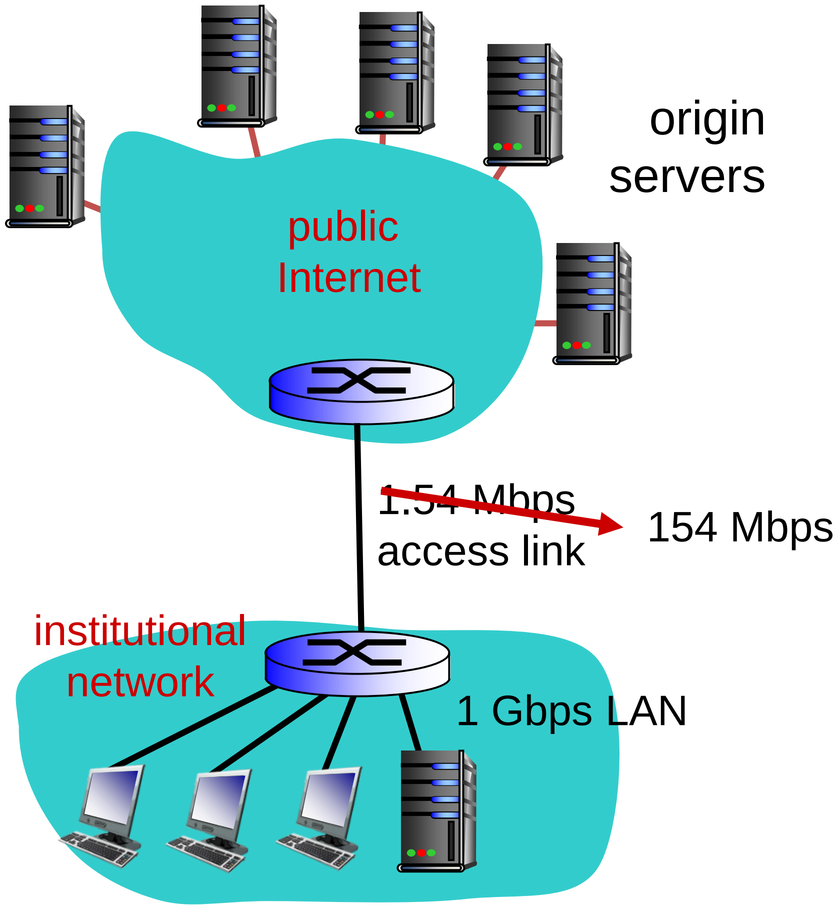
Timing example: no cache, fast access link

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~1.54 Mbps~~ → **154 Mbps**

consequences:

- LAN utilization: 0.15%
- access link utilization = ~~99%~~ → 0.99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + µsecs
  → msecs



Cost: increased access link speed (not cheap!)

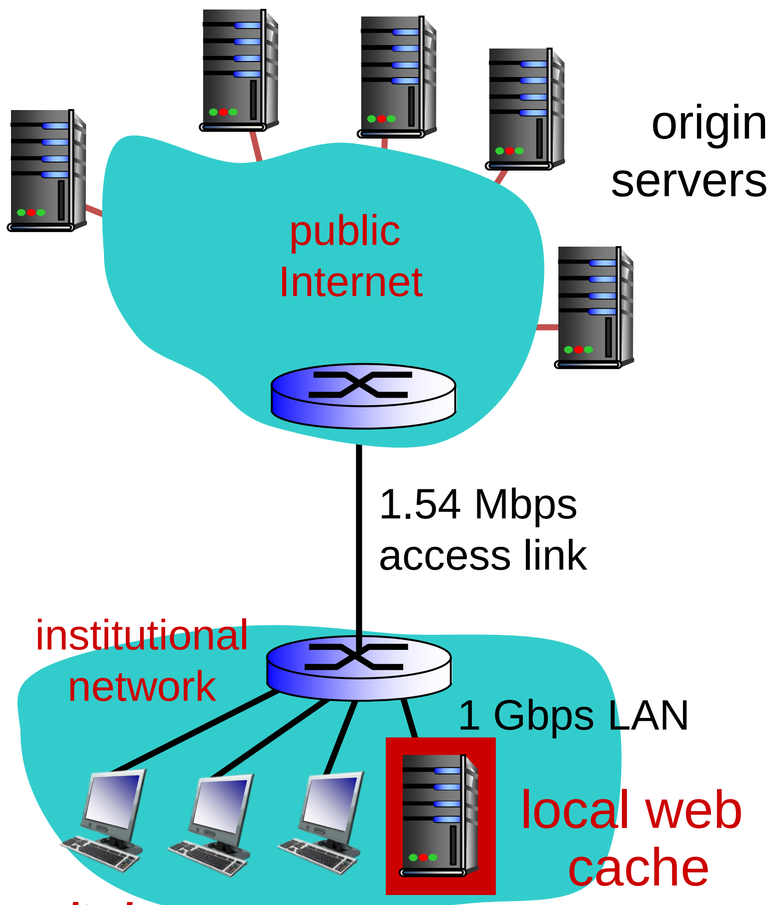
Timing example: slow link, local cache

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 0.15%
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + µsecs



How to compute link utilization, delay?

Cost: web cache (cheap!)

