Nhập môn mạng máy tính

Chương 1: Giới thiệu

1.1 Internet là gì?

Hàng triệu các thiết bị máy tính được kết nối:

- Hosts = hệ thống đầu cuối
- Chạy ứng dụng mạng

Các liên kết truyền thông:

- Cáp quang, cáp đồng, radio, vệ tinh
- Tốc độ truyền: băng thông (bandwidth)

Chuyển mạch gói: chuyển tiếp gói tin (khối dữ liệu)

- Thiết bị định tuyến (routers) và thiết bị chuyển mạch (switches)

Internet: "mạng của các mạng"

- Các nhà cung cấp dịch vụ mạng (ISPs) được kết nối với nhau.

Các giao thức điều khiển gửi, nhận thông tin

- Ví dụ: TCP, IP, HTTP, Skype, 802.11

Các chuẩn Internet

- RFC: Request for comments
- IETF: Internet Engineering Task Force

Cơ sở hạ tầng cung cấp các dịch vụ cho các ứng dụng:

- Web, VoIP, email, games, thương mại điện tử, mạng xã hội, …

Cung cấp giao diện lập trình cho các ứng dụng

- Cái móc (hooks) cho phép gửi và nhận các chương trình ứng dụng để "kết nối" với Internet
- Cung cấp các lựa chọn dịch vụ, tương tự như dịch vụ bưu chính.

Giao thức định nghĩa định dạng, thứ tự các thông điệp được gởi và nhận giữa các thực thể mạng, và các hành động được thực hiện trên việc truyền và nhận thông điệp

1.2 Mạng biên

Mạng biên (network edge):

- Nút mạng đầu cuối (end-system, host): PC, điện thoại, máy chủ, máy tính nhúng
- Mạng truy cập (access network): đường truyền, thiết bị kết nối (router, switch, hub,...)

Chức năng của host (end-system):

- Nhận data từ tầng ứng dụng (application) rồi chia nhỏ thành các packets, chiều dài là L bits
- Truyền packet trong mạng với tốc độ truyền R
- Tốc độ truyền của đường link, còn được gọi là khả năng/công suất của đường link, còn được gọi là băng thông của đường link

$$\text{Độ trễ truyền gói (packet)} = \frac{\text{Chiều dài gói (bits)}}{\text{Băng thông đường link(bps)}} = \frac{\text{L}}{\text{R}}$$

1.3 Mạng lõi

Lưới các bộ định tuyến (router) được kết nối với nhau.

Hai chức năng chính của mạng lõi:

- Routing: xác định đường đi từ nguồn đến đích được thực hiện bởi các gói tin
- Fowarding: chuyển các packet từ đầu vào của bộ định tuyến đến đầu ra thích hợp của bộ định tuyến đó

Chuyển mạch gói

- Host chia nhỏ dữ liệu từ tầng ứng dụng thành các packet
- Chuyển tiếp các gói từ bộ định tuyến này đến bộ định tuyến tiếp theo trên đường đi từ nguồn tới đích
- Mỗi packet được truyền với công suất lớn nhất của đường truyền

Độ trễ hàng đợi, sự mất mát: nếu tốc độ đến của đường link cao hơn tốc độ truyền của đường link trong một khoảng thời gian:

- Các packet sẽ xếp hàng và đợi được truyền tải tạo ra độ trễ hàng đợi

- Các packet có thể bị bỏ (mất) nếu bộ nhớ hàng đợi (bộ nhớ đệm) đầy

Chuyển mạch kênh

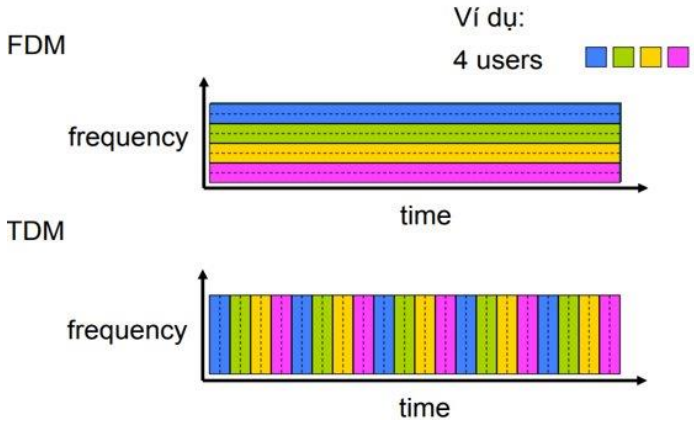
Khi 2 điểm đầu cuối muốn trao đổi thông tin sẽ thiết lập một kênh truyền riêng (circuit) với băng thông được cấp phát dành riêng cho cả hai cho tới khi truyền tin kết thúc

- Kênh không được chia sẽ nên sẽ rảnh rỗi lúc không truyền tin
- Thường được sử dụng trong các mạng điện thoại truyền thống

Ghép kênh: gửi dữ liệu của nhiều kênh khác nhau trên một đường truyền vật lý

Phân kênh: phân dữ liệu được truyền trên đường truyền vật lý vào các kênh tương ứng và chuyển cho đúng đích

- Ghép kênh theo thời gian (TDM): mỗi kết nối sử dụng tài nguyên trong thời gian được phân.
- Ghép kênh theo tần số (FDM): mỗi kết nối sử dụng một băng tần tín hiệu riêng



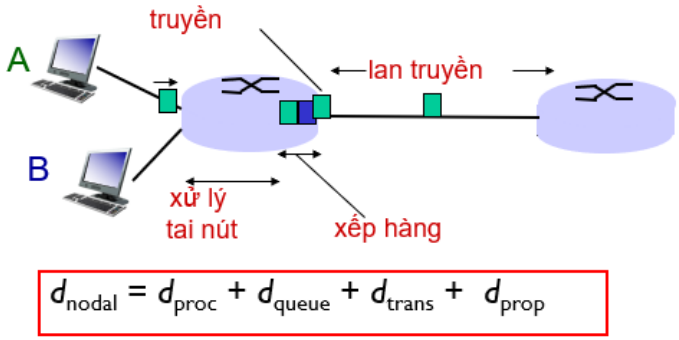
So sánh giữa chuyển mạch gói và chuyển mạch kênh

Chuyển mạch gói	Chuyển mạch kênh
❖ **Ưu điểm:** - Băng thông được sử dụng tốt hơn do không bị giới hạn trên một kênh riêng - Lỗi đường truyền không làm chậm trễ do có nhiều đường truyền khác nhau - Nhiều người có thể sử dụng chung một đường truyền ❖ **Nhược điểm:** - Độ trễ truyền lớn - Dễ xảy ra mất gói tin nếu dung lượng lớn	❖ **Ưu điểm:** - Tốc độ truyền và băng thông cố định - Không có thời gian chờ tại các nút chuyển tiếp - Có thể sử dụng lâu dài - Chất lượng ổn định, không bị mất gói ❖ **Nhược điểm:** - Nhiều kênh thì băng thông càng nhỏ - Chỉ một kênh truyền được sử dụng tại một thời điểm - Kênh truyền sẽ duy trì cho đến khi hai bên ngắt kết nối và có thể lãng phí băng thông nếu không truyền dữ liệu - Thời gian thiết lập kênh truyền quá lâu

