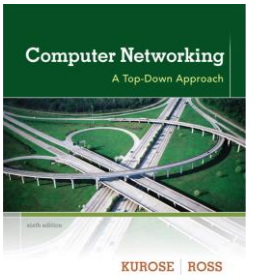
Chương 2
Tầng ứng dụng



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

Người dịch: Nguyễn Thanh Thủy

Tài liệu được dịch cho mục đích giảng dạy (được sự đồng ý của tác giả).

Chương 2: Tầng ứng dụng

Mục tiêu:

- Khái niệm, các vấn đề cài đặt giao thức ứng dụng mạng
  - Các mô hình dịch vụ tầng giao vận
  - Mô hình khách-chủ (client-server)
  - Mô hình điểm-điểm (peer-to-peer)
- Nghiên cứu một số giao thức tầng ứng dụng
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- Tạo một ứng dụng mạng
  - socket API

Chương 2: Nội dung

2.1 Nguyên lý của ứng dụng mạng
- Kiến trúc của ứng dụng
- Các yêu cầu của ứng dụng

2.2 Web và HTTP

2.3 FTP

2.4 Thư điện tử
- SMTP, POP3, IMAP

2.5 DNS

2.6 Ứng dụng P2P

2.7 Lập trình socket với UDP và TCP

Một số ứng dụng mạng

- Thư điện tử (e-mail)
- web
- Tin nhắn văn bản (text messaging)
- Truy nhập từ xa (remote login)
- Chia sẻ file P2P
- Trò chơi nhiều người trên mạng
- streaming video (YouTube, Hulu, Netflix)
- Điện thoại Internet (voice over IP) (ví dụ Skype)
- Hội thảo video thời gian thực
- Mạng xã hội
- Các ứng dụng tìm kiếm
- …
- …

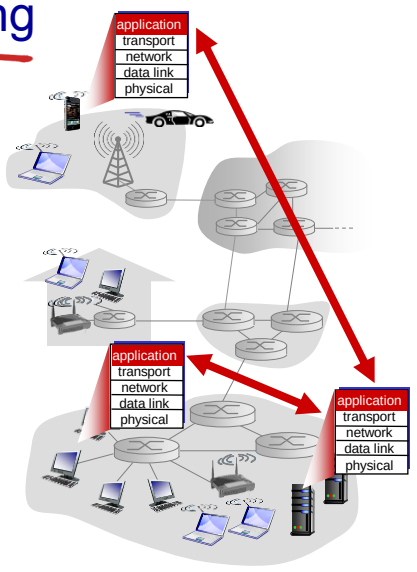
Tạo một ứng dụng mạng

Viết chương trình để:

- Chạy trên các *hệ thống đầu cuối* khác nhau
- Truyền thông qua mạng
- Ví dụ: phần mềm máy chủ web (web server) truyền thông với phần mềm trình duyệt (browser software)

Không cần viết chương trình ứng dụng cho các thiết bị trong phần lõi của mạng

- Các thiết bị trong phần lõi của mạng không chạy các ứng dụng người dùng.
- Các ứng dụng chạy trên thiết bị đầu cuối cho phép phát triển và phổ biến ứng dụng một cách nhanh chóng.

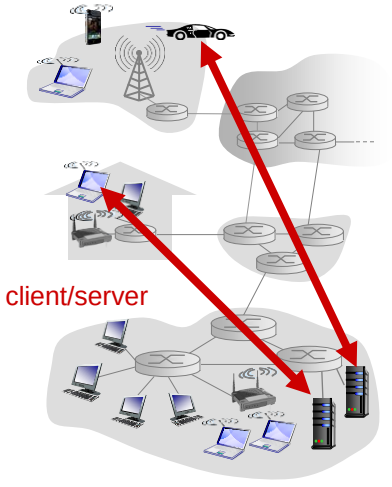


Kiến trúc của ứng dụng

Các ứng dụng có thể có kiến trúc dạng:

- Client-server (khách-chủ)
- Peer-to-peer (P2P, ngang hàng)

Kiến trúc client-server



server:

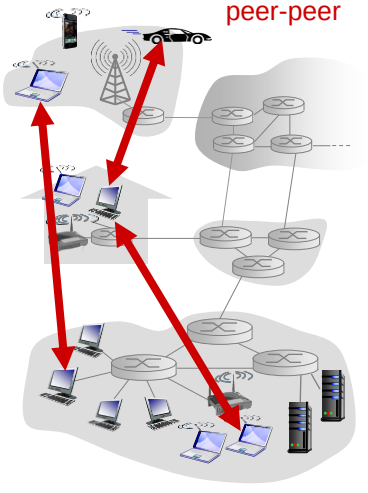
- Là host luôn hoạt động
- Có địa chỉ IP cố định
- Các trung tâm dữ liệu

client:

- Truyền thông với server
- Có thể được kết nối liên tục vào mạng hoặc không
- Có thể có địa chỉ IP thay đổi
- Không truyền thông trực tiếp với client khác

Kiến trúc P2P

- *Không có* server luôn hoạt động
- Các hệ thống đầu cuối (peer) truyền thông trực tiếp với nhau
- Mỗi peer yêu cầu dịch vụ từ một peer nào đó, và cung cấp dịch vụ lại cho các peer khác.
  - *Có khả năng tự mở rộng* – peer mới mang lại khả năng dịch vụ mới, cũng như có những yêu cầu dịch vụ mới.
- Các peer không kết nối liên tục và có thể thay đổi địa chỉ IP
  - Quản lý phức tạp



Tiến trình truyền thông

Tiến trình: chương trình chạy trên một host

- Trên cùng một host, hai tiến trình truyền thông với nhau qua truyền thông tiến trình nội bộ (được xác định bởi hệ điều hành)
- Các tiến trình trên các host khác nhau truyền thông với nhau bằng cách trao đổi các thông điệp

client, server

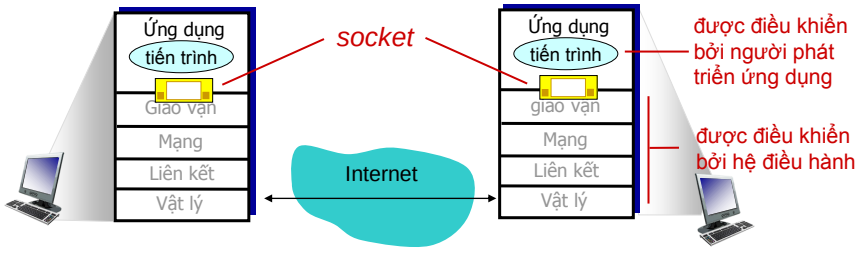
Tiến trình client: tiến trình khởi tạo truyền thông

Tiến trình server: tiến trình chờ được tiếp xúc

- Lưu ý: ứng dụng theo kiến trúc P2P có cả tiến trình client & tiến trình server

Socket

- Các tiến trình gửi/nhận các thông điệp đến/từ socket của nó
- Socket tương tự như một cửa vào/ra
  - Tiến trình gửi đẩy thông điệp ra bên ngoài cửa
  - Tiến trình gửi dựa trên cơ sở hạ tầng tầng giao vận ở phía bên kia cửa để truyền thông điệp đến socket của tiến trình nhận



Định địa chỉ cho tiến trình

- Để nhận các thông điệp, tiến trình phải có *định danh (identifier)*
- Thiết bị host phải có địa chỉ IP 32-bit duy nhất
- *Hỏi:* Địa chỉ IP của host mà trên đó tiến trình chạy có đủ để xác định định danh của tiến trình không?
  - *Trả lời:* Không, *do nhiều* tiến trình có thể cùng chạy trên một host.

- *Định danh* bao gồm cả địa chỉ IP và số hiệu cổng được kết hợp với tiến tình trên host.
- Ví dụ các số hiệu cổng:
  - HTTP server: 80
  - mail server: 25
- Để gửi thông điệp HTTP tới web server gaia.cs.umass.edu:
  - Địa chỉ IP: 128.119.245.12
  - Số hiệu cổng: 80

Các giao thức tầng ứng dụng

- Các loại thông điệp được trao đổi
  - Ví dụ: thông điệp yêu cầu (request), thông điệp đáp ứng (response)
- Cú pháp của thông điệp:
  - Các trường trong thông điệp & mô tả
- Ngữ nghĩa của thông điệp
  - Ý nghĩa thông tin của các trường
- Quy tắc về thời gian và cách thức các tiến trình gửi và đáp ứng thông điệp

Các giao thức mở (công khai):

- Được định nghĩa trong RFCs
- Cho phép tương tác
- Ví dụ: HTTP, SMTP

Các giao thức riêng (độc quyền):

- Ví dụ: Skype

Một ứng dụng cần những dịch vụ giao vận nào?

Toàn vẹn dữ liệu

- Một số ứng dụng (ví dụ: truyền file, web) yêu cầu truyền tin cậy 100%
- Các ứng dụng khác (ví dụ: audio) có thể chịu một số mất mát

Thời gian thực

- Một số ứng dụng (ví dụ: điện thoại Internet, trò chơi tương tác) yêu cầu độ trễ thấp mới "hiệu quả"

Băng thông

- Một số ứng dụng (ví dụ: đa phương tiện) yêu cầu băng thông tối thiểu để đảm bảo "hiệu quả".
- Các ứng dụng khác ("các ứng dụng mềm dẻo") có thể dùng băng thông nào cũng được

An toàn bảo mật

- Mã hóa, đảm bảo toàn vẹn dữ liệu, …

Một số yêu cầu dịch vụ giao vận với các ứng dụng phổ biến

Ứng dụng	Mất mát dữ liệu	Thông lượng	Thời gian thực
Truyền file	không	mềm dẻo	không
Thư điện tử	không	mềm dẻo	không
Web	không	mềm dẻo	không
Audio/video thời gian thực	chịu lỗi	audio: 5kbps-1Mbps video:10kbps-5Mbps	có, 100 msec
Lưu trữ audio/video	chịu lỗi	như trên	có, vài giây
Trờ chơi tương tác	chịu lỗi	một vài kbps	có, 100 msec
Tin nhắn nhanh	không	mềm dẻo	có và không

Dịch vụ giao thức tầng giao vận của Internet

Dịch vụ TCP:

- *Truyền tin cậy* giữa tiến trình gửi và tiến trình nhận
- *Điều khiển luồng:* bên gửi không lấn át bên nhận
- *Điều khiển tắc nghẽn:* điều tiết bên gửi khi mạng bị quá tải
- *Không cung cấp:* thời gian thực, đảm bảo băng thông tối thiểu, an toàn bảo mật
- *Hướng kết nối:* yêu cầu thiết lập kết nối giữa tiến trình client và tiến trình server

Dịch vụ UDP:

- *Truyền dữ liệu không tin cậy* giữa tiến trình gửi và tiến trình nhận
- *Không cung cấp:* truyền tin cậy, điều khiển luồng, điều khiển tắc nghẽn, thời gian thực, đảm bảo băng thông, an toàn bảo mật, hoặc thiết lập kết nối

Hỏi: Tại sao lại dùng cả hai dịch vụ? Dùng UDP để làm gì?

