

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**  
**Posts and Telecommunications Institute of Technology**

# **MẠNG MÁY TÍNH**

**Chương 2: Tầng ứng dụng – Application Layer**

# Nội dung

- Các nguyên tắc của các ứng dụng mạng
- Web và HTTP
- FTP
- Thư điện tử trên Internet
- DNS – Internet's Directory Service
- Các ứng dụng P2P (Peer – to- Peer)
- Video Streaming và các mạng phân phối nội dung (Content Distribution Networks)
- Lập trình socket với UDP và TCP

# Nội dung

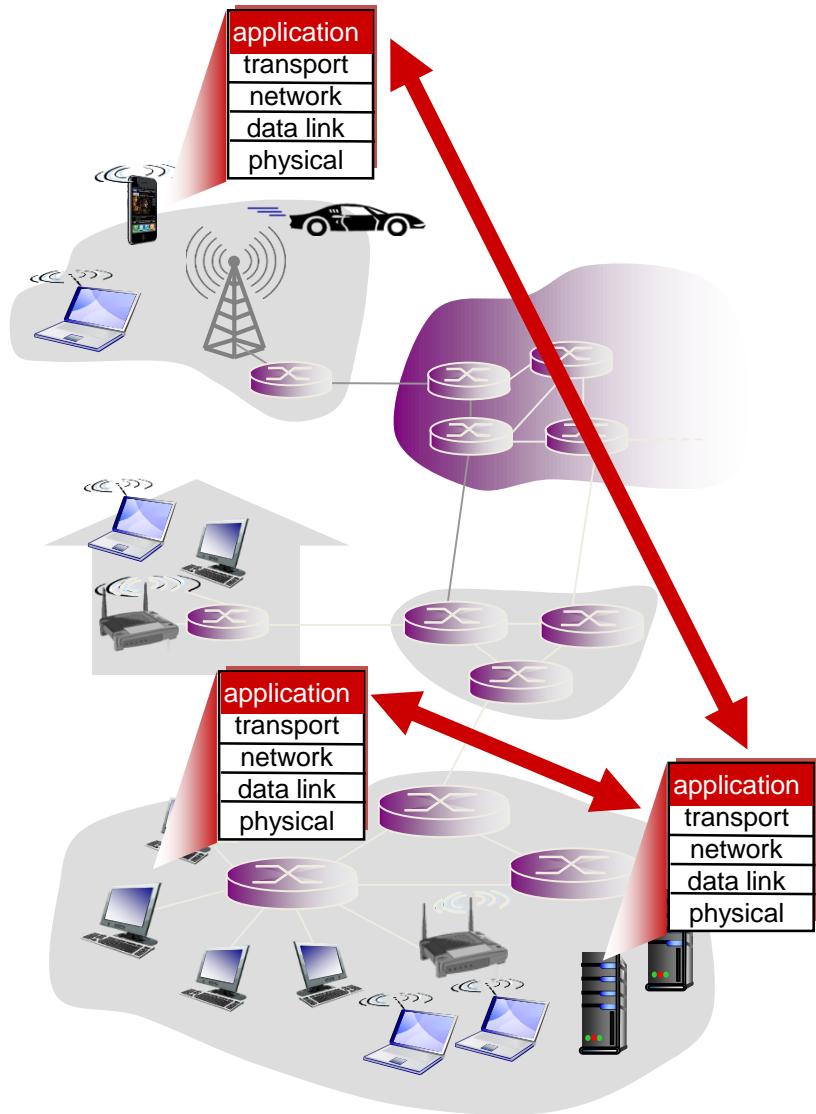
- Các nguyên tắc của các ứng dụng mạng
- Web và HTTP
- FTP
- Thư điện tử trên Internet
- DNS – Internet's Directory Service
- Các ứng dụng P2P (Peer – to- Peer)
- Video Streaming và các mạng phân phối nội dung (Content Distribution Networks)
- Lập trình socket với UDP và TCP

# 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) (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 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)



# Nội dung

- Các nguyên tắc của các ứng dụng mạng
  - Kiến trúc ứng dụng
  - Truyền thông giữa các tiến trình
  - Các dịch vụ giao vận khả dụng cho các ứng dụng
  - Các dịch vụ giao vận do Internet cung cấp
  - Các giao thức tầng ứng dụng

# Nội dung

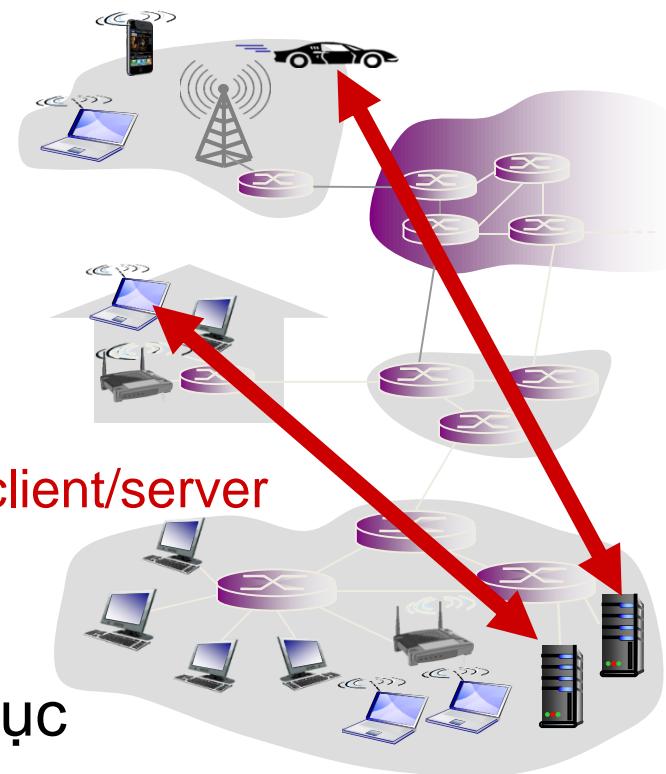
- Các nguyên tắc của các ứng dụng mạng
  - Kiến trúc ứng dụng
  - Truyền thông giữa các tiến trình
  - Các dịch vụ giao vận khả dụng cho các ứng dụng
  - Các dịch vụ giao vận do Internet cung cấp
  - Các giao thức tầng ứng dụng

# Các nguyên tắc của các ứng dụng mạng

- Kiến trúc ứng dụng
  - 02 **mô hình kiến trúc** được sử dụng nhiều trong các ứng dụng mạng hiện đại
    - Kiến trúc khách - chủ (Client – Server )
    - Kiến trúc ngang hàng P2P (Peer-to-Peer)

# Kiến trúc của ứng dụng

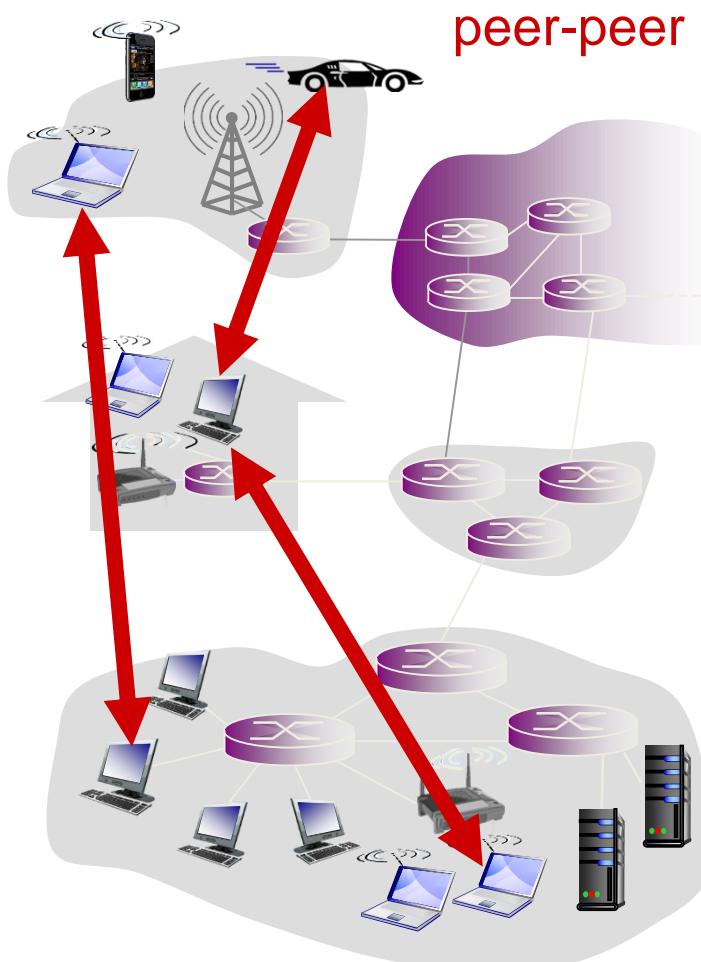
- Client-server (khách-chủ)
  - server:
    - Là host luôn hoạt động
    - Có địa chỉ IP cố định
    - Các trung tâm dữ liệu
  - client:
    - Truyền thông với server
    - Có thể được kết nối liên tục vào mạng hoặc không
    - Có thể có địa chỉ IP thay đổi
    - Không truyền thông trực tiếp với client khác



# Kiến trúc của ứng dụng

## • 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 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



# Nội dung

- Các nguyên tắc của các ứng dụng mạng
  - Kiến trúc ứng dụng
  - Truyền thông giữa các tiến trình
  - Các dịch vụ giao vận khả dụng cho các ứng dụng
  - Các dịch vụ giao vận do Internet cung cấp
  - Các giao thức tầng ứng dụng

# Truyền thông giữa các tiến trình

- Tiến trình - **Processes?**
- Truyền thông liên tiến trình - **interprocess communication?**
- Thông điệp - **Messages?**
- **Socket?**

# Truyền thông giữa các tiến trình

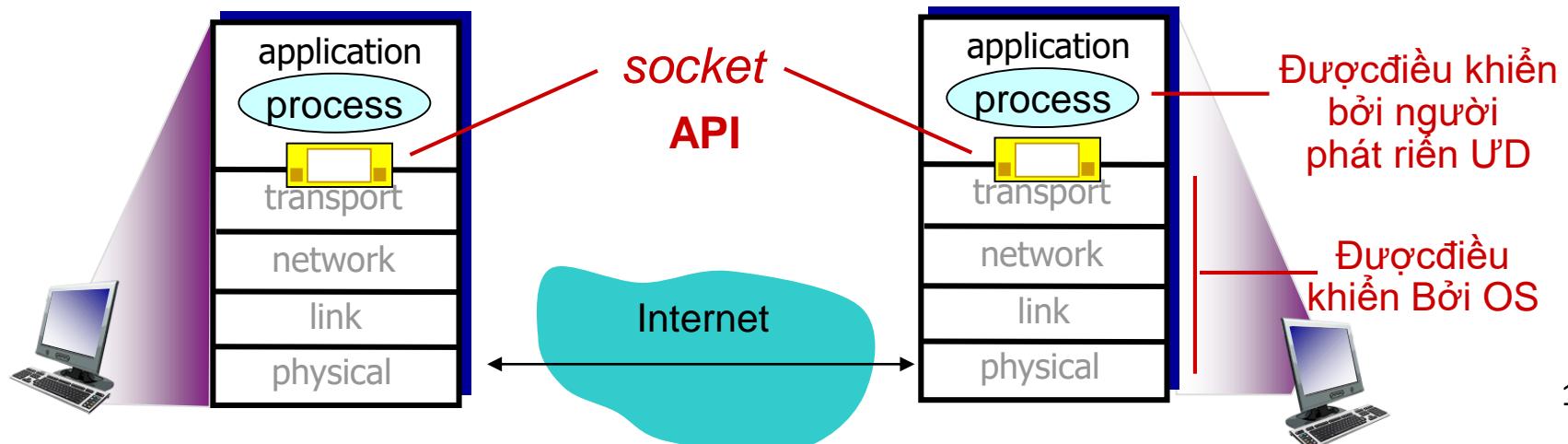
- *Tiến trình (processes)*: chương trình chạy trong một host
  - Trên cùng một host, hai tiến trình truyền thông với nhau qua **truyền thông tiến trình nội bộ (interprocess communication)**, được xác định bởi hệ điều hành)
  - Các tiến trình trên các host khác nhau truyền thông với nhau bằng cách **trao đổi các thông điệp (messages)** qua mạng

# Truyền thông giữa các tiến trình

- Các tiến trình Client-server
  - *Tiến trình client*: tiến trình khởi tạo truyền thông
    - Ví dụ với ứng dụng Web?
  - *Tiến trình server*: tiến trình chờ được liên lạc
    - Ví dụ với ứng dụng Web?
- Ứng dụng theo kiến trúc P2P có cả tiến trình client – server
  - Ví dụ hệ thống chia sẻ file P2P
    - Client = peer đang download file
    - Server = peer đang uploading file

# Truyền thông giữa các tiến trình

- 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 đầy 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.



# Truyền thông giữa các tiến trình

- Định danh cho tiến trình
  - Để nhận các thông điệp, tiến trình phải có *định danh (identifier)*
  - 2 thông tin định danh cho tiến trình nhận
    - Định danh host đích: **địa chỉ IP** của host nhận
    - Định danh cho tiến trình nhận ở host đích: số hiệu cổng – **port number**
      - HTTP server: 80
      - mail server: 25

# Nội dung

- Các nguyên tắc của các ứng dụng mạng
  - Kiến trúc ứng dụng
  - Truyền thông giữa các tiến trình
  - Các dịch vụ giao vận khả dụng cho các ứng dụng
  - Các dịch vụ giao vận do Internet cung cấp
  - Các giao thức tầng ứng dụng

## Các dịch vụ giao vận khả dụng cho các ứng dụng

- Truyền dữ liệu tin cậy (reliable data transfer)
- Thông lượng (throughput)
- Thời gian thực (timing)
- An toàn (security)

# Các dịch vụ giao vận khả dụng cho các ứng dụng

- Truyền dữ liệu tin cậy (reliable data transfer)
    - Hiện tượng packet loss
      - Gây ra hậu quả nghiêm trọng với một số ứng dụng:
        - Thư điện tử
        - Truyền file
        - Truy cập host từ xa
        - Truyền dữ liệu Web
        - Các ứng dụng tài chính
- => cần có giao thức tầng giao vận đảm bảo truyền dữ liệu đúng và đủ từ nguồn tới đích

# Các dịch vụ giao vận khả dụng cho các ứng dụng

- Thông lượng
  - Thông lượng khả dụng được đảm bảo ở một tốc độ xác định
    - Ứng dụng có thể yêu cầu một thông lượng đảm bảo là  $r$  bit/giây. Khi đó giao thức giao vận sẽ đảm bảo thông lượng khả dụng luôn ở ít nhất là  $r$  bit/giây
    - **Bandwidth-sensitive applications**
      - Ứng dụng điện thoại Internet mã hóa giọng nói ở tốc độ 32 kbps. Nó cần gửi dữ liệu lên mạng và dữ liệu này được truyền tới ứng dụng nhận với tốc độ 32 kbps.
    - **Elastic applications**
      - Email, truyền file, truyền Web

# Các dịch vụ giao vận khả dụng cho các ứng dụng

- Thời gian thực
  - Ví dụ
    - Đảm bảo mỗi bít mà bên gửi đưa vào socket sẽ đến được socket của bên nhận không quá 100 ms
  - Phù hợp với các ứng dụng tương tác thời gian thực vốn yêu cầu những ràng buộc chặt chẽ về thời gian truyền dữ liệu để đảm bảo hiệu quả
    - Điện thoại Internet
    - Các môi trường ảo,
    - Hội nghị từ xa
    - Game nhiều người chơi

# Các dịch vụ giao vận khả dụng cho các ứng dụng

- An toàn
  - Một giao thức giao vận có thể cung cấp cho một ứng dụng một hoặc nhiều dịch vụ an toàn.
  - Ví dụ
    - Trong host gửi, giao thức tầng giao vận có thể mã hóa tất cả các dữ liệu truyền bởi tiến trình gửi. Trong host nhận, giao thức tầng giao vận có thể giải mã dữ liệu trước khi truyền tới tiến trình nhận  
⇒ Cung cấp **dịch vụ tin cậy** (confidentiality) giữa 2 tiến trình
    - Dịch vụ **toàn vẹn dữ liệu (Integrity)**
    - Dịch vụ **xác thực đầu cuối (authentication)**

# Các dịch vụ giao vận khả dụng cho các ứng dụng

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
Trò chơi tương tác	chịu lỗi	một vài kbps	có, 100 msec
Tin nhắn nhanh	không	mềm dẻo	có và không

# Nội dung

- Các nguyên tắc của các ứng dụng mạng
  - Kiến trúc ứng dụng
  - Truyền thông giữa các tiến trình
  - Các dịch vụ giao vận khả dụng cho các ứng dụng
  - Các dịch vụ giao vận do Internet cung cấp
  - Các giao thức tầng ứng dụng

# Các dịch vụ giao vận do Internet cung cấp

- 2 giao thức giao vận
  - TCP
    - Dịch vụ hướng kết nối (connection-oriented service)
    - Dịch vụ truyền dữ liệu tin cậy (reliable data transfer service)
  - UDP
    - Dịch vụ phi kết nối (Connectionless)
    - Dịch vụ truyền dữ liệu không tin cậy (Unreliable data transfer service)

# Các dịch vụ giao vận do Internet cung cấp

- 2 giao thức giao vận
  - TCP
    - Dịch vụ hướng kết nối (connection-oriented service)
      - Thủ tục bắt tay
      - Kết nối song công hoàn toàn (full-duplex)
    - Dịch vụ truyền dữ liệu tin cây
      - Truyền dữ liệu không lỗi và theo trật tự
    - Ngoài ra có cơ chế kiểm soát tắc nghẽn

# Các dịch vụ giao vận do Internet cung cấp

- 2 giao thức giao vận
  - UDP
    - Dịch vụ phi kết nối (Connectionless)
      - Không có thủ tục bắt tay
    - Dịch vụ truyền dữ liệu không tin cậy (Unreliable data transfer service)
      - Không đảm bảo gói tin có đến được tiến trình đích
      - Có thể không theo trật tự đúng
    - Không có cơ chế kiểm soát tắc nghẽn

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

# Các dịch vụ giao vận do Internet cung cấp

- Các dịch vụ không được cung cấp bởi UDP và TCP
  - Tin cậy
  - **Thông lượng**
  - **Thời gian thực**
  - An toàn
    - SSL

# Các dịch vụ giao vận do Internet cung cấp

<b>Ứng dụng</b>	<b>Giao thức tầng ứng dụng</b>	<b>Giao thức tầng giao vận</b>
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

# Nội dung

- Các nguyên tắc của các ứng dụng mạng
  - Kiến trúc ứng dụng
  - Truyền thông giữa các tiến trình
  - Các dịch vụ giao vận khả dụng cho các ứng dụng
  - Các dịch vụ giao vận do Internet cung cấp
  - Các giao thức tầng ứng dụng

# Review

1. Các ứng dụng mạng hiện đại sử dụng 2 kiến trúc ứng dụng phổ biến là.....và.....
2. Các dịch vụ giao vận khả dụng cho các ứng dụng là.....
3. Các ứng dụng có yêu cầu thông lượng cụ thể được gọi là....., và các ứng dụng không có yêu cầu cụ thể về thông lượng thì được gọi là....
4. Một tiến trình gửi thông điệp lên mạng hay nhận thông điệp từ mạng phải đi qua một giao diện phần mềm được gọi là.....hay còn gọi là....giữa ứng dụng và mạng

# Review

5. Socket là giao diện giữa.....và.....trong một host
6. Socket là giao diện lập trình mà ở đó các....được xây dựng.
7. Ở phía tầng.....người phát triển ứng dụng kiểm soát mọi thứ của socket nhưng ở phía tầng.....lại kiểm soát rất ít thứ của socket
8. Mỗi tiến trình nhận được định danh bởi....

# Review

9. Communication giữa các host trong network là....
10. Process là ....
11. Process khởi tạo communication gọi là..
12. Process chờ được communicate gọi là..
13. Ví dụ HTTP client process là...
14. Ví dụ HTTP server process là...
15. Trong kiến trúc P2P có cả tiến trình....và tiến trình.....
16. Các process trên các host khác nhau communicate với nhau bằng cách...

# Các giao thức tầng ứng dụng

- Thông điệp
  - Cấu trúc?
  - Ý nghĩa của các trường khác nhau trong thông điệp?
  - Khi nào các tiến trình gửi thông điệp?

