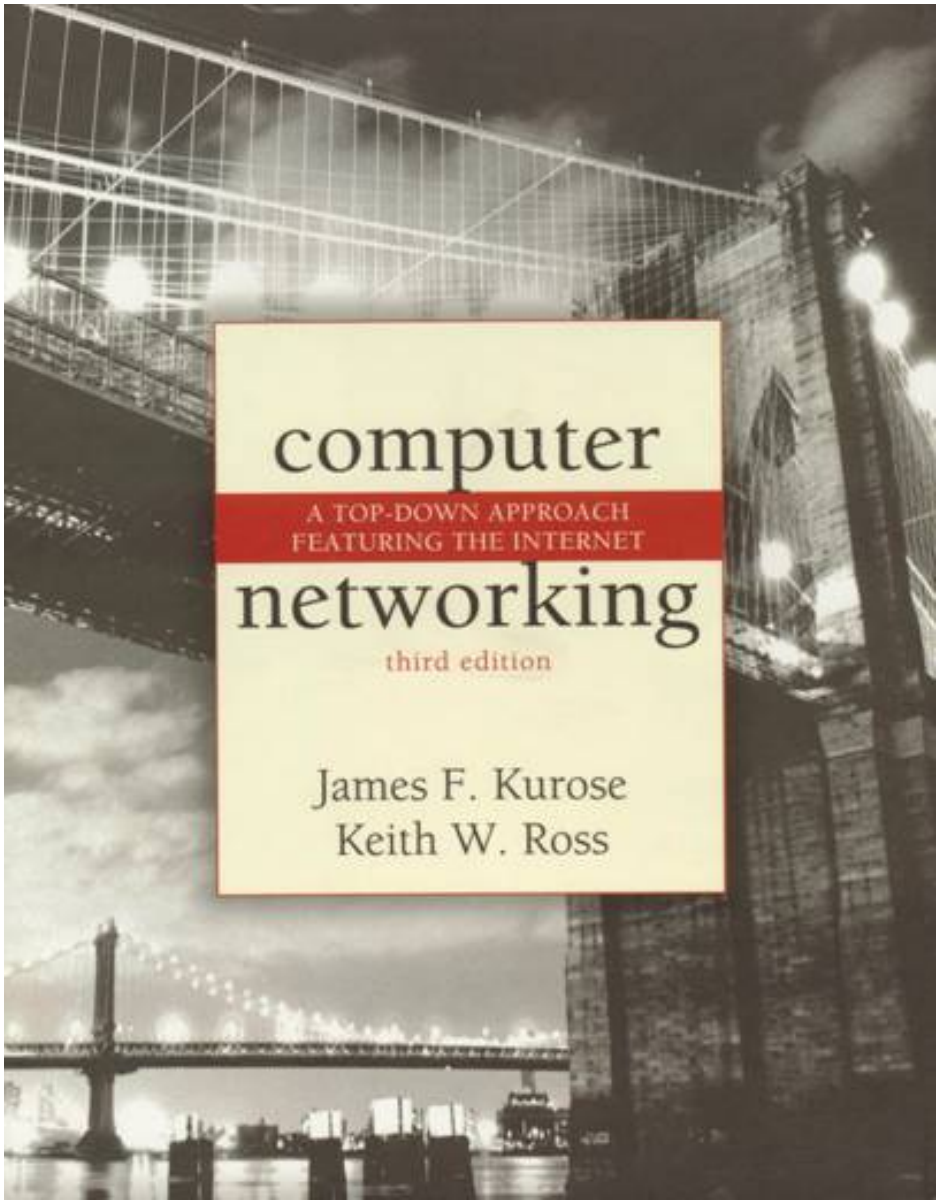


Chương 2
Lớp Application

Computer Networking: A Top Down Approach Featuring the Internet, 3rd edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2004.

Slide này được biên dịch sang tiếng Việt theo sự cho phép của các tác giả

Chương 2: Nội dung trình bày

- 2.1 Các nguyên lý của ứng dụng mạng
- 2.2 Web và HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Chia sẻ file P2P
- 2.7 Lập trình socket với TCP
- 2.8 Lập trình socket với UDP
- 2.9 Xây dựng một Web server

Chương 2: Lớp Application

Mục tiêu:

- Khái niệm, các khía cạnh hiện thực của các giao thức ứng dụng mạng
  - Các mô hình dịch vụ lớp transport
  - Mô hình client-server
  - Mô hình peer-to-peer
- Nghiên cứu giao thức thông qua xem xét một số giao thức lớp application
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- Lập trình ứng dụng mạng
  - socket API

Một số ứng dụng mạng

- E-mail
- Web
- Tin nhắn nhanh
- Đăng nhập từ xa
- Chia sẻ file P2P
- Trò chơi nhiều người trên mạng
- Streaming các video clips
- Điện thoại Internet
- Hội thảo video thời gian thực
- Tính toán lớn, tính toán song song

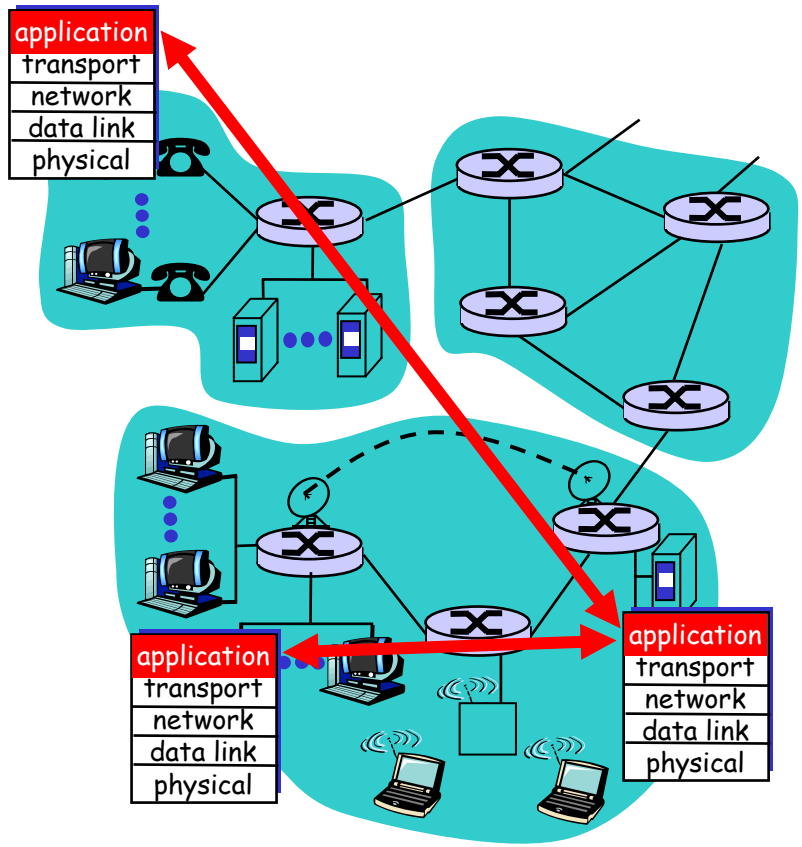
Tạo một ứng dụng mạng

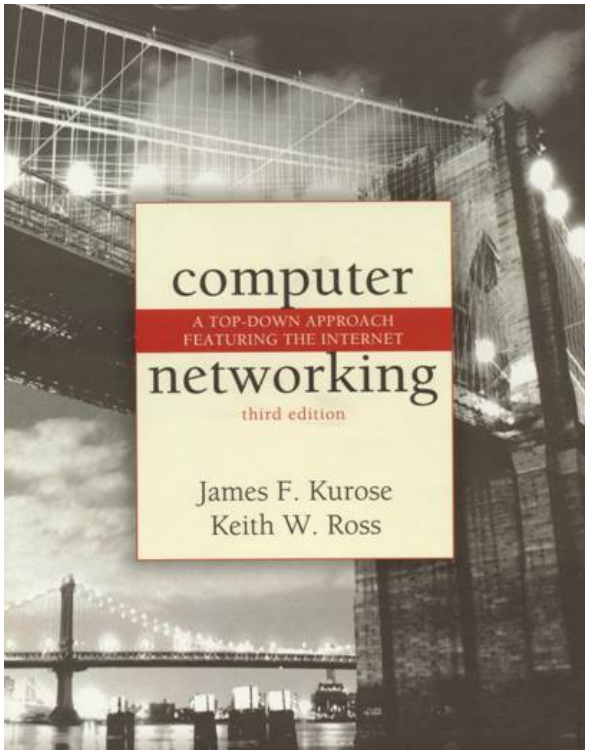
Viết chương trình để:

- chạy trên các hệ thống đầu cuối khác, và
- truyền thông qua mạng
- Ví dụ: Web: phần mềm Web server truyền thông với phần mềm trình duyệt

Phần mềm nhỏ viết cho các thiết bị trung tâm mạng

- các thiết bị trung tâm mạng không chạy các mã ứng dụng của người dùng
- ứng dụng trên các hệ thống đầu cuối cho phép phát triển ứng dụng nhanh, phổ biến



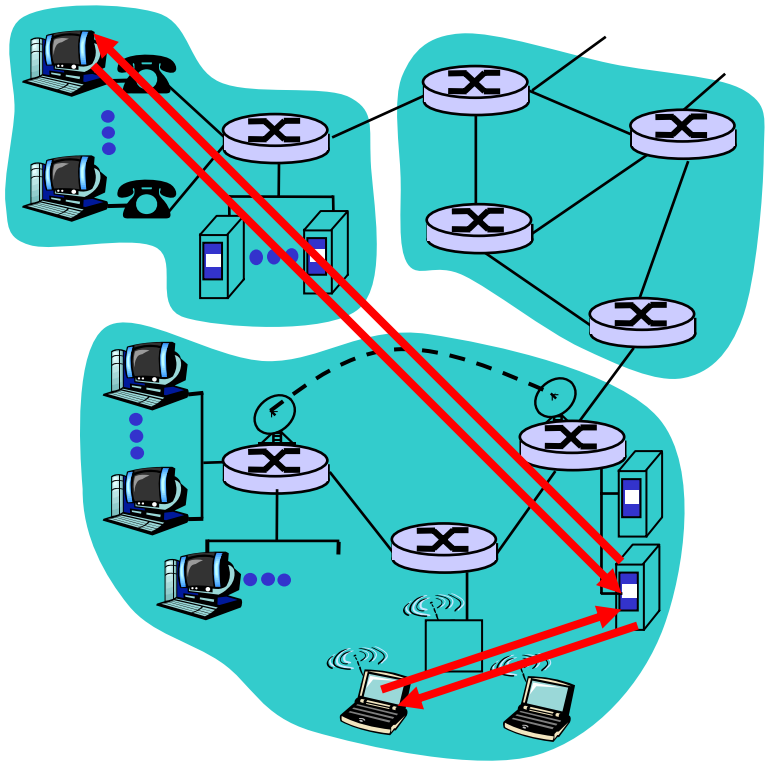


2.1 Các nguyên lý của ứng dụng mạng

Các kiến trúc của ứng dụng

- Client-server
- Peer-to-peer (P2P)
- Lai giữa client-server và P2P

Kiến trúc client-server



server:

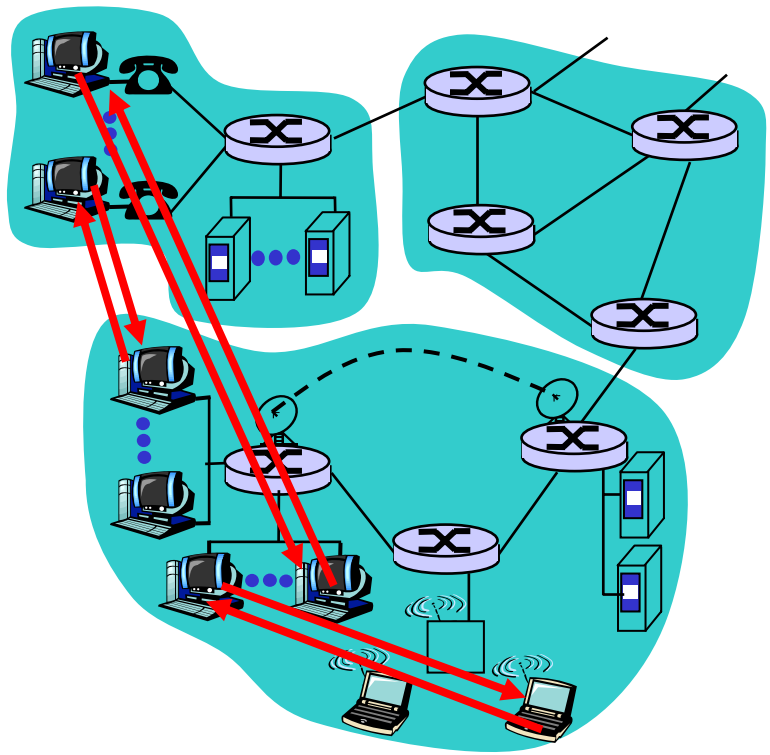
- host luôn hoạt động
- địa chỉ IP cố định
- nhóm các server để chia sẻ công việc

clients:

- truyền thông với server
- có thể kết nối không liên tục
- có thể có địa chỉ IP thay đổi
- không truyền thông trực tiếp với client khác

Kiến trúc P2P thuần túy

- không có server luôn hoạt động
- truyền thông trực tiếp với hệ thống đầu cuối bất kỳ
- các điểm kết nối không liên tục và thay đổi địa chỉ IP
- Ví dụ: Gnutella



Độ linh hoạt cao nhưng khó quản lý

Lai giữa client-server và P2P

Skype

- ứng dụng điện thoại Internet
- Tìm địa chỉ của thành viên ở xa: server trung tâm
- Kết nối trực tiếp Client-client (không thông qua server)

Tin nhắn nhanh

- Chat giữa 2 user là P2P
- Mô hình client-server:
  - User đăng ký địa chỉ IP của họ với server trung tâm khi trực tuyến
  - User tiếp xúc với server trung tâm để tìm địa chỉ IP của bạn

Tiến trình truyền thông

Tiến trình: chương trình chạy bên trong 1 host.

- trong cùng host, 2 tiến trình truyền thông dùng truyền thông nội bộ (do hệ điều hành xác định).
- các tiến trình trong các host khác nhau truyền thông bằng cách trao đổi các thông điệp

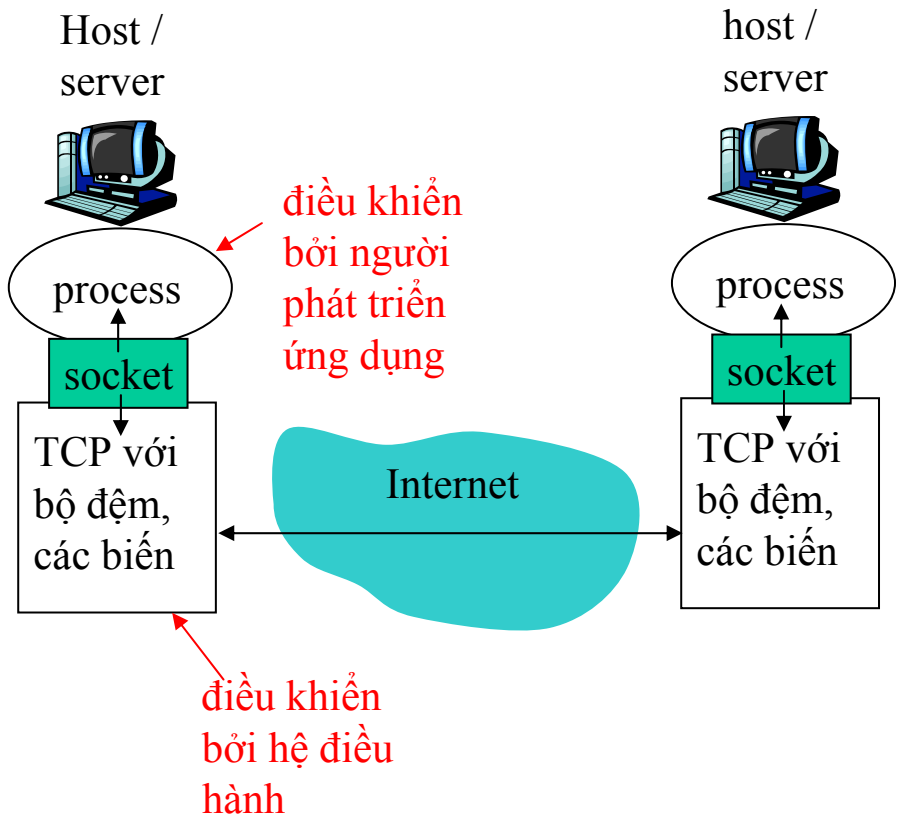
Tiến trình Client: tiến trình khởi tạo truyền thông

Tiến trình Server: tiến trình chờ để được tiếp xúc

- Chú ý: các ứng dụng với kiến trúc P2P có cả các tiến trình client và server.

Sockets

- các tiến trình gửi/nhận các thông điệp đến/từ socket của nó
- socket tương tự như cửa
  - tiến trình gửi đẩy thông điệp ra ngoài cửa
  - tiến trình nhận phụ thuộc vào hạ tầng lưu thông mang thông điệp đến socket thích hợp



- API: (1) lựa chọn giao thức vận chuyển; (2) khả năng chỉnh sửa một vài tham số (xem phần sau)

Tiến trình định địa chỉ

- để nhận được thông điệp, tiến trình phải có *nhân dạng (identifier)*
- thiết bị host phải có địa chỉ IP duy nhất
- Địa chỉ IP mà trên đó tiến trình đang chạy có đủ để nhận dạng tiến trình?
  - KHÔNG, nhiều tiến trình có thể chạy trên cùng 1 host

- *Nhân dạng* bao gồm cả địa chỉ IP và các số cổng (port) liên kết với tiến trình trên host.
- Ví dụ về số port:
  - HTTP server: 80
  - Mail server: 25
- Để gửi thông điệp HTTP cho web server gaia.cs.umass.edu :
  - IP address: 128.119.245.12
  - Port number: 80

Định nghĩa giao thức lớp ứng dụng

- các kiểu của trao đổi thông điệp
  - Ví dụ: yêu cầu, đáp ứng
- Cú pháp thông điệp:
  - Các trường nào trong thông điệp và làm sao mô tả?
- Ngữ nghĩa thông điệp
  - Ý nghĩa của thông tin trong các trường
- Các quy tắc để khi nào và làm sao các tiến trình gửi và đáp ứng các thông điệp

Các giao thức Public-domain:

- Định nghĩa trong RFC
- Cho phép cộng tác
- Ví dụ: HTTP, SMTP

Các giao thức độc quyền:

- Ví dụ: KaZaA

Dịch vụ vận chuyển nào ứng dụng không cần

Mất mát dữ liệu

- một số ứng dụng (vd: audio) có khả năng chịu lỗi
- các ứng dụng khác (vd: truyền file, telnet) yêu cầu dữ liệu tin cậy 100%

Định thì

- một số ứng dụng (vd: điện thoại Internet, trò chơi tương tác) yêu cầu độ trễ thấp để đạt hiệu quả

Bandwidth (băng thông)

- một số ứng dụng (vd: đa phương tiện) yêu cầu băng thông để đạt hiệu quả
- các ứng dụng khác mềm dẻo hơn có thể dùng bất kỳ băng thông nào cũng được

Một số yêu cầu đối với các ứng dụng phổ biến

Application	Data loss	Bandwidth	Time Sensitive
Truyền file	không	mềm dẻo	không
e-mail	không	mềm dẻo	không
Web	Không	mềm dẻo	không
audio/video thời gian thực	chịu lỗi	audio: 5kbps-1Mbps video:10kbps-5Mbps	có, 100 mili giây
audio/video đã lưu	chịu lỗi	Như trên	có, một vài giây
Trò chơi tương tác	chịu lỗi	Một vài kbps	có, 100 mili giây
Tin nhắn nhanh	không	mềm dẻo	Có và không

Các dịch vụ giao thức Internet transport

TCP:

- *connection-oriented:* cần thiết lập tiến trình giữa client và server
- *Vận chuyển tin cậy:* giữa tiến trình gửi và nhận
- *Điều khiển luồng:* người gửi sẽ không lấn át người nhận
- *Điều khiển tắc nghẽn:* điều tiết người gửi khi mạng quá tải
- *Không hỗ trợ:* định thì, bảo đảm băng thông tối thiểu

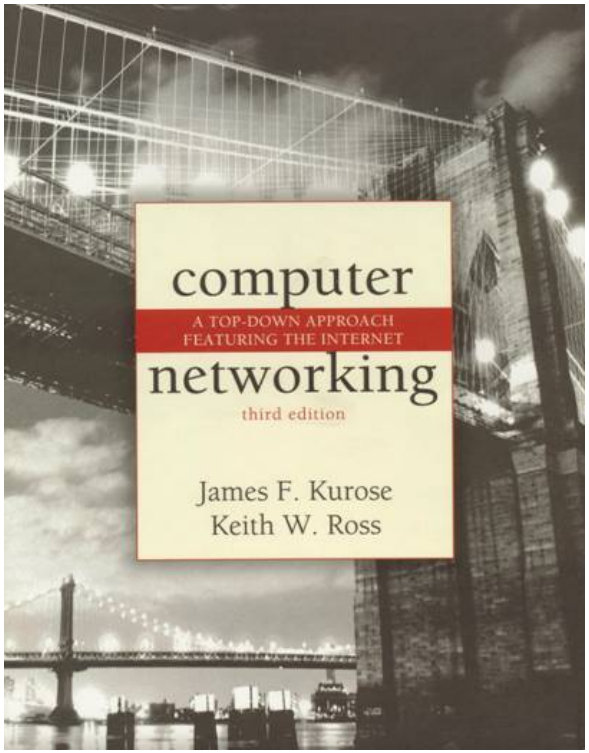
UDP:

- truyền dữ liệu không tin cậy giữa gửi và nhận
- Không hỗ trợ: thiết lập kết nối, tin cậy, điều khiển luồng, điều khiển tắc nghẽn, định thì, bảo đảm băng thông tối thiểu

Thế thì sinh ra UDP để làm gì?

