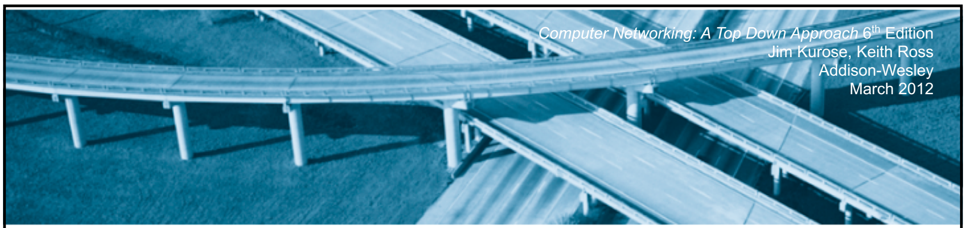


Computer Networking: A Top Down Approach 6th Edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

CHAPTER 2

Application Layer

CHAPTER 2

Application Layer

Our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer (P2P) paradigm

Overview:

- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS

2-2

CHAPTER 2

Roadmap:

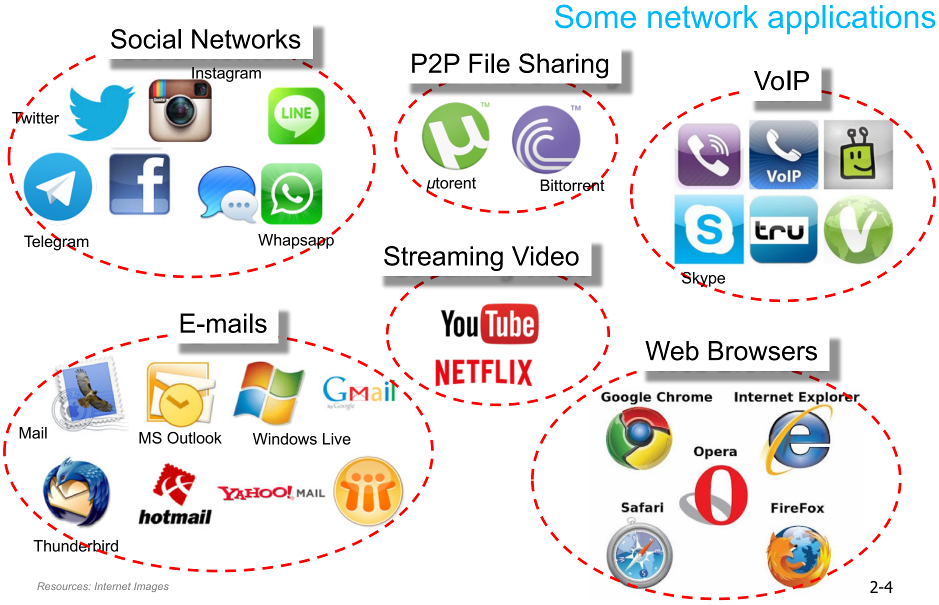
2.1 Principles of network applications
2.2 Web and HTTP
2.3 FTP
2.4 Electronic mail
- SMTP, POP3, IMAP

2.5 DNS
2.6 P2P applications

2-3

CHAPTER 2

Some network applications



Resources: Internet Images

2-4

CHAPTER 2

(2.1) Principles of Network Application

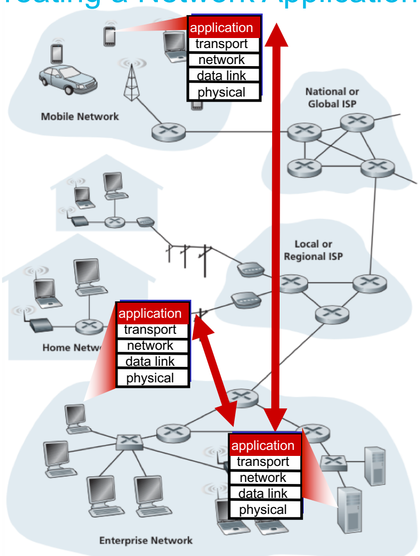
Creating a Network Application

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

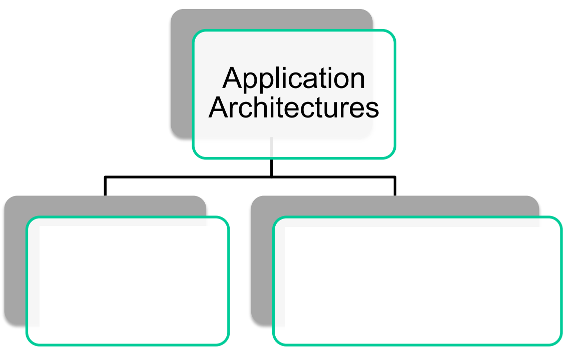
- network-core devices do not run user applications
- applications on *end systems* allows for rapid application development, propagation



2-5

CHAPTER 2

Network Application Architectures



(a) (b)

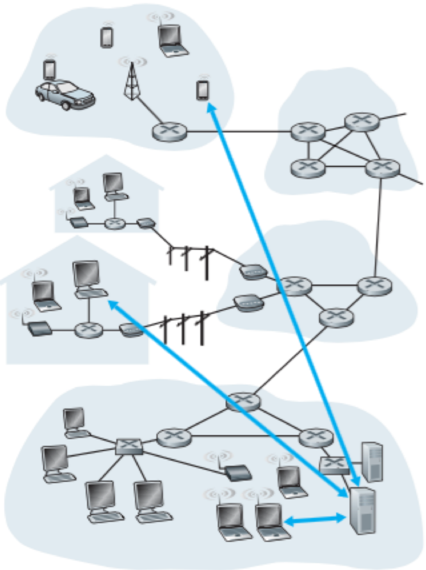
Figure: Possible structure of network applications

2-6

CHAPTER 2

Network Application Architectures

(a) Client-Server



______________:

- always-on host
- permanent IP address
- data centers for scaling

______________:

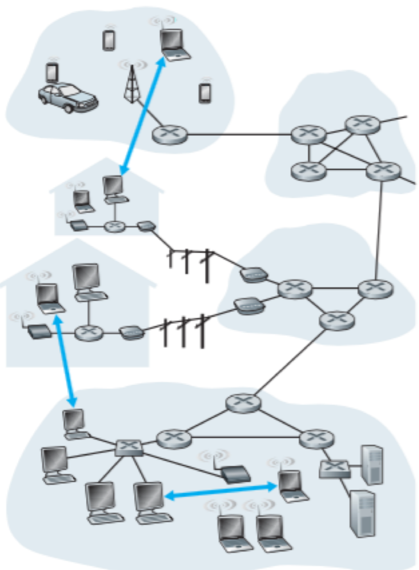
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

2-7

CHAPTER 2

Network Application Architectures

(b) Peer-to-Peer (P2P)



- *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

2-8 [BACK]

CHAPTER 2

Processes Communicating

Client and Server Processes

Process: program running within a host

- within same host → two processes communicate using inter-process communication (defined by OS)
- different hosts → processes communicate by exchanging messages

_____________: process that initiates communication

_____________: process that waits to be contacted

aside

- applications with P2P architectures have client processes and server processes

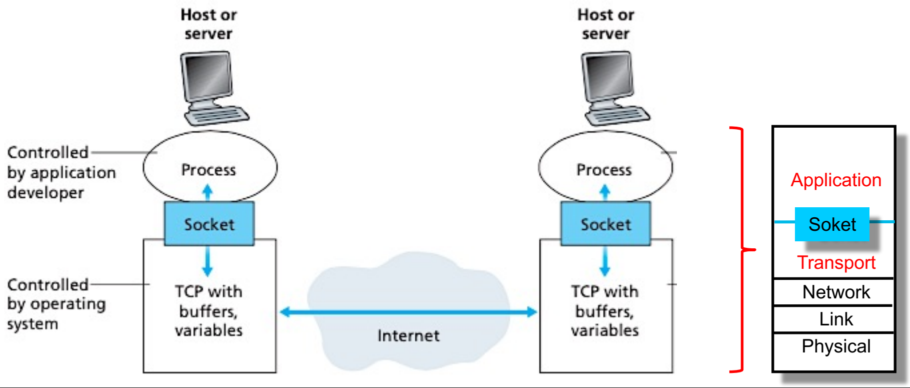
2-9

CHAPTER 2

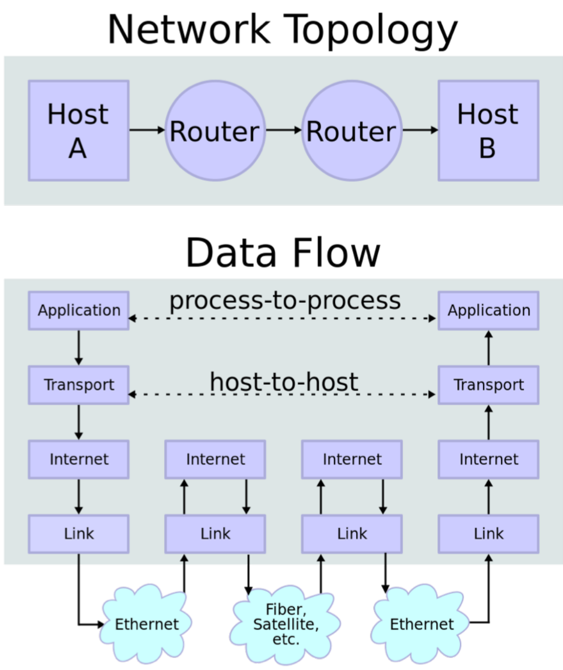
Processes Communicating

Interface: Sockets

- process sends/receives messages to/from its ___________
- socket analogous to door
  - sending process shoves (push) message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at *receiving process*



- Two Internet hosts connected via two routers and the corresponding layers used at each hop.
- The application on each host executes `read` and `write` operations as if the processes were directly connected to each other by some kind of data pipe.
- The underlying mechanisms that transmit **data** between the host computers are located in the lower protocol layers.



https://upload.wikimedia.org/wikipedia/commons/c/c4/IP_stack_connections.svg

2-11

CHAPTER 2

Processes Communicating

Addressing Processes

- to receive messages, process must have ________
- host device has unique 32-bit IP address

> Q: Does IP address of host on which process runs suffice (be adequate) for identifying the process?
>
> A: No, *many* processes can be running on same host

- *identifier* includes both IP address and ____________ associated with process on host.