1.4 Độ trễ, thông lượng trong mạng

4 nguồn gây ra chậm trễ gói tin

- d_{proc}: xử lý tại nút
  - Kiểm tra các bits lỗi
  - Xác định đường ra
  - Thông thường < ms
- d_{queue}: độ trễ xếp hàng
  - Thời gian đợi tại đường ra cho việc truyền dữ liệu
  - Phụ thuộc vào mức độ tắc nghẽn của bộ định tuyến
- d_{trans}: độ trễ do truyền
  - L: chiều dài gói tin (bits)
  - R: băng thông (bps)
  - d_{trans} = L/R
- d_{prop}: trễ do lan truyền
  - d: độ dài đường truyền vật lý
  - s: tốc độ lan truyền trong môi trường
  - d_{prop}= d/s



$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

Cường độ lưu thông = La/R, trong đó:

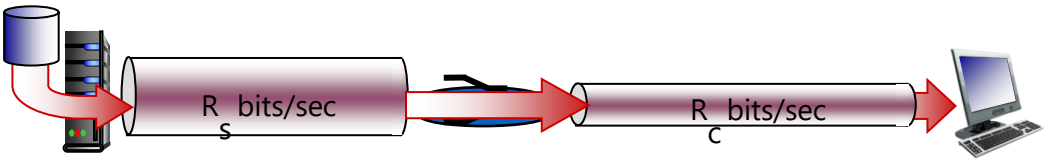
- R: Băng thông đường link(bps)
- L: Độ dài gói tin (bits)
- a: tỷ lệ trung bình gói tin đến

- La/R ~ 0: trễ trung bình nhỏ
- La/R -> 1: trễ trung bình lớn
- La/R > 1: nhiều "việc" đến hơn khả năng phục vụ, trễ trung bình vô hạn!

Thông lượng: tốc độ(bits/time unit) mà các bit được truyền giữa người gửi và nhận

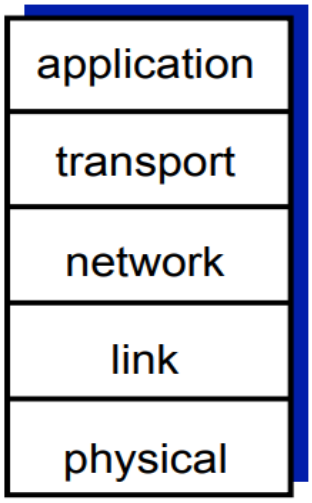
- Tức thời: tốc độ tại 1 thời điểm
- Trung bình: tốc độ trong khoảng thời gian

Đường link nút cổ chai: Đường link trên con đường từ điểm cuối này đến điểm cuối kia hạn chế thông lượng từ điểm cuối này đến điểm cuối kia



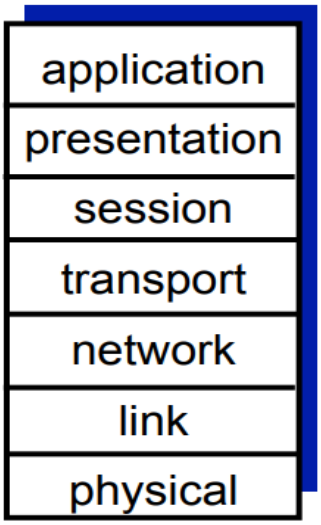
1.5 Các lớp giao thức

- Application: hỗ trợ các ứng dụng mạng
  - FTP, SMTP, HTTP,....
- Transport: chuyển dữ liệu từ tiến trình này đến tiến trình kia (process-process)
  - TCP, UDP,...
- Network: định tuyến những gói dữ liệu từ nguồn tới đích
  - IP, các giao thức định tuyến,...
- Link: chuyển dữ liệu giữa các thành phần mạng lân cận
  - Ethernet, 802.111 (WiFi), PPP,...
- Physical: bits "trên đường dây"



Mô hình ISO/OSI

- Presentation: cho phép các ứng dụng giải thích ý nghĩa của dữ liệu, ví dụ mã hóa, nén, những quy ước chuyên biệt
- Session: sự đồng bộ hóa, khả năng chịu lỗi, phục hồi sự trao đổi dữ liệu

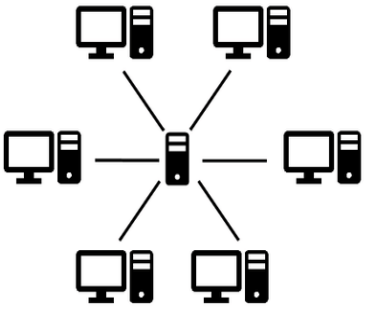


Chương 2: Tầng Application

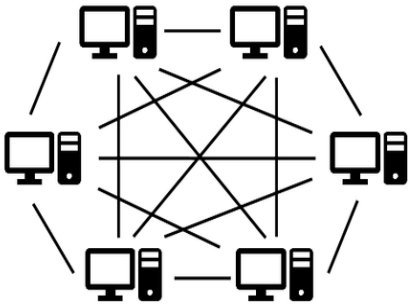
2.1 Các nguyên lý của các ứng dụng mạng

Các kiến trúc ứng dụng:

- Kiến trúc Client – Server
- Kiến trúc P2P (Peer to peer) (Mạng ngang hàng)



Client-Server network



P2P network

	Kiến trúc Client – Server	Kiến trúc P2P
Server	- Host luôn luôn hoạt động - Địa chỉ IP cố định - Trung tâm phục vụ và lưu trữ dữ liệu	- Không có server
Client	- Giao tiếp với server - Có thể kết nối không liên tục - Có thể dùng địa chỉ IP động - Không giao tiếp trực tiếp với các client khác	- Các hệ thống đầu cuối giao tiếp trực tiếp với nhau - Các peer yêu cầu dịch vụ từ các peer khác và cung cấp dịch vụ ngược lại từ các peer khác => **Có khả năng tự mở rộng** - Các peer được kết nối không liên tục và có thể thay đổi địa chỉ IP - Quản lí phức tạp

Các tiến trình liên lạc:

- Tiến trình (Process) là chương trình đang chạy trong một host
- Trong cùng một host, hai tiến trình giao tiếp với nhau bằng cách sử dụng truyền thông liên tiến trình (inter-process communication) được định nghĩa bởi hệ điều hành
- Các tiến trình trong các host khác nhau truyền thông với nhau bởi trao đổi các thông điệp (message)

Clients, Servers:

- Tiến trình Client: tiến trình khởi tạo truyền thông
- Tiến trình Server: tiến trình chờ đợi để được liên lạc

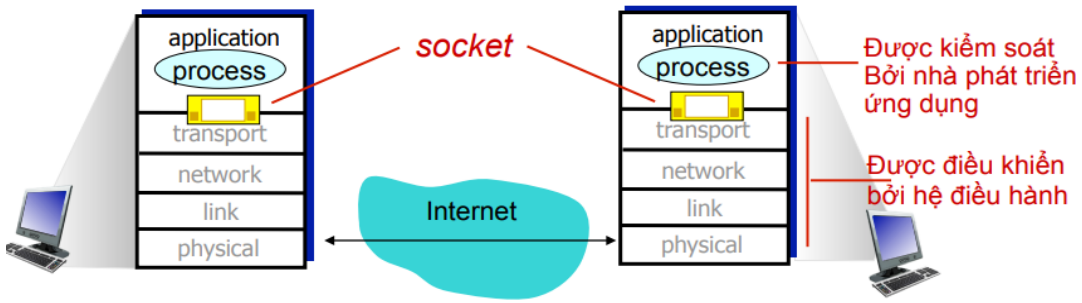
Chú ý: Các ứng dụng với kiến trúc P2P có cả tiến trình client và tiến trình server

Sockets

Điểm truy cập dịch vụ của tầng Transport

=> Socket tương tự một cổng ra vào

Các tiến trình sử dụng socket gọi dịch vụ của tầng giao vận để trao đổi thông điệp



Để nhận thông điệp, tiến trình phải có định danh (identifier),bao gồm:

- Địa chỉ IP
- Số cổng (port numbers)

Thiết bị host device có địa chỉ IP 32-bit duy nhất

Một số Port tầng Application:

Protocol	Port number	Protocol	Port number
FTP data	20	HTTP	80
FTP control	21	RIP	520
SSH	22	Telnet	23
DNS	53	HTTPS	443
LPD	515	TFTP	69
SMTP	25	NFS	2049
SNMP	161(TCP),162(UDP)	POP3	110

Các dịch vụ giao thức Transport Internet

- **Dịch vụ TCP:**
  - Reliable transport(truyền tải tin cậy) giữa tiến trình gửi và nhận
  - Flow control(điều khiển luồng): người gửi sẽ không áp đảo người nhận
  - Congestion control(điều khiển tắc nghẽn): điều tiết người gửi khi mạng quá tải
  - Connection-oriented(hướng kết nối): thiết lập được yêu cầu giữa tiến trình client và server

Ví dụ các giao thức TCP: FTP data, FPT control (port 21), Telnet, HTTPS, POP3 (port 110), SMTP, …

- **Dịch vụ UDP:**
  - Truyền tải dữ liệu không tin cậy giữa tiến trình gửi và nhận
  - Không hỗ trợ: độ tin cậy, điều khiển luồng, điều khiển tắc nghẽn, đảm bảo thông lượng, bảo mật và thiết lập kết nối

Ví dụ các giao thức UDP: RIP (port 520), TFTP, SNMP (port 161),…

Chú ý: các giao thức DNS, HTTP, SSH dùng chung cả 2 dịch vụ TCP và UDP.

So sánh giữa 2 dịch vụ:

- Giống nhau: Đều là các giao thức mạng TCP/IP, có chức năng kết nối các máy tính với nhau, có thể gửi dữ liệu cho nhau.
- Khác nhau:

TCP	UDP
Hướng kết nối	Hướng không kết nối
Thường dùng cho mạng WAN	Thường dùng cho mạng LAN
Độ tin cậy cao	Độ tin cậy thấp
Gửi dữ liệu dạng luồng byte	Gửi đi Datagram
Không cho phép mất gói tin	Cho phép mất gói tin
Đảm bảo việc truyền dữ liệu	Không đảm bảo việc truyền dữ liệu
Có sắp xếp thứ tự các gói tin	Không sắp xếp thứ tự các gói tin
Tốc độ truyền thấp hơn UDP	Tốc độ truyền cao

2.2 Web và HTTP

Web page: là một tập hợp các văn bản, hình ảnh, tệp tin (gọi chung là các đối tượng – objects) thích hợp với World Wide Web và được thực thi ở trình duyệt web. Mỗi đối tượng có thể được định danh địa chỉ bởi một URL.

HTTP (Hypertext Transfer Protocol): là giao thức web ở tầng Application

Mô hình Client / Server:

- Client: trình duyệt yêu cầu và nhận (sử dụng giao thức HTTP), hiển thị các đối tượng của web.
- Server: Web server gửi (sử dụng giao thức HTTP) các đối tượng để đáp ứng yêu cầu của Client
- RTT (Round Trip Time): khoảng thời gian (tính bằng ms) để một gói tin nhỏ đi từ Client đến Server và quay ngược trở lại

Các kết nối HTTP

<table>
<tr><th>HTTP không bền vững
(Nonpersistent HTTP)</th><th colspan="2">HTTP bền vững
(Persistent HTTP)</th></tr>
<tr><td>- Chỉ tối đa một đối tượng được gửi qua kết nối TCP. Kết nối sau đó sẽ bị đóng
- Tải nhiều đối tượng yêu cầu nhiều kết nối</td><td colspan="2">- Nhiều đối tượng có thể gửi qua một kết nối TCP giữa Client và Server</td></tr>
<tr><td>HTTP/1.0 (RFC 1945)</td><td colspan="2">HTTP/1.1 (RFC 2616)</td></tr>
<tr><td rowspan="2">Thời gian đáp ứng
- Một RTT để khởi tạo kết nối TCP
- Một RTT cho yêu cầu HTTP và vài byte đầu tiên của đáp ứng HTTP được trả về
- Thời gian truyền file
- Thời gian đáp ứng HTTP không bền vững = 2RTT + thời gian truyền file</td><td colspan="2">Thời gian đáp ứng</td></tr>
<tr><td>Persistent without pipelining</td><td>Persistent with pipelining</td></tr>
<tr><td></td><td>- 1RTT cho việc kết nối
- 1RTT cho mỗi đối tượng(không cần kết nối lại)</td><td>- 1RTT cho việc kết nối
- 1RTT cho tất cả đối tượng(không cần kết nối lại)</td></tr>
<tr><td>Các phương thức:
GET
POST
HEAD</td><td colspan="2">Các phương thức:
GET
POST
HEAD
PUT
DELETE</td></tr>
</table>

