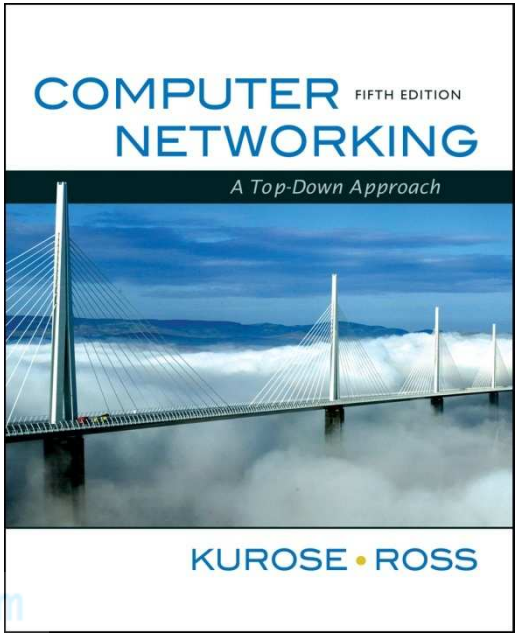
Chapter 2
Application Layer



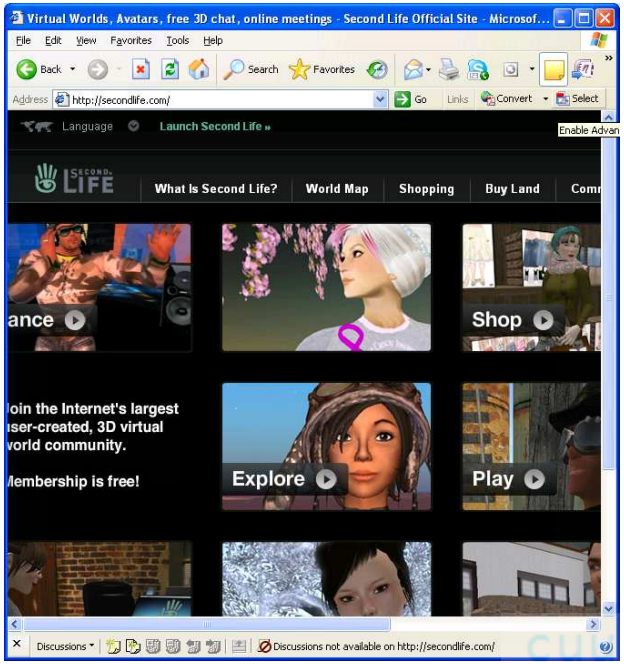
A note on the use of these ppt slides:

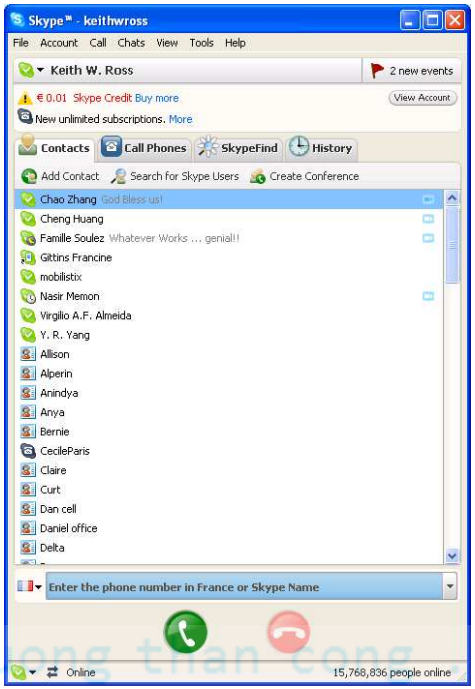
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

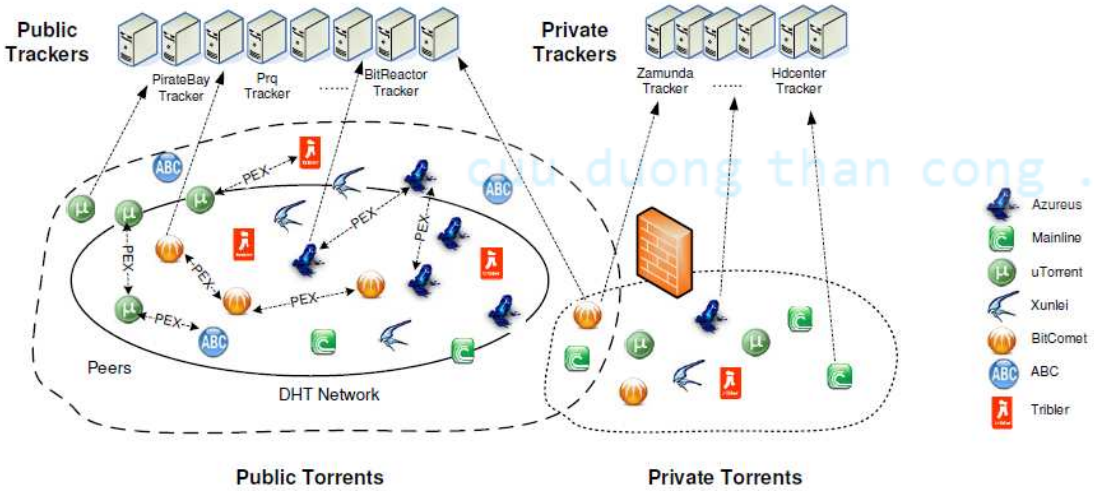
- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

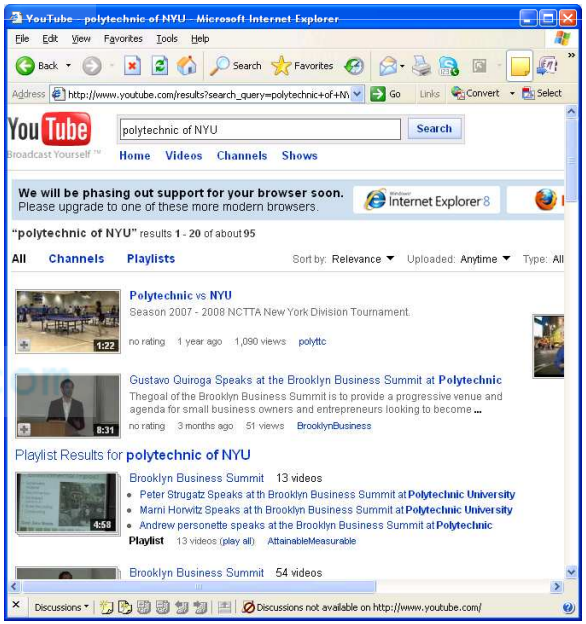
Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach, 5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.









Chapter 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 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

Chapter 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
- programming network applications
  - socket API

Some network apps

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video clips
- social networks
- voice over IP
- real-time video conferencing
- grid computing

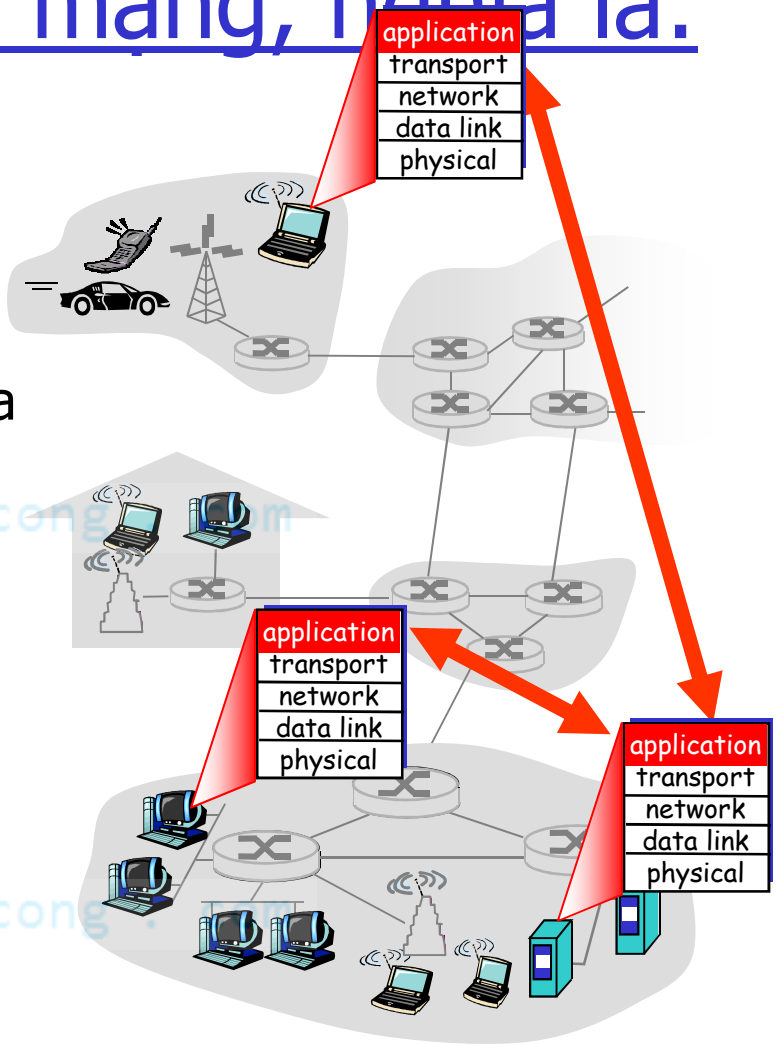
Tạo ra một ứng dụng mạng, nghĩa là:

viết chương trình sao cho:

- Có thể chạy trên các host khác nhau.
- Có thể truyền thông với nhau qua mạng
- e.g., web server software communicates with browser software

không cần phải viết các phần mềm chạy trên thiết bị mạng (router, switch, ...)

- Các thiết bị mạng ko chạy các apps do người dùng viết
- Lối này cho phép phát triển nhanh các apps chạy trên các host.



Chapter 2: Application layer

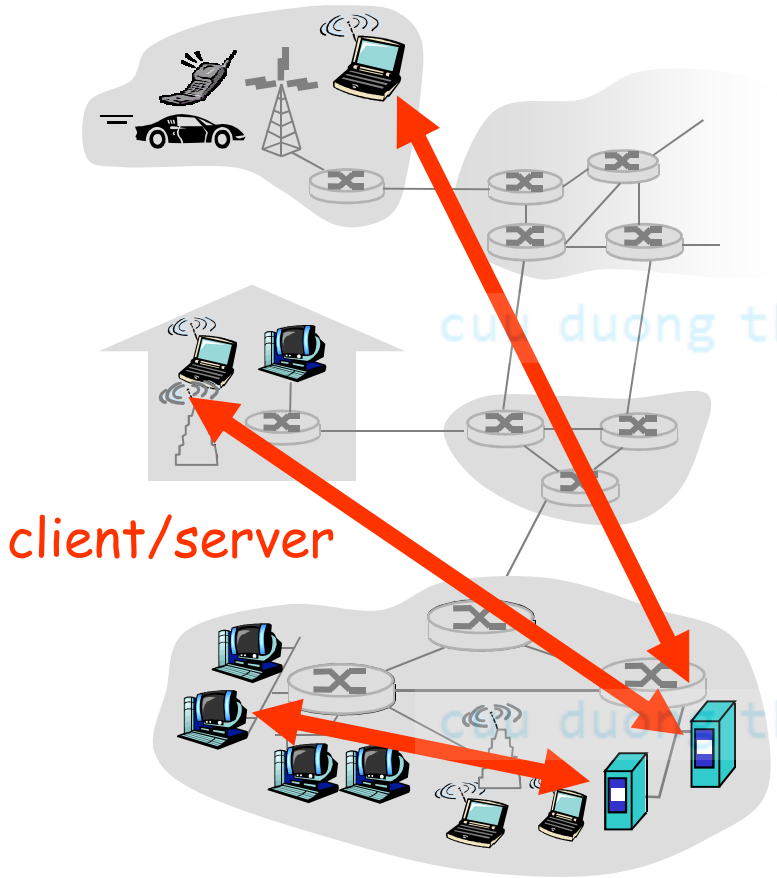
- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

Application architectures

- Kiến trúc ứng dụng là yếu tố phải quyết định trước tiên khi xây dựng ứng dụng mạng
  - Client-server
    - Including data centers / cloud computing
  - Peer-to-peer (P2P)
  - Hybrid of client-server and P2P

- Kiến trúc ứng dụng ≠ kiến trúc mạng (VD Internet có kiến trúc phân tầng!)

Client-server architecture



server:

- always-on host
- địa chỉ IP cố định
- Sử dụng server farms khi tải lớn

clients:

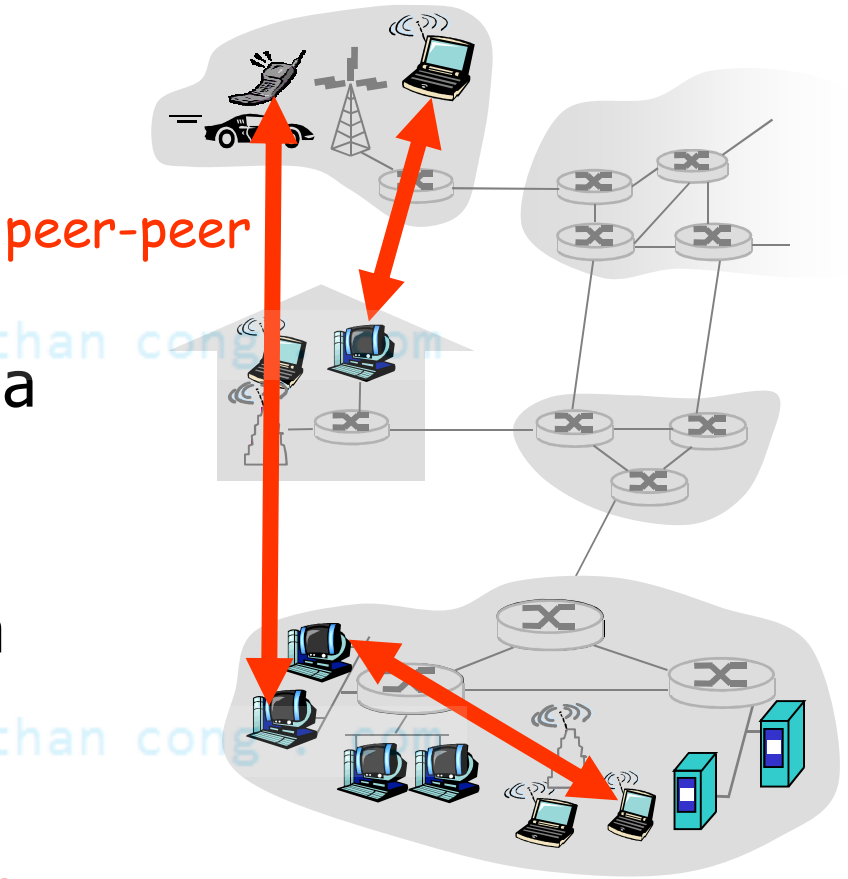
- Truyền thông với server
- Có thể kết nối không liên tục
- Có thể sử dụng địa chỉ IP động
- Không truyền thông trực tiếp với nhau.

Google Data Centers

Pure P2P architecture

- Ko dùng server
- Các host truyền thông trực tiếp với nhau
- Các peer có thể kết nối ko liên tục và thay đổi địa chỉ IP

VD: Gnutella – ứng dụng nguồn mở cho phép chia sẻ file



Có tính mở rộng cao nhưng khó quản lý.

Hybrid of client-server and P2P

Skype

- Là ứng dụng truyền âm trên mạng IP (VoIP)
- Server trung tâm:tìm địa chỉ của bên thứ 3
- Kết nối client-client là trực tiếp, ko qua Server

Instant messaging

- Dùng để tán gẫu giữa 2 user trên mạng
- Dịch vụ centralized service: phát hiện/định vị sự hiện diện/vị trí của client
  - User đăng ký địa chỉ IP của nó với server trung tâm mỗi khi online
  - User liên lạc với server trung tâm để tìm địa chỉ IP của bạn chat.

Processes communicating

Tiến trình (process): chương trình chạy trên 1 host.

- Trên cùng host, 2 tiến trình truyền thông với nhau sử dụng inter-process communication (được hệ điều hành định nghĩa).
- Các tiến trình trên các host khác nhau truyền thông nhau bằng trao đổi thông điệp (messages)

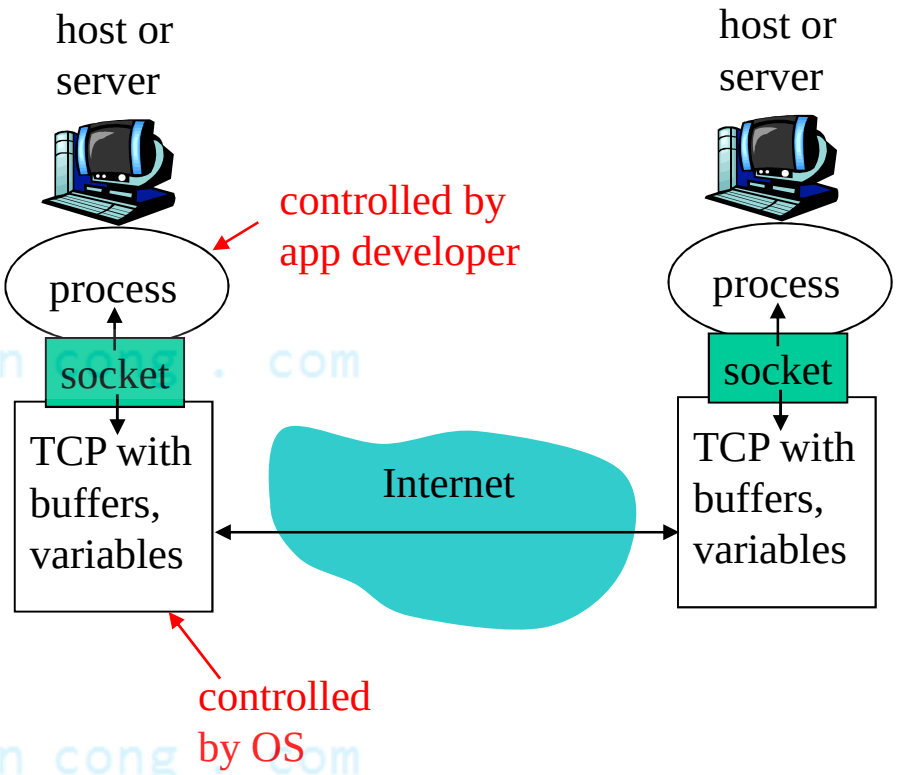
Tiến trình client : tiến trình khởi tạo truyền thông

Tiến trình server: tiến trình chờ đợi được liên lạc

- Lưu ý: các ứng dụng P2P có cả tiến trình client lẫn tiến trình server.

Sockets

- Các tiến trình gửi/nhận các thông điệp đến/từ socket của nó
- Tiến trình~ "ngôi nhà" => socket ~ "cửa ra vào"
  - Tiến trình đẩy thông điệp ra "cửa".
  - Hạ tầng vận chuyển ở "bên ngoài cửa" chuyển giao thông điệp tới socket của tiến trình bên nhận.



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

Addressing processes

- Để nhận thông điệp, tiến trình phải có định danh (*identifier)*
- Các host đều có 1 địa chỉ IP dài 32 bit, duy nhất.
- Bài tập: dùng lệnh `ipconfig` từ console, lấy địa chỉ IP trên máy tính Windows.
- *Q:* liệu địa chỉ IP của host mà process đang chạy trên đó, có đủ để xác định tiến trình?
  - *A:* Không đủ, vì có có *nhiều* tiến trình có thể chạy trên cùng host.
- *Định danh* bao gồm cả địa chỉ IP và số hiệu cổng (port number) được liên kết với tiến trình trên host.
- Ví dụ về các port number:
  - HTTP server: 80
  - Mail server: 25

App-layer protocol defines

- Các kiểu thông điệp được trao đổi,
  - e.g., request, response
- Ký pháp thông điệp:
  - Các field trên thông điệp và mô tả của từng field.
- Ngữ nghĩa thông điệp
  - Ý nghĩa của thông tin trên từng field
- Các quy tắc về thời điểm và cách thức các tiến trình gửi và phản hồi lại các thông điệp.

Các giao thức công hữu:

- Được định nghĩa ở RFCs
  - HTTP: RFC 2616
- Cho phép làm việc phối hợp.
- e.g., HTTP, SMTP, BitTorrent

Các giao thức tư hữu:

- e.g., Skype, ppstream

What transport service does an app need?

Mất dữ liệu

- Một số ứng dụng (VD audio) có thể chấp nhận mất dữ liệu.
- Các ứng dụng khác (e.g., file transfer, telnet) đòi hỏi phải truyền dữ liệu chính xác tuyệt đối

Kịp thời

- Một số ứng dụng (e.g., Internet telephony, interactive games) đòi hỏi độ trễ thấp ở mức chấp nhận được.

Băng thông

- Vài ứng dụng (e.g., multimedia) cần băng thông tối thiểu để đạt hiệu quả.
- Các ứng dụng khác ("elastic apps") có thể thích nghi với băng thông hiện có (VD email, truyền file,...)

An toàn

- Mã hóa, toàn vẹn dữ liệu, ...

