

# CS2505

## Section 2 - Application Layer

Copyright Notice: The CS2505 lecture notes are adapted from material provided by J.F. Kurose and K.W. Ross and includes material by C.J. Sreenan, L.L. Peterson, B.S. Davie, M. Freedman, K. Gummadi and others. This material is copyrighted and so these lecture notes must not be copied or distributed without permission.

The collage illustrates various applications and network concepts in the Application Layer:

- Virtual Worlds:** A screenshot of the Second Life virtual world interface, showing avatars and a virtual environment.
- Contact Lists:** A screenshot of a contact list from a messaging application, showing names and status.
- University Website:** A screenshot of the University of Colorado (UCC) website, showing the university's logo and navigation links.
- YouTube:** A screenshot of a YouTube page, showing a video player and channel information.
- Diagram:** A diagram illustrating the DHT (Distributed Hash Table) network for public and private torrents. It shows a central DHT network connected to various peers and trackers. Public trackers are shown on the left, and private trackers are shown on the right. The diagram also includes a legend for different types of peers (e.g., Abacus, Marlin, aTornet, Kurose, BitComet, ABC, Tracker).

CS2505: Application Layer 2

## Section 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
- 2.5 DNS
- 2.6 File Distribution

CS2505: Application Layer

3

## Section 2: Application Layer

### Our goals:

- conceptual, implementation aspects of network application protocols
  - ❖ transport-layer service models
  - ❖ client-server paradigm
  - ❖ peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - ❖ HTTP
  - ❖ FTP
  - ❖ SMTP / POP3 / IMAP
  - ❖ DNS

CS2505: Application Layer

4

## Section 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
- 2.5 DNS
- 2.6 File Distribution

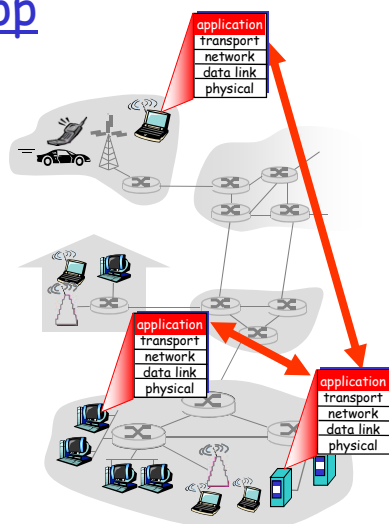
## Creating a network app

### 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

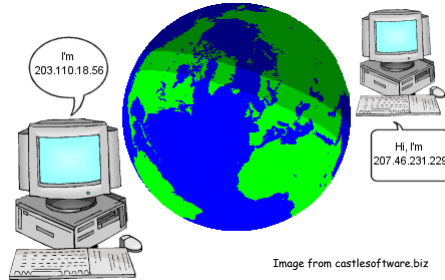
- ❖ Network-core devices do not run user applications
- ❖ applications on end systems allows for rapid app development, propagation



## Internet Addressing

Every computer on the Internet gets a unique "IP address"

- ❖ Assigned when it connects to the network
- ❖ Used to identify the destination when sending a packet
- ❖ Used by receivers to check who sent the packet (and to reply!)



For convenience we often use hostnames

- ❖ But these are mapped to a corresponding IP address when sending a packet
- ❖ E.g. cs1.ucc.ie → 143.239.75.218

CS2505: Application Layer

7

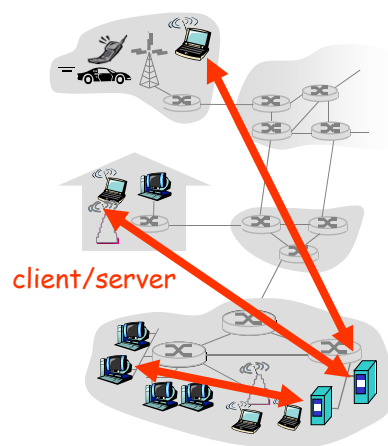
## Application architectures

- ❑ Client-server
  - ❖ Including data centers / cloud computing
- ❑ Peer-to-peer (P2P)
- ❑ Hybrid of client-server and P2P

CS2505: Application Layer

8

## Client-server architecture



### server:

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

### clients:

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

CS2505: Application Layer 9

## Google Data Centers

- ❑ Estimated cost of data center: \$600M
- ❑ Large Internet companies such as Google spend many billions of Euro on data centers

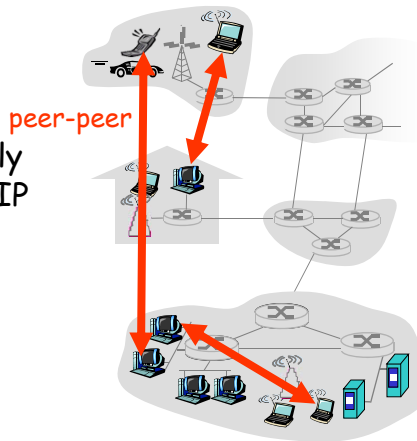


CS2505: Application Layer

10

## Pure P2P architecture

- ❑ no always-on server
- ❑ arbitrary end systems directly communicate
- ❑ peers are intermittently connected and change IP addresses



Highly scalable but  
difficult to manage

CS2505: Application Layer 11

## Hybrid of client-server and P2P

### Skype

- ❖ voice-over-IP P2P application
- ❖ centralized server: finding address of remote party:
- ❖ client-client connection: direct (not through server)

### Instant messaging

- ❖ chatting between two users is P2P
- ❖ centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

CS2505: Application Layer 12

## Processes communicating

**Process:** program running within a host.

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

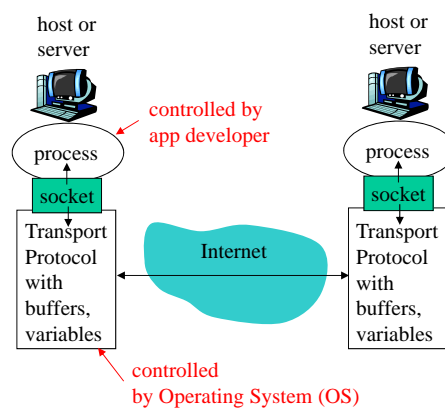
**Client process:** process that initiates communication

**Server process:** process that waits to be contacted

Note: applications with P2P architectures have client processes & server processes

## Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
  - ❖ sending process shoves message out door
  - ❖ sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process



API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)

## Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Exercise: use ipconfig from command prompt to get your IP address (Windows)
- Q: does IP address of host on which process runs suffice for identifying the process?
  - ❖ A: No, *many* processes can be running on same
- *Identifier* includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - ❖ HTTP server: 80
  - ❖ Mail server: 25

CS2505: Application Layer 15

## App-layer protocol defines

- Types of messages exchanged,
    - ❖ e.g., request, response
  - Message syntax:
    - ❖ what fields in messages & how fields are delineated
  - Message semantics
    - ❖ meaning of information in fields
  - Rules for when and how processes send & respond to messages
- Public-domain protocols:**
- defined in RFCs
  - allows for interoperability
  - e.g., HTTP, SMTP, BitTorrent
- Proprietary protocols:**
- e.g., Skype, ppstream

CS2505: Application Layer 16



## What transport service does an app need?

### Data loss

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

### Timing

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

### 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

### Security

- Encryption, data integrity, ...

CS2505: Application Layer 17

## Transport service requirements of common apps

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

CS2505: Application Layer 18

## Popular Internet transport protocols

### Transmission Control Protocol (TCP)

- ❑ Offers reliable end-to-end delivery between the sending and receiving processes
- ❑ Uses sequence numbers to detect lost or out-of-order packets; these are retransmitted (NB: extra delay)
- ❑ Additional features to control the sending rate so as to match the available network and receiver buffer capacity
- ❑ does not provide: timing, minimum throughput guarantees, security

### Unreliable Datagram Protocol (UDP)

- ❑ A very basic delivery service that offers no assurance about delivery between sending and receiving process
- ❑ does not provide: timing, minimum throughput guarantees, security, rate control

Q: why bother? Why is there a UDP?

CS2505: Application Layer 19

## Internet apps: application, transport protocols

<u>Application</u>	<u>Application layer protocol</u>	<u>Underlying transport protocol</u>
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

CS2505: Application Layer 20

## Section 2: Application layer

- ❑ 2.1 Principles of network applications
- ❑ 2.2 Web and HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Electronic Mail
- ❑ 2.5 DNS
- ❑ 2.6 File Distribution

CS2505: Application Layer 21

## Web and HTTP

### First some jargon

- ❑ Web page consists of objects
- ❑ 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
- ❑ Example URL:

`http://www.cs.ucc.ie/pic.gif`

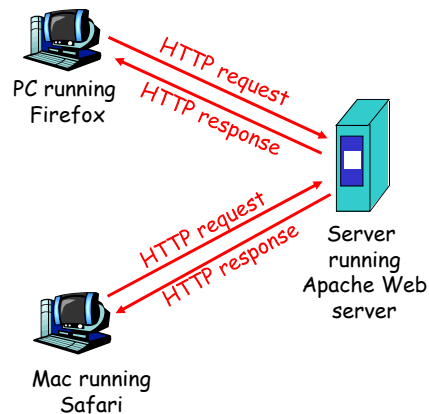
protocol      host name      path name

CS2505: Application Layer 22

## HTTP overview

### HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - ❖ *client*: browser that requests, receives, "displays" Web objects
  - ❖ *server*: Web server sends objects in response to requests
- Uses TCP as the Transport Protocol



CS2505: Application Layer 23

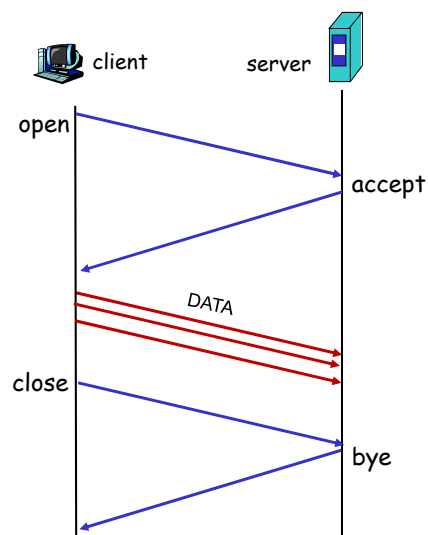
## TCP Basics

### TCP requires the sender and receiver to setup a connection

- ❖ Before they transfer any user data
- ❖ And to close it when they are finished

### This incurs delays

- ❖ Shown in the time-sequence diagram opposite as *Round-Trip Time (RTT)* delays



CS2505: Application Layer 24

## HTTP overview (continued)

### Uses TCP:

- ❑ client initiates TCP connection (creates socket) to server, port 80
- ❑ server accepts TCP connection from client
- ❑ HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- ❑ TCP connection closed

### HTTP is "stateless"

- ❑ server maintains no information about past client requests

#### aside Protocols that maintain "state" are complex!

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

CS2505: Application Layer 25

## HTTP connections

### Nonpersistent HTTP

- ❑ At most one object is sent over a TCP connection.