Các ứng dụng Internet: các giao thức tầng ứng dụng và giao vận

Ứng dụng	Giao thức tầng ứng dụng	Giao thức tầng giao vận
Thư điện tử	SMTP [RFC 2821]	TCP
Truy nhập từ xa	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Truyền file	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (ví dụ: YouTube), RTP [RFC 1889]	TCP hoặc UDP
Điện thoại Internet	SIP, RTP, giao thức độc quyền (ví dụ: Skype)	TCP hoặc UDP

Bảo mật TCP

TCP & UDP

- Không mã hóa
- Mật khẩu dạng bản rõ được gửi vào socket để truyền trên Internet theo dạng bản rõ

SSL (Secure Sockets Layer)

- Hỗ trợ kết nối TCP được mã hóa
- Toàn vẹn dữ liệu
- Xác thực thiết bị đầu cuối

SSL là bảo mật tại tầng ứng dụng

- Ứng dụng dùng các thư viện của SSL để "nói" với TCP

Socket API của SSL

- Mật khẩu dạng bản rõ được gửi vào socket để truyền trên Internet theo dạng đã được mã hóa

Chương 2: Nội dung

Web và HTTP

Một số thuật ngữ:

- *Trang web* bao gồm một số *đối tượng (object)*
- Đối tượng có thể là tệp HTML, ảnh JPEG, Java applet, tệp audio,…
- Trang web bao gồm *tệp HTML cơ bản* chứa *một số đối tượng được tham chiếu*
- Mỗi đối tượng được định địa chỉ bởi một *URL,* ví dụ:

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

Tên host | Tên đường dẫn

Khái quát về HTTP

HTTP: hypertext transfer protocol

- Giao thức tầng ứng dụng của Web
- Mô hình client/server
  - *client:* trình duyệt (browser) yêu cầu, nhận (sử dụng giao thức HTTP), và "hiển thị" các đối tượng Web
  - *server:* Máy chủ web (web server) gửi (sử dụng giao thức HTTP) các đối tượng đáp ứng cho yêu cầu.

Khái quát về HTTP

Dùng TCP:

- Client khởi tạo kết nối TCP (tạo socket) tới server, cổng 80
- Server chấp nhận kết nối TCP từ client
- Thông điệp HTTP (thông điệp giao thức tầng ứng dụng) được trao đổi giữa trình duyệt (HTTP client) và máy chủ Web (HTTP server)
- Đóng kết nối TCP

HTTP là "không trạng thái"

- Server không lưu giữ thông tin về những yêu cầu trước đó của client

Vấn đề liên quan

Giao thức lưu giữ "trạng thái" khá phức tạp!

- Lịch sử quá khứ (trạng thái) cần phải được lưu giữ
- Nếu server/client bị sự cố, thì quan điểm về "trạng thái" của chúng có thể không nhất quán, cần phải được hòa giải

Kết nối HTTP

HTTP không bền vững

- Chỉ có tối đa một đối tượng được gửi qua một kết nối TCP
  - Kết nối sau đó sẽ được đóng lại
- Việc tải về nhiều đối tượng sẽ yêu cầu nhiều kết nối

HTTP bền vững

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

HTTP không bền vững

Giả sử người dùng gõ URL:
`www.someSchool.edu/someDepartment/home.index`

(chứa văn bản, tham chiếu tới 10 ảnh jpeg)

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

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

2. HTTP client gửi *thông điệp HTTP yêu cầu* (chứa URL) vào trong socket kết nối TCP. Thông điệp chỉ ra là client muốn lấy 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 tệp HTML, hiển thị HTML. Phân tích tệp HTML, tìm 10 đối tượng jpeg được tham chiếu

6. Các bước từ 1 đến 5 được lặp lại cho từng đối tượng trong 10 đối tượng jpeg

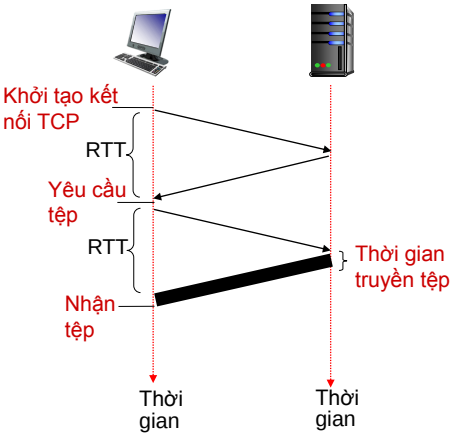
thời gian

HTTP không bền vững: thời gian đáp ứng

RTT (định nghĩa): thời gian để một gói tin nhỏ đi từ client đến server và quay lại.

Thời gian đáp ứng HTTP:

- Một RTT để khởi tạo kết nối TCP
- Một RTT để gửi HTTP yêu cầu và một vài byte HTTP đáp ứng được trả về
- Thời gian truyền tệp
- Thời gian đáp ứng của HTTP không bền vững = 2RTT+ thời gian truyền tệp



HTTP bền vững

Vấn đề với HTTP không bền vững:

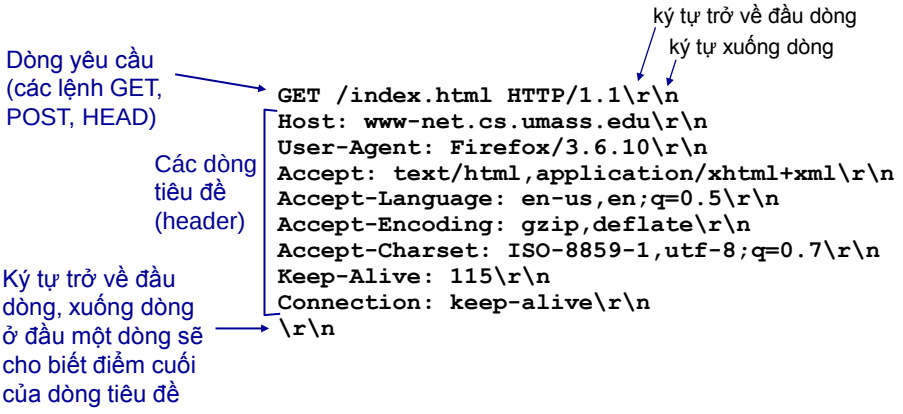
- Cần 2 RTT cho mỗi đối tượng
- Hệ điều hành liên quan đến *mỗi* kết nối TCP
- Các trình duyệt thường mở nhiều kết nối TCP song song để lấy các đối tượng được tham chiếu

HTTP bền vững:

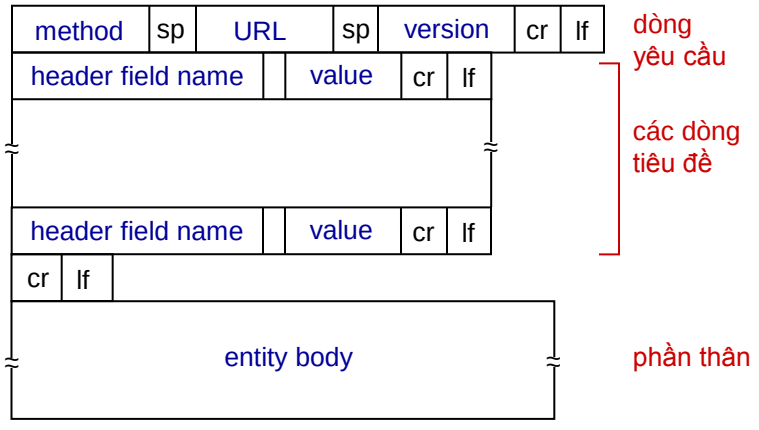
- Server để mở kết nối sau khi gửi đáp ứng
- Chuỗi các thông điệp HTTP tiếp theo giữa cùng client/server sẽ được gửi thông qua kết nối mở này
- Client gửi yêu cầu ngay sau khi gặp một đối tượng được tham chiếu
- Ít nhất là một RTT cho tất cả các đối tượng được tham chiếu

Thông điệp yêu cầu HTTP

- Có hai loại thông điệp HTTP: *yêu cầu và đáp ứng*
- Thông điệp HTTP yêu cầu:
  - ASCII (định dạng con người có thể đọc được)



Thông điệp HTTP yêu cầu: định dạng tổng quát



Tải lên form input

Phương pháp POST:

- Trang web thường chứa form input
- Đầu vào (input) được tải lên server trong phần thân thực thể (entity body)

Phương pháp URL:

- Dùng phương thức GET
- Đầu vào được tải lên trong trường URL của dòng yêu cầu:

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


Các loại phương thức

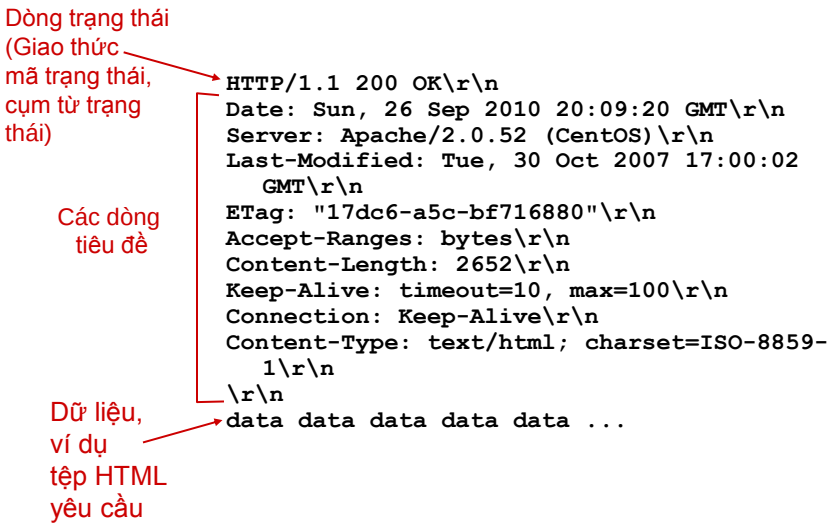
HTTP/1.0:

- GET
- POST
- HEAD
  - Yêu cầu server dời đối tượng được yêu cầu ra khỏi đáp ứng

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - Tải tệp lên trong phần thân thực thể tới đường dẫn được xác định trong trường URL
- DELETE
  - Xóa tệp được xác định trong trường URL

Thông điệp đáp ứng HTTP



Các mã trạng thái đáp ứng HTTP

- Mã trạng thái xuất hiện ngay trong dòng đầu tiên của thông điệp đáp ứng từ server đến client.
- Ví dụ một số mã:

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 bị di chuyển, vị trí mới được xác định ở phía sau thông điệp này (Location:)

400 Bad Request
- Server không hiểu thông điệp yêu cầu

404 Not Found
- Tài liệu yêu cầu không có trong server này

505 HTTP Version Not Supported

Tự kiểm tra HTTP (phía client)

1. Telnet đến Web server ưa thích của bạn:

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

Mở kết nối TCP ở cổng 80 (cổng mặc định của HTTP server) tại cis.poly.edu. Nhập yêu cầu gì đó và gửi tới cổng 80 tại cis.poly.edu

2. Nhập yêu cầu trong lệnh GET HTTP:

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

Bằng cách gõ lệnh này (nhấn enter 2 lần), bạn đã gửi yêu cầu GET tối thiểu (nhưng đầy đủ) tới HTTP server

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

(hoặc dùng Wireshark để xem thông điệp yêu cầu/đáp ứng HTTP bắt được)

Trạng thái User-server: các cookie

Nhiều trang Web sử dụng các cookie

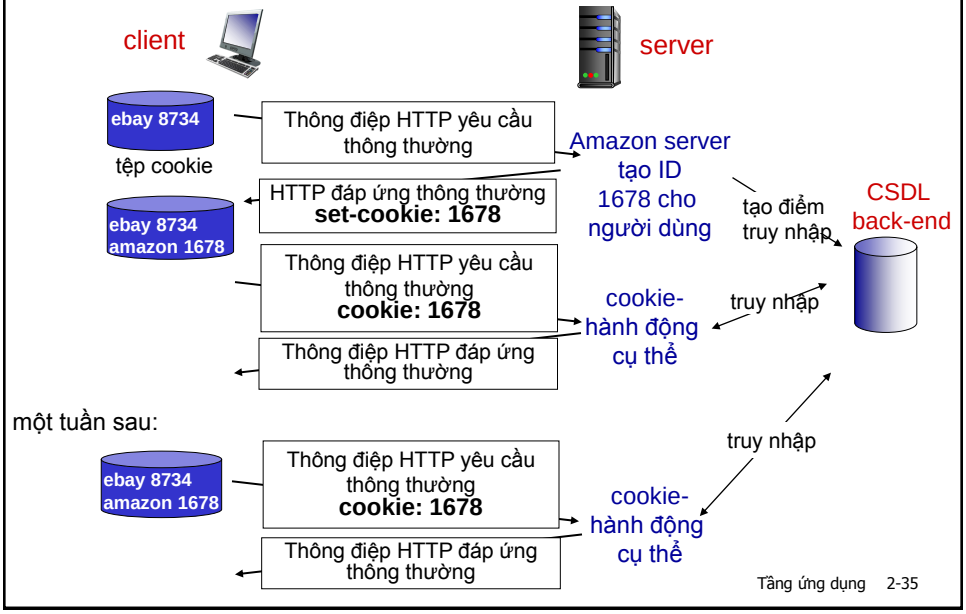
Bốn thành phần:

1) Dòng tiêu đề cookie của thông điệp HTTP *đáp ứng*
2) Dòng tiêu đề cookie trong thông điệp HTTP *yêu cầu* tiếp theo
3) Tệp cookie được lưu giữ trên host của người dùng, được quản lý bởi trình duyệt của người dùng.
4) Cơ sở dữ liệu back-end tại Web site

Ví dụ:

- Susan thường truy nhập Internet từ một PC
- Cô ấy vào một trang thương mại điện tử lần đầu tiên
- Khi yêu cầu HTTP khởi tạo đi đến trang, trang sẽ tạo ra:
  - Một ID duy nhất
  - Điểm truy nhập vào cơ sở dữ liệu back-end cho ID

Các cookie: lưu giữ "trạng thái" (tiếp)



Các cookie (tiếp)

Các cookie có thể được dùng cho những việc gì?

- Cấp phép
- Giỏ mua hàng
- Khuyến nghị
- Trạng thái phiên làm việc của người dùng (Web e-mail)

Ngoài ra

cookie và sự riêng tư:

- Cookie cho phép các site biết nhiều hơn về người dùng
- Người dùng có thể cung cấp tên và e-mail cho các site

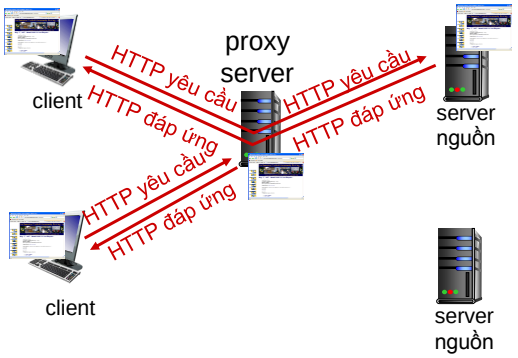
Làm cách nào để lưu giữ "trạng thái":

- Các điểm cuối giao thức: duy trì trạng thái tại bên gửi/bên nhận thông qua nhiều giao dịch
- Các cookie: các thông điệp HTTP mang trạng thái

Web caches (proxy server)

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

- Người dùng thiết lập trình duyệt: truy nhập Web qua vùng nhớ đệm (cache)
- Trình duyệt gửi tất cả các yêu cầu HTTP tới cache
  - Nếu đối tượng có trong cache: cache sẽ trả về đối tượng
  - Ngược lại, cache sẽ yêu cầu đối tượng từ server nguồn, sau đó sẽ trả đối tượng về cho client



Web caching

- Bộ nhớ cache hoạt động ở cả client và server
  - Server đáp ứng cho yêu cầu đầu tiên từ một client
  - Client gửi yêu cầu tới server gốc
- Cache thường được cài đặt bởi ISP (trường học, công ty, khu dân cư)

Tại sao lại cần Web caching?

- Làm giảm thời gian đáp ứng cho yêu cầu của client
- Làm giảm lưu lượng trên một liên kết truy nhập của tổ chức
- Nếu Internet thực hiện cache nhiều thì sẽ làm cho các nhà cung cấp nội dung "nghèo nàn" phân phối nhiều nội dung đó (cũng tương tự như chia sẻ file P2P).

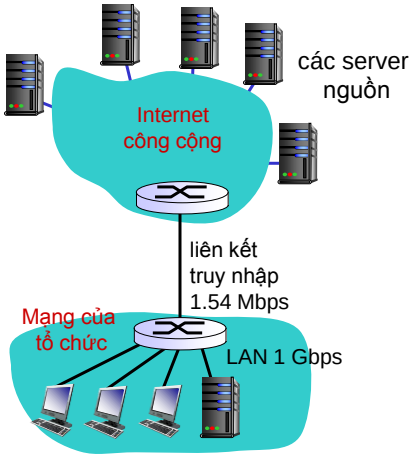
Ví dụ caching

Giả thiết:

- Kích thước của đối tượng trung bình: 100K bits
- Tốc độ yêu cầu trung bình từ các trình duyệt tới server nguồn: 15/giây
- Tốc độ dữ liệu trung bình tới trình duyệt: 1.50 Mbps
- RTT từ bộ định tuyến của tổ chức đến server nguồn bất kỳ: 2 giây
- Tốc độ của liên kết truy nhập: 1.54 Mbps

Kết quả:

- Độ khả dụng của LAN: 15%
- Độ khả dụng của liên kết truy nhập = 99% *Vấn đề!*
- Trễ tổng = trễ của Internet + trễ truy nhập + trễ của LAN
  = 2 giây + một số phút + (~ 1 giây)



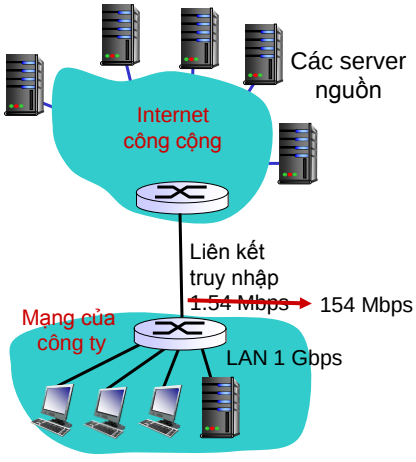
Ví dụ caching: tăng tốc độ của liên kết truy nhập lên

Giả thiết:

- Kích thước của đối tượng trung bình: 100K bits
- Tốc độ yêu cầu trung bình từ các trình duyệt tới server nguồn: 15/giây
- Tốc độ dữ liệu trung bình tới trình duyệt: 1.50 Mbps
- RTT từ bộ định tuyến của tổ chức đến server nguồn bất kỳ: 2 giây
- Tốc độ của liên kết truy nhập: ~~1.54 Mbps~~ **154 Mbps**

Kết quả:

- Độ khả dụng của LAN: 15%
- Độ khả dụng của liên kết truy nhập = ~~99%~~ **9.9%**
- Tổng trễ = trễ của Internet + trễ truy nhập + trễ của LAN **Một số giây**
  = 2 giây + ~~một số phút~~ + (~ 1 giây)



Chi phí: **để làm tăng tốc độ liên kết truy nhập (không hề rẻ!)**

Ví dụ caching: cài đặt cache cục bộ

Giả thiết:

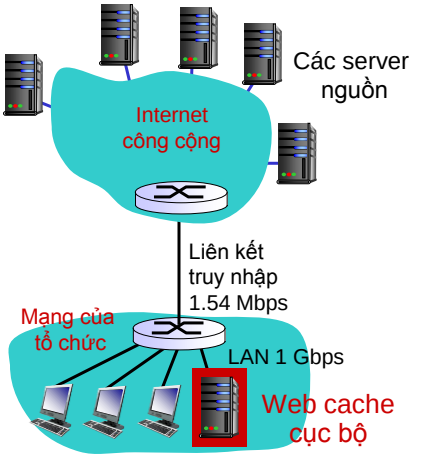
- Kích thước của đối tượng trung bình: 100K bits
- Tốc độ yêu cầu trung bình từ các trình duyệt tới server nguồn: 15/giây
- Tốc độ dữ liệu trung bình tới trình duyệt: 1.50 Mbps
- RTT từ bộ định tuyến của tổ chức đến server nguồn bất kỳ: 2 giây
- Tốc độ của liên kết truy nhập: 1.54 Mbps

Kết quả:

- Độ khả dụng của LAN: 15%
- Độ khả dụng của liên kết truy nhập = ?
- Tổng trễ = ?