Các giao thức lớp application, transport

Application	Giao thức lớp Application	Giao thức dưới lớp transport
e-mail	SMTP [RFC 2821] Telnet [RFC 854]	TCP
Truy cập terminal từ xa	HTTP [RFC 2616]	TCP
Web	FTP [RFC 959]	TCP
Truyền file	độc quyền	TCP
streaming multimedia	(vd: RealNetworks) độc quyền	TCP / UDP
Điện thoại Internet	(vd: Vonage,Dialpad)	UDP



2.2 Web và HTTP

Web và HTTP

Một số thuật ngữ chuyên môn

- Web page (trang Web) bao gồm các objects (đối tượng)
- Đối tượng có thể là file HTML, hình ảnh JPEG image, Java applet, file audio,...
- Trang Web file HTML cơ bản sẽ chứa một số đối tượng có tham chiếu
- Mỗi đối tượng có thể định địa chỉ bằng một URL
- Ví dụ URL:

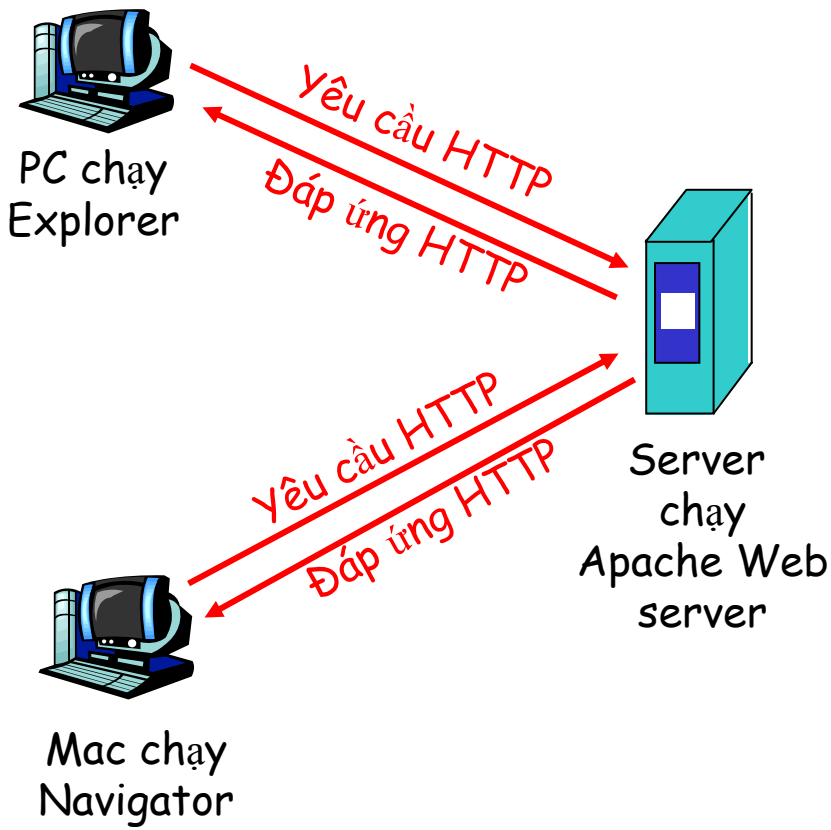
`www.someschool.edu/someDept/pic.gif`

Tên host | Tên đường dẫn

Tổng quan HTTP

HTTP: hypertext transfer protocol

- Giao thức lớp ứng dụng của Web
- Mô hình client/server
  - *Client:* trình duyệt yêu cầu, nhận và hiển thị các đối tượng Web
  - *Server:* Web server gửi các đối tượng đáp ứng cho yêu cầu
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



Tổng quan HTTP

Dùng TCP:

- client khởi tạo kết nối TCP (tạo socket) đến server, port 80
- server chấp nhận kết nối TCP từ client
- Các thông điệp HTTP (thông điệp giao thức lớp application) trao đổi giữa trình duyệt (HTTP client) và Web server (HTTP server)
- Đóng kết nối TCP

HTTP là "không trạng thái"

- server không giữ thông tin về các yêu cầu trước đó của client

vấn đề liên quan

Các giao thức nào giữ "trạng thái" là phức tạp!

- lịch sử quá khứ (trạng thái) phải giữ lại
- nếu server/client bị sự cố, cách nhìn của nó về "trạng thái" mâu thuẫn, phải được điều chỉnh

Các kết nối HTTP

HTTP không bền vững

- Chỉ có tối đa là 1 đối tượng được gửi qua 1 kết nối TCP.
- HTTP/1.0 dùng HTTP không bền vững

HTTP bền vững

- Nhiều đối tượng có thể được gửi qua 1 kết nối TCP đơn giữa client và server.
- HTTP/1.1 mặc nhiên dùng HTTP bền vững

HTTP không bền vững

Giả sử user nhập vào URL như sau:

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

(chứa text, tham chiếu đến 10 hình)

1a. HTTP client khởi tạo kết nối TCP connection đến HTTP server (tiến trình) tại www.someSchool.edu trên port 80

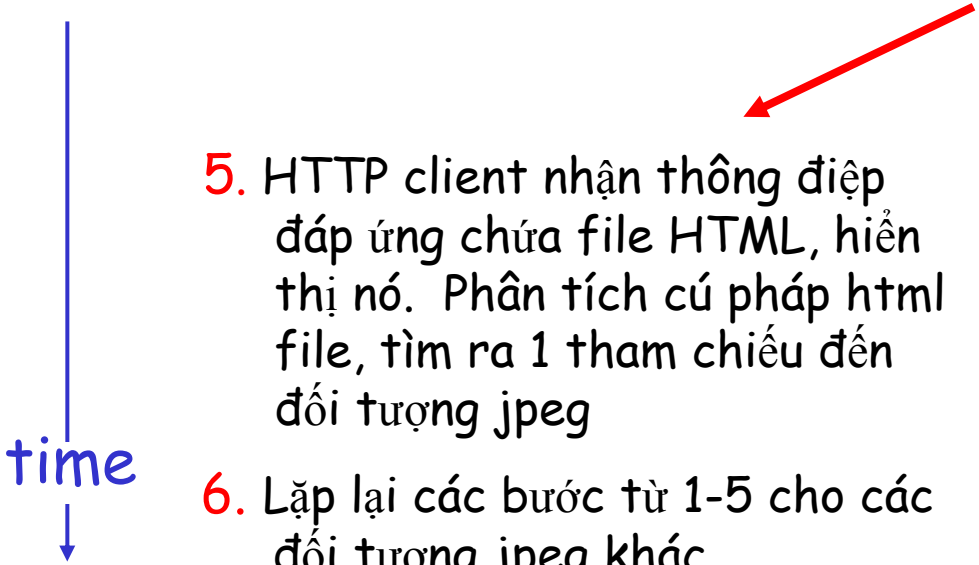
1b. HTTP server tại host www.someSchool.edu chờ kết nối TCP tại port 80. "chấp nhận" kết nối, thông báo cho client

2. HTTP client gửi HTTP *thông điệp yêu cầu* (chứa URL) vào trong socket kết nối TCP. Thông điệp chỉ rằng client muốn các đối tượng someDepartment/home.index

3. HTTP server nhận thông điệp yêu cầu, định dạng *thông điệp đáp ứng* chứa đối tượng được yêu cầu và gửi thông điệp vào trong socket của nó

Thời gian

HTTP không bền vững



4. HTTP server đóng kết nối TCP.

5. HTTP client nhận thông điệp đáp ứng chứa file HTML, hiển thị nó. Phân tích cú pháp html file, tìm ra 1 tham chiếu đến đối tượng jpeg

6. Lặp lại các bước từ 1-5 cho các đối tượng jpeg khác

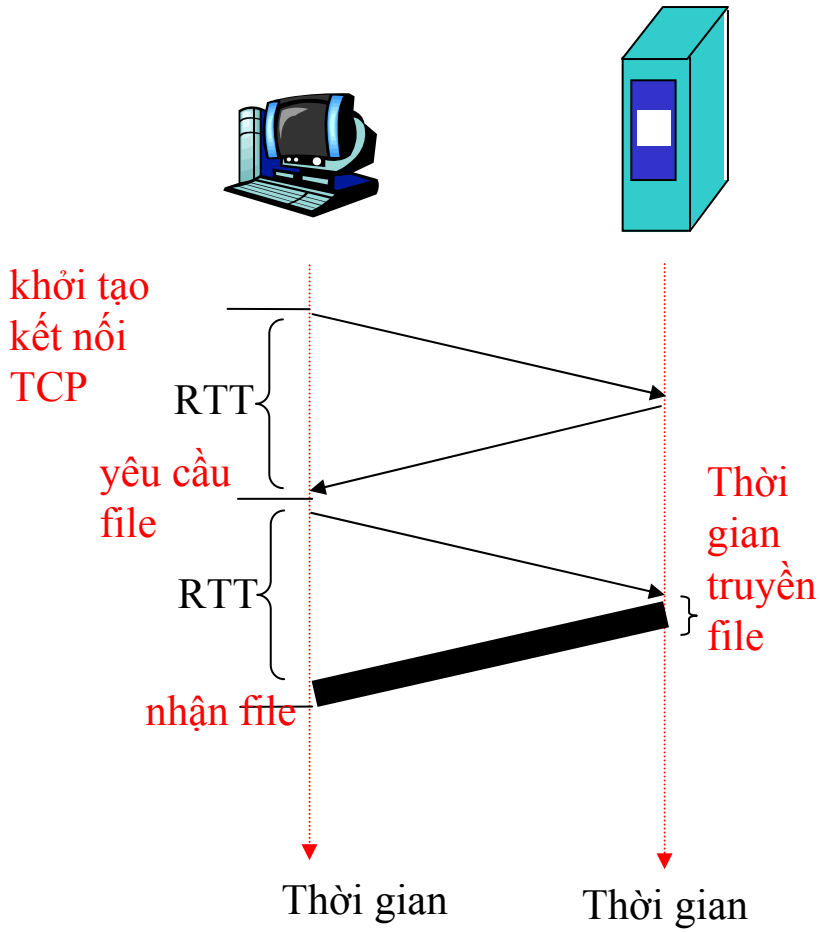
HTTP không bền vững: thời gian đáp ứng

Định nghĩa RTT: thời gian để gửi một gói nhỏ đi từ client đến server và quay lại.

Thời gian đáp ứng:

- Một RTT để khởi tạo kết nối TCP
- Một RTT cho yêu cầu HTTP và một vài byte đầu tiên của đáp ứng HTTP được trả về
- Thời gian truyền file

Tổng cộng = 2RTT+ Thời gian truyền file



HTTP bền vững

Vấn đề với HTTP không bền vững:

- Yêu cầu 2 RTT mỗi đối tượng
- Hệ điều hành liên quan đến mỗi kết nối TCP
- Các trình duyệt thường mở song song các kết nối TCP để đem về các tham chiếu đến các đối tượng

HTTP bền vững

- server bỏ kết nối sau khi mở để gửi đáp ứng leaves
- các thông điệp HTTP của tiến trình con cùng mô hình client/server gửi thông qua kết nối mở

Bền vững ***không có*** pipelining:

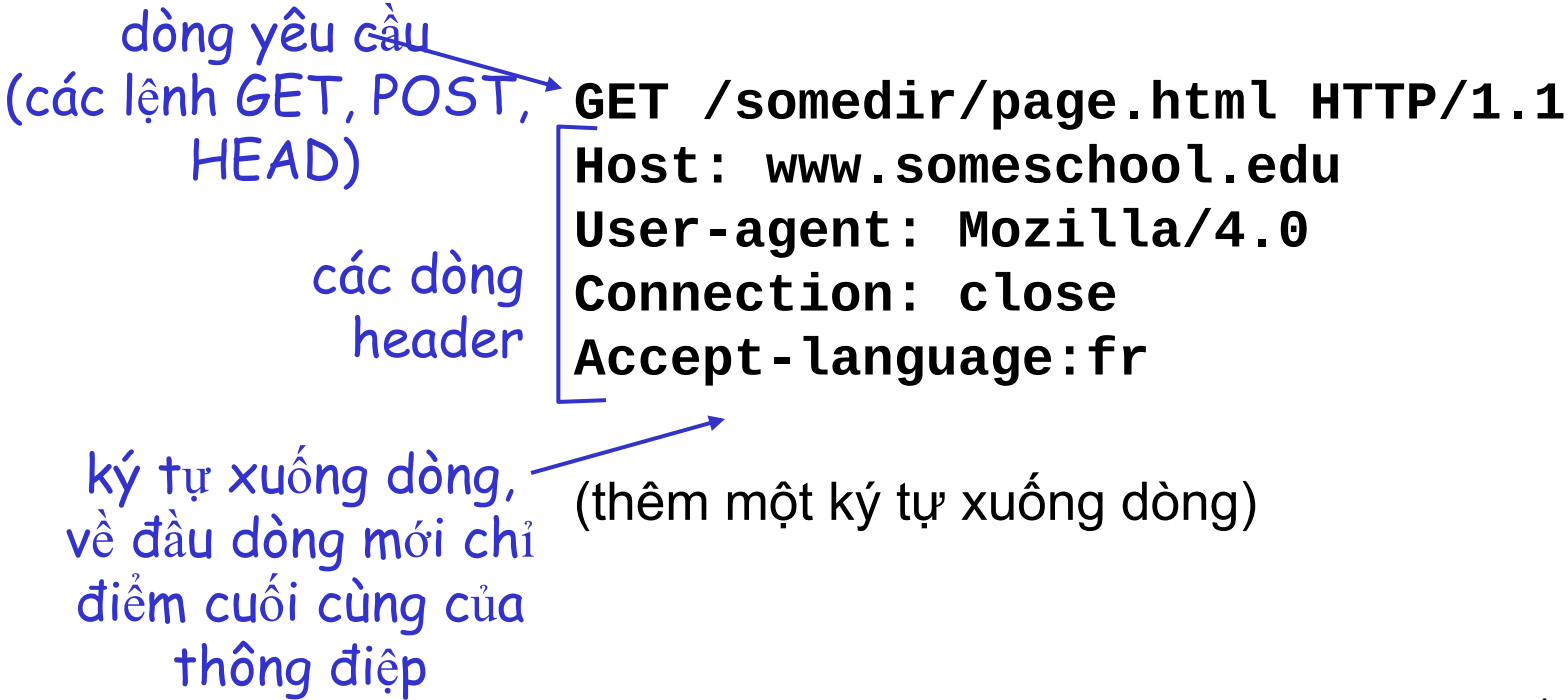
- client phát ra yêu cầu mới chỉ khi đáp ứng trước đó đã nhận xong
- 1 RTT cho mỗi đối tượng tham chiếu

Bền vững ***có*** pipelining:

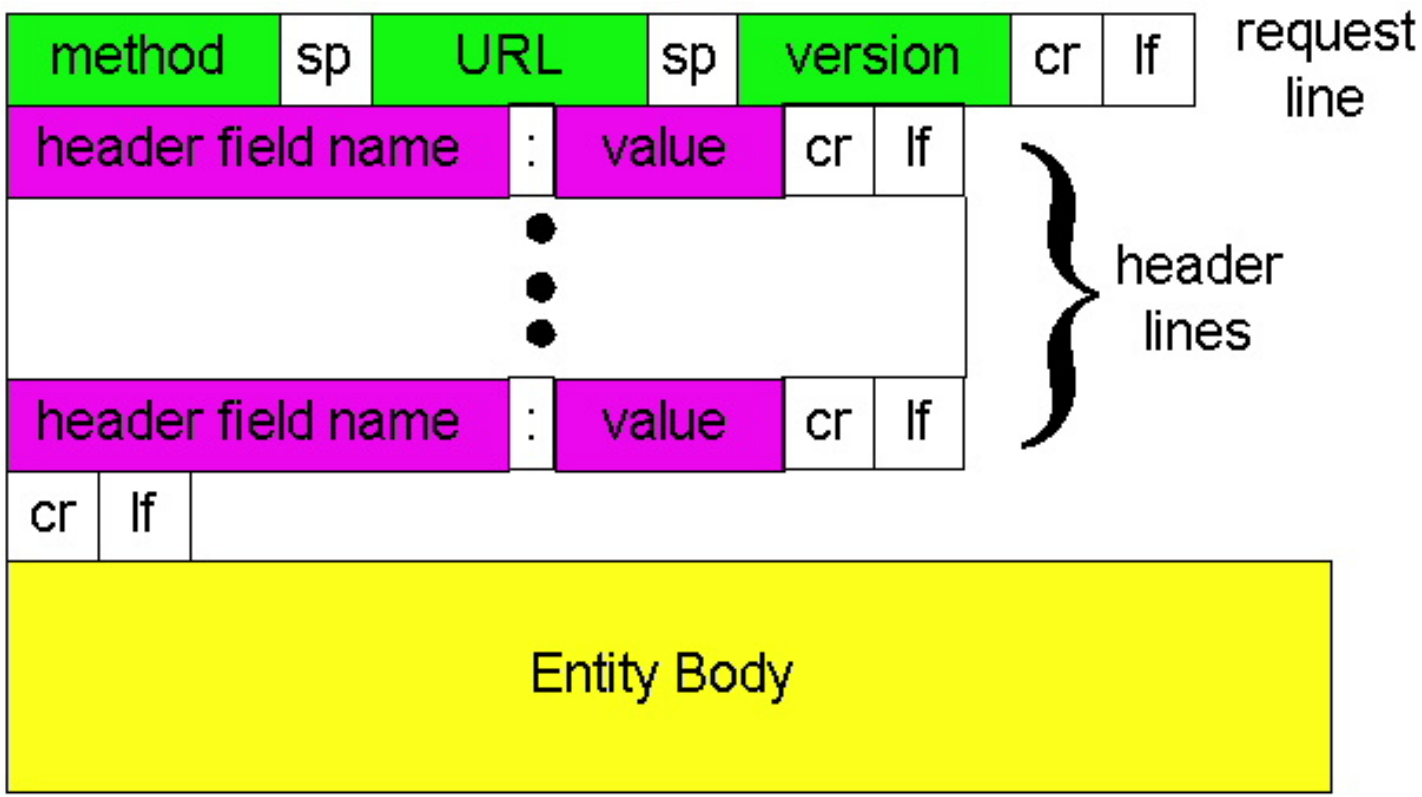
- mặc nhiên trong HTTP/1.1
- client gửi yêu cầu ngay sau khi gặp một đối tượng tham chiếu
- ít nhất 1 RTT cho tất cả đối tượng tham chiếu

Thông điệp yêu cầu HTTP

- 2 kiểu thông điệp HTTP: *yêu cầu, đáp ứng*
- Thông điệp yêu cầu HTTP:
  - ASCII (dạng thức con người có thể đọc được)



HTTP thông điệp yêu cầu: dạng thức tổng quát



Tải lên form input

Phương pháp Post:

- Web page thường chứa form input
- Input được tải lên vào server trong thân thực thể

Phương pháp URL:

- Dùng GET
- Input được tải lên trong trường URL của dòng yêu cầu:

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

Các kiểu phương pháp

HTTP/1.0

- GET
- POST
- HEAD
  - hỏi server để mặc đối tượng yêu cầu mà không đáp ứng

HTTP/1.1

- GET, POST, HEAD
- PUT
  - tải lên file trong thân thực thể đến đường dẫn được xác định trong trường URL
- DELETE
  - xóa file được xác định trong trường URL

Thông điệp đáp ứng HTTP

dòng trạng thái
(giao thức
mã trạng thái
cụm từ trạng thái)

các dòng
header

Dữ liệu, vd: file
HTML yêu cầu

```
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 ...
```

Các mã trạng thái đáp ứng HTTP

Trong dòng đầu tiên của thông điệp đáp ứng server-> client.