**Application-layer protocols**

# Các giao thức tầng ứng dụng

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

# Các giao thức tầng ứng dụng

- 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

# Các giao thức tầng ứng dụng

- Phân biệt
  - Giao thức tầng ứng dụng và ứng dụng mạng
    - Giao thức tầng ứng dụng chỉ là một phần của ứng dụng mạng (phần rất quan trọng)
  - Ví dụ
    - Ứng dụng Web
      - Ứng dụng client-server
      - Cho phép người dùng lấy các tài liệu từ Web server theo yêu cầu
      - Gồm nhiều thành phần
        - » Một chuẩn cho các định dạng tài liệu (HTML)
        - » Các trình duyệt Web (Firefox, IE)
        - » Web server (Apache)
        - » **Giao thức tầng ứng dụng (HTTP)**

# Nội dung

- Các nguyên tắc của các ứng dụng mạng
- **Web và HTTP**
- FTP
- Thư điện tử trên Internet
- DNS – Internet's Directory Service
- Các ứng dụng P2P (Peer – to- Peer)
- Video Streaming và các mạng phân phối nội dung (Content Distribution Networks)
- Lập trình socket với UDP và TCP

- Giới thiệu chung về HTTP
  - Giao thức truyền siêu văn bản (HTTP), giao thức tầng ứng dụng của Web, là trung tâm của Web.
  - Được định nghĩa trong [RFC 1945] và [RFC 2616].
  - HTTP được triển khai ở hai chương trình
    - Chương trình máy khách.
    - Chương trình máy chủ

- **Trang web** bao gồm một số đối tượng (*object*)
  - Tệp HTML, ảnh JPEG, ứng dụng, video clip...
  - Ví dụ
    - Nếu một trang web chứa văn bản HTML và 5 ảnh jpg *khi đó* trang web có 6 đối tượng
  - Tệp HTML cơ sở tham chiếu đến các đối tượng khác trong trang bằng URL của đối tượng

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

Tên host

Tên đường dẫn

# Web và HTTP

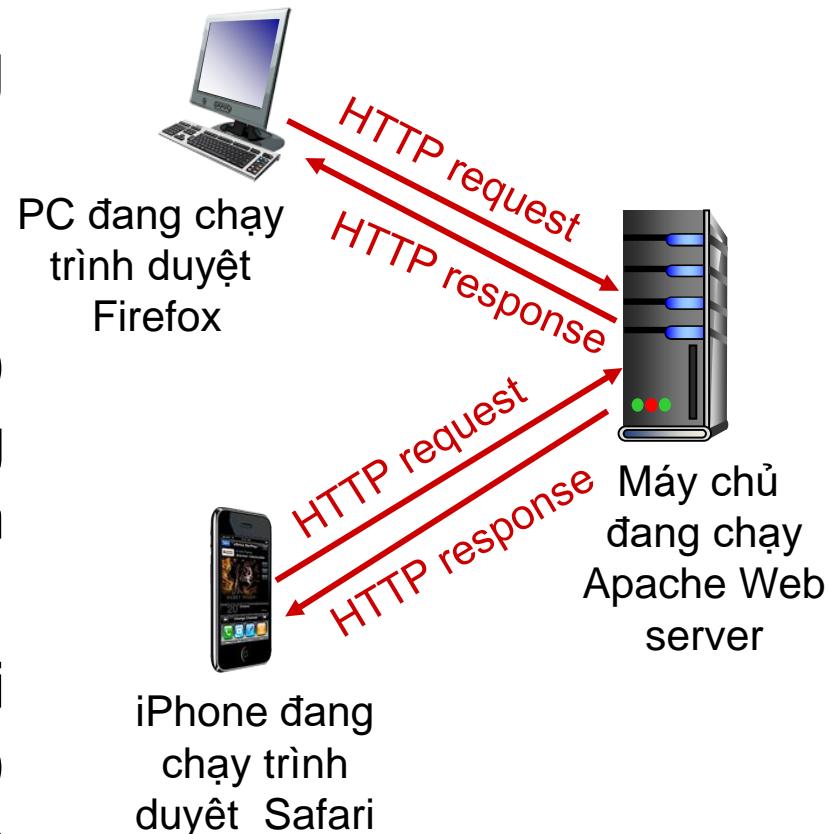
- HTTP: hypertext transfer protocol

- 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 gửi (sử dụng giao thức HTTP) các đối tượng đáp ứng theo yêu cầu.



# Web và HTTP

- HTTP sử dụng 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 giao thức “phi trạng thái” (**stateless protocol**)
  - 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

- Các kết nối bền vững và không bền vững (Persistent and Non-Persistent)
  - Bền vững
    - Tất cả các requests và các responses tương ứng được gửi trên cùng một kết nối TCP
  - Không bền vững
    - Mỗi cặp request/response được gửi qua một kết nối TCP riêng.

# Kết nối HTTP

- HTTP với kết nối không bền vững
  - Giả sử người dùng gõ URL:  
[www.someSchool.edu/someDepartment/home.index](http://www.someSchool.edu/someDepartment/home.index)  
(chứa 1 văn bản tham chiếu tới 10 ảnh jpeg)
    1. HTTP client khởi tạo kết nối TCP tới HTTP server (tiến trình) tại [www.someSchool.edu](http://www.someSchool.edu) trên cổng 80
    2. HTTP server tại host www.someSchool.edu, đang chờ kết nối TCP tại cổng 80, “chấp nhận” kết nối, thông báo cho client

# Kết nối HTTP

- HTTP không bền vững
  - Giả sử người dùng gõ URL:  
[www.someSchool.edu/someDepartment/home.index](http://www.someSchool.edu/someDepartment/home.index)
    3. 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
    4. HTTP server nhận thông điệp yêu cầu, định dạng *thông điệp đáp ứng* chưa đối tượng được yêu cầu, và gửi thông điệp vào trong socket của nó
    5. HTTP server đóng kết nối TCP.

# Kết nối HTTP

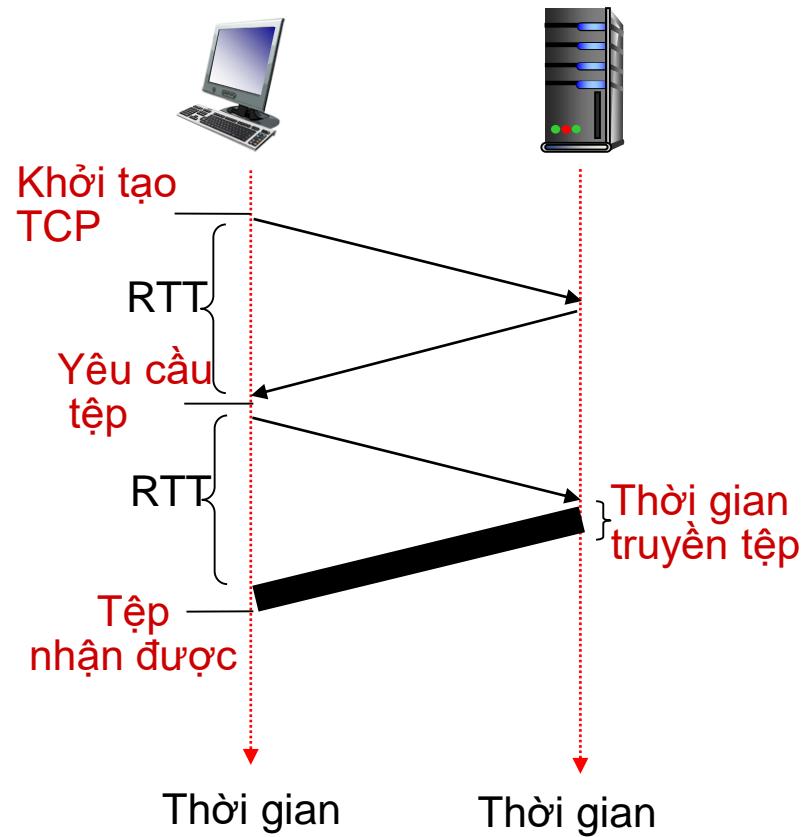
- HTTP không bền vững
  - Giả sử người dùng gõ URL:  
[www.someSchool.edu/someDepartment/home.index](http://www.someSchool.edu/someDepartment/home.index)
    6. HTTP client nhận thông điệp đáp ứng chứa tệp HTML, hiển thị HTML. Phân tích tệp HTML, tìm 10 đối tượng jpeg được tham chiếu.
    7. Các bước từ 1 đến 6 được lặp lại cho từng đối tượng trong 10 đối tượng jpeg

# Web và HTTP

- HTTP với các kết nối 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

# Web và HTTP

- HTTP không bền vững: thời gian đáp ứng
  - RTT (round-trip time): thời gian để một gói tin nhỏ đi từ client đến server và quay lại.
    - Gồm cả packet-propagation, packet-queuing và trễ packet-processing
  - Thời gian đáp ứng của HTTP không bền vững =  $2RTT + \text{thời gian truyền file HTML}$



# Web và HTTP

- HTTP với kết nối bền vững
  - Vấn đề với HTTP không bền vững:
    - Đặt gánh nặng cho máy chủ Web
    - 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 với kết nối bền vững
  - Server để mở kết nối sau khi gửi một response (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

**Chế độ mặc định của HTTP** sử dụng các **kết nối bền vững**

# Web và HTTP

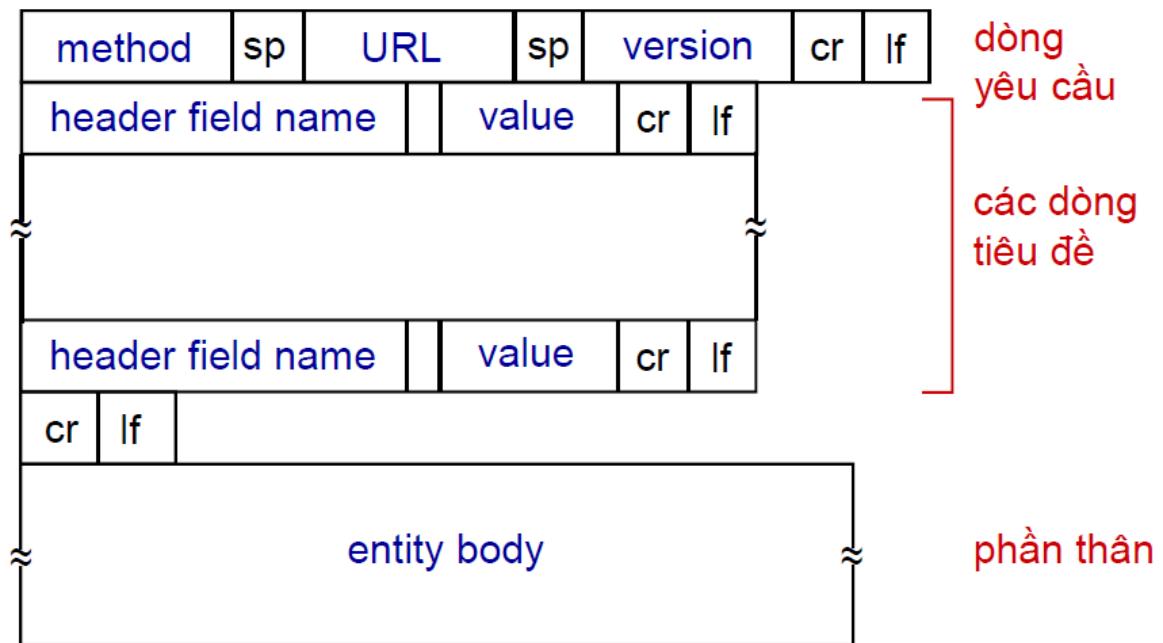
- Định dạng thông điệp HTTP
  - RFC 1945; RFC 2616; RFC 7540
  - Có hai loại thông điệp HTTP
    - Yêu cầu – Request
    - Đáp ứng - Response

# Web và HTTP

- Thông điệp HTTP yêu cầu

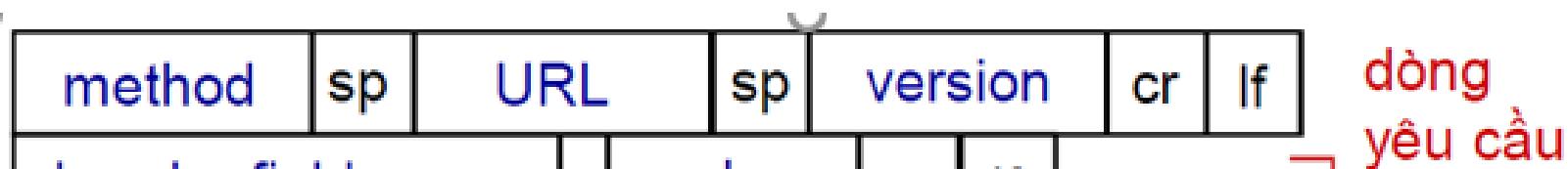
- Định dạng chung

- Dòng yêu cầu
    - Dòng tiêu đề
    - Phần thân



# Web và HTTP

- Thông điệp HTTP yêu cầu
  - Dòng yêu cầu có 3 trường
    - Trường phương thức – method
      - GET, POST, HEAD, PUT và DELETE. Hai phương thức phổ biến nhất là GET và POST.
    - Trường URL (đường dẫn)
      - Dùng để định danh nguồn tài nguyên mà client yêu cầu, bắt buộc phải có ít nhất là dấu “/”.
    - Trường phiên bản HTTP – version
      - Là phiên bản HTTP client đang sử dụng (thường là HTTP/1.0 hoặc HTTP/1.1)



# Web và HTTP

- Ví dụ thông điệp HTTP yêu cầu:
  - ASCII (định dạng con người có thể đọc được)

```
GET /somedir/page.html HTTP/1.1
```

```
Host: www.someschool.edu
```

```
Connection: close
```

```
User-agent: Mozilla/5.0
```

```
Accept-language: fr
```

# Web và HTTP

- Thông điệp HTTP yêu cầu
  - Các dòng tiêu đề Header
    - Các dòng này là không bắt buộc, viết ở định dạng **“Name:Value”**
    - Cho phép client gửi thêm các thông tin bổ sung về thông điệp HTTP request và thông tin về chính client.
    - Một số header thông dụng như:
      - Accept: loại nội dung có thể nhận được từ thông điệp response. Ví dụ: text/plain, text/html
      - Accept-Encoding: các kiểu nén được chấp nhận. Ví dụ: gzip, deflate, xz, exi...
      - Connection: tùy chọn điều khiển cho kết nối hiện thời. Ví dụ: Keep-Alive, Close...
      - Cookie: thông tin HTTP Cookie từ server

- Thông điệp HTTP yêu cầu
  - Phần thân
    - Là dữ liệu gửi từ client đến server trong gói tin HTTP request.
    - Đa số các gói tin gửi theo phương thức GET sẽ có Body trống, các phương thức như POST hay PUT thường dùng để gửi dữ liệu nên sẽ bao gồm dữ liệu trong trường Body.

# Web và HTTP

- Thông điệp đáp ứng HTTP
  - Gồm 3 phần

Dòng trạng thái  
(Giao thức  
mã trạng thái,  
cụm từ trạng  
thái)

Các dòng  
tiêu đề

Dữ liệu,  
ví dụ  
tệp HTML  
yêu cầu

```
HTTP/1.1 200 OK\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS)\r\nLast-Modified: Tue, 30 Oct 2007 17:00:02  
GMT\r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-  
1\r\n\r\n
```

data data data data data ...

- Thông điệp đáp ứng HTTP
  - Dòng trạng thái gồm 3 trường
    - Phiên bản giao thức (HTTP version)
      - Phiên bản của giao thức HTTP mà server hỗ trợ, thường là HTTP/1.0 hoặc HTTP/1.1
    - Mã trạng thái (Status code)
      - Mô tả trạng thái kết nối dưới dạng số, mỗi trạng thái sẽ được biểu thị bởi một số nguyên. Ví dụ: 200, 404, 302,...
    - Mô tả trạng thái (Status text)
      - Mô tả trạng thái kết nối dưới dạng văn bản một cách ngắn gọn, giúp người dùng dễ hiểu hơn so với mã trạng thái.
      - Ví dụ: 200 OK, 404 Not Found, 403 Forbiden,...

- Thông điệp đáp ứng HTTP
  - Các dòng tiêu đề
    - Có chức năng tương tự như gói tin request
    - Giúp server có thể truyền thêm các thông tin bổ sung đến client dưới dạng các cặp “**Name:Value**”.
  - Phần Body
    - Là nơi đóng gói dữ liệu để trả về cho client, thông thường trong duyệt web thì dữ liệu trả về sẽ ở dưới dạng một trang HTML để trình duyệt có thể thông dịch được và hiển thị ra cho người dùng.

# Web và HTTP

- Tự kiểm tra Web server (phía client)
  - Telnet đến Web server ưa thích

`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

`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 **tùy**

- Tương tác User-server: các cookie
  - **Cookies** là gì?
    - Cookies [RFC 6265] là file văn bản chứa thông tin của người dùng mà một Web server gửi đến và lưu trên máy tính của người dùng.
      - Cookie được lưu trữ ở đâu trên máy tính tùy vào trình duyệt và hệ điều hành sử dụng.
    - Trình duyệt web sẽ lưu và gửi lại máy chủ mỗi khi người dùng truy cập vào máy chủ.

# Web và HTTP

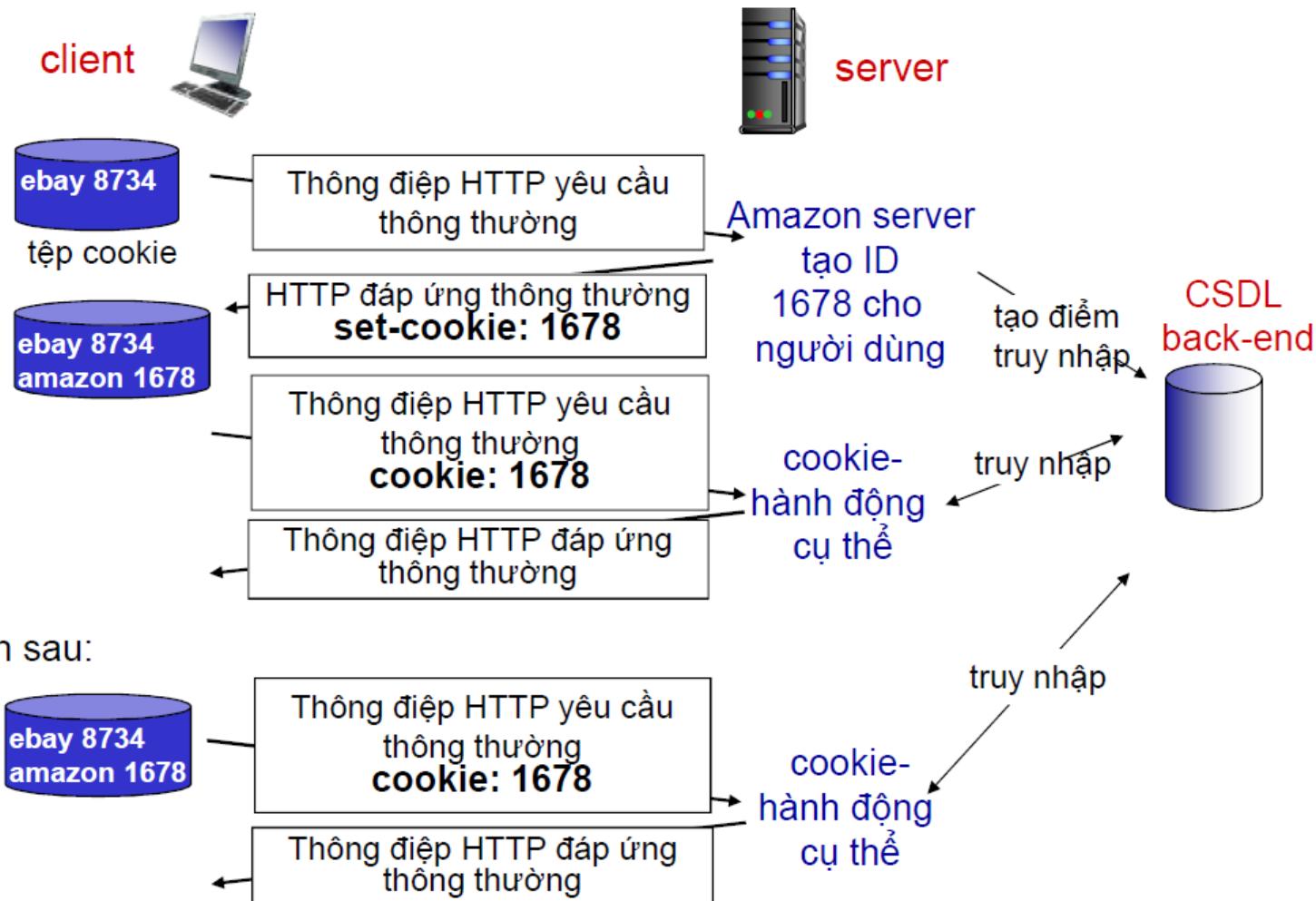
- Trạng thái User-server: các cookie
  - Ví dụ:
    - Susan thường truy nhập Internet từ một PC
    - Cô ấy vào một trang thương mại điện tử lần đầu tiên
    - Khi yêu cầu HTTP khởi tạo đi đến trang, trang sẽ tạo ra:
      - Một ID duy nhất
      - Điểm truy nhập vào cơ sở dữ liệu back-end cho ID
    - => Các thông tin này sẽ được lưu vào file nhỏ gọi là cookies. Ở lần truy nhập sau, khi susan truy cập vào trang này thì thông tin cookies sẽ được trình duyệt gửi tới server chứ không phải dùng phương thức GET hay POST.