- Example port numbers:
  - HTTP server: `80`
  - mail server: `25`
- to send HTTP message to *`www.utm.my`* web server:
  - IP address: `161.139.20.177`
  - port number: `80`

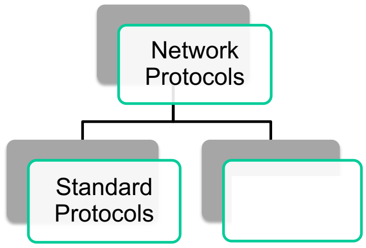
2-12

CHAPTER 2

Transport Service to Application

Application Layer Protocol Defines:

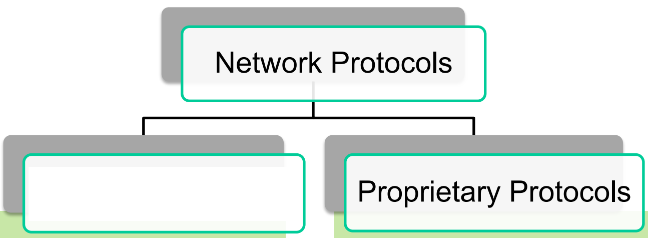
- types of messages exchanged,
  - *e.g.*, ___________, ___________
- message syntax:
  - what fields in messages & how fields are delineated (explained).
- message semantics
  - meaning of information in fields.
- rules for when and how processes send & respond to messages.



2-13

RFC (Request For Comments)
TCP/IP (Transmission Control Protocol / Internet Protocol)

- Two terms are often used in networking industry, when describing network protocols:



- Agreed and accepted by whole industry.
- Not vendor specific.
- Defined in RFCs
- Allows for interoperability.

- Developed by a single company for the devices (or OS) which they manufacture.
- Not scale well in multi-vendor equipment.

CHAPTER 2

Transport Service to Application

What Transport Service does an Application need?

Reliable data transfer

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer.
- other apps (e.g., audio) can tolerate some loss.

Throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get .

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective".

Security

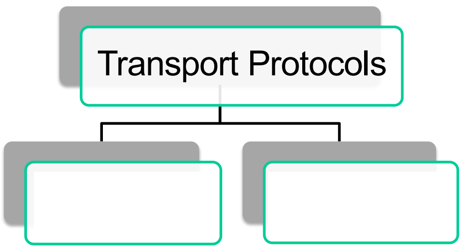
- ___________, data integrity.

2-15

CHAPTER 2

Transport Service Provided by Internet

- The Internet (generally, TCP/IP) makes the transport protocols available to applications:



- Application developers need to decide one of the transport protocol when creating a new network application.
- Each protocol offers a different set of services to the invoking applications.

UDP (User Datagram Protocol)
TCP (Transmission Control Protocol)
TCP/IP (Transmission Control Protocol / Internet Protocol)

2-16

CHAPTER 2

Transport Service Provided by Internet

Table: Transport service requirement - common applications

Application	Data Loss	Throughput	Time Sensitive
File transfer / download	No loss	Elastic	No
E-mails	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Text messaging	No loss	Elastic	Yes and No
Internet telephony / video conferencing	Loss-tolerant	Audio: 5kbps-1Mbps Video: 10kbps-5Mbps	Yes: 100's msec
Streaming stored audio video	Loss-tolerant	Audio: 5kbps-1Mbps Video: 10kbps-5Mbps	Yes: few secs
Interactive games	Loss-tolerant	Few kbps	Yes: 100's msec

2-17

CHAPTER 2

Transport Service Provided by Internet

TCP service:

- *reliable transport* between sending and receiving process.
- _______________: sender won't overwhelm receiver .
- _____________________: throttle (choke) sender when network overloaded.
- *does not provide:* timing, minimum throughput guarantee, security.
- *connection-oriented:* setup required between client and server processes.

UDP service:

- *unreliable data transfer* between sending and receiving process.
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Q: Why bother UDP?
Why is there a UDP?

2-18

CHAPTER 2

Transport Service Provided by Internet

SIP (Session Initiation Protocol)
RTP (Real-Time Transport Protocol)
RFC (Request For Comments)

Application	Application-Layer Protocol	Underlying Transport Protocol
E-mails	SMTP [RFC 5231]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP
Internet telephony	SIP [RFC 3261] RTP [RFC 3550] Proprietary (e.g. Skype)	UDP or TCP

Table: Popular Internet applications, their application-layer protocols, and their underlying transport protocols

Q: Why bother UDP?
Why is there a UDP?

2-19

CHAPTER 2

Transport Service Provided by Internet

Internet telephony application using UDP:

- Can often tolerate some loss.
- require minimum rates to be effective.
- Avoid TCP's congestion control mechanism and packet overheads.

Since many firewall are configured to block (most type of) UDP traffic, the application often designed to used TCP as backup if UDP fails.

Q: Why bother UDP?
Why is there a UDP?

2-20

CHAPTER 2 Transport Service Provided by Internet

Securing TCP

UDP (User Datagram Protocol)
TCP (Transmission Control Protocol)
API (Application Programming Interface)

TCP & UDP:

- no encryption.
- cleartext passwords sent into socket traverse Internet in cleartext.
- Potentially getting sniffed.

Solution:

SSL (____________________)

- provides encrypted TCP connection,
- data integrity, and
- end-point authentication.

SSL is at app layer

- Apps use SSL libraries, which "talk" to TCP.

SSL socket API

- cleartext passwords sent into socket traverse Internet are encrypted.

2-21

http://static1.squarespace.com/static/52ad1d91e4b00a98a27ba20e/52ae5168e4b0988b43f4361f/551196a8e4b0748e8e70b276/1427236855564/?format=1000w

CHAPTER 2 Network Applications

- New public domain and proprietary Internet applications are being developed everyday.
- This chapter will focus on some applications that are both pervasive and important:

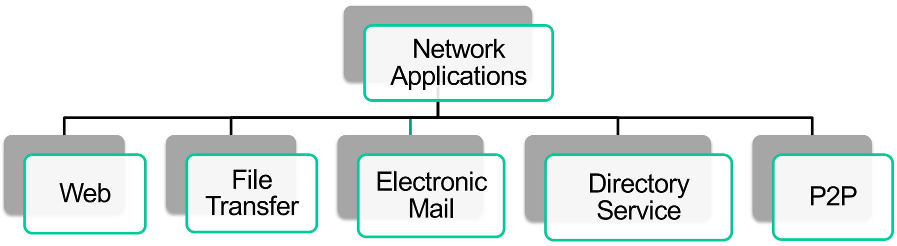


Figure: Some of important network applications

2-22

P2P (Point-to-Point)

CHAPTER 2

(2.2) The Web and HTTP

Introduction

- *Web page* consists of ___________.
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of *base HTML-file* which includes *several referenced objects.*
- Each object is addressable by a *URL (Uniform Resource Locator or Web address)* , e.g.,

```
www.utm.my/faculties-schools/pic.gif
```

host name: www.utm.my — path name: /faculties-schools/pic.gif

HTML (HyperText Markup Language)

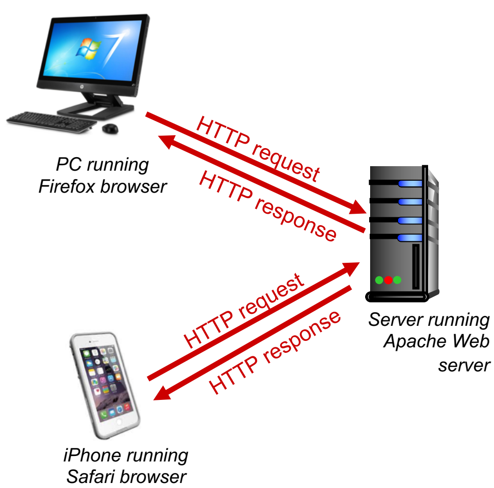
2-23

CHAPTER 2

Overview of HTTP

(HyperText Transfer Protocol)

- Web's application layer protocol.
- Client / server models:
  - ___________: browser that requests, receives, (using HTTP protocol) and "displays" Web objects.
  - *Server:* Web server sends (using HTTP protocol) objects in response to requests.



2-24

CHAPTER 2

Overview of HTTP

Uses TCP:

- client initiates TCP connection (creates ________) 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

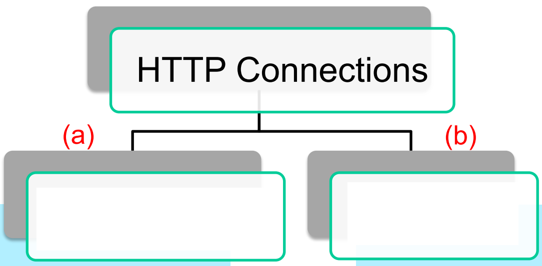
protocols that maintain "state" are complex!

- past history (state) must be maintained.
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled (resigned).