Một số mẫu:

200 OK

- yêu cầu thành công, đối tượng yêu cầu nằm ở phía sau thông điệp này

301 Moved Permanently

- đối tượng yêu cầu đã di chuyển, vị trí mới xác định ở phía sau thông điệp này (Location:)

400 Bad Request

- thông điệp yêu cầu server không hiểu

404 Not Found

- tài liệu yêu cầu không có trong server

505 HTTP Version Not Supported

Kiểm tra HTTP (phía client)

1. Telnet đến Web server ưa thích của bạn:

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

Mở kết nối TCP ở port 80 (port HTTP server mặc nhiên) tại cis.poly.edu. Mọi thứ nhập vào gửi đến ở port 80 tại cis.poly.edu

2. Nhập vào yêu cầu trong lệnh GET HTTP:

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

Do đánh lệnh này (enter 2 lần), bạn đã gửi yêu cầu GET tối thiểu (nhưng đầy đủ) đến HTTP server

3. Xem thông điệp đáp ứng gửi từ HTTP server!

Khảo sát hành động của HTTP

- telnet
- Ethereal

Trạng thái User-server: các cookie

Nhiều Web sites dùng các cookie

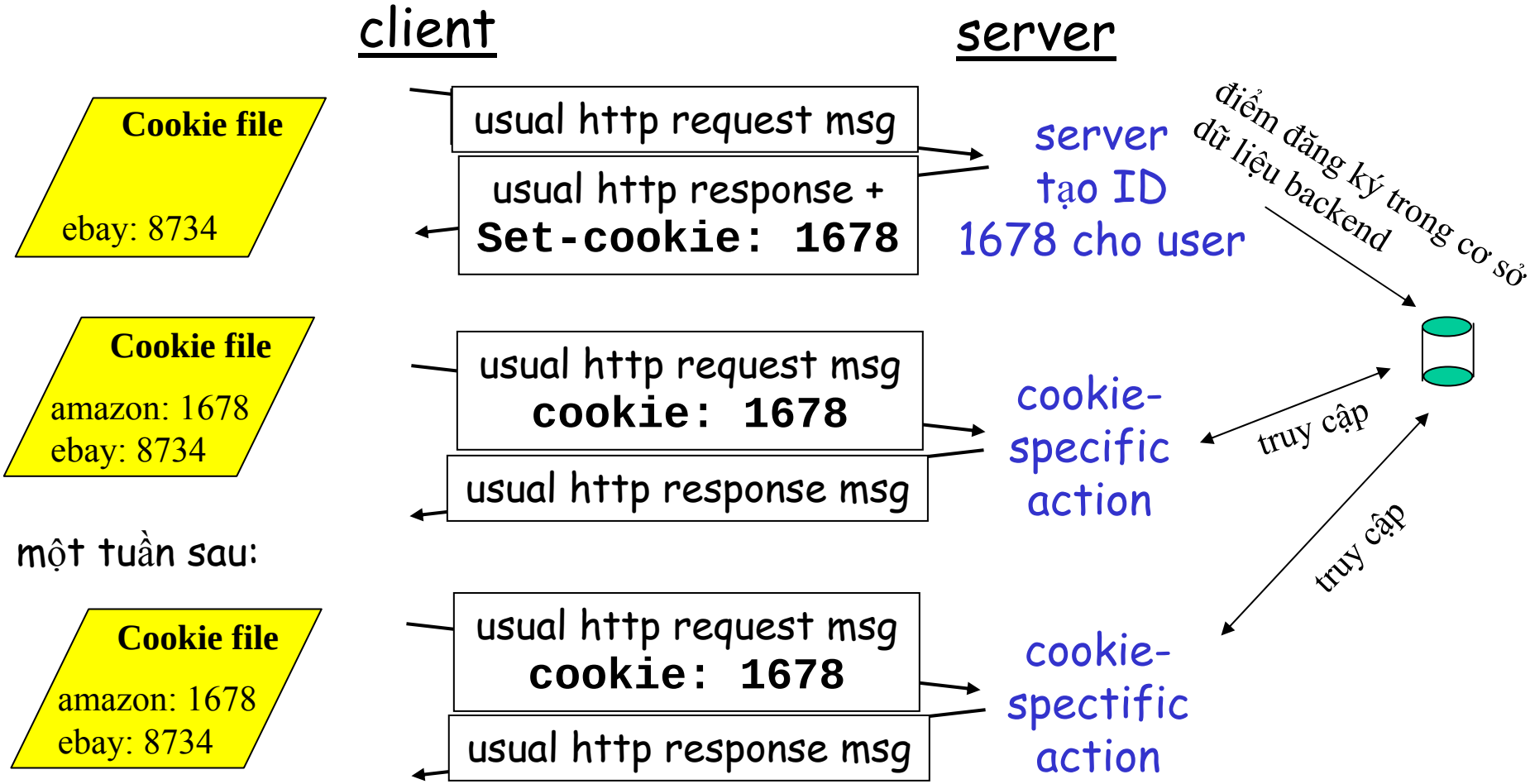
4 thành phần:

1) cookie header line của thông điệp đáp ứng HTTP
2) cookie header line trong thông điệp đáp ứng HTTP
3) cookie file lưu trong host của user, quản lý bởi trình duyệt của user
4) cơ sở dữ liệu back-end tại Web site

Ví dụ:

- Susan truy cập Internet luôn từ một PC
- She lần đầu tiên vào một e-commerce site xác định
- Khi yêu cầu khởi tạo HTTP đến site, site tạo một ID duy nhất và tạo một điểm đăng nhập trong cơ sở dữ liệu back-end cho ID đó

các cookie: lưu giữ "trạng thái" (tt.)



các cookie (tiếp)

Các cookie đem lại:

- sự cấp phép
- giỏ mua hàng
- các khuyến cáo
- trạng thái phiên làm việc của user (Web e-mail)

Làm thế nào để giữ "trạng thái":

- các thời điểm kết thúc giao thức: bảo trì trạng thái tại sender/receiver thông qua nhiều giao tác
- các cookie: trạng thái mang các thông điệp http

ngoài ra

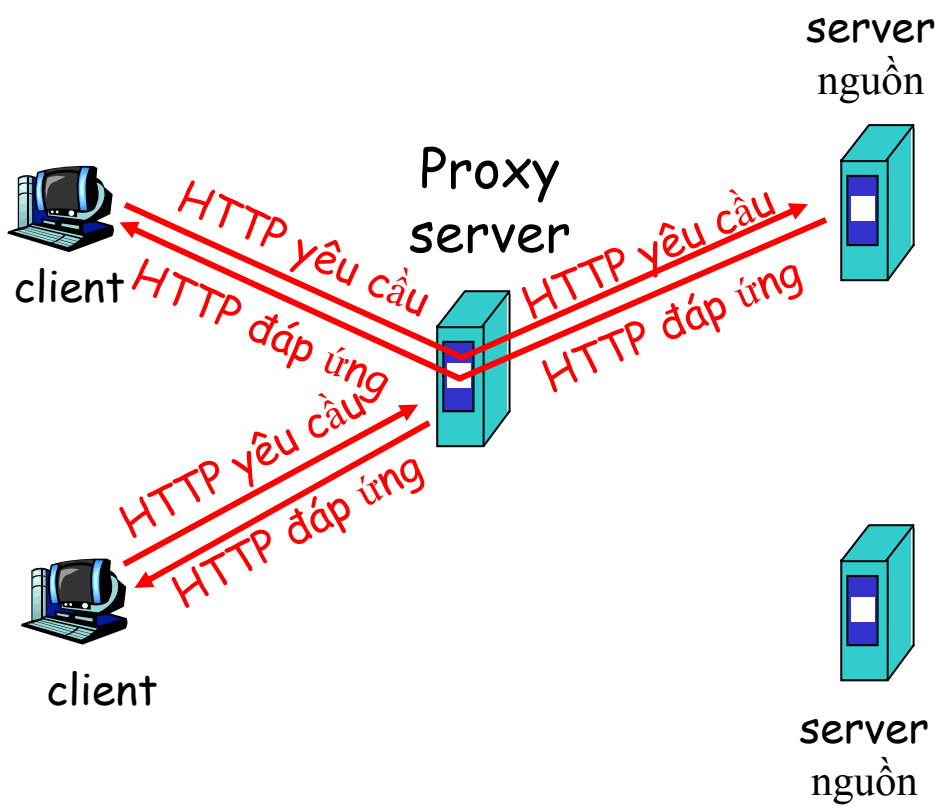
các cookie và sự riêng tư:

- các cookie cho phép các site biết nhiều hơn về bạn
- bạn có thể cung cấp tên và e-mail cho sites

Web caches (proxy server)

Mục tiêu: thỏa mãn yêu cầu của client không cần liên quan đến server nguồn

- user thiết lập trình duyệt: truy cập Web thông qua cache
- trình duyệt gửi tất cả yêu cầu HTTP cho cache
  - đối tượng trong cache: cache trả về đối tượng
  - ngược lại cache yêu cầu đối tượng từ server nguồn, sau đó trả về cho client



Web caching

- Cache hoạt động tại cả client và server
- Tiêu biểu cache được cài đặt bởi ISP (trường học, công ty, ISP riêng)

Tại sao dùng Web caching?

- Giảm thời gian đáp ứng cho yêu cầu của client
- Giảm lưu thông trên liên kết truy cập
- Internet rất ngờ nghệch với caches: cho phép những người cung cấp nội dung nghèo nàn phân phát hiệu quả nội dung đó (cũng vậy đối với P2P file sharing)

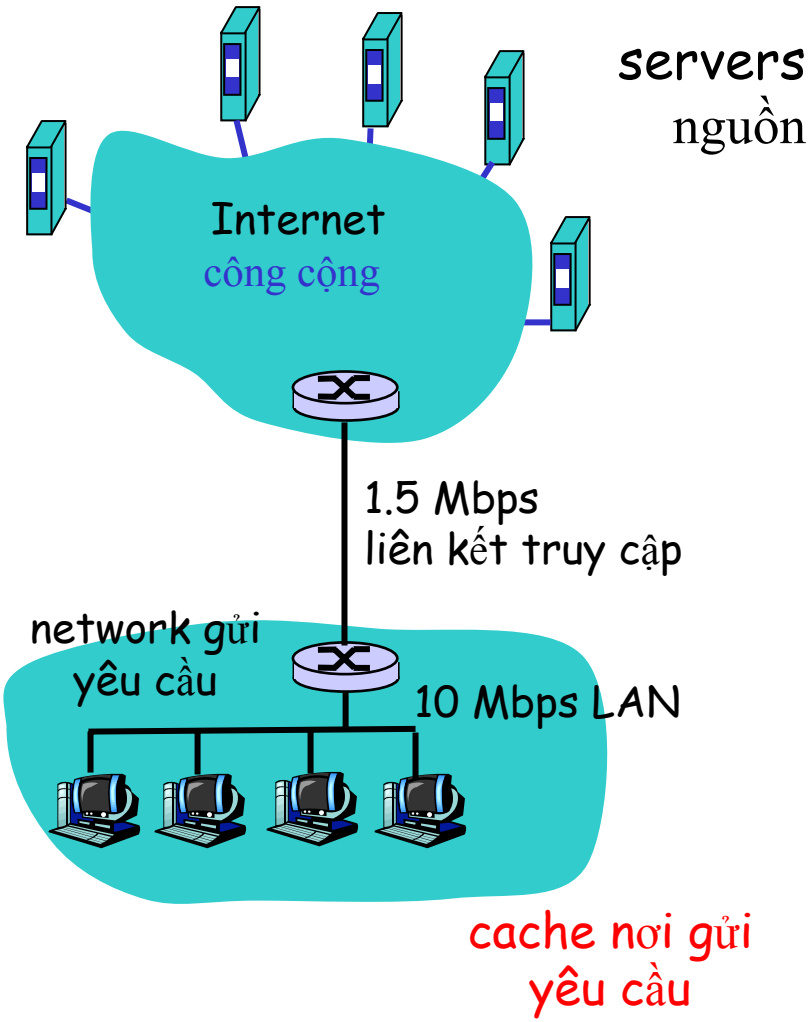
Ví dụ Caching

Giả sử

- kích thước trung bình đối tượng= 100,000 bits
- tốc độ trung bình yêu cầu từ trình duyệt đến server = 15/s
- độ trễ từ router nơi gửi yêu cầu đến server nguồn rồi quay lại = 2 s

Kết quả

- độ khả dụng của LAN = 15%
- độ khả dụng trên liên kết truy cập= 100%
- tổng thời gian trễ = trễ Internet + trễ truy cập+ trễ LAN = 2 s + các phút+ mili s



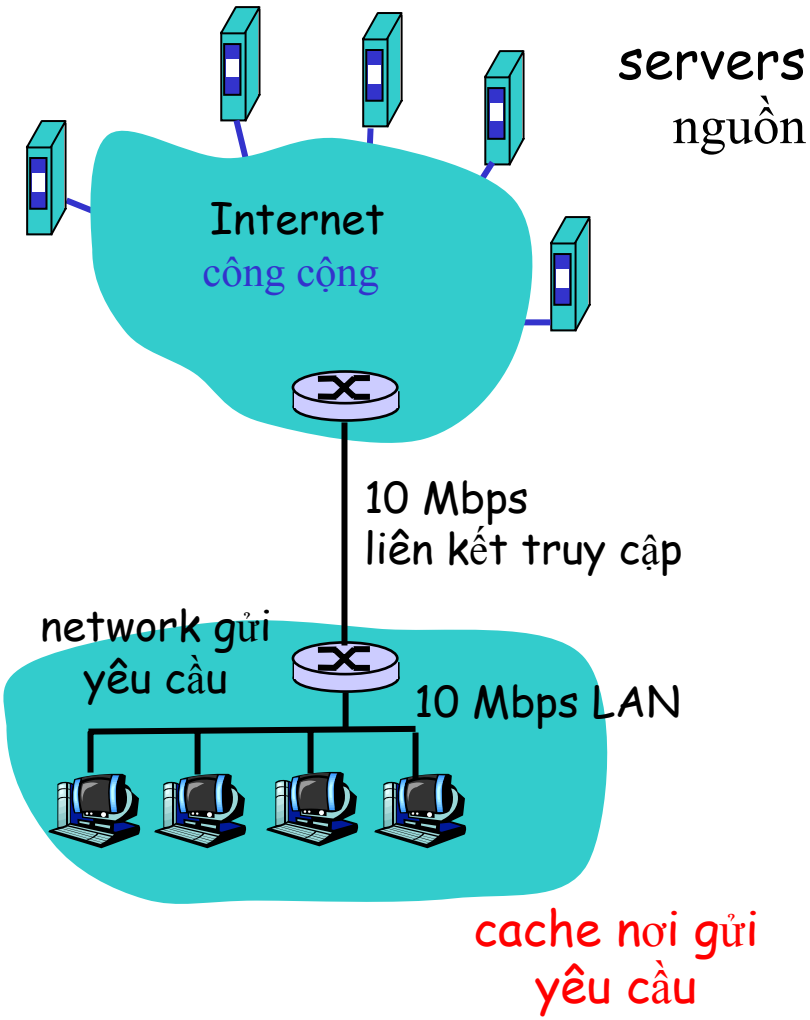
Ví dụ Caching (tiếp)

Giải pháp có thể

- tăng băng thông truy cập lên, ví dụ 10 Mbps

Kết quả

- độ khả dụng của LAN = 15%
- độ khả dụng trên liên kết truy cập = 15%
- tổng thời gian trễ = trễ Internet + trễ truy cập+ trễ LAN = 2 s + mili s + mili s
- thường tăng chi phí



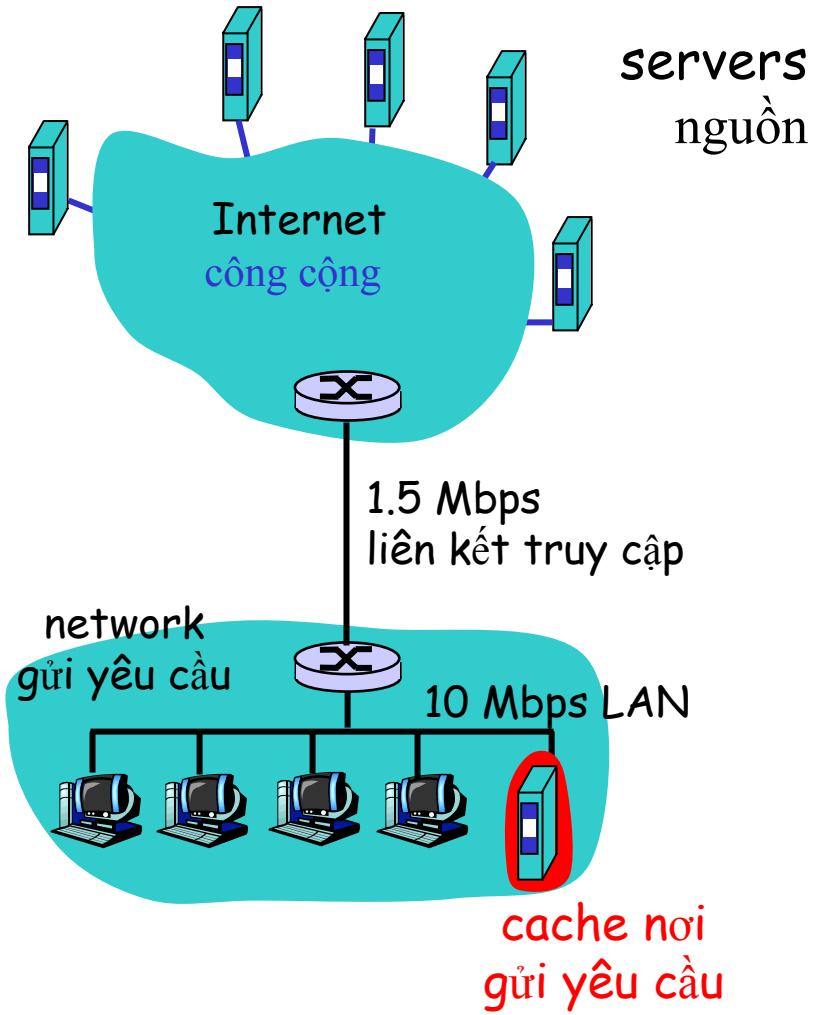
Ví dụ Caching (tiếp)

cài đặt cache

- tốc độ hỗ trợ là 0.4

kết quả

- 40% yêu cầu sẽ được thỏa mãn hầu như ngay lập tức
- 60% yêu cầu sẽ được thỏa mãn bởi server nguồn
- độ khả dụng trên liên kết truy cập giảm đến 60%, do trễ không đáng kể (vd 10 mili s)
- tổng thời gian trễ = trễ Internet + trễ truy cập+ trễ LAN = 0.6*(2.01) s + 0.4*mili s < 1.4 s