# Web và HTTP

- Tương tác User-server: các cookie
  - Bốn thành phần của cookies
    - Dòng tiêu đề cookie trong thông điệp đáp ứng HTTP
    - Dòng tiêu đề cookie trong thông điệp yêu cầu HTTP tiếp theo
    - Tập cookie được lưu trên host của người dùng, được quản lý bởi trình duyệt của người dùng.
    - Cơ sở dữ liệu back-end tại Web site

# Web và HTTP

- Các cookie: lưu giữ “trạng thái”

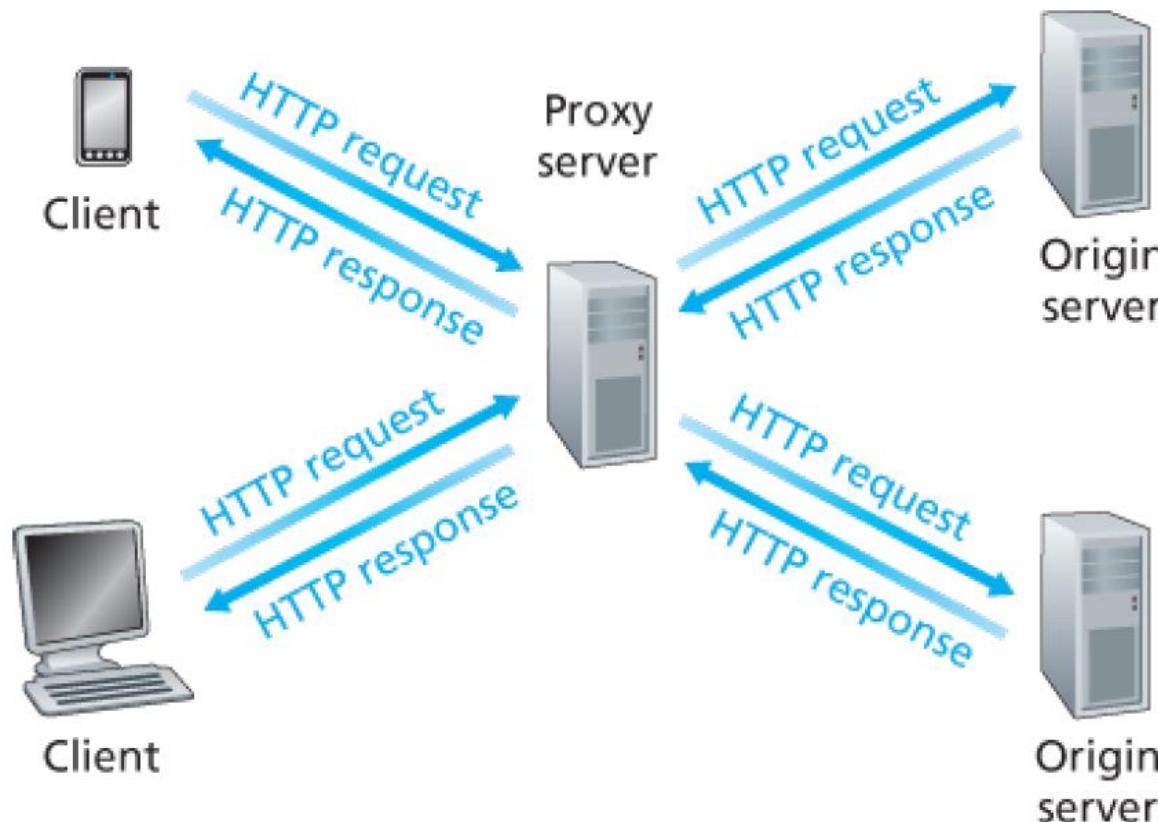


một tuần sau:

- Web caches (proxy server)
  - *Mục đích:* thỏa mãn yêu cầu của client mà không cần liên quan đến server nguồn
  - Web cache có ổ lưu trữ riêng và lưu các bản sao của các đối tượng được yêu cầu trong ổ lưu trữ này.
  - Có thể cấu hình trình duyệt của người dùng để tắt cả các yêu cầu HTTP của người dùng trước tiên được chuyển hướng đến Web cache.

# Web và HTTP

- Web caches (proxy server)
  - Ví dụ



***http://www.someschool.edu/campus.gif***

- Web caches
  - Bộ nhớ cache hoạt động đồng thời ở cả client và server
    - Khi nó nhận các requests từ và gửi các responses tới một trình duyệt thì nó là server.
    - Khi nó gửi các requests tới và nhận các responses từ một server nguồn thì nó là client.
  - Web cache thường được mua và cài đặt bởi ISP (trường học, công ty, khu dân cư)
    - Ví dụ một trường đại học có thể cài đặt một Web cache trên mạng khuôn viên của trường và cấu hình tất cả các trình duyệt trong mạng trỏ tới cache.

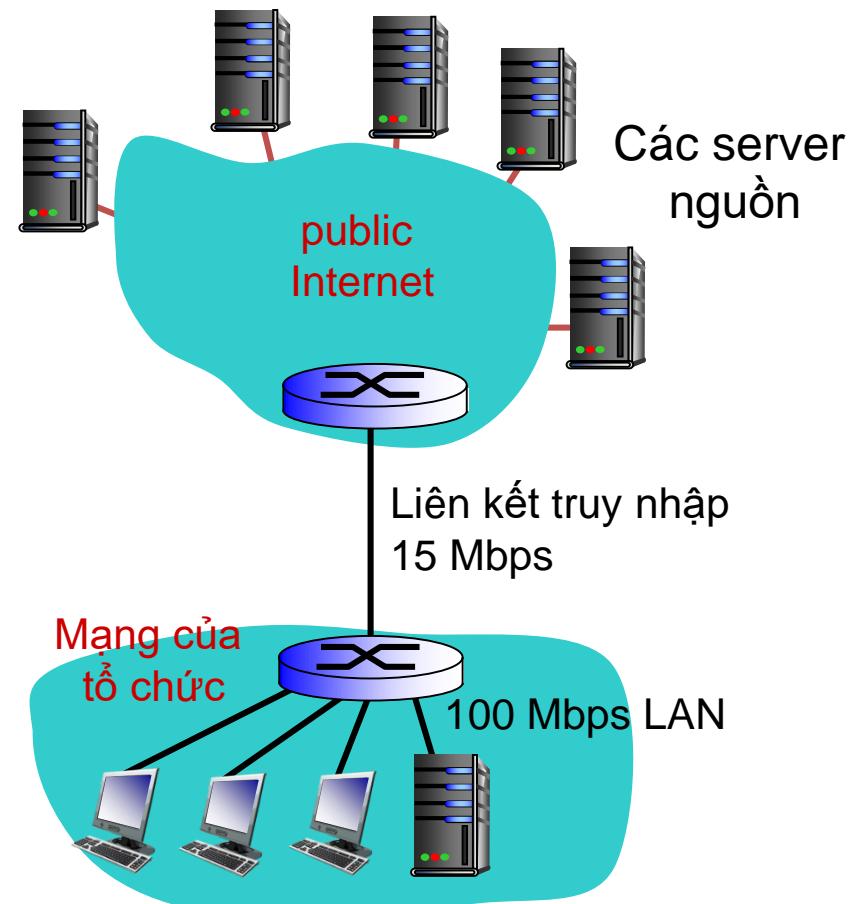
- Tại sao lại cần Web caching?
  - Làm giảm thời gian đáp ứng cho yêu cầu của client
  - Làm giảm lưu lượng trên một liên kết truy nhập của tổ chức
  - Về cơ bản có thể làm giảm đáng kể lưu lượng truy cập Web trên Internet nói chung, do đó cải thiện hiệu suất cho tất cả các ứng dụng.

# Web và HTTP

- Ví dụ caching

- *Giả thiết:*

- Dung lượng trung bình của đối tượng: 1Mb
    - Tốc độ yêu cầu trung bình từ các trình duyệt tới server nguồn: 15 request /giây
    - RTT từ bộ định tuyến của tổ chức đến server nguồn bất kỳ: **2 giây**
    - Tốc độ của liên kết truy nhập: 15 Mbps



**Tổng thời gian response = LAN delay + access delay + Internet delay**

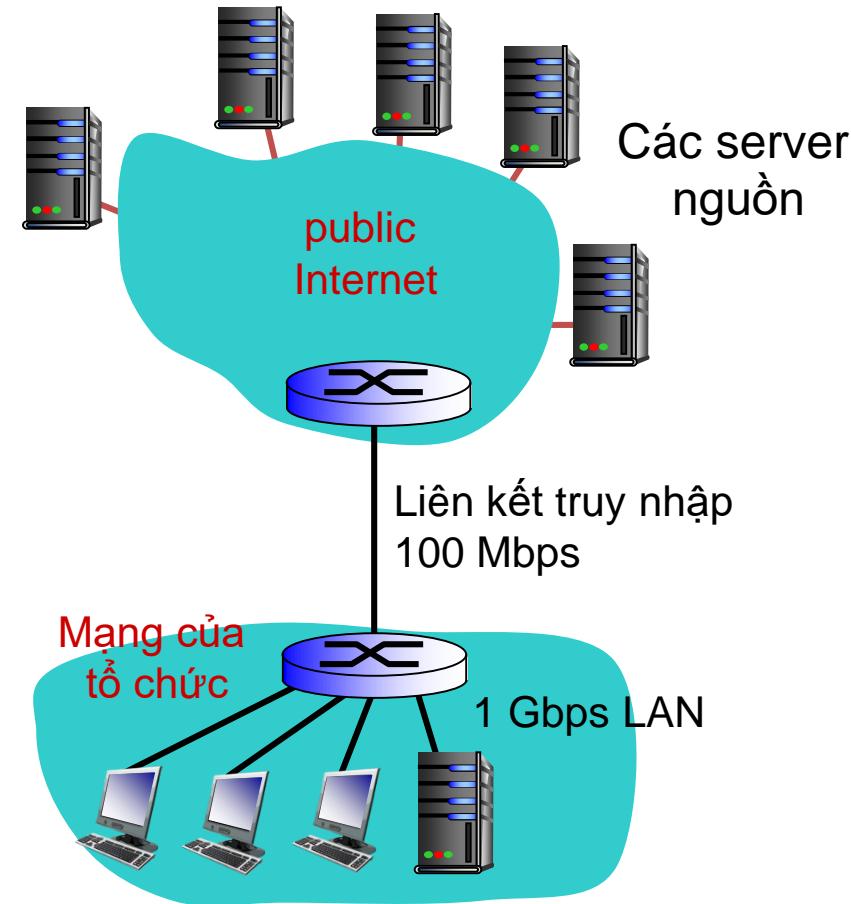
- Ví dụ caching
  - Cường độ lưu lượng trên LAN  
 $(15 \text{ requests/sec}) * (1 \text{ Mb/request}) / (100 \text{ Mbps}) = 0.15$
  - Cường độ lưu lượng trên liên kết truy cập (từ router Internet tới router của tổ chức)  
 $(15 \text{ requests/sec}) * (1 \text{ Mb/request}) / (15 \text{ Mbps}) = 1$ 

⇒ Cường độ lưu lượng = 0.15 trên LAN thường dẫn đến độ trễ nhiều nhất là vài chục mili giây, do đó ta có thể bỏ qua trễ LAN

⇒ Cường độ lưu lượng trên liên kết truy cập tiến tới 1 thì trễ trên liên kết sẽ rất lớn. Do đó thời gian response trung bình để thỏa mãn các request tầm vài phút.

# Web và HTTP

- Ví dụ caching
  - **Giải pháp 1:** Tăng tốc độ truy cập từ 15 Mbps lên 100 Mbps  
⇒ giảm cường độ lưu lượng truy cập trên liên kết xuống 0.15 ⇔ trễ không đáng kể giữa 2 router  
⇒ tổng thời gian response ≈ 2 giây



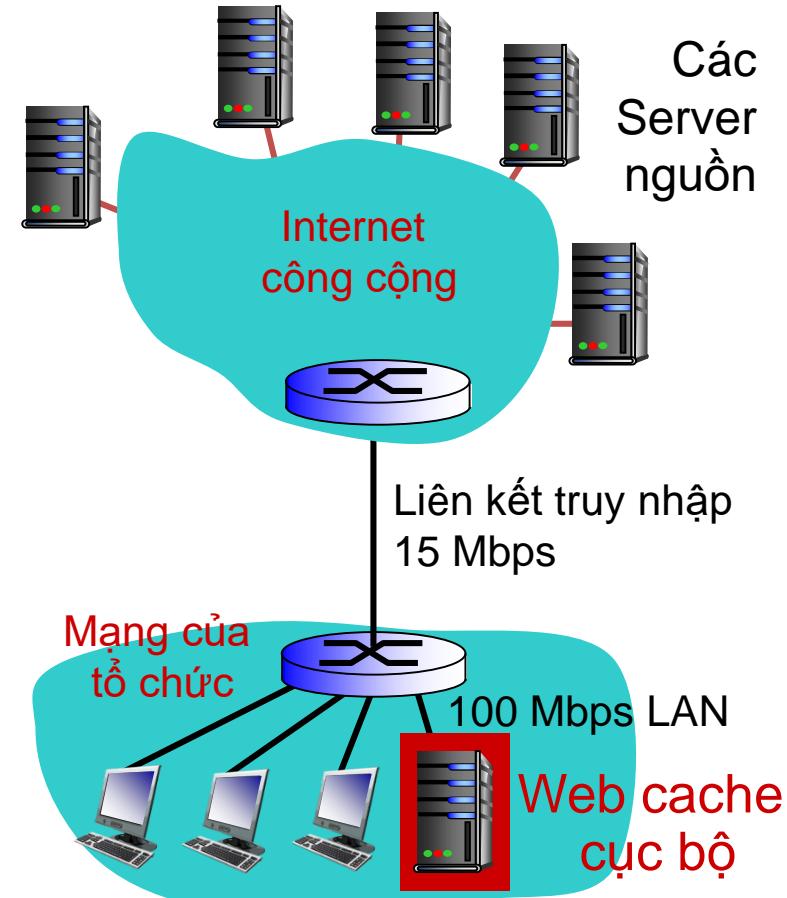
**Giải pháp 1 khá tốn kém**

# Web và HTTP

- Ví dụ caching
  - **Giải pháp 2:** Cài đặt Web cache trong mạng tổ chức
  - Tốc độ truy cập: 0.4

Gia sử 40% request trong LAN được đáp ứng trong 10 ms

$$\Rightarrow \text{Trễ trung bình} = 0.4 \cdot (0.01 \text{ s}) + 0.6 \cdot (2.01 \text{ s}) \sim 1.2 \text{ giây}$$



**Giải pháp 2 cung cấp thời gian phản hồi thấp hơn so với giải pháp đầu tiên và nó không yêu cầu tổ chức phải nâng cấp băng thông liên kết**

- GET có điều kiện
  - Mục tiêu: không gửi đối tượng nếu cache chưa cập nhật
  - Một thông điệp HTTP request được gọi là thông điệp GET có điều kiện nếu
    - Thông điệp request này sử dụng phương thức GET
    - Thông điệp request gồm dòng tiêu đề  
**If-Modified-Since: header line**

# Web và HTTP

client



**GET /fruit/kiwi.gif HTTP/1.1**  
**Host: www.exotiquecuisine.com**

1

**HTTP/1.1 200 OK**  
**Date: Sat, 3 Oct 2015 15:39:29**  
**Server: Apache/1.3.0 (Unix)**  
**Last-Modified: Wed, 9 Sep 2015 09:23:24**  
**Content-Type: image/gif**  
  
**(data data data data data ...)**

server

**GET /fruit/kiwi.gif HTTP/1.1**  
**Host: www.exotiquecuisine.com**  
**If-modified-since: Wed, 9 Sep 2015 09:23:24**

2

client

**HTTP/1.1 304 Not Modified**  
**Date: Sat, 10 Oct 2015 15:39:29**  
**Server: Apache/1.3.0 (Unix)**  
**(empty entity body)**

3

server

**HTTP/1.1 304 Not Modified**  
**Date: Sat, 10 Oct 2015 15:39:29**  
**Server: Apache/1.3.0 (Unix)**  
**(empty entity body)**

4

# Review Web và HTTP

1. Giao thức tầng ứng dụng của Web là....
2. Giao thức giao vận Web sử dụng là.....
3. Một trang web có thể chứa nhiều.....và mỗi .....web được xác định bởi địa chỉ.....
4. Tất cả các HTTP requests và các HTTP responses tương ứng được gửi trên cùng một kết nối TCP được gọi là kết nối....
5. Mỗi cặp HTTP request/response được gửi qua một kết nối TCP riêng được gọi là kết nối....
6. Kết nối HTTP thuộc kết nối.....

# Review Web và HTTP

7. Có 2 loại thông điệp HTTP đó là....
8. Thông điệp được gửi từ web client tới web server được gọi là thông điệp.....
9. Thông điệp được gửi từ web server tới web client được gọi là thông điệp.....
10. 3 phần chính của thông điệp HTTP request là....
11. Phương thức GET trong thông điệp HTTP request được đặt ở dòng.....trong trường  
.....

# Review Web và HTTP

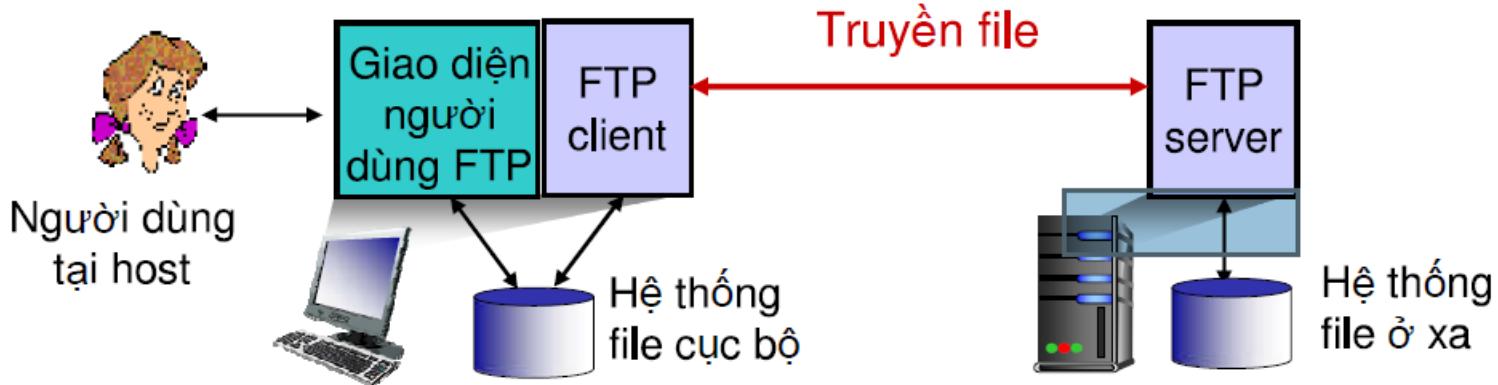
12. 3 thành phần chính của thông điệp HTTP response là.....
13. Trình duyệt web trên máy client sẽ lưu thông tin người dùng vào..... Mỗi khi người dùng truy cập vào web server, thông tin này sẽ được trình duyệt gửi cho web server.
14. Các request thay vì được gửi tới Web server nguồn có thể được gửi tới.....để giảm tải cho server nguồn và liên kết.
15. Phương thức .....giúp Web cache kiểm tra đối tượng mình lưu có được cập nhật theo web server nguồn hay không.

# Nội dung

- Các nguyên tắc của các ứng dụng mạng
- Web và HTTP
- **FTP**
- Thư điện tử trên Internet
- DNS – Internet's Directory Service
- Các ứng dụng P2P (Peer – to- Peer)
- Video Streaming và các mạng phân phối nội dung (Content Distribution Networks)
- Lập trình socket với UDP và TCP

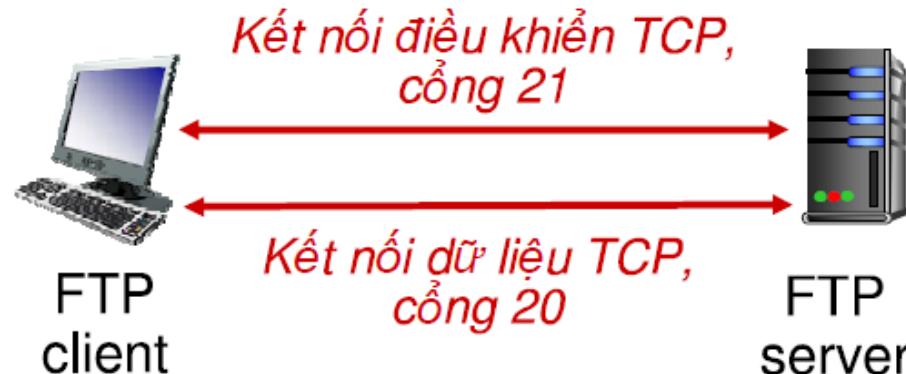
# FTP: Giao thức truyền file

- 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



- Kết nối dữ liệu và kết nối điều khiển riêng biệt nhau
  - FTP client tiếp xúc FTP server tại cổng 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) để gửi file tới client
  - Truyền xong file, server đóng kết nối dữ liệu

- Kết nối dữ liệu và kết nối điều khiển riêng biệt nhau
  - 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



- Mô hình FTP chia phần mềm trên mỗi thiết bị thành 2 thành phần giao thức logic chịu trách nhiệm cho mỗi kênh:
  - **Thành phần protocol interpreter (PI):** Là thành phần quản lý kênh điều khiển, phát và nhận lệnh và trả lời.
  - **Thành phần data transfer process (DTP):** chịu trách nhiệm gửi và nhận dữ liệu giữa client và server.
  - Ngoài hai thành phần trên, tiến trình phía client có 1 thành phần thứ ba là giao diện người dùng (user interface) dùng để tương tác với người dùng FTP, thành phần này không có ở sever.

- Các tiến trình Client
  - User Interface
    - Chương trình được chạy trên máy tính, cung cấp giao diện xử lí cho người dùng
      - Cho phép sử dụng các lệnh đơn giản hướng người dùng
      - Cho phép người dùng điều khiển phiên FTP theo dõi các thông tin và kết quả xảy ra trong tiến trình.

- Các tiến trình Client
  - **User Protocol Interpreter (User-PI)**
    - Quản lý kênh điều khiển phía Client.
    - Khởi tạo phiên kết nối FTP bằng việc phát hiện ra yêu cầu tới phía server-PI. Khi kết nối được thiết lập, nó xử lí các lệnh nhận được trên giao diện người dùng, gửi chúng tới Server-PI, và nhận phản hồi trả lại.
    - Quản lý tiến trình User-DTP.