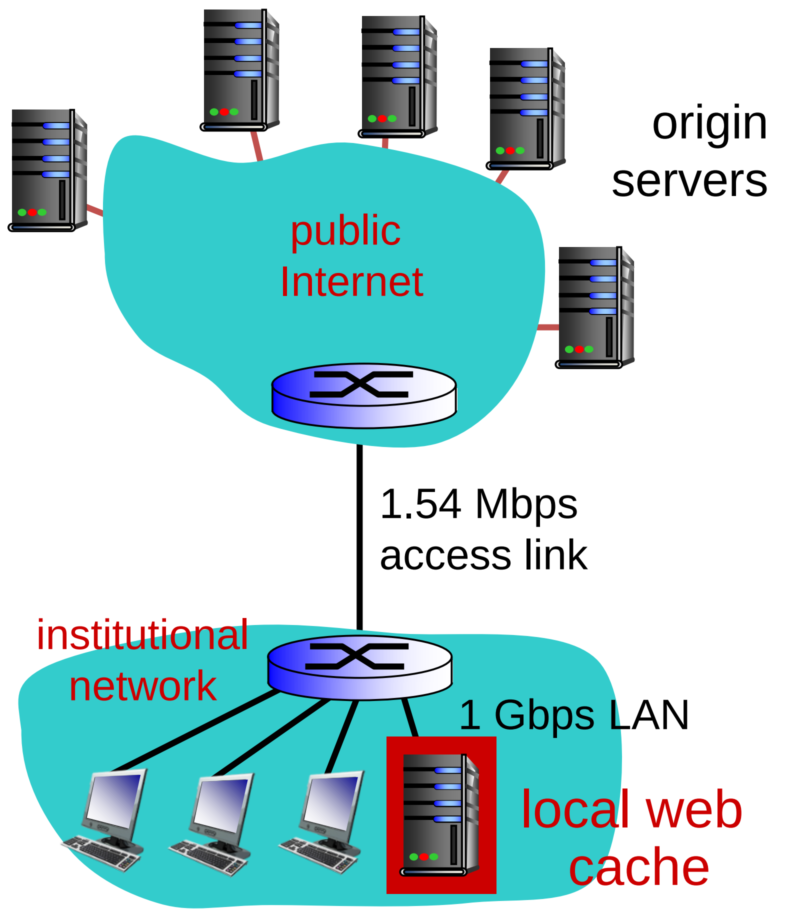
Local Cache: access link utilization and delay

suppose cache hit rate is 0.4

- 40% requests satisfied at cache,
- 60% requests satisfied at origin

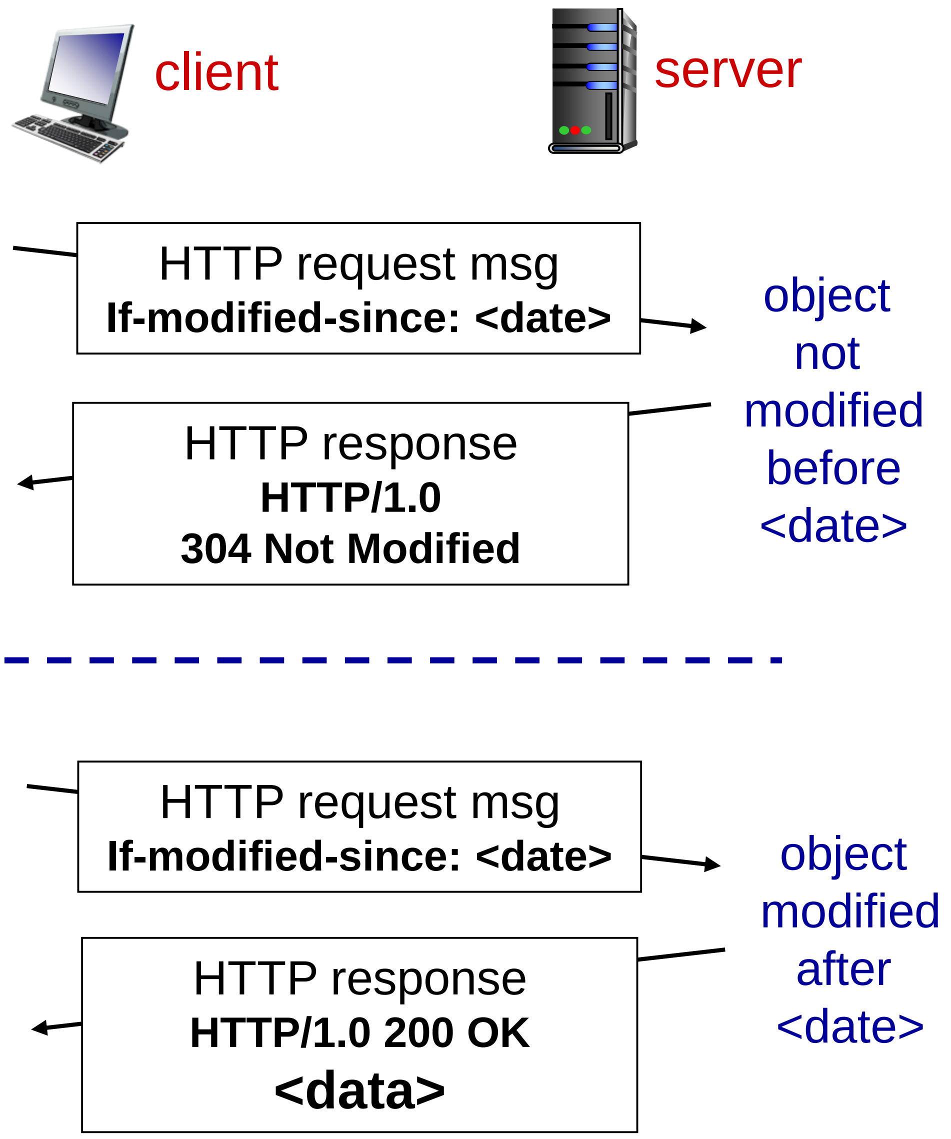
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = 58%
- total delay
  - = 0.6 * (delay from origin servers) + 0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs)
  - = ~ 1.2 secs

less than with 154 Mbps link (and cheaper too!)



Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *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`



Electronic Mail

SMTP – **Simple Mail Transfer Protocol**, RFC 5321 (2008)

- is the Internet standard for electronic mail (e-mail) transmission.
- **MIME extension** for large and binary files is described in RFC 3030
- MIME standard is described in RFC6512.

IMAP – **Internet Message Access Protocol**, RFC 3501 (IMAP4rev1)

- is a protocol for **email retrieval and storage**.
- IMAP allows an **e-mail client** to access e-mail on a **remote mail server.**
- Many updates, e.g. RFC 5737 – **IMAP Support for UTF-8**

POP – **Post Office Protocol,** RFC 6186 (POP3)

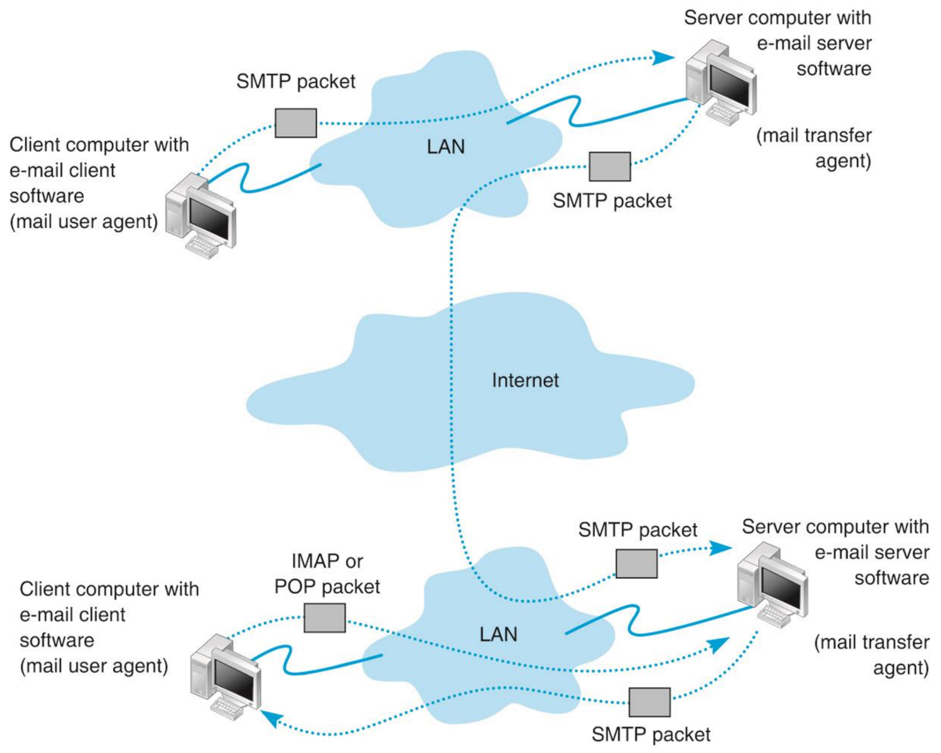
- is a protocol used by local e-mail clients to retrieve e-mail from a remote server over a TCP/IP connection.
- Current specification is updated with an extension and an authentication mechanisms (RFC 5034)

➢ IMAP and POP3 are supported by all modern **e-mail clients and servers**, and are the two most prevalent Internet standard protocols for e-mail retrieval.

Electronic Mail: Two-Tier Architecture

Two-Tier E-mail Architecture: **one client, one server**

- Each client computer runs an application called: a **mail user agent** aka an **e-mail client** e.g. Outlook or Mozilla Thunderbird
- The **e-mail client** is used to create the e-mail messages formatted as **SMTP** packets.
- SMTP packet includes the sender's and the destination **addresses**