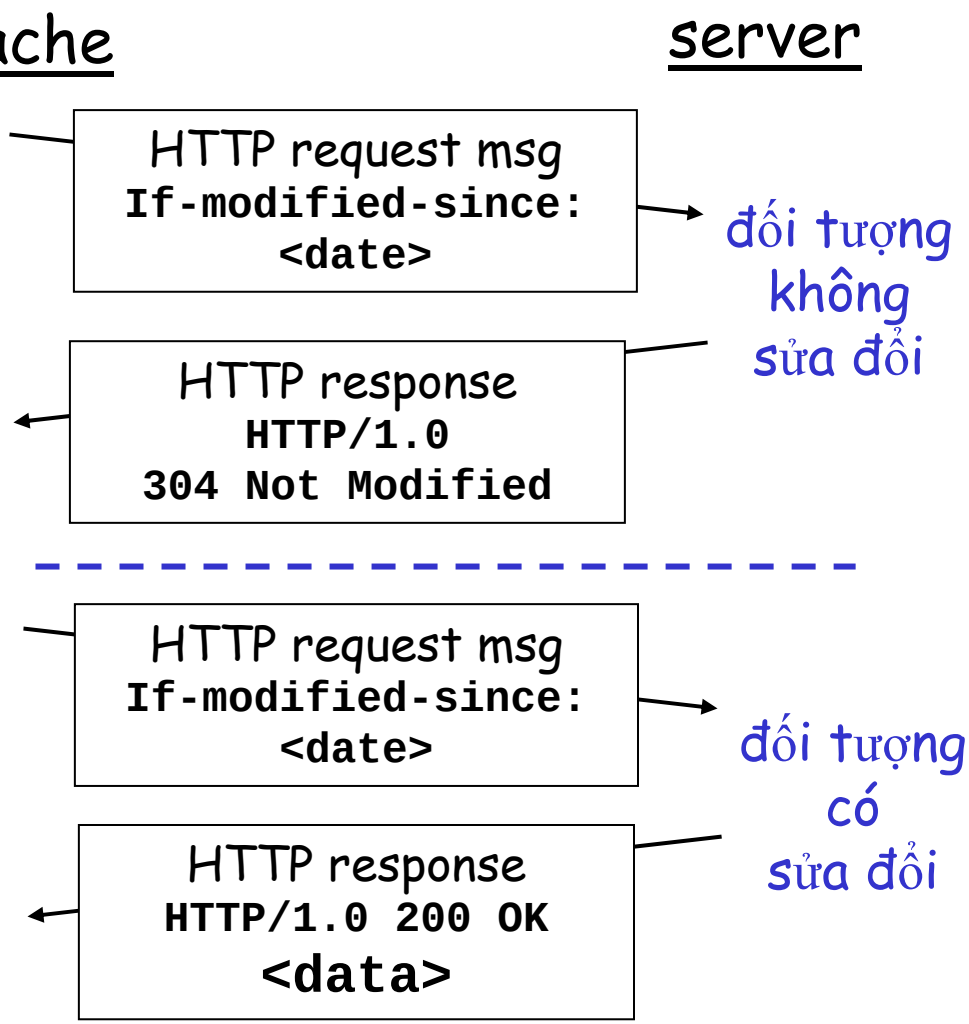
GET có điều kiện

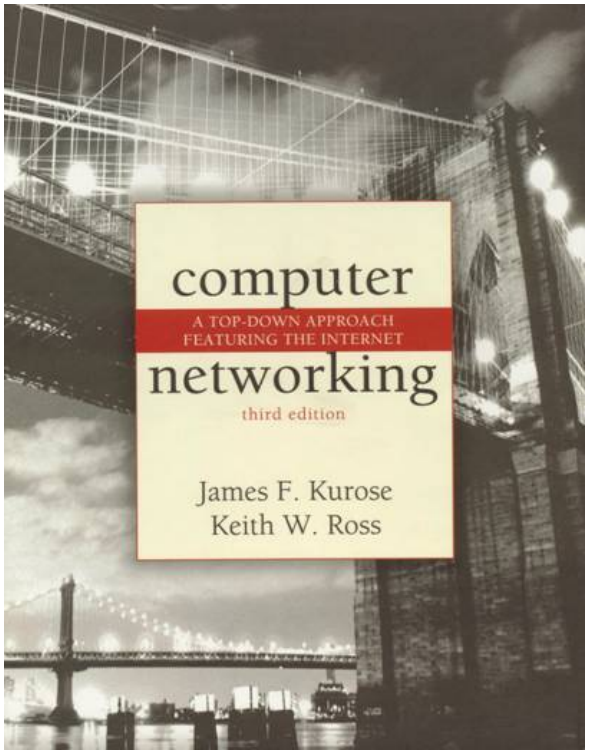
- Mục tiêu: không gửi đối tượng nếu cache đã cập nhật
- cache: xác định ngày của bản sao cache trong yêu cầu HTTP:

```
If-modified-since:
   <date>
```

- server: đáp ứng không chứa đối tượng nếu bản sao cache đã cập nhật:

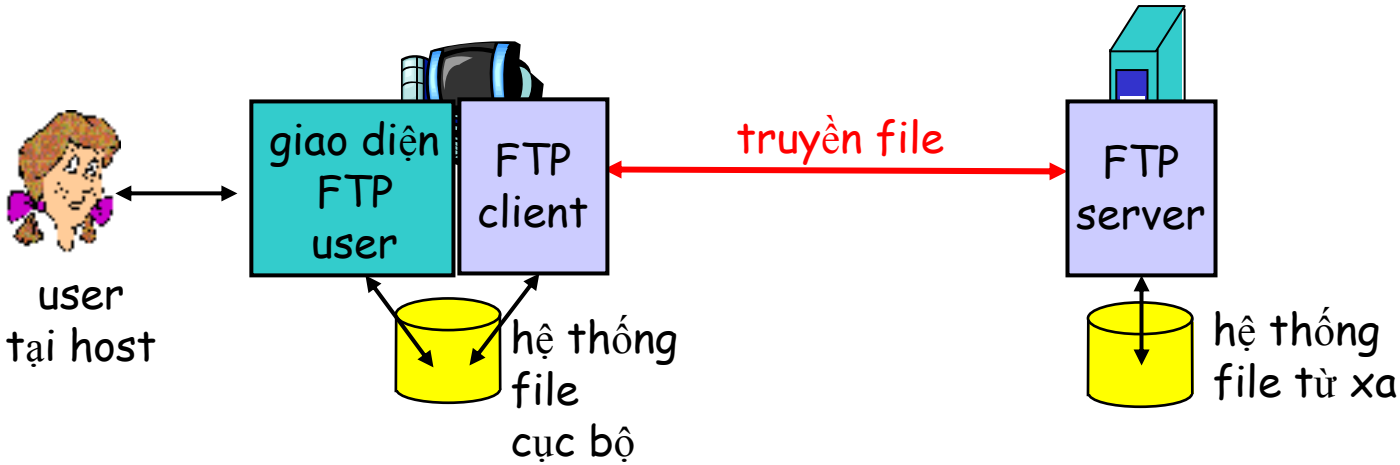
```
HTTP/1.0 304 Not
   Modified
```




computer
A TOP-DOWN APPROACH
FEATURING THE INTERNET
networking
third edition
James F. Kurose
Keith W. Ross

2.3 FTP

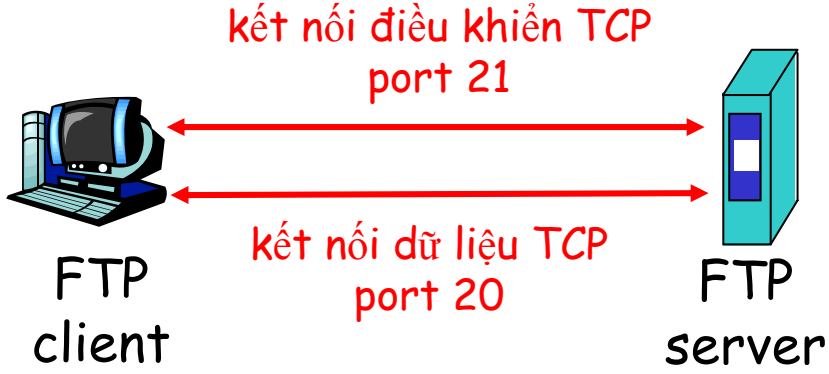
FTP: giao thức truyền file



- truyền file đến/từ host từ xa
- mô hình client/server
  - *client:* phía khởi tạo truyền (đến/từ host ở xa)
  - *server:* host ở xa
- ftp: RFC 959
- ftp server: port 21

FTP: kết nối dữ liệu, điều khiển riêng biệt

- FTP client tiếp xúc FTP server tại port 21, xác định TCP như giao thức transport
- Client lấy giấy phép thông qua kết nối điều khiển
- Client xem thư mục ở xa bằng việc gửi các lệnh thông qua kết nối điều khiển.
- Khi server nhận lệnh truyền file, server mở kết nối TCP thứ 2 (cho file) đến client
- Sau khi truyền 1 file, server đóng kết nối dữ liệu



- Server mở kết nối dữ liệu TCP khác để truyền file khác
- Điều khiển kết nối: "out of band"
- FTP server giữ lại "trạng thái": thư mục hiện hành, giấy phép trước đó

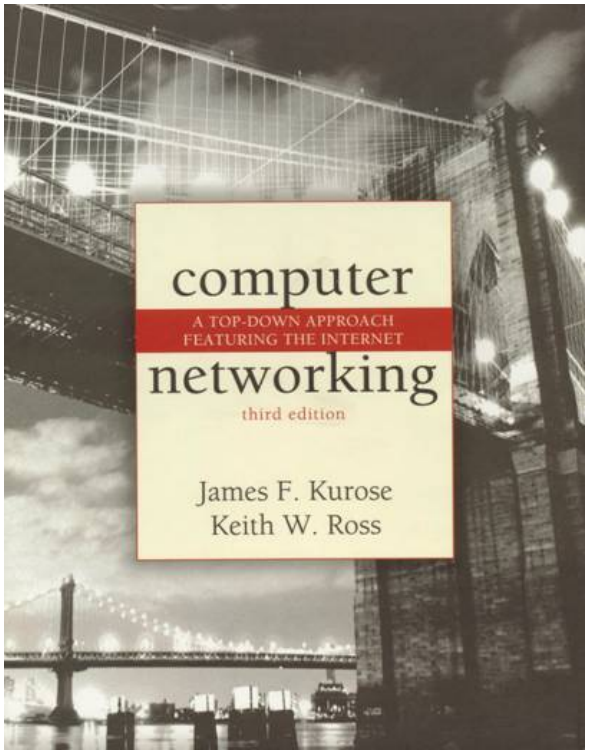
Các lệnh, phản hồi FTP

Ví dụ các lệnh:

- gửi như văn bản ASCII trên kênh điều khiển
- **USER *username***
- **PASS *password***
- **LIST** trả về danh sách của file trong thư mục hiện hành
- **RETR filename** trích chọn (lấy) file
- **STOR filename** lưu (đặt) file vào trong host ở xa

Ví dụ mã trả về

- mã trạng thái và cụm (như HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**



2.4 Electronic Mail

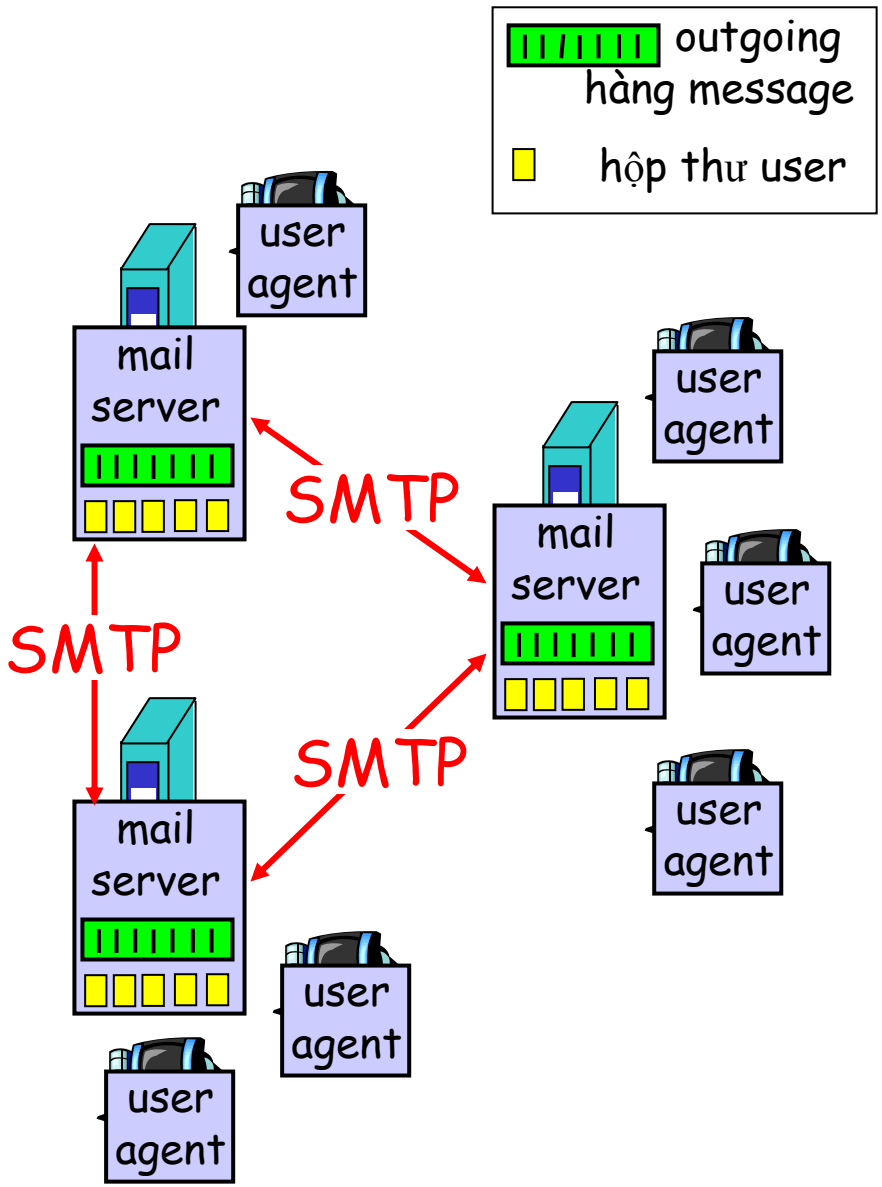
Electronic Mail

3 thành phần quan trọng:

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

User Agent

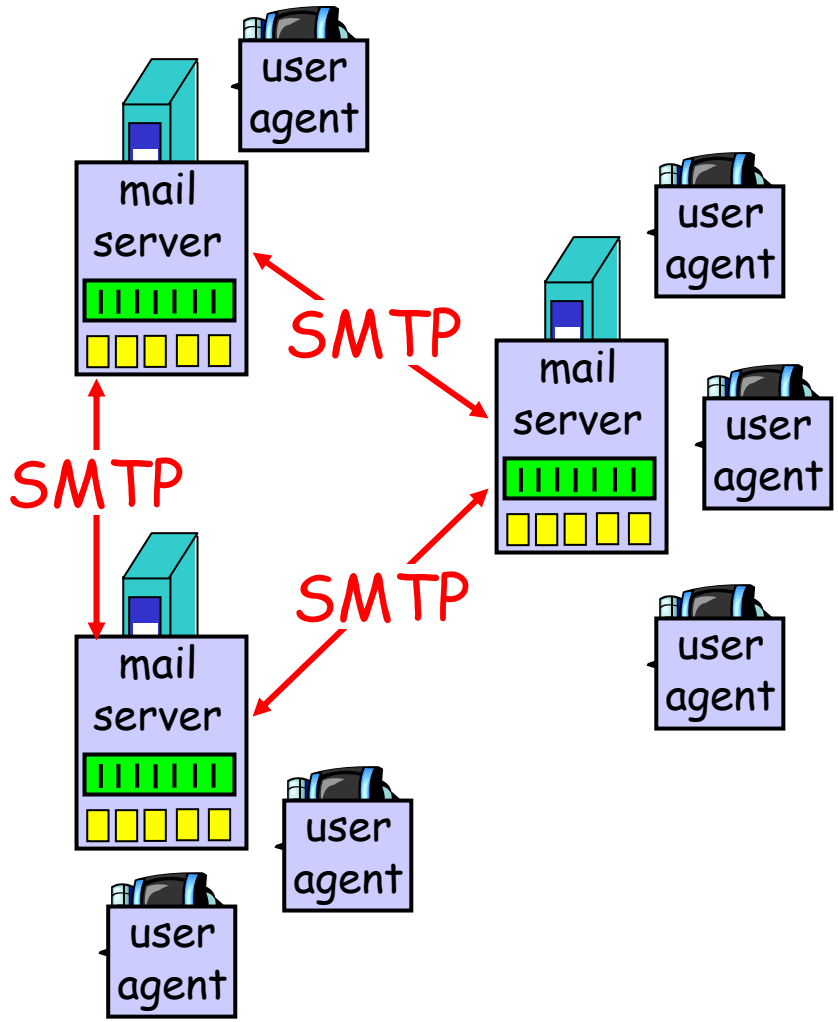
- còn gọi là "mail reader"
- viết, sửa đổi, đọc các thông điệp mail
- Ví dụ: Eudora, Outlook, elm, Netscape Messenger
- các thông điệp đi và đến được lưu trên server



Electronic Mail: mail servers

Mail Servers

- mailbox (hộp thư) chứa các thông điệp đến user
- hàng thông điệp cho các thông điệp email ra ngoài (chuẩn bị gửi)
- giao thức SMTP giữa các mail servers để gửi các thông điệp email
  - client: mail server gửi
  - "server": mail server nhận

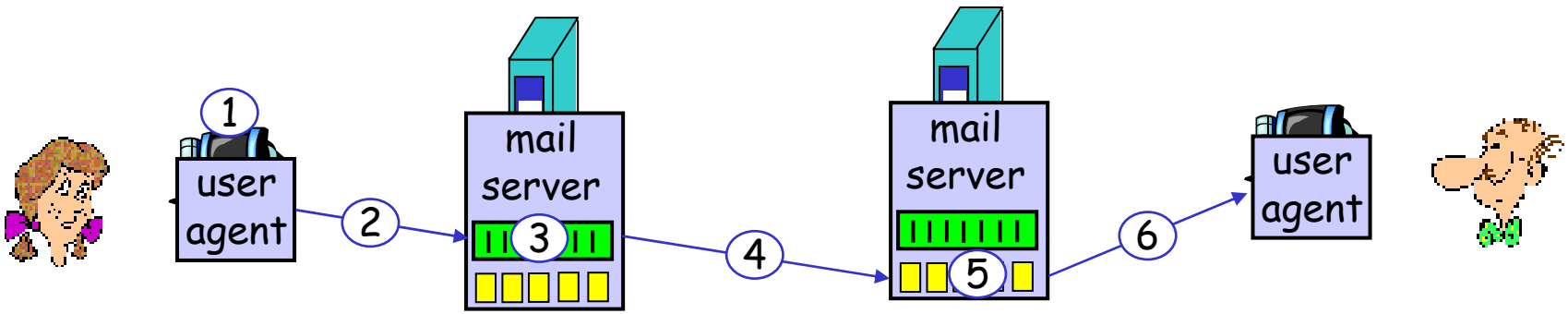


Electronic Mail: SMTP [RFC 2821]