### Persistent HTTP

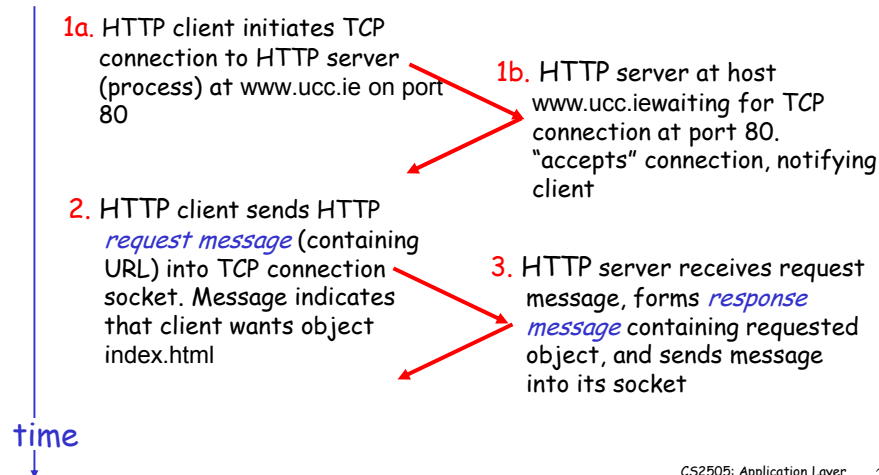
- ❑ Multiple objects can be sent over single TCP connection between client and server.

CS2505: Application Layer 26

## Nonpersistent HTTP

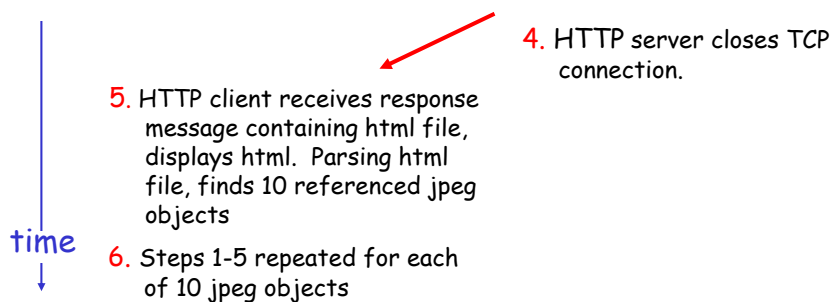
Suppose user enters URL `www.ucc.ie/index.html`

(contains text,  
references to 10  
jpeg images)



CS2505: Application Layer 27

## Nonpersistent HTTP (cont.)



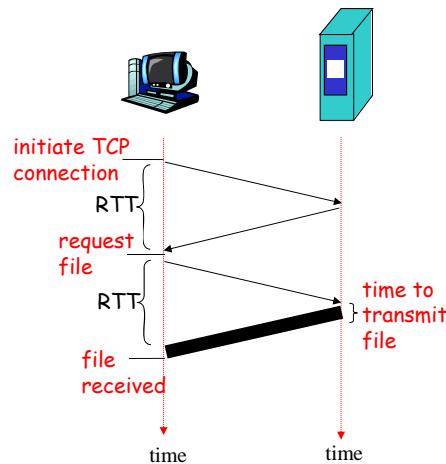
CS2505: Application Layer 28

## Non-Persistent HTTP: Response time

**Definition of RTT:** time for a small packet to travel from client to server and back.

### 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
- total =  $2RTT + \text{transmit time}$**



CS2505: Application Layer 29

## Persistent HTTP

### Nonpersistent 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 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

CS2505: Application Layer 30

## HTTP request message

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

request line  
(GET, POST,  
HEAD commands)

header  
lines

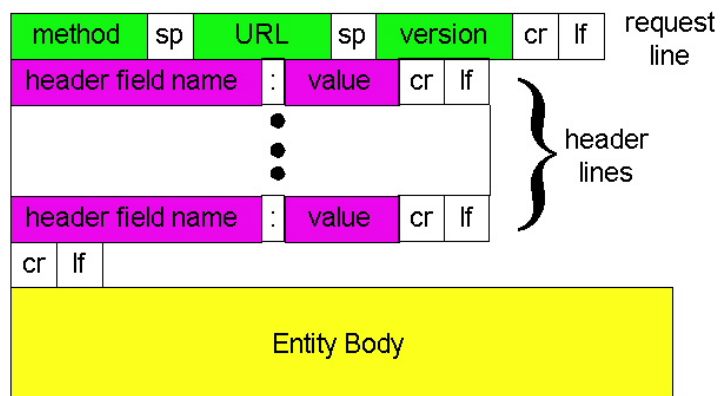
Carriage return,  
line feed  
indicates end  
of message

```
GET /index.html HTTP/1.1
Host: www.ucc.ie
User-agent: Mozilla/4.0
Connection: close
Accept-language: en
```

(extra carriage return, line feed)

CS2505: Application Layer 31

## HTTP request message: general format



CS2505: Application Layer 32



## Uploading form input

### Post method:

- ❑ Web page often includes form input
- ❑ Input is uploaded to server in entity body

### URL method:

- ❑ Uses GET method
- ❑ Input is uploaded in URL field of request line:

`www.somesite.com/animalsearch?monkeys&banana`

CS2505: Application Layer 33

## Method types

### HTTP/1.0

- ❑ GET
- ❑ POST
- ❑ HEAD
  - ❖ asks server to leave requested object out of response

### HTTP/1.1

- ❑ GET, POST, HEAD
- ❑ PUT
  - ❖ uploads file in entity body to path specified in URL field
- ❑ DELETE
  - ❖ deletes file specified in the URL field

CS2505: Application Layer 34

## HTTP response message

status line  
(protocol  
status code  
status phrase)

header  
lines

data, e.g.,  
requested  
HTML file

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 .....
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

CS2505: Application Layer 35

## HTTP response status codes

In first line in server->client response message.

A few sample codes:

### **200 OK**

- ❖ request succeeded, requested object later in this message

### **301 Moved Permanently**

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

### **400 Bad Request**

- ❖ request message not understood by server

### **404 Not Found**

- ❖ requested document not found on this server

### **505 HTTP Version Not Supported**

CS2505: Application Layer 36

## Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.cs.ucc.ie 80
```

Opens TCP connection to port 80 (default HTTP server port) at www.cs.ucc.ie  
Anything typed in sent to port 80 at www.cs.ucc.ie

2. Type in a GET HTTP request:

```
GET /~cjs/ HTTP/1.1  
Host: www.cs.ucc.ie
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!

CS2505: Application Layer 37

## User-server state: cookies

Many major Web sites use cookies

### Four components:

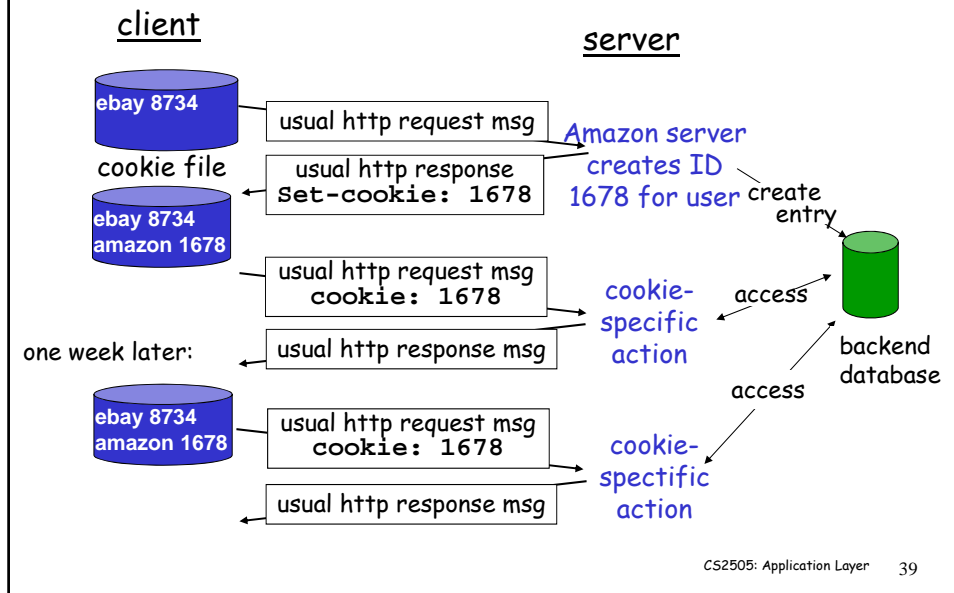
- 1) cookie header line of HTTP *response* message
- 2) cookie header line in HTTP *request* message
- 3) cookie file kept on user's host, managed by user's browser
- 4) back-end database at Web site

### Example:

- Susan always access Internet always 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

CS2505: Application Layer 38

## Cookies: keeping "state" (cont.)



## Cookies (continued)

### What cookies can bring:

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

### How to keep "state":

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

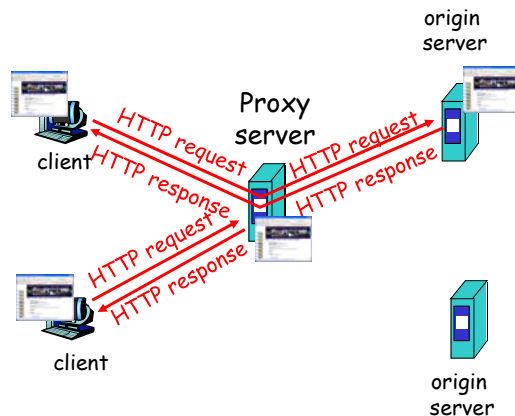
### Cookies and privacy:

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

## Web caches (proxy server)

**Goal:** satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - ❖ object in cache: cache returns object
  - ❖ else cache requests object from origin server, then returns object to client



CS2505: Application Layer 41

## More about Web caching

- cache acts as both client and 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 (but so does P2P file sharing)

CS2505: Application Layer 42

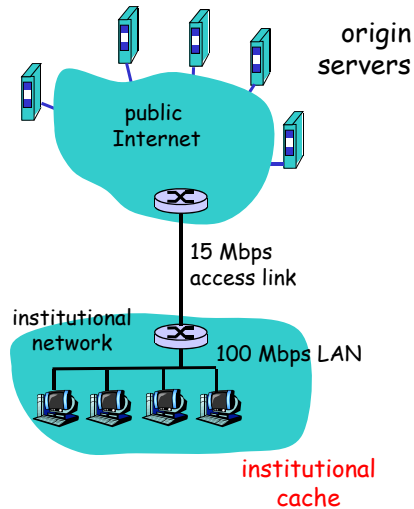
## Caching example

### Assumptions

- average object size = 1,000,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

### Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay  
= 2 sec + minutes + milliseconds



CS2505: Application Layer 43

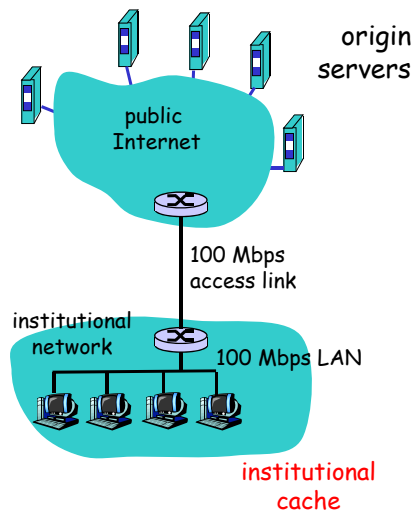
## Caching example (cont)

### possible solution

- increase bandwidth of access link to, say, 100 Mbps

### consequence

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay  
= 2 sec + msec + msec
- often a costly upgrade



CS2505: Application Layer 44

## Caching example (cont)

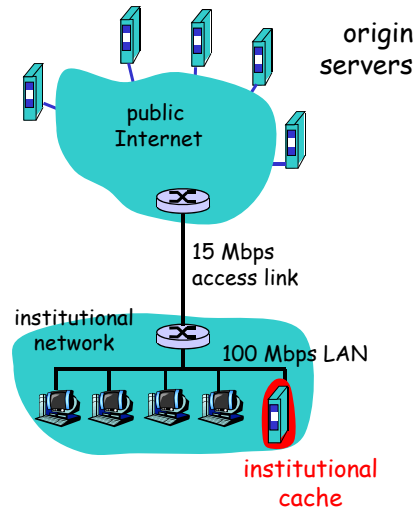
### possible solution: install cache

- suppose hit rate is 0.4

### consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay  

$$= .6 * (2.01 \text{ secs}) + .4 * \text{milliseconds} < 1.4 \text{ secs}$$



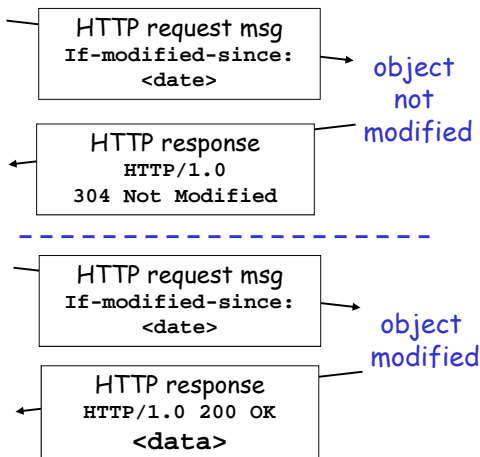
CS2505: Application Layer 45

## Conditional GET

- **Goal:** don't send object if cache has up-to-date cached version
- **cache:** specify date of cached copy in HTTP request  
`If-modified-since: <date>`
- **server:** response contains no object if cached copy is up-to-date:  
`HTTP/1.0 304 Not Modified`

### cache

### server



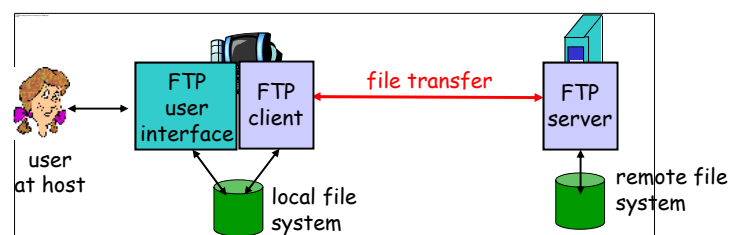
CS2505: Application Layer 46

## Section 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 **FTP**
- 2.4 Electronic Mail
- 2.5 DNS
- 2.6 File Distribution

CS2505: Application Layer 47

## FTP: the file transfer protocol




- transfer file to/from remote host
- client/server model
  - ❖ *client*: side that initiates transfer (either to/from remote)
  - ❖ *server*: remote host
- ftp: RFC 959
- ftp server: port 21

CS2505: Application Layer 48




## FTP: separate control, data connections

- ❑ FTP client contacts FTP server at port 21, TCP is transport protocol
  - ❑ client authorized over control connection
  - ❑ client browses remote directory by sending commands over control connection.
  - ❑ when server receives file transfer command, server opens 2<sup>nd</sup> TCP connection (for file) to client
  - ❑ after transferring one file, server closes data connection.
- 

FTP client

TCP control connection  
port 21

TCP data connection  
port 20



FTP server
- server opens another TCP data connection to transfer another file.
  - control connection: "out of band"
  - FTP server maintains "state": current directory, earlier authentication

CS2505: Application Layer 49

## FTP commands, responses

### 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

CS2505: Application Layer 50

## Section 2: Application layer

- ❑ 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 File Distribution

CS2505: Application Layer 51

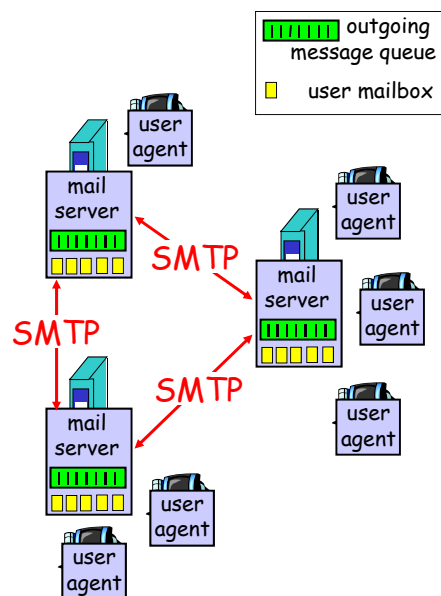
## Electronic Mail

### Three major components:

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

### User Agent

- ❑ a.k.a. "mail reader"
- ❑ composing, editing, reading mail messages
- ❑ e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- ❑ outgoing, incoming messages stored on server

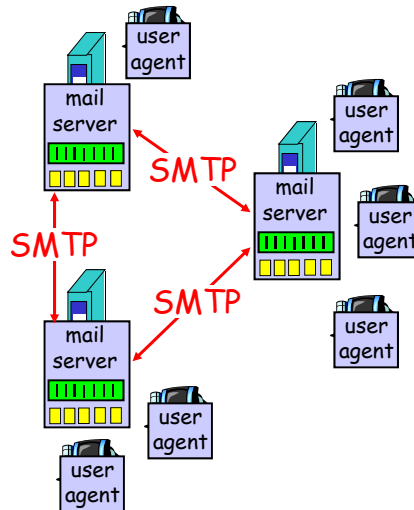


CS2505: Application Layer 52

## Electronic Mail: mail servers

### Mail Servers

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



CS2505: Application Layer 53

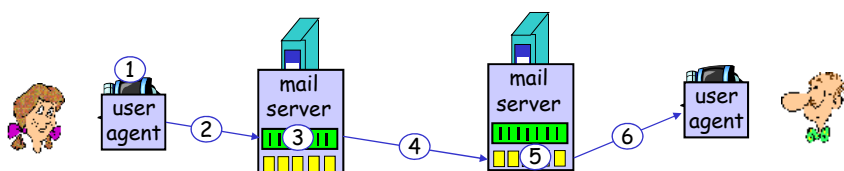
## Electronic Mail: SMTP [RFC 2821]

- ❑ uses TCP to reliably transfer email message from client to server, port 25
- ❑ direct transfer: sending server to receiving server
- ❑ three phases of transfer
  - ❖ handshaking (greeting)
  - ❖ transfer of messages
  - ❖ closure
- ❑ command/response interaction
  - ❖ **commands**: ASCII text
  - ❖ **response**: status code and phrase
- ❑ messages must be in 7-bit ASCII by default (MIME extension to handle other data types)

CS2505: Application Layer 54

## Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message and "to" bob@someuni.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



CS2505: Application Layer 55

## Sample SMTP interaction

```
S: 220 someuni.edu
C: HELO yahoo.com
S: 250 Hello yahoo.com, pleased to meet you
C: MAIL FROM: <alice@yahoo.com>
S: 250 alice@yahoo.com... Sender ok
C: RCPT TO: <bob@someuni.edu>
S: 250 bob@someuni.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: How is the weather there?
C: It's like a hurricane here today.
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 someuni.edu closing connection
```

CS2505: Application Layer 56

## Try SMTP interaction for yourself:

- ❑ `telnet servername 25`
- ❑ see 220 reply from server
- ❑ enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

## SMTP: final words

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

### Comparison with HTTP:

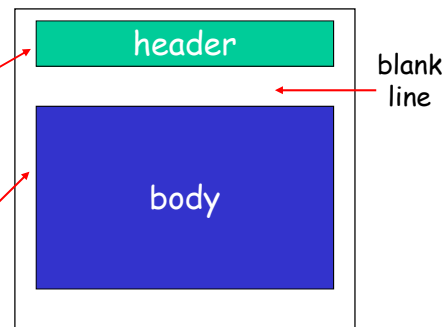
- ❑ HTTP: pull
- ❑ SMTP: push
- ❑ both have ASCII command/response interaction, status codes
- ❑ HTTP: each object encapsulated in its own response msg
- ❑ SMTP: multiple objects sent in multipart msg

## Mail message format

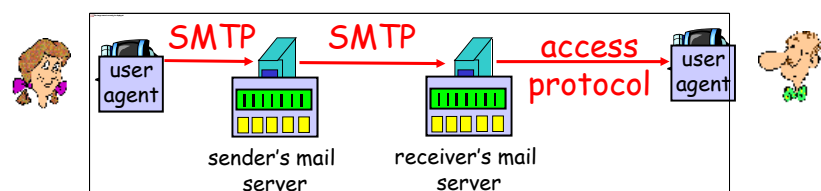
SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

- header lines, e.g.,
  - ❖ To:
  - ❖ From:
  - ❖ Subject:*different from SMTP commands!*
- body
  - ❖ the "message"