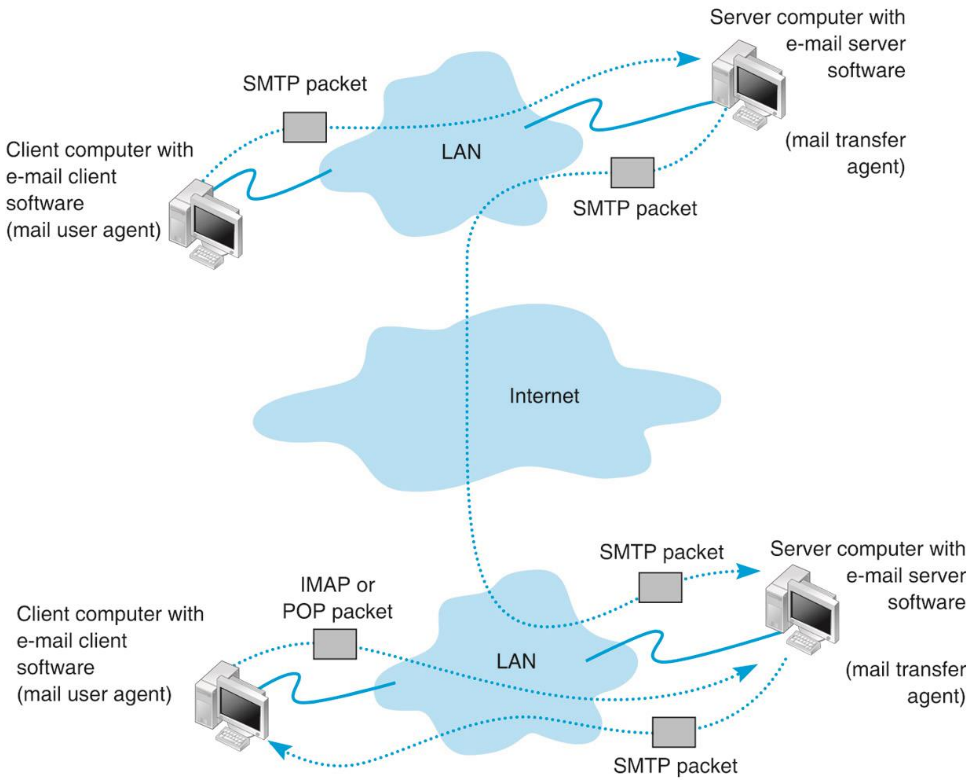


Server computer with e-mail server software
(mail transfer agent)
SMTP packet
LAN
Client computer with e-mail client software (mail user agent)
SMTP packet
Internet
SMTP packet
Server computer with e-mail server software
IMAP or POP packet
Client computer with e-mail client software (mail user agent)
LAN
(mail transfer agent)
SMTP packet

- The user e-mail client sends the SMTP packet to a **mail server** that runs an application layer software: a **mail transfer package** (mail server software)

Two-Tier E-mail Architecture: sending the e-mail

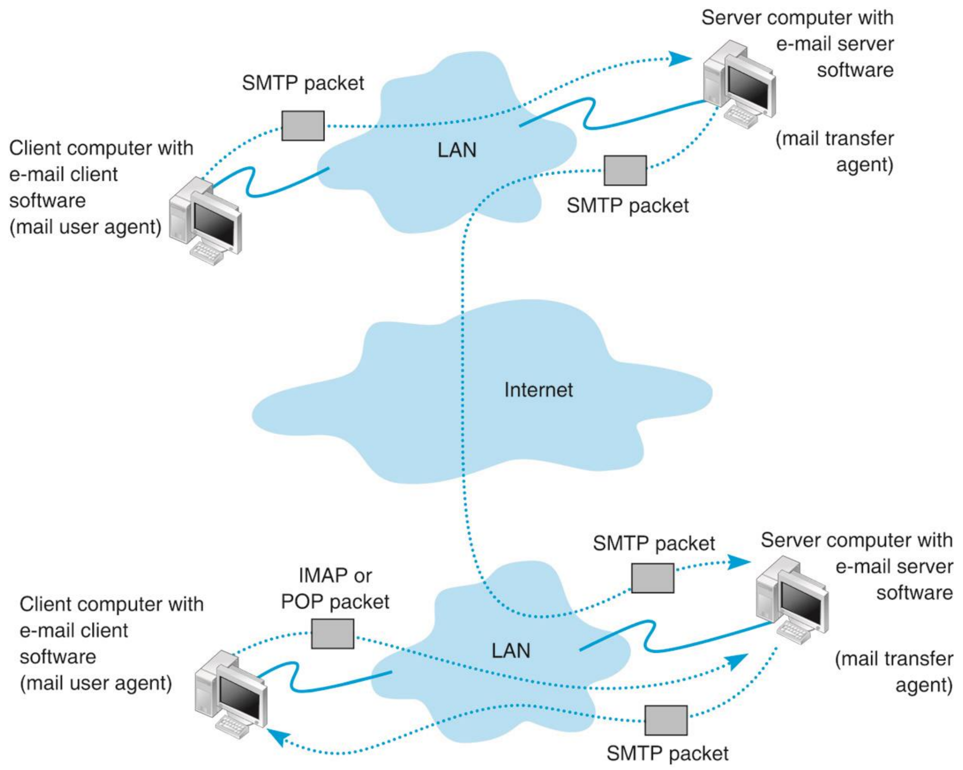
- The mail server reads the SMTP packet to find the destination address and then **sends the packet** through the network until it reaches the mail server specified in the destination address.
- The mail transfer agent on the **destination server** then stores the message in the receiver's mailbox on that server.


Server computer with e-mail server software
(mail transfer agent)
SMTP packet
Client computer with e-mail client software (mail user agent)
LAN
SMTP packet
Internet
SMTP packet
Server computer with e-mail server software
IMAP or POP packet
Client computer with e-mail client software (mail user agent)
LAN
(mail transfer agent)
SMTP packet

- The message sits in the mailbox assigned to the user who is to receive the message until s/he checks for new mail.
- The **SMTP standard** covers message transmission **between** mail servers and **between** the originating e-mail client and its mail server.