- dùng TCP để truyền tin cậy thông điệp email từ client đến server trên port 25
- truyền trực tiếp: server gửi đến server nhận
- 3 kênh truyền
  - bắt tay (chào hỏi)
  - truyền thông điệp
  - đóng
- tương tác lệnh/phản hồi
  - lệnh: văn bản ASCII
  - phản hồi: mã trạng thái và cụm
- các thông điệp phải ở dạng mã ASCII 7-bit

Tình huống: Alice gửi cho Bob

1) Alice dùng UA viết thông điệp và "gửi đến" bob@someschool.edu
2) UA của Alice gửi thông điệp của cô ấy đến mail server; thông điệp được gia nhập vào hàng đợi
3) Phía Client của SMTP mở kết nối TCP với mail server của Bob
4) SMTP client gửi thông điệp của Alice trên kết nối TCP
5) mail server của Bob đặt thông điệp vào hộp thư của Bob
6) Bob kích hoạt trình user agent đọc thông điệp



Ví dụ tương tác SMTP

```
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
```

Thử nghiệm tương tác SMTP:

- **`telnet servername 25`**
- thấy 220 trả lời từ server
- nhập các lệnh HELO, MAIL FROM, RCPT TO, DATA, QUIT

lệnh trên cho phép bạn gửi email không cần dùng email client (reader)

SMTP

- SMTP dùng các kết nối bền vững
- SMTP yêu cầu các thông điệp (header & body) phải ở dạng thức 7-bit ASCII
- SMTP server dùng CRLF.CRLF xác định kết thúc thông điệp

So sánh với HTTP:

- HTTP: kéo
- SMTP: đẩy
- tất cả đều có tương tác lệnh/đáp ứng, các mã trạng thái ASCII
- HTTP: mỗi đối tượng được đóng kín trong thông điệp đáp ứng của nó
- SMTP: nhiều đối tượng được gửi trong thông điệp nhiều phần

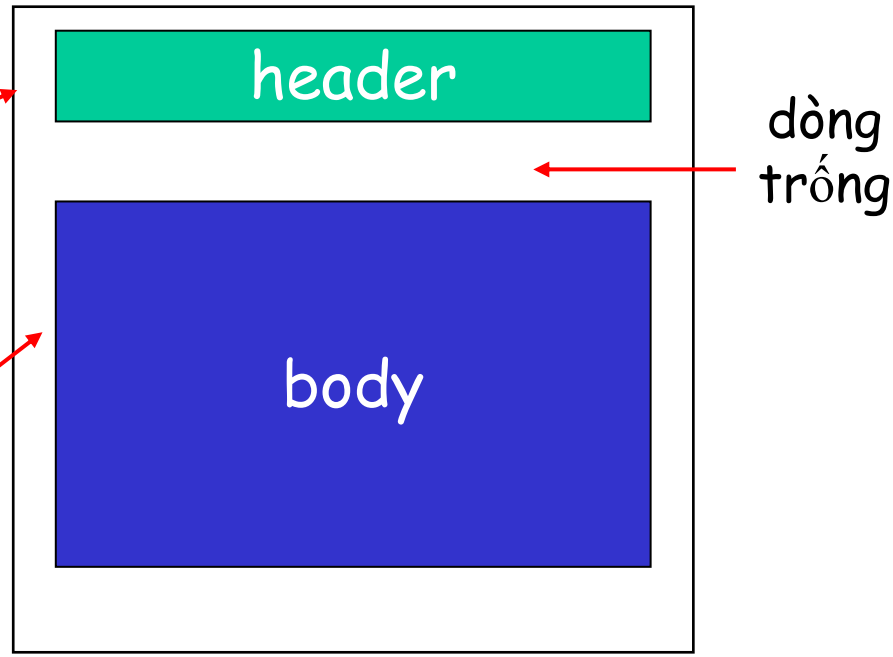
Dạng thức thông điệp email

SMTP: giao thức cho trao đổi các thông điệp email

RFC 822: chuẩn cho dạng thức văn bản:

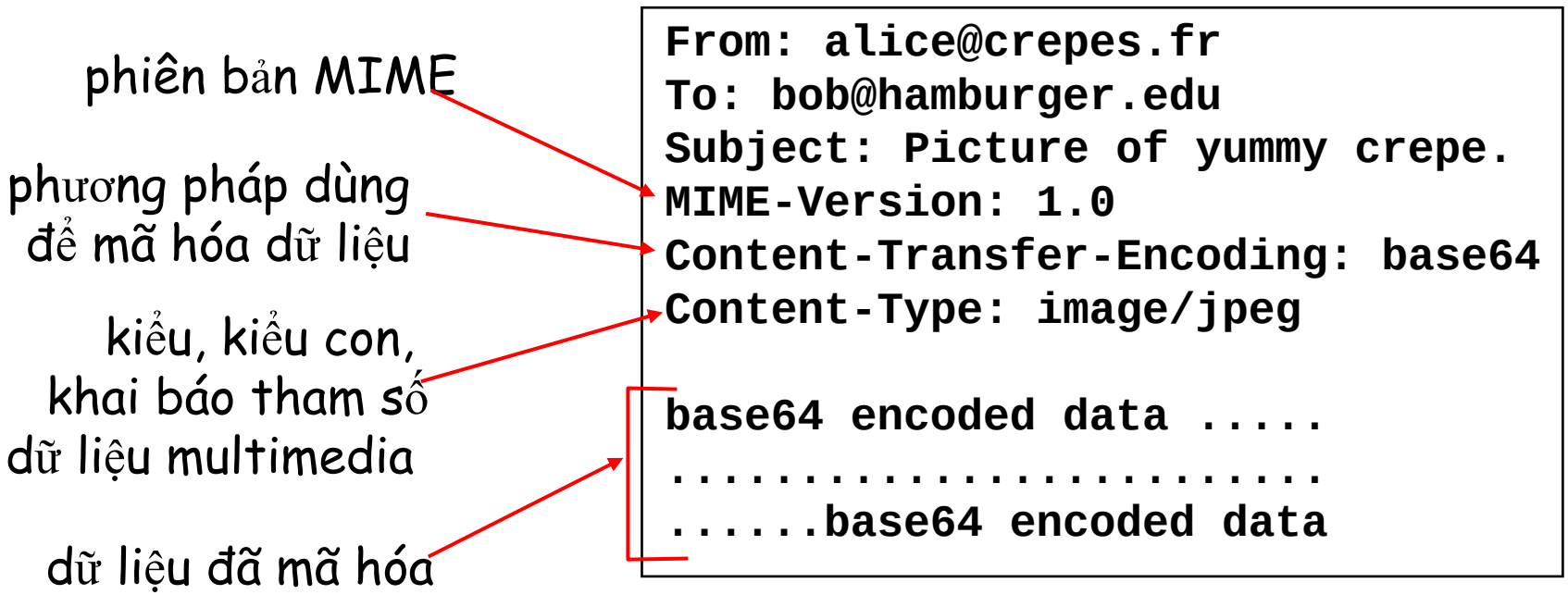
- các dòng header, ví dụ:
  - To:
  - From:
  - Subject:

  khác với các lệnh SMTP!
- body
  - "thông điệp", chỉ có các ký tự ASCII

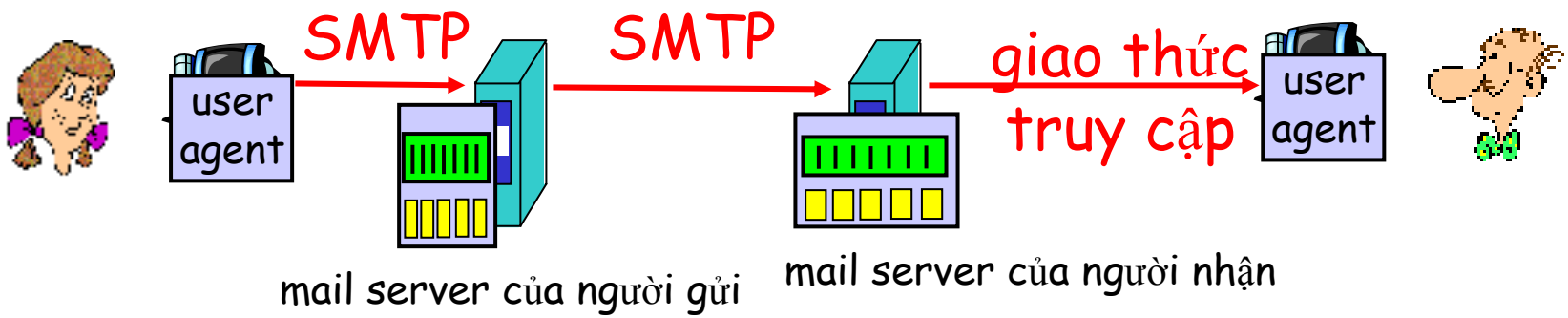


Dạng thức thông điệp: các mở rộng multimedia

- MIME: mở rộng email multimedia, RFC 2045, 2056
- các dòng bổ sung trong header của thông điệp khai báo kiểu nội dung MIME



Các giao thức truy cập email



- SMTP: truyền dẫn/lưu trữ vào server của người nhận
- Giao thức truy cập email: trích xuất từ server
  - POP: Post Office Protocol [RFC 1939]
    - cấp phép (agent <--> server) và download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - nhiều tính năng (phức tạp hơn)
    - điều khiển các thông điệp đã lưu trên server
  - HTTP: Hotmail , Yahoo! Mail,...

Giao thức POP3

giai đoạn cấp phép

- các lệnh phía client:
  - **user:** khai báo username
  - **pass:** password
- các đáp ứng phía server
  - **+OK**
  - **-ERR**

giai đoạn giao dịch, client:

- **list:** liệt kê các số thông điệp
- **retr:** trích xuất thông điệp theo số
- **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
```

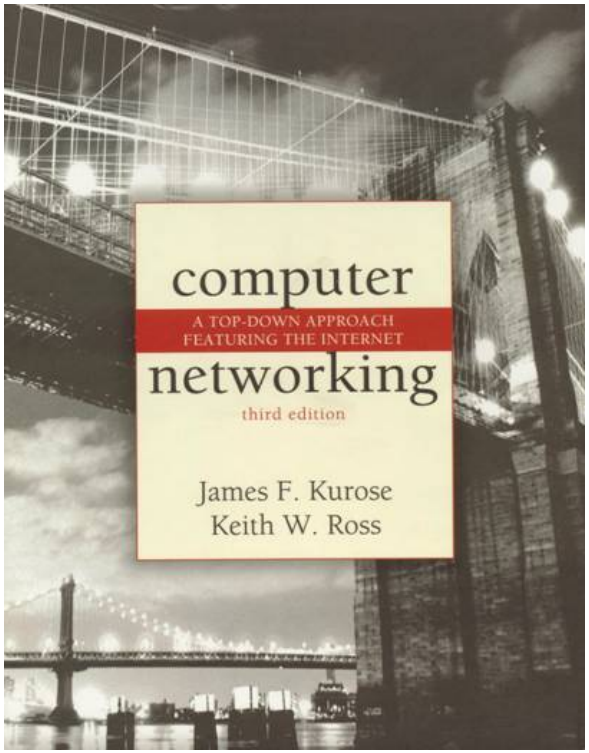
POP3 và IMAP

nghiên cứu thêm về POP3

- Ví dụ trước dùng chế độ "tải xuống và xóa".
- Bob không thể đọc lại email nếu thay đổi client
- "tải xuống-và-giữ": sao chép các thông điệp trên các client khác nhau
- POP3 không giữ trạng thái của các phiên làm việc

IMAP

- Giữ tất cả thông điệp tại 1 vị trí: server
- Cho phép user tổ chức các thông điệp theo dạng thư mục
- IMAP giữ trạng thái xuyên suốt các phiên làm việc:
  - các tên của thư mục và ánh xạ giữa ID của thông điệp và tên thư mục


computer
A TOP-DOWN APPROACH
FEATURING THE INTERNET
networking
third edition
James F. Kurose
Keith W. Ross

2.5 DNS

DNS: Domain Name System

Con người: nhiều cách nhận dạng:

- SSN, tên, #hộ chiếu

Internet hosts, routers:

- địa chỉ IP (32 bit) dùng cho các gói định địa chỉ
- "tên", ví dụ: www.yahoo.com – dùng bởi con người

Ánh xạ giữa địa chỉ IP và tên?

Domain Name System:

- *cơ sở dữ liệu phân bố* hiện thực theo tổ chức phân cấp của nhiều *servers tên*
- *giao thức lớp application* host, routers, name servers để truyền thông với các tên *phân giải* (địa chỉ/dịch ra tên)
  - lưu ý: chức năng lõi Internet, hiện thực như giao thức lớp application
  - phức tạp ở "biên" mạng

DNS

Các dịch vụ DNS

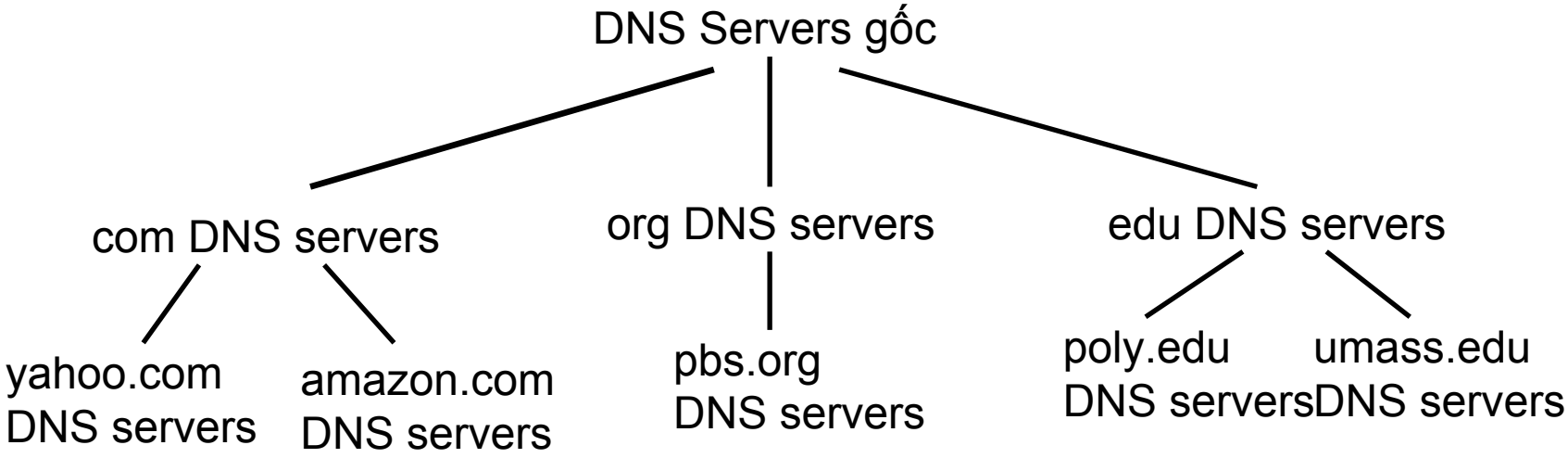
- Tên Host chuyển thành địa chỉ IP
- Bí danh Host
  - các tên đúng chuẩn và bí danh
- Bí danh Mail server
- Tải phân bố
  - Các Web server bản sao: tập các địa chỉ IP cho 1 tên đúng chuẩn

Tại sao không tập trung hóa DNS?

- một điểm chịu lỗi
- lưu lượng
- khoảng cách cơ sở dữ liệu tập trung
- bảo trì

không *linh hoạt!*

Cơ sở dữ liệu cấu trúc, phân bố

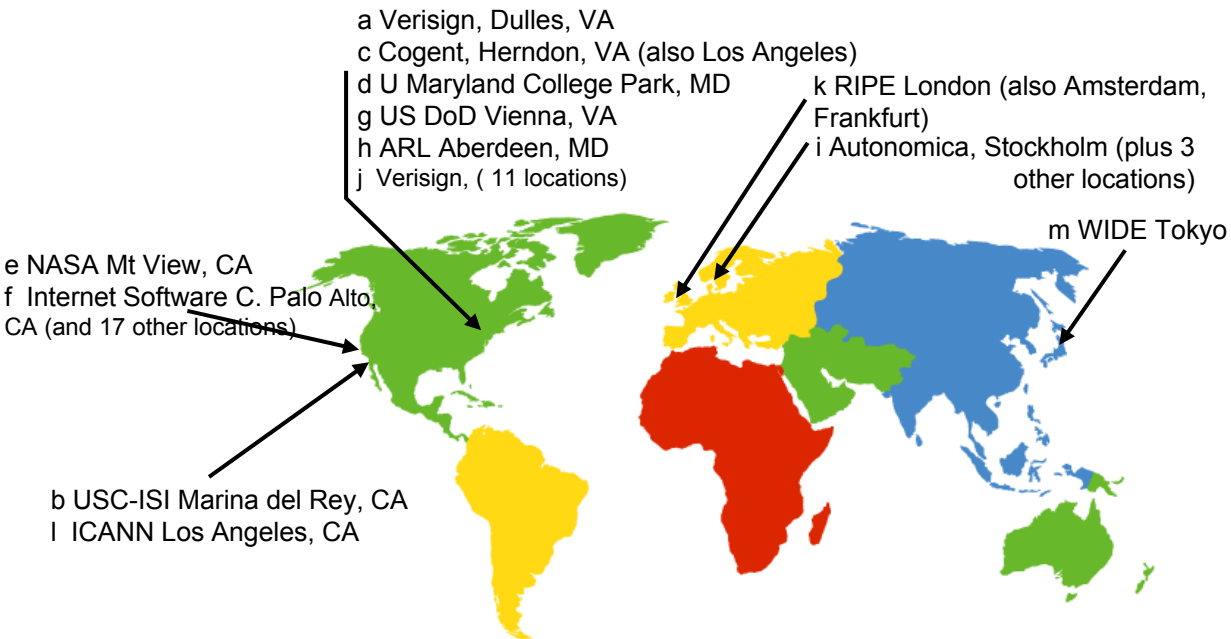


Client muốn IP cho www.amazon.com:

- Client hỏi một server gốc (root) để tìm com DNS server
- Client hỏi com DNS server để lấy amazon.com DNS server
- Client hỏi amazon.com DNS server để lấy địa chỉ IP của www.amazon.com

DNS: các server tên gốc

- tiếp xúc qua server tên cục bộ nào không thể phân giải tên
- server tên gốc:
  - tiếp xúc server tên có thẩm quyền nếu ánh xạ tên không xác định
  - lấy ánh xạ
  - trả về ánh xạ đến server tên cục bộ



13 name servers gốc trên toàn cầu

TLD và Server có thẩm quyền

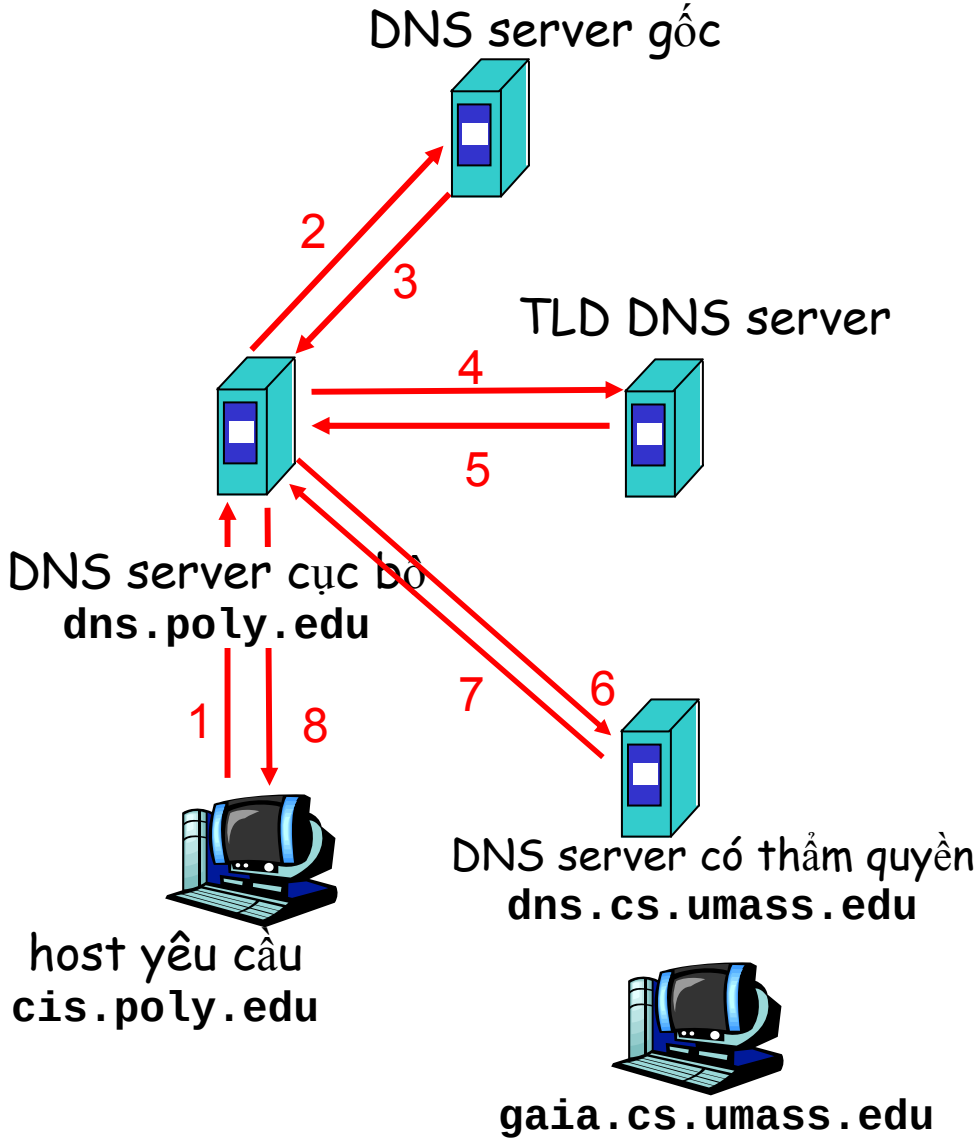
- Các server Top-level domain (TLD) : chịu trách nhiệm cho tên miền com, org, net, edu,... và các tên miền quốc gia như uk, fr, ca, jp.
  - Lĩnh vực giáo dục cho edu TLD
- Các DNS server có thẩm quyền: DNS server của tổ chức, cung cấp các tên host có thẩm quyền để ánh xạ IP cho server (ví dụ: Web và mail).
  - Không thể duy trì bởi tổ chức hoặc người cung cấp dịch vụ

Server tên cục bộ

- Không hoàn toàn theo cấu trúc phân cấp
- Mỗi ISP (ISP cá nhân, công ty, trường học) có một server cục bộ như vậy.
  - cũng gọi là "server tên mặc nhiên"
- Khi một host tạo một truy vấn DNS, truy vấn đó được gửi tới DNS server cục bộ của nó
  - Hoạt động như một proxy, chuyển truy vấn vào cho tổ chức phân cấp

Ví dụ

- Host tại cis.poly.edu muốn địa chỉ IP của gaia.cs.umass.edu



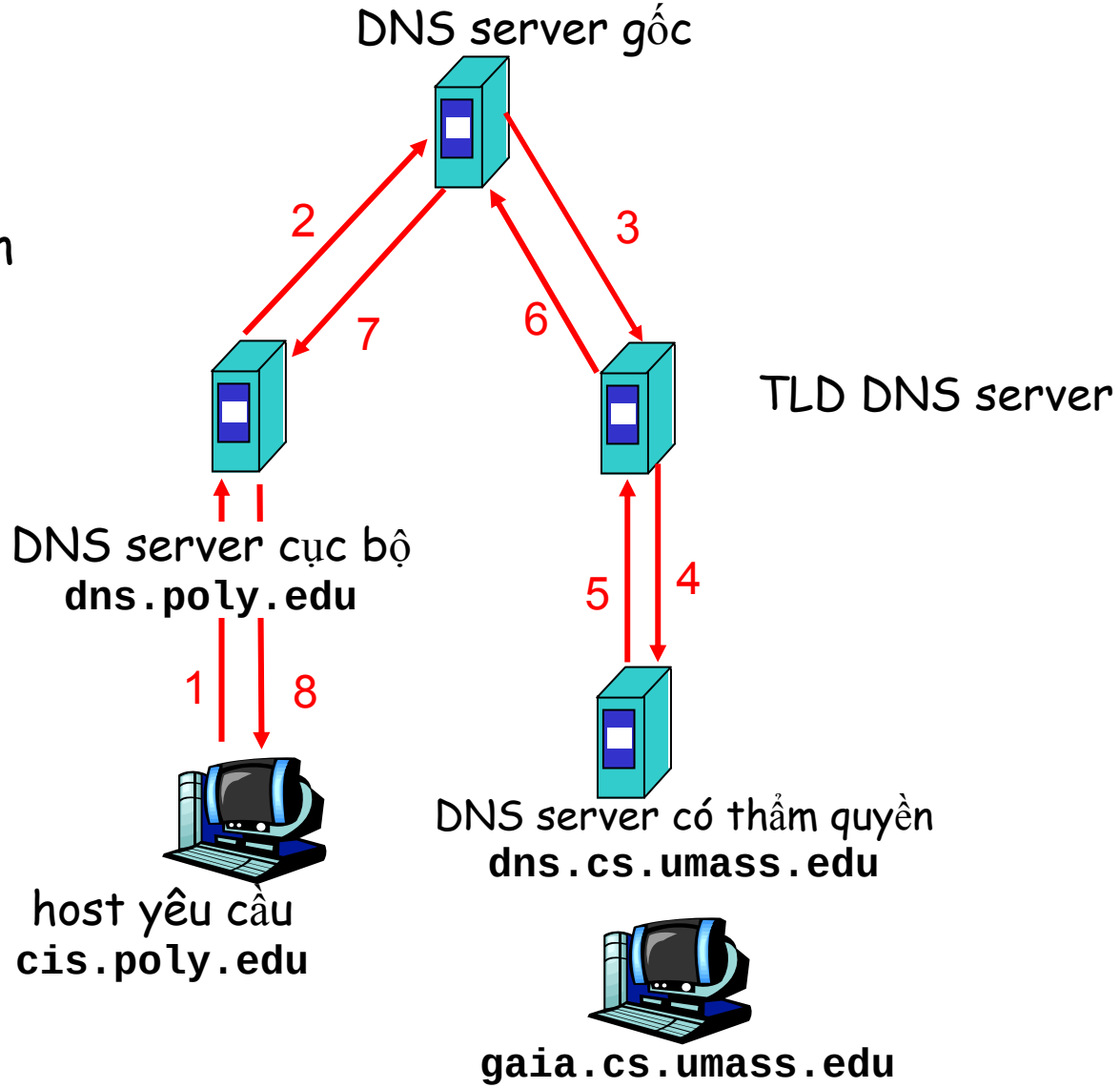
Các truy vấn đệ quy

truy vấn đệ quy:

- đẩy trách nhiệm phân giải tên cho server tên đã tiếp xúc được
- tải quá nặng?

truy vấn tuần tự:

- tên đã tiếp xúc được trả lời với tên của server
- "Tôi không biết tên đó, nhưng có thể hỏi server này"



DNS: caching và cập nhật các record

- một khi server tên học cách ánh xạ, nó *cache* ánh xạ
  - điểm đăng nhập cache sẽ thoát ra (biến mất) sau một vài lần
  - TLD servers điển hình sẽ được cache trong các server tên cục bộ
    - Do đó server tên gốc sẽ không thường xuyên được truy cập
- cơ chế cập nhật/thông báo bên dưới được thiết kế bởi IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

Các DNS record

DNS: cơ sở dữ liệu phân bố lưu trữ các record tài nguyên (RR)

dạng thức RR: **(name, value, type, ttl)**

- Type=A
  - **name** là tên host
  - **value** là địa chỉ IP
- Type=NS
  - **name** là tên miền (vd: foo.com)
  - **value** là tên host của server tên có thẩm quyền cho tên miền này
- Type=CNAME
  - **name** là bí danh của tên "chuẩn" (tên thực)

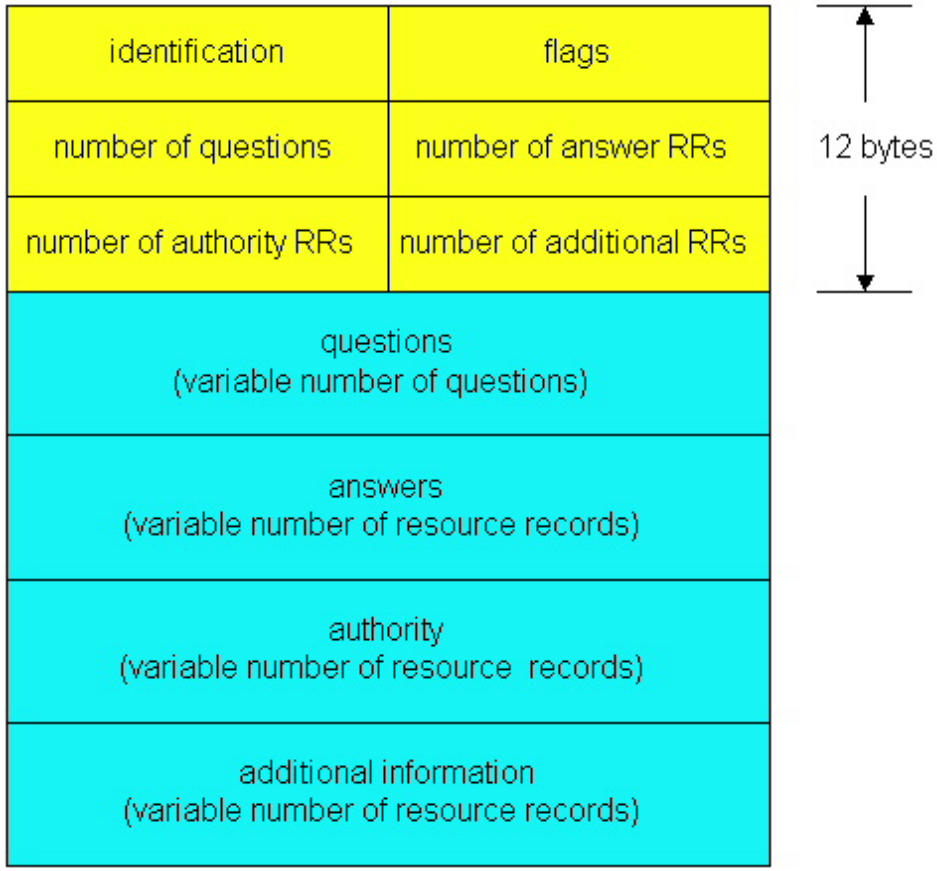
    `www.ibm.com` là tên thực `servereast.backup2.ibm.com`
  - **value** là tên chuẩn
- Type=MX
  - **value** là tên của email server liên kết với **name**

Giao thức và các thông điệp DNS

Giao thức DNS: các thông điệp *truy vấn* và *trả lời*, đều có cùng *dạng thức thông điệp*

header thông điệp

- identification: 16 bit # cho truy vấn, trả lời cho truy vấn dùng cùng #
- flags:
  - truy vấn hoặc trả lời
  - đệ quy mong chờ
  - đệ quy sẵn sàng
  - trả lời được cấp phép



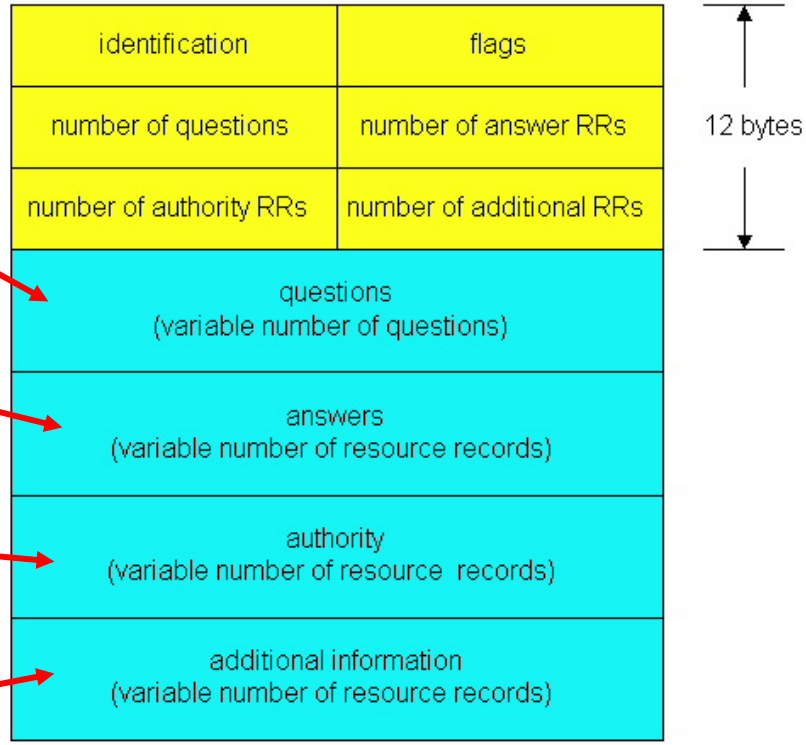
Giao thức và các thông điệp DNS

các trường Name, type cho 1 truy vấn

các RR trong đáp ứng cho truy vấn

các record cho các server có thẩm quyền

thông tin "hữu ích" bổ sung có thể sẽ dùng

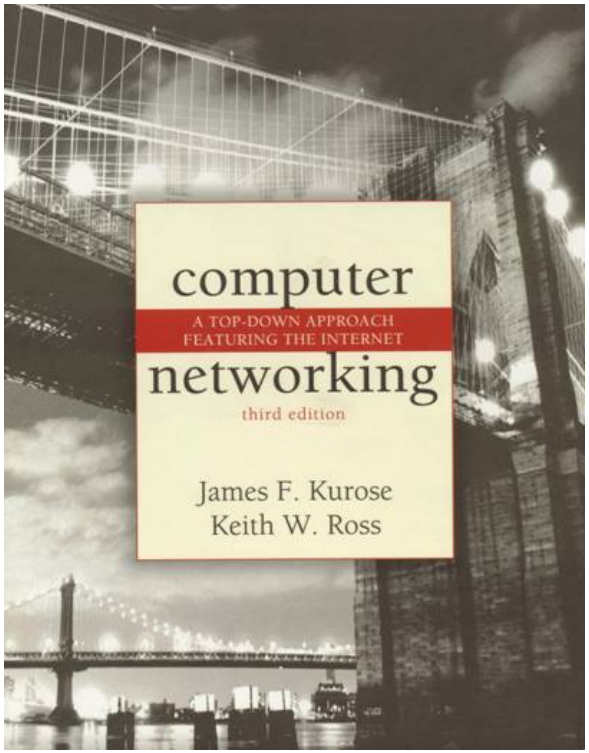


Chèn các record vào DNS

- Ví dụ: mới tạo "Network Utopia"
- Đăng ký tên miền networkuptopia.com tại một registrar (ví dụ: Network Solutions)
  - Cần cung cấp cho registrar tên và địa chỉ IP của server tên có thẩm quyền (primary và secondary) của bạn
  - Registrar chèn 2 RR vào trong com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- Đưa vào trong server có thẩm quyền record Type A cho www.networkuptopia.com và bản ghi Type MX cho networkutopia.com