- Các tiến trình Client
  - **User Data Transfer Process (User-DTP)**
    - Là bộ phận DTP nằm ở phía người dùng, làm nhiệm vụ gửi hoặc nhận dữ liệu từ Server-DTP.
    - User-DTP có thể thiết lập hoặc lắng nghe yêu cầu kết nối kênh dữ liệu trên server. Nó tương tác với thiết bị lưu trữ file phía client.

- Tiến trình bên phía server
  - **Server Protocol Interpreter (Server-PI)**
    - Quản lý điều khiển kết nối trên server.
    - Lắng nghe yêu cầu kết nối hướng từ users trên cổng dành riêng. Khi kết nối đã được thiết lập, nó nhận lệnh từ User-PI, gửi trả lời lại và quản lý tiến trình truyền dữ liệu trên server.

- Tiến trình bên phía server
  - **Server Data Transfer Process (Server-DTP)**
    - Gửi hoặc nhận file từ bộ phận User-DTP.
    - Server DTP vừa làm nhiệm vụ thiết lập kết nối kênh dữ liệu và lắng nghe một kết nối kênh dữ liệu từ user.
    - Nó tương tác với server file trên hệ thống cục bộ để đọc và chép file.

- Mô hình hoạt động của FTP

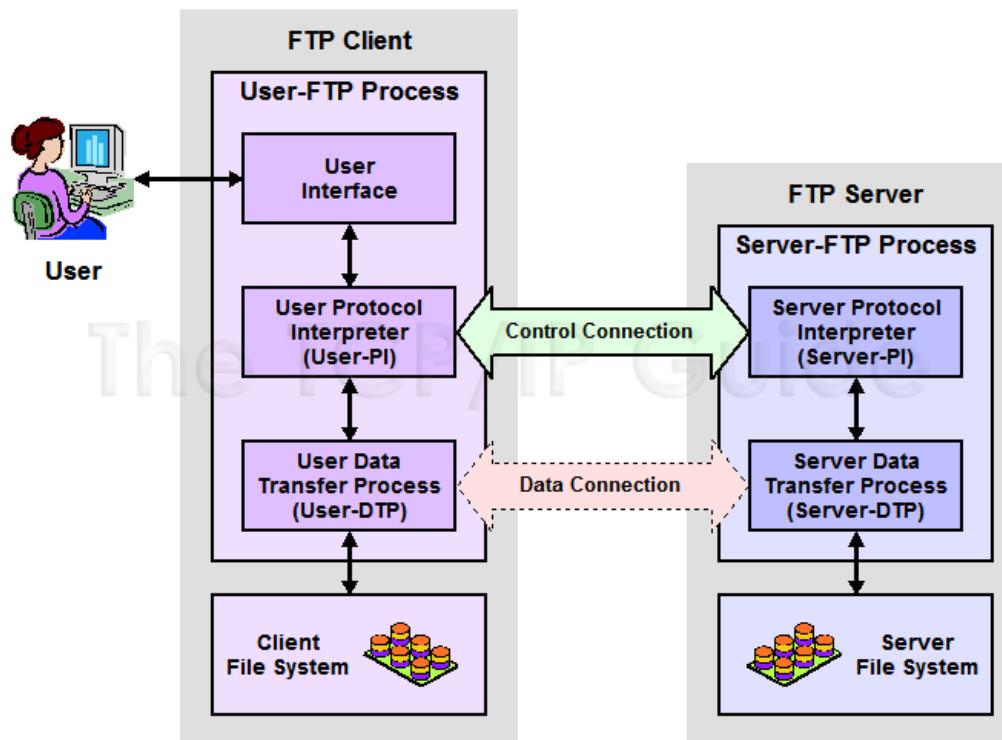


Figure 288: File Transfer Protocol (FTP) Operational Model

FTP is a client/server protocol, with communication taking place between the *User-FTP Process* on the client and the *Server-FTP Process* on the server. Commands, replies and status information are passed between the *User-PI* and *Server-PI* over the *control connection*, which is established once and maintained for the session. Data is moved between devices over *data connections* that are set up for each transfer.

- 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***
    - **PAS *password***
    - **LIST** trả về danh sách các file trong thư mục hiện hành
    - **RETR *filename*** trích chọn (lấy) file
    - **STOR *filename*** lưu (đặt) file vào host từ xa

- Các lệnh và đáp ứng của FTP
  - 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**

- Review questions
  - Kiến trúc ứng dụng?
  - Giao thức tầng giao vận?
  - Có những kết nối nào được thiết lập?
  - Hạn chế lớn nhất của FTP là gì?

# Nội dung

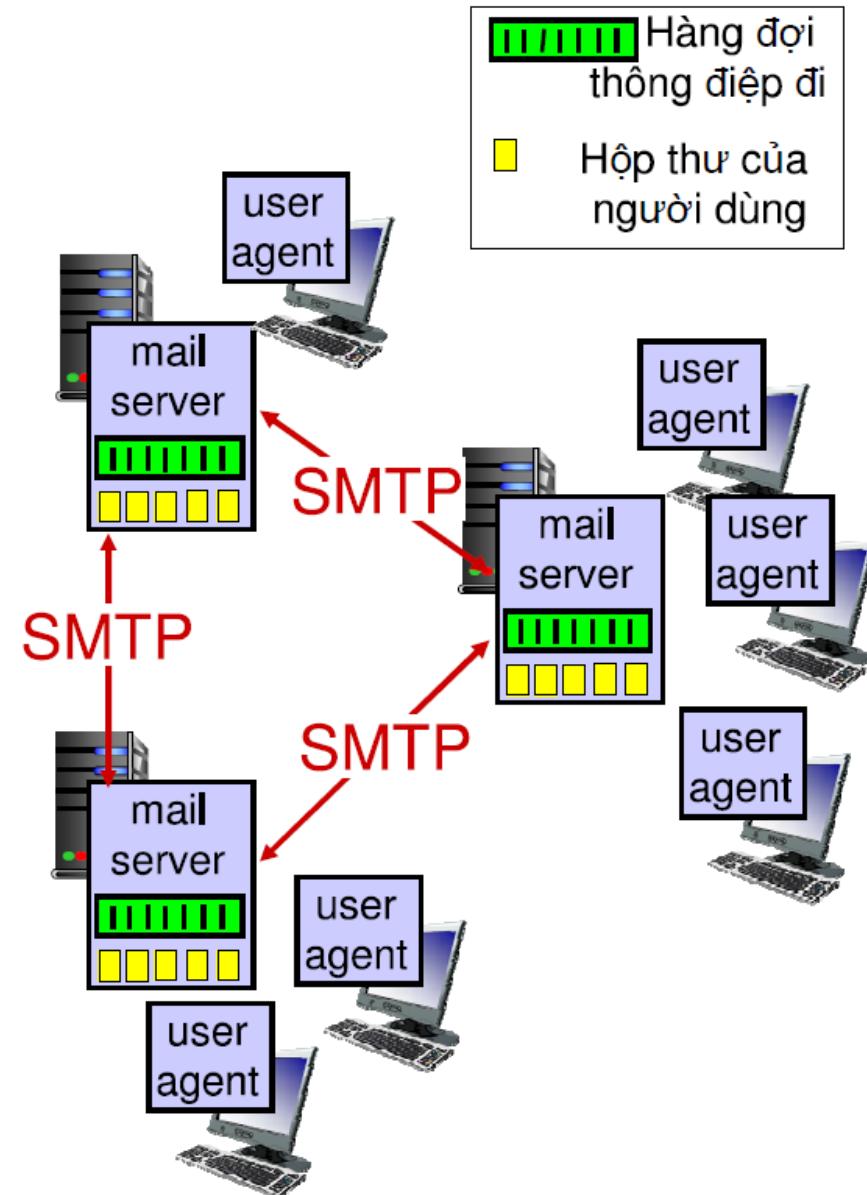
- Các nguyên tắc của các ứng dụng mạng
- Web và HTTP
- FTP
- **Thư điện tử trên Internet**
- DNS – Internet's Directory Service
- Các ứng dụng P2P (Peer – to- Peer)
- Video Streaming và các mạng phân phối nội dung (Content Distribution Networks)
- Lập trình socket với UDP và TCP

# Thư điện tử

- 3 thành phần chính
  - User agent
  - Mail server
  - Giao thức truyền thư đơn giản SMTP (simple mail transfer protocol)

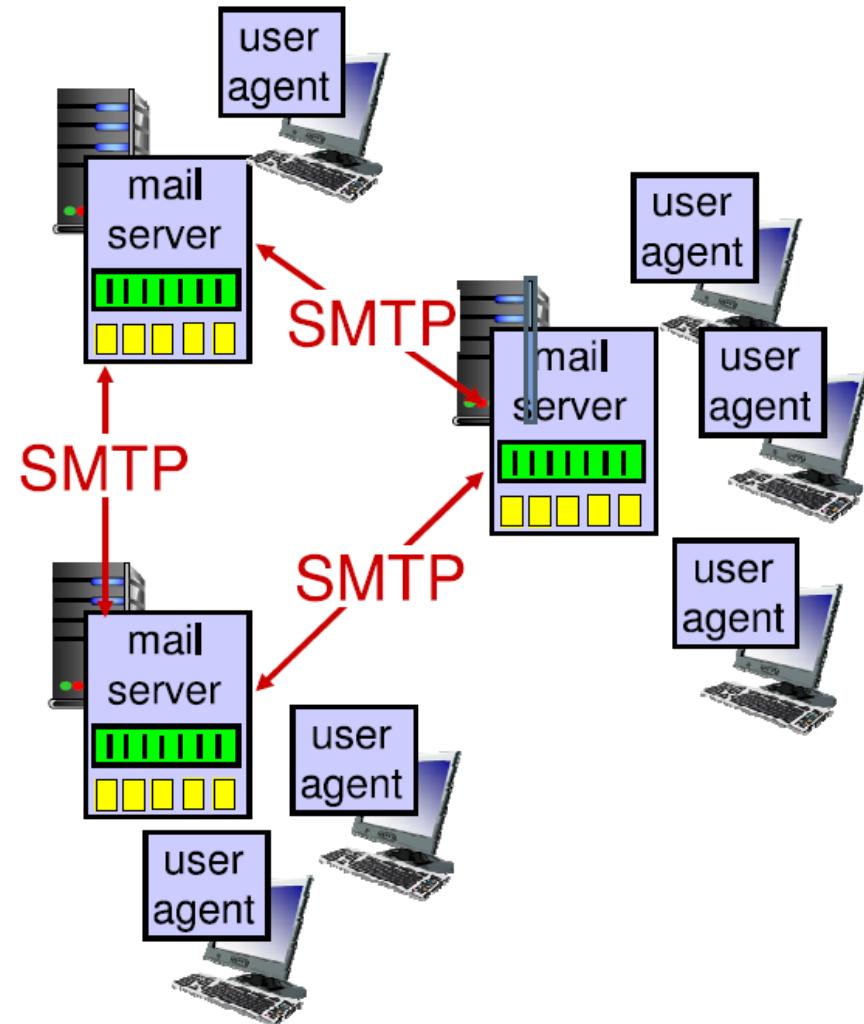
# Thư điện tử

- 3 thành phần chính
  - 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



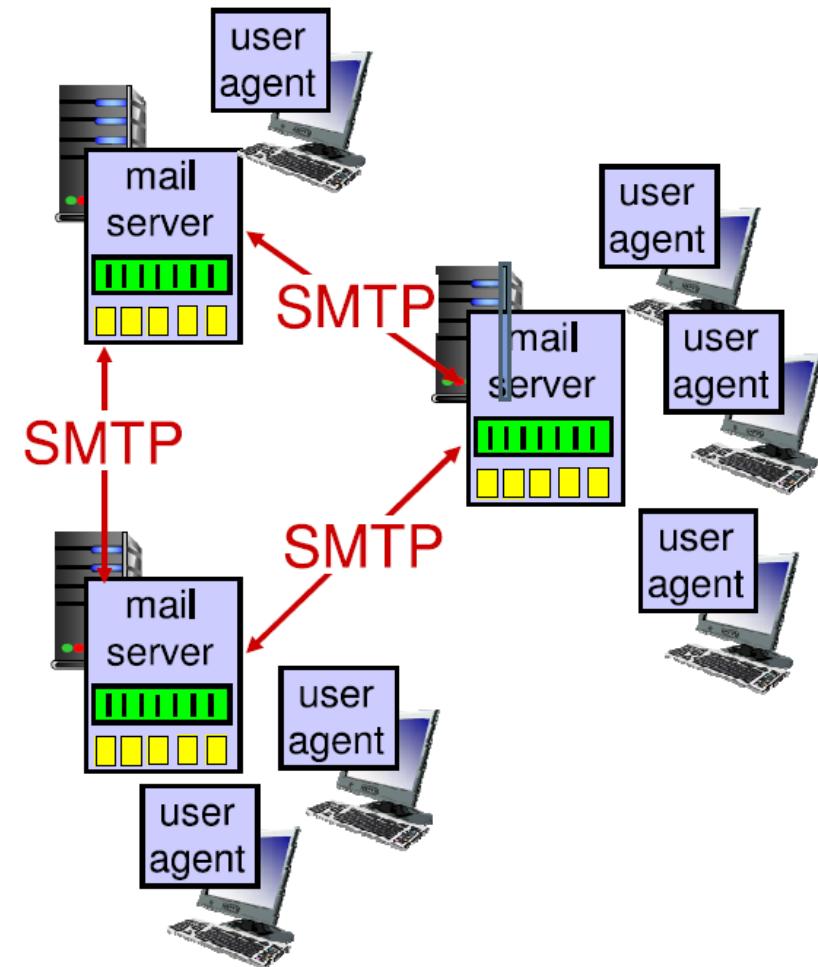
# Thư điện tử

- 3 thành phần chính
  - 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)



# Thư điện tử

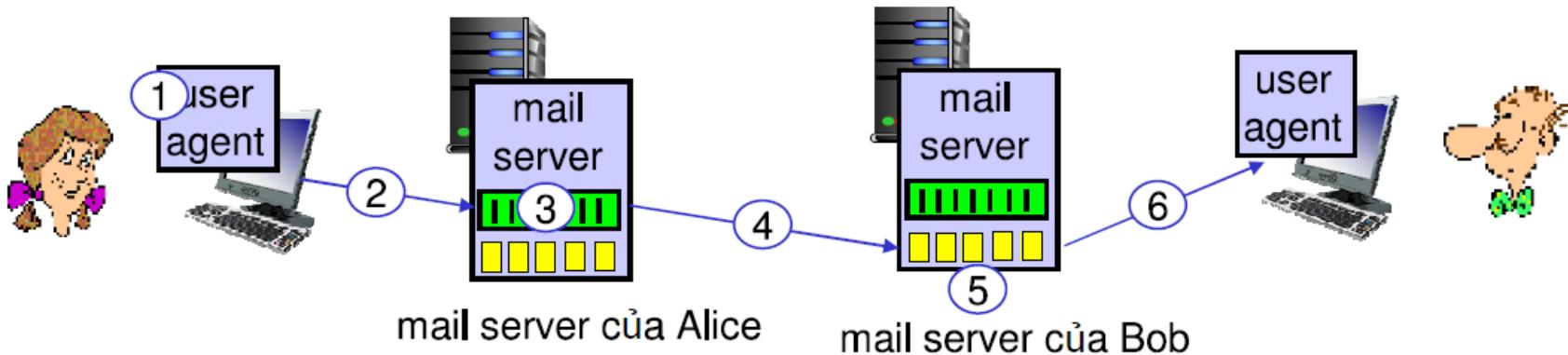
- 3 thành phần chính
  - Giao thức SMTP giữa các mail server thực hiện gửi các thông điệp thư
    - Client: gửi đến mail server
    - Server: mail server nhận



# Thư điện tử

## • Kịch bản: Alice gửi thông điệp tới Bob

- Alice dùng UA soạn thảo thông điệp để “gửi tới” [bob@someschool.edu](mailto:bob@someschool.edu)
- UA của Alice gửi thông điệp tới mail server của cô ấy; thông điệp được đặt trong hàng đợi thông điệp
- Phía SMTP client mở kết nối TCP tới mail server của Bob
- SMTP client gửi thông điệp của Alice qua kết nối TCP
- Mail server của Bob đặt thông điệp trong hộp thư của Bob
- Bob kích hoạt UA để đọc thông điệp thư



# 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-bít

# Thư điện tử

- 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

# Thư điện tử

- Thủ 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)

# Thư điện tử

- SMTP
  - SMTP sử dụng kết nối bền vững
  - SMTP thông điệp yêu cầu (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

# Thư điện tử

- So sánh SMTP với HTTP
  - Điểm giống
    - Sử dụng các kết nối liên tục
  - Điểm khác
    - HTTP: kéo
      - Ai đó tải thông tin lên máy chủ Web và người dùng sử dụng HTTP để **kéo** thông tin từ máy chủ
      - Kết nối TCP được khởi tạo bởi máy muốn nhận tệp tin
    - SMTP: đẩy
      - Server gửi thư đẩy tệp đến server nhận
      - Kết nối TCP là do máy muốn gửi tệp khởi tạo.

# Thư điện tử

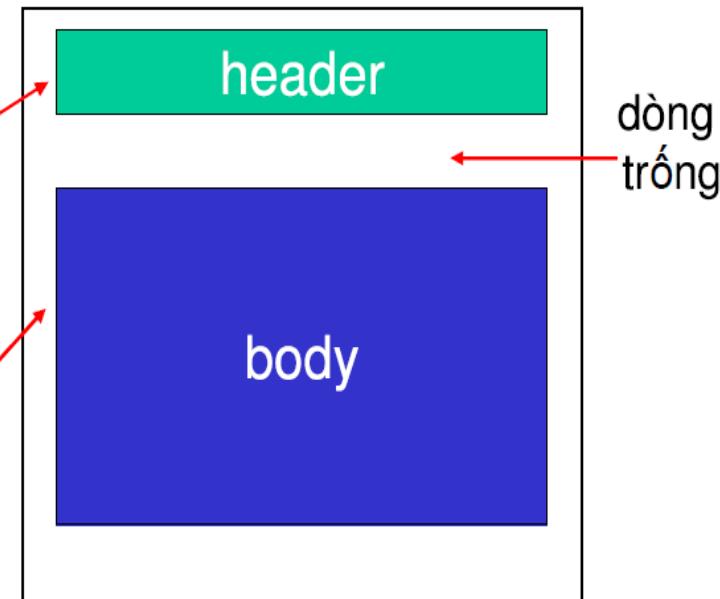
- So sánh SMTP với HTTP
  - Điểm khác
    - SMTP yêu cầu mỗi thông báo, gồm nội dung của mỗi thư, ở **định dạng ASCII 7-bit**. Nếu tin nhắn chứa các ký tự không phải là ASCII 7 bit hoặc chứa dữ liệu nhị phân thì thông báo phải được mã hóa thành ASCII 7-bit.
    - HTTP không áp đặt điều này.

# Thư điện tử

- So sánh SMTP với HTTP
  - Điểm khác
    - HTTP đóng gói từng đối tượng trong thông điệp HTTP response.
    - SMTP đặt tất cả các đối tượng của thông điệp vào một thông điệp

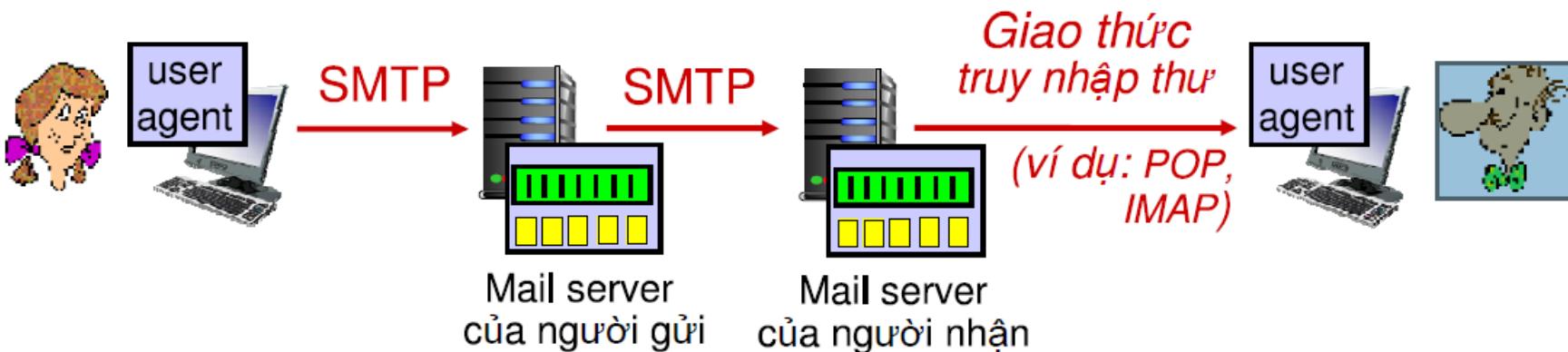
# Thư điện tử

- 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



# Thư điện tử

- 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 trên server
    - HTTP: gmail, Hotmail, Yahoo Mail,...