2-25

CHAPTER 2

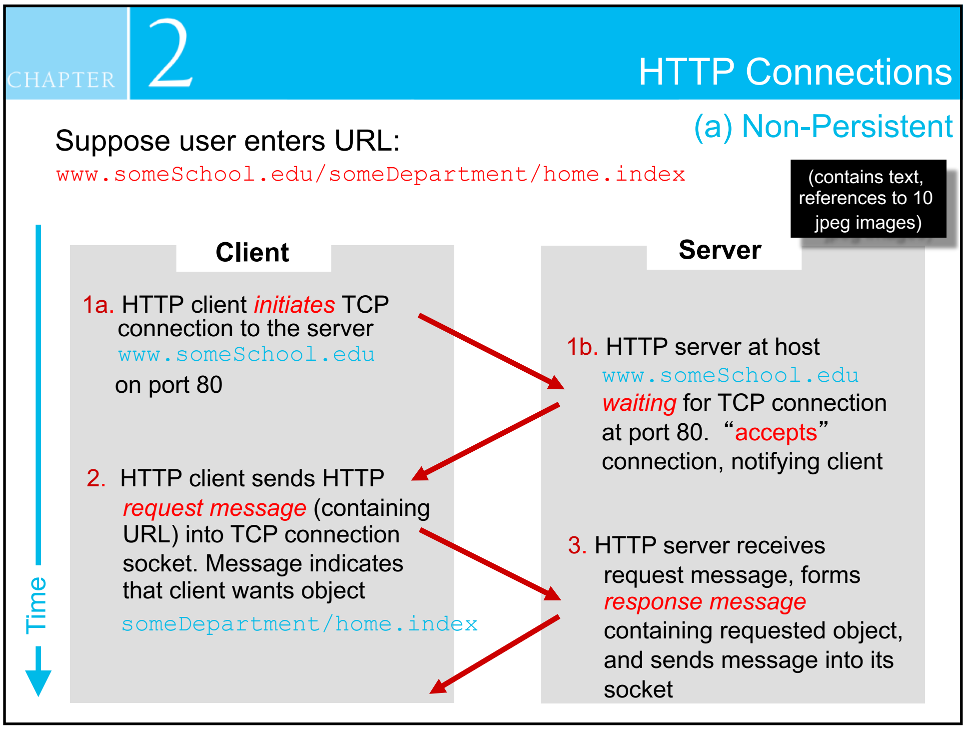
HTTP Connections

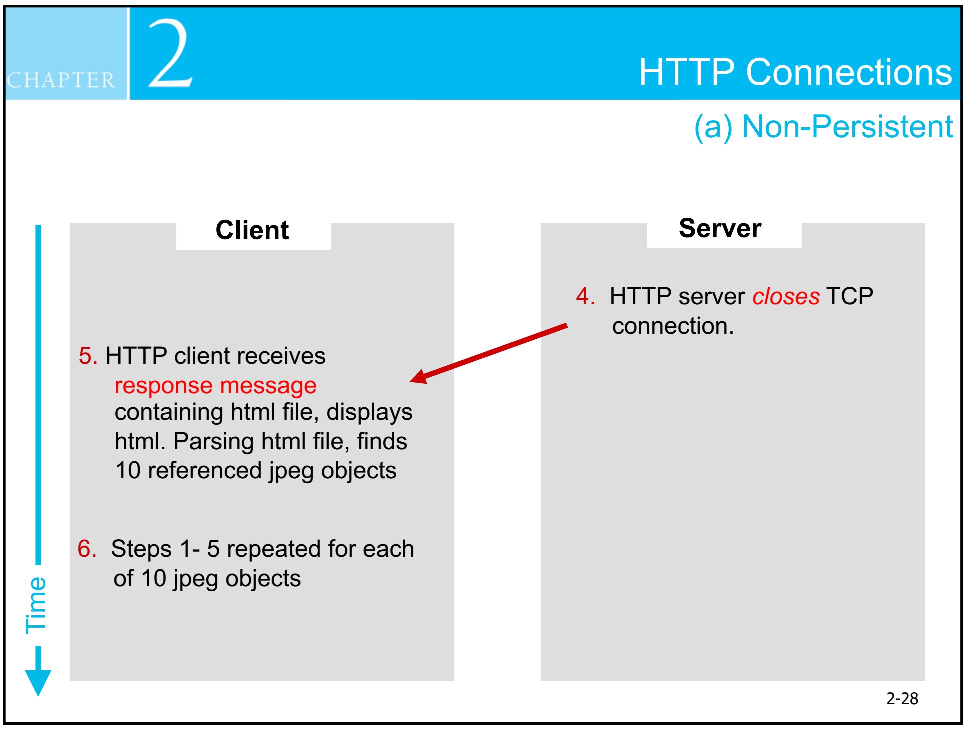


- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections.

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

2-26


CHAPTER 2
HTTP Connections
(a) Non-Persistent
Suppose user enters URL:
www.someSchool.edu/someDepartment/home.index
(contains text, references to 10 jpeg images)
Client
Server
1a. HTTP client initiates TCP connection to the server 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 URL) 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


CHAPTER 2
HTTP Connections
(a) Non-Persistent
Client
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
6. Steps 1- 5 repeated for each of 10 jpeg objects
Time
2-28

CHAPTER 2

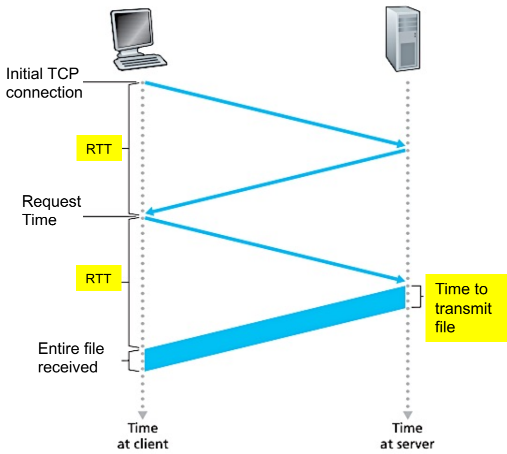
HTTP Connections

(a) Non-Persistent

Round-Trip Time (RTT):
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: $= 2RTT + (FileTransmissionTime)$

CHAPTER 2

HTTP Connections

(b) Persistent

Non-Persistent HTTP issues:

- requires 2 RTTs per object
- Operating System (OS) overhead for each TCP connection - initiate TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

(e.g. `index.html` contains text, references to 10 jpeg images)

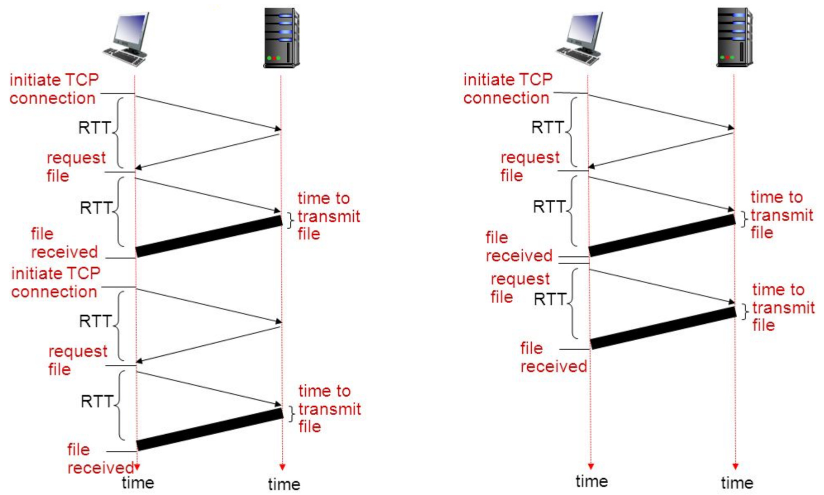
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

2-30

CHAPTER 2

HTTP Connections

Example: The HTTP connections for two objects.



(a) Non-persistent HTTP (b) Persistent HTTP

CHAPTER 2

HTTP Message Format

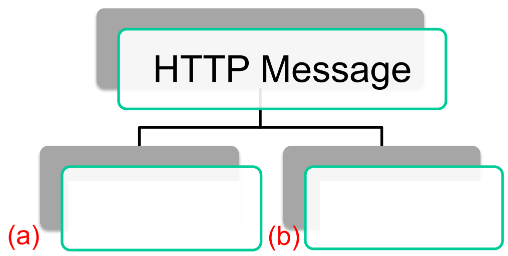


Figure: Types of HTTP messages

2-32

CHAPTER 2

HTTP Message Format

(a) Request Message

- Example a message written in ordinary ASCII text (human-readable format)

carriage return character
line-feed character

request line (GET, POST, HEAD commands)

```
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
```

header lines

(*carriage return, line feed*) at start of line indicates end of header lines

2-33

CHAPTER 2

HTTP Message Format

(a) Request Message

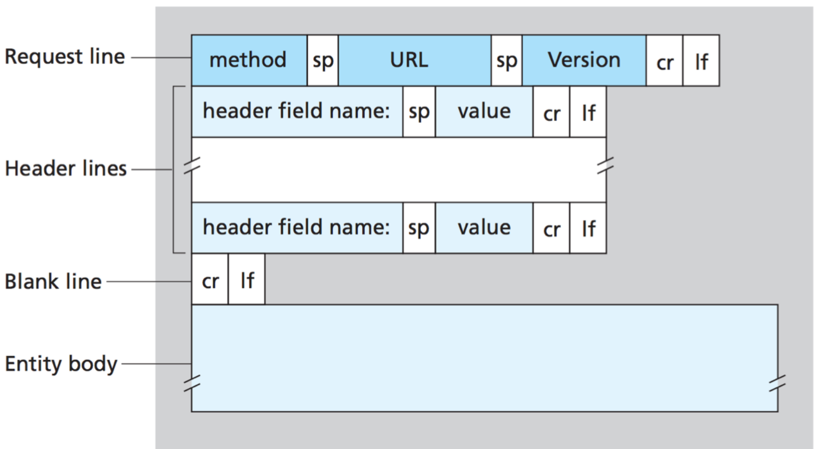


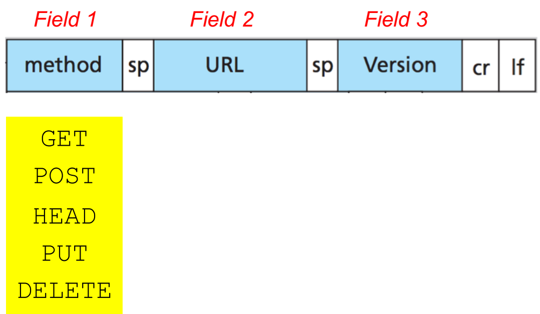
Figure: General format of an HTTP request message

cr (carriage return)
lf (line-feed)

CHAPTER 2

HTTP Message Format

(a) Request Message



- Majority of HTTP request messages use the `GET` method
- A browser requests an object with the request object identified in the URL field
- Example: `GET /index.html HTTP/1.1`

2-35

CHAPTER 2

HTTP Message Format

(b) Response Message

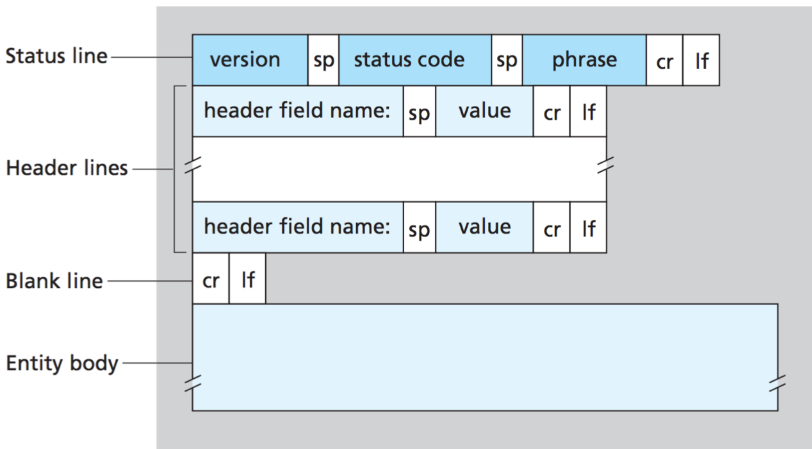


Figure: General format of an HTTP response message

cr (carriage return)
lf (line-feed)

CHAPTER 2

HTTP Message Format

(b) Response Message

status line (protocol status code status phrase)

header lines

```
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 ...
```

data, e.g., Requested HTML file

2-37

CHAPTER 2

HTTP Message Format

(b) Response Message

HTTP response status codes:

- status code appears in 1st line in server-to-client response message.