- Làm sao người khác có thể biết được địa chỉ IP Web site của bạn?



2.6 Chia sẻ file P2P

Chia sẻ file P2P

Ví dụ

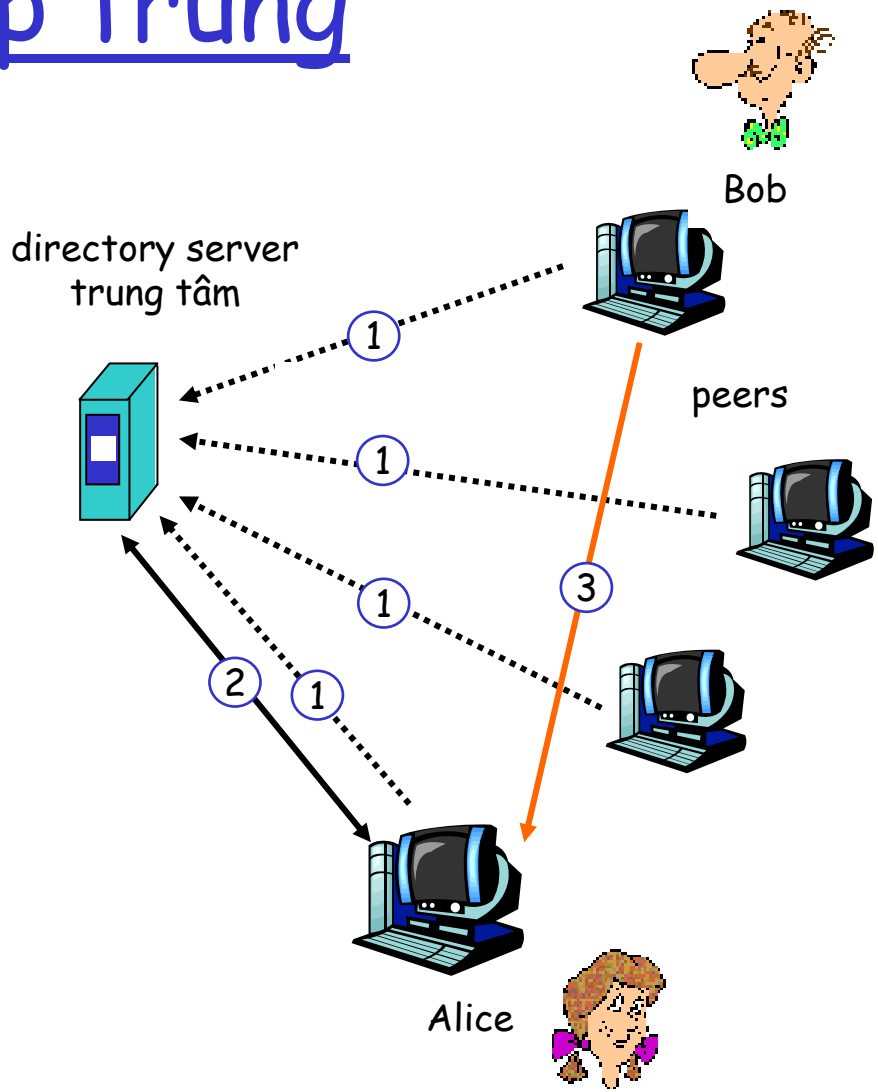
- Alice chạy ứng dụng P2P client trên máy tính xách tay của cô
- Kết nối không liên tục vào Internet; lấy địa chỉ IP cho mỗi kết nối
- Hỏi về "Hey Jude"
- Ứng dụng sẽ hiển thị những peer khác có bản sao của Hey Jude.
- Alice chọn 1 trong những peer, là Bob.
- File được sao chép từ máy tính của Bob: HTTP
- Trong khi Alice tải xuống, các user khác tải lên từ Alice.
- Alice là 1 peer đóng cả vai trò Web client và Web server tạm thời.

Tất cả peer là server = độ linh hoạt cao!

P2P: directory tập trung

đầu tiên do "Napster" thiết kế

1) khi peer kết nối, nó thông báo cho server trung tâm:
   - địa chỉ IP
   - nội dung
2) Alice truy vấn "Hey Jude"
3) Alice yêu cầu file từ Bob



P2P: các vấn đề với directory tập trung

- Một điểm chịu lỗi
- Hiện tượng tắc nghẽn "cổ chai"
- Xâm phạm bản quyền

truyền file không tập trung nhưng tìm kiếm nội dung thì tập trung cao độ

Tràn ngập truy vấn: Gnutella

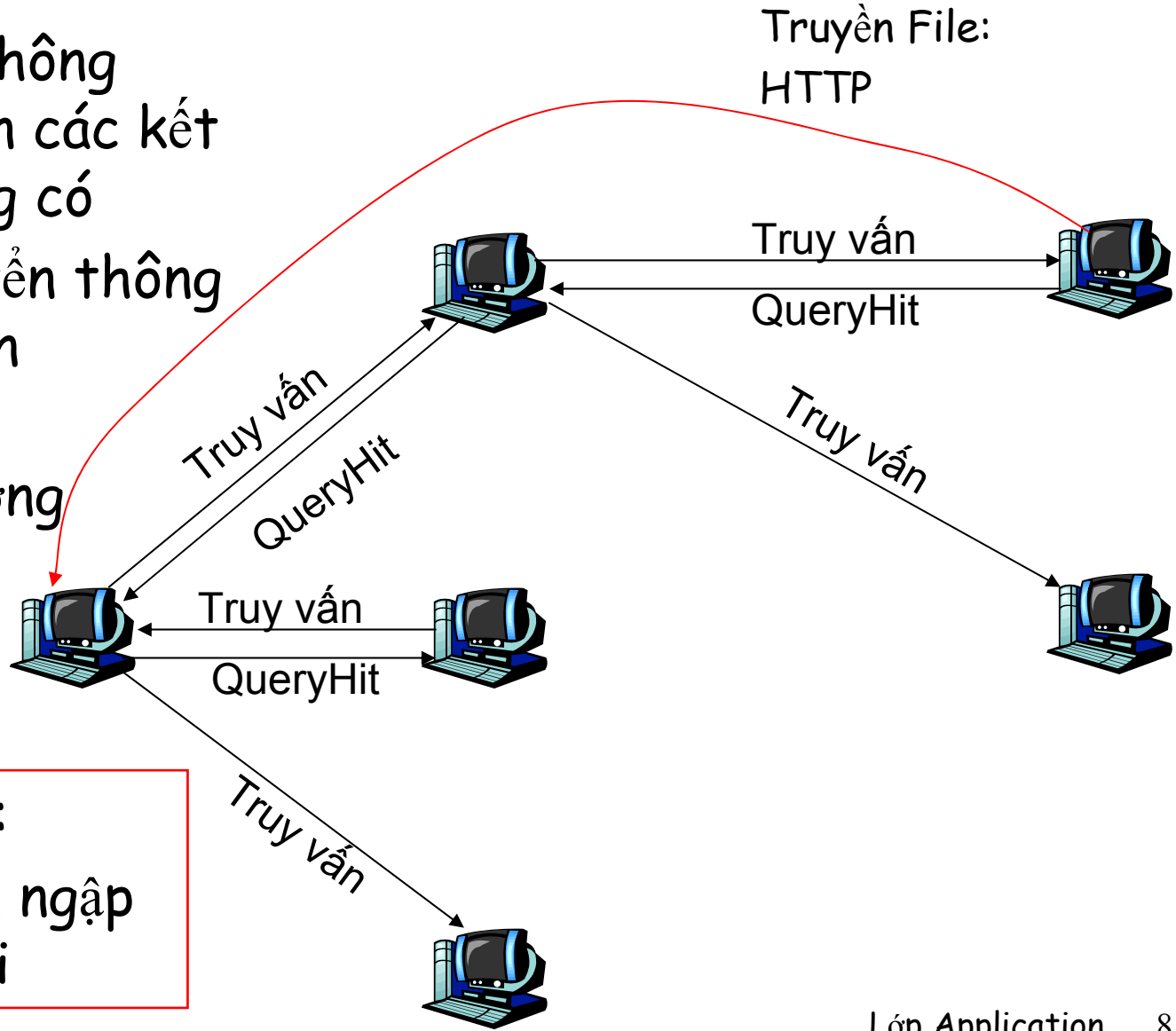
- hoàn toàn phân bố
  - không có server tập trung
- giao thức tên miền chung
- nhiều Gnutella client hiện thực giao thức

overlay network: đồ thị

- có cạnh giữa peer X và Y nếu có 1 kết nối TCP
- tất cả các peer đang hoạt động và các cạnh là mạng overlay network
- cạnh không phải là một liên kết vật lý
- peer sẽ kết nối với < 10 peer lân cận

Gnutella: giao thức

- Truy vấn thông điệp gửi trên các kết nối TCP đang có
- peer chuyển thông điệp truy vấn
- QueryHit gửi trên đường ngược chiều



Độ linh hoạt:

hạn chế tràn ngập theo phạm vi

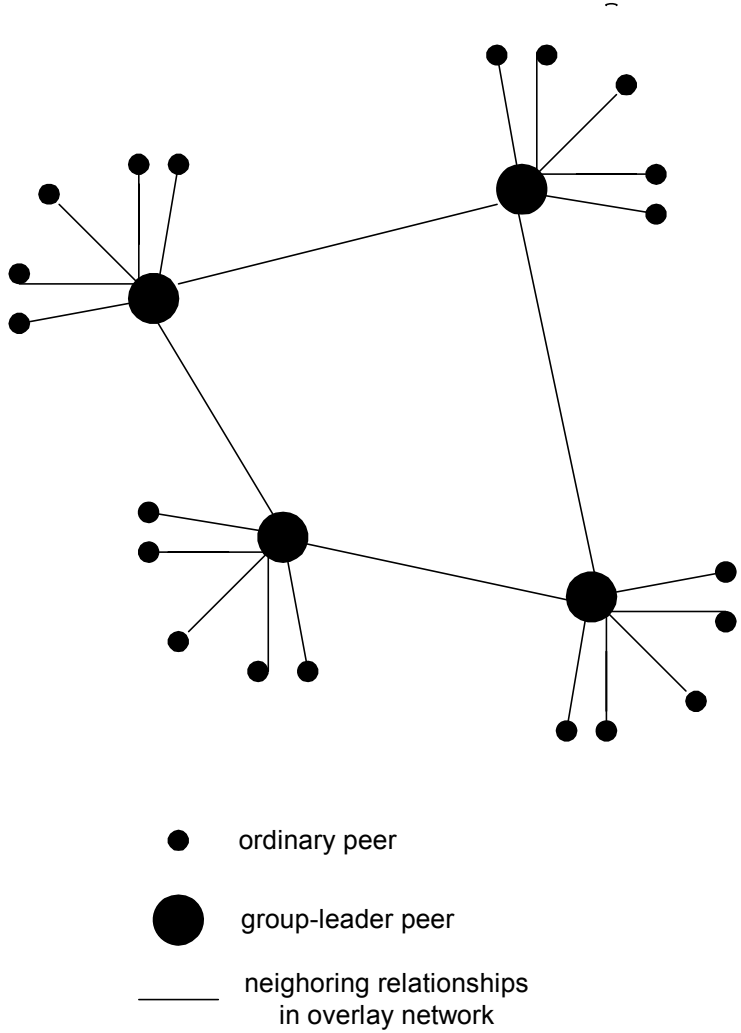
Gnutella: hội tụ Peer

1. Hội tụ peer X phải tìm một số peer khác trong Gnutella network: dùng một danh sách các peer dự tuyển
2. X lần lượt thử tạo TCP với các peer trên danh sách cho đến khi kết nối thiết lập được với Y
3. X gửi thông điệp Ping đến Y; Y chuyển thông điệp Ping.
4. Tất cả các peer nhận thông điệp Ping sẽ trả lời bằng thông điệp Pong
5. X nhận được nhiều thông điệp Pong. Nó sau đó có thể thiết lập thêm các kết nối TCP.

Peer leaving: xem một số vấn đề mạng gia đình!

KaZaA

- Mỗi peer là một hoặc được gán thành chỉ huy nhóm
  - TCP kết nối giữa peer và nhóm chỉ huy của nó
  - TCP kết nối giữa một số cặp nhóm chỉ huy.
- Nhóm chỉ huy theo dõi nội dung trong tất cả thành viên bên trong



KaZaA: Truy vấn

- Mỗi file có một băm (hash) hoặc một mô tả
- Client gửi từ khóa truy vấn đến nhóm chỉ huy của nó
- Nhóm chỉ huy đáp ứng với truy vấn của nhóm:
  - ứng với mỗi so trùng: metadata, hash, địa chỉ IP
- Nếu nhóm chỉ huy chuyển truy vấn cho nhóm khác, chúng phản hồi thích hợp
- Client sau đó chọn các file để download
  - Các yêu cầu HTTP dùng hash như một nhân dạng gửi đến cho peer quản lý file mong muốn

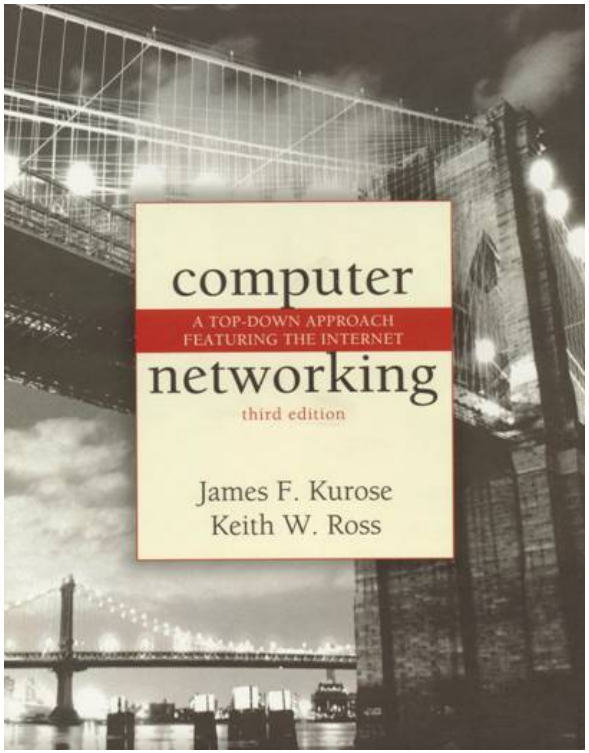
KaZaA: các thủ thuật

- Hạn chế khi tải lên đồng thời
- Xếp hàng yêu cầu
- Khích lệ ưu tiên
- tải xuống song song

Tìm hiểu thêm thông tin ở:

- J. Liang, R. Kumar, K. Ross, "Understanding KaZaA,"

(Web site: cis.poly.edu/~ross)



2.7 Lập trình socket với TCP

(xem thêm slide Lập trình socket)

Lập trình socket

Mục tiêu: nghiên cứu cách xây dựng ứng dụng truyền thông client/server dùng sockets

Socket API

- đã được giới thiệu trong BSD4.1 UNIX, 1981
- rõ ràng tại, sử dụng và giải phóng bởi ứng dụng
- mô hình client/server
- 2 kiểu dịch vụ lưu thông qua socket API:
  - datagram không tin cậy
  - tin cậy, truyền byte theo streaming

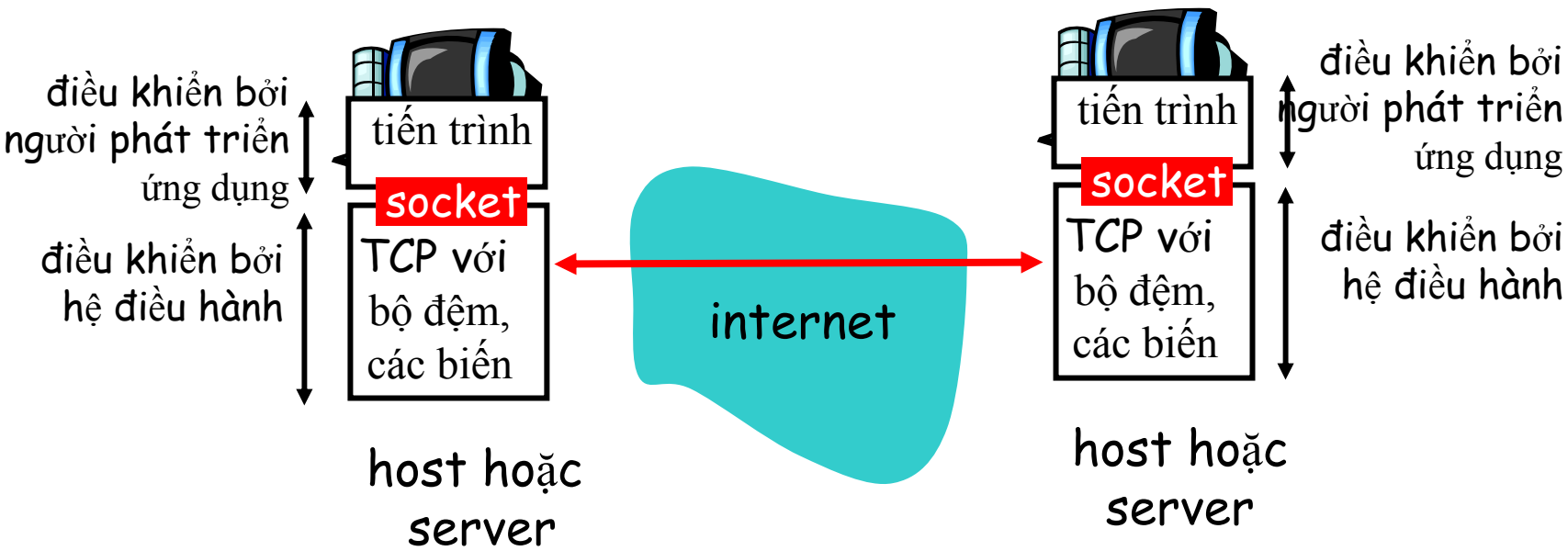
socket

một giao diện *host-cục bộ*, *tạo bởi ứng dụng*, *điều khiển bởi hệ điều hành*
interface (một "cửa")
trong đó tiến trình ứng dụng có thể gửi và nhận các thông điệp đến/từ các tiến trình khác

Lập trình socket dùng TCP

Socket: một cánh cửa giữa tiến trình ứng dụng và giao thức transport (UCP hoặc TCP)

Dịch vụ TCP: truyền tin cậy các **bytes** từ một tiến trình đến tiến trình khác



Lập trình socket với TCP

Client phải tiếp xúc với server

- tiến trình server phải chạy trước
- server phải tạo socket (cửa) mời client đến tiếp xúc

Client tiếp xúc server bằng:

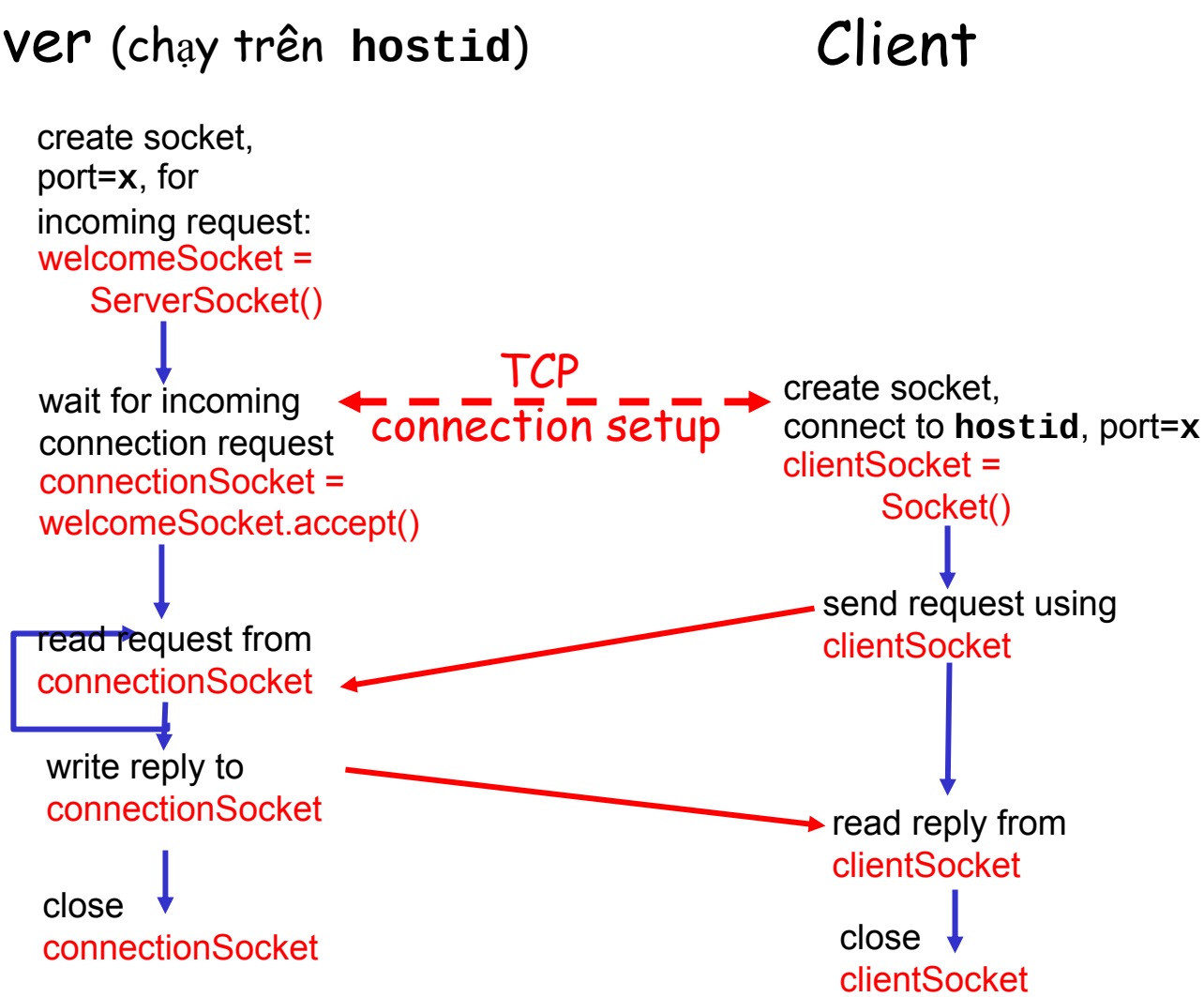
- tạo socketTCP client cục bộ
- xác định địa chỉ IP, số port của tiến trình server
- Khi client tạo socket: client TCP thiết lập kết nối với server TCP

- Khi đã được tiếp xúc bởi client, server TCP tạo socket mới cho tiến trình server để truyền thông với client
  - cho phép server "nói chuyện" với nhiều client
  - số port dùng để phân biệt các client (xem tiếp trong chương 3)

Nhìn dưới góc độ ứng dụng

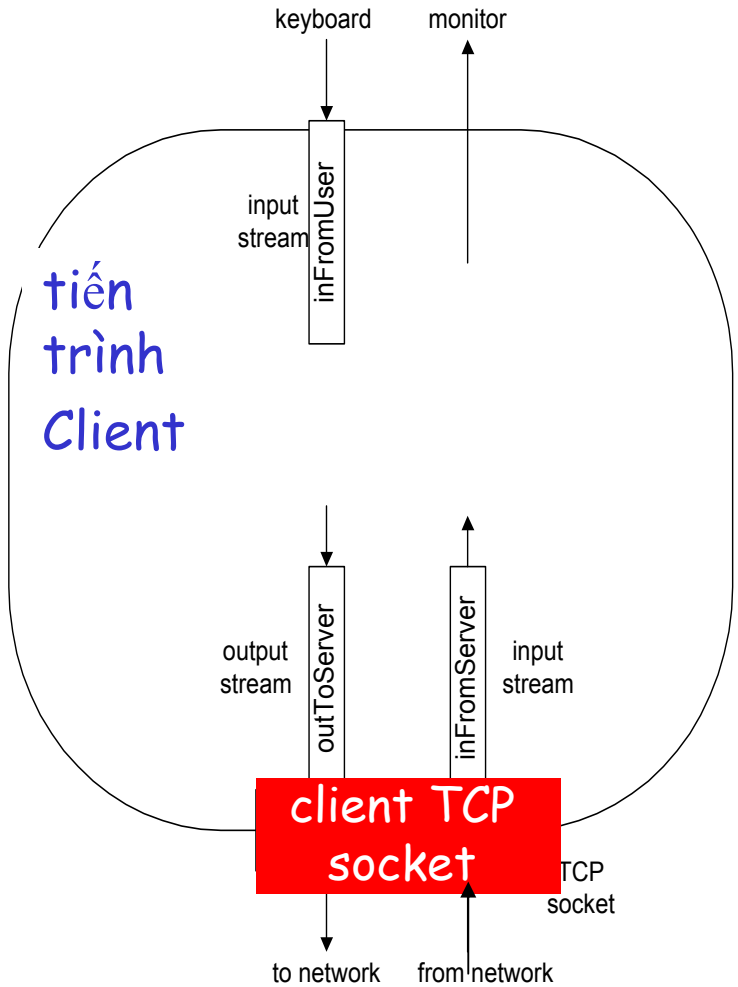
- *TCP cung cấp việc truyền các byte tin cậy và theo thứ tự giữa client và server*

Giao tiếp socket Client/server: TCP



Thuật ngữ Stream

- Một stream là một chuỗi các ký tự được "chảy" vào hoặc ra khỏi một tiến trình
- Một input stream để chỉ nguồn vào của một tiến trình, vd: bàn phím hoặc socket.
- Một output stream để chỉ nguồn ra của một tiến trình, vd: màn hình hoặc socket.



Lập trình socket dùng TCP

Ví dụ ứng dụng client-server :

1) client đọc các dòng từ input chuẩn (**inFromUser** stream) , gửi đến server thông qua socket (**outToServer** stream)
2) server đọc các dòng từ socket
3) server chuyển các dòng thành chữ hoa, gửi ngược trở lại cho client
4) client đọc, in các dòng đã sửa đổi từ socket (**inFromServer** stream)

Ví dụ: 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());
```