## Mail access protocols



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

## POP3 protocol

### authorization phase

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

### transaction phase, client:

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

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

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
```

CS2505: Application Layer 61

## POP3 (more) and IMAP

### More about POP3

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

### IMAP

- ❑ Keep all messages in one place: the server
- ❑ Allows user to organize messages in folders
- ❑ IMAP keeps user state across sessions:
  - ❖ names of folders and mappings between message IDs and folder name

CS2505: Application Layer 62

## Section 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
- **2.5 DNS**
- 2.6 File Distribution

CS2505: Application Layer 63

## DNS: Domain Name System

**People:** many identifiers:

- ❖ name, passport #

**Internet hosts, routers:**

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

**Q:** map between host name and its IP address?

**Domain Name System:**

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

CS2505: Application Layer 64



## DNS

### DNS services

- ❑ hostname to IP address translation
- ❑ host aliasing
  - ❖ Canonical, alias names
- ❑ mail server aliasing
- ❑ load distribution
  - ❖ replicated Web servers: set of IP addresses for one canonical name

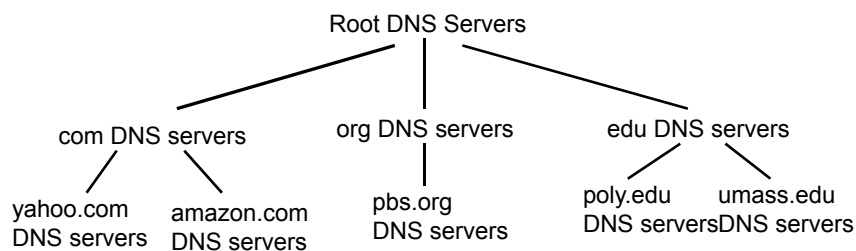
### Why not centralize DNS?

- ❑ single point of failure
- ❑ traffic volume
- ❑ distant centralized database
- ❑ maintenance

doesn't *scale*!

CS2505: Application Layer 65

## Distributed, Hierarchical Database



### Client wants IP for www.amazon.com; 1<sup>st</sup> approx:

- ❑ client queries a 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

CS2505: Application Layer 66

## DNS: Root name servers

- ❑ 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



## TLD and Authoritative Servers

- ❑ **Top-level domain (TLD) servers:**
  - ❖ responsible for com, org, net, edu, etc, and all top-level country domains ie, uk, fr, ca, jp.
  - ❖ Network Solutions maintains servers for com TLD
  - ❖ Educause for edu TLD
- ❑ **Authoritative DNS servers:**
  - ❖ organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
  - ❖ can be maintained by organization or service provider

## Local Name Server

- ❑ does not strictly belong to hierarchy
- ❑ each ISP (residential ISP, company, university) has one.
  - ❖ also called "default name server"
- ❑ when host makes DNS query, query is sent to its local DNS server
  - ❖ acts as proxy, forwards query into hierarchy

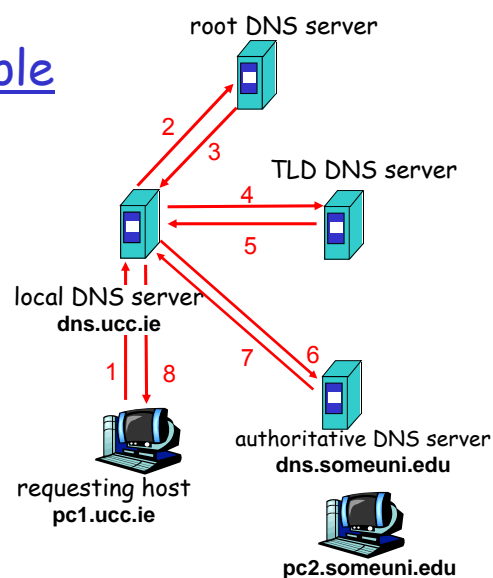
CS2505: Application Layer 69

## DNS name resolution example

- ❑ Host at pc1.ucc.ie wants IP address for pc2.someuni.edu

### iterated query:

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

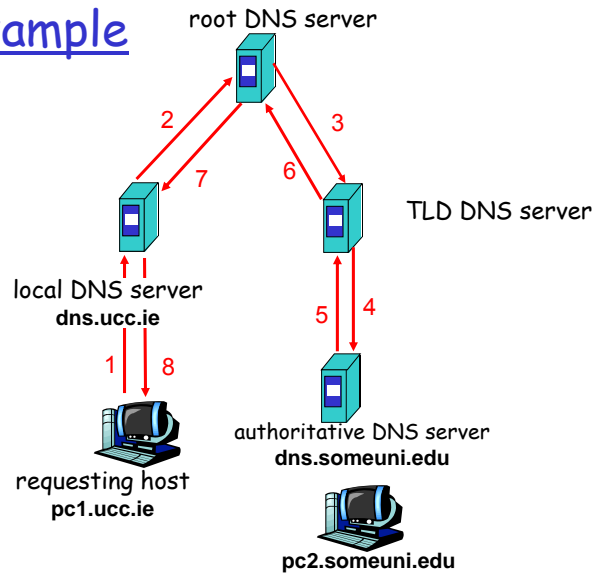


CS2505: Application Layer 70

## DNS name resolution example

### recursive query:

- puts burden of name resolution on contacted name server
- heavy load?



CS2505: Application Layer 71

## DNS: caching and updating records

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

CS2505: Application Layer 72

## DNS records

DNS: distributed db storing resource records (RR)

RR format: (name, value, type, ttl)

- Type=A
  - ❖ name is hostname
  - ❖ value is IP address
- Type=CNAME
  - ❖ name is alias name for some "canonical" (the real) name  
www.ibm.com is really servereast.backup2.ibm.com
  - ❖ value is canonical name
- Type=NS
  - ❖ name is domain (e.g. foo.com)
  - ❖ value is hostname of authoritative name server for this domain
- Type=MX
  - ❖ value is name of mailserver associated with name

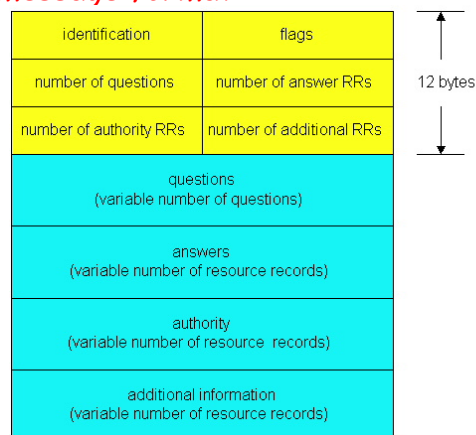
CS2505: Application Layer 73

## DNS protocol, messages

DNS protocol: *query* and *reply* messages, both with same binary-encoded *message format*

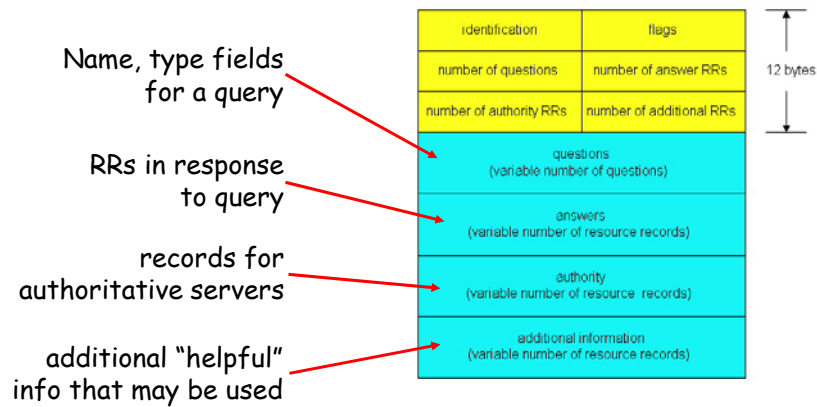
msg header

- **identification**: 16 bit # for query, reply to query uses same #
- **flags**:
  - ❖ query or reply
  - ❖ recursion desired
  - ❖ recursion available
  - ❖ reply is authoritative



CS2505: Application Layer 74

## DNS protocol, messages



CS2505: Application Layer 75

## Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkutopia.com at *DNS registrar* (e.g., Network Solutions)
  - ❖ provide names, IP addresses of authoritative name server (primary and secondary)
  - ❖ registrar inserts two RRs into com TLD server:
 

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```
- create authoritative server Type A record for www.networkutopia.com; Type MX record for networkutopia.com
- How do people get IP address of your Web site?

CS2505: Application Layer 76

## Section 2: Application layer

- 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 File Distribution
  - ❖ Peer-to-Peer (P2P), Content Distribution Networks (CDN), Streaming Video

CS2505: Application Layer 77

## Problem of File Distribution

- A common requirement is to use the Internet to distribute files to end-users
  - ❖ music, video, movies, binaries, software patches, web objects, etc
- We need mechanisms that can do this in a scalable manner
  - ❖ Scalable in the number of requesting users
  - ❖ Scalable in the number of requested files
- Issues are
  - ❖ Response time to end-user for delivery
  - ❖ Network congestion
  - ❖ Server load

CS2505: Application Layer 78

## File Distribution: Mirror Sites

- Client-Server model but using *mirror* sites
  - ❖ The file is replicated across a set of servers (each known as a mirror)
  - ❖ User selects a mirror, typically based on geographical proximity and/or an indication of how busy the mirror is
- Issues
  - ❖ Need to ensure consistency of file replicas on each mirror
  - ❖ Users may converge on just a few mirrors, eg especially in highly populated areas

CS2505: Application Layer

79

## File Distribution: CDNs

- CDN = Content Distribution Network
  - ❖ An *automated* approach to the use of mirrors
  - ❖ Use of CDN is *transparent* to end-users
- CDN composed of large number of servers
  - ❖ Organised as an *overlay* network
  - ❖ Distributed globally, strategically placed, often at edge of backbone network
  - ❖ Each server keeps replicas of popular files
  - ❖ CDN management system responsible for deciding how many replicas and in which locations (based on expected/actual usage)

CS2505: Application Layer

80



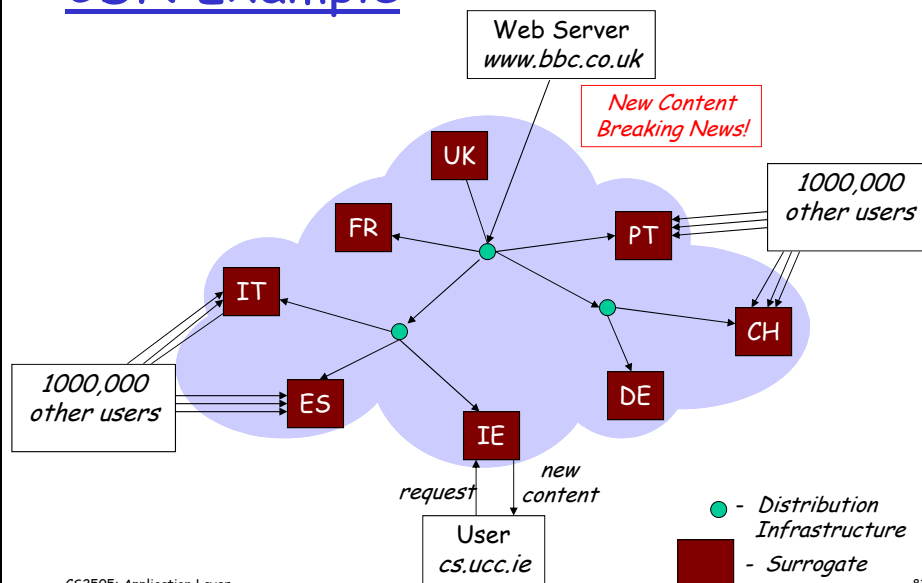
## File Distribution: CDNs

- Overlay network to distribute content from origin servers to users
- Avoids large amounts of same data repeatedly traversing potentially congested links on the Internet
- Reduces Web server load
- Reduces user perceived latency
- Tries to route around congested networks

CS2505: Application Layer

81

## CDN Example



## File Distribution: CDNs

### □ Issues

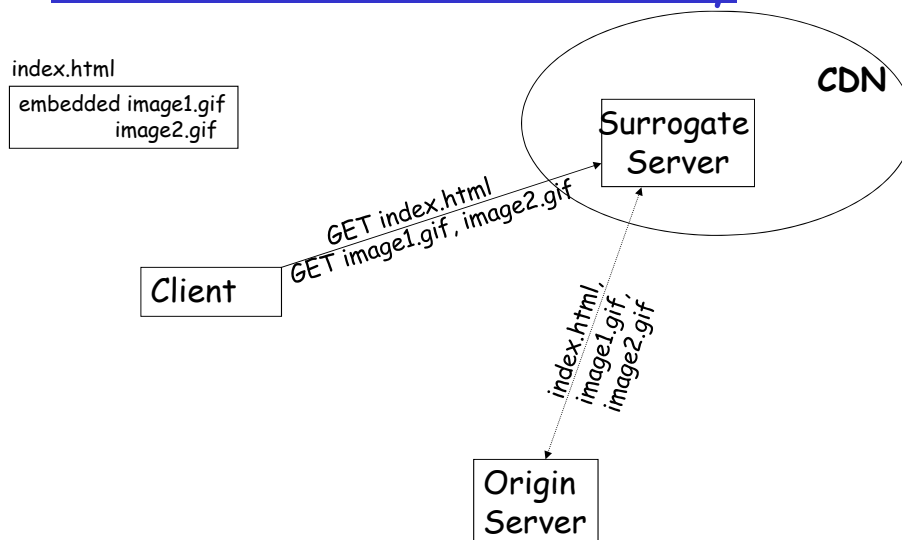
- ❖ High costs for having many servers
- ❖ How to direct clients to best server?
  - How to define best? Answer: \_\_\_\_\_
- ❖ Redirection mechanism
  - Often using DNS, eg send request to *www.xyz.com* but response is to *www.xyz.ie*
  - Or HTML rewriting, where the embedded URLs are changed dynamically to include the target server

### □ Example is Akamai, with tens of thousands of servers worldwide

CS2505: Application Layer

83

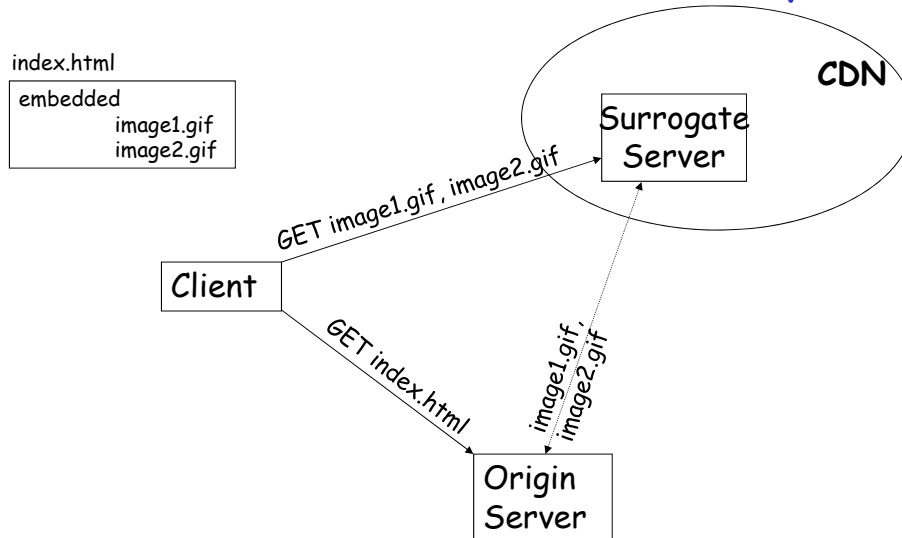
## CDN with Full Site Delivery



CS2505: Application Layer

84

## CDN with Partial Site Delivery



CS2505: Application Layer

85

## CDN -versus- Caches

- ❑ CDN is proactive while a cache is reactive
- ❑ Different purposes
  - ❖ Caches to reduce network traffic
  - ❖ CDN to improve delivery of content
- ❑ Different "customer"
  - ❖ CDN caters to content providers (web servers)
  - ❖ Cache caters to web users

CS2505: Application Layer

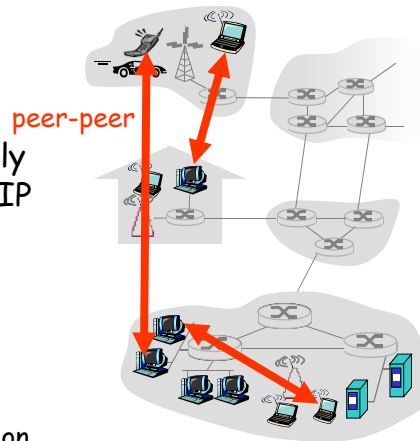
86

## Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

- Three topics:

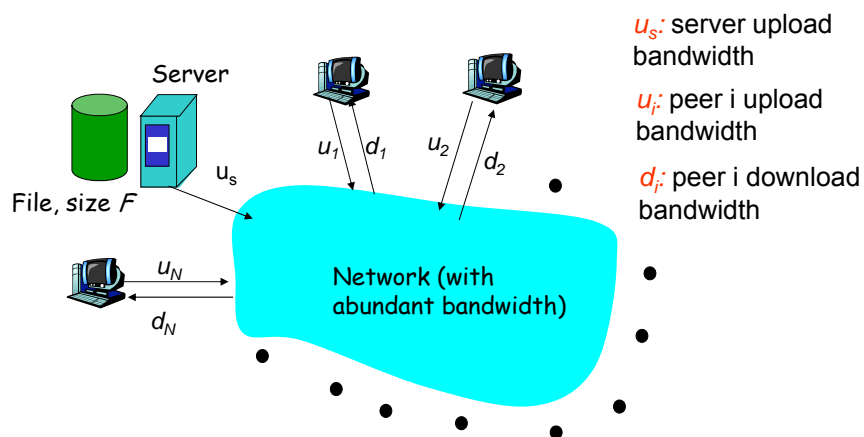
- ❖ File distribution performance
- ❖ Searching for information



CS2505: Application Layer 87

## File Distribution: Server-Client vs P2P

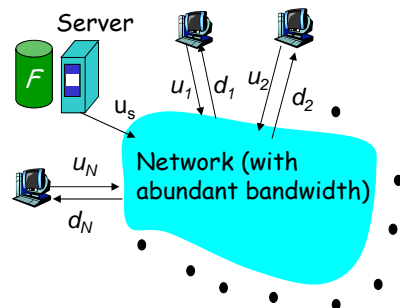