Các mã trạng thái đáp ứng HTTP

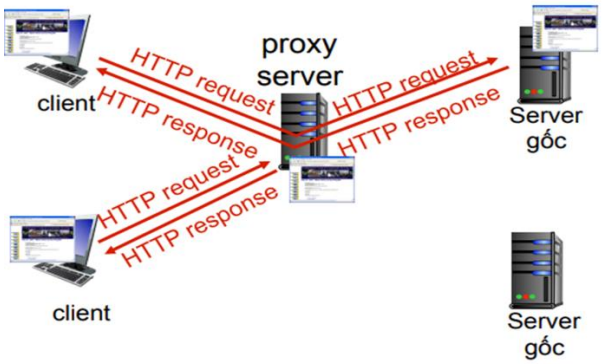
- Mã trạng thái xuất hiện trong dòng đầu tiên trong thông điệp đáp ứng từ server tới client
- Một số mã trạng thái thường gặp:
  - 200 OK – Yêu cầu thành công, đối tượng được yêu cầu sau ở trong thông điệp này
  - 301 Moved Permanently – Đối tượng được yêu cầu đã di chuyển, vị trí mới được xác định sau trong thông điệp này
  - 400 Bad Request – Thông điệp yêu cầu không được hiểu bởi server
  - 404 Not Found – Tài liệu được yêu cầu không tìm thấy trên server này
  - 505 HTTP Version Not Supported – Máy chủ không hỗ trợ phiên bản giao thức HTTP

Cookies

- Là dữ liệu được gửi từ server tới trình duyệt của người dùng (Trạng thái User-server). Trình duyệt sẽ lưu trữ cookie này và gửi lại theo mỗi HTTP request về cho cùng server đó
- Về cơ bản, cookie dùng để nói cho server biết các request đến từ một trình duyệt
- Do HTTP là stateless, mọi request đến server đều giống nhau, nên server không thể phân biệt request được gửi đến là từ một client đã thực hiện request trước đó hay từ một client mới
- Gồm 4 thành phần:
  - Cookie header line của thông điệp đáp ứng HTTP
  - Cookie header line trong thông điệp đáp ứng HTTP kế tiếp
  - File cookie được lưu trữ trên host của người dùng, được quản lý bởi trình duyệt của người sử dụng
  - Cơ sở dữ liệu back-end tại Website

Một số ứng dụng: sự cấp phép, giỏ mua hàng, các khuyến cáo, trạng thái phiên làm việc của user (Web email)

Web caches (proxy server)



Mục tiêu:

- Đáp ứng yêu cầu của client mà không cần liên quan đến server gốc (server chứa đối tượng mà client cần)
- Giảm thời gian đáp ứng cho yêu cầu của client
- Giảm lưu lượng trên đường link truy cập của một tổ chức

FTP

Giao thức truyền file

- Truyền file đến/từ host từ xa

Mô hình client-server:

- Client: phía khởi tạo truyền
- Server: host ở xa

Kết nối điều khiển: TCP ở port 21

Kết nối dữ liệu: TCP ở port 20

2.4 Email

Gồm ba thành phần chính: user agents, mail servers, SMTP

- User agent (tác nhân người dùng):
  - Soạn thảo, sửa đổi, đọc các thông điệp email
  - Các thông điệp đi và đến được lưu trên server
  - Ví dụ: Outlook, Thunderbird, …
- Mail servers:
  - Hộp thư (mailbox) chứa thông điệp đến user
  - Hàng thông điệp (message queue) của các thông điệp mail ra ngoài (chuẩn bị gửi)
  - Giao thức SMTP giữa các mail server để gửi các thông điệp email
    - Client: gửi mail đến server
    - "Server": nhận mail từ server

- SMTP [RFC 2821], port 25 (Simple Mail Transfer Protocol)
  - Sử dụng TCP để truyền thông điệp email một cách tin cậy từ client đến server
  - Truyền trực tiếp: server gửi đến server nhận
  - 3 giai đoạn truyền:
    - Thiết lập kết nối
    - Truyền thông điệp
    - Đóng kết nối
- Tương tác lệnh/ phản hồi tương tự HTTP, FTP:
  - Lệnh: văn bản ASCII
  - Phản hồi: mã và cụm trạng thái
- Thông điệp phải ở dạng mã ASCII 7 bit

⇨ SMTP dùng kết nối bền vững, yêu cầu thông điệp phải ở dạng ASCII 7 bit, dùng CRLF để xác định kết thúc thông điệp

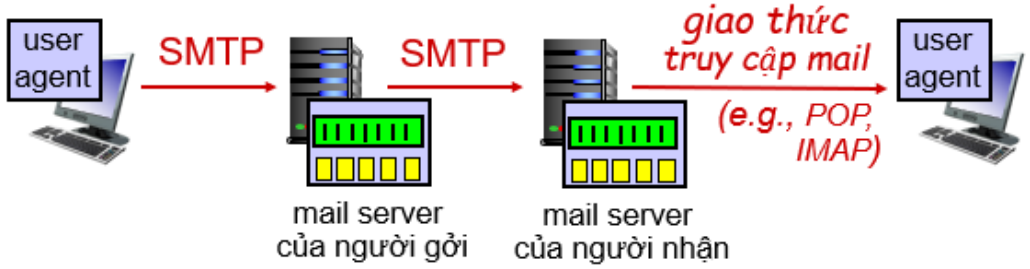
So sánh SMTP và HTTP

Giống nhau: đều có tương tác lệnh/phản hồi, các mã trạng thái dạng ASCII.

Khác nhau:

SMTP	HTTP
Đẩy	Kéo
Nhiều đối tượng được gửi trong thông điệp nhiều phần	Mỗi đối tượng được đóng gói trong thông điệp phản hồi của nó

Các giao thức truy cập mail



SMTP: truyền dẫn/lưu trữ vào server của người nhận

Giao thức truy cập mail: trích xuất từ server

- POP(Post Office Protocol)[RFC 1939]
  - Đăng nhập vào lấy hết thư từ mail server về client
- IMAP(Internet Mail Access Protocol) [RFC 1730]
  - Phức tạp hơn POP
  - Cho phép lưu trữ và xử lý thư trên máy chủ
- HTTP: Gmail, Yahoo, Outlook,..

2.5 DNS

DNS(Domain Name System): hệ thống phân giải tên miền, chuyển đổi các tên miền website sang địa chỉ IP dạng số và ngược lại

Ta đã biết rằng các thiết bị người dùng truy cập vào host device thông qua địa chỉ IP 32-bit và cổng Port. Nhưng chúng thường khó nhớ hơn so với việc truy cập thông qua tên, ví dụ "Google.com". Vì thế ta cần dịch vụ có thể dịch ngược từ tên miền ra địa chỉ IP tương ứng và ngược lại

Phân giải tên miền:

- Truy vấn lặp: server được liên lạc sẽ trả lời với tên server đã liên lạc
- Truy vấn đệ quy: đẩy trách nhiệm phân giải tên cho name server đã được tiếp xúc

  => Tải nặng tại các tầng trên của hệ thống phân cấp

Các dịch vụ DNS:

- Dịch tên host ra địa chỉ IP
- Bí danh host
- Bí danh mail server
- Phân phối tải

Các DNS có thẩm quyền:

- DNS server của riêng tổ chức cung cấp các tên host có thẩm quyền để ánh xạ địa chỉ IP cho các host được đặt tên của tổ chức đó.
- Có thể được duy trì bởi tổ chức hoặc nhà cung cấp dịch vụ.