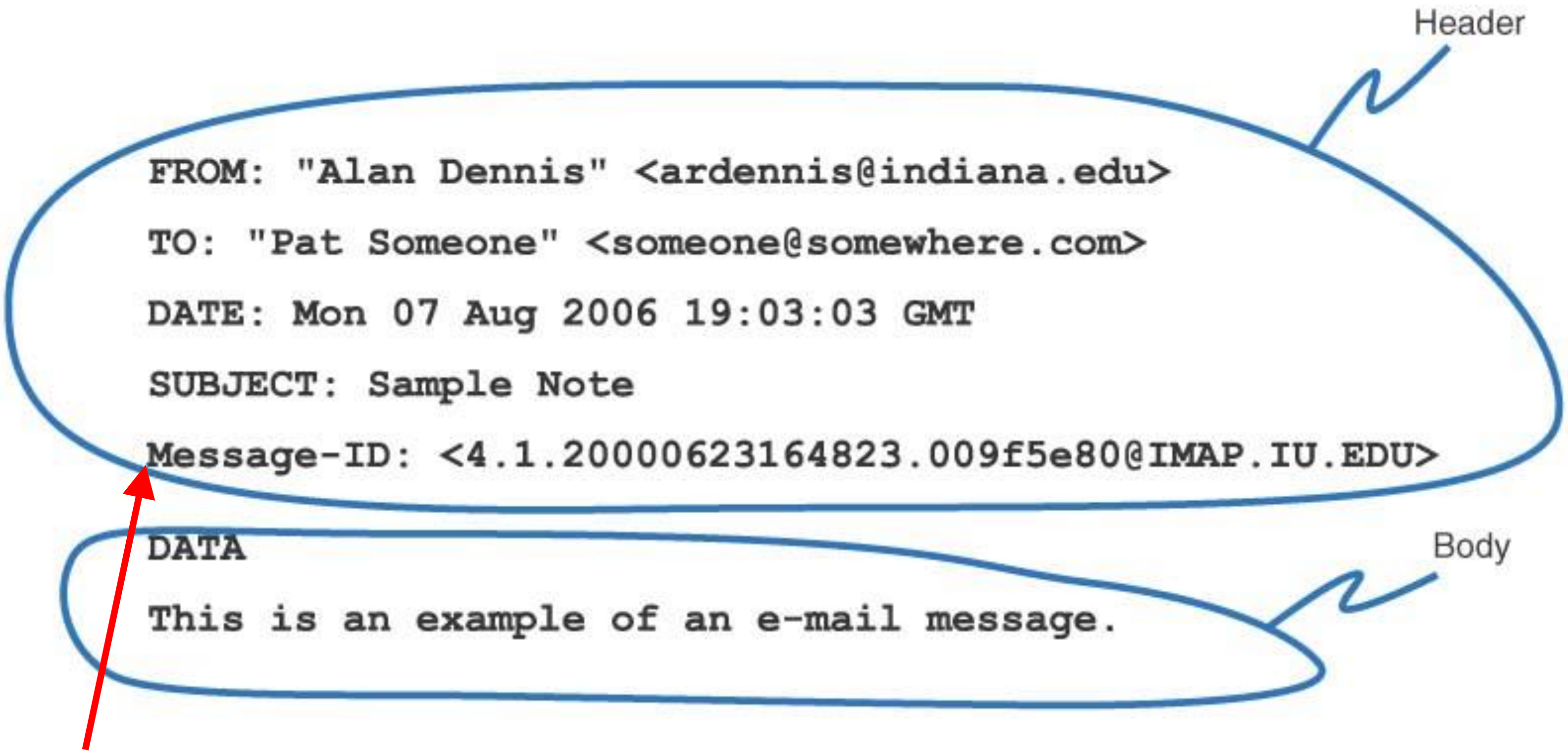
Two-Tier E-mail Architecture: reading the e-mail

- Two commonly used standards for communication between mail client and mail server:
- **Post Office Protocol (POP)** : message is downloaded to the client and removed from the server
- **Internet Message Access Protocol (IMAP):** message remains on the server


Server computer with e-mail server software
(mail transfer agent)
SMTP packet
LAN
Client computer with e-mail client software (mail user agent)
SMTP packet
Internet
SMTP packet
Server computer with e-mail server software
IMAP or POP packet
Client computer with e-mail client software (mail user agent)
LAN
(mail transfer agent)
SMTP packet

- To read the message the e-mail client sends an IMAP (or POP) packet that asks for the user's mailbox contents.
- When the mail server receives the IMAP (or POP) request, it sends the original SMTP packet created by the message sender to the client computer, which the user reads with the e-mail client.

Example of a SMTP Message/packet

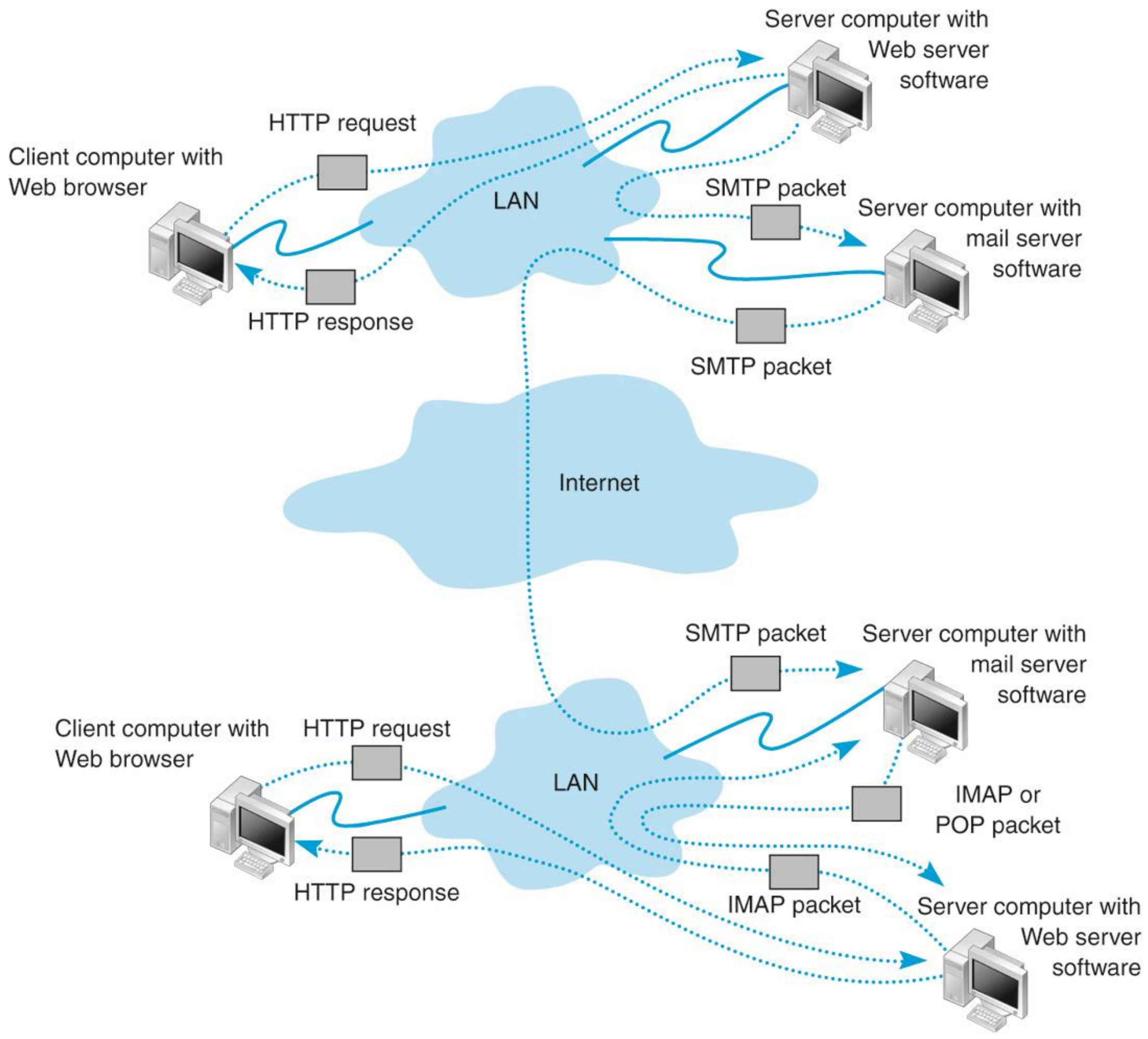


```
FROM: "Alan Dennis" <ardennis@indiana.edu>
TO: "Pat Someone" <someone@somewhere.com>
DATE: Mon 07 Aug 2006 19:03:03 GMT
SUBJECT: Sample Note
Message-ID: <4.1.20000623164823.009f5e80@IMAP.IU.EDU>
DATA
This is an example of an e-mail message.
```