Transport service requirements of common apps

Loại ứng dụng	Mất dữ liệu	Băng thông	Kịp thời
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

Internet transport protocols services

TCP service:

- *connection-oriented:* cần quá trình thiết lập kết nối ("handshake") giữa các tiến trình server và client
- *Vận chuyển bảo đảm (reliable transport)* giữa 2 tiến trình gửi và nhận
  - Đúng nội dung, ko mất gói tin, ko trùng gói tin
  - Đúng thứ tự
- *flow control:* bên gửi ko gửi quá nhanh, làm tràn dữ liệu bên nhận
- *congestion control:* điều tiết tốc độ gửi dữ liệu khi mạng quá tải (ko hẳn tốt!).
- *Ko bảo đảm:* tính kịp thời, băng thông tối thiểu, an ninh

UDP service:

- Vận chuyển thường (ko bảo đảm) giữa các tiến trình gửi và nhận.
- Ko bảo đảm: quá trình thiết lập thông số, vận chuyển đảm bảo, flow control, congestion control, kịp thời, bảo đảm băng thông tối thiểu hay, an ninh

Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

Ứng dụng	Giao thức ở tầng ứng dụng	Giao thức vận chuyển
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

Chapter 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 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

Web and HTTP

First some jargon

- Trang Web bao gồm các đối tượng (objects)
- Đối tượng có thể là file HTML, ảnh JPEG, Java applet, file âm thanh,...
- Hầu hết trang Web bao gồm file HTML cơ sở (base HTML file) và nhiều đối tượng được tham chiếu
- Mỗi một đối tượng được định vị bằng một URL
- VD URL:

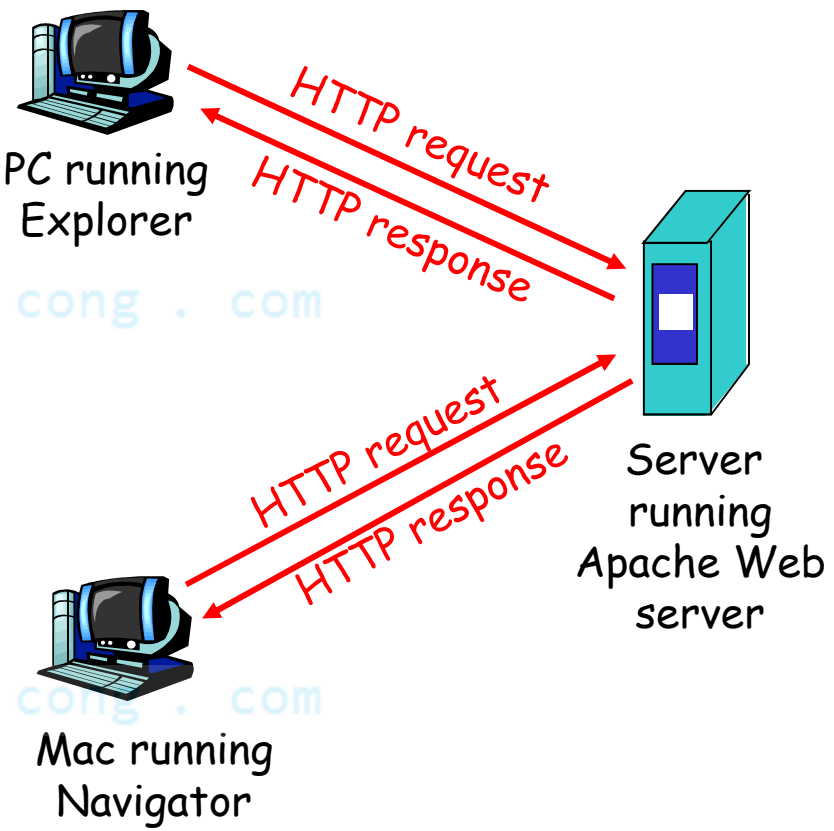
`www.someschool.edu/someDept/pic.gif`

host name | path name

HTTP overview

HTTP: hypertext transfer protocol

- Là giao thức ở tầng ứng dụng của Web
- HTTP/1.0 (RFC 1945)
- HTTP/1.1 (RFC 2616)
- Mô hình client/server
  - *client:* là trình duyệt (browser) cho phép yêu cầu (request), nhận và "hiển thị" các đối tượng Web.
  - *server:* Web server gửi các đối tượng đến client để đáp ứng các yêu cầu tương ứng từ client.



HTTP overview (continued)

Uses TCP:

- Client khởi tạo kết nối TCP (tạo socket) server, port 80.
- Server chấp nhật kết nối TCP từ phía client
- Trình duyệt và Web Server trao đổi các HTTP messages (chính là các message của giao thức ở tầng ứng dụng)
- Kết nối TCP được đóng.

HTTP is "stateless"

- Server không lưu lại bất kỳ thông tin nào về các request từ phía client

aside

Protocols that maintain "state" are complex!

- Thông tin quá khứ phải được quản lý
- Nếu liên kết server/client bị đổ bể, thông tin về trạng thái của 2 phía có thể ko nhất quán, cần được điều chỉnh.

HTTP connections

HTTP sử dụng cả 2 cách: nonpersistent connection hoặc persistent connection (mặc định)

Nonpersistent HTTP

- Tối đa một object được gửi qua kết nối TCP.

Persistent HTTP

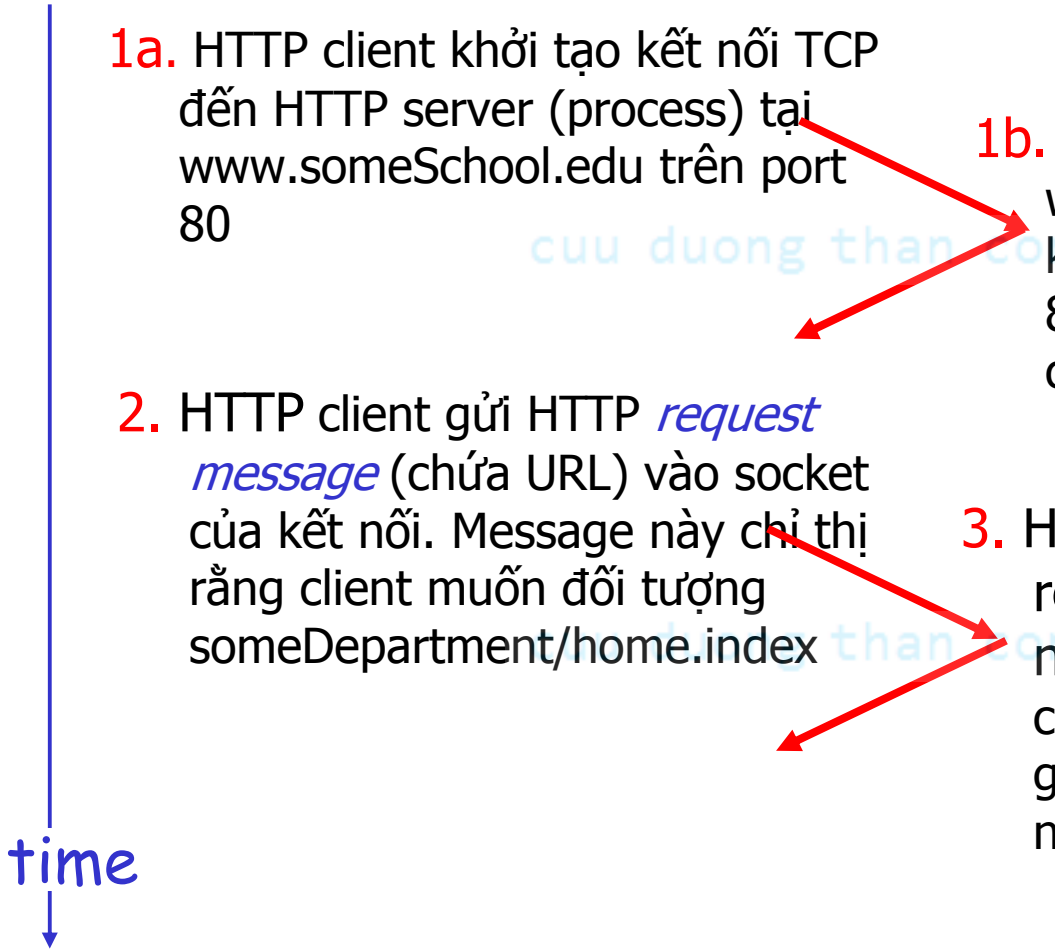
- Nhiều đối tượng có thể được gửi qua 1 kết nối TCP duy nhất giữa client và server

Nonpersistent HTTP

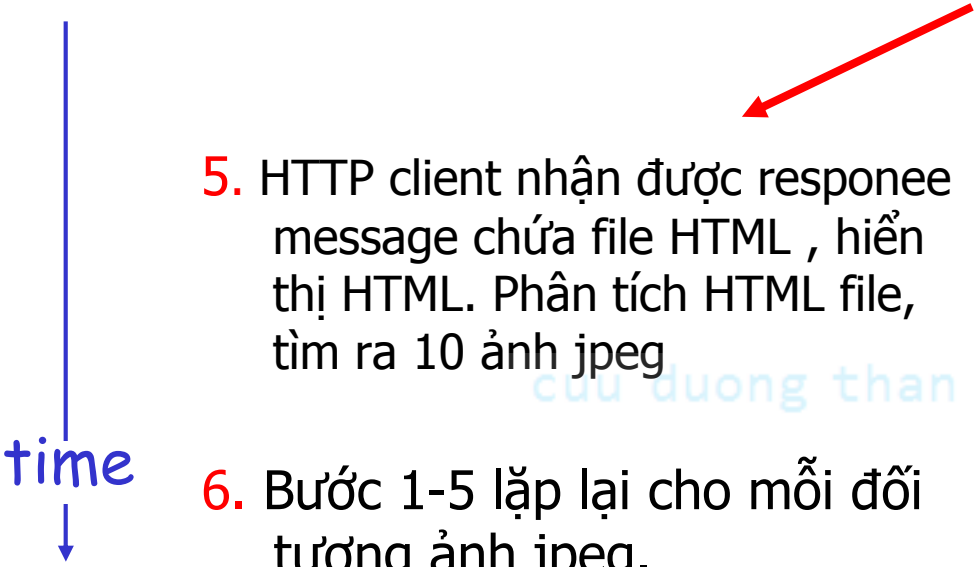
(trang này gồm text, tham chiếu đến 10 ảnh jpeg)

Giả sử user nhập vào chuỗi URL

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



Nonpersistent HTTP (cont.)



4. HTTP server đóng kết nối TCP

5. HTTP client nhận được responee message chứa file HTML , hiển thị HTML. Phân tích HTML file, tìm ra 10 ảnh jpeg

6. Bước 1-5 lặp lại cho mỗi đối tượng ảnh jpeg.

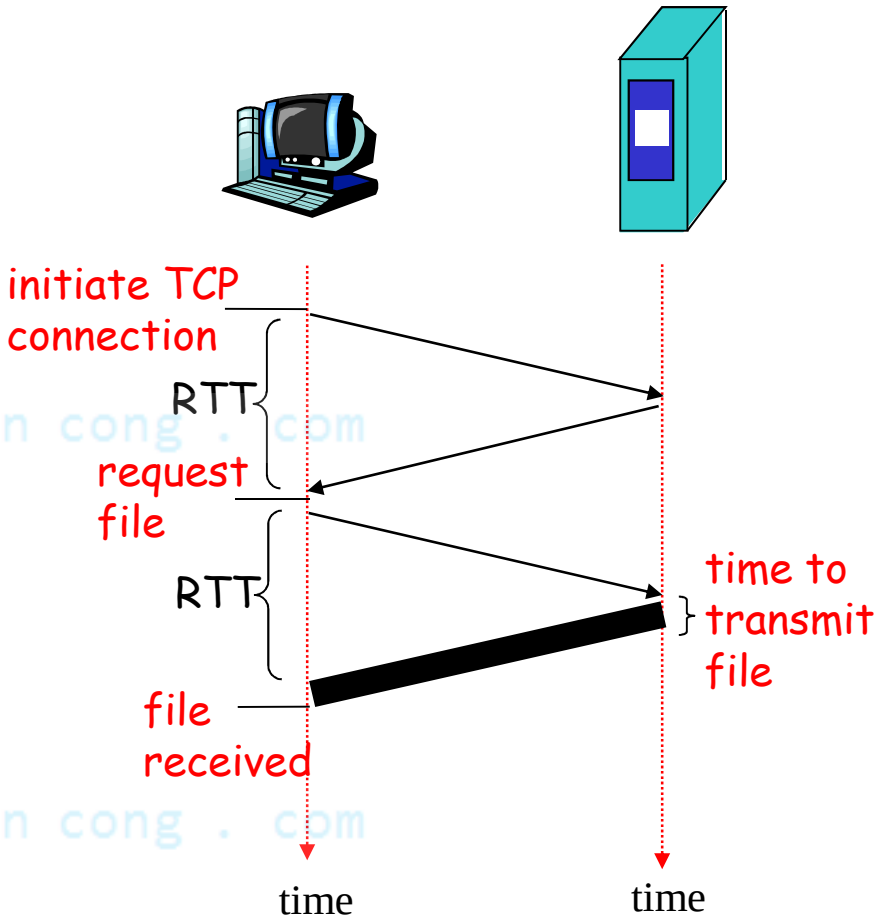
Non-Persistent HTTP: Response time

Đinh nghĩa RTT(Round trip time): là thời gian để 1 gói tin nhỏ đi từ client đến server và quay trở về.

Response time:

- Cần 1 RTT để khởi tạo kết nối TCP từ client > server
- Cần 1 RTT để client gửi request HTTP và vài byte của HTTP respons trả về
- Thời gian truyền file

total = 2RTT+transmit time



Persistent HTTP

Nonpersistent HTTP issues:

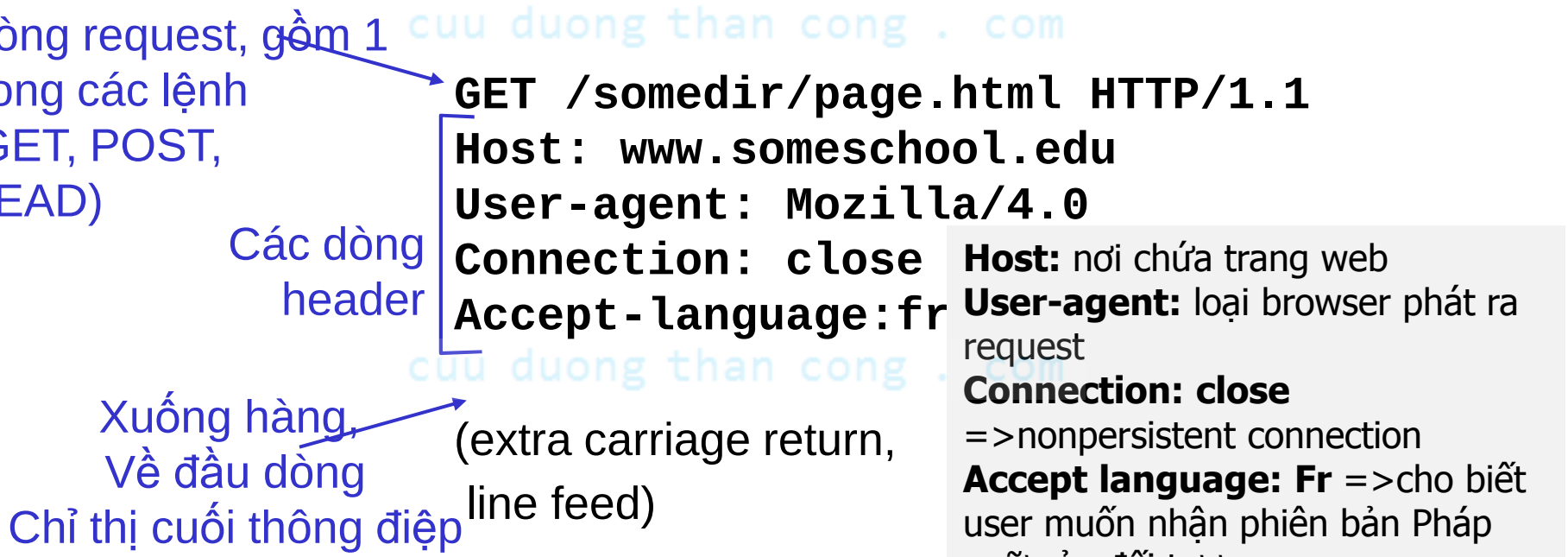
- Cần 2 RTT cho mỗi đối tượng
- Tốn chi phí xử lý của HĐH cho từng kết nối TCP
- Trình duyệt thường phải mở nhiều kết nối TCP đồng thời để tải các đối tượng Web

Persistent HTTP

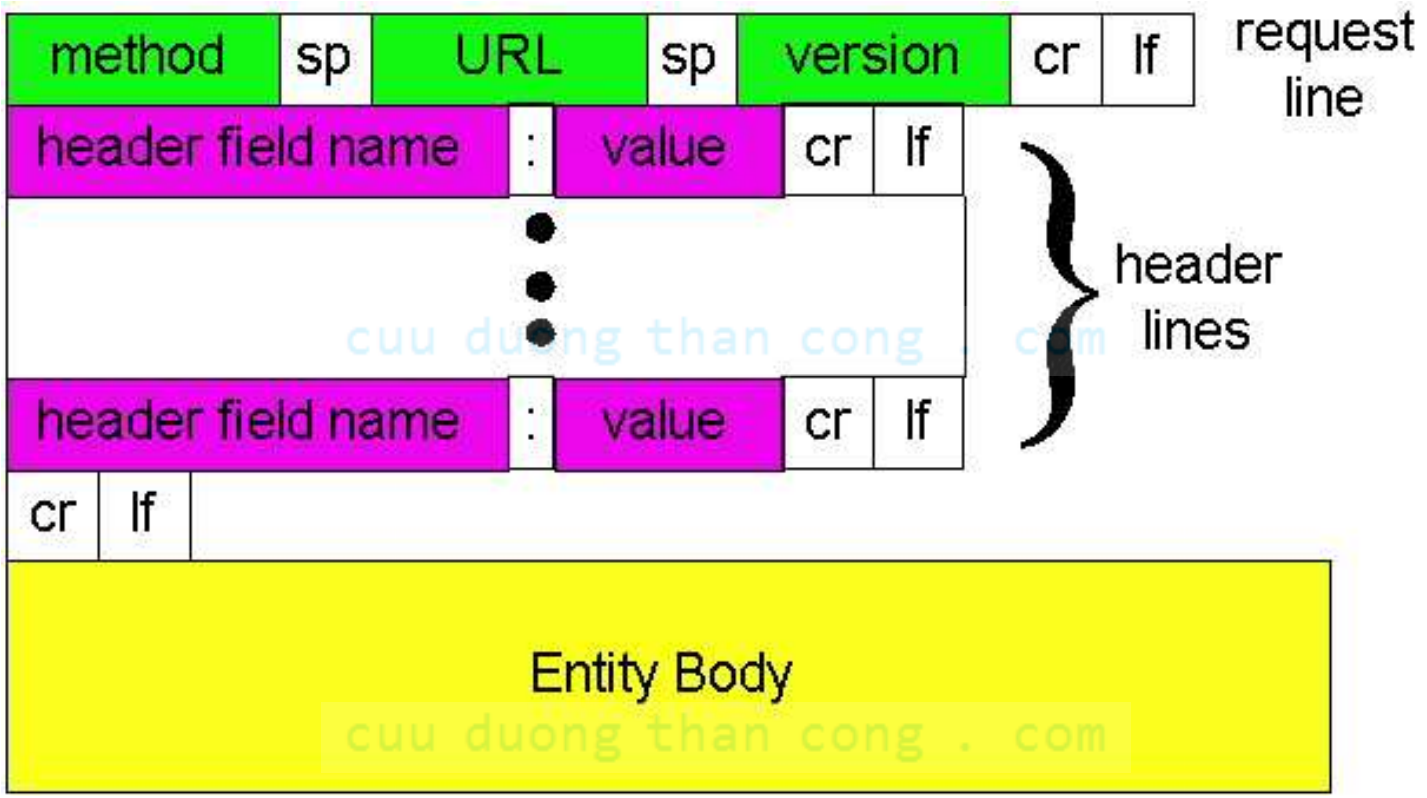
- Server vẫn duy trì kết nối sau khi gửi các phản hồi.
- Các HTTP message sau đó giữa client và server làm việc trên cùng kết nối đang mở.
- Client gửi request ngay khi nó gặp các đối tượng.
- Tốn một RTT cho tất cả các đối tượng được tải.