Tính độ khả dụng của liên kết và trễ như thế nào?

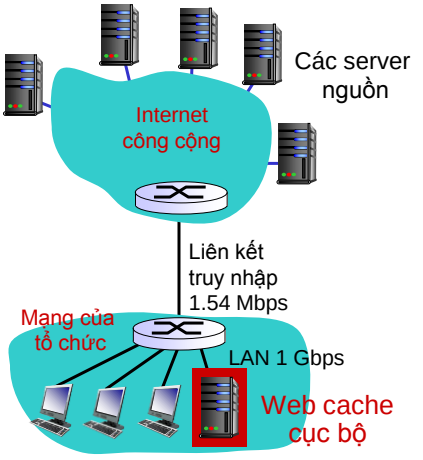
Chi phí: **web cache (rất rẻ!)**



Ví dụ caching: cài đặt cache cục bộ

Tính toán độ khả dụng của liên kết truy nhập, độ trễ khi dùng cache:

- Giả sử tỷ lệ hỗ trợ của cache là 0.4
  - 40% yêu cầu được thỏa mãn tại cache, 60% yêu cầu được thỏa mãn tại nguồn
- Độ khả dụng của liên kết:
  - 60% yêu cầu dùng liên kết truy nhập
- Tốc độ dữ liệu tới các trình duyệt qua liên kết truy nhập = 0.6*1.50 Mbps = 0.9 Mbps
  - Độ khả dụng = 0.9/1.54 = 0.58
- Tổng trễ:
  - = 0.6 * (trễ từ các server nguồn) +0.4 * (trễ khi được thỏa mãn tại cache)
    = 0.6 * (2.01s) + 0.4 * (0.01s)
    = 1.21s
  - Ít hơn so với liên kết 154 Mbps (và chi phí rẻ hơn rất nhiều!)

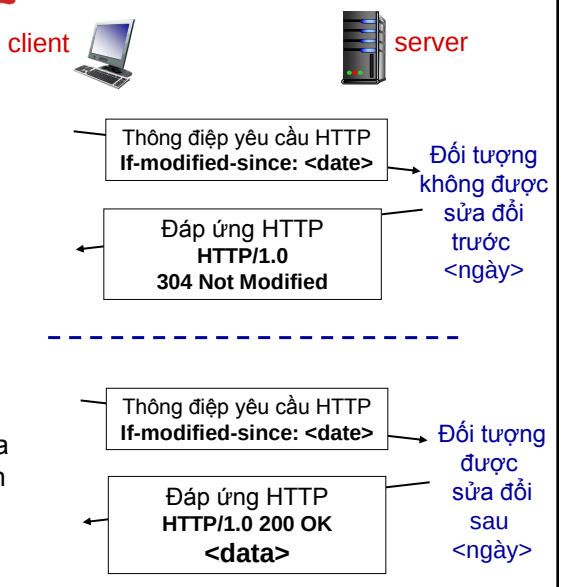


GET có điều kiện

- *Mục tiêu:* không gửi đối tượng nếu cache đã cập nhật
  - Không có trễ truyền đối tượng
  - Độ khả dụng của liên kết thấp hơn
- *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 cach đã được cập nhật:

  HTTP/1.0 304 Not Modified



Chương 2: Nội dung

2.1 Nguyên lý của ứng dụng mạng
- Kiến trúc của ứng dụng
- Các yêu cầu của ứng dụng

2.2 Web và HTTP

2.3 FTP

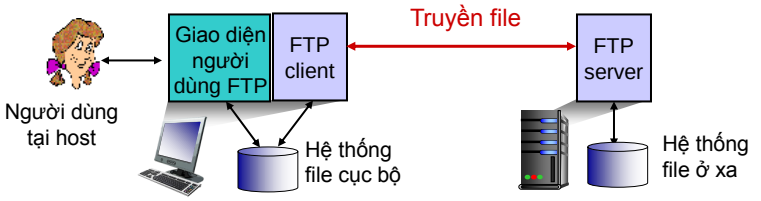
2.4 Thư điện tử
- SMTP, POP3, IMAP

2.5 DNS

2.6 Ứng dụng P2P

2.7 Lập trình socket với UDP và TCP

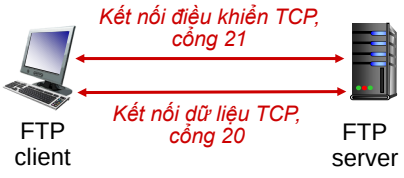
FTP: giao thức truyền file (file transfer protocol)



- Truyền file đến/từ host ở xa
- Mô hình client/server
  - *client:* phía khởi tạo việc truyền (đến/từ host ở xa)
  - *server:* host ở xa
- ftp: RFC 959
- ftp server: cổng 21

FTP: kết nối dữ liệu và kết nối điều khiển riêng biệt nhau

- FTP client tiếp xúc với FTP server tại cổng số 21, sử dụng TCP
- client được cấp phép qua kết nối điều khiển
- client duyệt thư mục ở xa, gửi lệnh qua kết nối điều khiển
- Khi server nhận lệnh truyền file, *server* mở kết nối TCP thứ 2 (kết nối dữ liệu) (cho file) *tới* client
- Sau khi truyền một file, server đóng kết nối dữ liệu.



- server mở một kết nối dữ liệu khác để truyền một file khác.
- Kết nối điều khiển: *"out of band"*
- FTP server duy trì "trạng thái": thư mục hiện hành, sự cấp phép trước đó.

Các lệnh và đáp ứng của FTP

Ví dụ các lệnh:

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

Ví dụ các mã lệnh trả về

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

Chương 2: Nội dung

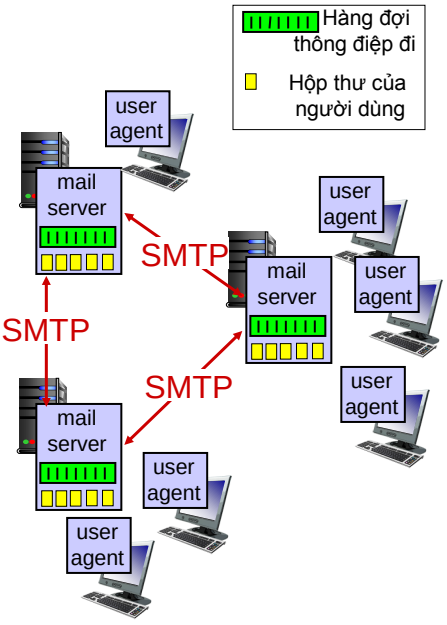
Thư điện tử

Ba thành phần chính:

- user agent
- mail server
- Giao thức truyền thư đơn giản (simple mail transfer protocol): SMTP

User Agent

- Còn được gọi là "mail reader"
- Soạn thảo, sửa, đọc các thông điệp thư
- Ví dụ: Outlook, Thunderbird, iPhone mail client
- Các thông điệp đi/đến được lưu trên server



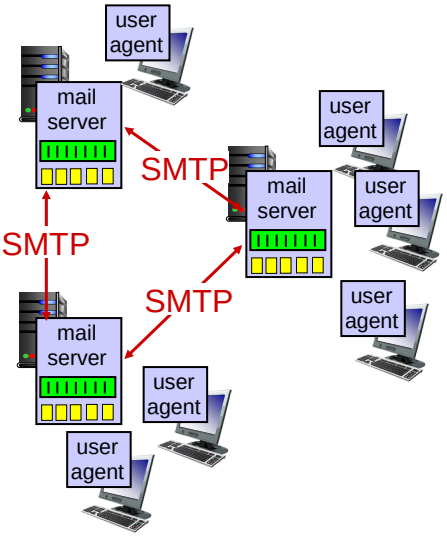
Thư điện tử: mail server

mail server:

- *Hộp thư (mailbox)* chứa các thông điệp thư đi đến người dùng
- *Hàng đợi thông điệp (message queue)* của các thông điệp thư đi ra ngoài (chuẩn bị được gửi đi)

Giao thức SMTP giữa các mail server thực hiện gửi các thông điệp thư

- client: gửi đến mail server
- server: mail server nhận

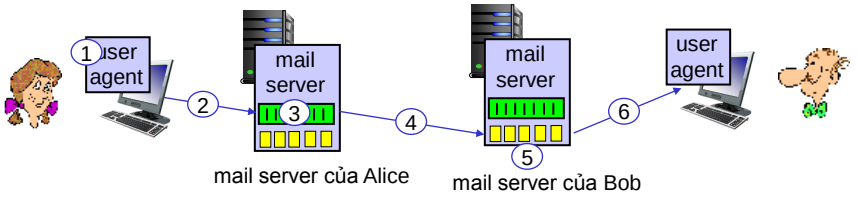


Thư điện tử: SMTP [RFC 2821]

- Sử dụng TCP để truyền tin cậy thông điệp thư điện tử từ client đến server, trên cổng số 25
- Truyền trực tiếp: từ server gửi tới server nhận
- Truyền theo 3 bước
  - Bắt tay (chào hỏi)
  - Truyền thông điệp
  - Đóng
- Tương tác lệnh/đáp ứng (như HTTP, FTP)
  - Lệnh: văn bản ASCII
  - Đáp ứng: mã trạng thái và các cụm từ trạng thái
- Các thông điệp phải ở dạng mã ASCII 7-bit

Kịch bản: Alice gửi thông điệp tới Bob

1) Alice dùng UA soạn thảo thông điệp để "gửi tới" `bob@someschool.edu`
2) UA của Alice gửi thông điệp tới mail server của cô ấy; thông điệp được đặt trong hàng đợi thông điệp
3) Phía client của SMTP mở kết nối TCP tới mail server của Bob.
4) SMTP client gửi thông điệp của Alice qua kết nối TCP
5) mail server của Bob đặt thông điệp trong hộp thư của Bob
6) Bob kích hoạt user agent để đọc thông điệp thư