- The **Message ID** field is used to provide a unique identification code so that the message can be tracked
- **Attachments**, a non-text files, are converted into text using **MIME standard** and included into the SMPT packet. E-mail client converts the attachment into the original format

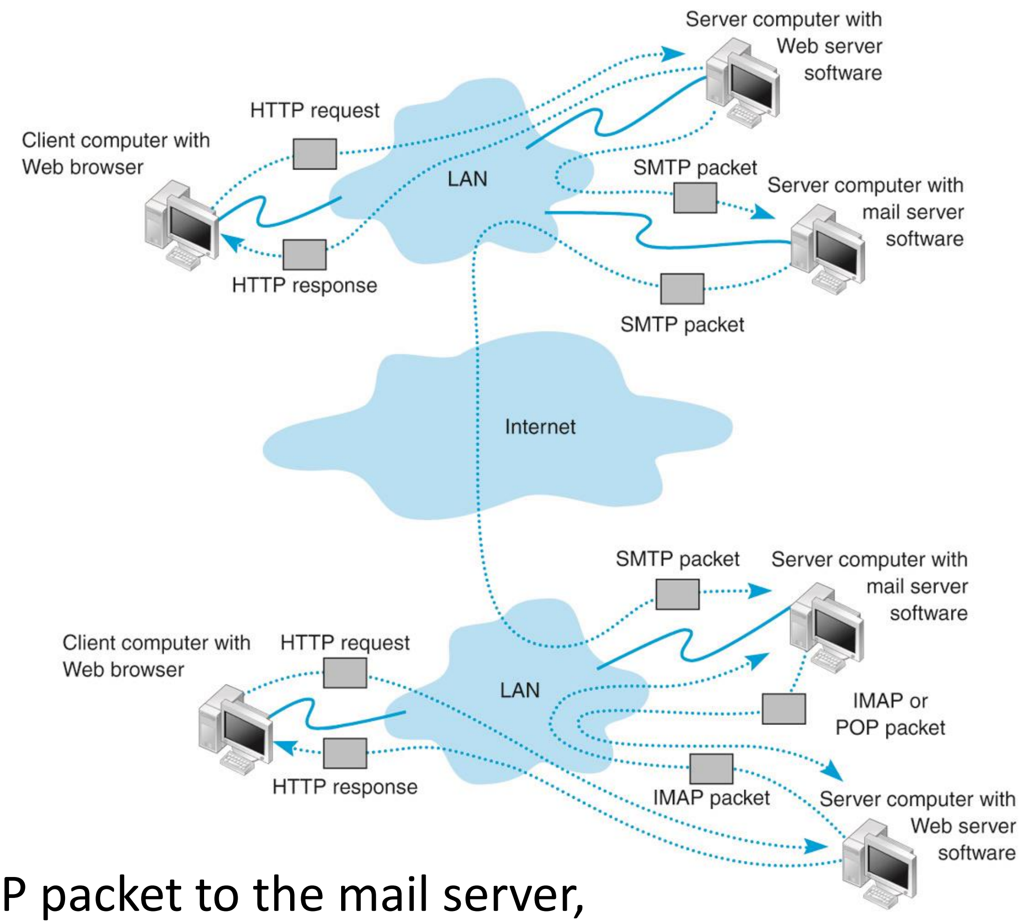
Three-Tier Thin Client aka Web-mail Architecture - 1

- The three-tier thin client-server e-mail architecture uses a **Web server** and **Web browser** to provide access to your e-mail.
- No e-mail client on a client computer.
- With the browser you connect to a page on a **Web server** that lets you write the e-mail message by **filling in a form**.