# Thư điện tử

## • Giao thức POP3

### – Giai đoạn xác thực.

- 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
```

```
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
```

### – Giai đoạn giao dịch, client:

- **List**: liệt kê các thông điệp
- **Retr**: trích xuất thông điệp theo số
- **Dele**: xóa
- **quit**

# Thư điện tử

- Giao thức POP3 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

# Thư điện tử

- Giao thức POP3 và IMAP
  - 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

# Nội dung

- Các nguyên tắc của các ứng dụng mạng
- Web và HTTP
- FTP
- Thư điện tử trên Internet
- **DNS – Domain Name Service**
- Các ứng dụng P2P (Peer – to- Peer)
- Video Streaming và các mạng phân phối nội dung (Content Distribution Networks)
- Lập trình socket với UDP và TCP

# Hệ thống tên miền (DNS)

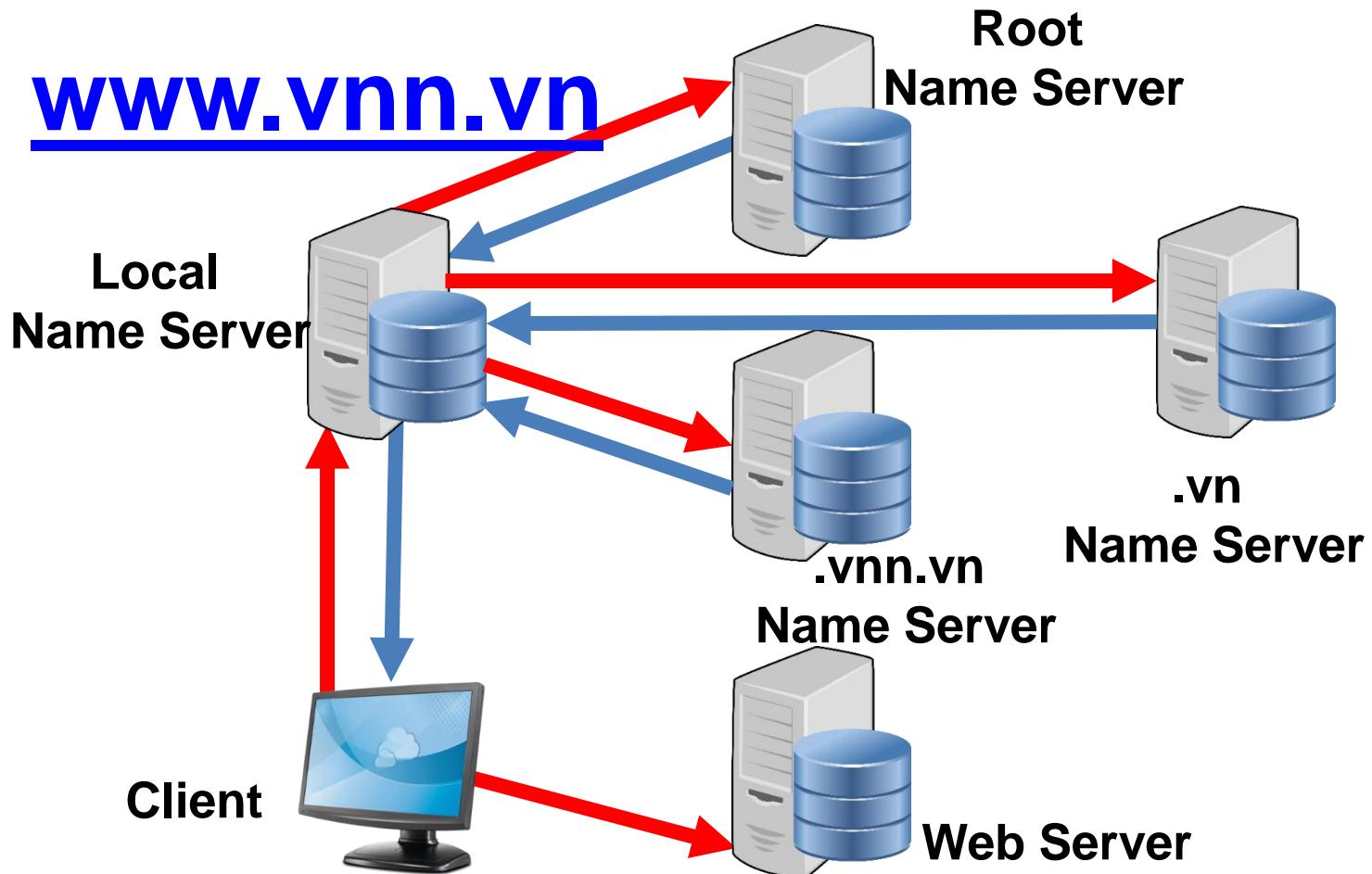
- *Con người*: có nhiều định danh
    - CMT, tên, số hộ chiếu
  - *Các host và router trên Internet*
    - Địa chỉ IP (32 bit, 48 bit) - được dùng để gán địa chỉ cho các gói tin
    - “tên miền”, ví dụ: www.yahoo.com-được con người sử dụng
- Hỏi:** làm cách nào để ánh xạ giữa địa chỉ IP và tên miền, và ngược lại?

# Hệ thống tên miền (DNS)

- Hệ thống tên miền
  - Cơ sở dữ liệu phân tán được cài đặt phân cấp với nhiều server tên miền
    - DNS Server thường là các máy UNIX chạy phần mềm BIND (Berkeley Internet Name Domain)
  - Giao thức tầng ứng dụng: cho phép các host có thể truy vấn đến CSDL tên miền phân tán
    - Giao thức DNS chạy qua UDP và sử dụng port 53
  - DNS thường được sử dụng bởi các giao thức ứng dụng khác (HTTP và SMTP) để dịch hostname do người dùng cung cấp sang IP

# Hệ thống tên miền (DNS)

- Các dịch vụ cung cấp bởi DNS
  - Dịch hostnames sang IP addresses



# Hệ thống tên miền (DNS)

- Các dịch vụ cung cấp bởi DNS
  - Host aliasing
    - Máy chủ có hostname phức tạp có thể có một hoặc nhiều tên bí danh (alias name).
      - Ví dụ hostname **relay1.west-coast.enterprise.com** có thể có 2 bí danh **Enterprise.com** và **www.enterprise.com**.
      - Trong trường hợp này, hostname **relay1.west-coast.enterprise.com** được coi là một hostname chính quy. Hostname bí danh, khi hiển thị, thường dễ nhớ hơn hostname chính quy.
    - Một ứng dụng có thể sử dụng DNS để lấy hostname chính quy cho hostname bí danh được cung cấp cũng như địa chỉ IP của máy chủ.

# Hệ thống tên miền (DNS)

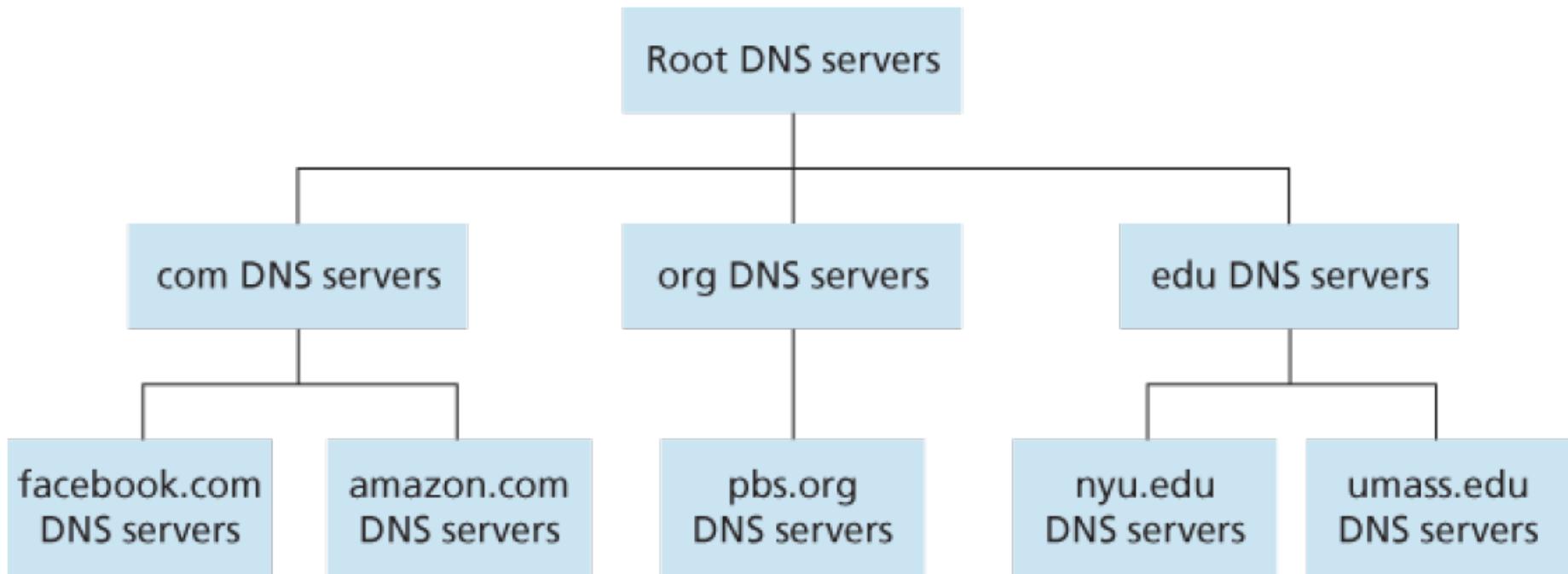
- Các dịch vụ cung cấp bởi DNS
  - Mail server aliasing
    - Ví dụ địa chỉ thư đơn giản, dễ nhớ [bob@yahoo.mail](mailto:bob@yahoo.mail)
      - Tuy nhiên hostname của Yahoo mail server thường phức tạp và khó nhớ hơn là yahoo.com (ví dụ tên chính quy [relay1.west-coast.yahoo.com](http://relay1.west-coast.yahoo.com))
      - Ứng dụng mail có thể sử dụng DNS để lấy hostname chính quy cho một hostname bí danh được cung cấp cũng như địa chỉ IP của host.
        - » Thực tế bản ghi MX cho phép mail server của công ty và Web server có các hostname (bí danh) giống nhau.
          - Ví dụ hostname bí danh của cả Web server và mail server của một công ty là [enterprise.com](http://enterprise.com)

# Hệ thống tên miền (DNS)

- Các dịch vụ cung cấp bởi DNS
  - Phân bổ tải trọng (Load distribution)
    - DNS cũng được dùng để thực hiện phân bổ tải trọng giữa các replicated servers
      - Các site bận như **cnn.com** được sao chép qua nhiều server, mỗi server chạy trên một end system khác nhau và có địa chỉ IP khác nhau.
      - Với các replicated Web server, 1 hostname chính quy ánh xạ tới 1 tập các địa chỉ IP khác nhau => khi client truy vấn DNS một hostname chính quy, server sẽ response toàn bộ tập địa chỉ IP tương ứng với hostname nhưng **xoay vòng thứ tự** của các địa chỉ trong mỗi câu trả lời ⇔ phân bổ lưu lượng giữa các replicated servers

# Hệ thống tên miền (DNS)

- Cấu trúc phân cấp của DNS



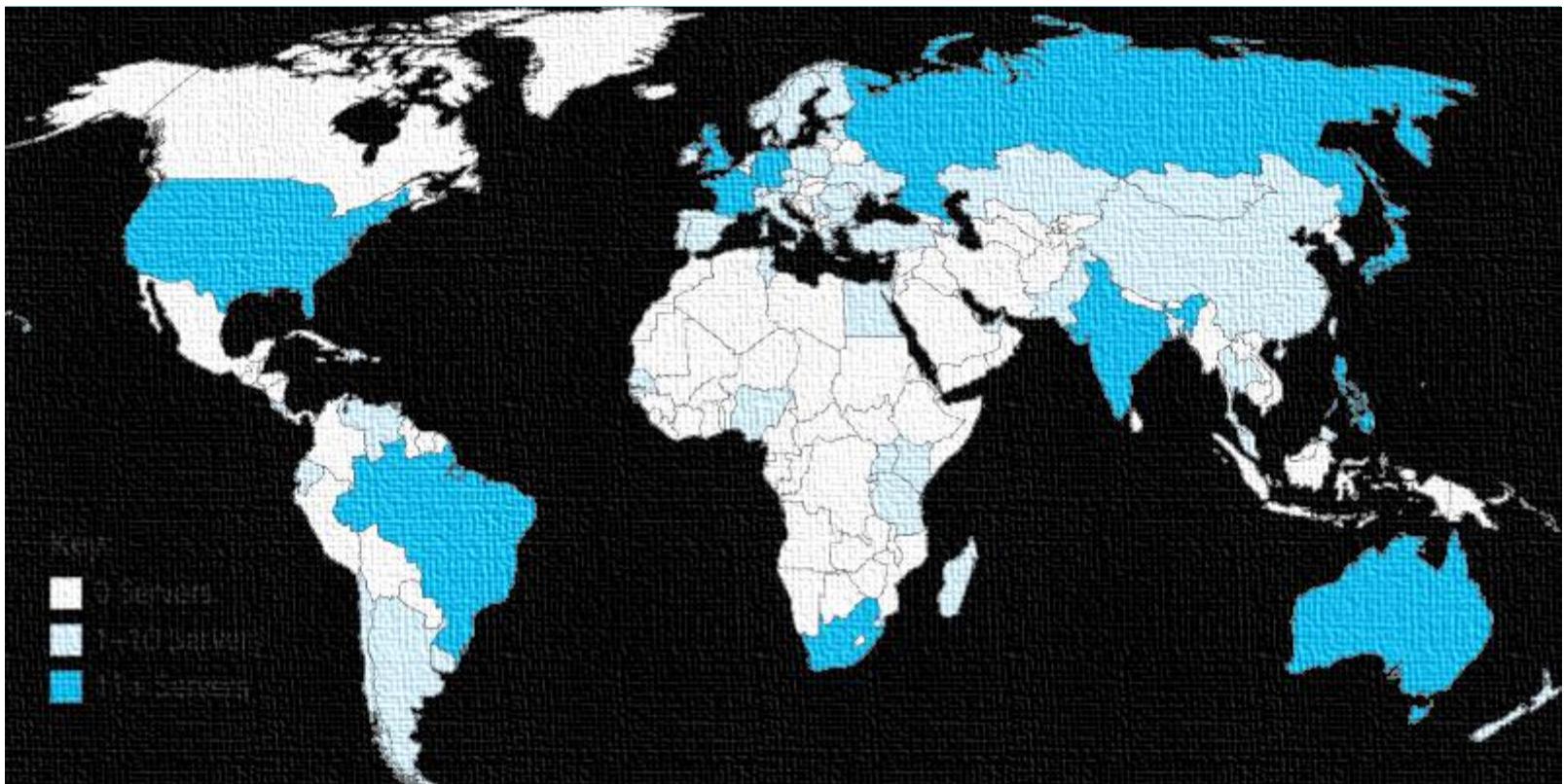
**Root DNS servers**

**Top-level domain (TLD) DNS servers**

**Authoritative DNS servers**

# Hệ thống tên miền (DNS)

- Cấu trúc phân cấp của DNS
  - Root DNS servers



# Hệ thống tên miền (DNS)

- Cấu trúc phân cấp của DNS
  - Top-level domain (TLD) DNS servers
    - Chịu trách nhiệm về:
      - Các tên miền dùng chung như **com, org, net, edu, aero, jobs, museums..**
        - » Công ty Verisign Global Registry Services quản lý các TLD servers for the cho **.com**
        - » Công ty Educause quản lý TLD servers cho **.edu**
      - Các tên miền cấp quốc gia như **uk, fr, ca, jp...**
    - TLD servers cung cấp các địa chỉ IP cho các DNS server có thẩm quyền (authoritative DNS servers).

# Hệ thống tên miền (DNS)

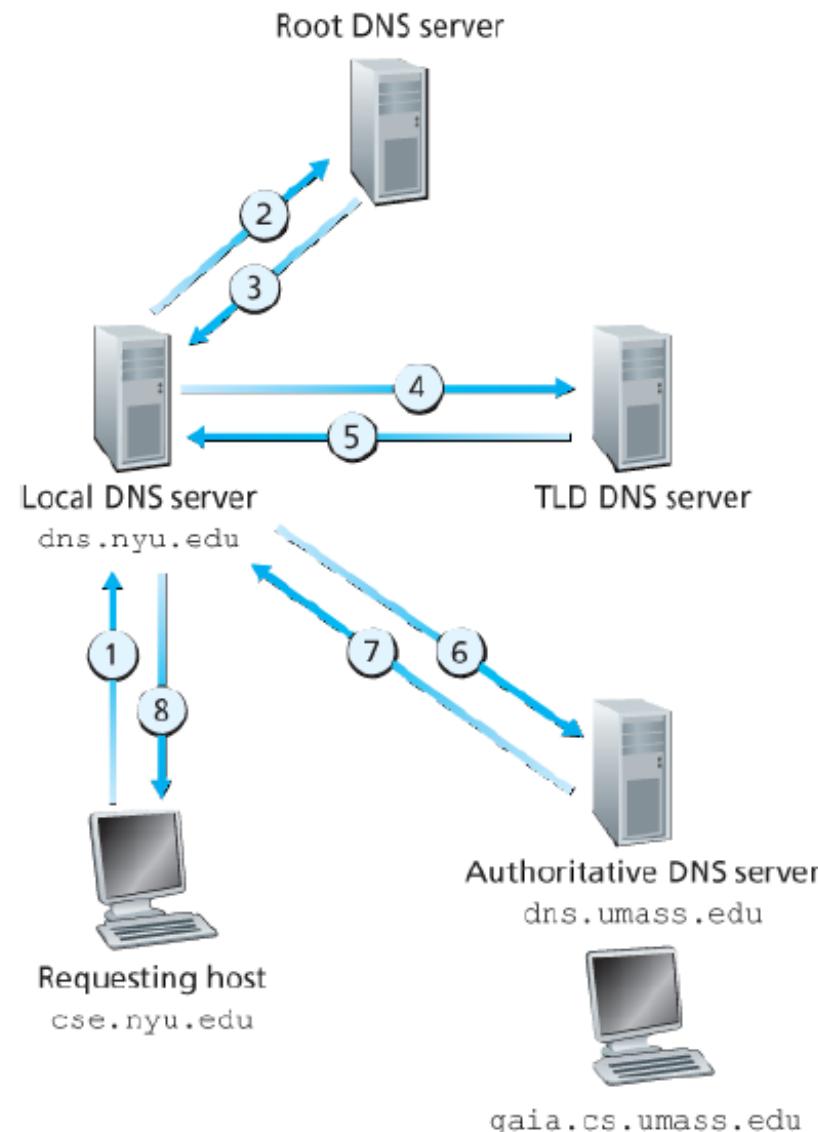
- Cấu trúc phân cấp của DNS
  - Authoritative DNS servers
    - Mỗi tổ chức có máy chủ (Web, mail servers) có thể truy cập công khai trên Internet đều phải cung cấp các bản ghi DNS có thể truy cập công khai để ánh xạ tên hostname – địa chỉ IP.
    - Authoritative DNS server của tổ chức lưu các bản ghi DNS này.
    - Một tổ chức có thể chọn triển khai Authoritative DNS server của riêng mình để lưu DNS records hoặc trả tiền cho việc lưu trữ này tại một Authoritative DNS server của nhà cung cấp dịch vụ.

# Hệ thống tên miền (DNS)

- Máy chủ DNS cục bộ (**local DNS server**)
  - Không hoàn toàn theo cấu trúc phân cấp
  - Mỗi ISP (ISP khu dân cư, công ty, trường học) có một server DNS 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
    - Hoạt động giống như proxy, chuyển tiếp truy vấn vào hệ thống phân cấp

# Hệ thống tên miền (DNS)

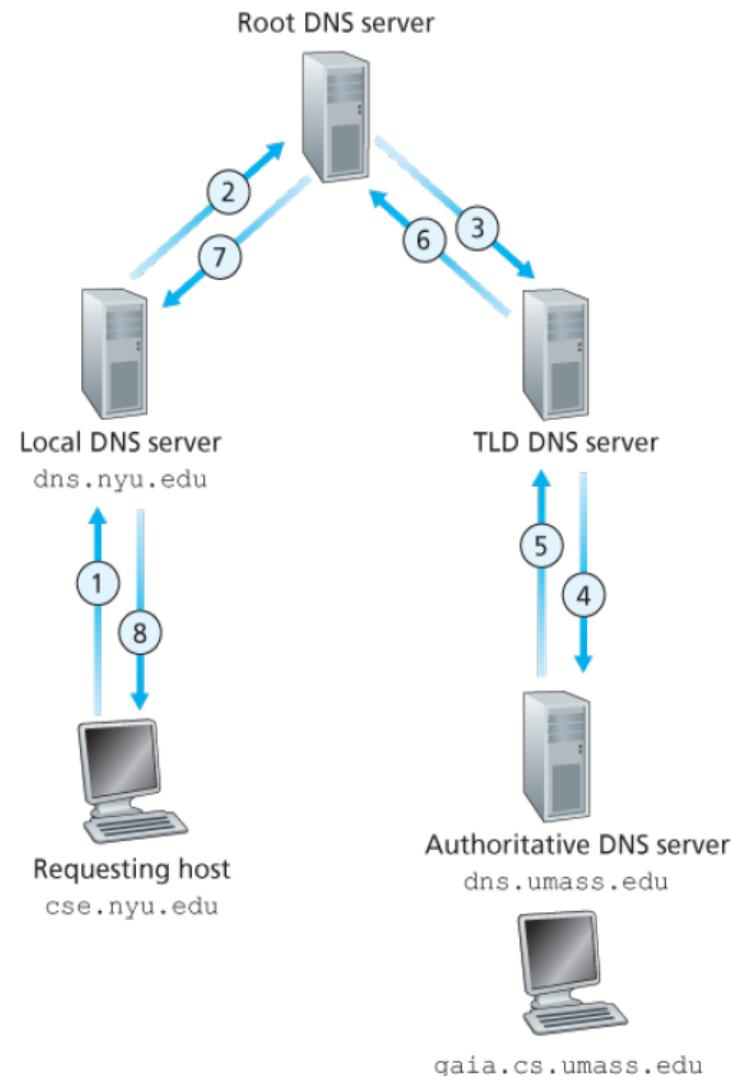
- Ví dụ phân giải tên DNS
  - Host tại **cis.poly.edu** muốn lấy địa chỉ IP của **gaia.cs.umass.edu**
    - Truy vấn lặp
      - Server được hỏi sẽ trả lời với tên của server sẽ có thể hỏi được tiếp
      - “Tôi không biết tên đó, nhưng có thể hỏi server này”



# Hệ thống tên miền (DNS)

- 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



# Hệ thống tên miền (DNS)

- 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
  - Một bản ghi **type A** cung cấp một ánh xạ hostname-IP chuẩn.
  - Ví dụ: (*relay1.bar.foo.com*, 145.37.93.126, A)

# Hệ thống tên miền (DNS)

- 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=NS**: **Name** là tên miền (ví dụ [foo.com](#)), **Value** là tên host của server có thẩm quyền cho tên miền này
  - Ví dụ: ([foo.com](#), [dns.foo.com](#), NS)

# Hệ thống tên miền (DNS)

- 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=CNAME:** **Name** là tên bí danh của tên “chuẩn” (tên thật), **Value** là tên chuẩn
  - Ví dụ: ([foo.com](http://foo.com), [relay1.bar.foo.com](http://relay1.bar.foo.com), CNAME)
- **Type=MX:** **Value** là tên chuẩn của mail server có tên bí danh là **name**
  - Ví dụ: ([foo.com](http://foo.com), [mail.bar.foo.com](http://mail.bar.foo.com), MX)

# Hệ thống tên miền (DNS)

- Bản ghi DNS (DNS records)
  - **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), **Value** là tên chuẩn
  - **Type=MX**: **Value** là tên chuẩn của mail server có tên bí danh là **name**

# Hệ thống tên miền (DNS)

- Giao thức, thông điệp DNS
  - Các thông điệp *truy vấn* và *trả lời* đều có cùng *định dạng thông điệp*

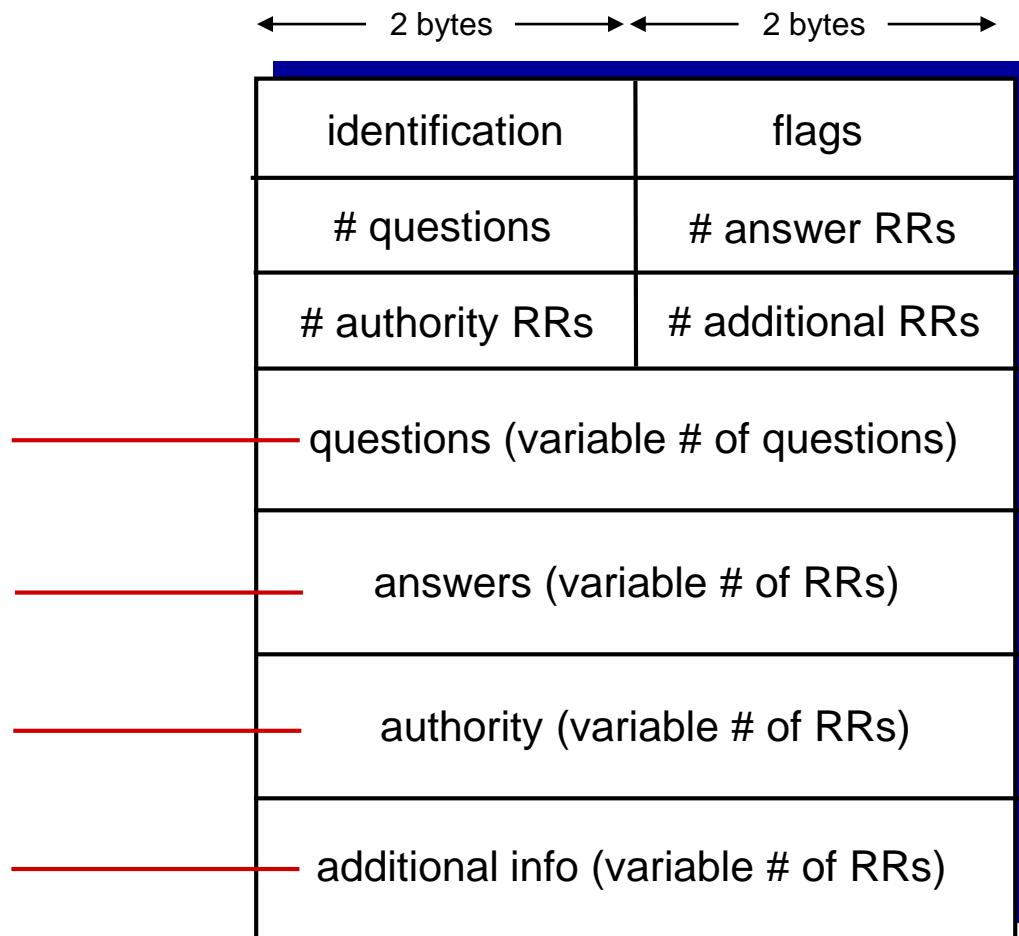
## Tiêu đề thông điệp

- **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

Identification	Flags	
Number of questions	Number of answer RRs	12 bytes
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional “helpful” info that may be used

# Hệ thống tên miền (DNS)

- Giao thức, thông điệp DNS



Trường tên, loại truy vấn

RRs trong trả lời  
truy vấn

Các bản ghi của các server  
có thẩm quyền

Thông tin hữu ích có thể  
được sử dụng

# Hệ thống tên miền (DNS)

- 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 2 bản ghi RR vào trong .com TLD server:
      - (**networkutopia.com, dns1.networkutopia.com, NS**)
      - (**dns1.networkutopia.com, 212.212.22.1, A**)
    - Tạo bản ghi loại A vào trong server có thẩm quyền cho [www.networkutopia.com](http://www.networkutopia.com); bản ghi loại MX cho record cho networkutopia.com

# Hệ thống tên miền (DNS)

- 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

# Hệ thống tên miền (DNS)

- Tấn công DNS
  - Tấn công DDoS
    - Chuẩn 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

# DNS Review

- Dùng DNS để làm gì? Tại sao cần
- DNS server chứa dữ liệu gì?
- Bản ghi nguồn RR của DNS gồm những trường thông tin nào?
- Có mấy kiểu DNS server?
- Cấu trúc phân cấp của tên miền?
- Request/response DNS được thực hiện như thế nào?
- Định dạng thông điệp request và response của DNS có giống nhau không?

# Nội dung

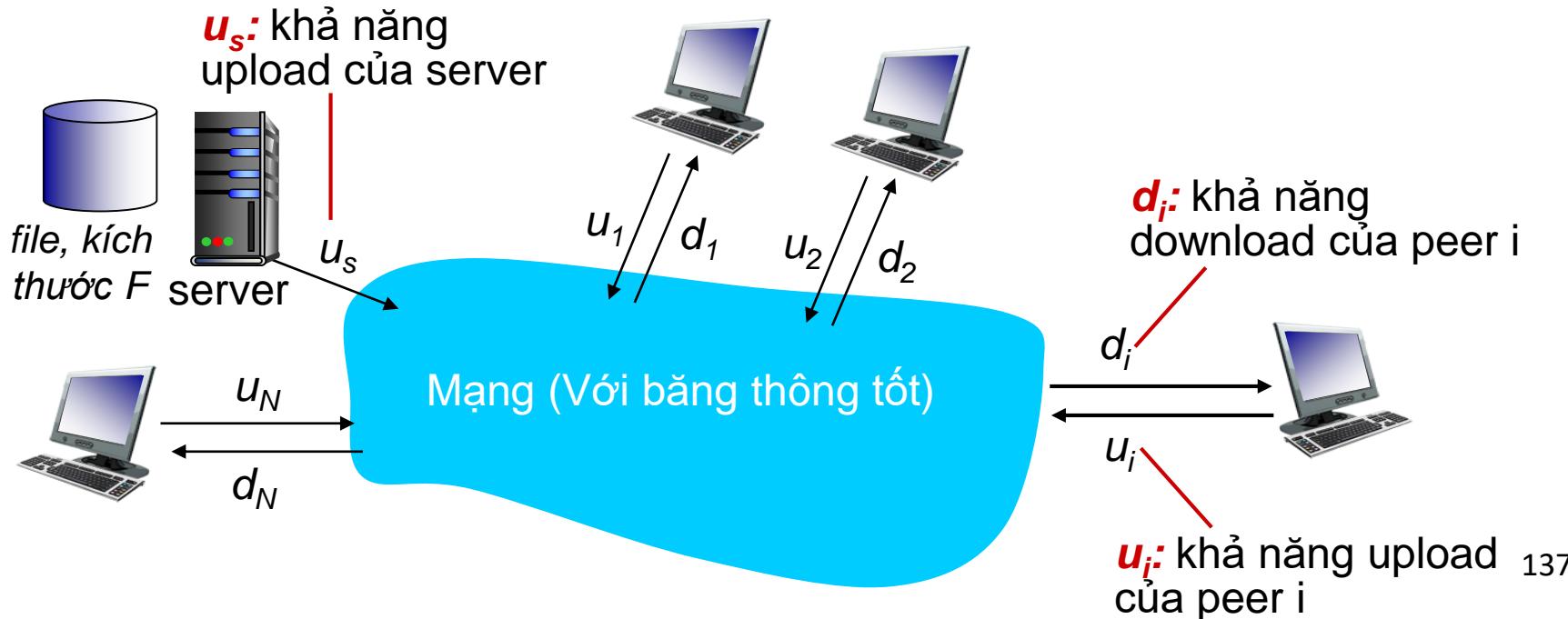
- Các nguyên tắc của các ứng dụng mạng
- Web và HTTP
- FTP
- Thư điện tử trên Internet
- DNS – Internet's Directory Service
- Các ứng dụng P2P (Peer – to- Peer)
- Video Streaming và các mạng phân phối nội dung (Content Distribution Networks)
- Lập trình socket với UDP và TCP

# Kiến trúc mạng P2P thuần túy

- Không cần 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 (Spype)

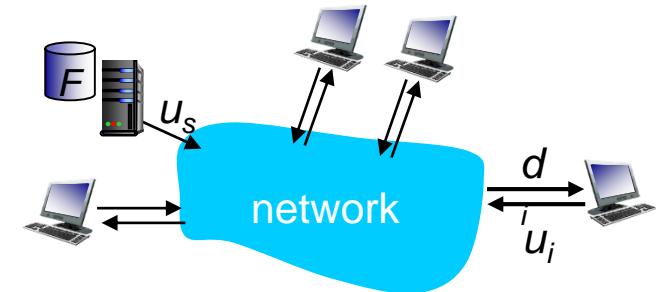
# Chia sẻ file: client-server và P2P

- 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



# Chia sẻ file: client-server và P2P

- 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:  $F/d_{min}$



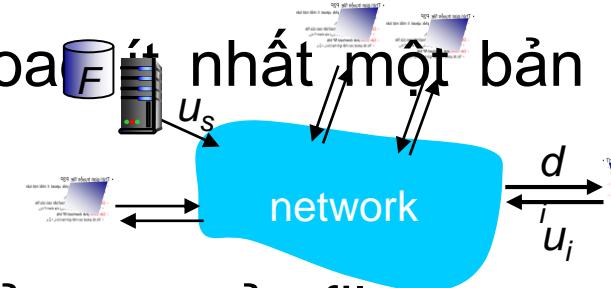
Thời gian để phân phối  $F$   
tới  $N$  client sử dụng  
cách tiếp cận  
client-server

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

Tăng tuyến tính theo  $N$

# Chia sẻ file: client-server và P2P

- Thời gian truyền file: P2P