Question: How much time to distribute file from one server to  $N$  peers?



CS2505: Application Layer 88

## File distribution time: server-client

- server sequentially sends  $N$  copies:
  - ❖  $NF/u_s$  time
- client  $i$  takes  $F/d_i$  time to download



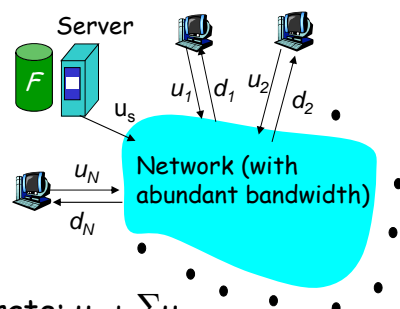
Time to distribute  $F$  to  $N$  clients using client/server approach  $= d_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$

increases linearly in  $N$  (for large  $N$ )

CS2505: Application Layer 89

## File distribution time: P2P

- server must send one copy:  $F/u_s$  time
- client  $i$  takes  $F/d_i$  time to download
- $NF$  bits must be downloaded (aggregate)
  - fastest possible upload rate:  $u_s + \sum u_i$

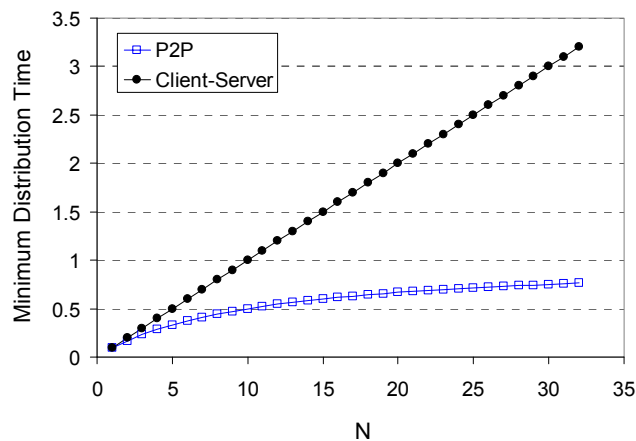


$$d_{p2p} = \max \{ F/u_s, F/\min_i(d_i), NF/(u_s + \sum u_i) \}$$

CS2505: Application Layer 90

## Server-client vs. P2P: example

Client upload rate =  $u$ ,  $F/u = 1$  hour,  $u_s = 10u$ ,  $d_{\min} \geq u_s$



CS2505: Application Layer 91

## BitTorrent

- Very popular P2P protocol written by Bram Cohen (in Python) in 2001
  - ❖ Lots of clients available for all platforms
- Nodes organise as an overlay network and volunteer to participate in distributing a file
- "Pull-based" "swarming" approach
  - ❖ Each file split into smaller **pieces**
  - ❖ Nodes request desired pieces from neighbours (peers)
  - ❖ Pieces not downloaded in sequential order
- Encourages contribution by all nodes

CS2505: Application Layer 92

## BitTorrent

- ❑ A user wishing to distribute a file X creates a small torrent descriptor file Xt and distributes it via web, email, etc
- ❑ Then make actual file X available on node known as the *seed*
- ❑ Users wanting file X pass the torrent descriptor Xt to their client which then contacts the seed and/or peers
- ❑ Over time there is less dependence on the seed, thus providing scalability

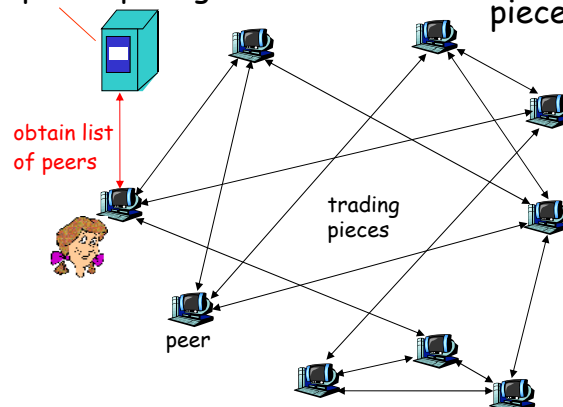
CS2505: Application Layer 93

## BitTorrent

P2P file distribution

tracker: tracks peers participating in torrent

torrent: group of peers exchanging pieces of a file



CS2505: Application Layer 94

## BitTorrent

- file divided into fixed size *pieces*, typically 256KB
- peer joining torrent:
  - ❖ has no pieces, but will accumulate them over time
  - ❖ registers with tracker to get list of peers, connects to subset of peers ("neighbours")
- while downloading, peer uploads pieces to other peers
- peers may come and go (known as *churn*)
- once peer has entire file, it may (selfishly) leave, or (altruistically) remain (as an additional seed)

CS2505: Application Layer 95

## BitTorrent

### Pulling Pieces

- at any given time, different peers have different subsets of file pieces
- periodically, a peer (Alice) asks each neighbour for list of pieces that they have
- Alice sends requests for her missing pieces
  - ❖ rarest first

### Sending Pieces: tit-for-tat

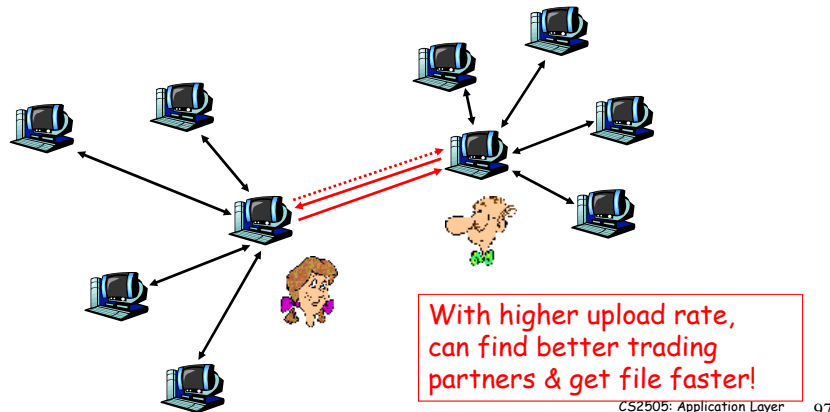
- Alice sends pieces to small subset of neighbours currently sending her pieces *at the highest rate*
  - ❖ re-evaluate periodically, eg every 10 secs
- every 30 secs: randomly select another peer, starts sending pieces
  - ❖ newly chosen peer may join top subset
  - ❖ "optimistically unchoke"
  - ❖ Why bother?

CS2505: Application Layer 96



## BitTorrent: Tit-for-tat

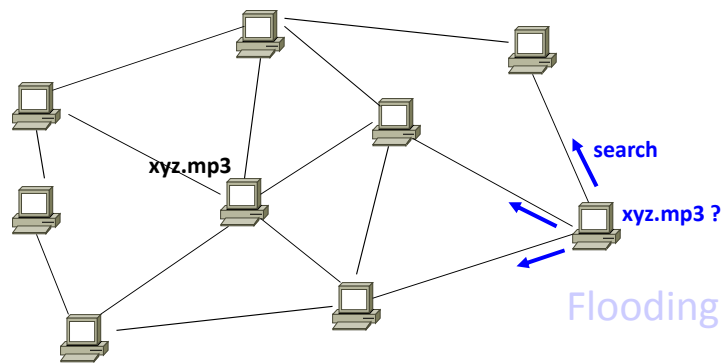
- (1) Alice "optimistically unchokes" Bob
- (2) Alice becomes one of Bob's top-four providers; Bob reciprocates
- (3) Bob becomes one of Alice's top-four providers



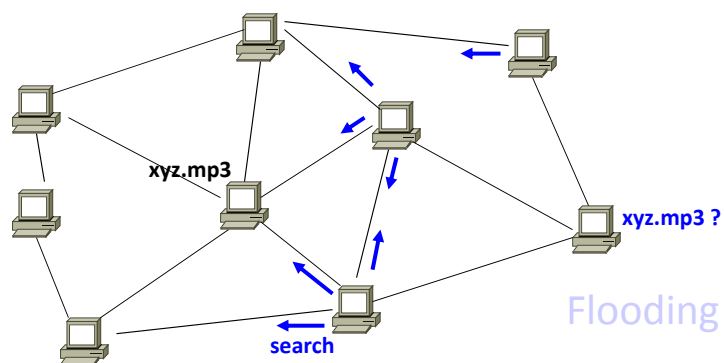
## Tracking

- ❑ Trackers were used in 1<sup>st</sup> generation P2P protocols, e.g. Napster
- ❑ Problem with use of a (centralised) tracker
  - ❖ Vulnerability to single node
  - ❖ Privacy concerns (tracker knows all peers in torrent)
- ❑ 2<sup>nd</sup> generation used flooding, e.g. Gnutella
  - ❖ Lookup request was delivered hop-by-hop to *every* node in the P2P network
  - ❖ Significant overhead

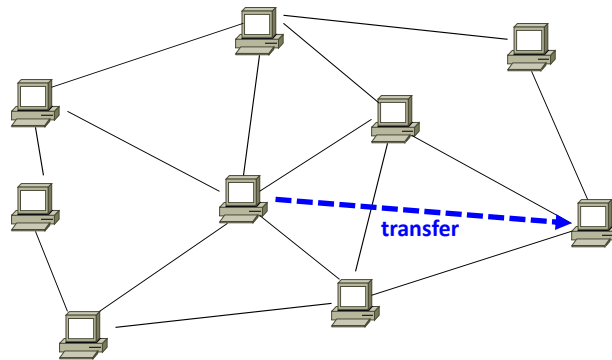