Server computer with Web server software
HTTP request
Client computer with Web browser
LAN
SMTP packet
Server computer with mail server software
HTTP response
SMTP packet
Internet
SMTP packet
Server computer with mail server software
Client computer with Web browser
HTTP request
LAN
IMAP or POP packet
HTTP response
IMAP packet
Server computer with Web server software

Web-mail Architecture - 2

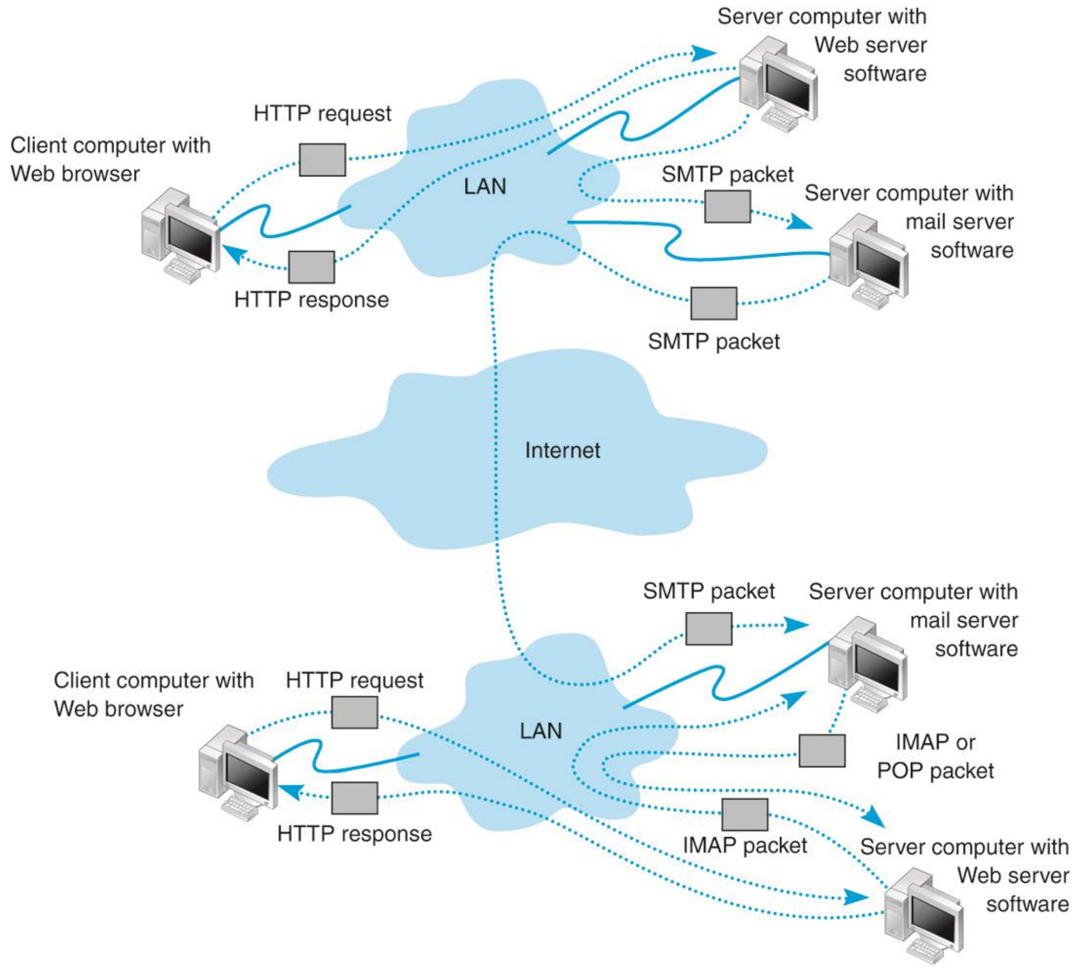
- When you click the send button, your Web browser sends the form information to the **Web server** inside an HTTP request.
- The Web server runs a program that takes the information from the HTTP request and builds an **SMTP packet** that contains the e-mail message.
- It also sends an HTTP response back to the client.


Server computer with Web server software
HTTP request
Client computer with Web browser
LAN
SMTP packet
Server computer with mail server software
HTTP response
SMTP packet
Internet
SMTP packet
Server computer with mail server software
Client computer with Web browser
HTTP request
LAN
IMAP or POP packet
HTTP response
IMAP packet
Server computer with Web server software

- The Web server then sends the SMTP packet to the mail server, which processes the SMTP packet as though it came from a client computer.
- The SMTP packet flows through the network in the same manner as before.
- When it arrives at the destination mail server, it is placed in the receiver's mailbox

Web-mail Architecture - 3

- When the receiver wants to check his/her mail, s/he uses a Web browser to send an HTTP request to a Web server
- A program on the Web server processes the request and sends the appropriate IMAP (or POP) request to the mail server.
- The mail server responds with the original SMTP packet, which a program on the Web server converts into an HTTP response and sends to the client.
- The client then displays the e-mail message in the Web browser.



Server computer with Web server software
HTTP request
Client computer with Web browser
LAN
SMTP packet
Server computer with mail server software
HTTP response
SMTP packet
Internet
SMTP packet
Server computer with mail server software
Client computer with Web browser
HTTP request
LAN
IMAP or POP packet
HTTP response
IMAP packet
Server computer with Web server software

Web-mail Architecture - 4

- The three-tier approach using a Web browser is much more complicated than the two-tier approach.
- However it is simpler to have just a Web browser on the client computer rather than to require the user to install a special e-mail client on his or her computer and then set up the e-mail client to connect to the correct mail server using either POP or IMAP.
- It is simpler for the users to just type the URL of the Web server providing the mail services into their browsers and begin using mail.
- This also means that users can check their e-mail from a public computer anywhere on the Internet.
- Note that the **sender and receiver do not have to use the same** architecture for their e-mail.