DNS name server cục bộ:

- Không hoàn toàn theo cấu trúc phân cấp.

- Mỗi ISP (nhà cung cấp dịch vụ Internet) có một server cục bộ.
- Khi một host tạo một truy vấn DNS, truy vấn sẽ được gởi đến DNS server cục bộ của nó.

Một khi name server học cách ánh xạ, nó sẽ caches ánh xạ đó.

- Các mục cache sẽ biến mất sau một vài lần TTL (time to live, là thời gian tồn tại của một bản ghi (record) cấu hình tên miền được nhớ bởi một máy chủ DNS trung gian)
- TLD servers thường được cache trong các name server cục bộ
- Các name server gốc không thường xuyên được truy cập

Các mục được cache có thể hết hạn sử dụng

- Nếu tên host thay đổi địa chỉ IP, có thể không được biết đến trên Internet cho đến khi tất cả TTL hết hạn

Cơ chế cập nhật/thông báo được đề xuất bởi chuẩn IETF

Các DNS record

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

- Type A lưu ở authoritative (mp3.zing.vn, ...)
  - name: tên host
  - value: địa chỉ IP
- Type NS lưu ở TLD (Zing.vn, ...)
  - name: tên miền
  - value: tên host của name server có thẩm quyền cho tên miền này
- Type CNAME lưu ở TLD, chứa tên miền thật, (www.ibm.com is really servereast.backup2.ibm.com)
  - name: bí danh của một số "tên chuẩn"
  - value: tên chuẩn
- Type MX lưu ở TLD dùng cho mail (mail.bar.foo.com,...)
  - value: tên của mail server được liên kết với name

Nhược điểm của DNS:

- Nếu điểm tập trung bị hỏng, toàn bộ hệ thống sẽ tê liệt
- Số lượng yêu cầu phục vụ tại điểm tập trung rất lớn

- Chi phí bảo trì hệ thống rất lớn
- Khó khắc phục khi xảy ra sự cố, dễ bị tấn công

2.6 Lập trình socket với UDP và TCP

Hai loại socket cho 2 dịch vụ transport:

- UDP: datagram không đáng tin cậy
- TCP: tin cậy, byte được định hướng dòng

Lập trình socket với UDP

UDP: không "kết nối" giữa client và server

- Không bắt tay trước khi gửi dữ liệu
- Bên gửi chỉ rõ địa chỉ IP đích và số port cho mỗi packet
- Bên nhận lấy địa chỉ IP và số port của người gửi từ packet được nhận

=> Dữ liệu được truyền có thể bị mất hoặc được nhận không thứ tự

Python UDPClient

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

Bao gồm thư viện socket của Python
Tạo socket UDP cho server
Nhận thông điệp từ bàn phím người dùng
Đính kèm tên server, port đến thông điệp; gởi vào tron socket
Đọc các ký tự trả lời từ socket vào chuỗi
In ra chuỗi được nhận và đóng socket

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

Tạo UDP socket
Đính kèm socket đến số port cục bộ12000
Lặp mãi mãi
Đọc từ UDP socket vào trong thông điệp, lấy địa chỉ IP của client(địa chỉ IP client và port)
Gởi chuỗi chữ hoa trở lại cho client này

Lập trình socket với TCP

TCP: client phải tiếp xúc với server

- Tiến trình server phải được chạy trước
- Server phải tạo socket để mời client đến liên lạc
- Tạo socket TCP, xác định địa chỉ IP, số port của tiến trình server
- Khi client tạo socket: client TCP thiết lập kết nối đến server TCP
- Khi đã tiếp xúc với client: server TCP tạo socket mới cho tiến trình server để truyền thông với client đó

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

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)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

Tạo TCP socket cho server, port 12000 ở xa

Không cần đính kèm tên server, port

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 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

Tạo socket TCP chào đón

server bắt đầu lắng nghe các yêu cầu TCP đến

Lặp mãi mãi

server đợi accept() cho yêu cầu đến, socket mới được tạo trở về

Đọc các byte từ socket nhưng không đọc địa chỉ như UDP)

Đóng kết nối đến client này(nhưng không đóng socket chào đón)

Chương 3: Tầng Transport

3.1 Các dịch vụ tầng vận chuyển

Cung cấp truyền thông logic giữa các tiến trình ứng dụng đang chạy trên các host khác nhau

Các giao thức (protocol) chạy trên các hệ thống đầu cuối

- Phía gửi: chia nhỏ các thông điệp (message) ứng dụng thành các segments, sau đó chuyển các segments này cho tầng Mạng
- Phía nhận: tái kết hợp các segments thành các thông điệp (message), các thông điệp này được chuyển lên tầng Ứng dụng

Giao thức tầng Vận chuyển dành cho các ứng dụng: TCP và UDP

Quan hệ giữa Tầng Vận chuyển và tầng Mạng:

- Tầng Mạng: truyền thông logic giữa các host
- Tầng Vận chuyển: truyền thông logic giữa các tiến trình
    - Dựa trên dịch vụ tầng mạng

So sánh giao thức TCP và UDP

TCP	UDP
Hướng kết nối	Hướng không kết nối
Độ tin cậy cao	Độ tin cậy thấp
Gửi dữ liệu dạng luồng byte	Gửi đi Datagram
Không cho phép mất gói tin	Cho phép mất gói tin
Đảm bảo việc truyền dữ liệu	Không đảm bảo việc truyền dữ liệu
Có sắp xếp thứ tự các gói tin	Không sắp xếp thứ tự các gói tin
Tốc độ truyền thấp hơn UDP	Tốc độ truyền cao

3.2 Multiplexing và Demultiplexing

Multiplexing: Tại bên gửi

- Xử lý dữ liệu từ nhiều socket, thêm thông tin header về tầng Vận chuyển vào segment (được sử dụng sau cho demultiplexing)

Demultiplexing: Tại bên nhận

- Sử dụng thông tin trong header để chuyển segment vừa nhận vào đúng socket
- Host nhận các gói dữ liệu (datagram) IP
    - Mỗi gói dữ liệu có địa chỉ IP nguồn và đích
    - Mỗi gói dữ liệu mang một segment tầng Vận chuyển
    - Mỗi segment có số port nguồn và đích
- Host dùng các địa chỉ IP và số port để gởi segment đến socket thích hợp

Demultiplexing không kết nối

- Khi host nhận segment UDP
    - Kiểm tra số port đích trong segment
    - Đưa segment UDP đến socket có số port đó
- Các gói dữ liệu IP với cùng số port đích, nhưng khác địa chỉ IP nguồn và/hoặc khác số port nguồn sẽ được chuyển đến cùng socket tại máy đích