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
```


Tự thử nghiệm tương tác SMTP:

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

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

SMTP

- SMTP sử dụng kết nối bền vững
- SMTP yêu cầu thông điệp (phần tiêu đề & phần thân) ở dạng mã ASCII 7-bit
- SMTP server sử dụng `CRLF.CRLF` để xác định điểm kết thúc của thông điệp

So sánh với HTTP:

- HTTP: kéo
- SMTP: đẩy
- Cả hai đều sử dụng tương tác lệnh/đáp ứng, các mã trạng thái dạng bảng mã ASCII
- HTTP: mỗi đối tượng được đóng gói trong thông điệp đáp ứng của nó
- SMTP: nhiều đối tượng được gửi trong thông điệp nhiều phần

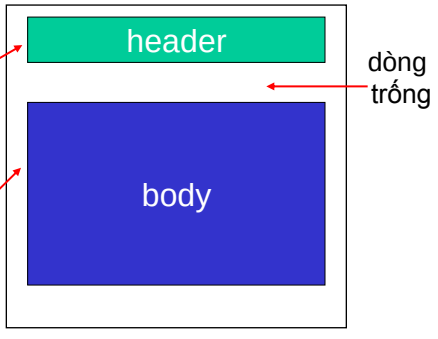
Định dạng thông điệp thư

SMTP: giao thức trao đổi các thông điệp thư

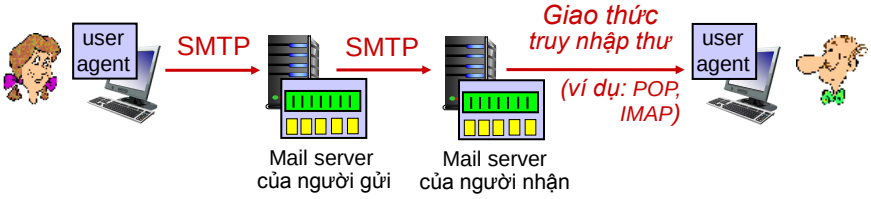
RFC 822: chuẩn cho định dạng thông điệp văn bản:

- Dòng tiêu đề, ví dụ:
  - To:
  - From:
  - Subject:

  Khác với các lệnh SMTP MAIL FROM, RCPT TO!
- Phần thân: "thông điệp"
  - Chỉ dùng các ký tự mã ASCII



Giao thức truy nhập thư



- SMTP: phân phối/lưu trữ thư tới/tại server của người nhận
- Giao thức truy nhập thư: trích xuất thư từ server
  - POP: Post Office Protocol [RFC 1939]: cấp phép, tải thư
  - IMAP: Internet Mail Access Protocol [RFC 1730]: có nhiều đặc tính hơn, bao gồm cả những thao tác với các thông điệp được lưu trữ trên server
  - HTTP: gmail, Hotmail, Yahoo! Mail,...

Giao thức POP3

Giai đoạn cấp phép

- Các lệnh của client:
  - **user:** khai báo tên người dùng
  - **pass:** mật khẩu
- Các đáp ứng của server
  - **+OK**
  - **-ERR**

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

Giai đoạn giao dịch, client:

- **list:** liệt kê các số thông điệp
- **retr:** trích xuất thông điệp theo số
- **dele:** xóa
- **quit**

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (tiếp) và IMAP

POP3

- Ví dụ trước dùng POP3 với chế độ "tải và xóa"
  - Bob không thể đọc lại thư nếu thay đổi client
- POP3 với chế độ "tải và lưu giữ": sao các thông điệp lên các client khác nhau
- POP3 không giữ trạng thái của các phiên làm việc

IMAP

- Lưu giữ tất cả các thông điệp tại một nơi là server
- Cho phép người dùng tổ chức các thông điệp theo dạng các thư mục
- Lưu giữ trạng thái của người dùng qua các phiên làm việc:
  - Đặt tên thư mục và ánh xạ giữa các ID của thông điệp với tên thư mục

Chương 2: Nội dung

DNS: Hệ thống tên miền (domain name system)

Con người: có nhiều định danh:
- CMT, tên, số hộ chiếu

Các host và router trên Internet:
- Địa chỉ IP (32 bit) – được dùng để gán địa chỉ cho các gói tin
- "tên miền", ví dụ: www.yahoo.com – được con người sử dụng

Hỏi: Làm cách nào để ánh xạ giữa địa chỉ IP và tên miền, và ngược lại?

Hệ thống tên miền:
- *Cơ sở dữ liệu phân tán* được cài đặt phân cấp với nhiều *server tên miền*
- *Giao thức tầng ứng dụng:* để các host, các server tên miền truyền thông được thì cần phải *phân giải* các tên miền (diễn dịch địa chỉ/tên miền)
  - Chú ý: chức năng lõi của Internet, được cài đặt như giao thức tầng ứng dụng
  - Phức tạp tại "phần cạnh" của mạng

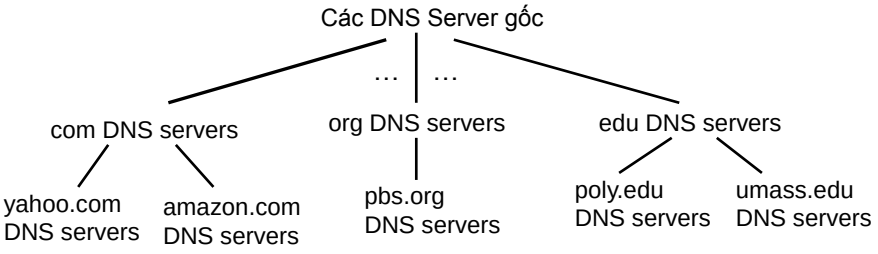
DNS: các dịch vụ và cấu trúc

Các dịch vụ của DNS
- Dịch tên host thành địa chỉ IP
- Bí danh của host
  - Các tên đúng chuẩn, các tên là bí danh
- Bí danh mail server
- Phân tán tải
  - Nhân rộng các máy chủ Web: nhiều địa chỉ IP tương ứng với một tên

Tại sao không tập trung hóa trong DNS?
- Một điểm chịu lỗi
- Lưu lượng
- Cơ sở dữ liệu tập trung ở xa
- Bảo trì

Trả lời: Không thể thực hiện với quy mô lớn!

DNS: cơ sở dữ liệu phân tán và phân cấp

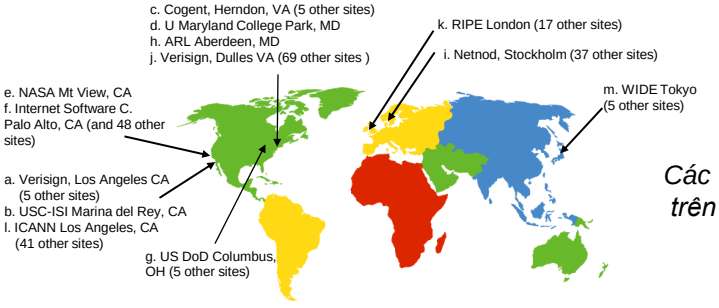


client muốn IP cho www.amazon.com:

- client truy vấn server gốc để tìm com DNS server
- client truy vấn .com DNS server để lấy amazon.com DNS server
- client truy vấn amazon.com DNS server để lấy địa chỉ IP của www.amazon.com

DNS: Các server tên gốc

- Được tiếp xúc bởi server tên cục bộ mà server này không có khả năng phân giải tên
- Server tên gốc:
  - Liên lạc với server tên có thẩm quyền nếu việc ánh xạ tên không được xác định
  - Lấy ánh xạ
  - Trả lại ánh xạ cho server tên cục bộ



Các server tên gốc trên toàn thế giới

Các server TLD và server có thẩm quyền

Các server top-level domain (TLD):

- Chịu trách nhiệm cho tên miền com, org, net, edu, aero, jobs, museums, và tất cả các tên miền cấp cao nhất thuộc quốc gia như: uk, fr, ca, jp
- Network Solutions duy trì các server cho .com TLD
- Educause cho .edu TLD

Các server DNS có thẩm quyền:

- Các DNS server của tổ chức, cung cấp các tên host có thẩm quyền cho các ánh xạ IP với các host của tổ chức (ví dụ web và mail).
- Có thể được duy trì bởi các tổ chức hoặc nhà cung cấp dịch vụ

Server tên DNS cục bộ

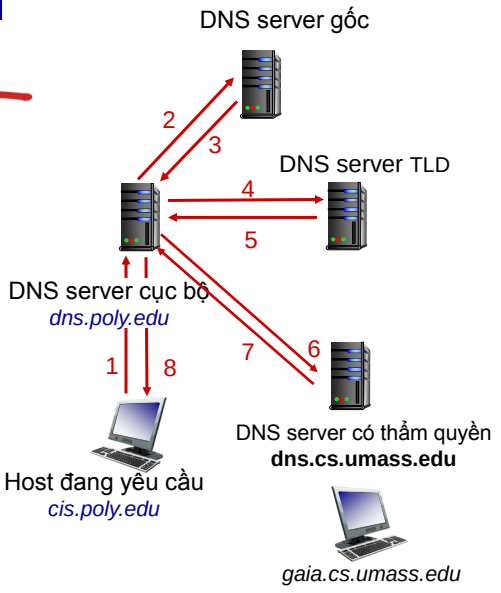
- Không hoàn toàn theo cấu trúc phân cấp
- Mỗi ISP (ISP khu dân cư, công ty, trường học) có một server cục bộ
  - Cũng được gọi là "server tên mặc định"
- Khi một host tạo một truy vấn DNS, truy vấn này sẽ được gửi tới server DNS cục bộ
  - Có một vùng đệm cục bộ dịch chuyển cặp tên-địa chỉ gần nhất (tuy nhiên vẫn có thể đã bị quá hạn, chưa được cập nhật!)
  - Hoạt động giống như proxy, chuyển tiếp truy vấn vào hệ thống phân cấp

Ví dụ phân giải tên DNS

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

Truy vấn lặp:

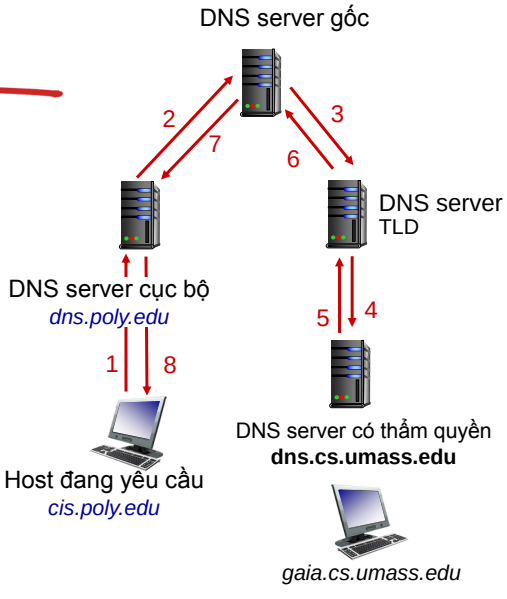
- Server được hỏi sẽ trả lời với tên của server sẽ có thể hỏi được tiếp
- "Tôi không biết tên đó, nhưng có thể hỏi server này"



Ví dụ phân giải tên DNS

Truy vấn đệ quy:

- Đặt trách nhiệm phân giải tên lên server tên được hỏi
- Tải nặng tại các cấp cao hơn trong hệ thống phân cấp?



DNS: caching và cập nhật các bản ghi

- Khi server tên học cách ánh xạ, nó sẽ *cache* ánh xạ
  - Mục cache sẽ bị quá hạn (bị xóa) sau một vài lần (TTL)