  - *Việc truyền của server*: phải upload  $F$  nhất  $u_s$  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 upload  $NF$  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

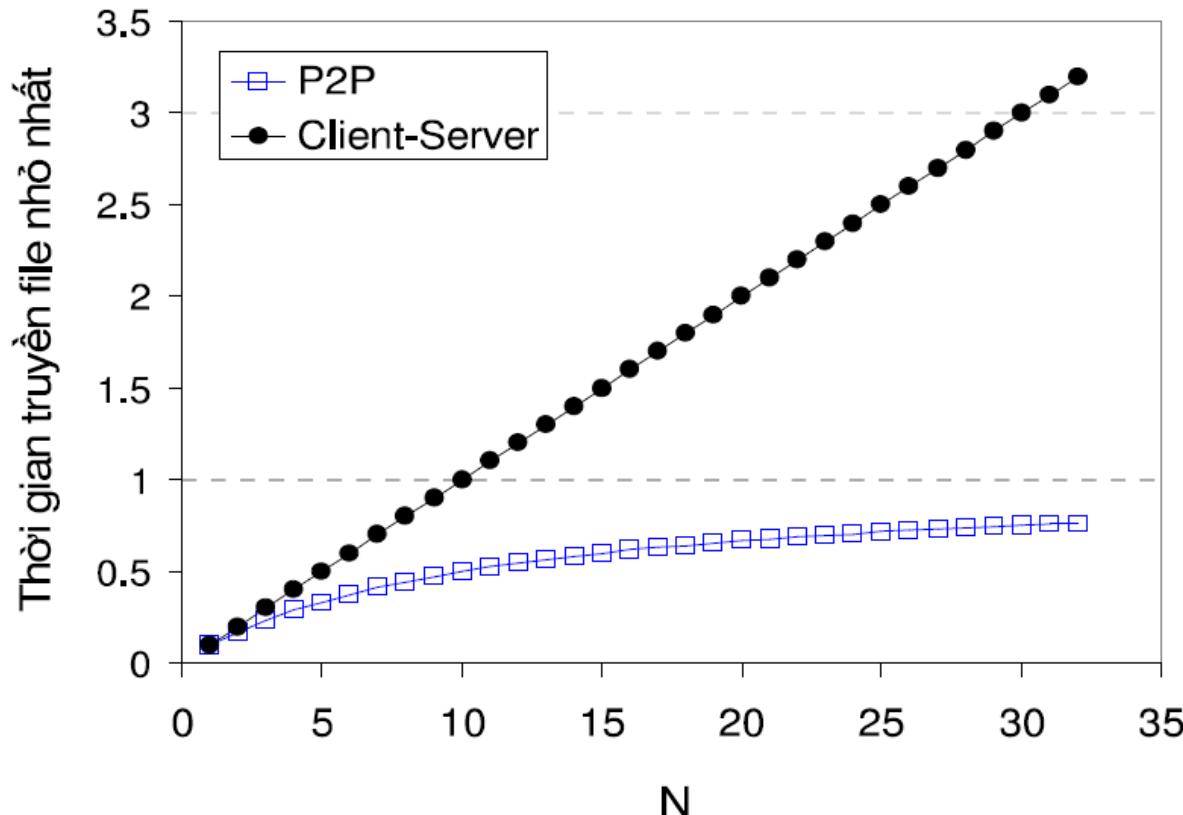
$$D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

Tăng tuyến tính theo  $N \dots$

... nhưng để thực hiện việc này, mỗi peer phải có khả năng phục vụ<sup>139</sup>

# Chia sẻ file: client-server và P2P

- Ví dụ so sánh client-server với P2P
  - Tốc độ upload của client =  $u$ ,  $F/u=1$  giờ,  $u_s = 10u$ ,  $d_{min} \geq u_s$

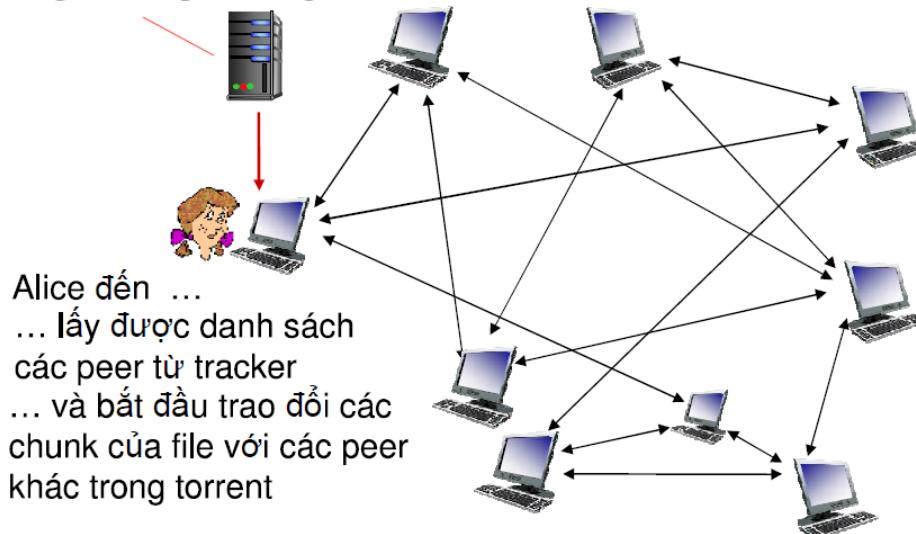


- BitTorrent

- File được chia thành các chunk (phần) có kích thước 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



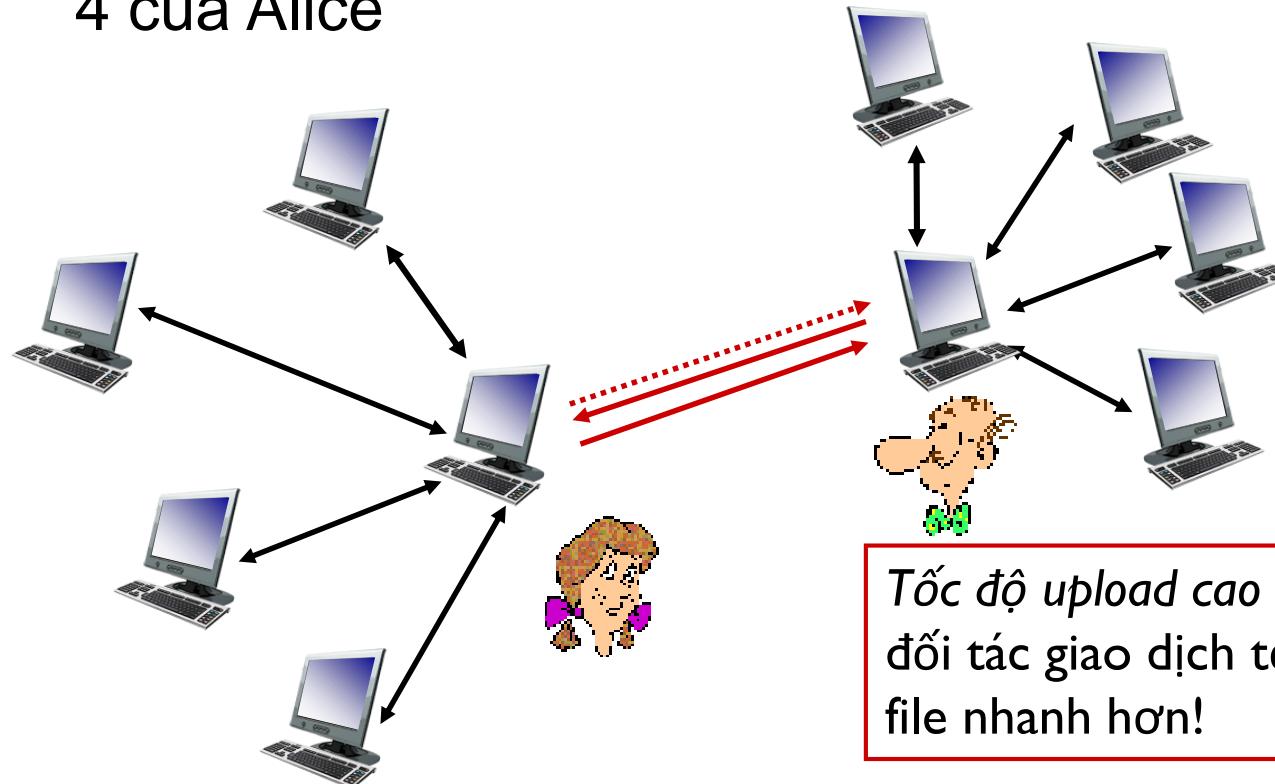
- BitTorrent
  - Peer tham gia vào torrent
    - Không có các chunk, nhưng sẽ tích lũy chúng qua thời gian từ các peer khác
    - Đăng ký với tracker để nhận được danh sách các peer, kết nối với tập con của các peer (“các hàng xóm”)
  - Trong khi download, peer sẽ upload các chunk tới các peer khác
  - Peer có thể thay đổi các peer mà nó sẽ trao đổi các chunk
  - 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 yêu cầu các chunk còn thiếu từ các peer, phần hiếm nhất trước

- BitTorrent: yêu cầu lấy, gửi các chunk của file
  - 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-fo-tat

- Alice “tối ưu hóa không chặn” Bob
- Alice trở thành một nhà cung cấp trong top 4 của Bob
- 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!

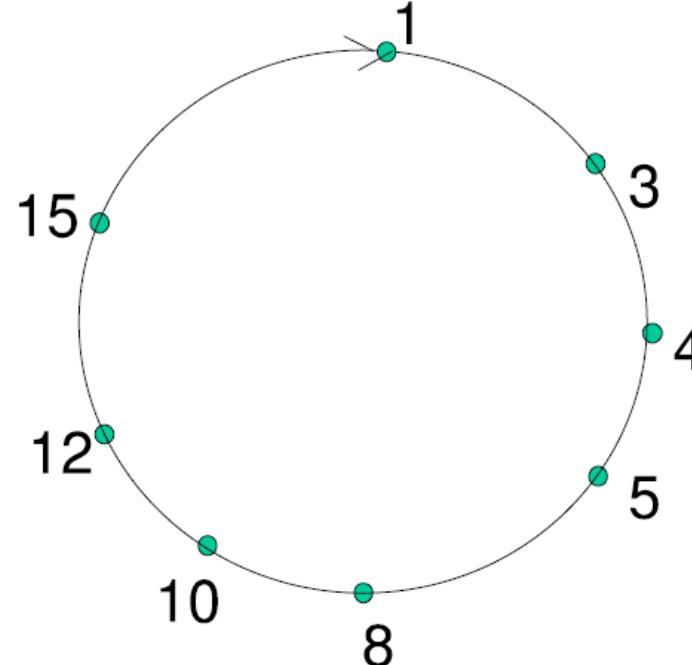
- Bảng băm phân tán-Distributed Hash Table (DHT)
  - DHT: là một *cơ sở dữ liệu P2P phân tán*
  - Cơ sở dữ liệu chứa các cặp (khóa, giá trị) (key, value)
  - Ví dụ
    - Khóa: Số thứ tự; Giá trị: tên người
    - Khóa: tiêu đề bộ phim; Giá trị: địa chỉ IP
  - Phân tán các cặp (key,value) qua (hàng triệu) 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** 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
    - Gán số nguyên cho từng peer
    - Đặt cặp (khóa, giá trị) trong một peer mà nó là **gần nhất** với khóa

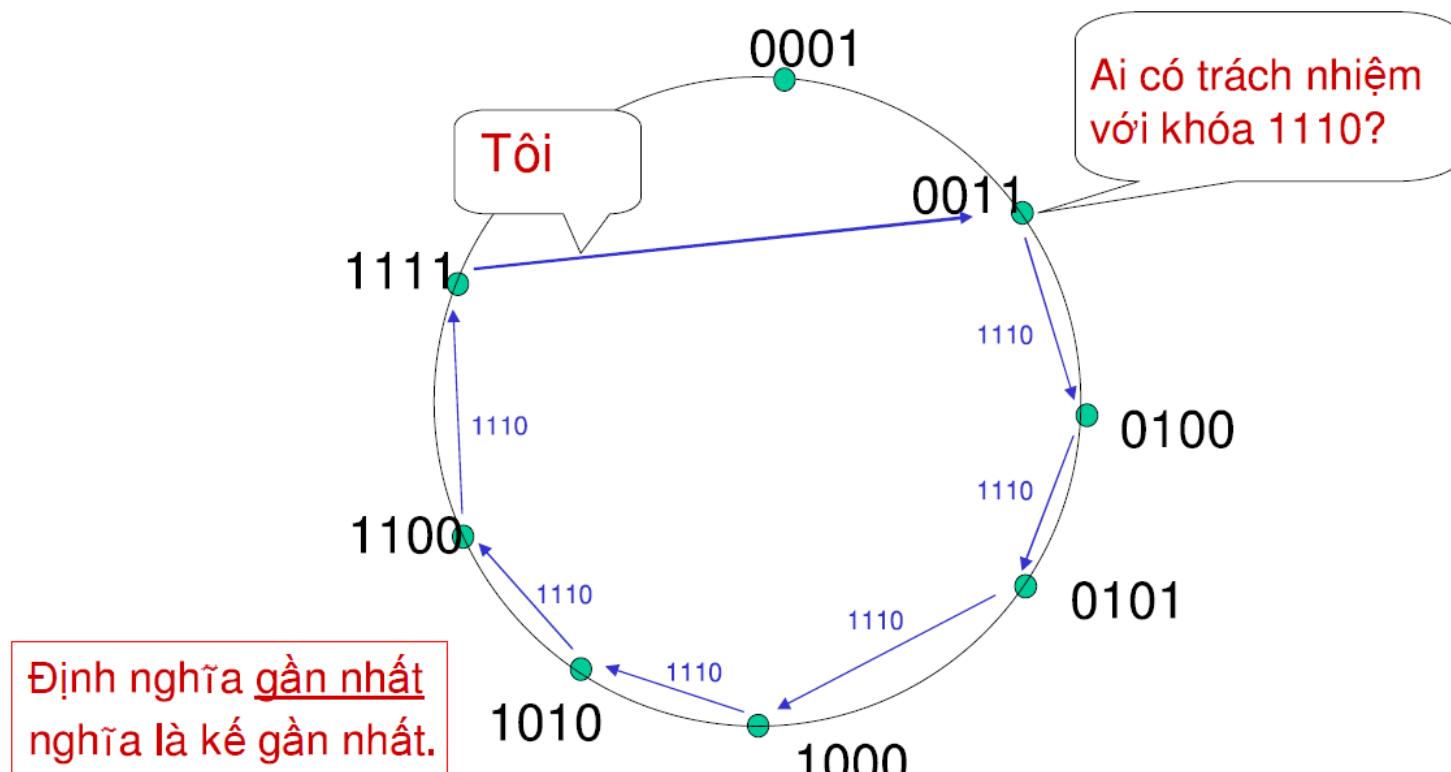
- Định danh DHT
  - Gán định danh số nguyên cho mỗi peer trong dãy  $[0, 2^n - 1]$ 
    - Mỗi định danh được biểu diễn bởi n bit
  - Yêu cầu mỗi khóa là một số nguyên trong cùng dãy
  - Để nhận được khóa số nguyên, cần băm khóa gốc
    - Ví dụ: khóa = **hash** (“Led Zeppelin IV”)
    - Đây là lý do tại 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: 1,3,4,5,6,10,12,14;
    - Khóa = 13, thì peer kế liền với nó là 14
    - Khóa = 15, thì peer kế liền là 1

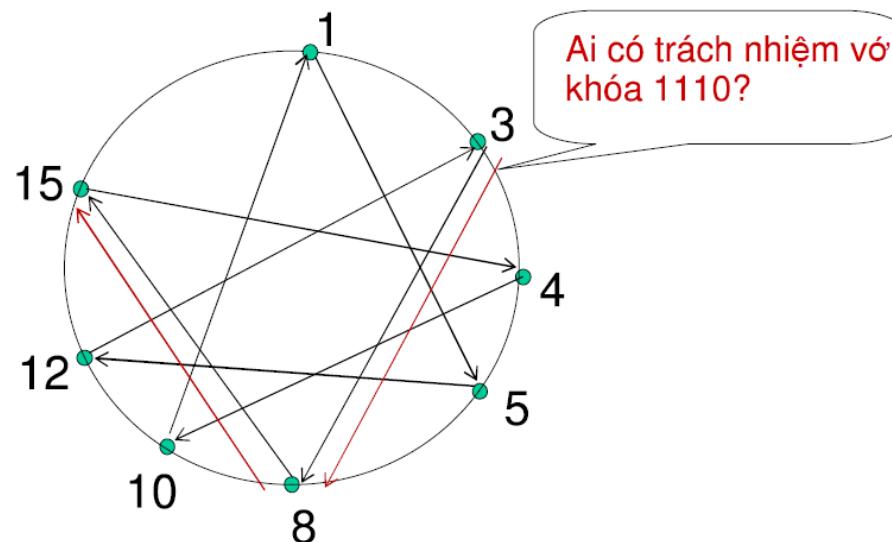
- DHT vòng (1)
  - Mỗi peer *chỉ biết* về peer kế tiếp và peer tiền nhiệm ngay trước nó
  - Mạng che phủ (“overlay network”)



- DHT vòng (1)
  - Trung bình cần  $O(N)$  thông điệp để giải quyết truy vấn, khi có  $N$  peer



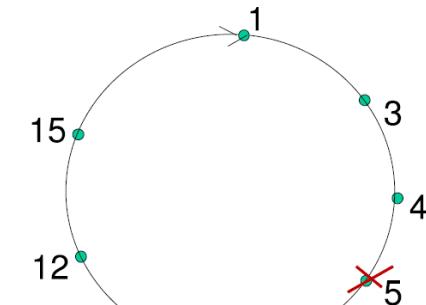
- DHT vòng với peer tắt (shortcut)
  - Mỗi peer giữ theo dõi địa chỉ IP của peer kế tiếp, tiền nhiệm và các peer tắt
  - Sẽ giảm từ 6 xuống còn 2 thông điệp
  - Có thể thiết kế các peer tắt để có  $O(\log N)$  hàng xóm, thì sẽ có  $O(\log N)$  thông điệp trong truy vấn



- Peer churn

- 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
    - Ví dụ: peer 5 đột ngột rời đi
      - 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 (liền kề ngay thứ cấp).



# Nội dung

- Các nguyên tắc của các ứng dụng mạng
- Web và HTTP
- FTP
- Thư điện tử trên Internet
- DNS – Internet's Directory Service
- Các ứng dụng P2P (Peer – to- Peer)
- Video Streaming và các mạng phân phối nội dung (Content Distribution Networks)
- Lập trình socket với UDP và TCP

# Internet video

- Streaming
  - Là công nghệ được sử dụng để truyền dữ liệu tới máy tính và các thiết bị di động thông qua Internet.
  - Streaming sẽ truyền dữ liệu - thường là audio và video- dưới dạng một “dòng chảy” liên tục, cho phép người nhận xem/nghe gần như ngay lập tức.
- Tải dần dần (Progressive Download) so với streaming
  - Khi nào người nhận có thể bắt đầu sử dụng nội dung tải về
  - Chuyện gì xảy ra sau khi người nhận dùng xong

# Internet video

- Streaming video (video streaming)
  - Streaming video (luồng video) thực chất là quá trình truyền các frame của file video tới người nhận.
    - Video là một chuỗi các ảnh (image), được hiển thị ở một tốc độ nhất định, ví dụ 24 hay 30 images /giây.
      - Ảnh số là một mảng các pixel. Mỗi pixel được mã hóa bởi một số bít nhất định để biểu diễn màu sắc và cường độ sáng
    - Đặc điểm quan trọng của video đó là nó có thể nén  
↔ đánh đổi giữa chất lượng video và **bitrate**
      - Video nén với bitrate từ 100 kbps (video chất lượng thấp) tới hơn 3 Mbps (phát trực tuyến phim độ nét cao)

- Streaming video (video streaming)
  - Cùng một video, có thể nén với nhiều bitrate khác nhau và mỗi cái có chất lượng khác nhau.
    - Ví dụ một video được nén với bitrate 300 kbps, 1 Mbps, và 3 Mbps.
    - Người dùng có thể chọn tốc độ nào mình muốn xem dựa trên băng thông hiện có.
      - Chọn tốc độ 3 Mbps nếu dùng kết nối Internet tốc độ cao
      - Chọn tốc độ 300 kbps nếu xem video bằng smartphone kết nối 3G

# HTTP streaming and DASH

- HTTP streaming
  - Video được lưu tại HTTP server như một file thông thường với URL
    - Người dùng muốn xem video -> client thiết lập một kết nối TCP tới server và gửi HTTP GET request cho URL này.
    - Server gửi file video trong thông điệp HTTP response
    - Phía client nhận bytes và lưu vào buffer ứng dụng client. Khi số bytes vượt quá ngưỡng định trước, ứng dụng client bắt đầu phát lại.

# HTTP streaming and DASH

- DASH (Dynamic Adaptive Streaming over HTTP)
  - HTTP Streaming: tất cả client đều nhận được **video mã hóa với bitrate như nhau**, mặc dù lượng **băng thông** sẵn có **cho các client không giống nhau**  
⇒ DASH:
    - **Video được mã hóa với bitrate khác nhau** tương ứng với mức chất lượng khác nhau.
    - Client đưa ra yêu cầu động cho các đoạn video dài vài giây.
      - Khi băng thông cao, client lựa chọn các đoạn video có bitrate cao, ngược lại nó chọn các đoạn video có bitrate thấp
    - Cho phép client thích ứng với băng thông hiện có nếu như băng thông end-to-end hiện có thay đổi trong phiên truyền

# HTTP streaming and DASH

- **DASH**

- Các phiên bản video được lưu trong HTTP server với các URL khác nhau.
- HTTP server có một tệp kê khai, cung cấp URL cho từng phiên bản cùng với bitrate của nó
- Client trước tiên yêu cầu tệp kê khai và học các phiên bản khác nhau này. Sau đó nó lựa chọn một chunk ở mỗi thời điểm bằng cách chỉ ra URL và một dải byte trong một thông điệp request HTTP GET.
- Trong khi tải chunks, client cũng đo băng thông nhận được và chạy thuật toán xác định rate để lựa chọn chunk cho request tiếp theo.
- Đương nhiên, nếu client có nhiều video được buffered và nếu như băng thông đo được là cao, nó sẽ chọn chunk có bitrate cao. Ngược lại nó sẽ chọn chunk có bitrate thấp

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Quản lý nhiều server ở nhiều vị trí địa lý khác nhau.
  - Các server lưu các bản sao của Web content (video, document, image, audio) và hướng từng yêu cầu của người dùng tới một vị trí CDN có khả năng cung cấp sự trải nghiệm tốt nhất cho người dùng.

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Private CDN
    - CDN riêng, được sở hữu bởi chính nhà cung cấp nội dung;
    - ví dụ: CDN của Google phân phối video YouTube và các loại nội dung khác.
  - Third-party CDN
    - CDN bên thứ ba, phân phối nội dung thay mặt cho nhiều nhà cung cấp nội dung;
    - Ví dụ: Akamai, Limelight và Level-3.

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - CDN thường áp dụng một trong hai triết lý đặt máy chủ
    - Enter Deep
      - Thâm nhập sâu vào các mạng truy cập của ISP bằng cách triển khai các cụm máy chủ trong các ISP truy cập trên khắp thế giới
        - » Ví dụ: Akamai tiên phong sử dụng triết lý này với các cụm máy chủ được đặt ở khoảng 1.700 vị trí với mục tiêu tiếp cận gần hơn với người dùng cuối, từ đó đó cải thiện độ trễ và thông lượng bằng cách giảm số lượng liên kết và bộ định tuyến giữa người dùng cuối và máy chủ CDN
      - Do thiết kế phân tán cao nên việc duy trì và quản lý các cụm là một thách thức.

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - CDN thường áp dụng một trong hai triết lý đặt máy chủ
    - Bring Home
      - Được nhiều công ty CDN áp dụng (ví dụ Limelight)
      - Đưa ISP về nhà bằng cách xây dựng các cụm lớn với số lượng sites ít hơn (ví dụ: vài chục).
      - Thay vì vào bên trong các ISP truy cập, các CDN này thường đặt các cụm máy chủ của chúng trong các điểm trao đổi Internet (Internet Exchange Points-IXP).
      - So với Enter Deep, Bring Home có chi phí quản lý và bảo trì thấp hơn, nhưng có thể có độ trễ cao hơn và thông lượng thấp hơn cho người dùng cuối.

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Khi các cụm server của CDN đã có sẵn, CDN sao chép nội dung trên các cụm của nó.
  - CDN có thể không muốn đặt một bản sao của mọi video trong mỗi cụm server, vì một số video hiếm khi được xem hoặc chỉ phổ biến ở một số quốc gia.

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Trên thực tế, nhiều CDN không đẩy video vào cụm của họ mà thay vào đó sử dụng chiến lược kéo đơn giản
    - Nếu khách hàng yêu cầu video từ một cụm không lưu trữ video, thì cụm đó sẽ truy xuất video (từ kho lưu trữ trung tâm hoặc từ cụm khác) và lưu trữ cục bộ một bản sao trong khi streaming video tới client ở cùng một thời điểm.
    - Tương tự như Web cache, khi bộ nhớ của một cụm bị đầy, nó sẽ xóa các video không được sử dụng thường xuyên.

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Hoạt động của CDN