Codes	Description
`200 OK`	request succeeded, requested object later in this message
`301 Moved Permanently`	requested object moved, new `URL` specified in `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`	Requested HTTP protocol version not supported by the server

2-38

CHAPTER 1

Try Yourself 1

1. Telnet to your favorite Web server:

```
telnet www.manutd.com 80
```

Opens TCP connection to port 80 (default http server port) at `www.manutd.com`
Anything typed in sent to port 80 at `www.manutd.com`

2. Type in a `GET` http request:

```
GET /index.html HTTP/1.0
```

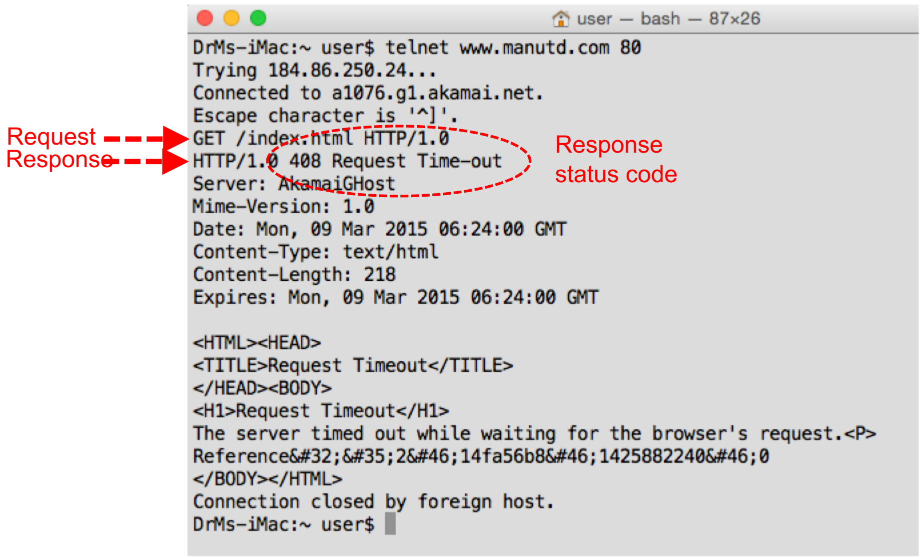
By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server
* you can't see what you are typing

3. Look at response message sent by http server!

1-39

CHAPTER 1

Try Yourself 1



CHAPTER 1

Try Yourself 2

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

Opens TCP connection to port 80 (default http server port) at `cis.poly.edu`
Anything typed in sent to port 80 at `cis.poly.edu`

2. Type in a GET http request:

```
GET /~ross/ HTTP/1.1
Host:cis.poly.edu
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server
* you can't see what you are typing

3. Look at response message sent by http server!

1-41

CHAPTER 1

Try Yourself 2

user — telnet — 87×26

```
DrMs-iMac:~ user$ telnet cis.poly.edu 80
Trying 128.238.32.79...
Connected to cis.poly.edu.
Escape character is '^]'.
GET /~ross/ HTTP/1.1
Host:cis.poly.edu

HTTP/1.1 301 Moved Permanently
Date: Mon, 09 Mar 2015 06:14:38 GMT
Server: Apache/1.3.41 (Unix) mod_perl/1.31
Location: http://nyu.edu/projects/keithwross/
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

ef
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://nyu.edu/projects/keithwross/">here</A>.<P>
</BODY></HTML>