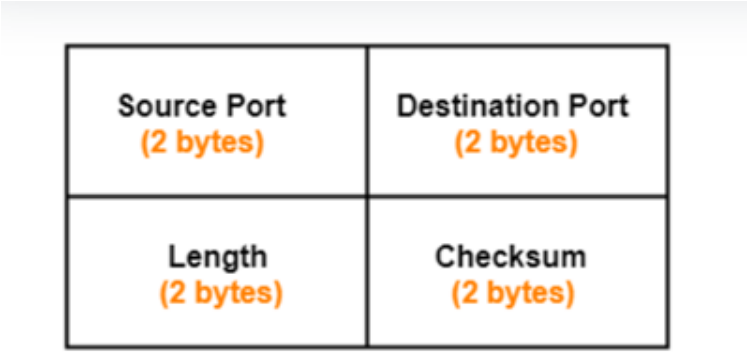
Demultiplexing hướng kết nối

- Socket TCP được xác định bởi 4 yếu tố
    - Địa chỉ IP nguồn
    - Số port nguồn
    - Địa chỉ IP đích
    - Số port đích
- Demux: nơi nhận dùng tất cả 4 giá trị trên để điều hướng segment đến socket thích hợp
- Host server có thể hỗ trợ nhiều socket TCP đồng thời: Mỗi socket được xác định bởi bộ 4 của nó
- Các web server có các socket khác nhau cho mỗi kết nối từ client

  ** HTTP không bền vững sẽ có socket khác nhau cho mỗi yêu cầu*

3.3 Vận chuyển phi kết nối UDP

- UDP(RFC 768): đơn giản, không rườm rà là một giao thức thuộc Transport
- Các segment của UDP có thể bị:
  - Mất mát
  - Vận chuyển không theo thứ tự đến ứng dụng
- Connectionless(phi kết nối): không bắt tay giữa bên nhận và bên gửi; mỗi segment được xử lí độc lập
- Ứng dụng của UDP: DNS, SNMP... Các ứng dụng đa phương tiện trực tuyến chịu mất mát data nhưng cần tốc độ như live video, stream...



UDP Header

- UDP segment header
- Payload: 32bits
- UDP checksum: dò tìm "các lỗi" (các bit cờ được bật) trong các segment đã được truyền

3.4 Các nguyên lý truyền dữ liệu tin cậy

RDT 1.0

- Hoạt động trên một kênh hoàn toàn đáng tin cậy, tức là, nó giả định rằng kênh cơ bản có:
  - Không có lỗi bit
  - Không mất gói tin
  - Một số hàm:
    - Bên gửi:
      - rdt_send(): được gọi bởi tầng Ứng dụng. Chuyển dữ liệu cần truyền đến tầng Ứng dụng bên nhận

        - udt_send(): được gọi bởi rdt, để truyền các gói trên kênh không tin cậy đến nơi nhận
    - Bên nhận:
        - deliver_data(): được gọi bởi rdt để chuyển dữ liệu đến tầng cao hơn
        - rdt_rcv(): được gọi khi gói dữ liệu đến kênh của bên nhận
- FSM(finite state machines) riêng biệt cho cả bên nhận và gửi
    - Bên gửi: Khi người gửi gửi dữ liệu từ lớp ứng dụng, RDT chỉ cần chấp nhận dữ liệu từ lớp trên thông qua sự kiện rdt_send (data). Sau đó, nó đưa dữ liệu vào một gói (thông qua sự kiện make_pkt (packet, data)) và gửi gói vào kênh bằng sự kiện udp_send (packet).
    - Bên nhận: Khi nhận dữ liệu từ kênh, RDT chỉ cần chấp nhận dữ liệu thông qua sự kiện rdt_rcv (data). Sau đó, nó extract dữ liệu từ packet(thông qua hành động make_pkt (packet, data)) và gửi dữ liệu đến lớp ứng dụng bằng cách sử dụng sự kiện delivery_data (data).
        - Bên nhận không yêu cầu phản hồi vì kênh hoàn toàn đáng tin cậy, tức là không có lỗi nào có thể xảy ra trong quá trình truyền dữ liệu qua kênh bên dưới.

RDT 2.0

- Nguyên nhân ra đời: Trong quá trình truyền dữ liệu xảy ra lỗi thì sẽ khắc phục như nào
- RDT 2.0 hoạt động trên cậy qua kênh lỗi bit. Đây là một mô hình thực tế hơn để kiểm tra các lỗi bit có trong một kênh trong khi truyền nó có thể là các bit trong gói bị hỏng. Các lỗi bit như vậy xảy ra trong các thành phần vật lý của mạng khi một gói được truyền. Trong trường hợp này, chúng ta sẽ giả sử rằng tất cả các gói đã truyền được nhận theo thứ tự mà chúng đã được gửi đi (cho dù chúng có bị hỏng hay không)
- Phát hiện lỗi: Checksum Field
- Khắc phục lỗi: dùng **ACK, NAK**
    - ACKs: Packet nhận được là đúng và không bị hỏng .Bên nhận thông báo cho bên gửi đã nhận packet thành công
    - NAKs: Packet nhận được bị hỏng. Bên nhận thông báo cho bên gửi packet bị lỗi và bên gửi phải gửi lại packet nào được xác nhận là NAK