  - Các server TLD điển hình thường cache trong các server tên cục bộ
    - Do đó, các server tên gốc sẽ không thường xuyên được truy cập
- Các mục cache có thể bị *quá hạn*
  - Nếu host thay đổi địa chỉ IP, thì Internet sẽ có thể không biết được cho đến khi TTL hết hạn
- Cơ chế cập nhật, thông báo được đề xuất bởi chuẩn IETF
  - RFC 2136

Bản ghi DNS (DNS records)

DNS: cơ sở dữ liệu phân tán lưu trữ các bản ghi nguồn (resource records - RR)

Định dạng RR: `(name, value, type, ttl)`

type=A
- **name** là tên máy
- **Value** là địa chỉ IP

type=NS
- **name** là tên miền (ví dụ: foo.com)
- **value** là tên host của server có thẩm quyền cho tên miền này

type=CNAME
- **name** là tên bí danh của tên "chuẩn" (tên thật)
- `www.ibm.com` tên thật là `servereast.backup2.ibm.com`
- **value** là tên chuẩn

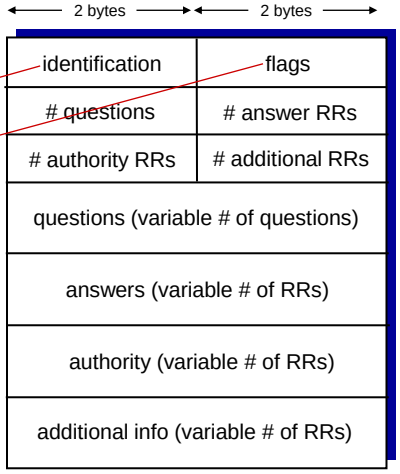
type=MX
- **value** là tên của mail server được kết hợp với **name**

Giao thức, thông điệp DNS

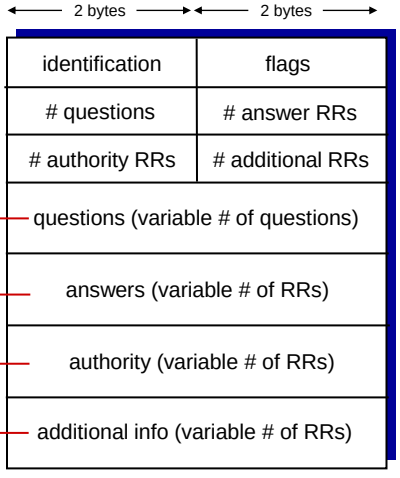
- Các thông điệp *truy vấn* và *trả lời* đều có cùng *định dạng thông điệp*

Tiêu đề thông điệp

- identification: 16 bit cho truy vấn/trả lời
- flags:
  - Truy vấn hoặc trả lời
  - Đệ quy chờ
  - Đệ quy sẵn sàng
  - Trả lời có thẩm quyền



Giao thức, thông điệp DNS



Trường tên, loại truy vấn

RR trong trả lời cho truy vấn

Bản ghi của server có thẩm quyền

Thông tin "hữu ích" có thể được sử dụng

Chèn thêm bản ghi vào trong DNS

- Ví dụ: Tạo mới "Network Utopia"
- Đăng ký tên miền networkuptopia.com tại *DNS registrar* (Ví dụ: Network Solutions)
  - Cung cấp tên, địa chỉ IP của server tên có thẩm quyền (sơ cấp và thứ cấp) của bạn
  - Registrar sẽ chèn hai bản ghi RR vào trong .com TLD server:
    `(networkutopia.com, dns1.networkutopia.com, NS)`
    `(dns1.networkutopia.com, 212.212.212.1, A)`
- Tạo bản ghi loại A vào trong server có thẩm quyền cho www.networkuptopia.com; bản ghi loại MX record cho networkutopia.com

Tấn công DNS

Tấn công DDoS

- Tấn công server gốc với lưu lượng lớn
  - Không thành công cho đến nay
  - Lọc lưu lượng
  - DNS server cục bộ cache IP của TLD server, cho phép bỏ qua server gốc
  - Tấn công TLD servers
  - Tiềm tàng nhiều nguy hiểm hơn

Chuyển hướng tấn công

- Man-in-middle
  - Truy vấn đánh chặn
- Đầu độc DNS
  - Gửi trả lời giả mạo đến DNS server, mà các server này có thể cache lại

Khai thác DNS cho DDoS

- Gửi truy vấn với địa chỉ nguồn giả mạo: IP đích
- Yêu cầu khuếch đại

Chương 2: Nội dung

2.1 Nguyên lý của ứng dụng mạng
- Kiến trúc của ứng dụng
- Các yêu cầu của ứng dụng

2.2 Web và HTTP

2.3 FTP

2.4 Thư điện tử
- SMTP, POP3, IMAP

2.5 DNS

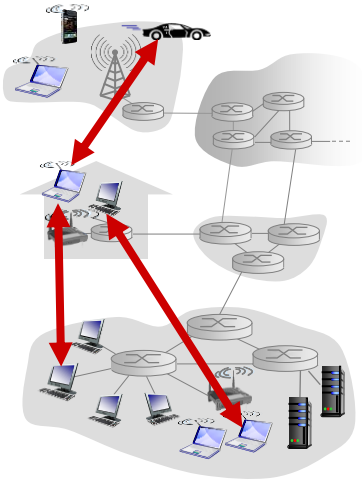
2.6 Ứng dụng P2P

2.7 Lập trình socket với UDP và TCP

Kiến trúc mạng P2P thuần túy

- *Không* cần phải có server luôn hoạt động
- Các hệ thống cuối tùy ý kết nối trực tiếp
- Các peer (các nút mạng) không cần kết nối liên tục vào hệ thống mạng và có thể thay đổi địa chỉ IP.

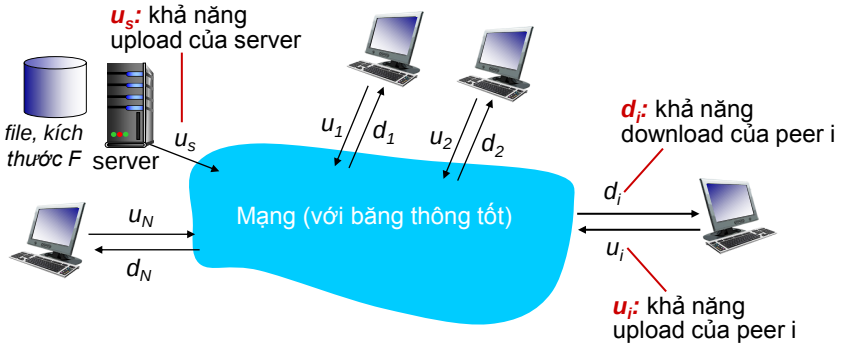
Ví dụ:
- Chia sẻ file (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



Chia sẻ file: client-server và P2P

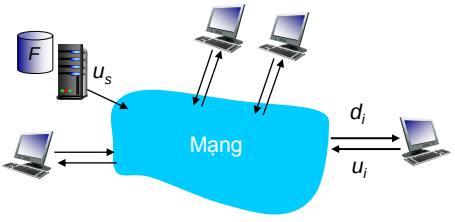
Hỏi: Cần mất bao lâu để truyền (phân phối) file (có kích thước *F*) từ một server đến *N* peer?

- Khả năng upload/download (tải lên/tải về) của peer bị giới hạn bởi tài nguyên.



Thời gian truyền file: client-server

- *Việc truyền của server:* phải gửi tuần tự (upload) *N bản sao của* file:
  - Thời gian gửi một bản sao: F/u_s
  - Thời gian gửi N bản sao: NF/u_s
- *Client:* Mỗi client phải download bản sao của file
  - d_{min} = tốc độ download nhỏ nhất của client
  - Thời gian download (ứng với d_{min}) của client: F/d_{min}



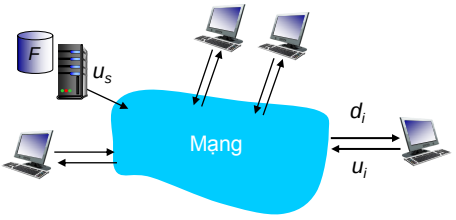
Thời gian để phân phối F tới N client sử dụng cách tiếp cận client-server

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

Tăng tuyến tính theo N

Thời gian truyền file: P2P

- *Việc truyền của server:* phải upload ít nhất một bản sao của file:
  - Thời gian để gửi một bản: F/u_s
- *Client:* mỗi client phải download bản sao của file
  - Thời gian download (ứng với d_{min}) của client : F/d_{min}
- *Các client:* phải download *NF* bits
  - Tốc độ upload cao nhất (giới hạn) là $u_s + \sum u_i$



Thời gian để phân phối F tới N client sử dụng cách tiếp cận P2P

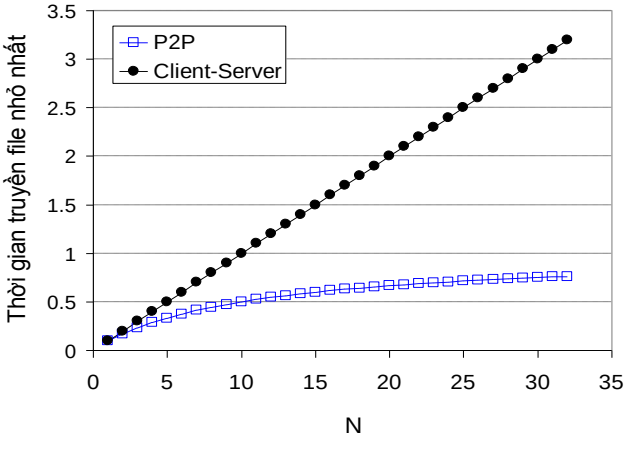
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

Tăng tuyến tính theo *N* …

… nhưng để thực hiện việc này, mỗi peer phải có khả năng phục vụ

Ví dụ so sánh client-server với P2P

Tốc độ upload của client = u, F/u = 1 giờ, $u_s = 10u$, $d_{min} \geq u_s$

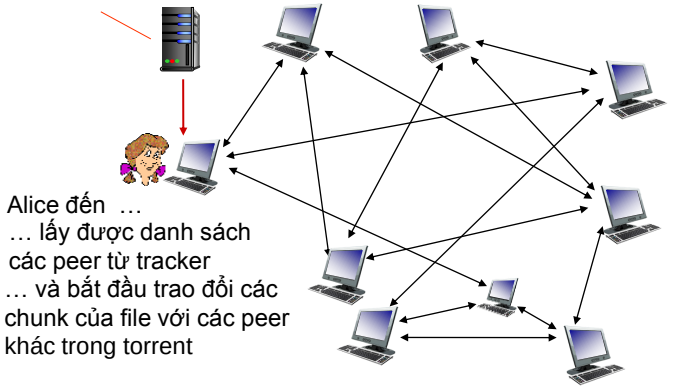


Truyền file P2P: BitTorrent

- File được chia thành các chunk (phần) có kích thước là 256Kb
- Các peer trong torrent gửi/nhận các chunk của file

tracker: kiểm tra các peer đang tham gia trong torrent

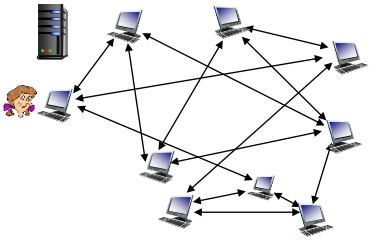
torrent: nhóm các peer trao đổi các chunk của một file



Alice đến …
… lấy được danh sách các peer từ tracker
… và bắt đầu trao đổi các chunk của file với các peer khác trong torrent

Truyền file P2P: BitTorrent

- peer tham gia vào torrent:
  - Không có các chunk, nhưng sẽ tích lũy chúng qua thời gian từ các peer khác
  - Đăng ký với tracker để nhận được danh sách các peer, kết nối với tập con của các peer ("các hàng xóm")



- Trong khi download, peer sẽ upload các chunk tới các peer khác
- peer có thể thay đổi các peer mà nó sẽ trao đổi các chunk
- *churn:* các peer có thể đến hoặc đi
- Khi peer có được toàn bộ file, nó có thể (ích kỷ) rời đi hoặc (vị tha) ở lại trong torrent

BitTorrent: yêu cầu lấy, gửi các chunk của file

Yêu cầu lấy chunk:

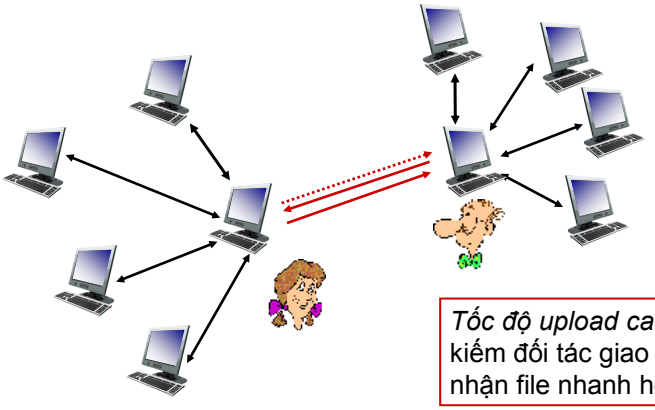