    - Khi một trình duyệt trong host của người dùng được hướng dẫn truy xuất một video cụ thể (được xác định bằng URL), CDN phải chặn yêu cầu này để nó có thể (1) **xác định cụm máy chủ CDN** phù hợp cho máy khách tại thời điểm đó và (2) **chuyển hướng yêu cầu** của client tới một server trong cụm đó.
      - Hầu hết CDN sử dụng DNS để chặn và chuyển hướng các request

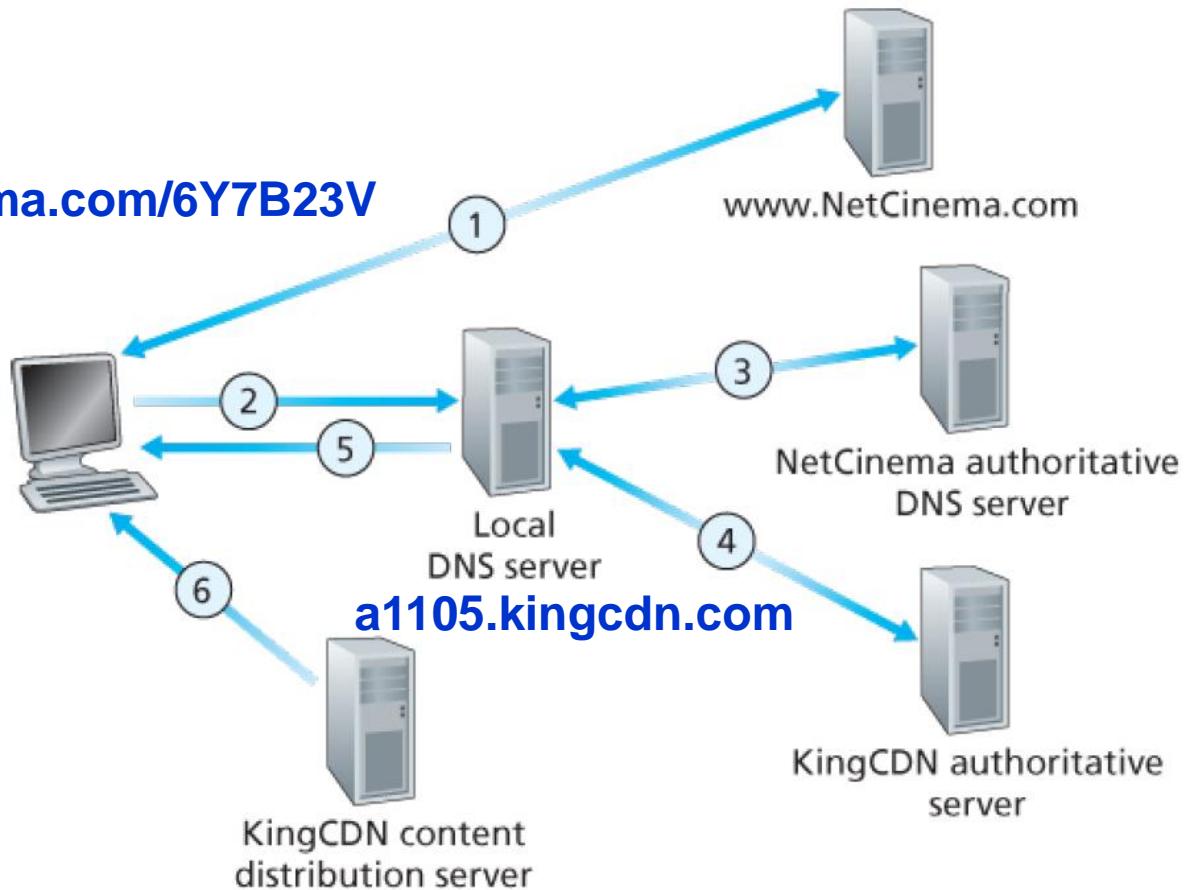
# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Chặn và chuyển hướng các request dùng DNS
    - Ví dụ nhà cung cấp nội dung NetCinema sử dụng CDN bên thứ ba là KingCDN để phân phối video của mình cho các khách hàng của mình
      - Trên các trang Web NetCinema, mỗi video của nó được gán một URL bao gồm chuỗi “video” và số định danh duy nhất cho chính video đó. ví dụ, **Transformers7** có thể được gán <http://video.netcinema.com/6Y7B23V>

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Chặn và chuyển hướng các request dùng DNS

<http://video.netcinema.com/6Y7B23V>



# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Chiến lược lựa chọn cụm máy chủ
    - Cơ chế để **chuyển hướng động** các client đến một cụm máy chủ hoặc một trung tâm dữ liệu trong CDN.
    - Phương pháp sử dụng
      - Gán client cho cụm gần nó nhất về mặt địa lý
      - Dựa trên các điều kiện lưu lượng hiện tại

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Chiến lược lựa chọn cụm máy chủ
    - Gán client cho cụm gần nó nhất về mặt địa lý
      - Sử dụng cơ sở dữ liệu vị trí địa lý thương mại (Quova, Max-Mind). Mỗi địa chỉ IP DNS cục bộ được ánh xạ tới một vị trí địa lý. Khi một yêu cầu DNS nhận được từ LDNS cụ thể, CDN chọn cụm gần nhất về mặt địa lý, tức là cụm cách LDNS ít km nhất “theo đường chim bay”.
      - Giải pháp này có thể hoạt động khá tốt cho một lượng lớn client. Tuy nhiên với một số client, nó lại không tốt vì cụm gần nhất về mặt địa lý có thể không phải là cụm gần nhất về độ dài hay số chặng (hop) của đường mạng.

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Chiến lược lựa chọn cụm máy chủ
    - Gán client cho cụm gần nó nhất về mặt địa lý
      - Một vấn đề cổ hữu với tất cả các phương pháp tiếp cận dựa trên DNS là một số người dùng cuối được cấu hình để sử dụng các LDNS được định vị từ xa, trong trường hợp đó, vị trí LDNS có thể xa vị trí của client.
      - Hơn nữa, chiến lược đơn giản này bỏ qua sự thay đổi về độ trễ và băng thông khả dụng theo thời gian của các đường truyền Internet, luôn gán cùng một cụm cho một client cụ thể.

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Chiến lược lựa chọn cụm máy chủ
    - Dựa trên các điều kiện lưu lượng hiện tại
      - Để xác định cụm tốt nhất cho client dựa trên điều kiện lưu lượng hiện tại, CDN có thể thực hiện các phép đo định kỳ theo thời gian thực về độ trễ và hiệu suất mát mát giữa các cụm và client.
        - » Ví dụ: Mỗi cụm trong một CDN định kỳ gửi các thăm dò (ví dụ: thông điệp ping hoặc truy vấn DNS) đến tất cả LDNS trên khắp thế giới.
      - Nhược điểm của cách tiếp cận là nhiều LDNS được cấu hình để không phản hồi với thăm dò như vậy.

# Các mạng phân phối nội dung

- CDN (Content Distribution Networks)
  - Netflix
  - YouTube
  - Kankan

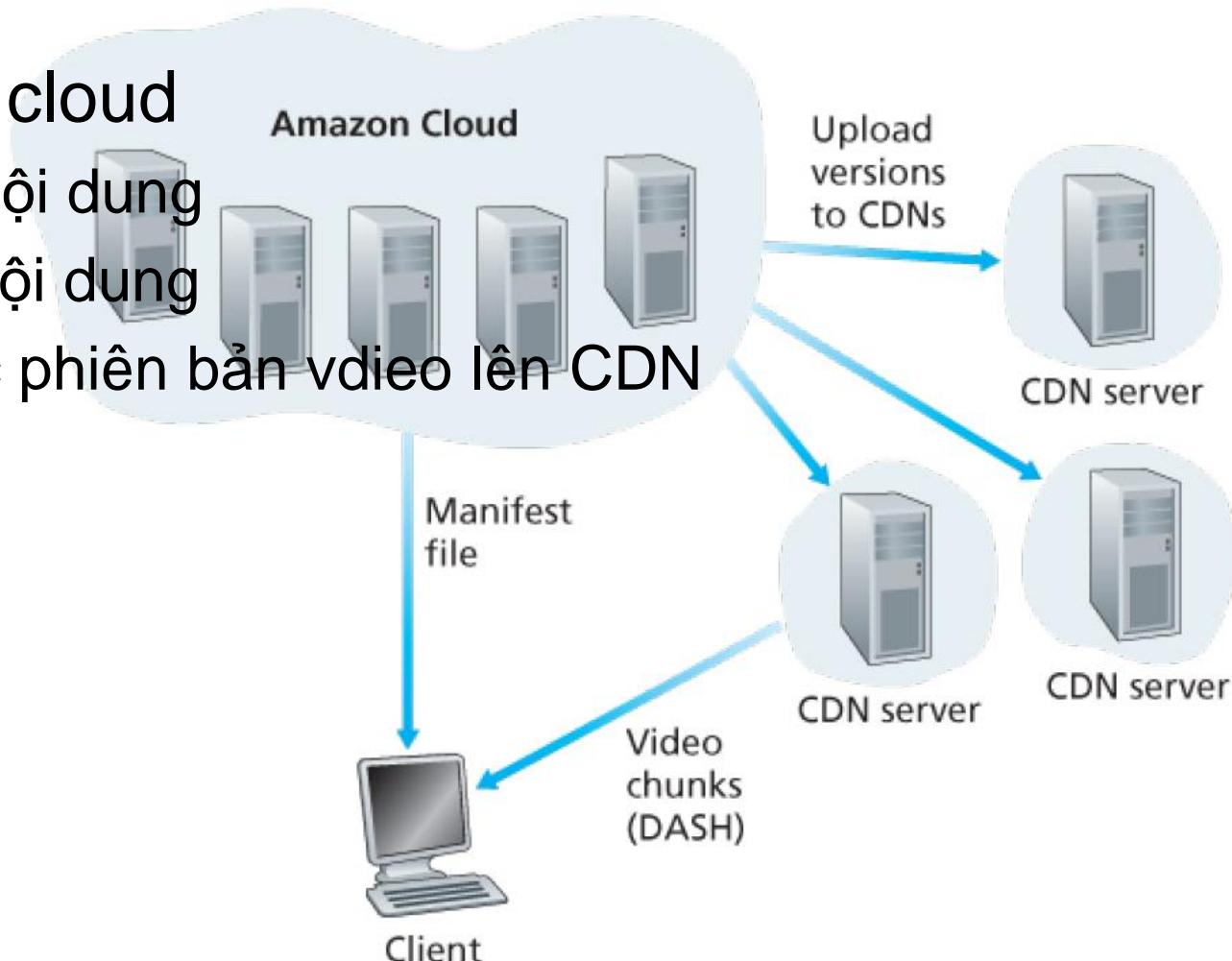
# Các mạng phân phối nội dung

- Netflix
  - Tạo ra 37% lưu lượng download trong các ISP khu dân cư ở Bắc Mỹ vào năm 2015, Netflix đã trở thành nhà cung cấp dịch vụ hàng đầu cho phim truyền hình và phim trực tuyến tại Mỹ.
  - Hệ thống phân phối video Netflix có hai thành phần chính
    - Đám mây Amazon
    - Cơ sở hạ tầng CDN riêng của Netflix

# Các mạng phân phối nội dung

- Netflix
  - Amazon cloud

- Nhập nội dung
- Xử lý nội dung
- Tải các phiên bản video lên CDN



# Các mạng phân phối nội dung

- Netflix
  - Tương tác client-server
    - Các trang web để duyệt thư viện video của Netflix đặt trong các máy chủ của Amazon cloud.
    - Khi người dùng lựa chọn một bộ phim, phần mềm Netflix chạy trong Amazon cloud trước tiên xác định CDN server nào có bản sao bộ phim này. Trong số các server có phim yêu cầu, phần mềm sẽ chọn server “tốt nhất” cho yêu cầu đó.
      - Nếu client đang sử dụng ISP khu dân cư mà có server CDN Netflix được cài đặt trong ISP đó, và server này có bản sao bộ phim yêu cầu thì server này sẽ được chọn. Nếu không một server ở IXP liền kề sẽ thường được chọn.

# Các mạng phân phối nội dung

- Netflix
  - Tương tác client-server
    - Khi Netflix lựa chọn xong CDN server nó gửi cho client địa chỉ IP của server cùng với file liệt kê có chứa các URL cho các version khác nhau của bộ phim yêu cầu.
    - Sau đó client và server CDN tương tác trực tiếp với nhau sử dụng DASH
      - Client gửi thông điệp request HTTP GET để request các chunks từ các version khác nhau của movie (chunk của Netflix có độ dài xấp xỉ 4 giây).
      - Trong khi các chunks đang được tải về, client đo thông lượng nhận được và chạy giải thuật xác định bitrate để chọn chất lượng chunk yêu cầu tiếp theo.

# Các mạng phân phối nội dung

- Netflix
  - Tương tác client-server
    - **Không sử dụng chuyển hướng DNS** khi kết nối một client cụ thể tới một server CDN, thay vào đó phần mềm Netflix (chạy trên Amazon cloud) trực tiếp hướng dẫn client sử dụng một server CDN cụ thể.
    - **Sử dụng push caching** không dùng pull caching
      - Nội dung được đẩy lên các server theo lịch trình và ở những giờ không cao điểm

# Các mạng phân phối nội dung

- YouTube
  - Là site chia sẻ video lớn nhất thế giới
    - 300 giờ video được tải lên mỗi phút
    - Hàng tỷ video được xem trong mỗi ngày
  - Youtube bắt đầu tháng 4 năm 2005 và được Google mua lại vào tháng 11 năm 2006.
  - Giống Netflix, google sử dụng private CDN để phân phối videos và đã cài đặt các cụm máy chủ ở hàng trăm vị trí IXP và ISP khác nhau.
    - Từ những địa điểm này và trực tiếp từ các trung tâm dữ liệu khổng lồ của mình, Google phân phối các video trên YouTube.

# Các mạng phân phối nội dung

- YouTube
  - Sử dụng pull caching và chuyển hướng DNS
  - Chiến lược lựa chọn các cụm máy chủ
    - Hướng client tới cụm mà RTT giữa client và server là nhỏ nhất
    - Để cân bằng tải giữa các cụm máy chủ, đôi khi client được điều hướng (qua DNS) tới một cụm xa hơn.
  - Sử dụng HTTP streaming; không sử dụng DASH mà yêu cầu người dùng chọn thủ công một version

# Các mạng phân phối nội dung

- Kankan
  - Triển khai CDN theo kiến trúc client-server như Netflix và Google là tốn kém
    - Với các máy chủ chuyên dụng, được vận hành bởi CDN riêng của mình, cả Netflix và Google đều phải trả tiền cho cả hạ tầng phần cứng máy chủ và băng thông máy chủ sử dụng để phân phối video.
  - Kankan sử dụng kiến trúc P2P
    - P2P video streaming rất giống với BitTorrent file downloading.

# Các mạng phân phối nội dung

- Kankan
  - Kiến trúc P2P
    - Khi một peer muốn xem một video, nó liên lạc với một tracker để xem các peer khác trong hệ thống có bản copy của video này không.
    - Peer yêu cầu khi đó sẽ gửi yêu cầu về chunks của video song song tới các peer khác có video này.
    - Tuy nhiên, khác BitTorrent download, các yêu cầu được ưu tiên thực hiện cho các chunks sẽ được phát lại trong tương lai gần để đảm bảo phát lại liên tục

# Các mạng phân phối nội dung

- Kankan
  - Gần đây Kankan chuyển hướng sử dụng hệ thống streaming kết hợp CDN-P2P
    - Trong hầu hết các trường hợp, client request phần đầu của nội dung từ máy chủ CDN và song song request nội dung từ các peer.
    - Khi tổng lưu lượng P2P đủ để phát lại video, client sẽ ngừng streaming từ CDN và chỉ stream từ các peers. Nhưng nếu lưu lượng P2P không đủ, client sẽ khởi động các kết nối CDN và quay lại chế độ streaming kếp hợp CDN-P2P
  - => Giảm trễ khởi động và giảm thiểu tối đa việc dựa vào cơ sở hạ tầng server và băng thông

# Nội dung

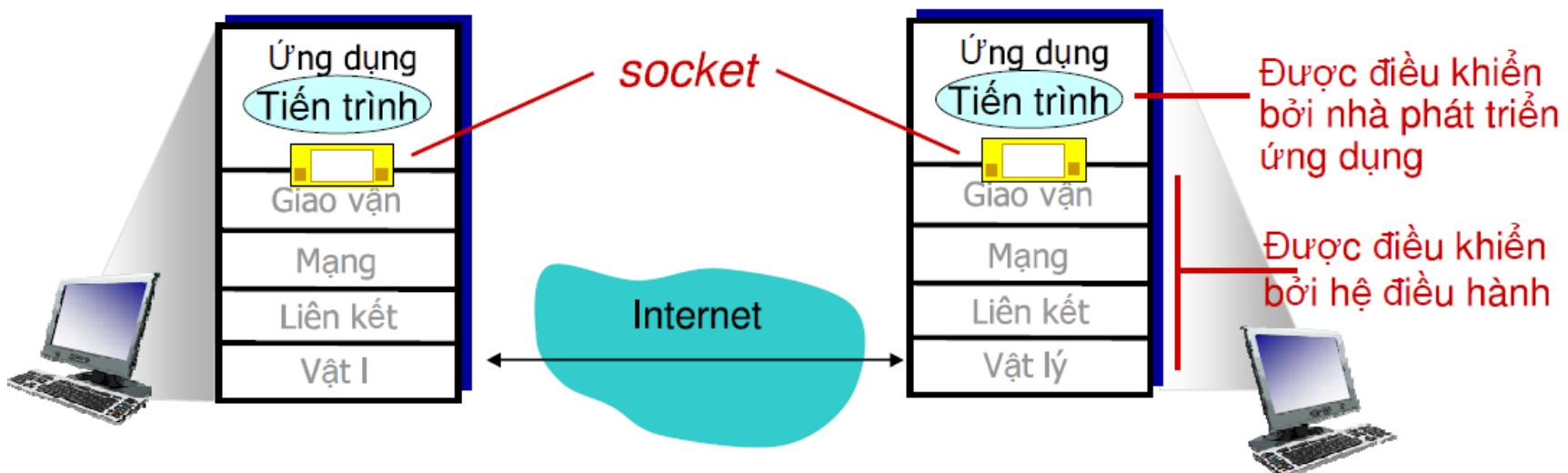
- Nguyên lý của ứng dụng mạng
  - Kiến trúc của ứng dụng
  - Các yêu cầu của ứng dụng
- Web và HTTP
- FTP
- Thư điện tử
  - SMTP, POP3, IMAP
- DNS
- Ứng dụng P2P
- Lập trình socket với UDP và TCP

# Lập trình Socket

- Các **network application programs** được tạo ra như thế nào?
    - Một ứng dụng mạng cơ bản gồm một cặp (**chương trình client, chương trình server**). 2 chương trình này nằm ở hai hệ thống cuối khác nhau.
    - Khi các chương trình này được thực thi, một **tiến trình client** và một **tiến trình server** được tạo ra và các tiến trình này giao tiếp với nhau bằng cách đọc từ hoặc ghi tới **socket**
- => Khi tạo một ứng dụng mạng, nhiệm vụ chính của người phát triển ứng dụng là phải viết code cho cả các chương trình client và server.

# 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

- 2 kiểu network applications
  - Ứng dụng mở (Open application)
    - Nguyên tắc hoạt động được chỉ rõ trong các tài liệu RFC hoặc các tài liệu chuẩn khác (được công bố công khai)
    - Khi viết các chương trình client và server phải tuân thủ theo các nguyên tắc được quy định trong các tài liệu này
    - Ví dụ:
      - Google Chrome browser giao tiếp với một Apache Web server
      - Một BitTorrent client giao tiếp với BitTorrent tracker.

# Lập trình Socket

- 2 kiểu network applications
  - Ứng dụng độc quyền (proprietary network application)
    - Các chương trình client và server sử dụng giao thức tầng ứng dụng không được công bố công khai trong RFC hay các tài liệu khác.

# Lập trình Socket

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

# Lập trình Socket

- 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

# Lập trình Socket

- Tương tác client/server socket: UDP

## server (chạy trên serverIP)

Tạo socket, port= x:

```
serverSocket =  
socket(AF_INET,SOCK_DGRAM)
```

Đọc datagram từ  
serverSocket

Ghi trả lời vào  
serverSocket  
địa chỉ client,  
số hiệu cổng  
cụ thể

## client

Tạo socket:

```
clientSocket =  
socket(AF_INET,SOCK_DGRAM)
```

Tạo datagram với IP của server  
và port=x; gửi datagram qua  
clientSocket

Đọc datagram từ  
clientSocket

Đóng  
clientSocket

# Lập trình Socket

- Tương tác client/server socket: UDP
  - Client program: UDPClient.py
  - Server program: UDPServer.py
- Test cặp chương trình client-server
  - Chạy UDPClient.py trên một host, chạy UDPServer.py trên một host khác
  - Thực thi UDPServer.py trong server host
    - Tạo ra một process trong server chạy không tải cho đến khi nó được một client nào đó liên hệ.
  - Sau đó, thực thi UDPClient.py trong client host
    - Tạo ra một process trong máy khách

# Lập trình Socket

- Client program: UDPClient.py

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

- Server program: UDPServer.py

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("", serverPort))
print("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

# Lập trình Socket

- Tương tác client/server socket: UDP
  - Kịch bản 1
    - Client gửi một chuỗi ký tự sang cho server
    - Server đếm số lần chữ cái “s” xuất hiện trong chuỗi và gửi lại cho client số lần xuất hiện này.
  - Kịch bản 2
    - Client gửi một chuỗi ký tự viết thường sang cho server
    - Server chuyển thành chuỗi ký tự viết hoa và chuyển chuỗi ký tự viết hoa này sang cho client
    - Client sau khi nhận được chuỗi ký tự viết hoa có thể tiếp tục gửi thêm các chuỗi ký tự khác sang cho server.

# Lập trình Socket

- 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

# Lập trình Socket

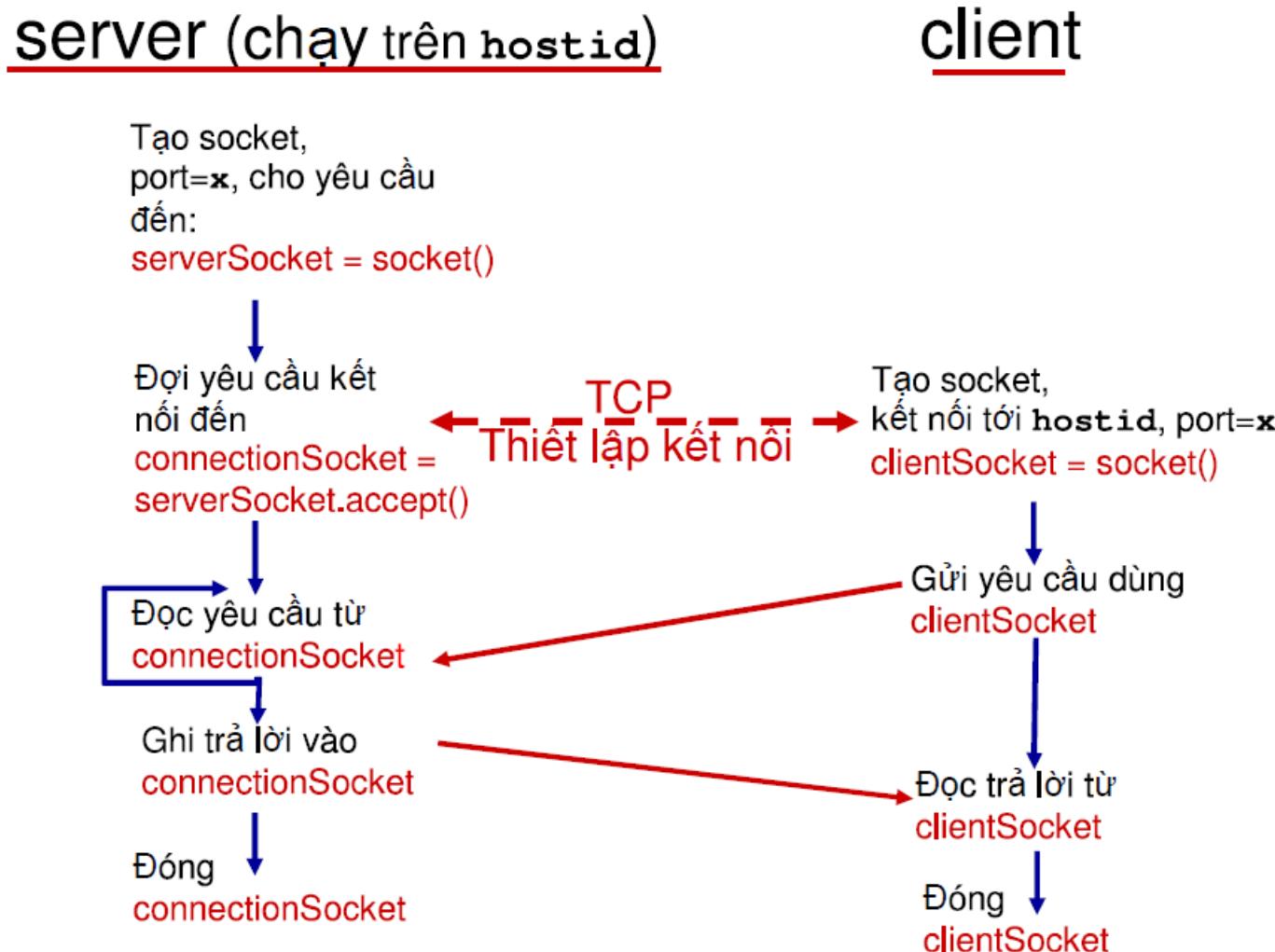
- Lập trình socket với TCP
  - Khi được tiếp xúc bởi client, TCP server sẽ tạo socket mới cho tiến trình server để truyền thông với client
    - Cho phép server “nói chuyện” với nhiều client
    - Các số hiệu cổng nguồn được dùng để phân biệt các client

## Quan điểm ứng dụng

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

# Lập trình Socket

- Tương tác socket client/server : TCP



# Lập trình Socket

- Ví dụ: TCP client

## *Python TCPClient*

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

Tạo TCP socket cho  
server, port từ xa 12000



clientSocket = socket(AF\_INET, SOCK\_STREAM)



Không cần gắn tên name,  
port



clientSocket.connect((serverName, serverPort))

# Lập trình Socket

- Ví dụ: TCP server

## *Python TCPServer*

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(("",serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
connectionSocket.close()
```

Tạo socket TCP welcoming



```
from socket import *
```

```
serverPort = 12000
```

```
serverSocket = socket(AF_INET,SOCK_STREAM)  
serverSocket.bind(("",serverPort))
```

```
serverSocket.listen(1)
```

```
print 'The server is ready to receive'
```

```
while True:
```

server đợi accept() cho các  
yêu cầu đang đến, socket mới  
được tạo khi trả lời lại



```
connectionSocket, addr = serverSocket.accept()
```

Đọc các bytes từ socket  
(nhưng không đánh địa chỉ  
như trong UDP)



```
sentence = connectionSocket.recv(1024).decode()
```

```
capitalizedSentence = sentence.upper()
```

```
connectionSocket.send(capitalizedSentence.
```

```
encode())
```

Đóng kết nối tới client  
(nhưng không welcoming  
socket)



```
connectionSocket.close()
```

# Bài tập chương 1+2

- Kiểm tra thường xuyên
  1. <https://forms.gle/VmujxvbcdTvKeTCq5>
  2. https://forms.gle/CnB1VM3xWCjRXFAu5