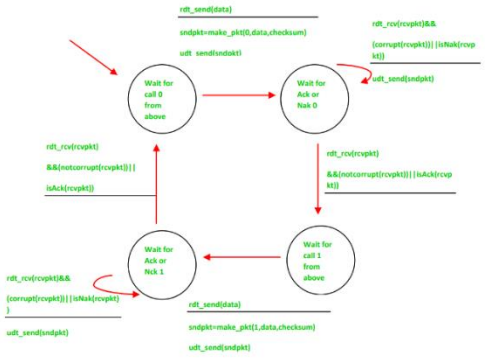
- FSM:
  - Bên gửi:
    - Có hai trạng thái. Ở một trạng thái, giao thức phía gửi đang đợi dữ liệu được truyền từ lớp trên xuống lớp dưới. Ở trạng thái khác, giao thức người gửi đang chờ một gói ACK hoặc NAK từ người nhận (phản hồi)
    - Nếu một ACK được nhận, tức là rdt_rcv (rcvpkt) && is ACK (rcvpkt), người gửi biết rằng gói được truyền gần đây nhất đã được nhận đúng và do đó giao thức trở lại trạng thái chờ dữ liệu từ lớp trên
    - Nếu một NAK được nhận, giao thức sẽ truyền lại packet cuối cùng và đợi một ACK hoặc NAK được người nhận trả về để phản hồi lại gói dữ liệu được truyền lại. Điều quan trọng cần lưu ý là khi người nhận ở trạng thái chờ ACK hoặc NAK, nó không thể lấy thêm dữ liệu từ lớp trên, điều đó sẽ chỉ xảy ra sau khi người gửi nhận được ACK và rời khỏi trạng thái này. Do đó, người gửi sẽ không gửi một phần dữ liệu mới cho đến khi chắc chắn rằng người nhận đã nhận đúng gói tin hiện tại, do hành vi này của giao thức mà giao thức này còn được gọi là *Stop and Wait Protocol*
    - Bên nhận: Trang web bên nhận có một trạng thái duy nhất, ngay khi packet đến, bên nhận sẽ trả lời bằng ACK hoặc NAK, tùy thuộc vào việc packet nhận được có bị hỏng hay không, tức là bằng cách sử dụng rdt_rcv (rcvpkt) && corrupt (rcvpkt) ở đâu một packet được nhận và được phát hiện là có lỗi hoặc rdt_rcv (rcvpkt) && not corrupt (rcvpkt) trong đó packet nhận được là đúng.
- **Cơ chế mới so với RDT 1.0**:
  - Phát hiện lỗi
  - Phản hồi: Bên nhận sử dụng ACK và NAK phản hồi về bên gửi
- **Vấn đề của RDT 2.0**: Nếu một ACK hoặc NAK bị hỏng, người gửi không có cách nào để biết liệu người nhận đã nhận chính xác phần dữ liệu được truyền cuối cùng hay chưa.

RDT 2.1

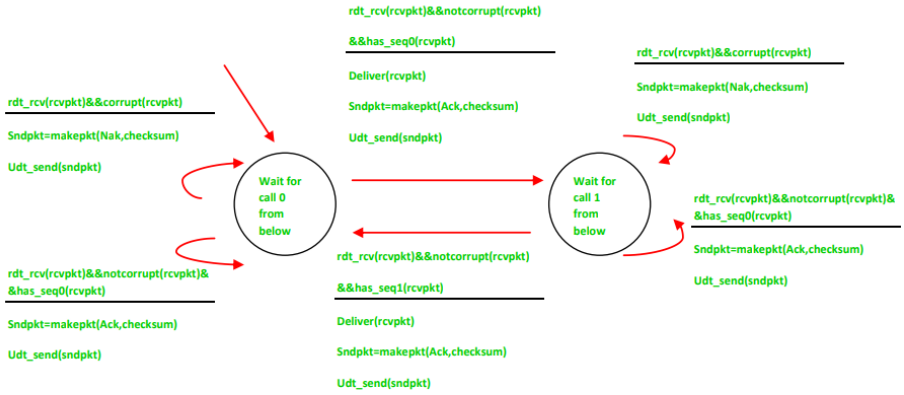
- Nguyên nhân ra đời: Giao thức RDT 2.0 giải thích chức năng trong kênh có lỗi bit. Nó giới thiệu các ACKs và NAKs, nếu người nhận nhận được gói tin

bị hỏng máy nhận sẽ gửi thông báo xác nhận NAK và ngược lại trong trường hợp chính xác. Nó không thành công khi ACK bị hỏng. RDT 2.1 giải quyết vấn đề này

- Các cách giải quyết vấn đề ACK bị hỏng:
  - Việc bổ sung thêm các bit tổng kiểm tra để phát hiện gói bị hỏng và cũng để khôi phục gói. Cần thêm dữ liệu và xử lý các packet này
  - Gửi lại dữ liệu với các xác nhận bị hỏng bởi người gửi. Đây sẽ là một lỗ hổng vì không có xử lý trùng lặp ở phía người nhận
  - Thêm một trường bổ sung tạo thành một sequence bit và người nhận có thể kiểm tra xem đó có phải là packet trùng lặp hay packet được truyền lại hay không (Tối ưu)
- FSM
  - Bên gửi:
    - Logic của sequence number là người gửi gửi các gói có số thứ tự '0' và '1' theo cách khác vì có thể theo dõi quá trình truyền liên tục của cùng một gói
    - Trạng thái 1 (*Ở trạng thái trên cùng bên trái):* được gọi là '*Wait for 0*' là bắt đầu, trạng thái Bắt đầu đợi cho đến khi nó nhận được thông báo từ lớp ứng dụng phía trên. Sau khi nó được nhận dưới dạng lớp truyền tải, nó sẽ thêm một tiêu đề lớp truyền tải được thêm với số thứ tự '0' được gửi vào mạng
    - Trạng thái 2 (*Trên cùng bên phải*): Sau khi gói được truyền vào mạng, nó sẽ chuyển sang trạng thái tiếp theo. Nếu ACK đã nhận bị hỏng hoặc trong trường hợp NAK, nó sẽ gửi lại gói tin. Trạng thái khác chuyển sang trạng thái tiếp theo
    - Trạng thái 3 và Trạng thái 4 giống như trạng thái 1 và trạng thái 2 nhưng nó gửi gói tin với số thứ tự là '0'



- Bên nhận:
  - Trạng thái 1 (*Trái*): Nếu nó nhận được gói bị hỏng, nó sẽ gửi NAK cho yêu cầu gửi lại. Ngược lại, nếu nó nhận được gói không bị hỏng có số thứ tự '1', nó sẽ gửi một ACK. Ngược lại, nếu nó nhận được gói không bị hỏng có số thứ tự '0' thì nó sẽ chuyển sang trạng thái tiếp theo
  - Trạng thái 2 (*Phải*): Nếu nó nhận được gói bị hỏng, nó sẽ gửi NAK cho yêu cầu gửi lại. Ngược lại, nếu nó nhận được gói không bị hỏng có số thứ tự '0', nó sẽ gửi một ACK. Nếu không, nếu nó nhận được gói không bị hỏng có số thứ tự '1', nó sẽ chuyển sang trạng thái tiếp theo