## P2P Flooding on Overlays



## P2P Flooding on Overlays

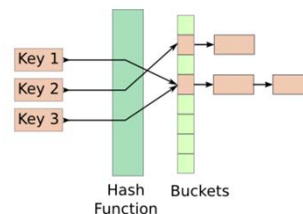


## P2P Flooding on Overlays



## Distributed Tracking

- ❑ 3<sup>rd</sup> generation uses a distributed tracker e.g. Chord, Kademlia (used in BitTorrent)
  - ❖ scalable queries when the data is spread over large numbers of peers
  - ❖ implemented as a Distributed Hash Table (DHT)
- ❑ Hash Table
  - ❖ data structure that maps “keys” to “values”
  - ❖ essential building block in software systems



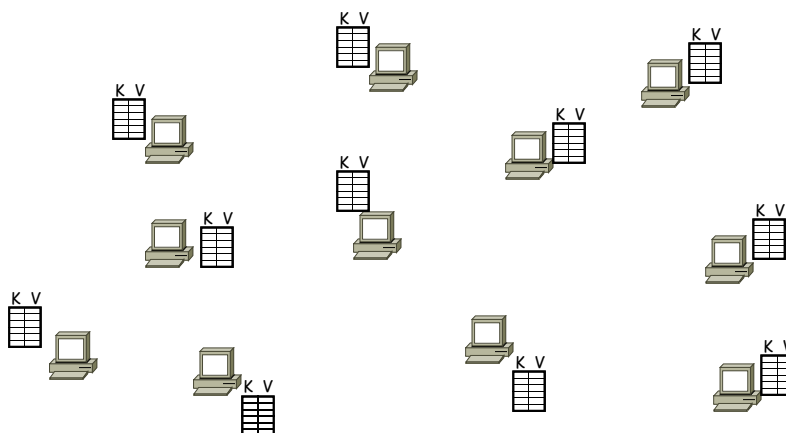
## Distributed Hash Tables

- DHTs are *content-addressable*
  - ❖ In effect a database with (**key**, **value**) pairs, e.g. (content name, IP address)
- Interface
  - ❖ insert(key, value)
  - ❖ lookup(key)
- Every DHT node plays a part in enabling the lookup function:
  - ❖ Given *key* as input; it routes messages to the node holding the *key*

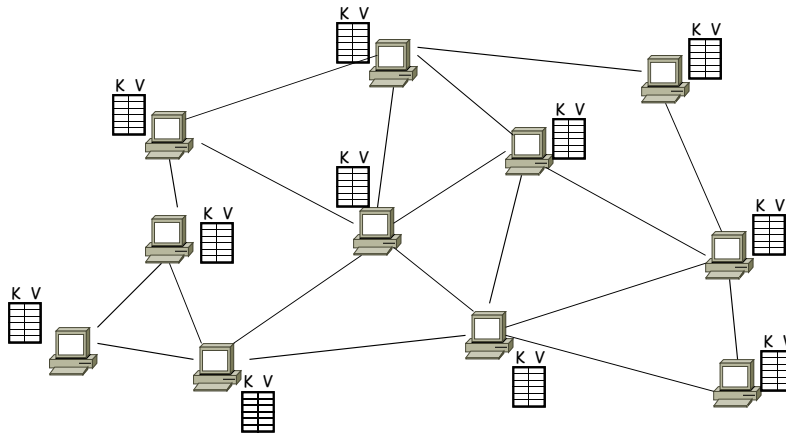
CS2505: Application Layer

103

## DHT: basic idea

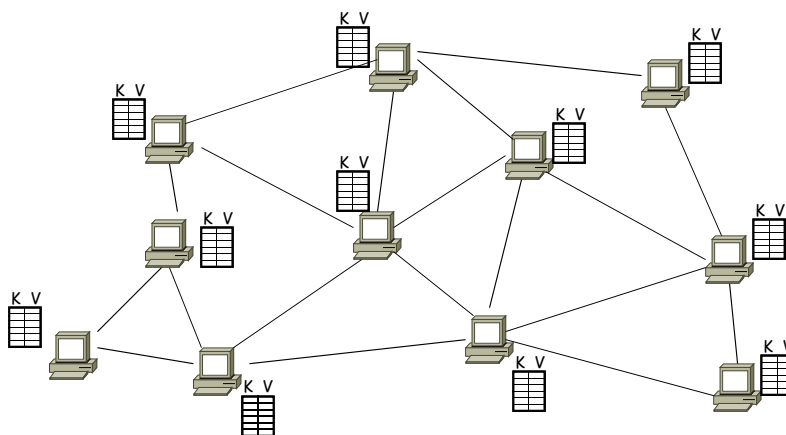


## DHT: basic idea



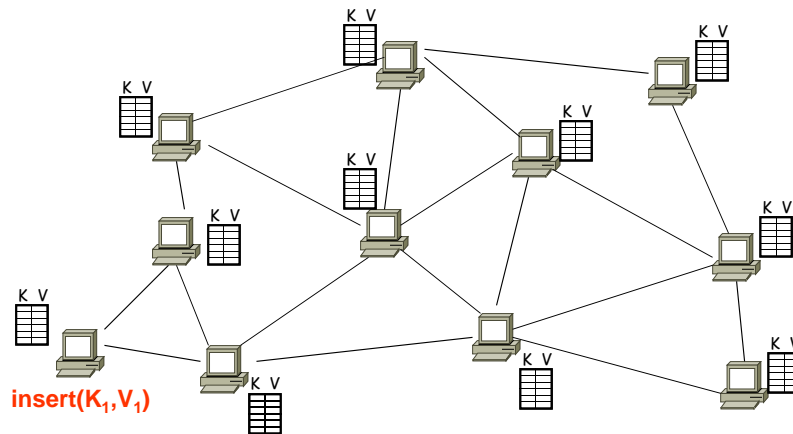
Neighboring nodes are “connected” at the application-level

## DHT: basic idea



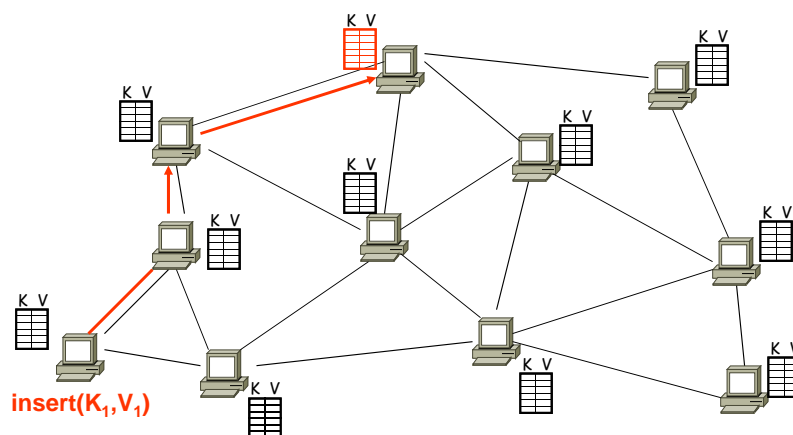
Operation: take *key* as input; route messages to node holding *key*

## DHT: basic idea



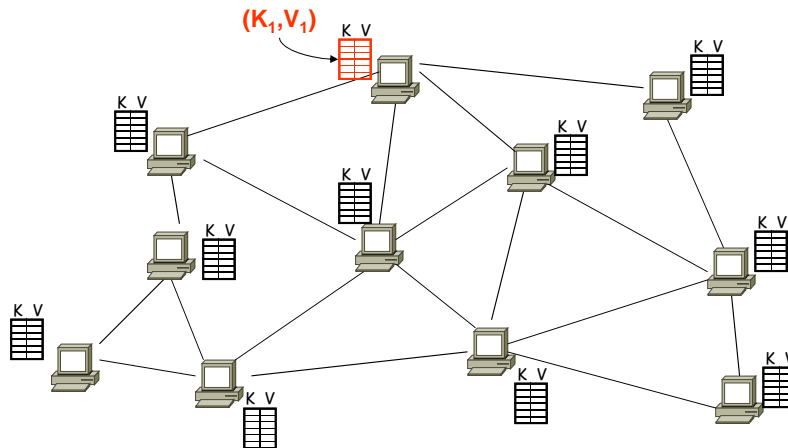
Operation: take *key* as input; route messages to node holding *key*

## DHT: basic idea



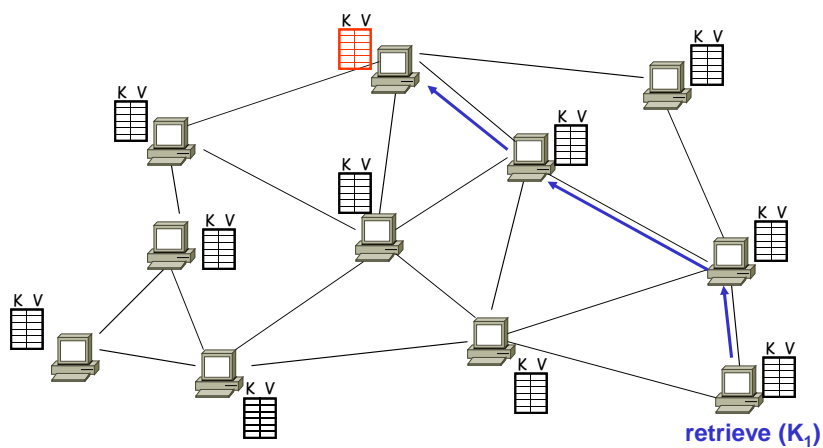
Operation: take *key* as input; route messages to node holding *key*

## DHT: basic idea



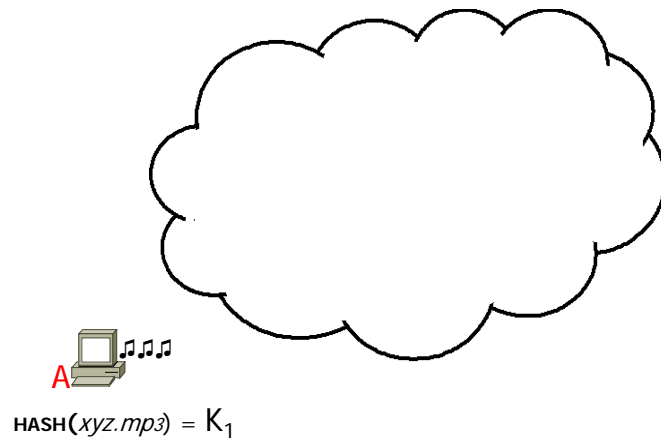
Operation: take *key* as input; route messages to node holding *key*

## DHT: basic idea

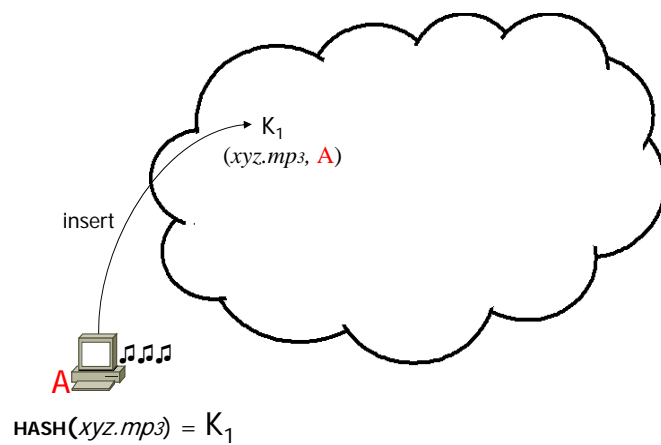


Operation: take *key* as input; route messages to node holding *key*

## Content Addressability in a DHT

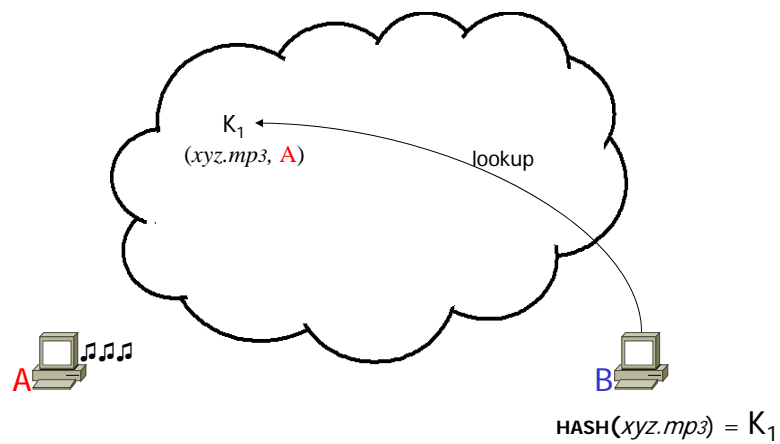


## Content Addressability in a DHT

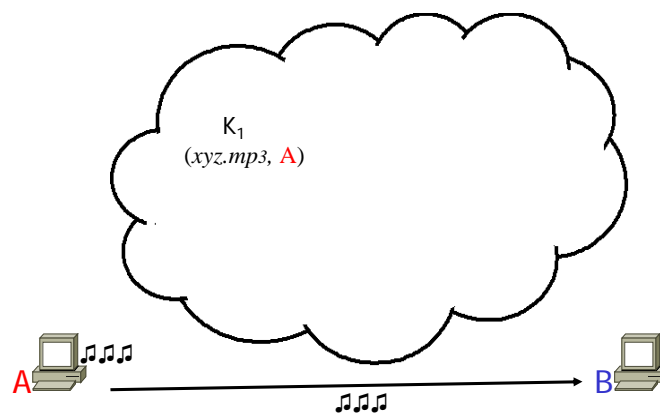




## Content Addressability in a DHT



## Content Addressability in a DHT



## DHT Operations

- ❑ Distribute (key, value) pairs over millions of peers - ideally spread evenly
- ❑ Any peer can **query** database with a key
  - ❖ database returns values matching that key
  - ❖ can be multiple matches (i.e. replicated content)
- ❑ Each peer may only know about a small number of other peers
- ❑ Robust to peers coming and going (churn)
- ❑ Peers can also **insert** (key, value) pairs, when they acquire new content

CS2505: Application Layer

115

## DHT Identifiers

- ❑ More convenient to store and search on numerical representation of key
- ❑ To get integer keys, hash original key.
  - ❖ eg, key =  $h(\text{"Led Zeppelin IV"})$
- ❑ Assign integer ID to each peer in range  $[0, 2^n - 1]$ .
  - ❖ Each ID can be represented by  $n$  bits.
- ❑ Require each key to be an integer in **same range**.
- ❑ Use hash to map a key to a peer which will store the details of *where* to find the content, i.e. list of peers

CS2505: Application Layer

116

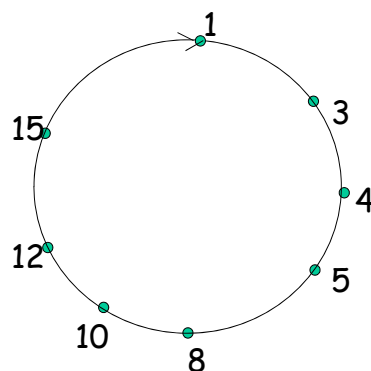
## How to assign keys to peers?

- Central issue:
  - ❖ Assigning (key, value) pairs to peers
- Rule: assign key to the peer that has the exact ID or the **closest** ID
- Convention in lecture: *closest* is the ID which is the **immediate successor** of the key
- E.g.  $n=4$ ; peers: 1,3,4,5,8,10,12,14;
  - ❖ key = 13, then successor peer = 14
  - ❖ key = 15, then successor peer = 1

CS2505: Application Layer

117

## Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network"

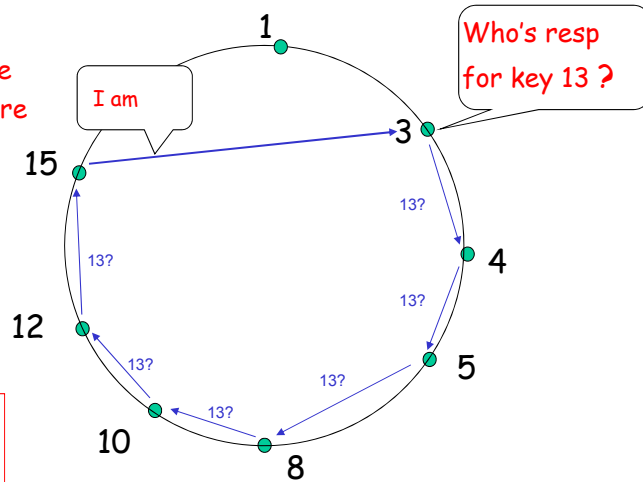
CS2505: Application Layer

118

## Circular DHT (2)

$O(N)$  messages  
on avg to resolve  
query, when there  
are  $N$  peers

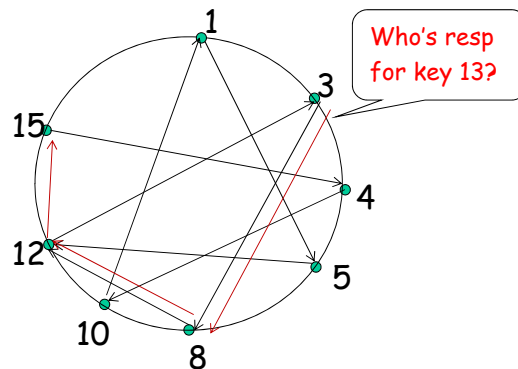
Define closest  
ID as closest  
successor



CS2505: Application Layer

119

## Circular DHT with Shortcuts

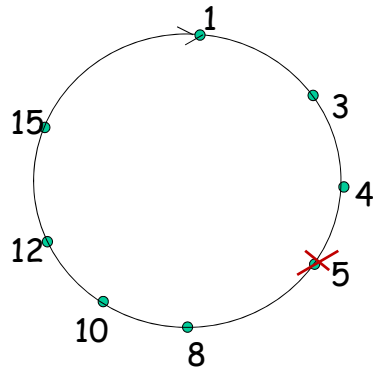


- Each peer keeps track of IP addresses of predecessor, successor, and short cuts.
- Reduced from 6 to 3 messages.
- Possible to design shortcuts so  $O(\log N)$  neighbours,  $O(\log N)$  messages in query