0
```

Response status code

CHAPTER 2

User-Server Interaction: Cookies

Overview

- An **HTTP cookie** (also called **web cookie**, **Internet cookie**, **browser cookie** or simply **cookie**)
  - a small piece of data
  - sent from a website and stored on the user's computer by the user's web browser while the user is browsing.

https://en.wikipedia.org/wiki/HTTP_cookie

2-43

CHAPTER 2

User-Server Interaction: Cookies

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`"?

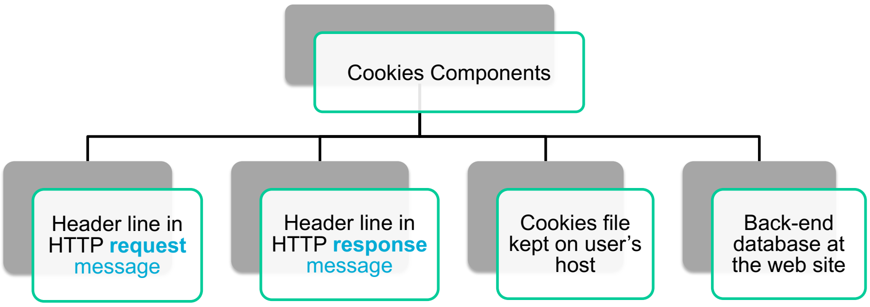
- protocol endpoints: maintain state at sender / receiver over multiple transactions
- cookies: http messages carry state

2-44

CHAPTER 2

User-Server Interaction: Cookies

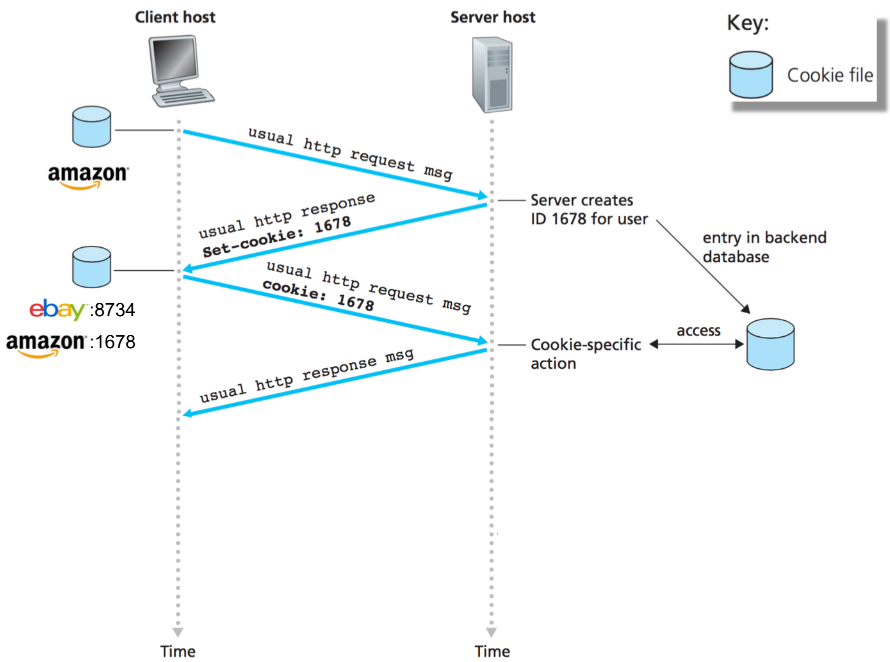
- Many Web sites use cookies

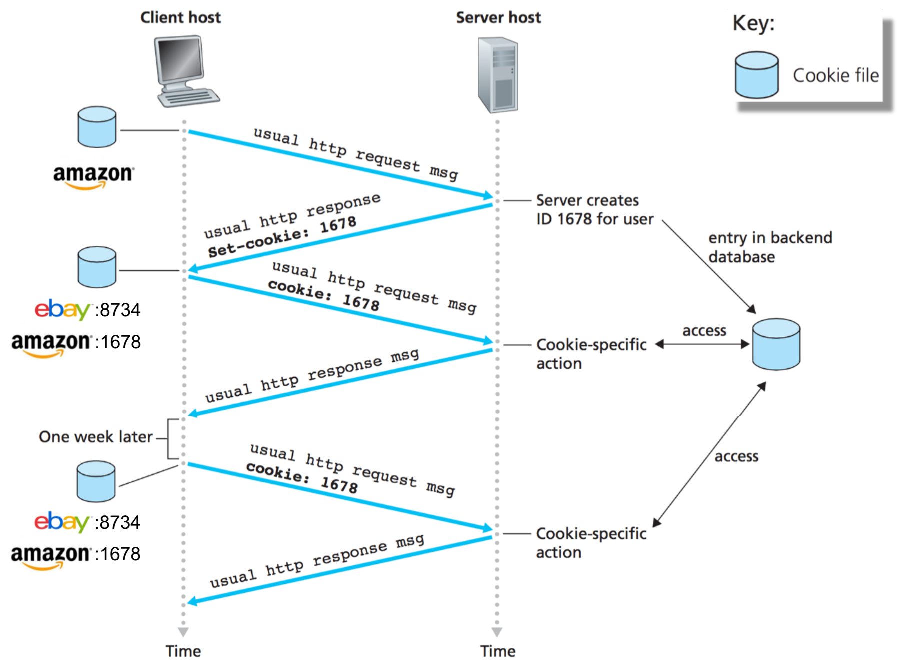


Example:

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

2-45





CHAPTER 2

Web Caching (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 in cache*: cache returns object
  - *Else*: cache requests object from origin server, then returns object to client

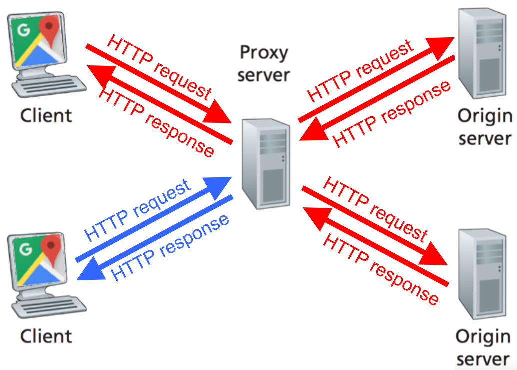


Figure: Clients requesting object through a Web cache

2-48

CHAPTER 2

Web Caching (Proxy Server)

- 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 (so too does P2P file sharing)

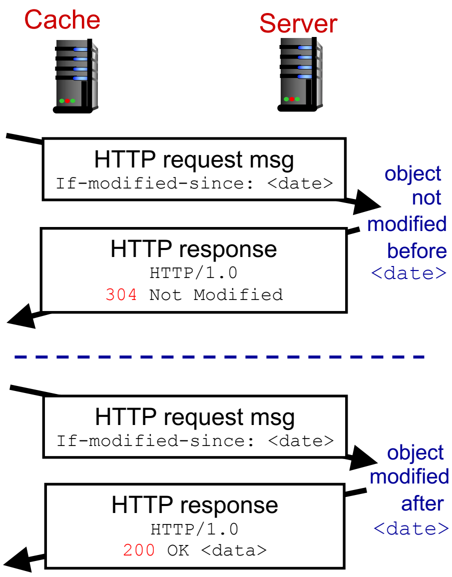
2-49

CHAPTER 2

Web Caching (Proxy Server)

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`



2-50

CHAPTER 2

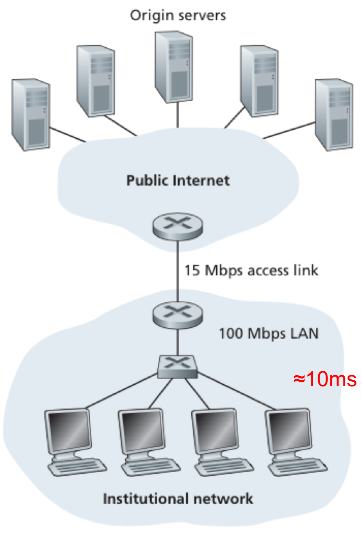
Web Caching (Proxy Server)

Example (a): Slim Access Link

Assumptions:

- Average object size: 1*Mbits*
- Average request rate from browsers to origin servers: 15 *requests/sec*
- RTT from institutional router to any origin server: 2 *sec* (Internet delay)
- LAN Delay: ≈10*ms*
- Access link rate: 15*Mbps*

Size Object, *L* = 1*Mbits*
R_{LAN} = 100*Mbps*
R_{Link} = 15*Mbps*
a = 15*req/sec*
Internet delay = 2*sec*



RTT (Round-Trip Time)

2-51

CHAPTER 2

Web Caching (Proxy Server)

Example (a): Slim Access Link

Consequences:

- Traffic Intensity LAN, La/R:

  $= ((1\times10^6)(15))/(100\times10^6)$

  $= 0.15 = 15\%$ (Utilization)

- Traffic Intensity access link, La/R :

  $= ((1\times10^6)(15))/(15\times10^6)$

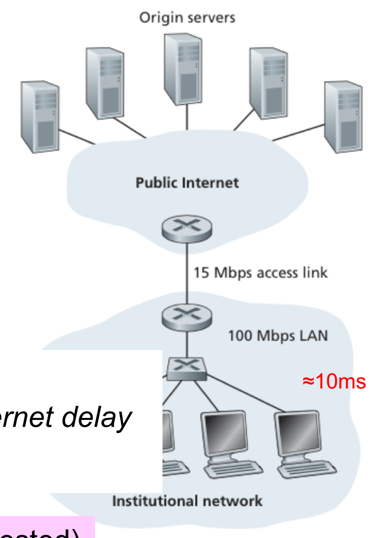
  $= 1 = 100\%$ (Utilization)

  problem!

- Total Response Time:

  = *LAN delay* + *access link delay* + *Internet delay*

  = *μsec* + *minutes* + 2*sec* = *minutes*



Traffic intensity= La/R → 1 (heavy congested)

2-52

CHAPTER 2

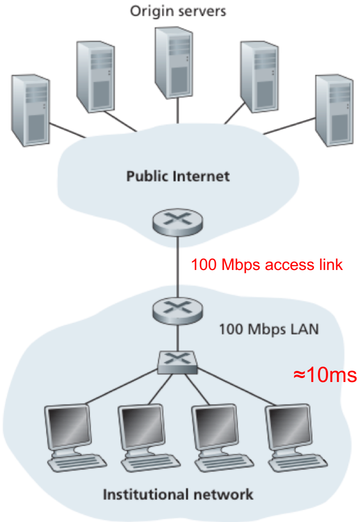
Web Caching (Proxy Server)

Example (b): Fatter Access Link

Assumptions:

- Average object size: 1*Mbits*
- Average request rate from browsers to origin servers: 15 *requests/sec*
- RTT from institutional router to any origin server: 2 *sec* (Internet delay)
- LAN Delay: ≈10*ms*
- Access link rate: 15*Mbps* → 100*Mbps*

Size Object, *L* = 1*Mbits*
R_{LAN} = 100*Mbps*
R_{Link} = 100*Mbps*
a = 15*req/sec*
Internet delay = 2*sec*



2-53

CHAPTER 2

Web Caching (Proxy Server)

Example (b): Fatter Access Link

Consequences:

- Traffic Intensity LAN, La/R:

  $= ((1\times10^6)(15))/(100\times10^6)$

  $= 0.15 = 15\%$ (Utilization)

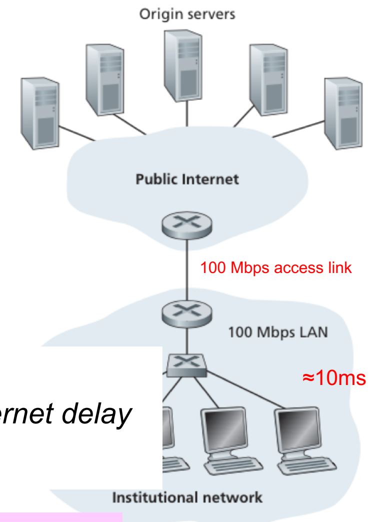
- Traffic Intensity access link, La/R :

  $= ((1\times10^6)(15))/(100\times10^6)$

  $= 0.15 = 15\%$ (Utilization)
  (reduce to 15%)

- Total Response Time:
  = *LAN delay* + *access link delay* + *Internet delay*
  = *μsec* + *μsec* + 2*sec* = *secs*

Cost: increased access link speed (not cheap!)



2-54

CHAPTER 2

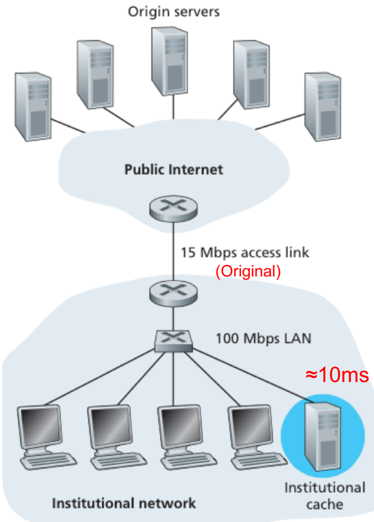
Web Caching (Proxy Server)

Example (c): Adding Local Cache

Assumptions:

- Average object size: 1*Mbits*
- Average request rate from browsers to origin servers: 15 *requests/sec*
- RTT from institutional router to any origin server: 2 *sec* (Internet delay)
- LAN Delay: ≈10*ms*
- Access link rate: 15*Mbps* → Original

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache; delay at cache = 10*msec*,
  - 60% requests satisfied at origin



2-55

CHAPTER 2

Web Caching (Proxy Server)

Example (c): Adding Local Cache

Consequences:

- Traffic Intensity LAN, $La/R = 15\%$
- Traffic Intensity access link, La/R :

$$= ((1 \times 10^6)(15))/(15 \times 10^6) * 60\%$$

$$= 0.6 = 60\%$$ (reduced from 1.0 to 0.6)

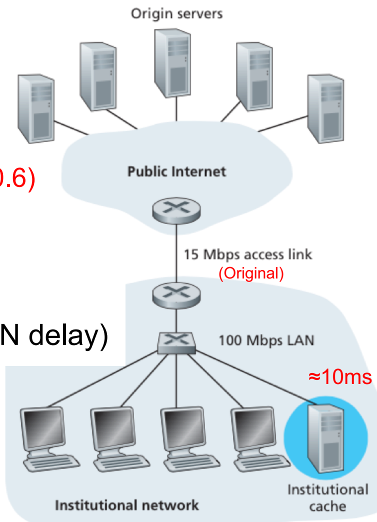
- Average Total Response Time:

= *0.4 * (delay at cache)* +
*0.6 * (delay from origin server* + LAN delay)

$$= 0.4 * 0.01 + 0.6 * (2 + 0.01)$$

$$= 0.004 + 1.206 = 1.21s$$

Cost: Web cache (cheap!)



2-56

CHAPTER 2

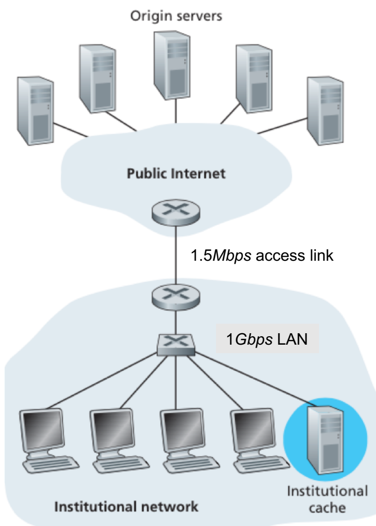
Exercise 2.2a

The diagram shows an institutional network connected to public Internet. Answer the following questions based on the assumption below:

Average object size: 100K bits, average request rate from browsers to origin servers: 15/sec, access link rate: 1.5 Mbps, & access LAN: 1Gbps

Calculate the access link utilization:

a) Without web cache server.
b) Without web cache server with the access link rate is increased to 3*Mbps*.
c) With web cache server which has a hit rate of 50% and the link capacity is unchanged (1.5Mbps)
d) Discuss the results in (b) and (c) (in terms of utilization). What is your conclusion?



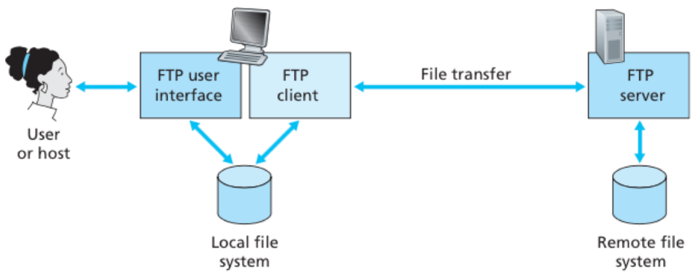
2-57

CHAPTER 2

(2.3) File Transfer: FTP

(File Transfer Protocol)

- transfer file to / from remote host
- Client / server model
  - *Client:* side that initiates transfer (either to/from remote)
  - *Server:* remote host
- FTP : RFC 959
- FTP server : *Port 21*



2-58

CHAPTER 2

Separate Control, Data Connections

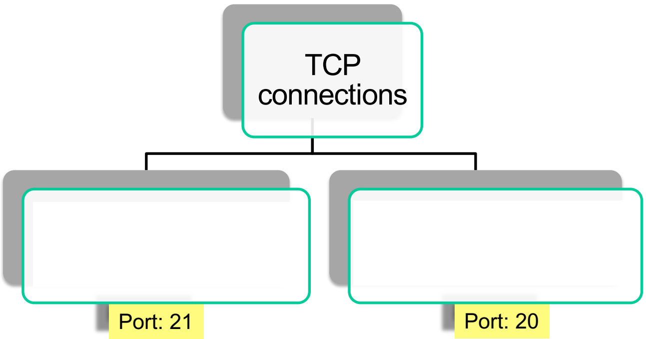
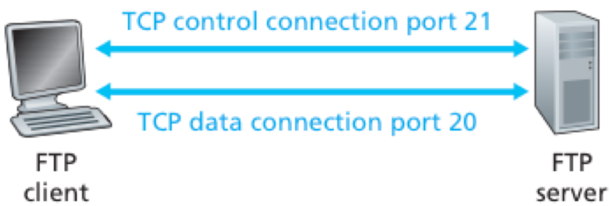


Figure: Two parallel TCP connections for FTP to transfer a file

2-59

CHAPTER 2

Separate Control, Data Connections

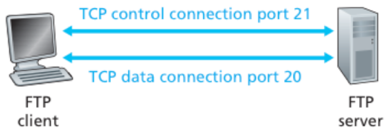


- FTP *client* contacts FTP server at `port 21`, using TCP control connection:
  - *client* authorized
  - *client* browses remote directory, sends commands
- when server receives file transfer command, *server* opens *2nd* TCP data connection (for file) *to client*
- after transferring one file, *server* closes data connection

2-60

CHAPTER 2

Separate Control, Data Connections



- server opens another TCP data connection to transfer another file
- Control connection:
  *"**Out of band**: Out-of-band control passes control data on a separate connection from main data."*
- FTP server maintains "state": current directory, earlier authentication

2-61

CHAPTER 2

FTP Commands and Reponses

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`

2-62

CHAPTER 1

Try Yourself 3

1. ftp to an ftp server using a terminal console:

```
> ftp
> open
> speedtest.tele2.net
> username: anonymous
> password: <enter>
```

2. Look at response message sent by ftp server!

drm — ftp — 66×17

```
Last login: Wed Mar  1 22:36:40 on ttys000
Drs-MacBook-Pro:~ drm$ ftp
ftp> open speedtest.tele2.net
Trying 90.130.70.73...
Connected to speedtest.tele2.net.
220 (vsFTPd 2.3.5)
Name (speedtest.tele2.net:drm): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