- Tại bất kỳ thời điểm nào, các peer khác nhau đều có các tập con các chunk khác nhau của file.
- Theo định kỳ, Alice sẽ hỏi mỗi peer về danh sách các chunk mà họ có
- Alice sẽ yêu cầu các chunk còn thiếu từ các peer, phần hiếm nhất trước

Gửi chunk: tit-for-tat

- Alice gửi các chunk tới 4 peer hiện đang gửi chunk của cô ấy với *tốc độ cao nhất*
  - Các peer khác đang bị chặn bởi Alice (không nhận được chunk từ cô ấy)
  - Đánh giá lại top 4 sau mỗi 10 giây
- Cứ mỗi 30 giây: chọn các peer khác một cách ngẫu nhiên, và bắt đầu gửi chunk
  - "Tối ưu hóa không chặn" peer này
  - Peer được chọn mới sẽ được gia nhập vào top 4

BitTorrent: tit-for-tat

(1) Alice "tối ưu hóa không chặn" Bob
(2) Alice trở thành một nhà cung cấp trong top 4 của Bob;
(3) Ngược lại, Bob trở thành một nhà cung cấp trong top 4 của Alice



Tốc độ upload cao hơn: tìm kiếm đối tác giao dịch tốt hơn, nhận file nhanh hơn!

Bảng băm phân tán - Distributed Hash Table (DHT)

- DHT: là một *cơ sở dữ liệu P2P phân tán*
- Cơ sở dữ liệu chứa các cặp (khóa, giá trị) (key, value); Ví dụ:
  - Khóa: Số thứ tự; Giá trị: tên người
  - Khóa: tiêu đề bộ phim; Giá trị: địa chỉ IP
- Phân tán các cặp (key, value) qua (hàng triệu) các peer
- Mỗi peer truy vấn DHT theo khóa
  - DHT trả lại các giá trị tương ứng với khóa
- Các peer cũng có thể chèn thêm (insert) các cặp (khóa, giá trị)

Hỏi: Làm cách nào để gán khóa cho các peer?

- Vấn đề chính:
  - Gán các cặp (khóa, giá trị) cho các peer.
- Ý tưởng cơ bản:
  - Chuyển mỗi khóa thành một số kiểu số nguyên (integer)
  - Gán số nguyên cho từng peer
  - Đặt cặp (khóa, giá trị) trong một peer mà nó là gần nhất với khóa

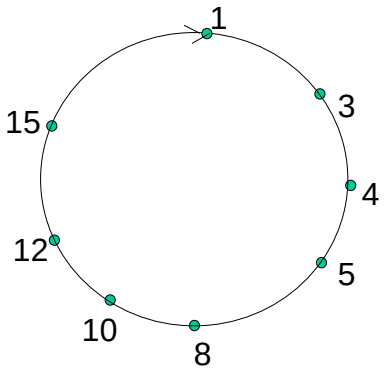
Định danh DHT

- Gán định danh số nguyên cho mỗi peer trong dãy $[0,2^n-1]$ cho một số *n*.
  - Mỗi định danh được biểu diễn bởi *n* bit.
- Yêu cầu mỗi khóa là một số nguyên trong cùng dãy
- Để nhận được khóa số nguyên, cần băm khóa gốc
  - *Ví dụ:* khóa = hash("Led Zeppelin IV")
  - Đây là lý do tại sao nó được gọi là một *bảng "băm" phân tán*

Gán khóa cho các peer

- Quy tắc: gán khóa cho peer mà có ID *gần nhất*.
- Trong bài giảng: gần nhất là *giá trị kế liền ngay* với khóa.
- Ví dụ: *n*=4; các peer: 1,3,4,5,8,10,12,14;
  - khóa = 13, thì peer kế liền với nó là 14
  - khóa = 15, thì peer kế liền là 1

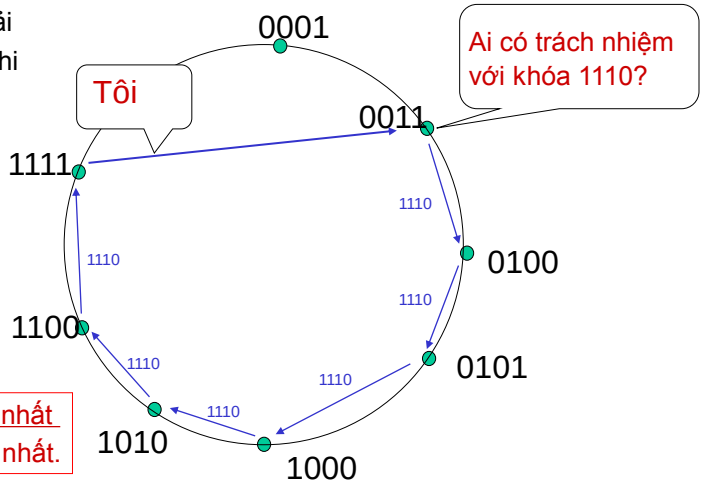
DHT vòng (1)



- Mỗi peer *chỉ biết* về peer kế tiếp và peer tiền nhiệm ngay trước nó
- Mạng che phủ ("overlay network")

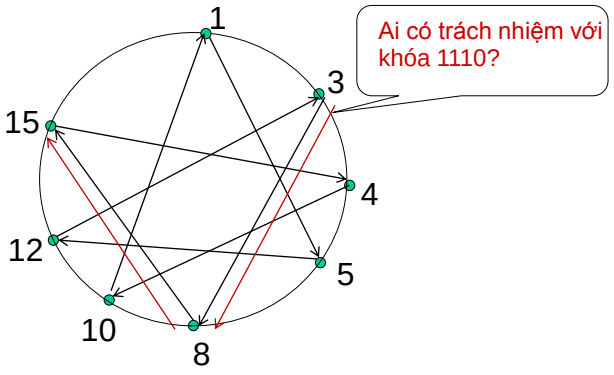
DHT vòng (1)

Trung bình cần *O(N)* thông điệp để giải quyết truy vấn, khi có *N* peer



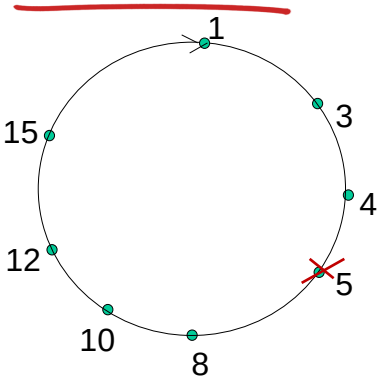
Định nghĩa gần nhất nghĩa là kế gần nhất.

DHT vòng với peer tắt (shortcut)



- Mỗi peer giữ theo dõi địa chỉ IP của peer kế tiếp, tiền nhiệm và các peer tắt.
- Sẽ giảm từ 6 xuống còn 2 thông điệp.
- Có thể thiết kế các peer tắt để có *O(log N)* hàng xóm, thì sẽ có *O(log N)* thông điệp trong truy vấn

Peer churn



Ví dụ: peer 5 đột ngột rời đi

Xử lý peer churn:

- Các peer có thể đến hoặc đi (churn)
- Mỗi peer biết được địa chỉ của 2 peer kế của nó
- Định kỳ mỗi peer sẽ ping đến 2 peer kế của nó để kiểm tra sự tồn tại
- Nếu peer kế liền ngay nó rời đi, thì nó sẽ chọn peer kế tiếp để làm peer kế liền ngay của nó

- peer 4 phát hiện ra peer 5 rời đi, nó sẽ làm việc với peer 8 kế liền ngay với nó; hỏi peer 8 xem peer nào sẽ làm peer kế liền ngay của nó được; sau đó sẽ làm việc tiếp với peer kế liền ngay với peer 8 này (kế liền ngay thứ cấp).
- peer 13 muốn kết nối với ai?

Chương 2: Nội dung

2.1 Nguyên lý của ứng dụng mạng
- Kiến trúc của ứng dụng
- Các yêu cầu của ứng dụng

2.2 Web và HTTP

2.3 FTP

2.4 Thư điện tử
- SMTP, POP3, IMAP

2.5 DNS

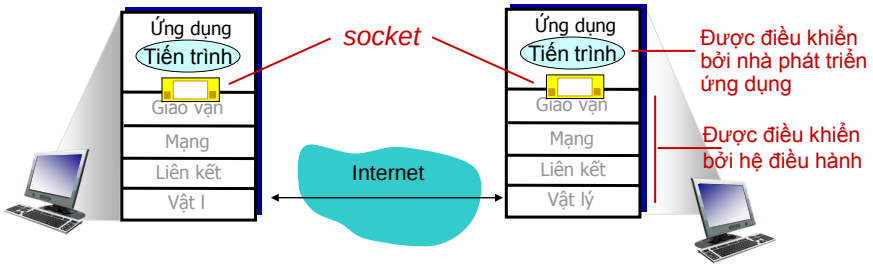
2.6 Ứng dụng P2P

2.7 Lập trình socket với UDP và TCP

Lập trình Socket

Mục đích: hiểu được cách xây dựng ứng dụng truyền thông client/server dùng socket

socket: là cánh cửa giữa các tiến trình ứng dụng và giao thức giao vận end-to-end



Lập trình Socket

Hai loại socket cho hai dịch vụ tầng giao vận:

- *UDP:* truyền các gói tin không tin cậy
- *TCP:* truyền tin cậy, truyền dòng byte có hướng

Ví dụ ứng dụng:

1. Client đọc vào một dòng ký tự (dữ liệu) từ bàn phím và gửi dữ liệu đến server.
2. Server nhận dữ liệu và chuyển các ký tự thành dạng ký tự viết hoa.
3. Server gửi dữ liệu đã được chuyển thành dạng viết hoa về cho client.
4. Client nhận dữ liệu và hiển thị dòng ký tự lên màn hình.

Lập trình mạng trên Java

- Gói java.net
  - InetAddress
  - ServerSocket
  - Socket
  - URL
  - URLConnection
  - DatagramSocket

Lập trình mạng trên Java

- InetAddress class
  - Class mô tả về địa chỉ IP
  - Các phương thức getLocalHost, getByName, hay getAllByName để tạo một InetAddress instance:
    public static InetAddess InetAddress.getByName(String hostname)
    public static InetAddess [] InetAddress.getAllByName(String hostname)
    public static InetAddess InetAddress.getLocalHost()
  - Để lấy địa chỉ IP hay tên dùng các phương thức:
    getHostAddress()
    getHostName()

Lập trình Socket *với UDP*

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

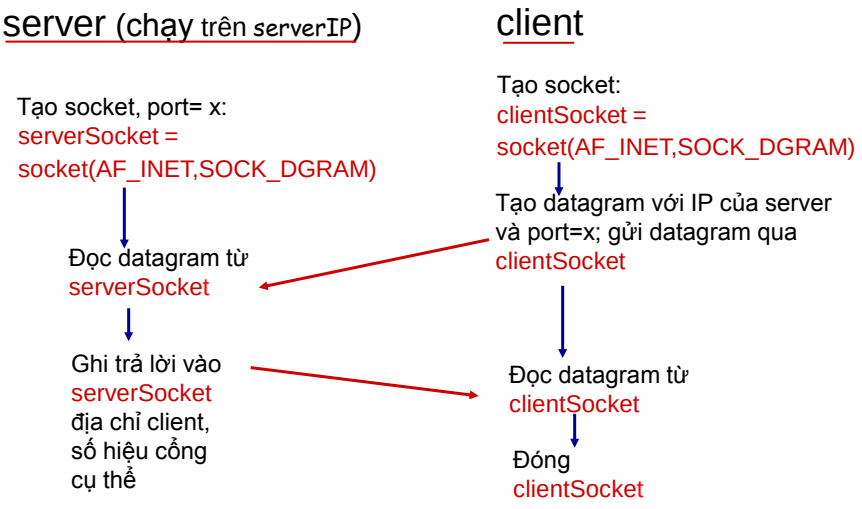
- Không bắt tay trước khi gửi dữ liệu
- Bên gửi gắn địa chỉ IP và số hiệu cổng đích vào trong mỗi gói tin
- Bên nhận sẽ trích địa chỉ IP và số hiệu cổng của bên gửi từ gói tin nhận được

UDP: dữ liệu được truyền có thể bị mất hoặc không đúng trình tự khi nhận

Quan điểm ứng dụng:

- UDP cung cấp truyền không tin cậy theo các nhóm byte ("các gói tin") giữa client và server

Tương tác client/server socket: UDP



Ví dụ: UDP client (Java client)

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

class UDPClient {
   public static void main(String args[]) throws Exception
   {
```