HTTP request message

- Hai kiểu thông điệp HTTP : *request, response*
- Thông điệp HTTP request:
  - ASCII (human-readable format)



HTTP request message: general format



Uploading form input

POST method:

- Trang Web thường bao gồm các form input
- Phần dữ liệu người dùng nhập vào, được gắn vào phần thân (entity body) của thông điệp request và gửi về cho server

URL method:

- Sử dụng GET method
- Phần dữ liệu người dùng nhập vào, được gắn vào trong field URL ở dòng request


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

Method types

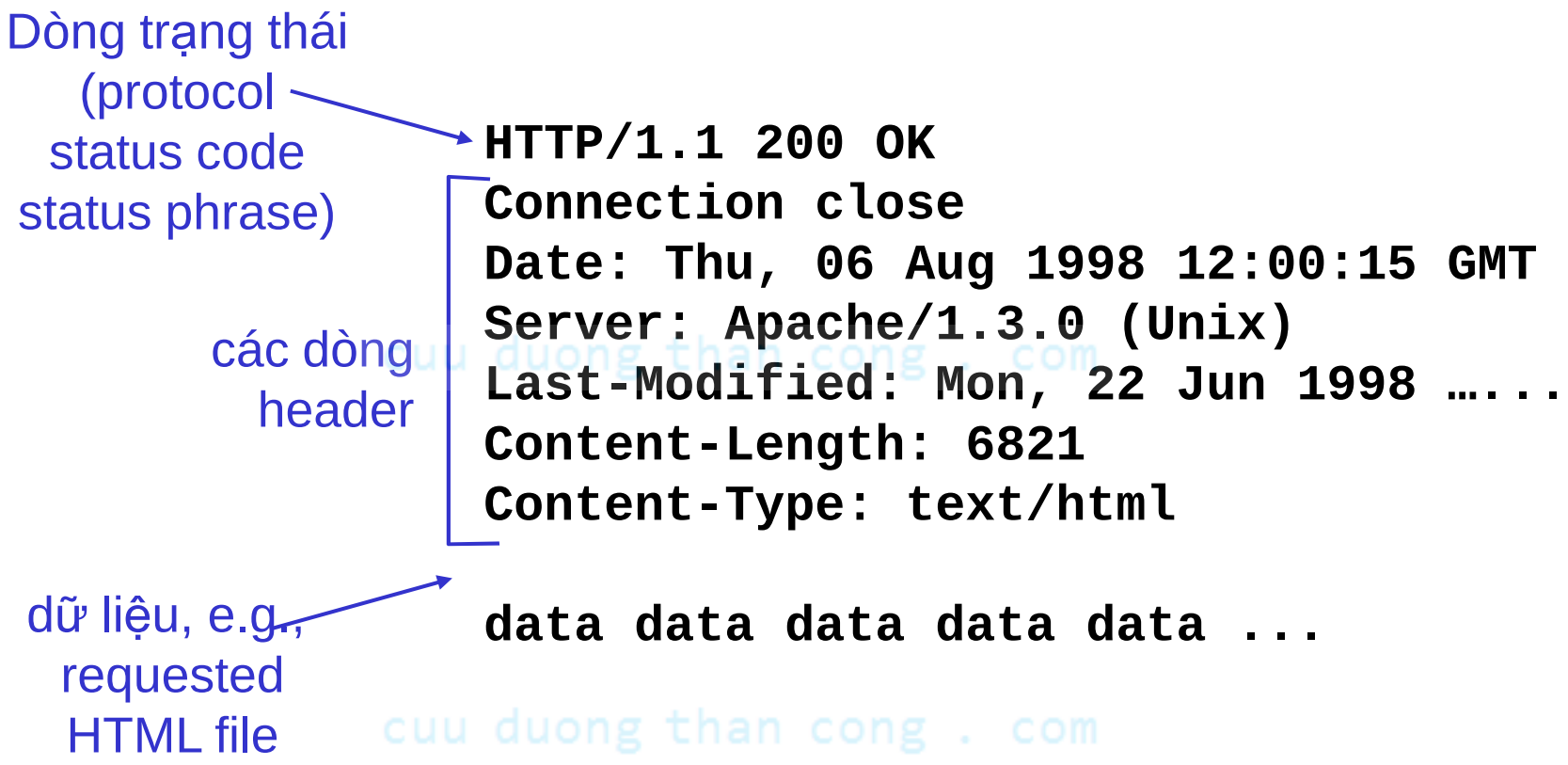
HTTP/1.0

- GET
- POST
- HEAD
  - Dùng cho mục đích debug
  - Yêu cầu server ko cần gửi đối tượng về client trong thông điệp response

HTTP/1.1

- GET, POST, HEAD
- PUT
  - Upload file có nội dung ở phần thân (entity body) của thông điệp request, đến nơi được chỉ ra ở URL field.
- DELETE
  - Xóa file được chỉ ra trong URL field

HTTP response message



HTTP response status codes

Nằm trên dòng đầu tiên của thông điệp phản hồi từ server

Một vài mã trạng thái:

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

Trying out HTTP (client side) for yourself

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

3. Look at response message sent by HTTP server!

User-server state: cookies

Many major Web sites use cookies

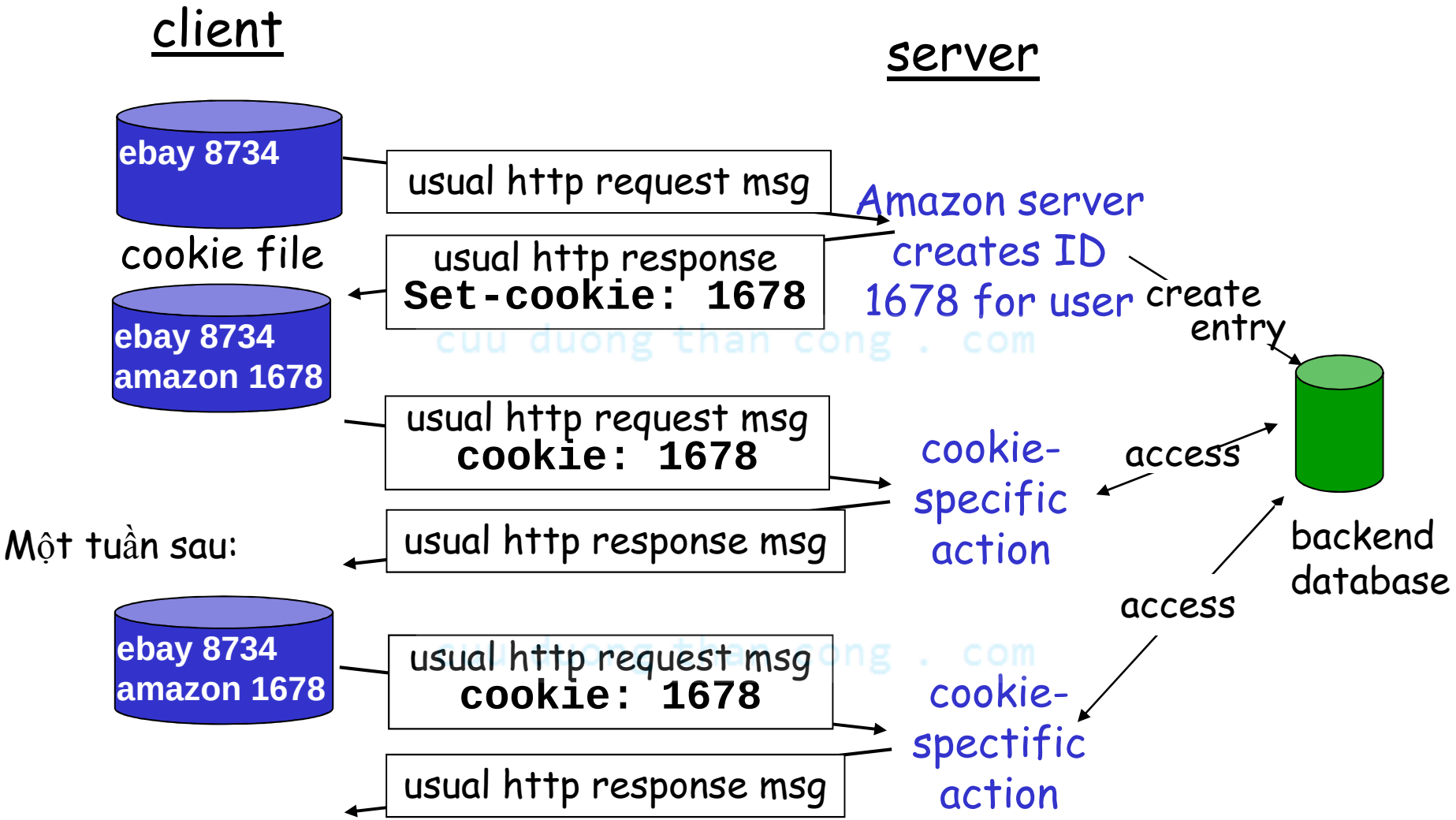
Four components:

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 luôn truy cập Internet từ máy PC
- Lần đầu ghé thăm site e-commerce, có sd cookies
- Khi e-commnerce server lần đầu nhận được 1 HTTP requests, nó tạo ra:
  - ID duy nhất cho Susan từ máy PC.
  - 1 dòng trong backend database ứng với ID

Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can bring:

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

aside

Cookies and privacy:

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

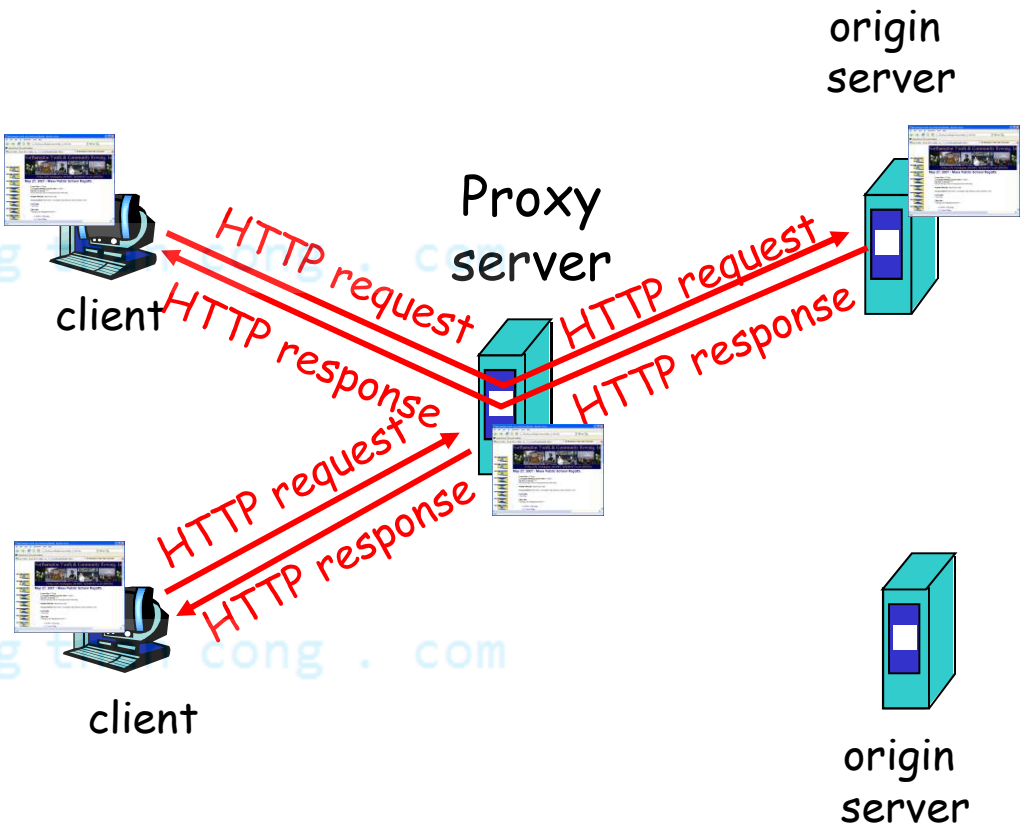
How to keep "state":

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

Web caches (proxy server)

Goal: satisfy client request without involving origin server

- User thiết lập trình duyệt: truy cập Web thông qua cache
- Trình duyệt gửi tất cả các HTTP requests tới cache
  - Nếu có đối tượng trong cache: cache trả về đối tượng cho trình duyệt
  - Ngược lại, cache "request" đối tượng từ server gốc, sau đó mới trả về cho trình duyệt.



More about Web caching

- Cache hành xử như client và như server
- Thông thường, cache được ISP (trường ĐH, công ty, các ISP địa phương) thiết lập.

Why Web caching?

- Giảm thời gian đáp ứng cho các request từ client
- Giảm lưu lượng đường truy cập ra ngoài của tổ chức
- Internet dense with caches: enables "poor" content providers to effectively deliver content (but so does P2P file sharing)

Caching example

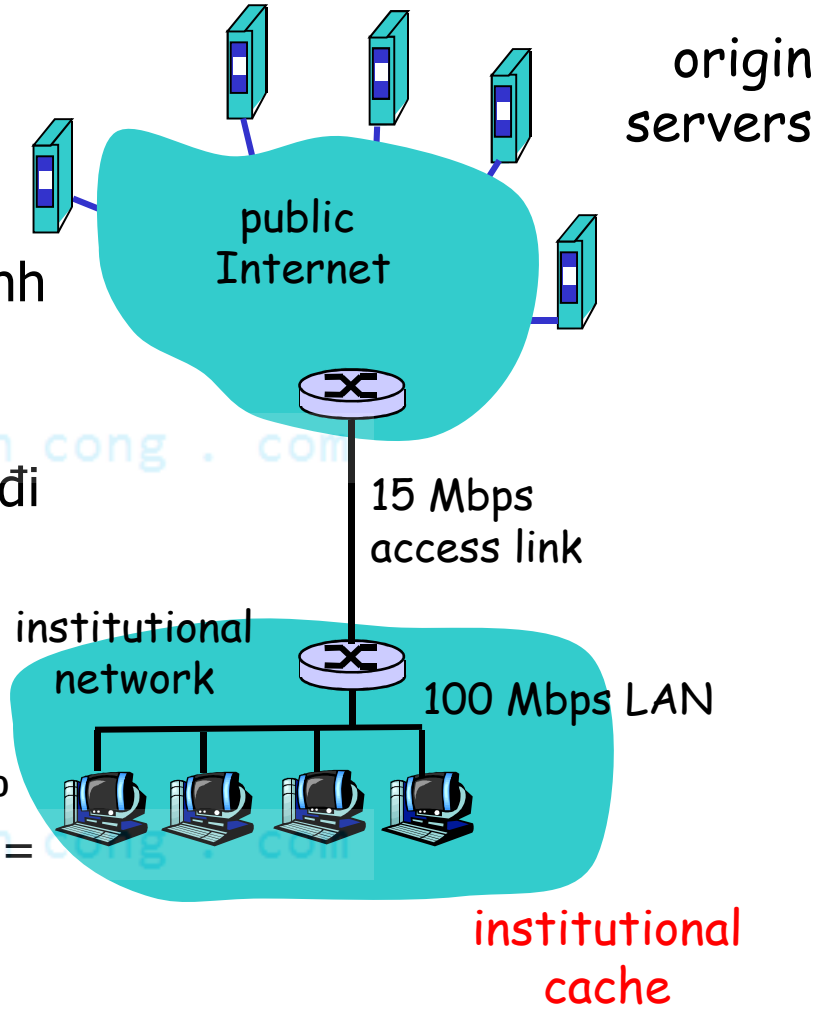
Giả định

- Kích thước bình quân của 1 đối tượng web = 1,000,000 bits
- Số lượng request trung bình từ trình duyệt của tổ chức gửi đến origin servers = 15 lần/giây
- Độ trễ tính từ router của tổ chức, đi đến origin server và quay về lại router = 2 giây

Hệ quả

- Mức độ sử dụng Web trên LAN = 15%
- Mức độ sử dụng Web trên access link = 100%
- Độ trễ tổng cộng = Internet delay + access delay + LAN delay

  = 2 sec + minutes + milliseconds



Caching example (cont)

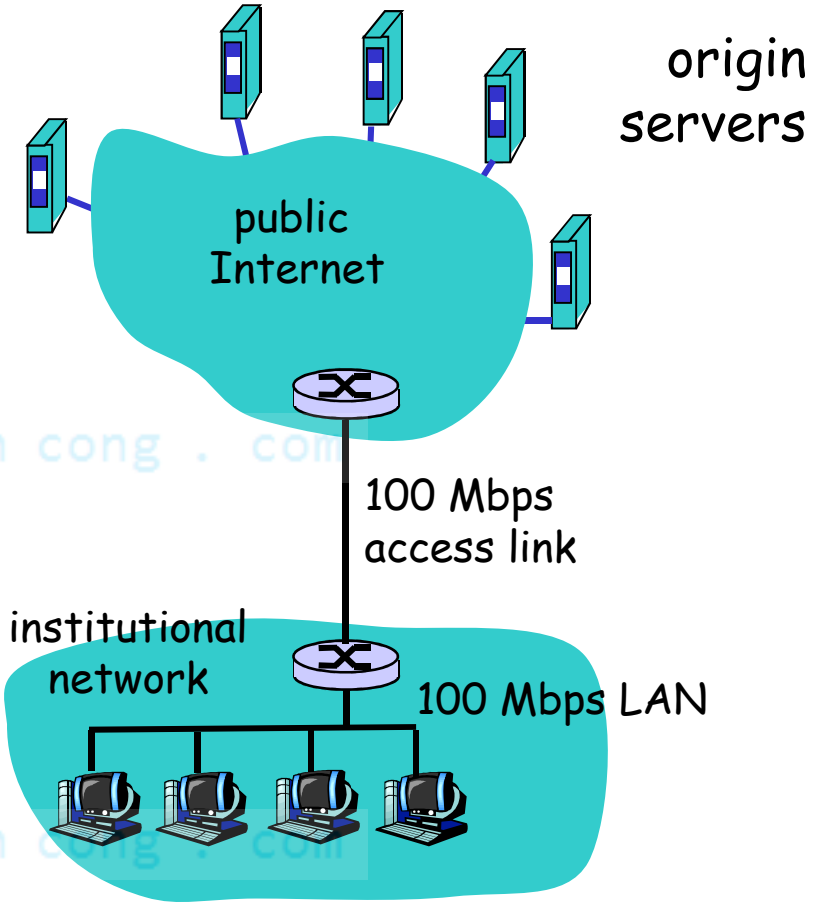
Giải pháp 1

- Tăng băng thông access link (đường truyền từ router của tổ chức ra router của Internet) lên 100 Mbps

consequence

- Mức độ sử dụng Web trên LAN = 15% = (15 lần/sec * 1000000 bits/lần)/100Mbps=15%
- Mức độ sử dụng Web trên access link= 15%
- Độ trễ tổng cộng= Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs
- Chi phí nâng cấp đường truyền rất đắt



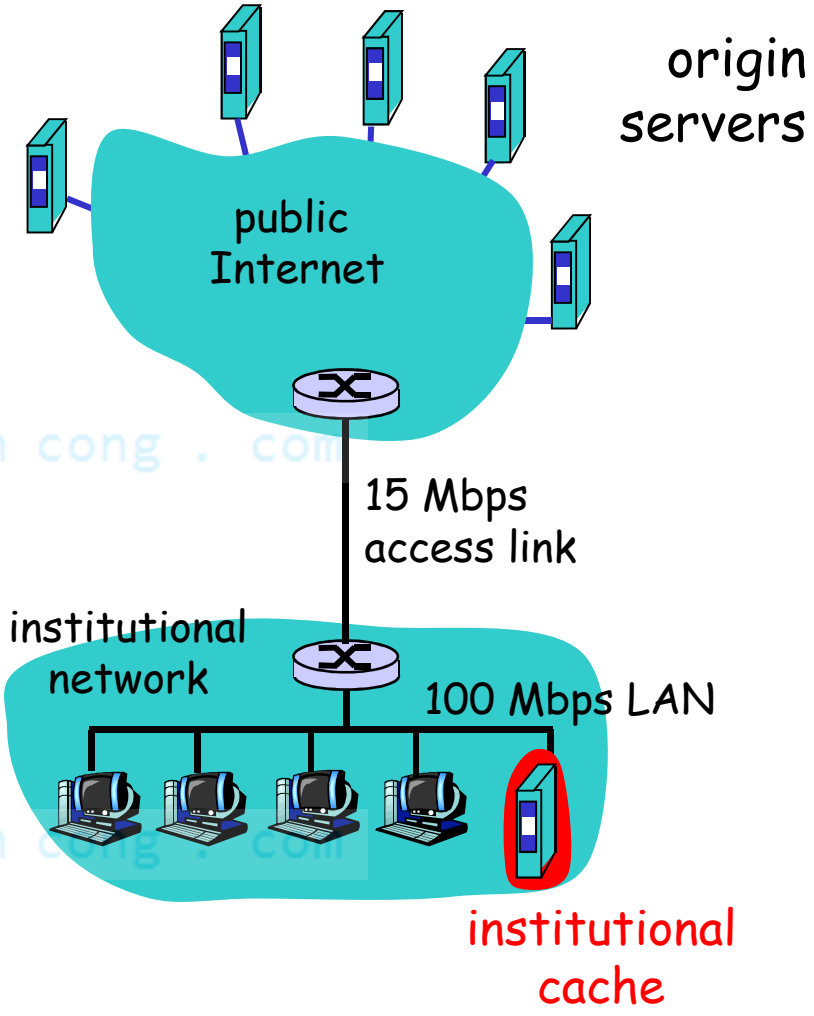
Caching example (cont)

Giải pháp khả thi: dùng cache

- Giả sử tỷ lệ đáp ứng của cache (hit rate) là 0.4

Hệ quả

- Có 40% request sẽ được đáp ứng gần như tức thì.
- Có 60% request sẽ được origin server đáp ứng.
- Mức độ sử dụng Web trên access link giảm xuống còn 60%, dẫn đến độ trễ không đáng kể (khoảng 10 msec trên đường truyền 15 Mbps)
- Độ trễ tổng cộng = Internet delay + access delay + LAN delay = .6*(2.01) secs + .4*milliseconds < 1.4 secs



Conditional GET

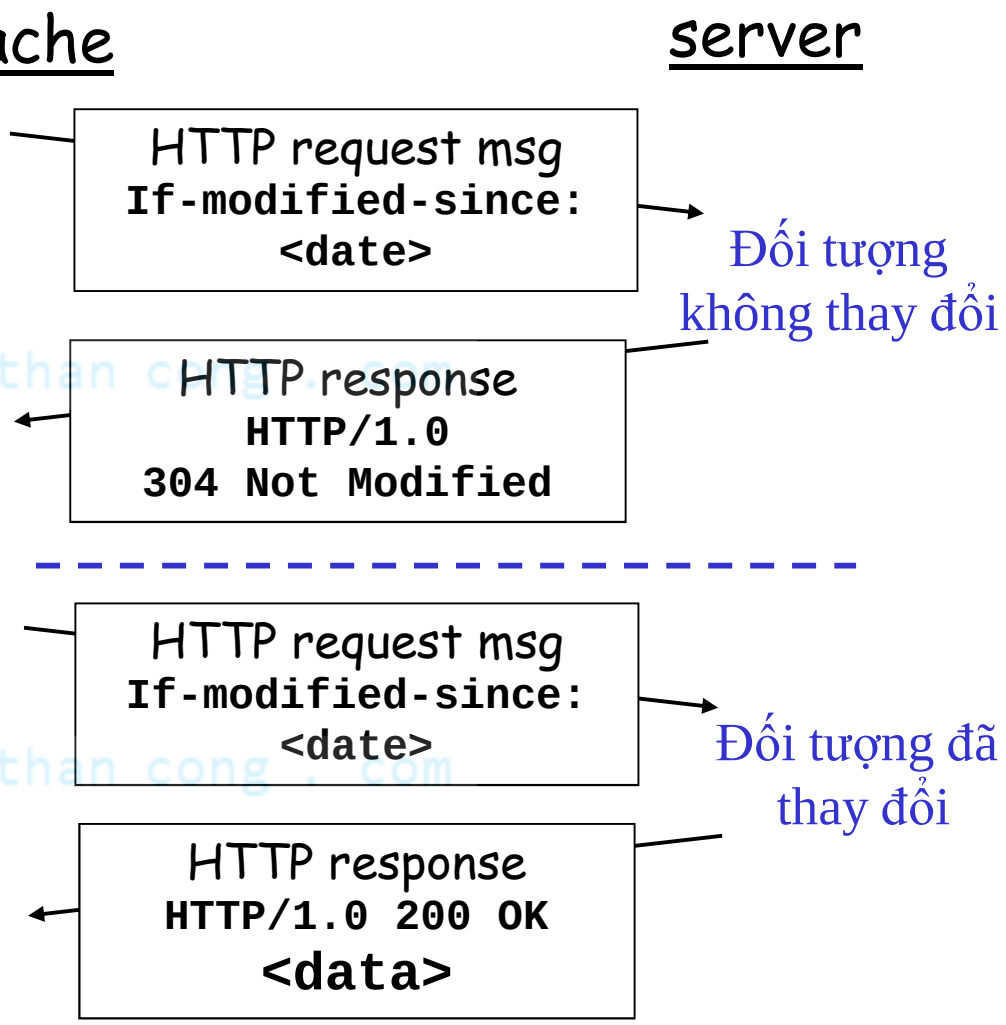
- Mục đích: không để server gửi đối tượng về cache nếu như cache đã có phiên bản được cập nhật của đối tượng
- Phía cache: khi gửi request đến server, cache chỉ rõ ngày cập nhật sau cùng của *đối tượng đang được lưu* ở cache

`If-modified-since: <date>`

Và cache y/cầu server chỉ gửi đối tượng nếu như đối tượng có thay đổi.