CHAPTER 1

Try Yourself 3

3. Then continue to view the content with command: `> ls -1`

4. What you got?

drm — ftp — 84×23

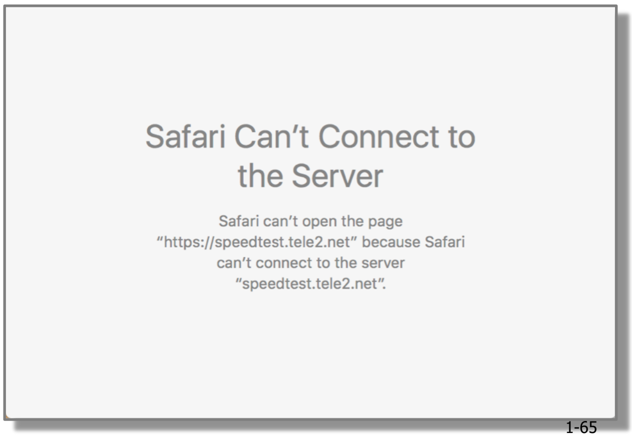
```
ftp> ls
229 Entering Extended Passive Mode (|||26041|).
150 Here comes the directory listing.
-rw-r--r--    1 0        0        107374182400 Feb 19  2016 1000GB.zip
-rw-r--r--    1 0        0        107374182400 Feb 19  2016 100GB.zip
-rw-r--r--    1 0        0          102400 Feb 19  2016 100KB.zip
-rw-r--r--    1 0        0        104857600 Feb 19  2016 100MB.zip
-rw-r--r--    1 0        0        10737418240 Feb 19  2016 10GB.zip
-rw-r--r--    1 0        0        10485760 Feb 19  2016 10MB.zip
-rw-r--r--    1 0        0        1073741824 Feb 19  2016 1GB.zip
-rw-r--r--    1 0        0            1024 Feb 19  2016 1KB.zip
-rw-r--r--    1 0        0         1048576 Feb 19  2016 1MB.zip
-rw-r--r--    1 0        0        209715200 Feb 19  2016 200MB.zip
-rw-r--r--    1 0        0        20971520 Feb 19  2016 20MB.zip
-rw-r--r--    1 0        0         2097152 Feb 19  2016 2MB.zip
-rw-r--r--    1 0        0         3145728 Feb 19  2016 3MB.zip
-rw-r--r--    1 0        0        524288000 Feb 19  2016 500MB.zip
-rw-r--r--    1 0        0        52428800 Feb 19  2016 50MB.zip
-rw-r--r--    1 0        0          524288 Feb 19  2016 512KB.zip
-rw-r--r--    1 0        0         5242880 Feb 19  2016 5MB.zip
drwxr-xr-x    2 105      108          4096 Mar 01 15:50 upload
226 Directory send OK.
ftp>
```

1-64

CHAPTER 1

Try Yourself 4

1. Type the URL address to connect to the ftp server using any web browser: https://speedtest.tele2.net
2. Look at response message.

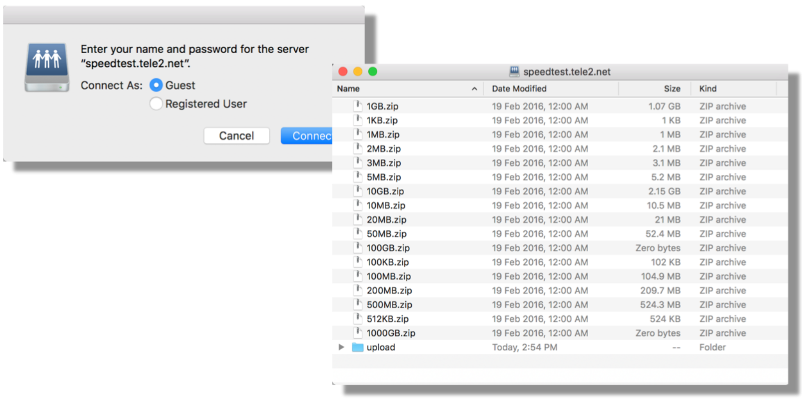


1-65

CHAPTER 1

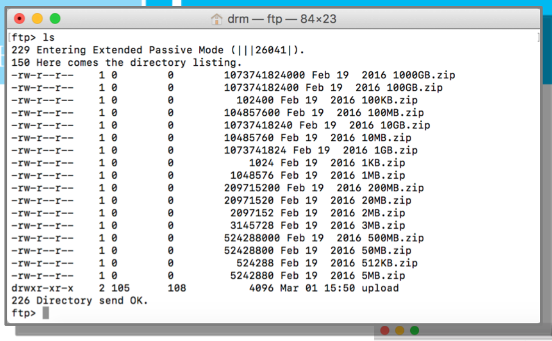
Try Yourself 4

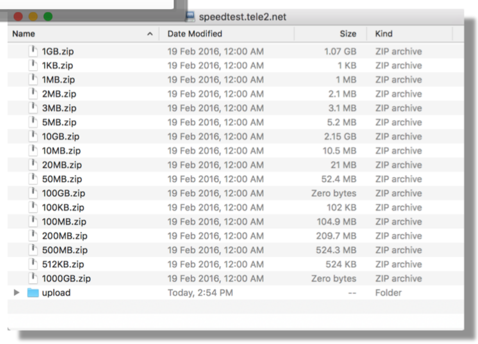
3. Now type the URL address to connect to the ftp server using any web browser: ftp://speedtest.tele2.net
4. Look at response message.



1-66

Try Yourself 4





Try Yourself 4

5. Compare to the method done before (using terminal).

1-67

CHAPTER 2

(2.4) Electronic Mail in the Internet

- E-mail is an asynchronous communication medium (people send and read messages) in their convenient time.
- In contrast with ordinary postal mail, e-mail: is fast, easy to distribute and inexpensive
- Modern e-mail has many powerful features: *hyperlinks, embedded photo*, …

E-Mail Components

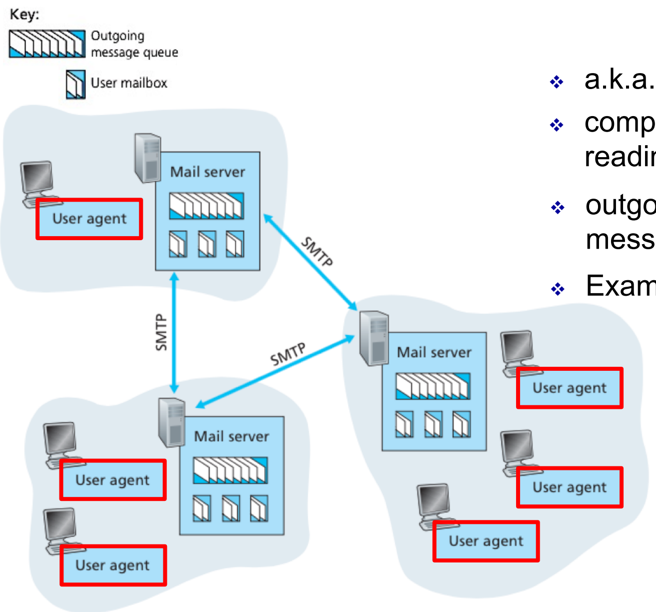
(a) (b) (c)

SMTP (Simple Mail Transfer Protocol)

2-68

CHAPTER 2

(a) User Agents

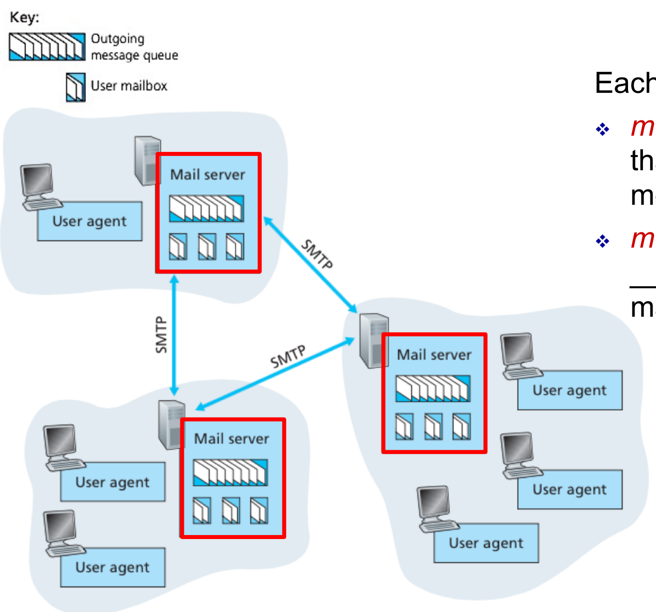


- a.k.a. "*mail reader*"
- composing, editing, reading mail messages
- outgoing, incoming messages stored on server
- Examples:

2-69

CHAPTER 2

(b) Mail Servers



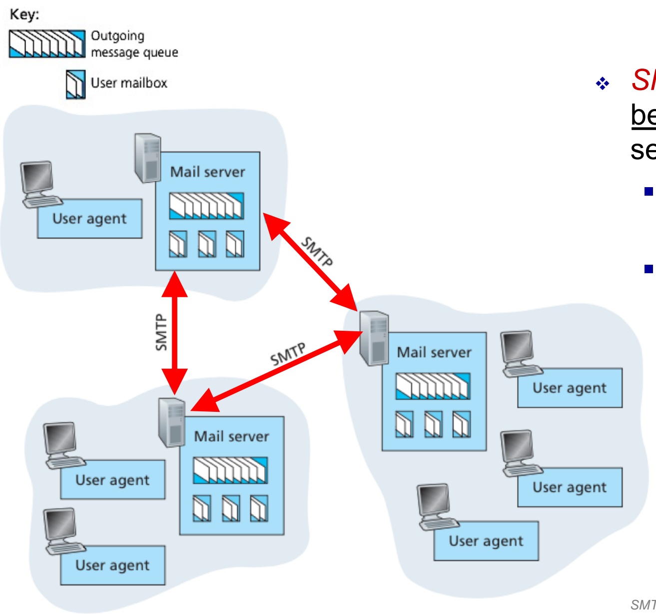
Each user has :

- *mailbox* in a mail server that contains __________ messages
- *message queue:* __________ (to be sent) mail messages

2-70

CHAPTER 2

(c) SMTP



- *SMTP protocol* : between mail servers to send email messages
  - *client*: sending mail server
  - "*server*": receiving mail server

SMTP (Simple Mail Transfer Protocol)

2-71

CHAPTER 2

SMTP [RFC 2821]

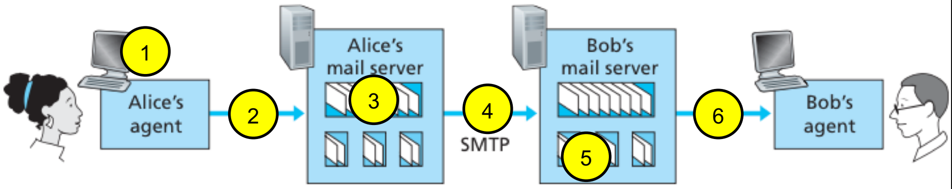
- Uses TCP to reliably transfer email message from client to server, port 25
- Direct transfer: sending server to receiving server
- Three phases of transfer:
  - handshaking (greeting)
  - transfer of messages
  - closure
- Command / response interaction (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- Messages must be in 7-bit ASCI