tạo input stream →

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

tạo client socket →

```
     DatagramSocket clientSocket = new DatagramSocket();
```

dịch hostname thành địa chỉ IP dùng DNS →

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

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

     String sentence = inFromUser.readLine();
     sendData = sentence.getBytes();
```

Ví dụ: UDP client (Java client)

tạo dữ liệu gửi datagram, 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: UDP server (Java server)

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

class UDPServer {
  public static void main(String args[]) throws Exception
    {
```

tạo datagram socket tại port 9876

```
      DatagramSocket serverSocket = new DatagramSocket(9876);

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

      while(true)
        {
```

tạo không gian để nhận datagram

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

nhận datagram

```
        serverSocket.receive(receivePacket);
```

Ví dụ: UDP server (Java server)

```
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);
        }
    }
}
```

lấy địa chỉ IP, số hiệu cổng của người gửi

tạo datagram để gửi tới client

ghi datagram vào socket

kết thúc vòng lặp while, quay lại và chờ datagram khác

Lập trình socket *với TCP*

client phải tiếp xúc với server

- Tiến trình server phải chạy trước
- server phải tạo socket (cửa) để đón client tiếp xúc

client tiếp xúc với server bằng cách:

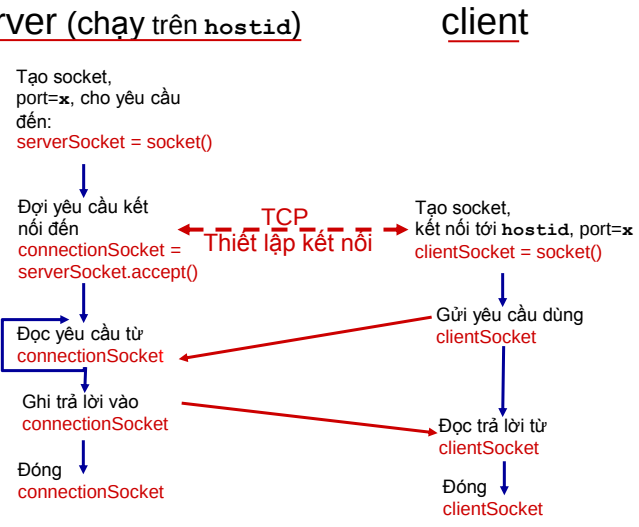
- Tạo TCP socket, xác định địa chỉ IP, số hiệu cổng của tiến trình server
- *Khi client tạo socket:* TCP client sẽ thiết lập kết nối tới TCP server

- Khi được tiếp xúc bởi client, *TCP server sẽ tạo socket mới* cho tiến trình server để truyền thông với client
  - Cho phép server "nói chuyện" với nhiều client
  - Các số hiệu cổng nguồn được dùng để phân biệt các client (xem thêm trong Chương 3)

Quan điểm ứng dụng:

TCP cung cấp truyền tin cậy, truyền dòng byte theo đúng trình tự giữa client và server.

Tương tác client/server socket: TCP



Ví dụ: TCP client (Java client)

```
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: TCP client (Java client)

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: TCP server (Java server)

```
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 đón tiếp xúc tại port 6789

chờ client tiếp xúc với server

tạo input stream, gắn vào socket

Ví dụ: TCP server (Java server)

tạo output stream, gắn vào socket

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

đọc dòng từ socket

```
clientSentence = inFromClient.readLine();

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

ghi dòng ra từ socket

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

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

Xây dựng một Web server đơn giản

- Quản lý một yêu cầu HTTP
- Chấp nhận yêu cầu
- Phân tích cú pháp phần tiêu đề (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, có thể yêu cầu file dùng trình duyệt (ví dụ: IE)
- Xem giáo trình để biết thêm chi tiết

Chương 2: Tổng kết

Trình bày các vấn đề liên quan đến ứng dụng mạng!

- Kiến trúc của ứng dụng
  - client-server
  - P2P
- Các yêu cầu dịch vụ của ứng dụng:
  - Truyền tin cậy, băng thông, trễ
- Mô hình dịch vụ giao vận của Internet
  - Hướng kết nối, truyền tin cậy: TCP
  - Truyền không tin cậy, truyền gói tin: UDP
- Các giao thức cụ thể:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- Lập trình socket: TCP socket, UDP socket

Chương 2: Tổng kết

Quan trọng hơn: được học về các giao thức!

- Trao đổi giữa các thông điệp yêu cầu/đáp ứng:
  - client yêu cầu thông tin hoặc dịch vụ
  - server đáp ứng với dữ liệu, hoặc mã trạng thái
- Định dạng thông điệp:
  - Phần tiêu đề (header): các trường với thông tin về dữ liệu
  - Dữ liệu: thông tin được truyền thông

Các vấn đề quan trọng:

- Thông điệp điều khiển và thông điệp dữ liệu
  - in-band, out-of-band
- Tập trung hóa và không tập trung hóa
- Không trạng thái và có trạng thái
- Truyền thông điệp tin cậy và truyền thông điệp không tin cậy
- "Sự phức tạp tại phần cạnh của mạng"