RFC5321: SMTP
D.1. A Typical SMTP Transaction Scenario

This SMTP example shows mail sent by Smith at host bar.com, and to Jones, Green, and Brown at host foo.com. Here we assume that host bar.com contacts host foo.com directly. The mail is accepted for Jones and Brown. Green does not have a mailbox at host foo.com.

```
S: 220 foo.com Simple Mail Transfer
Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<Smith@bar.com>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: RCPT TO:<Green@foo.com>
S: 550 No such user here
C: RCPT TO:<Brown@foo.com>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

What next in e-mail?

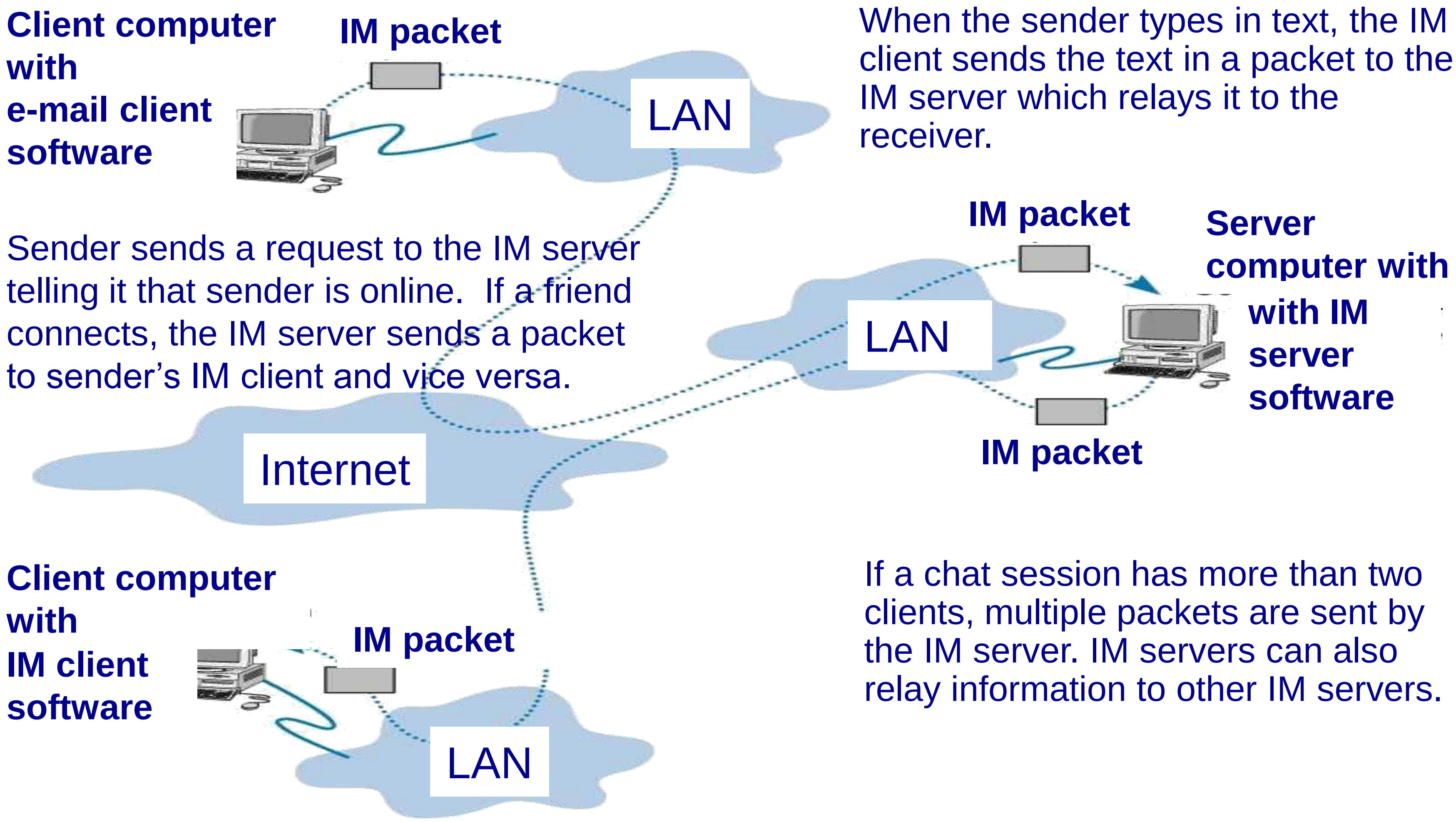
- MIME SMTP extension for large and binary files is described in RFC 3030
- MIME standard is described in RFC6512
- S/MIME defined in RFC 3369, 3370, 3850 and 3851

  I hope you will present details in the NS part
- IMAP – Internet Message Access Protocol, RFC 3501 (IMAP4rev1)

Instant Messaging (IM). Top view.

- A client-server program that allows real-time typed messages to be exchanged
  - Client needs an IM client software
  - Server needs an IM server package
- Most Instant Messaging software allow voice and video packets to be sent so that you computer is converted into a (video) telephone
- Examples include Skype and Line, WeChat, Viber, …
- Two step process:
  - Telling IM server that you are online
  - Chatting

How Instant Messaging Works


Client computer with e-mail client software
IM packet
LAN
IM packet
Server computer with with IM server software
LAN
IM packet
Internet
Client computer with IM client software
IM packet
LAN

When the sender types in text, the IM client sends the text in a packet to the IM server which relays it to the receiver.

Sender sends a request to the IM server telling it that sender is online. If a friend connects, the IM server sends a packet to sender's IM client and vice versa.

If a chat session has more than two clients, multiple packets are sent by the IM server. IM servers can also relay information to other IM servers.