- Phía server: không cần gửi đối tượng nếu như biết đối tượng ko thay đổi

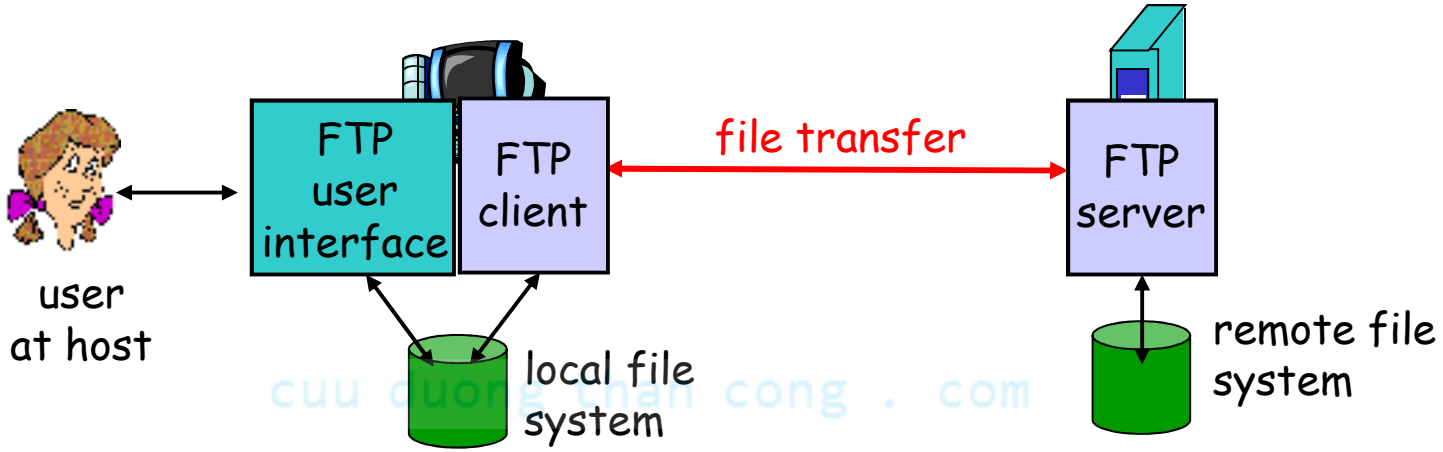
`HTTP/1.0 304 Not Modified`



Chapter 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 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

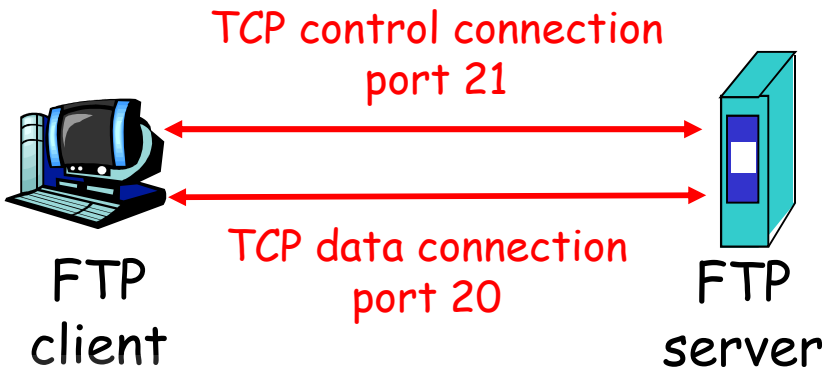
FTP: the file transfer protocol



- Truyền file đến/từ một host ở xa.
- Mô hình Client/server
  - *client:* bên khởi tạo việc truyền/nhận
  - *server:* host ở xa
- ftp: RFC 959
- ftp server: port 21

FTP: separate control, data connections

- FTP client liên lạc với FTP Server ở port 21, sử dụng giao thức TCP.
- Client truy cập đến Server qua *kết nối điều khiển* (gọi là control conection)
  - Client có thể làm nhiều việc (VD xem thư mục của Server, tải file, đổi thư mục hiện hành,…) bằng cách gửi lệnh qua kết nối điều khiển
  - FTP được coi là gửi thông tin điều khiển ngoài dải băng ("out of band")
- Khi server nhận được lệnh chuyển file, nó sẽ mở 1 kết nối thứ 2 đến client (gọi là *kết nối dữ liệu*, data connection), để truyền file.



- Sau khi truyền xong file, server sẽ đóng kết nối dữ liệu này.
- Server sẽ mở một kết nối dữ liệu khác để truyền file khác
- FTP server có duy trì "trạng thái": thư mục hiện hành, các chứng thực trước đó.

FTP commands, responses

Sample commands:

- Được gửi như đoạn ASCII text qua kết nối điều khiển
- **USER *username***
- **PASS *password***
- **LIST** trả về danh sách file trong thư mục hiện hành
- **RETR filename** tải xuống file
- **STOR filename** đặt file lên 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**

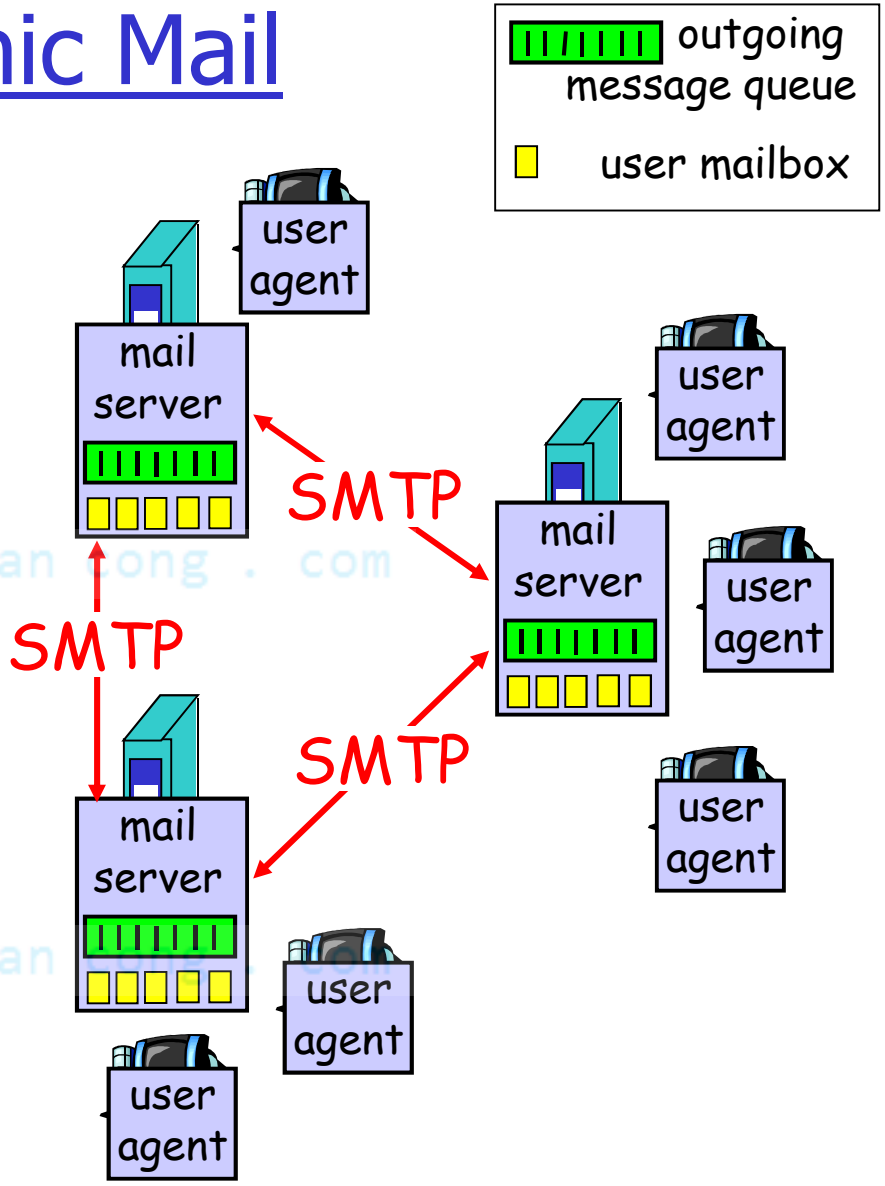
Chapter 2: Application layer

- 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 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

Electronic Mail

Ba thành phần chính:

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



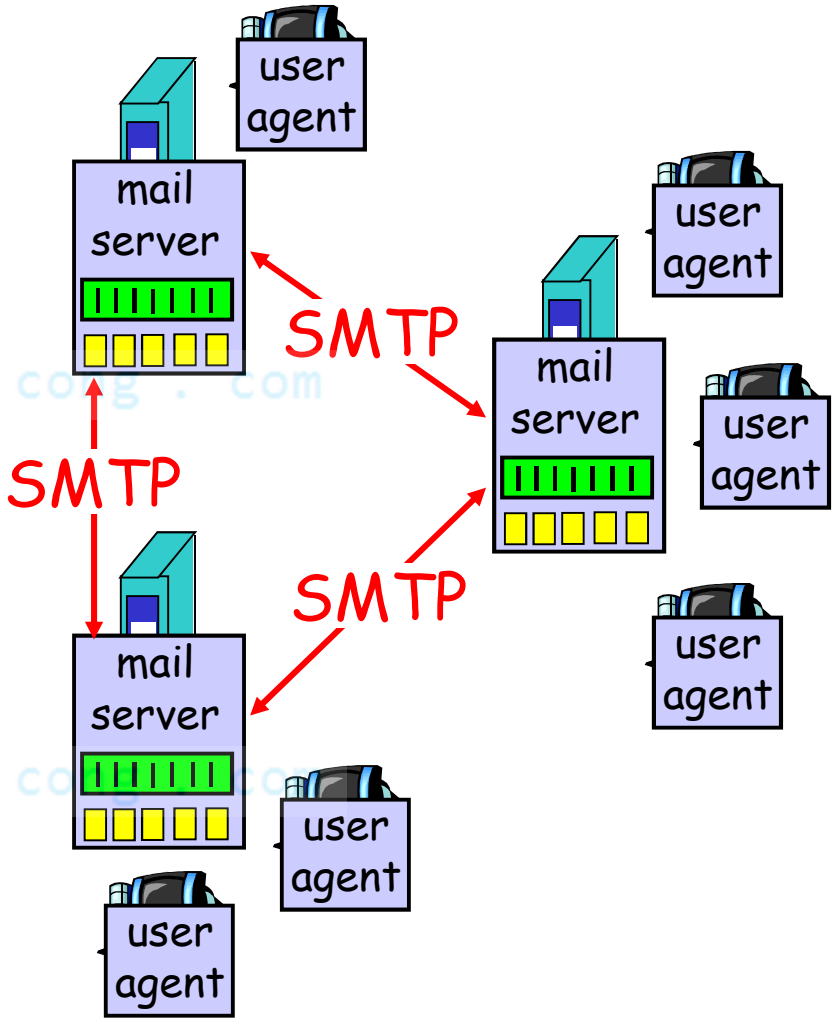
User Agent

- Còn gọi là "Chương trình đọc mail"
- Soạn message, đọc message
- e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- Các message đến, gửi đi đều được lưu trữ ở server

Electronic Mail: mail servers

Mail Servers

- mailbox chứa các message đến
- message queue chứa các message được gửi đi
- SMTP protocol giao thức giữa các mail server, để gửi messages
  - client: sending mail server
  - "server": receiving mail server

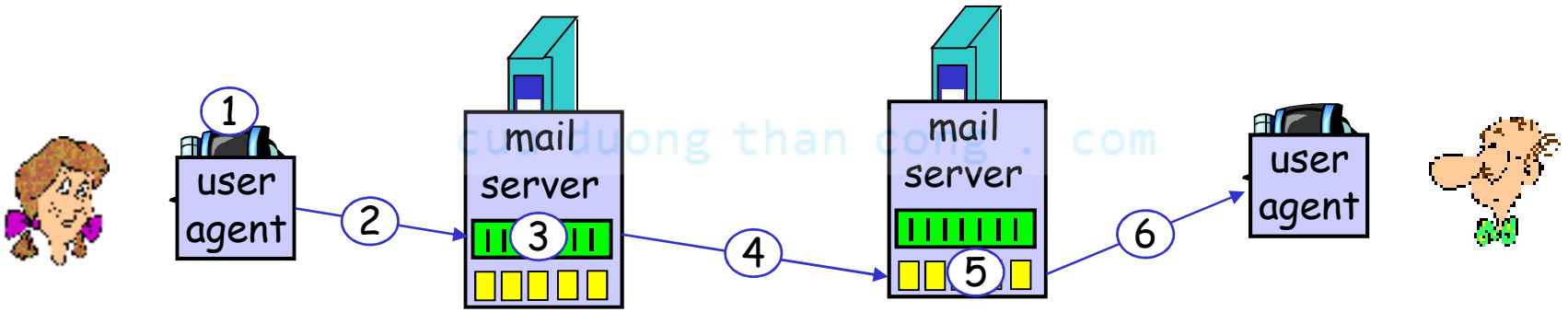


Electronic Mail: SMTP [RFC 2821]

- Sử dụng TCP để "gửi bảo đảm" các message từ client đến server, dùng port 25.
- Chuyển trực tiếp : từ server-gửi đến server-nhận
- Ba giai đoạn gửi
  - Bắt tay (chào mừng, ở tầng Ứng dụng)
  - Chuyển message
  - Đóng
- Sử dụng lối tương tác: lệnh/phản hồi
  - Lệnh (commands): ASCII text
  - Phản hồi (response): mã trạng thái (status code) và cụm từ mô tả
- Các thông điệp phải sử dụng bảng mã ASCII 7-bit

Scenario: Alice sends message to Bob

1) Alice sử dụng 1 user agent để soạn 1 message và đề gửi tới `bob@someschool.edu`
2) User Agent gửi message tới mail server; message được xếp vào hàng đợi.
3) Mail server phía Alice mở một kết nối TCP connection đến mail server phía Bob
4) Mail server phía Alice gửi message của Alice qua kết nối TCP đến Mail server phía Bob
5) Mail server phía Bob đặt message vào mailbox của Bob
6) Bob dùng user agent để đọc message.



Sample SMTP interaction

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

Try SMTP interaction for yourself:

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands
- above lets you send email without using email client (reader)

SMTP: final words

- SMTP sử dụng kết nối thường trực (persistent connections)
- SMTP đòi hỏi message (bao gồm cả header và body) sử dụng bảng mã ASCII 7-bit
- SMTP server sử dụng CRLF.CRLF để xác định điểm cuối của message

So sánh với HTTP:

- HTTP: kéo
- SMTP: đẩy
- Cả hai đều sử dụng lối tương tác lệnh/phản hồi, mã trạng thái.
- HTTP: mỗi đối tượng được bao bọc trong chính msg phản hồi của chính nó
- SMTP: nhiều đối tượng được gửi trong các *thông điệp đa phần* (multipart msg)

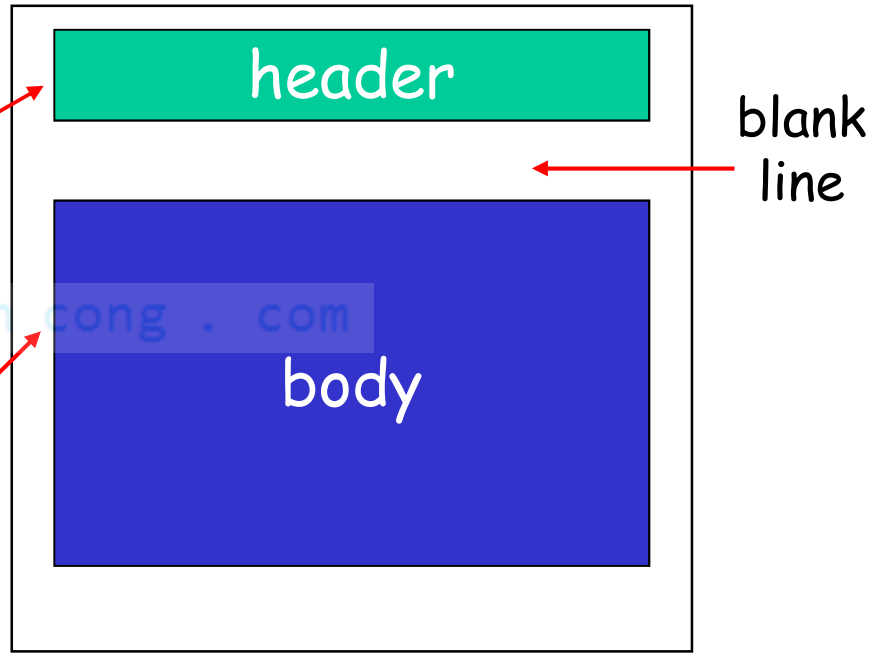
Mail message format

SMTP: giao thức trao đổi message

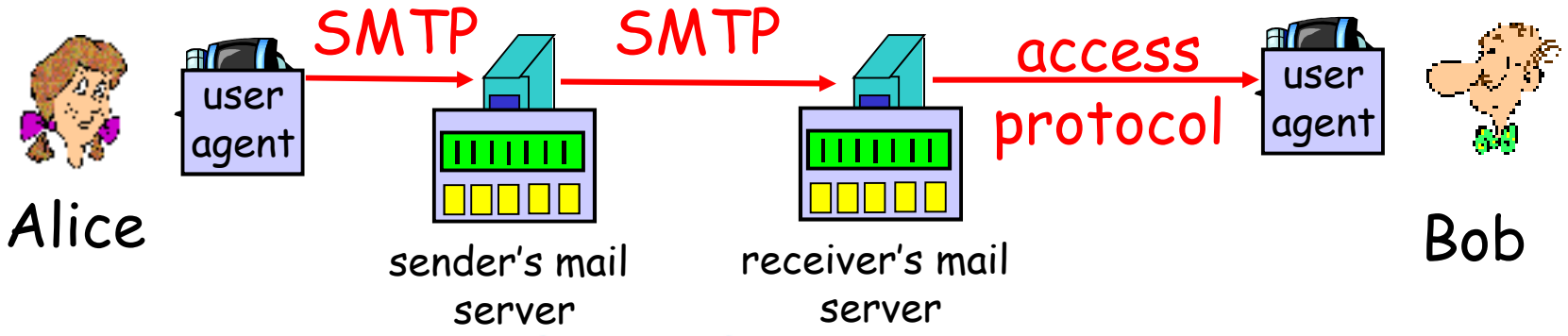
RFC 822: chuẩn định dạng msg dạng văn bản:

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

  Khác với các lệnh SMTP!
- Phần thân (body)
  - Nội dung msg, chỉ bao gồm các ký tự ASCII



Mail access protocols



- SMTP: chuyển tiếp/lưu trữ msg vào server của bên nhận
- Để tải msg từ server về, sử dụng giao thức truy cập mail:
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) và tải msg
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - Nhiều tính năng hơn (phức tạp hơn)
    - Thao tác trên các msg được lưu ở server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

Giai đoạn xác thực (authorization phase)

- Các lệnh từ client:
  - **`user:`** khai báo user
  - **`pass:`** mật mã
- Các phản hồi từ server
  - **`+OK`**
  - **`-ERR`**

Giai đoạn giao dịch (transaction phase), client:

- **`list:`** liệt kê số hiệu msg
- **`retr:`** Tải msg theo số thứ tự
- **`dele:`** xóa
- **`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
```

POP3 (more) and IMAP

More about POP3

- Ví dụ trước sử dụng chế độ "tải và xóa mail".
- Bob ko thể đọc lại email nếu anh ta thay đổi client
- "tải và giử mail": chép lại các msg trên các client khác nhau
- POP là giao thức phi trạng thái (stateless) qua nhiều session

IMAP

- Giữ tất cả các message ở một nơi: server
- Cho phép user tổ chức các msg vào các folders
- IMAP giữ các user state qua các session
  - Tên của các folders và ánh xạ giữa Msg ID và tên folder

Chapter 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 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

DNS: Domain Name System

Người: nhiều "tên":

- CMND, tên, Số hộ chiếu

Internet hosts, routers:

- Địa chỉ IP (32 bit) – được sử dụng để đánh địa chỉ cho việc truyền datagram
- "tên miền", e.g., ww.yahoo.com – dành cho con người

Q: ánh xạ giữa địa chỉ IP và tên?

Domain Name System:

- *CSDL phân bố*

  được triển khai trên một hệ thống các *name servers*, được tổ chức phân cấp.
- *application-layer protocol* : cho phép host và name servers giao tiếp với nhau nhằm *phân giải* tên (chuyển đổi địa chỉ IP/hostname)
  - Lưu ý: là chức năng lõi của Internet, được cài đặt như là một application-layer protocol
  - complexity at network's "edge"

DNS

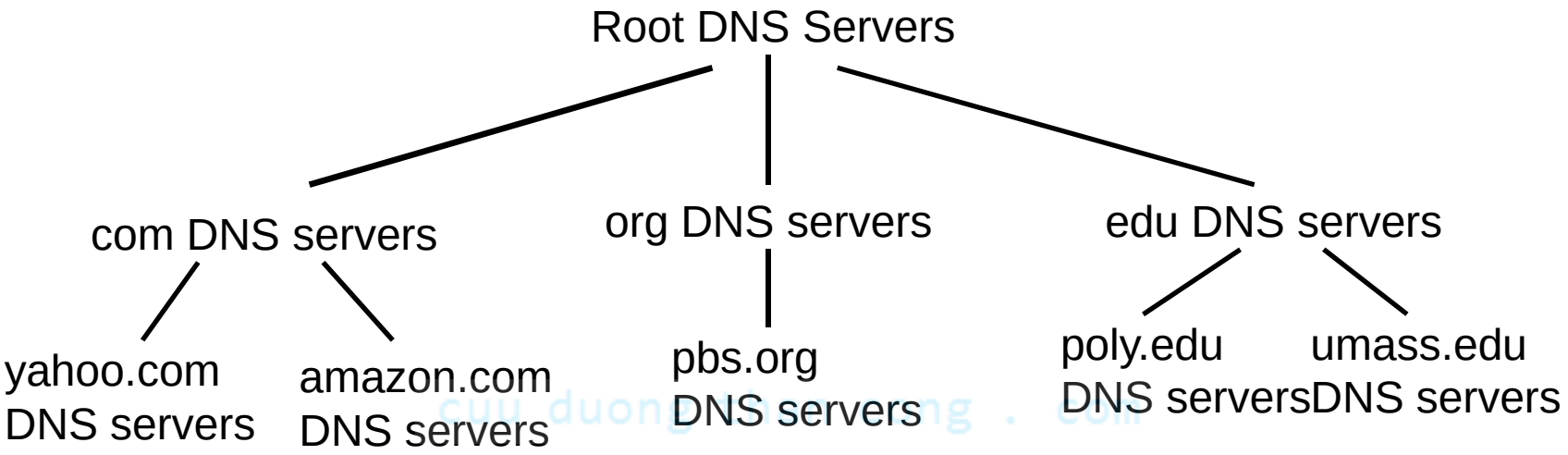
Các dịch vụ DNS cung cấp

- Cho biết địa chỉ IP khi biết hostname
- Đặt bí danh host
  - Canonical, alias names
- Đặt bí danh cho mail server
- Phân bố tải
  - replicated Web servers: sử dụng nhiều địa chỉ IP ứng với một tên chính thức.

Tại sao ko nên tập trung DNS?

- Nguy cơ hỏng hóc tại DNS tập trung làm cho toàn hệ thống Internet bị ảnh hưởng.
- Lưu lượng tập trung tại DNS lớn=> đáp ứng chậm
- 1 DNS Server ko thể ở gần tất cả các client => làm gia tăng các kết nối đường dài đến DNS => tắt nghẽn.
- Bảo trì 1 DNS cho toàn cầu!

Distributed, Hierarchical Database



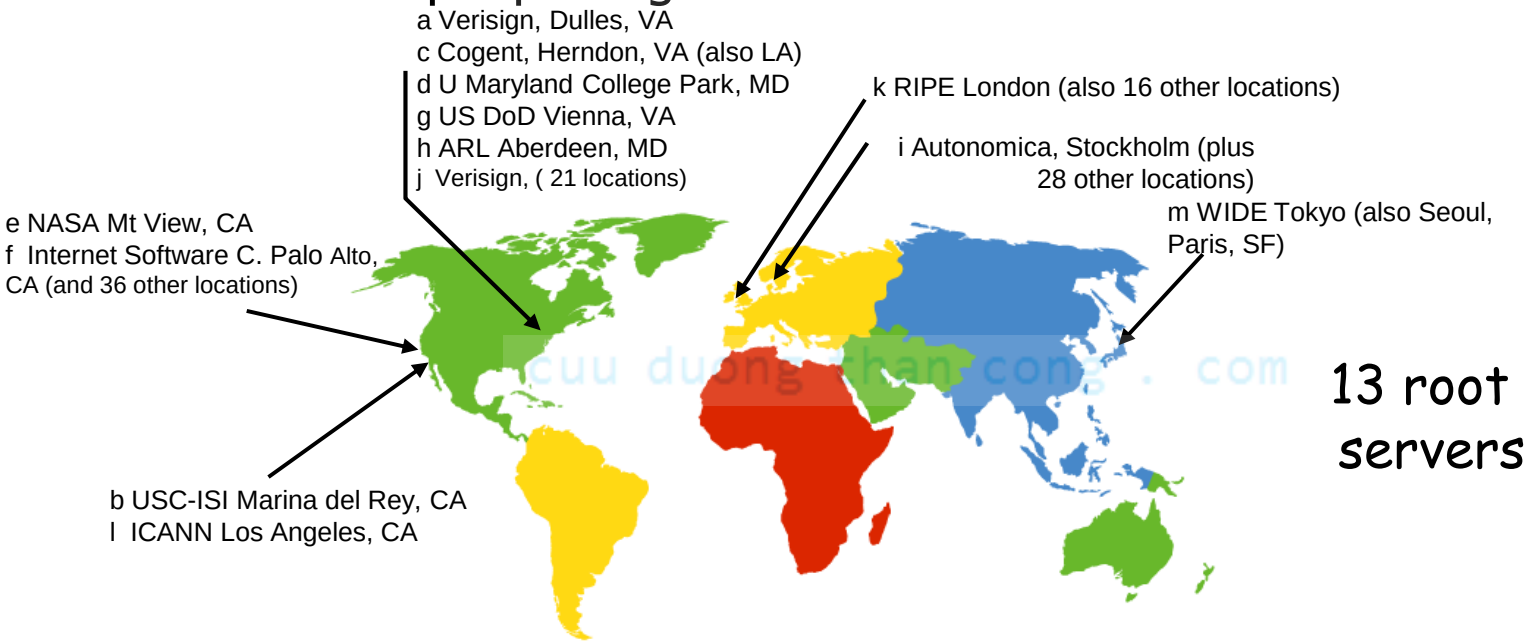
Client wants IP for www.amazon.com; 1st approx:

- Client hỏi root server để tìm DNS Server phụ trách tên miền com.
- Client lại hỏi tiếp DNS Server phụ trách tên miền com, để tìm DNS Server phụ trách tên miền amazon.com
- Client lại hỏi DNS Server phụ trách tên miền amazon.com, để lấy địa chỉ IP cho www.amazon.com

DNS: Root name servers

- Khi các local name server không thể phân giải tên miền (do ko lưu trong DB), nó sẽ liên lạc với root name server
- root name server:
  - Liên lạc với các authoritative name server và yêu cầu server này lo dùm
  - Nhận kết quả phân giải từ authoritative name server
  - Trả kết quả phân giải về local name server

Recursive query



13 root name servers worldwide

TLD and Authoritative Servers

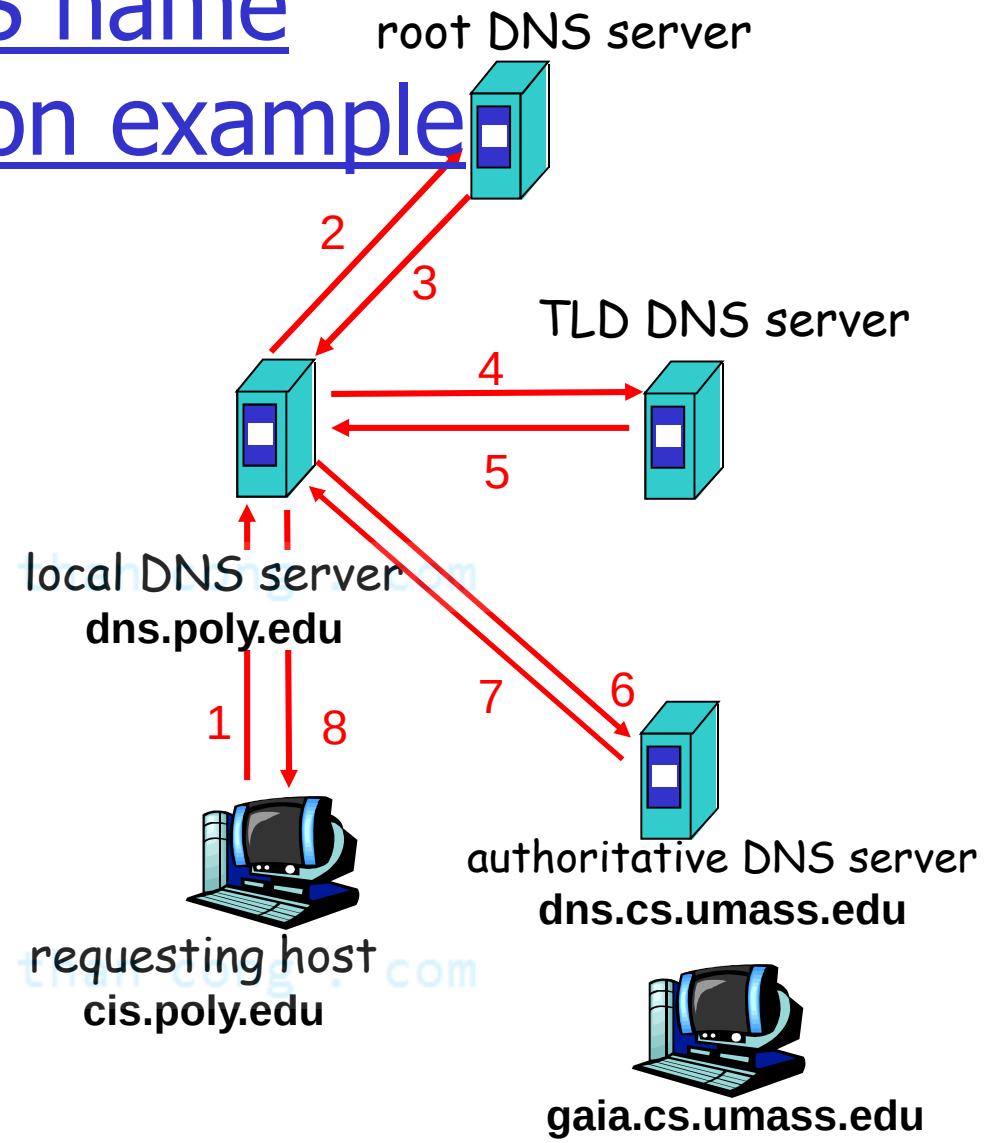
- Top-level domain (TLD) servers:
  - Chịu trách nhiệm đối với các tên com, org, net, edu,... và tất cả các tên miền quốc gia mức 1 như uk, fr, ca, vn, ...
    - Network Solutions quản lý các servers dành cho TLD com
    - Educause quản lý các servers dành cho TLD edu
- Authoritative DNS servers:
  - DNS server của các tổ chức, cung cấp tất cả các ánh xạ tên-địa chỉ IP của các server của tổ chức (e.g Web, mail)
  - Các DNS Server này được chính các tổ chức hoặc nhà cung cấp dịch vụ quản lý.

Local Name Server

- Không hoàn toàn thuộc về Hệ thống DNS được tổ chức theo cấu trúc phân cấp.
- Mỗi ISP (ISP địa phương, công ty, trường đại học) đều có.
  - Cũng được gọi là "default name server"
- Khi một host truy vấn DNS, câu hỏi được gửi đến cho local DNS Server
  - Làm việc như 1 máy đại diện (proxy), chuyển tiếp truy vấn DNS đến hệ thống phân cấp DNS.

DNS name resolution example

- Một host ở cis.poly.edu muốn biết địa chỉ IP address của gaia.cs.umass.edu

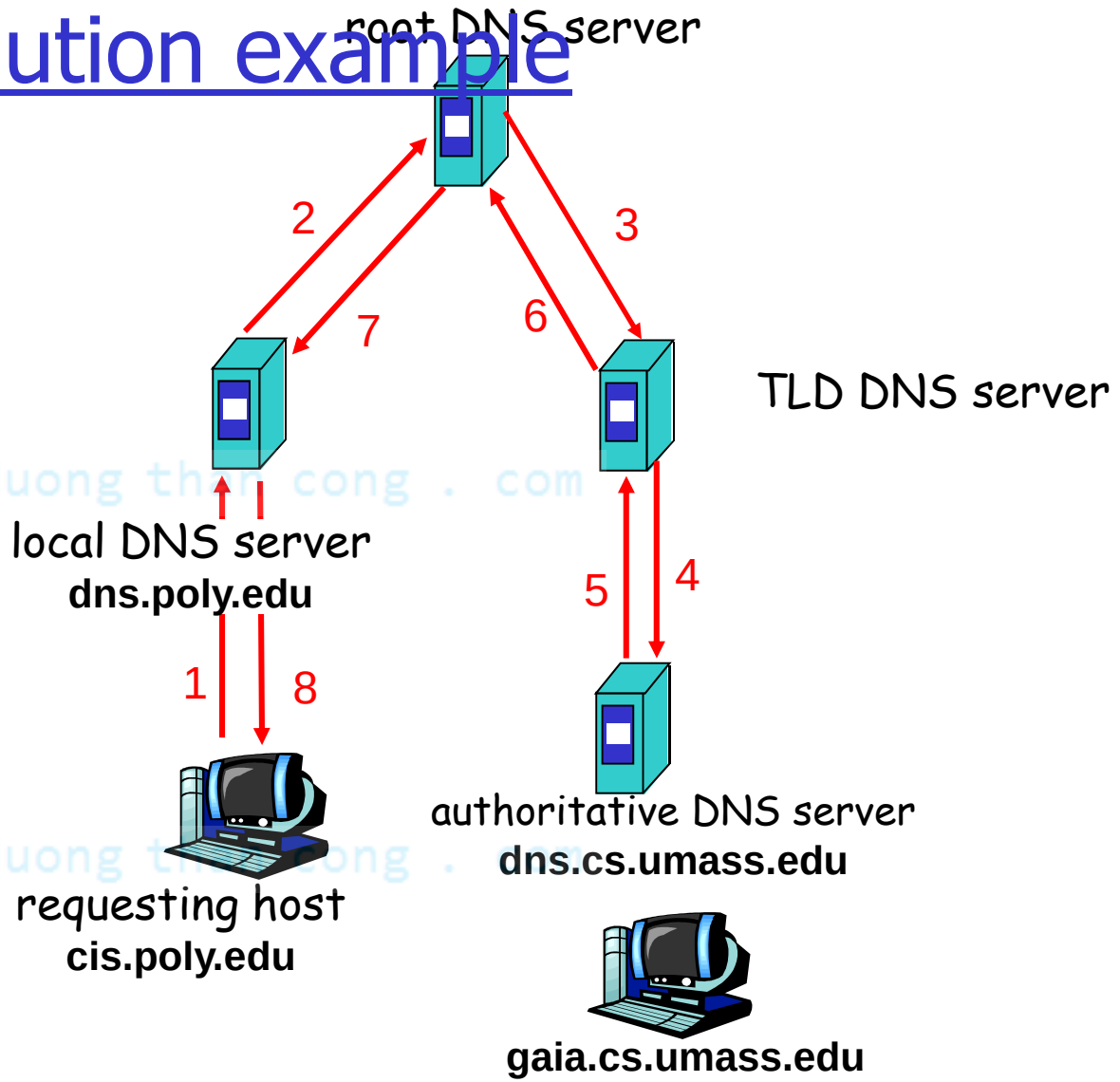


- iterated query:
- Server được hỏi sẽ trả lời là tên của server cần hỏi kế tiếp
- "I don't know this name, but ask this server"

DNS name resolution example

recursive query:

- Đặt trách nhiệm phân giải tên miền hoàn toàn lên các server được hỏi.
- heavy load?



DNS: caching and updating records

- Mỗi khi name server có được mapping, nó sẽ lưu cache lại mapping này
  - Các mục cache sẽ hết thời hiệu (biến mất) sau một thời gian.
  - TLD Server thường được lưu cache tại các local name servers
    - Do đó các root name server ít khi được viếng thăm.
- Cơ chế cập nhật/báo tin được thực hiện theo thiết kế của IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

DNS records

DNS: CSDL phân bố, lưu các resource records (RR)

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

- Type=A
  - **name** là hostname
  - **value** là IP address
- Type=NS
  - **name** là domain (VD foo.com)
  - **value** là hostname của authoritative name server dành cho domain này
- Type=CNAME
  - **name** is bí danh của vài tên chính thức ("canonical" name)

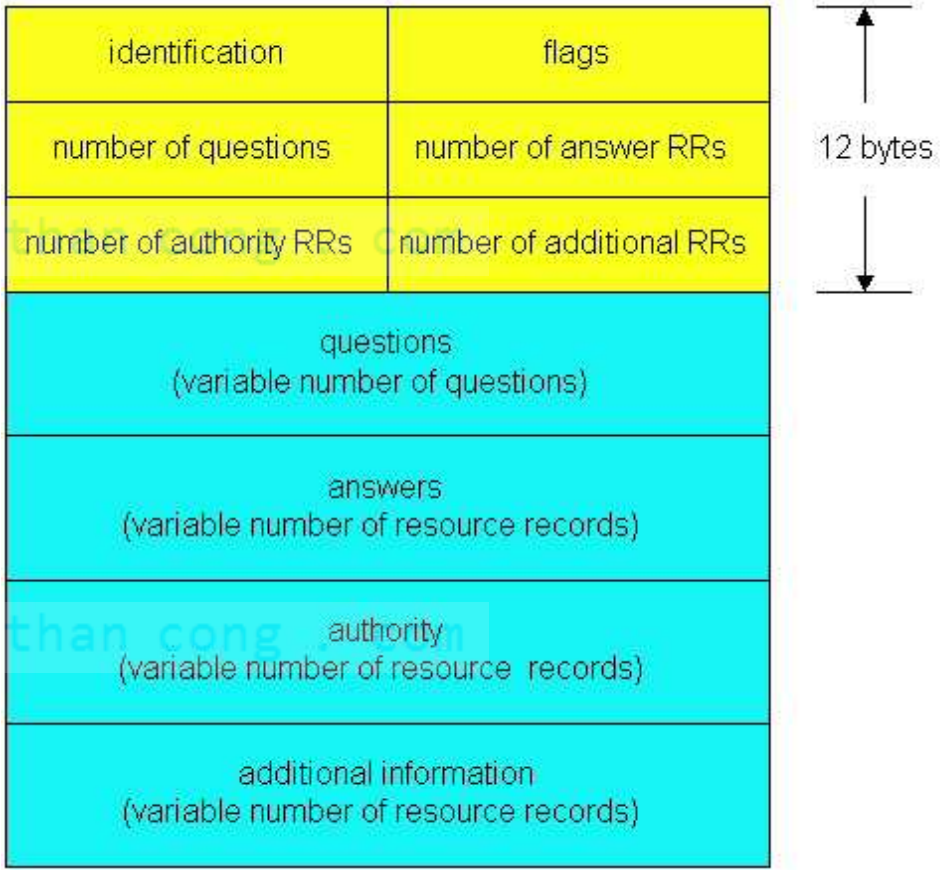
    www.ibm.com là bí danh của servereast.backup2.ibm.com
  - **value** là tên chính thức (canonical name)
- Type=MX
  - **value** là tên của mailserver được liên kết với **name**

DNS protocol, messages

DNS protocol : quy định các *query* msg và *reply* msg, cả hai có cùng định dạng message

Phần msg header

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



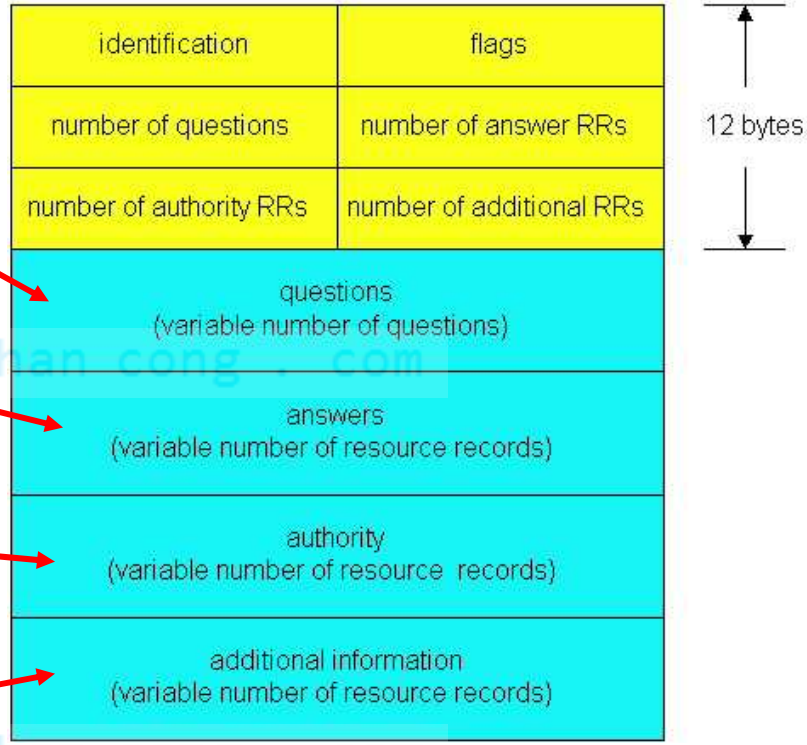
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkuptopia.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)
```

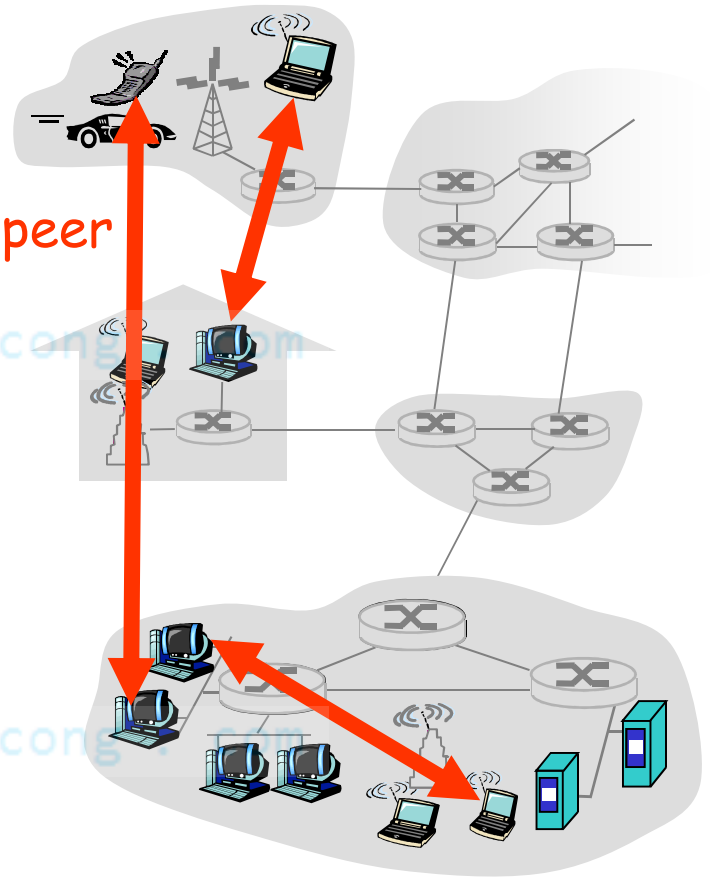
- create authoritative server Type A record for www.networkuptopia.com; Type MX record for networkutopia.com
- How do people get IP address of your Web site?