tạo input stream

tạo client socket, kết nối vào server

tạo output stream gắn vào socket

Ví dụ: Java client (TCP)

tạo input stream gắn vào socket

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

sentence = inFromUser.readLine();

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

modifiedSentence = inFromServer.readLine();

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

clientSocket.close();

    }
}
```

Gửi dòng đến server

đọc dòng từ server

Ví dụ: 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()));
```

tạo socket mời tiếp xúc tại port 6789

Chờ, client tiếp cận với server

Tạo input stream, gắn vào socket

Ví dụ: Java server (TCP)

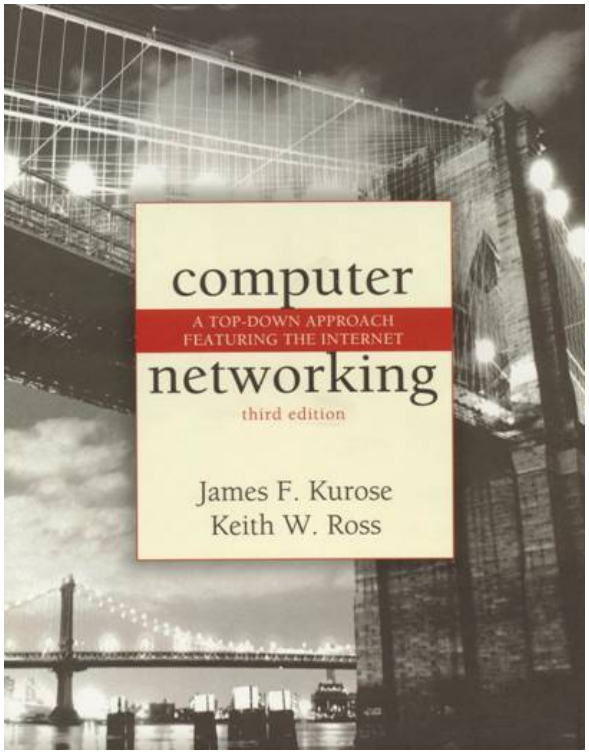
Tạo output stream, gắn vào socket

Đọc dòng từ socket

Viết dòng ra từ socket

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

clientSentence = inFromClient.readLine();

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

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

kết thúc vòng lặp while quay lại và chờ cho kết nối của client khác



2.8 Lập trình socket với UDP

(xem thêm slide Lập trình socket)

Lập trình socket *với UDP*

UDP: không "kết nối" giữa client và server

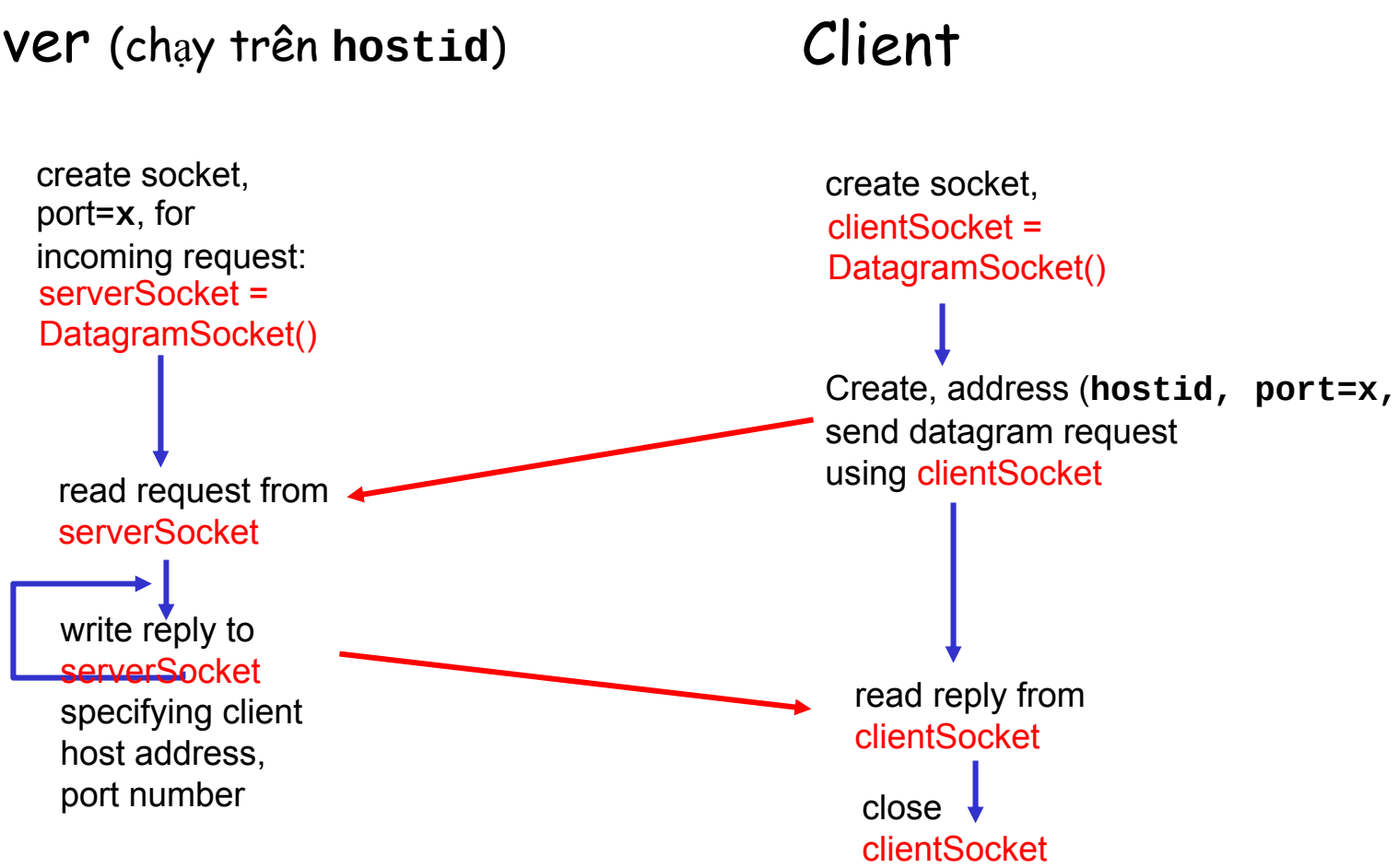
- không bắt tay
- người gửi rõ ràng gắn địa chỉ IP và port của đích vào mỗi gói
- phải trích địa chỉ IP, port của người gửi từ gói đã nhận

UDP: dữ liệu truyền có thể không theo thứ tự, hoặc bị mất mát

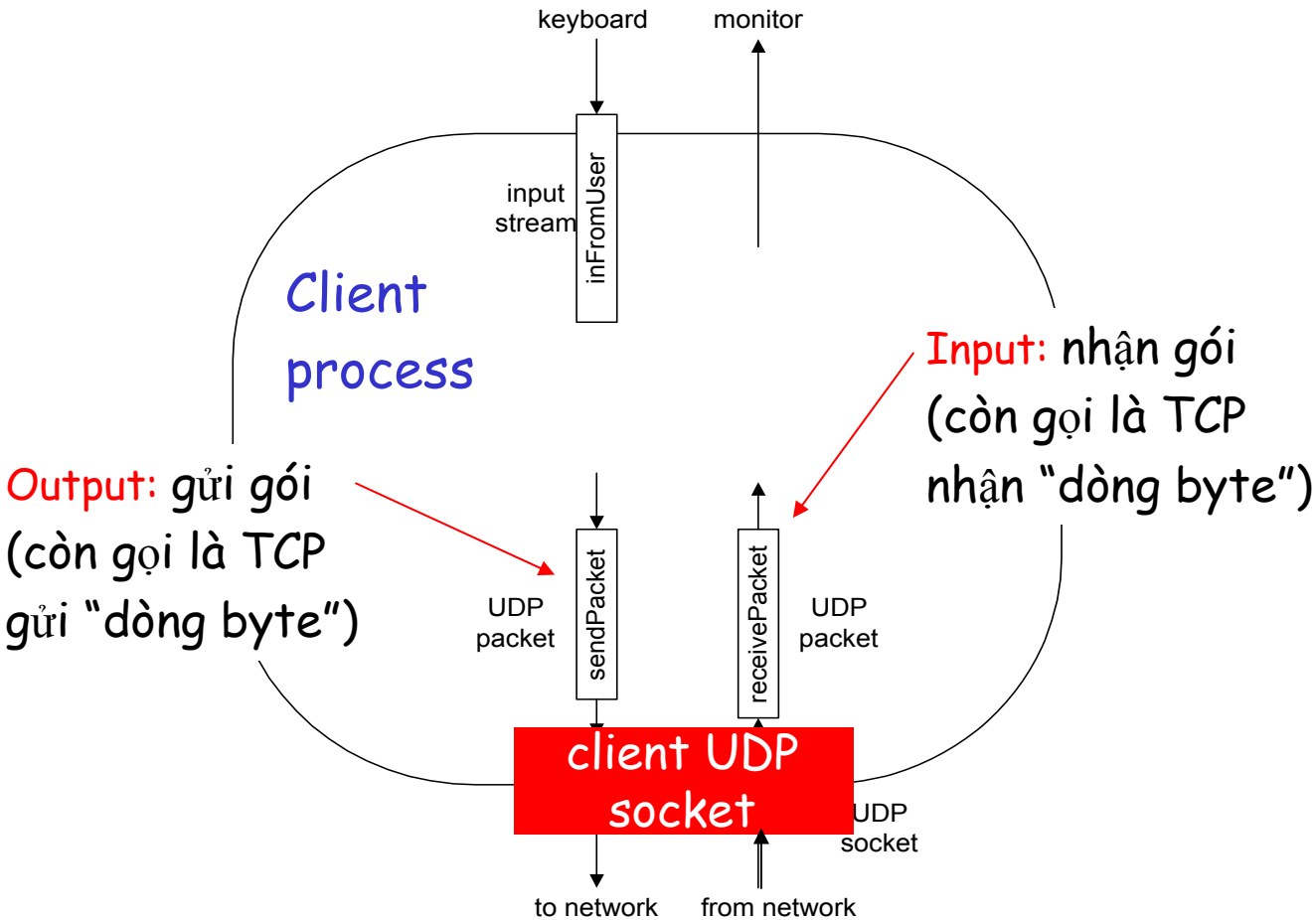
góc nhìn ứng dụng

UDP cung cấp việc truyền không tin cậy một nhóm các byte ("datagrams") giữa client và server

Giao tiếp socket Client/server: UDP



Ví dụ: Java client (UDP)



Ví dụ: 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();
```

tạo input stream

tạo client socket

dịch hostname thành địa chỉ IP dùng DNS

Ví dụ: Java client (UDP).

tạo datagram với dữ liệu để gửi, độ dài, địa chỉ IP, port

gửi datagram đến server

đọc datagram từ 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();
}

}
```

Ví dụ: Java server (UDP)

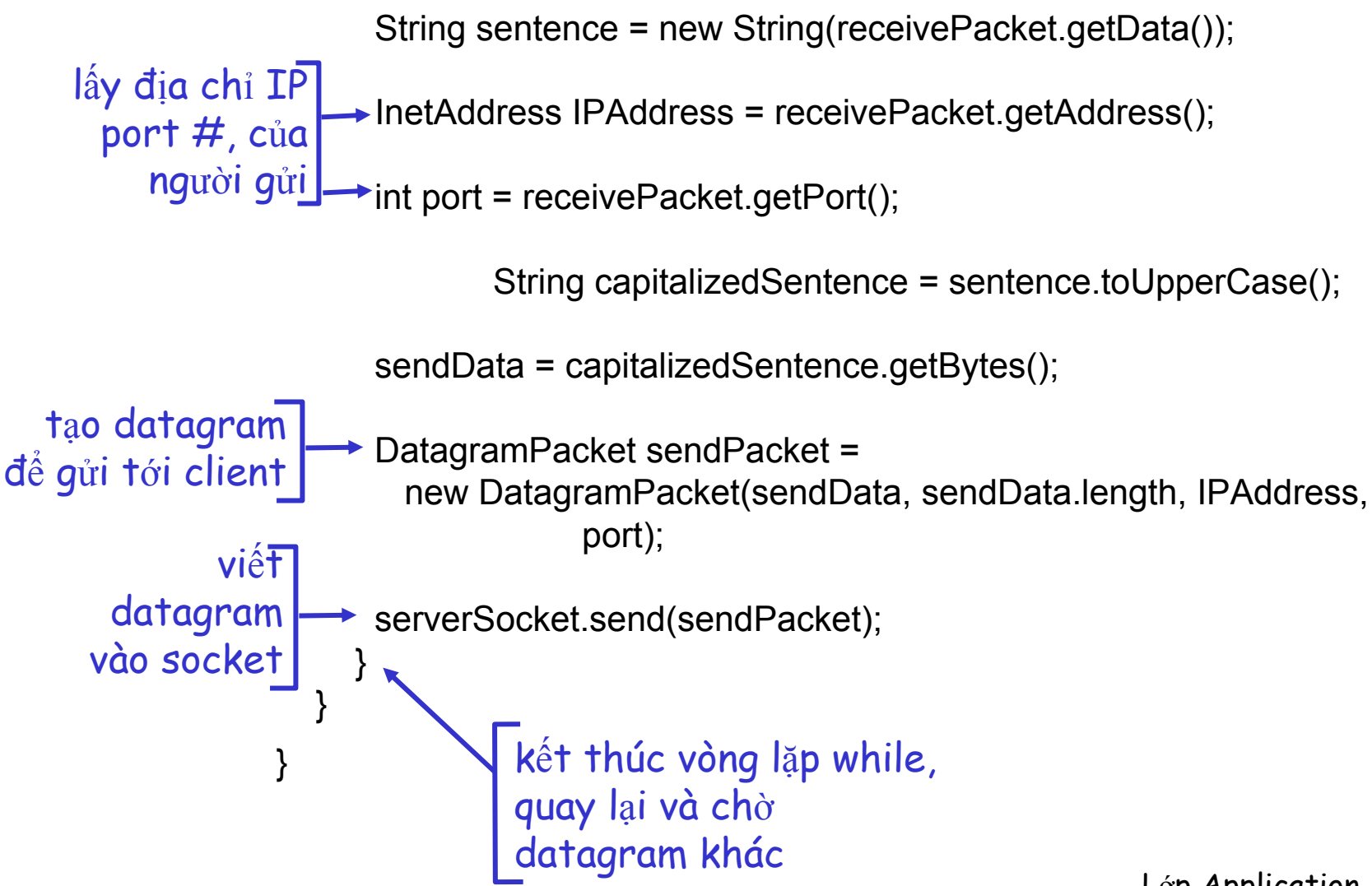
```
import java.io.*;
import java.net.*;

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);
```

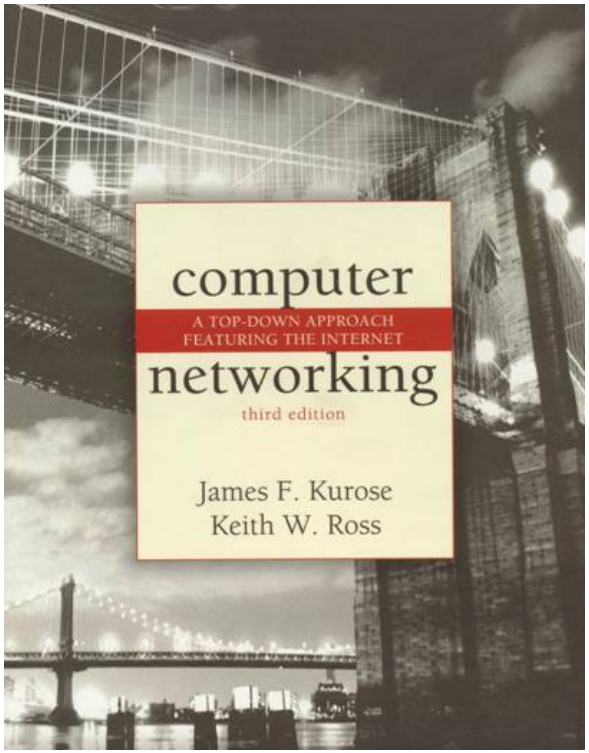
Tạo datagram socket tại port 9876

Tạo không gian để nhận datagram

nhận datagram

Ví dụ: Java server (UDP)





2.9 Xây dựng một Web server

Xây dựng 1 Web server đơn giản

- quản lý 1 yêu cầu HTTP
- chấp nhận yêu cầu
- phân tích cú pháp header
- lấy file được yêu cầu từ hệ thống file của server
- tạo thông điệp đáp ứng HTTP:
  - các dòng header + file
- gửi đáp ứng đến client

- sau khi tạo server, bạn có thể yêu cầu file dùng trình duyệt (ví dụ: IE)
- xem giáo trình để biết thêm chi tiết

Chương 2: Tổng kết

Tổng quan một số vấn đề về ứng dụng mạng!

- Các kiến trúc ứng dụng
  - client-server
  - P2P
  - lai
- các yêu cầu dịch vụ:
  - tin cậy, bandwidth, trễ
- mô hình dịch vụ Internet transport
  - connection-oriented, tin cậy: TCP
  - không tin cậy, datagrams: UDP
- các giao thức đặc biệt:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
- lập trình socket

Chương 2: Tổng kết

Phần quan trọng: nghiên cứu về *các giao thức*

- trao đổi thông điệp yêu cầu/trả lời điển hình:
  - client yêu cầu thông tin hoặc dịch vụ service
  - server đáp ứng với dữ liệu, mã trạng thái
- các dạng thông điệp:
  - headers: các trường cho biết thông tin về dữ liệu
  - dữ liệu: thông tin để truyền thông
- điều khiển với các thông điệp dữ liệu
  - in-band, out-of-band
- tập trung và không tập trung
- không trạng thái và có trạng thái
- truyền thông điệp tin cậy và không tin cậy
- "sự phức tạp của các vấn đề liên quan đến mạng"