2-72

CHAPTER 2

SMTP [RFC 2821]

Example: Alice Sends Message to Bob



1) Alice uses UA to compose message "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

2-73

CHAPTER 2

SMTP [RFC 2821]

Sample SMTP Interaction

```
> telnet mel.utm.my 25
     S: 220 mail-2 SMTP Server <Desknow> ready Tue <MYT>
     C: HELO yahoo.com
     S: 250 mail-2 Hello yahoo.com, <10.60.113.56>
     C: MAIL FROM: <alice@yahoo.com>
     S: 250 Sender anita@yahoo.com OK
     C: RCPT TO: <marinama@utm.my>
     S: 250 Recipient marinama@utm.my OK
     C: DATA
     S: 354 Enter mail, end with "." on a line by itself
     C: Do you like nasi kerabu?
     C:   How about nasi lemak?
     C: .
     S: 250 Message accepted for delivery
     C: QUIT
     S: 221 mel.utm.my closing connection
```

2-74

CHAPTER 2

SMTP [RFC 2821]

"Final word"

- 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

HTTP
- protocol
- Each object encapsulated in its own response message

SMTP
- protocol
- Multiple objects sent in multipart messages

- both (HTTP & SMTP) have ASCII *command / response* interaction, *status codes*

2-75

CHAPTER 2

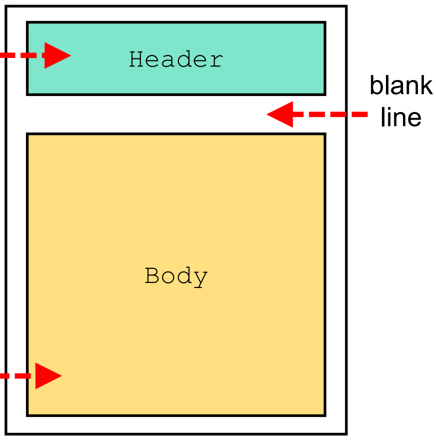
Mail Message Formats

- SMTP: protocol for exchanging email messages
- RFC 822: standard for text message format:
- Header lines, e.g.,

```
To: muhalim@utm.my
From: nzah@utm.my
Subject: Testing
```

Different from SMTP MAIL `FROM, RCPT TO:` commands!

- Body: the "message"
  - ASCII characters only



2-76

CHAPTER 2

Mail Message Protocol

- **SMTP:** delivery / storage to receiver's server

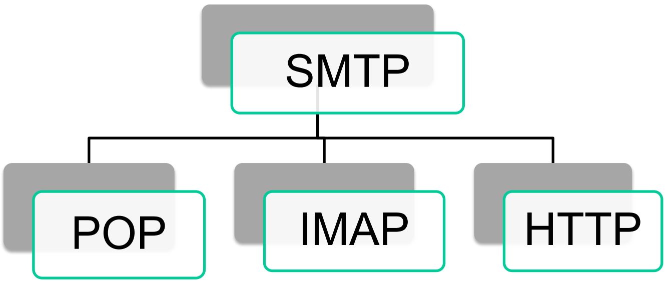


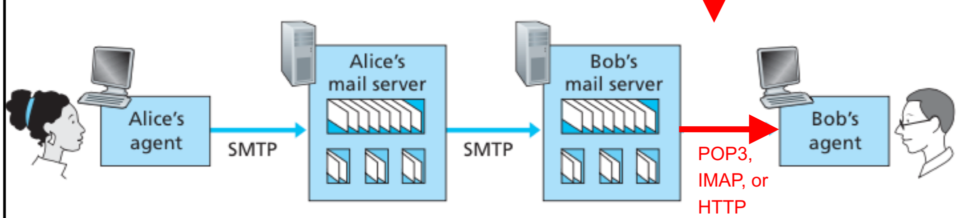
Figure: Three retrieval mail access protocol from server

POP (Post Office Protocol)
SMTP (Simple Mail Transfer Protocol)
IMAP (Internet Mail Access Protocol)
HTTP (Hyper Text Transfer Protocol)

CHAPTER 2

Mail Message Protocol

- POP: *Post Office Protocol* [RFC 1939]: authorization, download
- IMAP: *Internet Mail Access Protocol* [RFC 1730]: more features, including manipulation of stored messages on server
- HTTP: Gmail, Windows Live Hotmail, Yahoo!



2-78

CHAPTER 2

Mail Message Protocol

POP3

Authorization phase

- client commands:
  - `user`: declare username
  - `pass`: password
- server responses
  - +OK
  - -ERR

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

Transaction phase, client:

- `list`: list message numbers
- `retr`: retrieve message by number
- `dele`: delete
- `quit`

```
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
```

POP3 (Post Office Protocol 3)

2-79

CHAPTER 2

Mail Message Protocol

POP3

More about POP3

- previous example uses POP3 "***download and delete***" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "***download-and-keep***": copies of messages on different clients
- POP3 is **stateless** across sessions

2-80

CHAPTER 2

Mail Message Protocol

IMAP

- keeps all messages in one place: at **server**
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

IMAP (Internet Mail Access Protocol)

2-81

CHAPTER 2

(2.5) Directory Service: DNS

Overview

People: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., `www.yahoo.com` - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System (DNS): Port 53

- *Distributed database* implemented in hierarchy of many *name servers*
- *Application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

SSN (Social Security Number)

2-82

CHAPTER 2

DNS Services

- *Hostname to IP address translation*
- *Host aliasing*
  - canonical, alias names
  - Examples:

    `relay1.west-coast.enterprise.com` @

    `enterprise.com` or `www.enterprise.com`
- *Mail server aliasing*
  - Examples:

    `relay1.west-coast.hotmail`@ `enterprise.com`
- *Load distribution*
  - replicated Web servers: many IP addresses correspond to one name

2-83

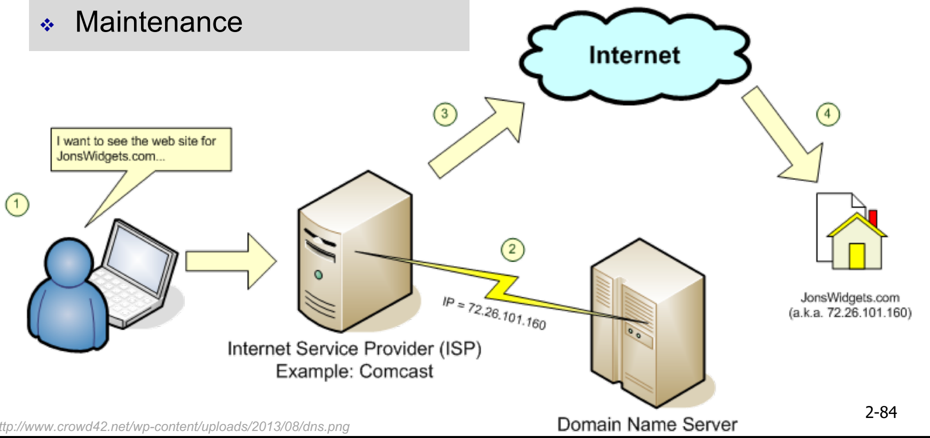
CHAPTER 2

DNS Structure

Why not centralize DNS?
- Single point of failure
- Traffic volume
- Distant centralized database
- Maintenance

A: *Doesn't scale!*

DNS: Centralize vs Distributed?



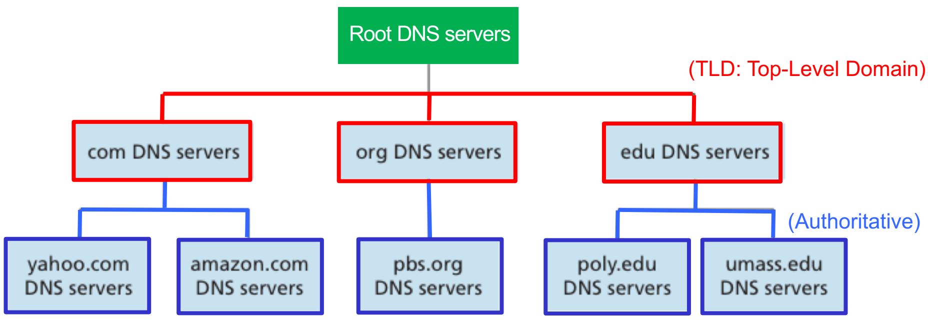
http://www.crowd42.net/wp-content/uploads/2013/08/dns.png

2-84

CHAPTER 2

DNS Structure

A Distributed, Hierarchical Database

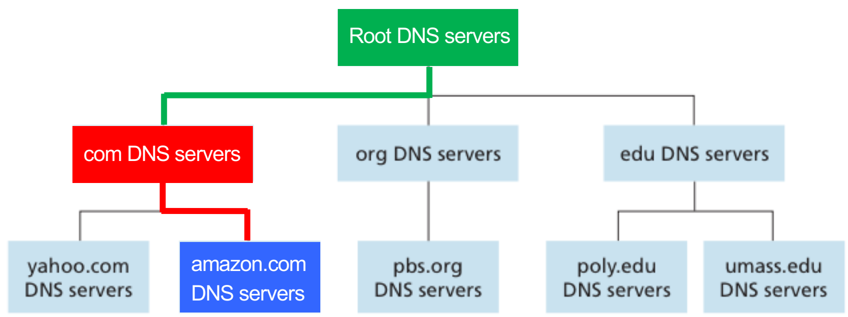


2-85

CHAPTER 2

DNS Structure

A Distributed, Hierarchical Database



Client wants IP for `www.amazon.com`; 1st approximation:

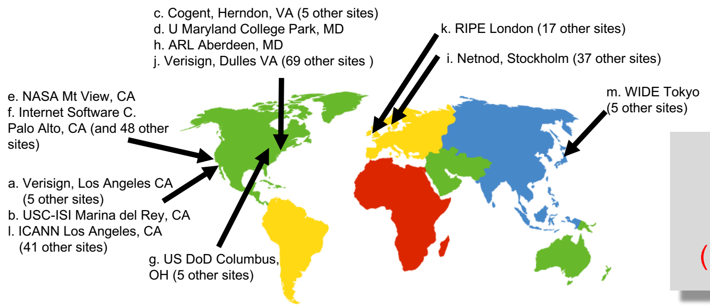
- client queries root server to find `.com` DNS server
- client queries `.com` DNS server to get `amazon.com` DNS server
- client queries `amazon.com` DNS server to get IP address for `www.amazon.com`

CHAPTER 2

DNS Structure

(a) Root Name Server

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



13 root name "servers" worldwide

(North America)

CHAPTER 2

DNS Structure

(b) TLD Server, (c) Authoritative DNS Server

Top-Level Domain (TLD) servers:

- responsible for `com`, `org`, `net`, `edu`, `gov` and all top-level country domains, *e.g.*: `uk`, `fr`, `ca`, `jp`
- Maintainer:
  - network solutions maintains servers for `.com` TLD
  - Educause for `.edu` TLD

Authoritative DNS servers:

- Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

2-88

CHAPTER 2

DNS Structure

Local DNS Server

- Another important type of DNS server
- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one
  - also called "*default name server*"
- When host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as *proxy*, forwards query into hierarchy

2-89

CHAPTER 2

DNS Structure

Local DNS Server

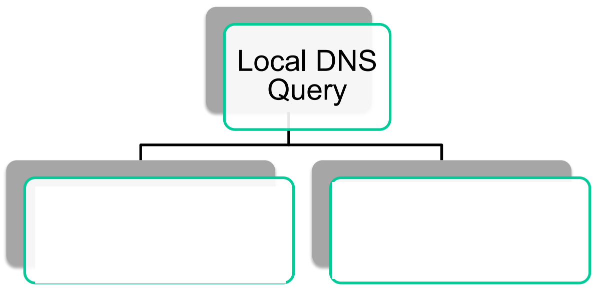


Figure: Two type of Local DNS Server queries

- Theoretically, any DNS query can be recursive or iterative.