Chapter 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 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

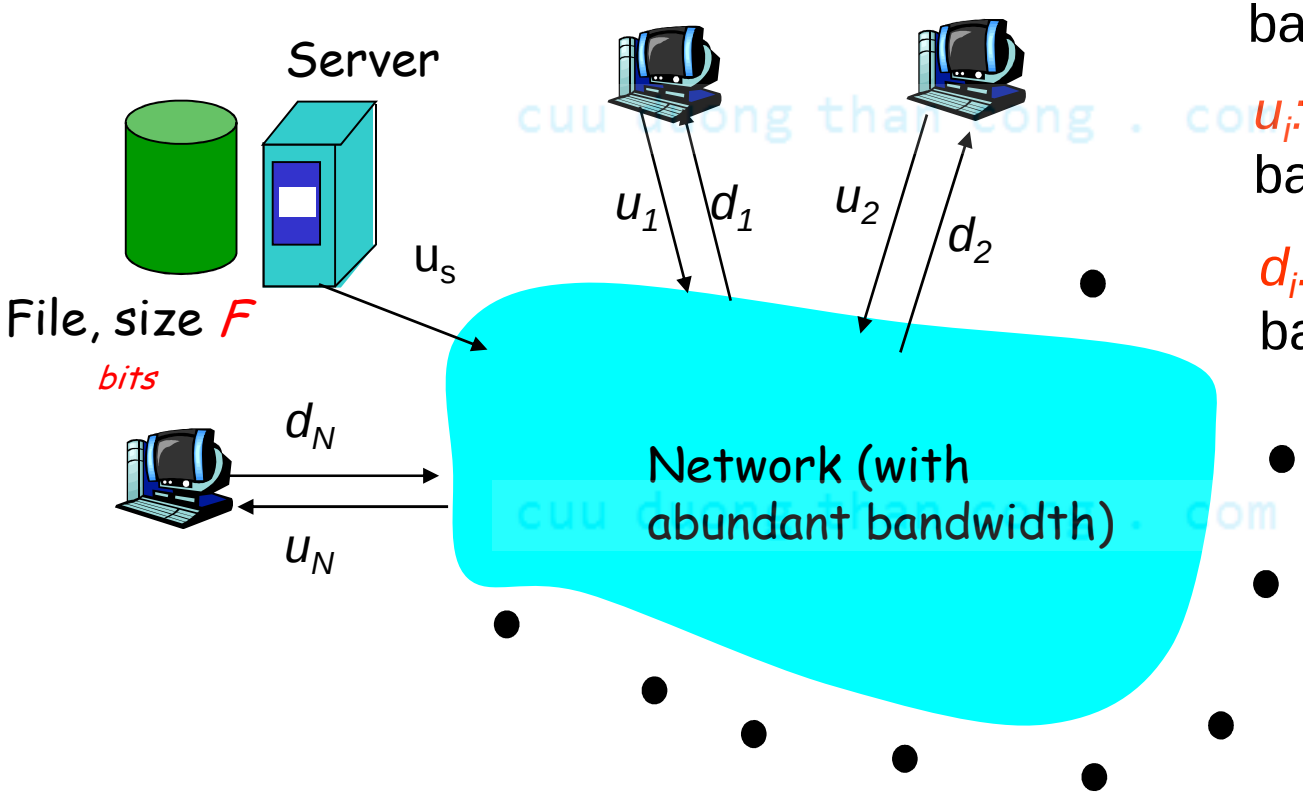
Pure P2P architecture

- Không có server chờ phục vụ.
- Các host truyền thông trực tiếp
- Các peers kết nối nhau không liên tục và thay đổi địa chỉ IP
- 3 trường hợp:
  - Phân phối file
  - Tìm thông tin
  - Case Study: Skype



File Distribution: Server-Client vs P2P

Câu hỏi : Truyền 1 file từ 1 server đến N peer mất bao lâu?



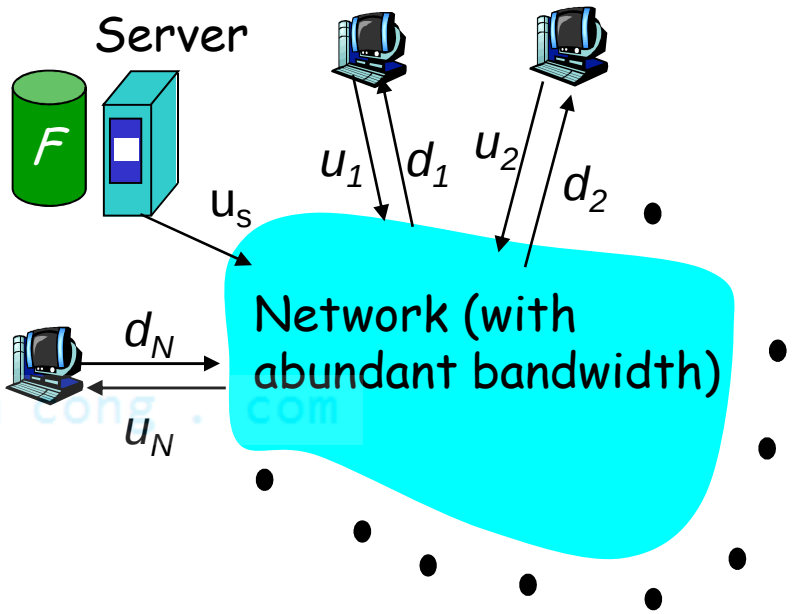
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: server-client

- Server lần lượt gửi N bản sao của file:
  - *hết* NF/u_s giây
- Client i tải 1 bản sao mất F/d_i giây



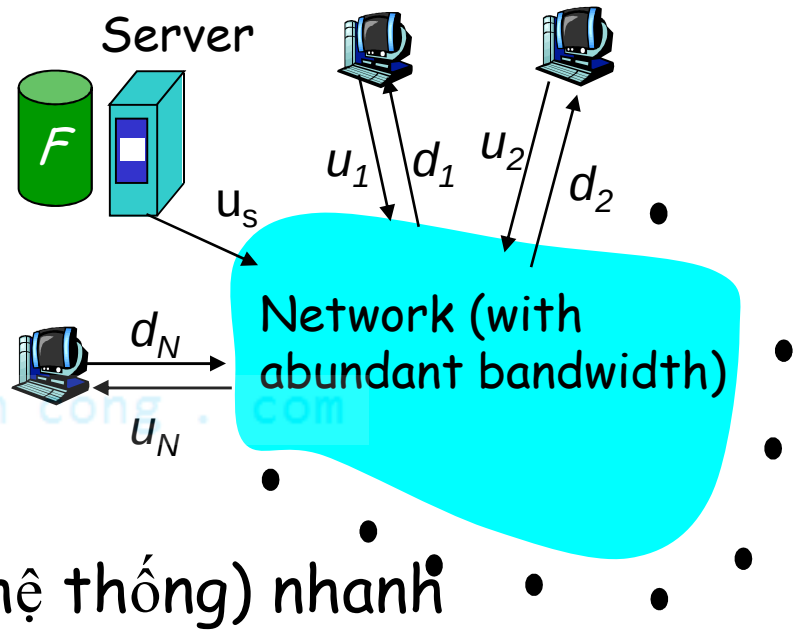
Thời gian để chuyển file F tới N clients sử dụng cách tiếp cận client/server

$$= d_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$$

Tăng tuyến tính theo N (với N đủ lớn)

File distribution time: P2P

- Server phải gửi 1 bản sao của file: hết F/u_s giây
- client i tải file hết F/d_i giây
- Tổng cộng có NF bits được tải

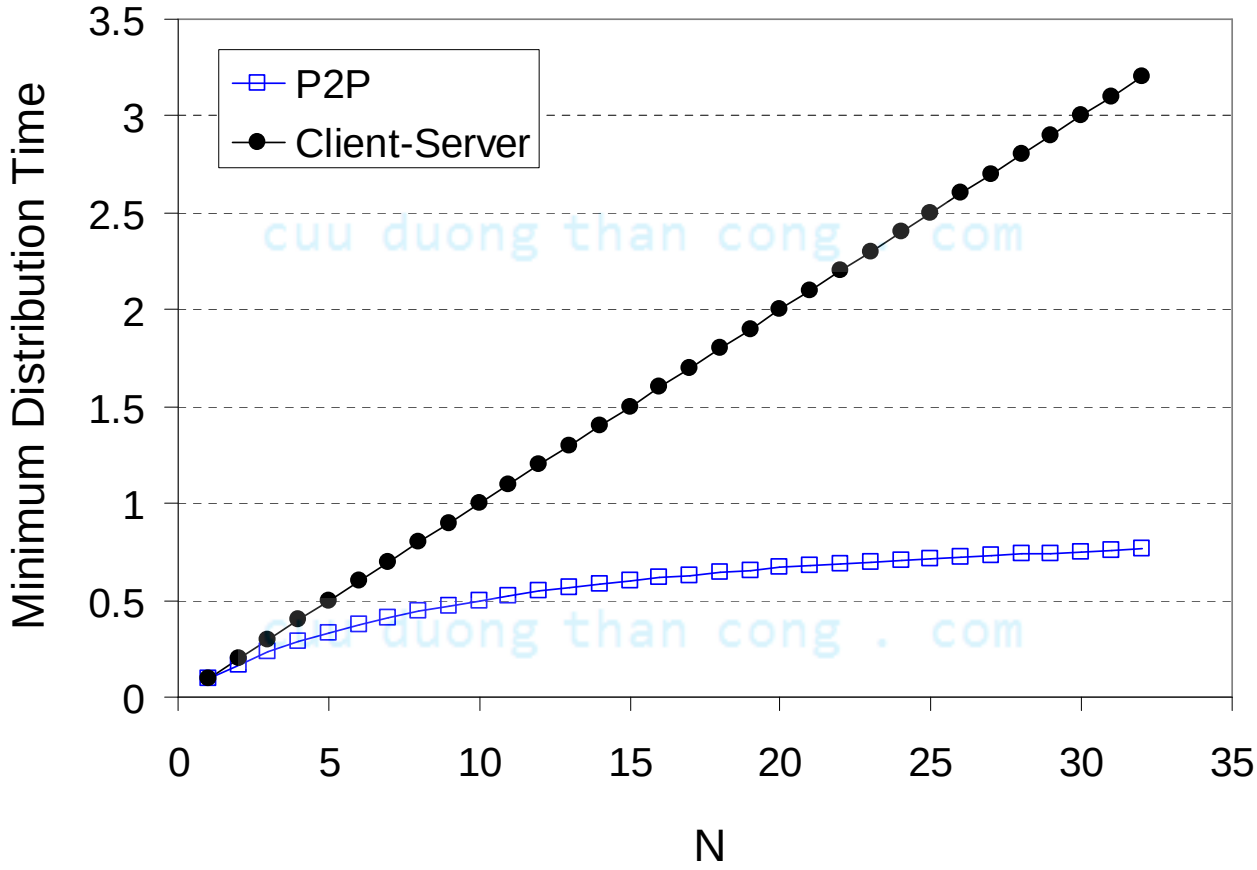


- Tốc độ upload (của toàn hệ thống) nhanh nhất có thể: $u_s + \Sigma u_i$

$$d_{P2P} = \max \{ F/u_s, F/\min(d_i)_i, NF/(u_s + \Sigma u_i) \}$$

Server-client vs. P2P: example

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

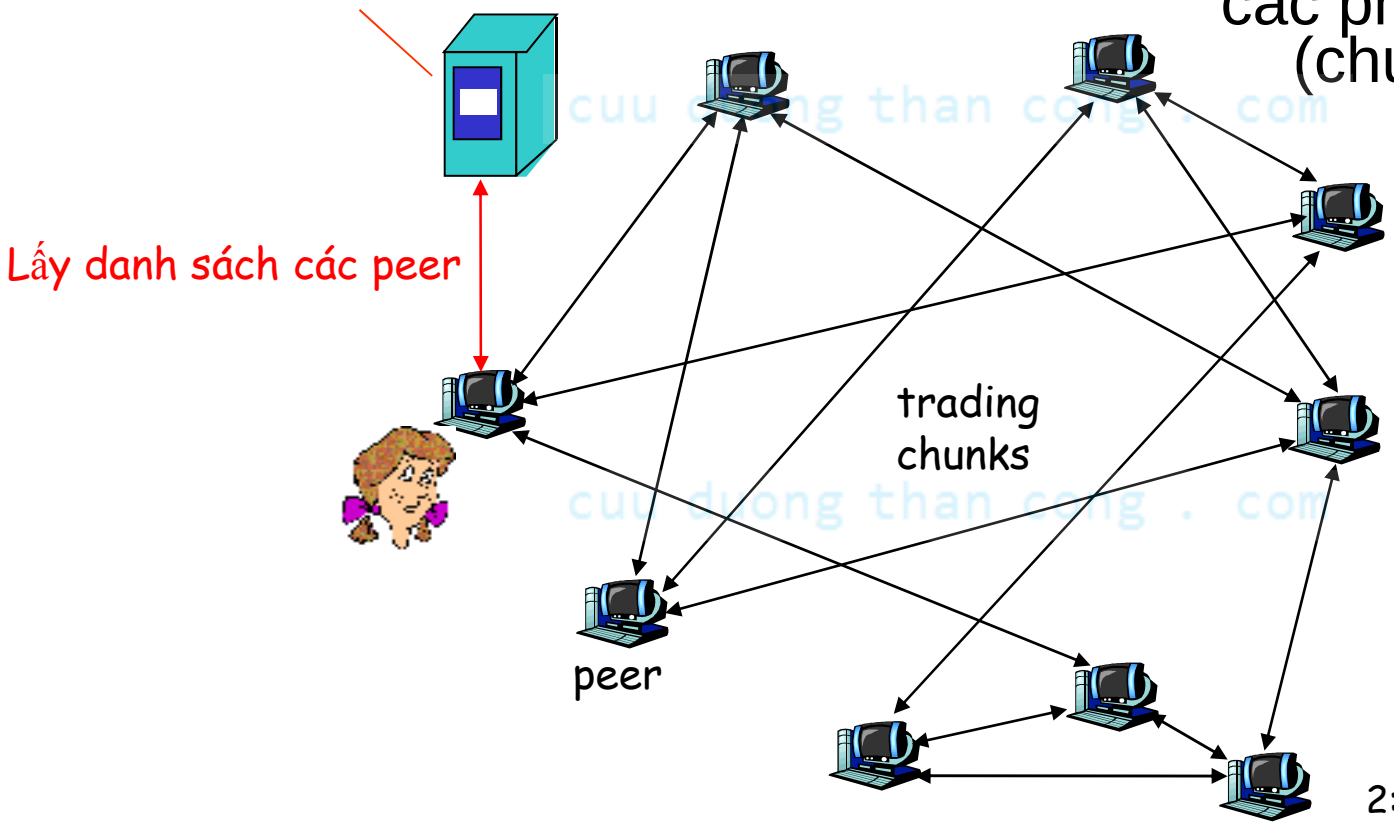


File distribution: BitTorrent

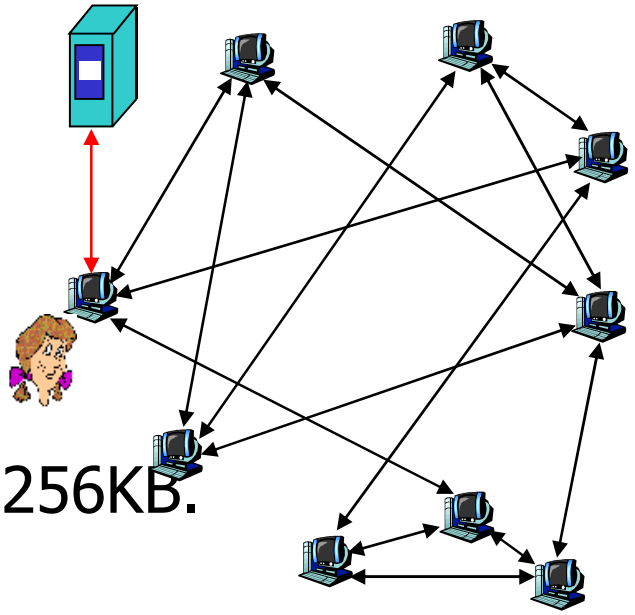
- P2P file distribution

tracker: theo dõi các peers đang tham gia vào torrent

torrent: nhóm các peers đang trao đổi các phân mãnh file (chunk of file)



BitTorrent (1)



- file được chia thành các mãnh (*chunks*)256KB.
- peer đang tham gia vào torrent:
  - chưa có mãnh nào, nhưng sẽ tích lũy các chunks trong suốt quá trình
  - ghi danh với tracker (để thu danh sách các peers), kết nối tới một tập con các peers ("láng giềng")
- Trong khi đang download file, peer cũng upload các chunks (mà nó có và được yêu cầu) cho các peer khác.
- peers có thể tham gia vào torrent và rút lui.
- Khi peer đã có cả file, nó có thể rời torrent (ích kỷ) hoặc ở lại torrent (vị tha)

BitTorrent (2)

Pulling Chunks

- Tại 1 thời điểm nhất định, các peer khác nhau đang có được các chunks khác nhau.
- Định kỳ, một peer (VD Alice) sẽ hỏi các láng giềng về danh sách các chunks mà chúng có.
- Peer này (Alice) gửi yêu cầu để xin các chunks mà nó còn thiếu.
  - Ưu tiên xin chunks nào mà ít láng giềng có nhất (rarest first)

Sending Chunks: tit-for-tat M1

- Alice gửi các chunks tới 4 láng giềng mà hiện thời chúng đang gửi cho Alice các chunks khác ở *mức độ cao nhất.*
  - Sau mỗi 10s, sẽ đánh giá lại top 4 này.
- Sau mỗi 30 giây: chọn ngẫu nhiên peer khác, gửi chunk cho peer này.
  - peer mới được chọn có thể tham gia vào top 4
  - "optimistically unchoke"

M1 Ăn miếng trả miếng

MVC, 5/1/2010

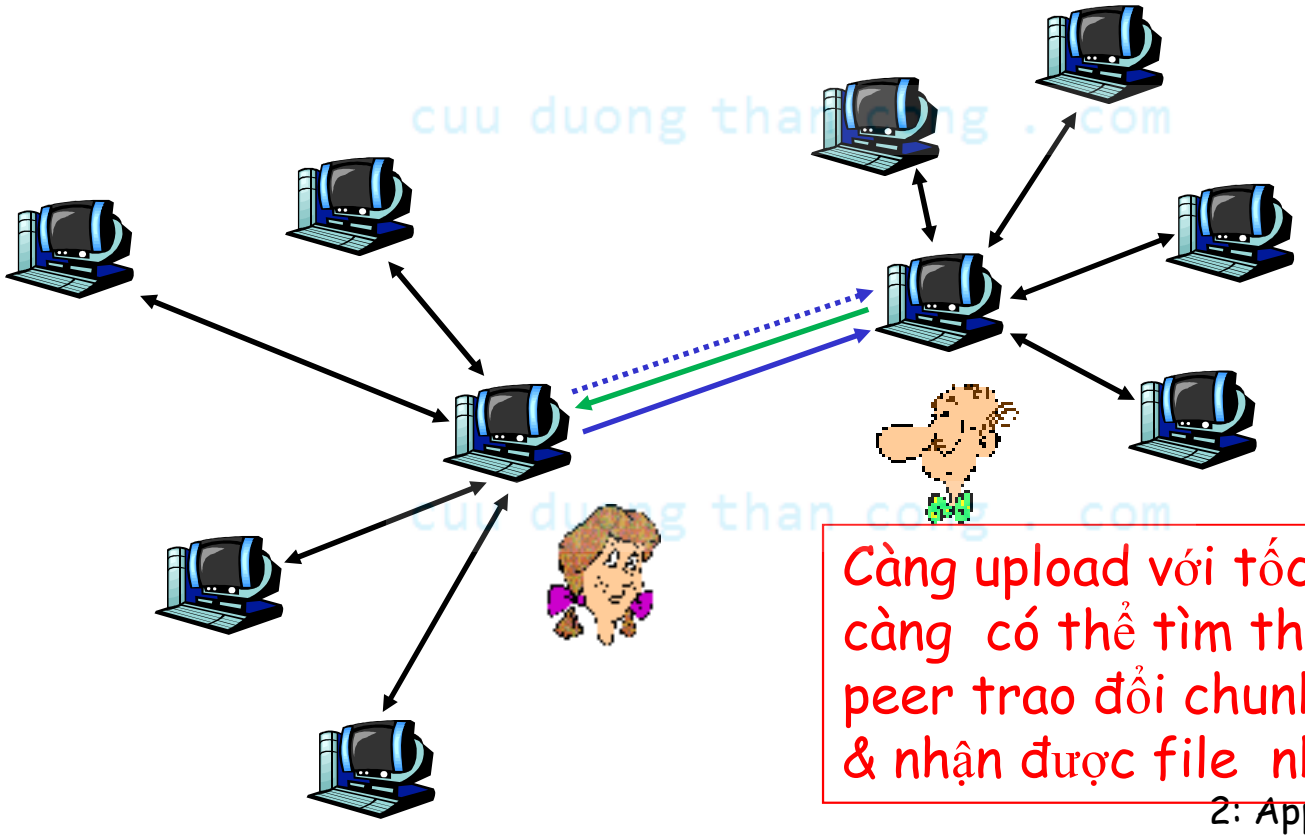
cuu duong than cong . com

BitTorrent: Tit-for-tat

(1) Alice chọn ngẫu nhiên được Bob và gửi chunks cho Bob

(2) Alice trở thành một trong 4 peer gửi chunks cho Bob nhiều nhất; Bob gửi chunks cho Alice đáp trả.