CS2505: Application Layer

120

## Peer Churn



- To handle peer churn, require each peer to know the IP address of its *two* successors and replicate their key/value pairs
- Each peer periodically pings its two successors to see if they are still alive.

- ❑ Peer 5 abruptly leaves
- ❑ Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- ❑ What if peer 13 wants to join?

CS2505: Application Layer

121

## DHTs - Requirements

- ❑ Decentralized: no central authority
- ❑ Scalable: low network traffic overhead
- ❑ Efficient: find items quickly (latency)
- ❑ Dynamic: nodes fail, new nodes join
- ❑ General-purpose: flexible naming

CS2505: Application Layer

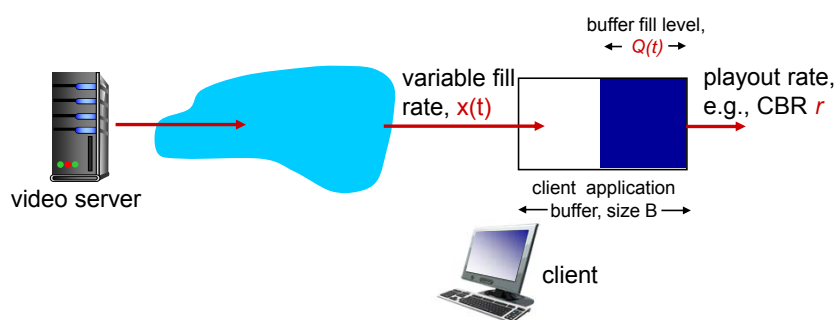
122

## Streaming stored video

- ❖ Video is the dominant traffic type on today's fixed and mobile networks
- ❖ Video files usually too large to download fully, so instead they are streamed at a rate that matches the playback rate
- ❖ *continuous playout constraint*: once client playout begins, playback must match original timing
  - ... but *network delays are variable* (jitter), so will need *client-side buffer* to match playout requirements

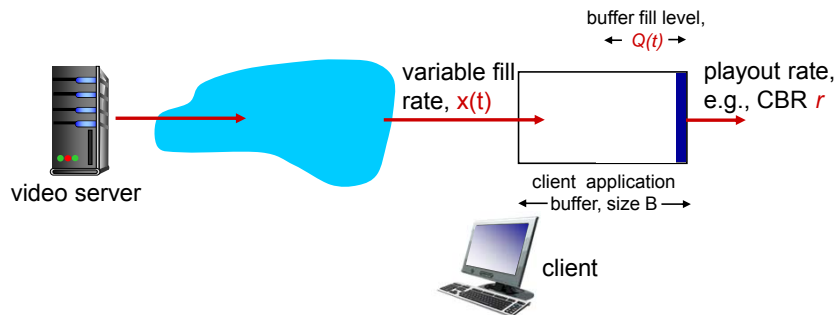
CS2505: Application Layer 123

## Client-side buffering, playout



CS2505: Application Layer 124

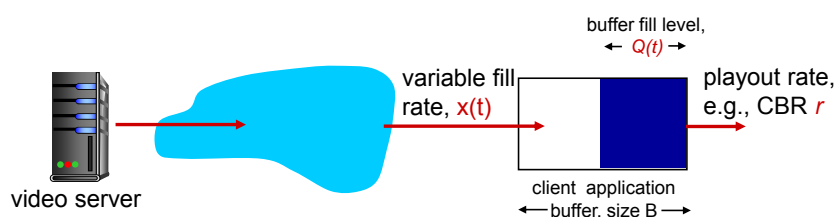
## Client-side buffering, playout



1. Initial fill of buffer until playout begins at  $t_p$
2. playout begins at  $t_p$ ,
3. buffer fill level varies over time as fill rate  $x(t)$  varies and playout rate  $r$  is constant

CS2505: Application Layer 125

## Client-side buffering, playout



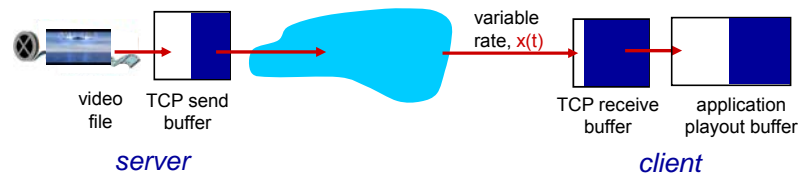
*playout buffering: average fill rate ( $\bar{x}$ ), playout rate ( $r$ ):*

- ❖  $\bar{x} < r$ : buffer eventually empties (causing freezing of video playout until buffer again fills)
- ❖  $\bar{x} > r$ : buffer will not empty, provided initial playout delay is large enough to absorb variability in  $x(t)$ 
  - *initial playout delay tradeoff*: buffer starvation less likely with larger delay, but larger delay until user begins watching

CS2505: Application Layer 126

## Streaming multimedia: HTTP

- ❖ multimedia file retrieved via HTTP GET
- ❖ send at maximum possible rate under TCP



- ❖ fill rate fluctuates due to variable network delivery delays
- ❖ Use larger playout delay: to smooth delivery rate
- ❖ HTTP/TCP passes more easily through firewalls

CS2505: Application Layer 127

## Streaming multimedia: DASH

- ❖ **DASH**: *D*ynamic, *A*daptive *S*teaming over *H*TT**P**
- ❖ **server**:
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file*: provides URLs for different chunks
- ❖ **client**:
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)

CS2505: Application Layer 128



## Streaming multimedia: DASH

- ❖ *DASH: Dynamic, Adaptive Streaming over HTTP*
- ❖ “intelligence” at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is “close” to client or has high available bandwidth)

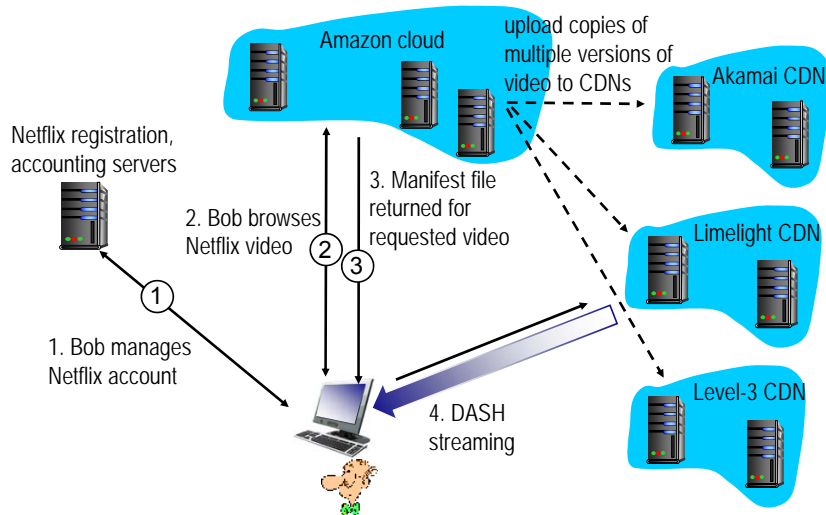
CS2505: Application Layer 129

## Case study: Netflix

- ❖ 35% downstream peak-hours US traffic (2014)
- ❖ owns very little infrastructure, uses 3<sup>rd</sup> party services:
  - own registration, payment servers
  - Amazon (3<sup>rd</sup> party) cloud services:
    - Netflix uploads studio master to Amazon cloud
    - create multiple version of movie (different encodings) in cloud
    - upload versions from cloud to CDNs
    - Cloud hosts Netflix web pages for user browsing
  - *three* 3<sup>rd</sup> party CDNs host/stream Netflix content: Akamai, Limelight, Level-3

CS2505: Application Layer 130

## Case study: Netflix



CS2505: Application Layer 131

## Section 2: Summary

our study of network apps now complete!

- ❑ application architectures
  - ❖ client-server
  - ❖ P2P
  - ❖ hybrid
- ❑ 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, Skype, DASH

CS2505: Application Layer 132

## Section 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 msgs
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- "complexity at network edge"

CS2505: Application Layer 133