2-90

CHAPTER 2

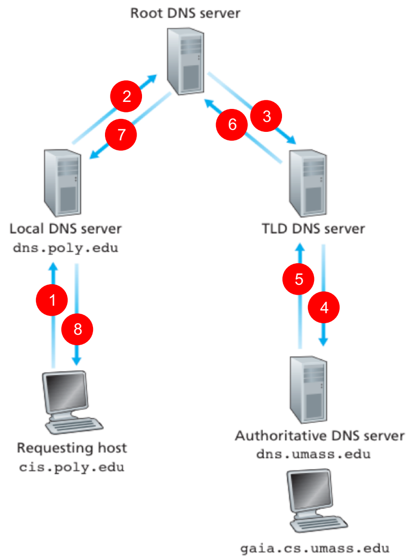
DNS Structure

Local DNS Server

Example 1:

Recursive query:
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?

The DNS queries typically follow the pattern in next Example 2:
- The query from host to local DNS server is recursive
- The remaining queries are iterative



TLD (Top-Level Domain)

CHAPTER 2

DNS Structure

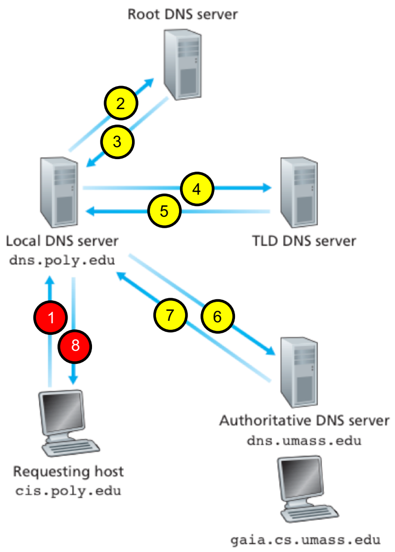
Local DNS Server

Example 2:

Host at `cis.poly.edu` wants IP address for `gaia.cs.umass.edu`

Recursive query:

Iterative query:
- contacted server replies with name of server to contact
- *"I don't know this name, but ask this server"*



TLD (Top-Level Domain)

CHAPTER 2

DNS Structure

DNS Caching

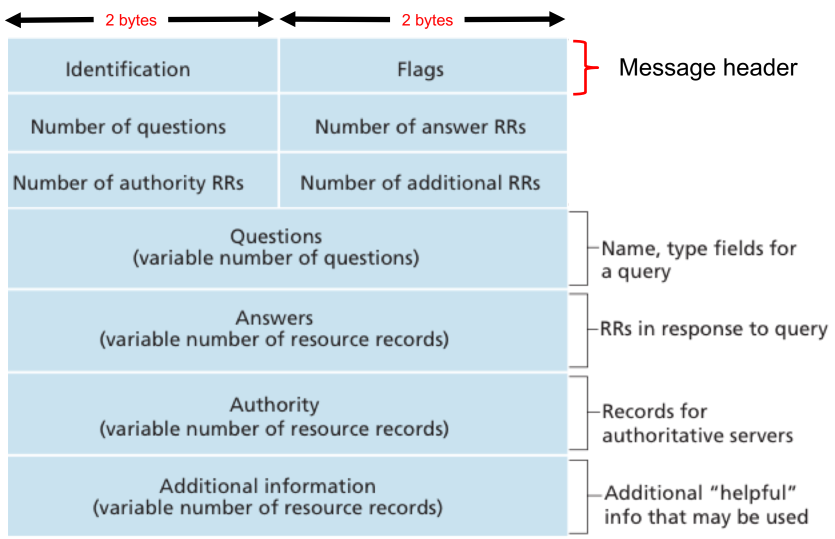
- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire (e.g. keep for 2 days)
- update/notify mechanisms proposed IETF standard
  - RFC 2136

TLD (Top-Level Domain)
TTL (Time-to-Live)
IETF (Internet Engineering Task Force)

CHAPTER 2

DNS Records and Messages

- *Query* and *reply* messages, both with same *message format*



CHAPTER 2

DNS Records and Messages

```
C:\Windows\system32\cmd.exe - nslookup
C:\Users\SKK>nslookup
Default Server:  ns2.utm.my
Address:  161.139.16.2

> www.cisco.com
Server:  ns2.utm.my
Address:  161.139.16.2

Non-authoritative answer:
Name:    e144.dscb.akamaiedge.net
Addresses:  2600:1417:9:195::90
          2600:1417:9:193::90
          184.26.192.170
Aliases:  www.cisco.com
          www.cisco.com.akadns.net
          wwwds.cisco.com.edgekey.net
          wwwds.cisco.com.edgekey.net.globalredir.akadns.net

> www.moe.gov.my
Server:  ns2.utm.my
Address:  161.139.16.2

Non-authoritative answer:
Name:    www.moe.gov.my
Address:  49.236.206.245

> www.google.com
Server:  ns2.utm.my
Address:  161.139.16.2

Non-authoritative answer:
Name:    www.google.com
Addresses:  2404:6800:4003:808::1013
          173.194.117.112
          173.194.117.115
          173.194.117.114
          173.194.117.113
          173.194.117.116
```

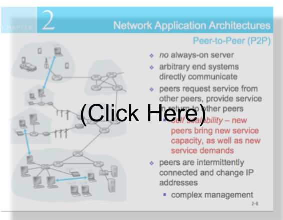
Q: How to send a DNS query message directly from the host to some DNS server?

A: Use *nslookup* program

CHAPTER 2

(2.6) Peer-to-Peer Applications

(Has been mentioned in slide 8)



Streaming Video

livestream

KanKan

P2P File Sharing

µtorent

Bittorrent

VoIP

Skype

2-96

CHAPTER 2

P2P File Distribution

BitTorrent™

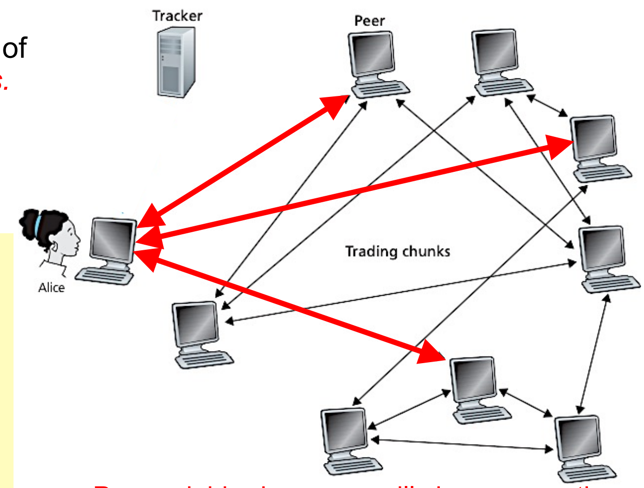
- *Torrent*: the collection of all peers participating in the distribution of a particular file .
- Typical chunk size of a file = 256 *Kbytes.*
- Peers in torrent send/receive file chunks.

Example:

Alice arrives …

… obtains list of peers from tracker

… and begins exchanging file chunks with peers in torrent



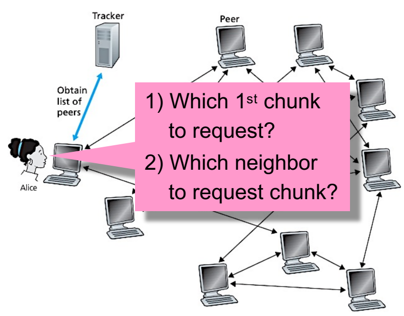
Peer neighboring peers will change over time

CHAPTER 2

P2P File Distribution

BitTorrent™

- Peer joining torrent:
  - has no chunks, but will *accumulate* them over time from other peers.
  - *registers* with ___________ to get list of peers, connects to subset of peers (“neighbors”).



- while *downloading*, peer *uploads* chunks to other peers.
- peer may *change* peers with whom it exchanges chunks.
- *Churn:* peers may come and go.
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent.

2-98

CHAPTER 2

P2P File Distribution

BitTorrent™

Requesting chunks:

- at any given time, different peers have different subsets of file chunks.
- periodically, Alice asks each peer for list of chunks that they have.
- Alice requests missing chunks from peers, __________ technique:
  - the chunks with fewest repeated copies among her neighbors
  - more quickly redistributed to equalized the numbers of copies for each chunk.

Sending chunks: Tit-For-Tat

- Alice sends chunks to those 4 peers currently sending her chunks *at highest rate:*
  - other peers are *choked* by Alice (do not receive chunks from her).
  - re-evaluate top 4 every 10 *secs.*
- every 30 *secs*: randomly select another peer, starts sending chunks
  - "optimistically *unchoke*" this peer
  - newly chosen peer may join top 4

2-99

CHAPTER 2

P2P File Distribution

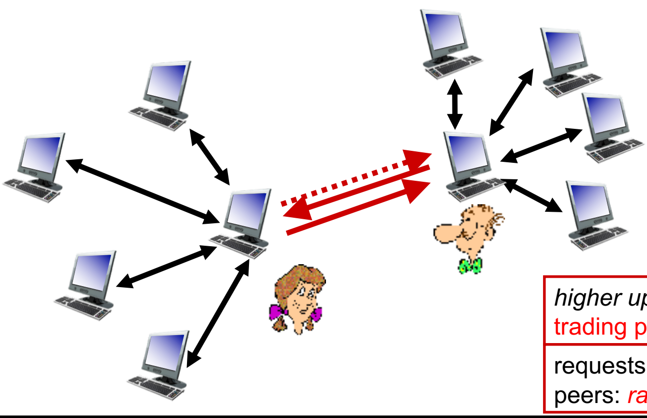
BitTorrent™

Tit-For-Tat :

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster !

requests missing chunks from peers: *rarest first* technique

CHAPTER 2 Summary

Our study of network applications now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent

2-101

CHAPTER 2 Summary

most importantly: learned about protocols!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - *headers*: fields giving info about data
  - *data*: info being communicated

important themes:

- control vs. data messages
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable message transfer
- “complexity at network edge”

2-102