(3) Bob trở thành một trong 4 peer gửi chunk cho Alice nhiều nhất.



Càng upload với tốc độ cao, càng có thể tìm thấy các peer trao đổi chunks dễ hơn & nhận được file nhanh hơn.

Distributed Hash Table (DHT)

- DHT = CSDL phân tán trên môi trường P2P

[distributed P2P database]

- CSDL chứa các cặp (key, value);
  - key: số CMND; value: Tên người
  - key: tên nội dung; value: Địa chỉ IP (nơi lưu nội dung)
- Peers truy vấn CSDL bằng cách cung cấp trị key
  - CSDL trả về giá trị của value khớp với trị key
- Peers còn có thể chèn [insert] các cặp (key, value) mới vào CSDL.

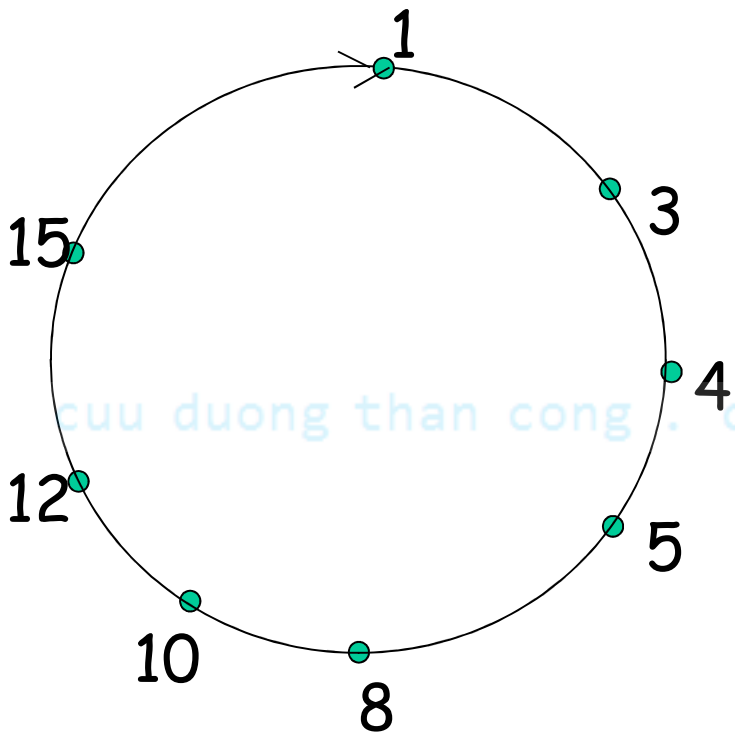
DHT Identifiers

- Mỗi peer được gán một số id để định danh, là số nguyên nằm trong vùng $[0,2^n-1]$ (n nguyên cố định)
  - Mỗi id có thể được biểu diễn qua chuỗi n bits.
- Đòi hỏi mỗi trị key là một số nguyên thuộc cùng vùng $[0,2^n-1]$
- Để thu được trị key nguyên như đòi hỏi trên, sử dụng hàm băm h, h nhận trị key gốc và trả về trị nguyên trong vùng $[0,2^n-1]$
  - eg, key = h("Led Zeppelin IV")
  - Đây là lý do tại sao ta gọi nó là DHT (distributed "hash" table)

Lưu cặp (key, value) ở peer nào?

- Vấn đề: Lưu trữ cặp (key, value) ở peer nào?
- Quy tắc: Lưu tại peer có id=key. Nếu không có peer như vậy thì lưu tại peer có id gần nhất.
- Quy ước trong bài giảng: id gần nhất là id nằm liền sau trong danh sách các id.
- VD: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, thì peer liền sau là 14
  - key = 15, thì peer liền sau là 1

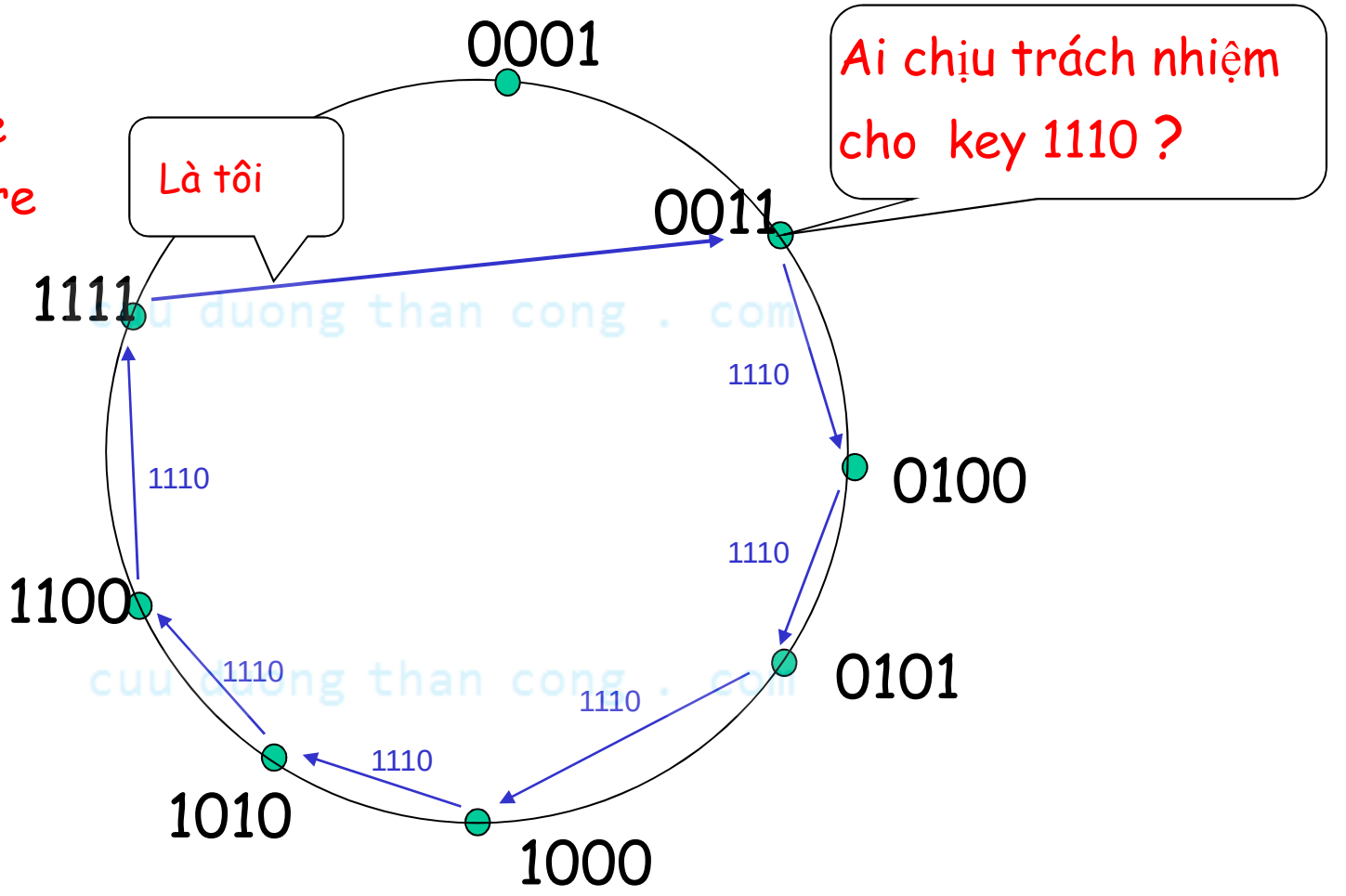
Circular DHT (1)



- Mỗi peer chỉ biết peer liền trước và liền sau.
- "Overlay network"

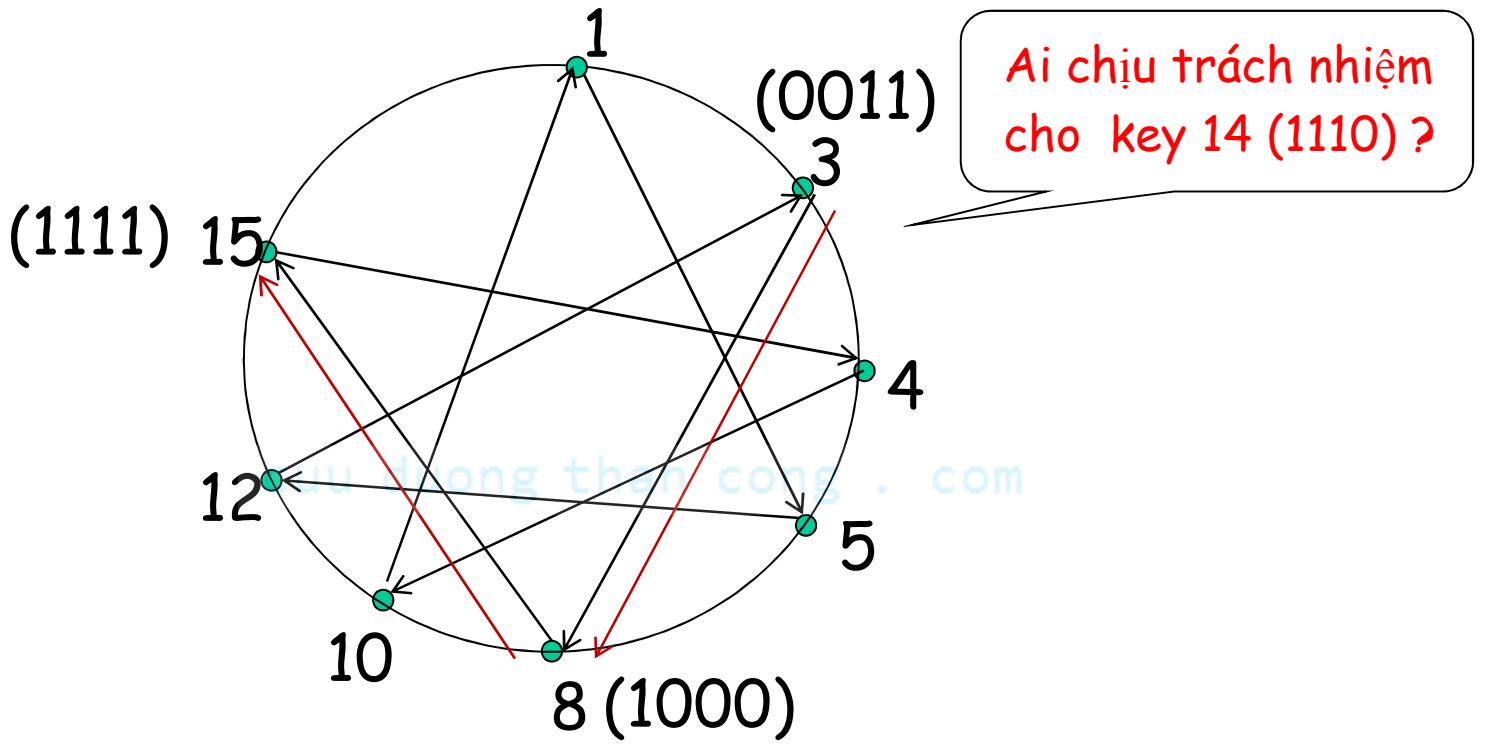
Circle DHT (2)

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



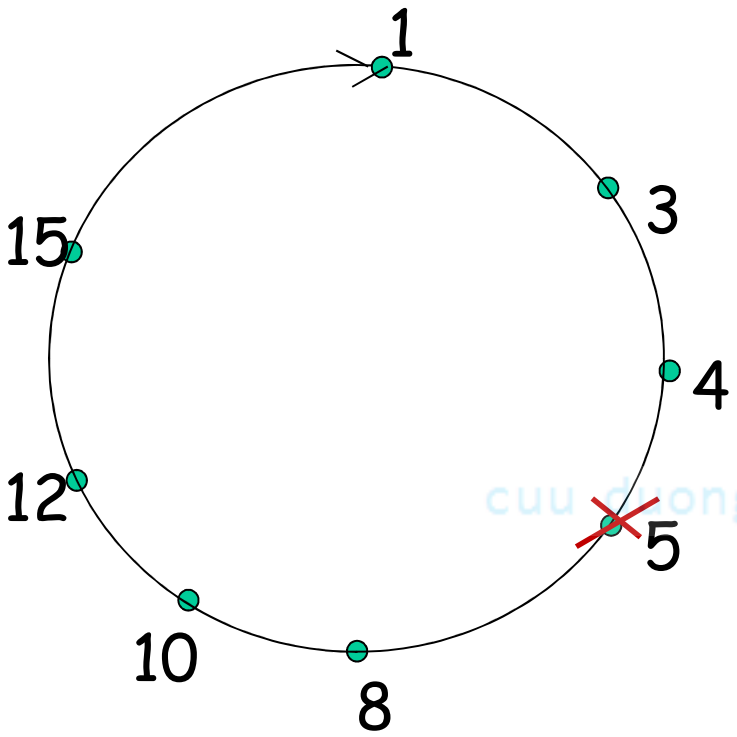
Define closest as closest successor

Circular DHT with Shortcuts



- Mỗi peer quản lý địa chỉ IP của peer liền trước, peer liền sau và các short cuts.
- Giảm từ 6 còn 2 messages.
- Có thể thiết kế shortcuts sao cho số láng giềng và số message khi truy vấn đều bằng O(log N)

Peer Churn

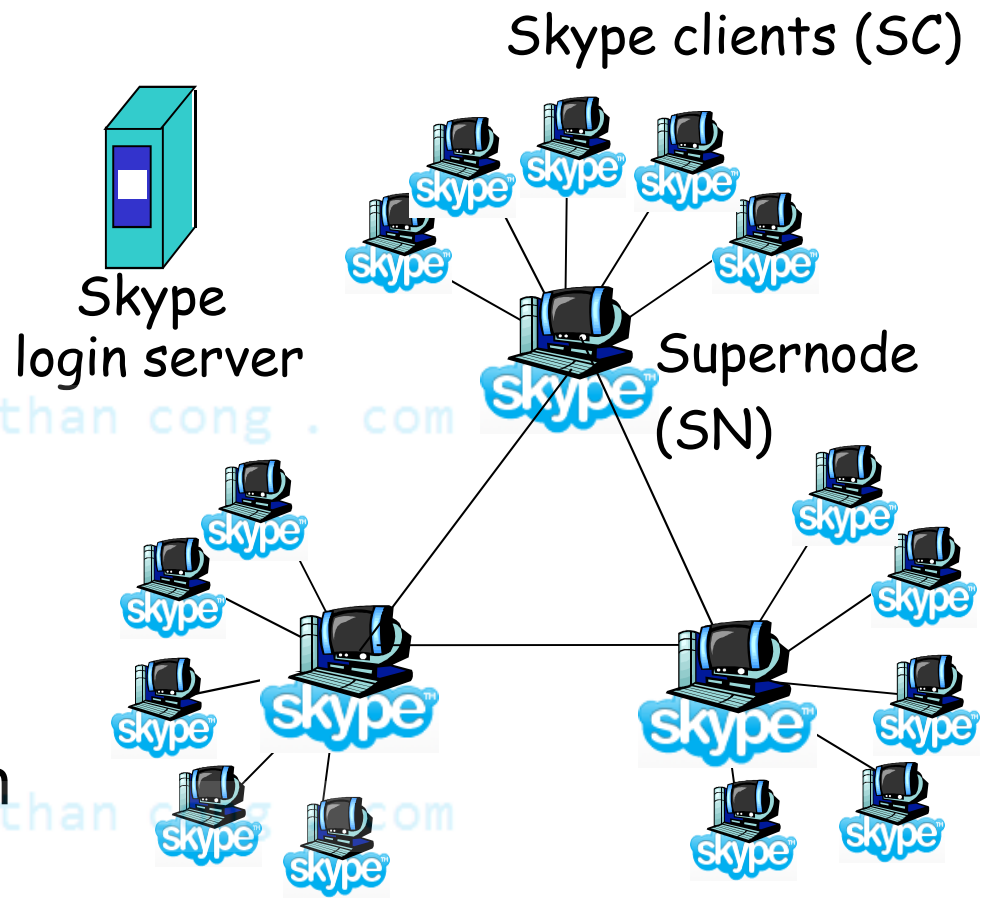


- Để xử lý trình trạng các peer *đến và đi mà không báo trước* (peer churn), mỗi peer cần phải biết các địa chỉ IP của 2 peer liền sau của nó.
- Mỗi peer sẽ ping định kỳ 2 peer liền sau của nó để xác định là chúng còn hiện hữu hay đã rút lui.

- Peer 5 đột ngột rời khỏi vòng.
- Peer 4 phát hiện ra điều này; nhận peer 8 làm peer liền sau (thứ nhất); và hỏi peer 8 "peer liền sau của 8?"; sau đó xem peer liền sau của 8 là peer kế liền sau (thứ hai)của nó.
- Nếu peer 13 muốn gia nhập vòng thì sao?

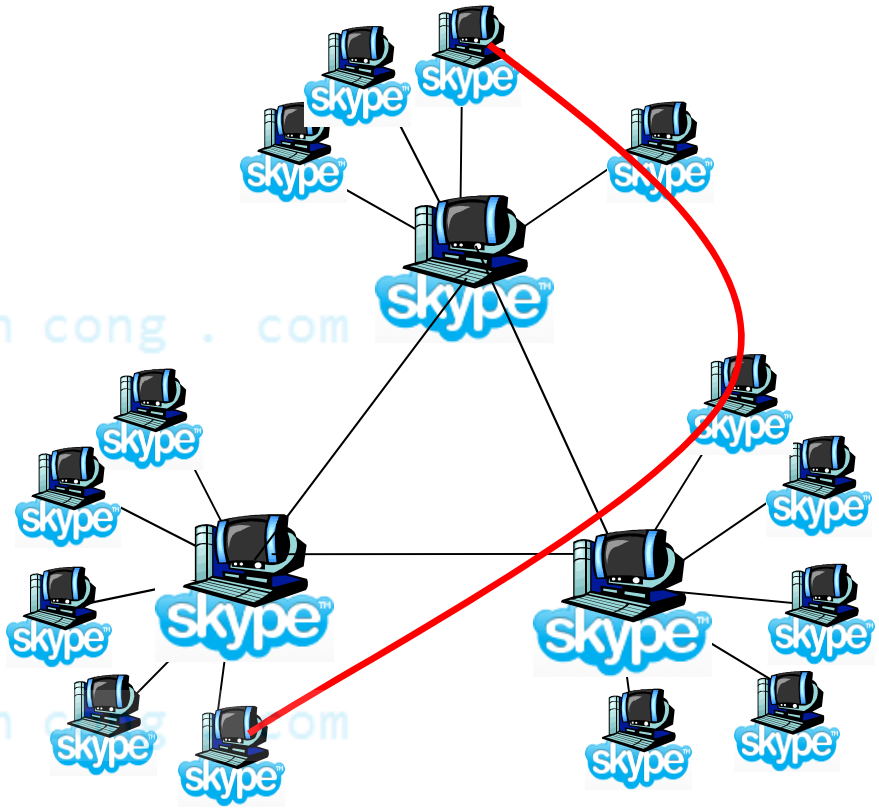
P2P Case study: Skype

- Bản chất P2P: các cặp user có thể truyền thông trực tiếp.
- Giao thức tầng ứng dụng sở hữu của Skype.
- Tổ chức phân cấp, sử dụng các Supernode
- Ánh xạ giữa username và địa chỉ IP, được lưu phân bố giữa các Supernode.



Peers as relays

- Vấn đề gặp phải khi cả 2 peer (VD Alice và Bob) đều nằm đằng sau "NAT"
  - NAT ngăn chặn các peer từ bên ngoài khởi tạo các kết nối (hay cuộc gọi) đến các peer bên trong
- Giải pháp:
  - Sử dụng các supernode của Alice và Bob, 1 supernode được chọn ra làm relay node (nút chuyển tiếp)
  - Từng peer sẽ khởi tạo kết nối với nút chuyển tiếp.
  - Do đó các peer có thể truyền thông cho nhau vượt qua NAT nhờ nút chuyển tiếp.



Chapter 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 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

Socket programming

Mục tiêu: học cách xây dựng ứng dụng client/server truyền thông với nhau sử dụng sockets

- Socket API
- Được giới thiệu trong BSD4.1 UNIX, 1981
- Được các ứng dụng tạo ra, sử dụng và giải phóng một cách tường minh.
- client/server paradigm
- Có 2 loại dịch vụ vận chuyển thông qua socket API
  - UDP
  - TCP

socket

A *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

Socket programming basics

- Server phải đang chạy trước khi client có thể gửi dữ liệu cho nó.
- Server phải có 1 socket (cửa) mà nhờ đó nó có thể nhận/ gửi đi các segment
- Tương tự, client cũng cần có 1 socket

- Socket có thể xác định trong nội bộ nhờ *số hiệu cổng* (port number)
  - Tương tự như số hiệu căn hộ trong 1 tòa nhà
- Client cần phải biết địa chỉ IP của Server và số hiệu cổng của socket.

Socket programming *with UDP*

UDP: không có "kết nối" giữa client và server

- Không bắt tay
- Bên gửi gắn địa chỉ IP và số hiệu cổng lên từng segment
- Server có thể trích xuất địa chỉ IP và số hiểu cổng của bên gửi từ segment nhận được.

Quan điểm ứng dụng

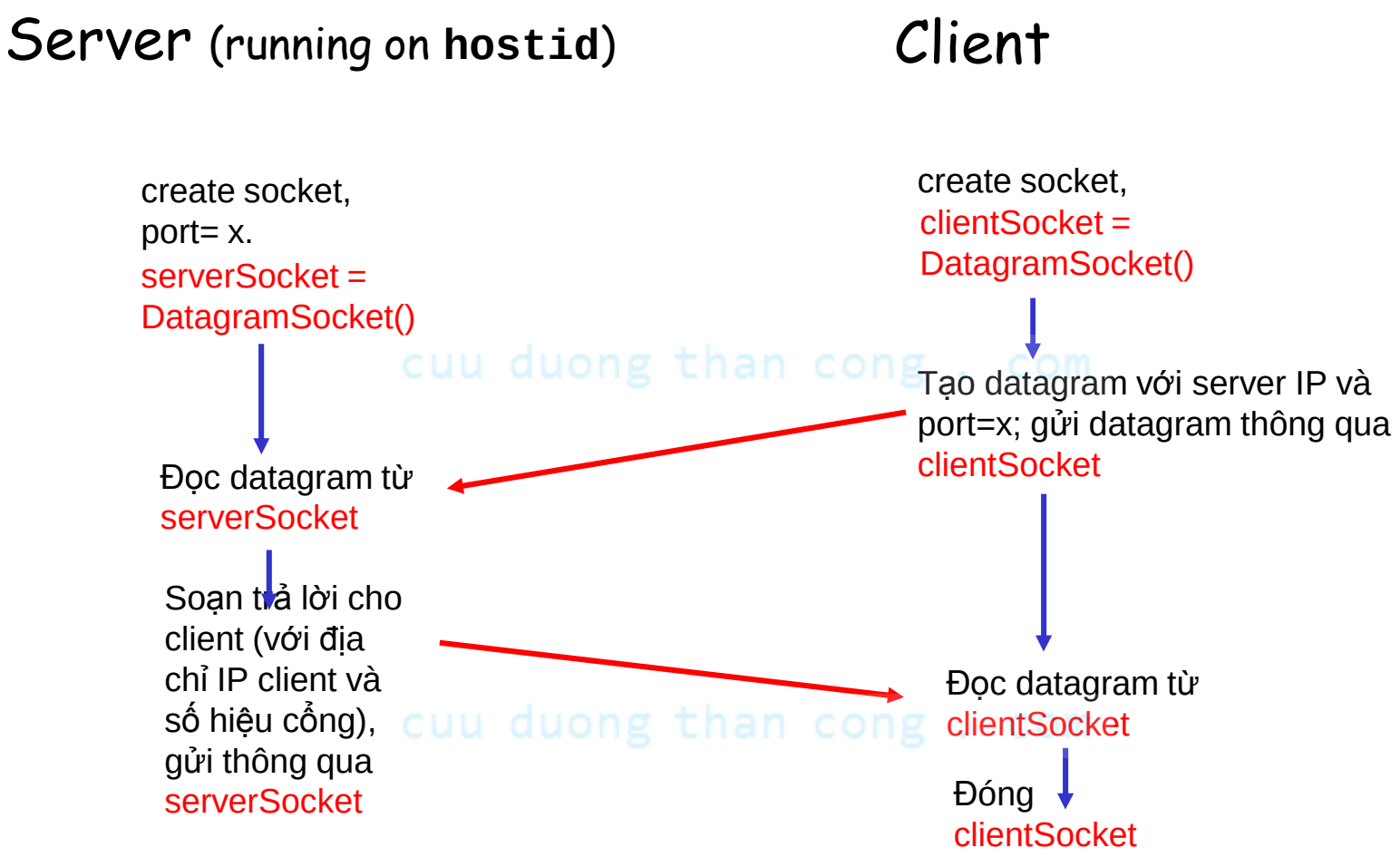
UDP cung cấp dịch vụ vận chuyển datagram "không đảm bảo tin cậy" giữa client and server

Lưu ý: Thuật ngữ chính thức gọi gói tin UDP là "datagram". Trong lớp này chúng ta sẽ gọi là "UDP segment"

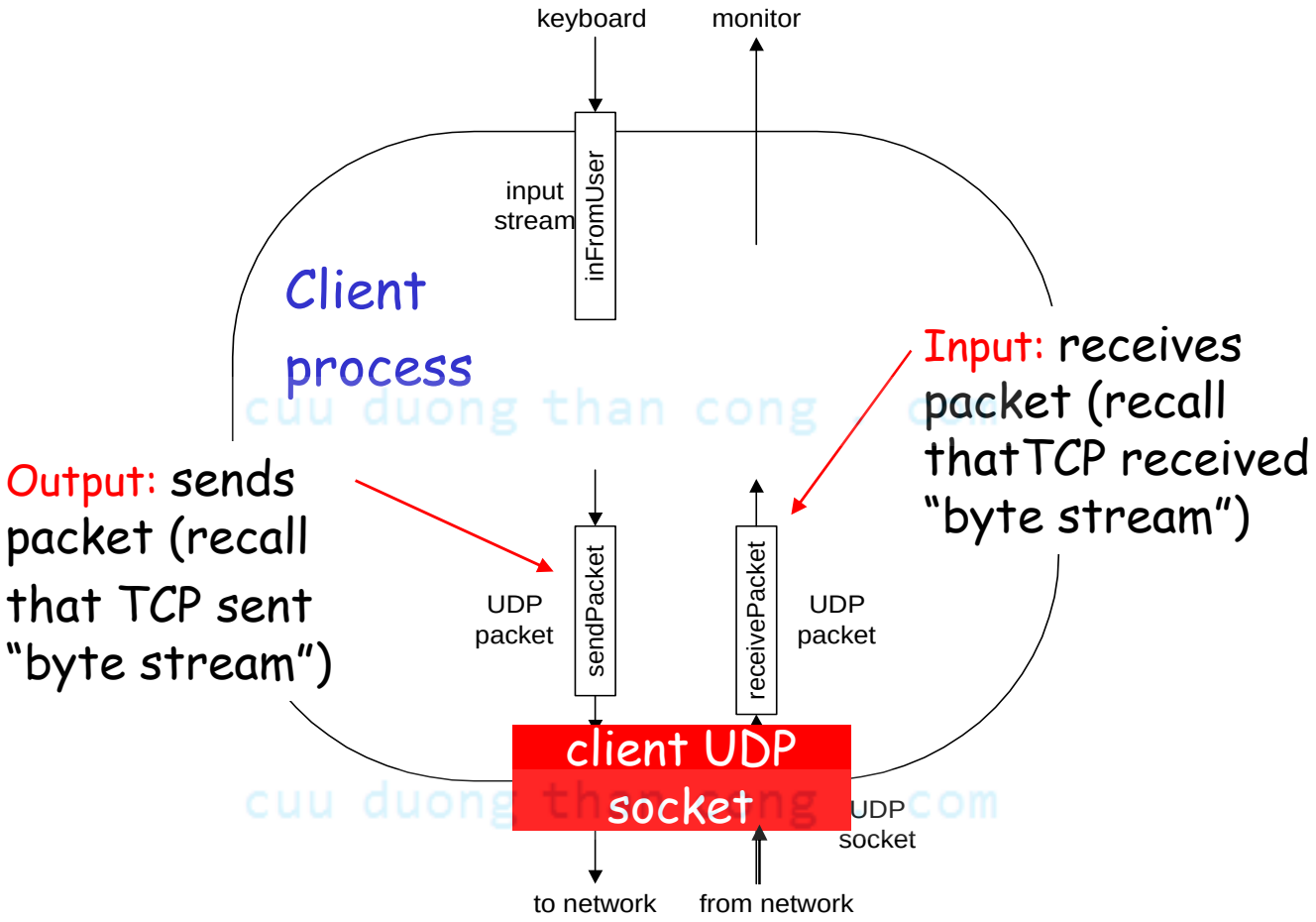
Running example

- Client:
  - Người sử dụng gõ vào một dòng văn bản
  - Chương trình client gửi dòng này đến server
- Server:
  - Server nhận được dòng văn bản
  - Server chuyển thành chữ hoa tất cả các ký tự
  - Gửi dòng văn bản đã được sửa đổi đến client
- Client:
  - Nhận dòng văn bản
  - Hiển thị

Client/server socket interaction: UDP



Example: Java client (UDP)



Example: Java client (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {
      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
   new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}

}
```

Example: Java server (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {
      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {
          DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
          serverSocket.receive(receivePacket);
```

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

Example: Java server (UDP), cont

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                      port);

serverSocket.send(sendPacket);
    }
  }
}
```

Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram

UDP observations & questions

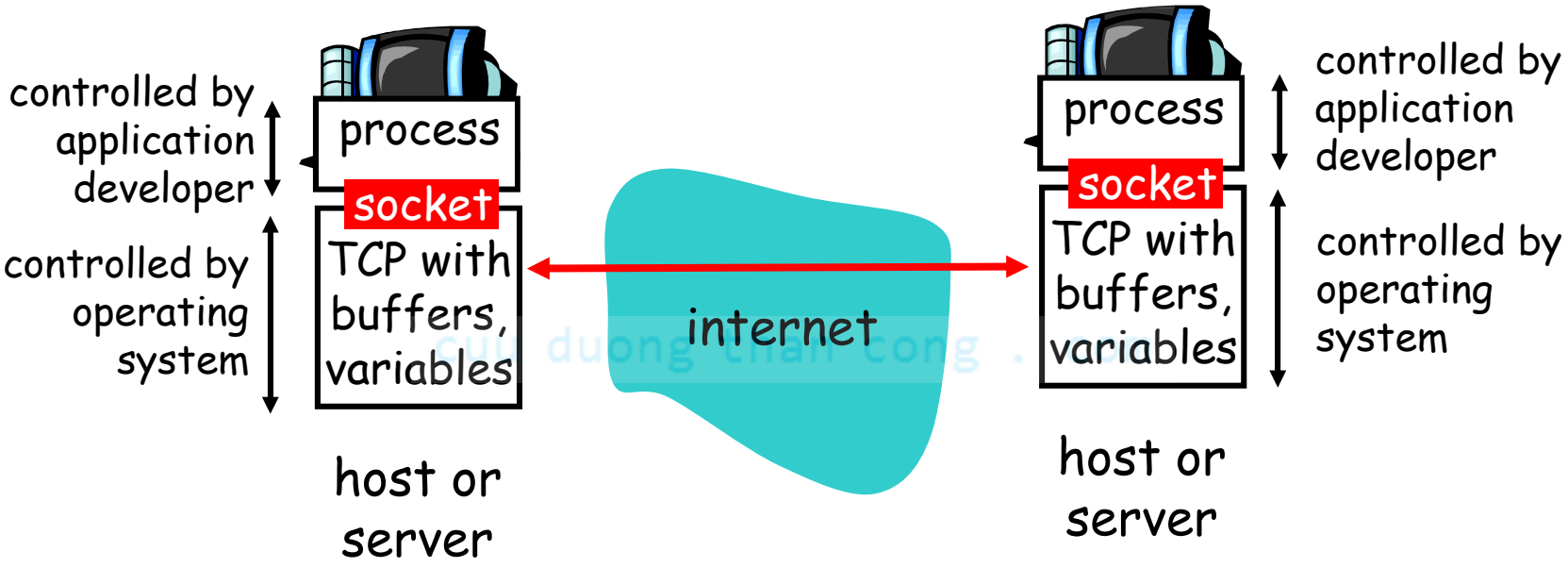
- Cả client và server đều dùng DatagramSocket
- Địa chỉ IP bên nhận và số hiệu cổng được gắn tường minh vào segment.
- Điều gì sẽ xảy ra nếu thay đổi cả hai clientSocket , serverSocket thành "mySocket"?
- Liệu client có thể gửi 1 segment đến 1 server mà không biết đến địa chỉ IP của server và/hoặc port number?
- Có thể nào nhiều client sử dụng server này?

Chapter 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 P2P applications
- 2.7 Socket programming with UDP
- 2.8 Socket programming with TCP

Socket-programming using TCP

TCP service: reliable transfer of **bytes** from one process to another



Socket programming *with TCP*

Client phải liên lạc với server

- Trước tiên, tiến trình server phải đang chạy.
- Server phải có socket được tạo sẵn, chờ đón các liên lạc từ phía client.

Client liên lạc với server bằng cách:

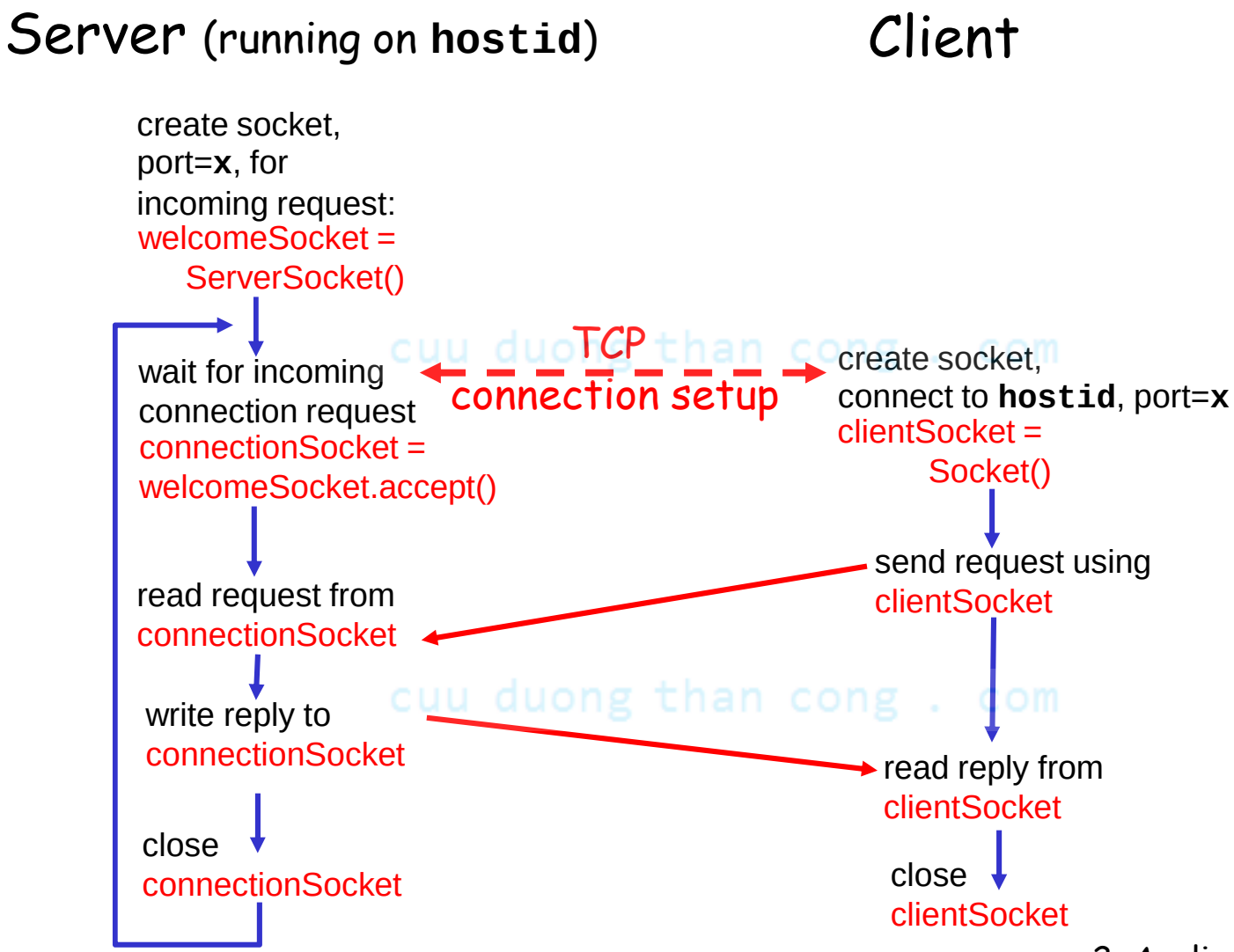
- Tạo client-local TCP socket
- Chỉ định địa chỉ IP, port number của server process
- Khi client tạo socket: client TCP thiết lập kết nới đến server TCP

- Khi được client liên lạc, server TCP tạo mới một socket để server process liên lạc với client
  - Cho phép server trao đổi với nhiều client
  - source port numbers được dùng để phân biệt client (nói rõ trong Ch. 3)

Quan điểm ứng dụng

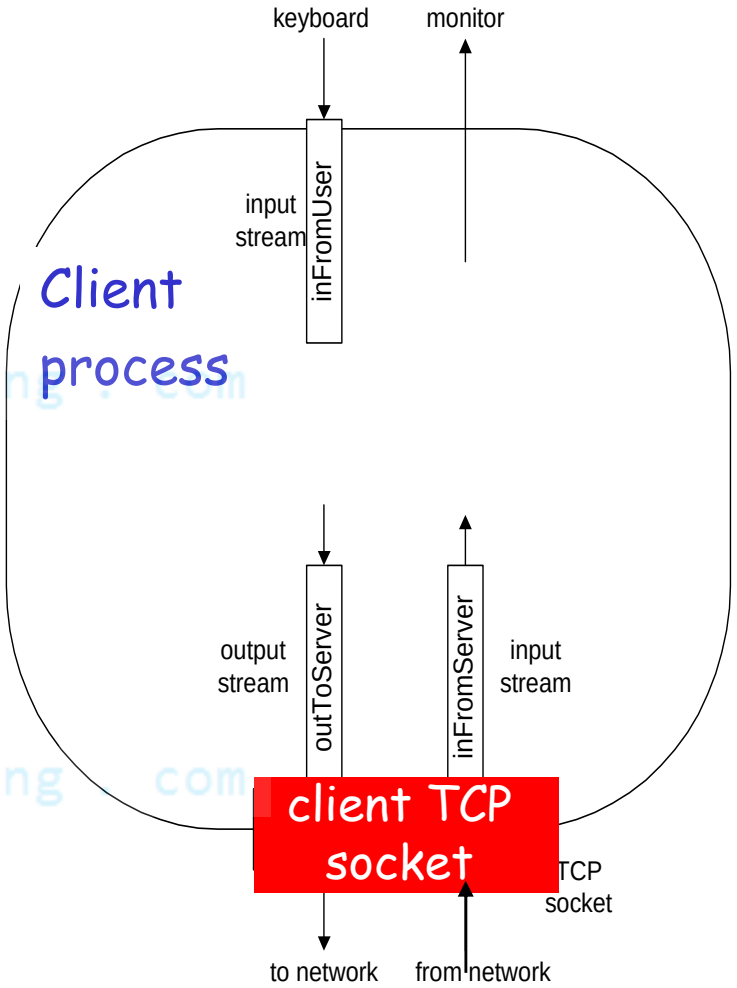
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Client/server socket interaction: TCP



Stream jargon

- Stream là dãy các ký tự đi vào hoặc ra khỏi một process
- Một input stream được liên kết với một vài nguồn nhập liệu dành cho process, VD bàn phím hoặc socket.
- Một output stream được liên kết với một đầu ra, VD màn hình hoặc socket.



Socket programming with TCP

VD về ứ/dụng client-server:

1) Client đọc dòng từ standard input (**`inFromUser`** stream), gửi đến server thông qua socket (**`outToServer`** stream)

2) Server đọc dòng từ socket

3) Server chuyển đổi ký tự trong dòng thành chữ hoa, gửi ngược lại client.

4) Client đọc, in ra dòng bị sửa đổi từ socket (**`inFromServer`** stream)

Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

  }
}
```

Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java server (TCP), cont

Create output stream, attached to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Read in line from socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Write out line to socket

```
outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

End of while loop, loop back and wait for another client connection

TCP observations & questions

- Server có 2 kiểu socket:
  - ServerSocket và Socket
- Khi client gõ vào cửa "serverSocket", server tạo ra connectionSocket và hoàn tất TCP conx.
- Địa chỉ IP đích và số hiệu cổng không được gắn vào segment tường minh.
- Liệu nhiều clients có thể sử dụng server?.
-

Chapter 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
- socket programming

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