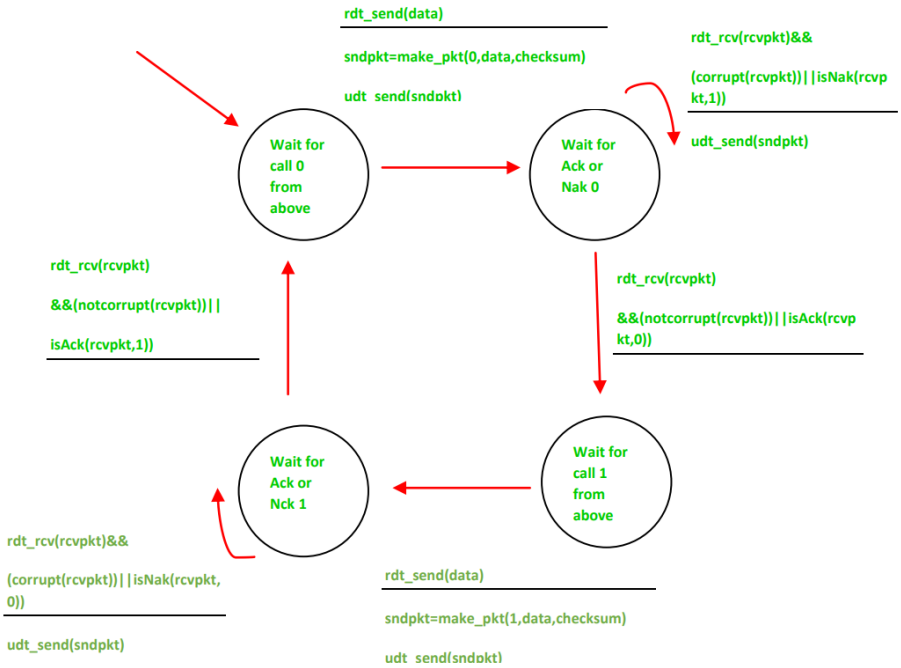
- Nhược điểm:
  - Không quản lý gói tin trùng lặp
  - Không hoạt động trong một kênh bị mất gói

RDT 2.2

- Thay đổi nổi bật trong RDT 2.2 là loại trừ NAK
- FSM:
  - Bên gửi:
    - Trạng thái 1 (*Trên cùng bên trái*): Ở trạng thái trên cùng bên trái được gọi là '*Wait for 0*' là bắt đầu, trạng thái Bắt đầu đợi cho đến khi nhận được thông báo từ lớp ứng dụng phía trên. Sau khi nó được nhận dưới dạng lớp truyền tải, nó sẽ thêm header lớp truyền tải được thêm với số thứ tự '0' được gửi vào mạng
    - Trạng thái 2 *(Trên cùng bên phải):* Trong trạng thái này, giao thức kiểm tra xem xác nhận gói tin có bị hỏng hay seq '1' được nhận là

xác nhận hay không. Trong trường hợp này, người gửi gửi lại gói tin vì trình tự được truyền đi không bằng trình tự nhận được. Nếu người gửi nhận được một số thứ tự không bị hỏng và đúng, Nó sẽ chuyển sang trạng thái tiếp theo

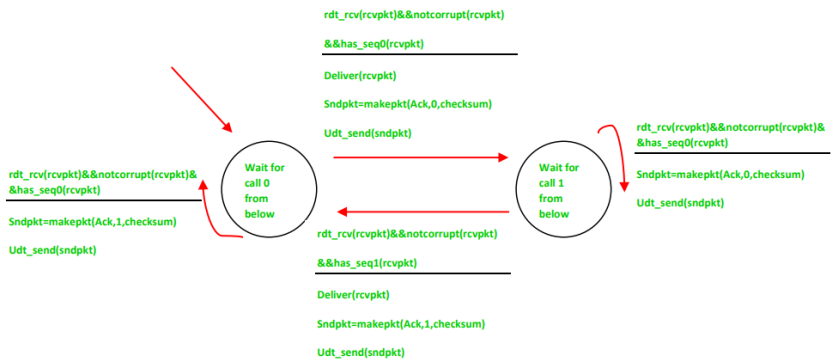
- Trạng thái 3 *(Dưới cùng bên phải):* Ở trạng thái trên cùng bên trái của hình được gọi là '*Wait for 1*' đợi cho đến khi nó nhận được thông báo từ lớp ứng dụng phía trên. Sau khi nó được nhận dưới dạng lớp truyền tải, nó sẽ thêm một tiêu đề lớp truyền tải được thêm vào với số thứ tự '1' được gửi vào mạng
- Trạng thái 4 *(Dưới cùng bên trái):* Trong trạng thái này, giao thức kiểm tra xem xác nhận gói tin có bị hỏng hay seq '0' được nhận là xác nhận hay không. Trong trường hợp này, người gửi gửi lại gói tin vì trình tự được truyền đi không bằng trình tự nhận được. Nếu người gửi nhận được một số thứ tự không bị hỏng và đúng, Nó sẽ chuyển sang trạng thái tiếp theo



- Bên nhận:
  - Trạng thái 1 (*Bên trái*): Nếu gói nhận được bị hỏng hoặc có seq với '1', nó sẽ gửi xác nhận với sequence number '1' cho nó để cho người gửi biết rằng gói được gửi không theo thứ tự. Nếu gói không bị

hỏng và có sequence number '0', máy thu sẽ trích xuất dữ liệu và gửi xác nhận với seq '0'. Nó chuyển sang trạng thái tiếp theo.

- Trạng thái 2 (*Bên phải*): Nếu gói nhận được bị hỏng hoặc có seq với '0', nó sẽ gửi xác nhận với sequence number '0' cho nó để cho người gửi biết rằng gói được gửi không theo thứ tự. Nếu gói tin không bị hỏng và có sequence number '1', người nhận sẽ trích xuất dữ liệu và gửi xác nhận với seq '1'. Nó chuyển sang trạng thái tiếp theo.



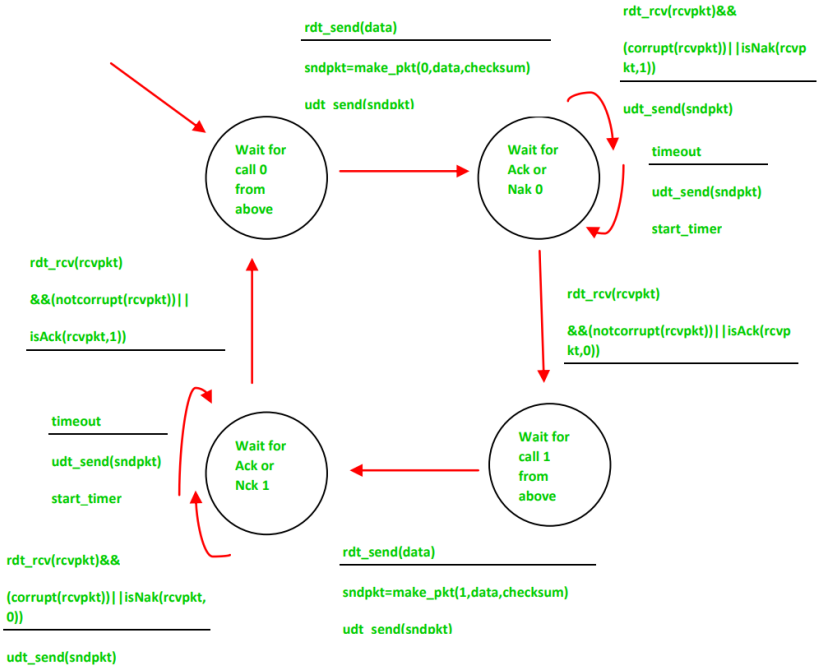
- Nhược điểm: RDT 2.2 không giải quyết mất gói.

RDT 3.0

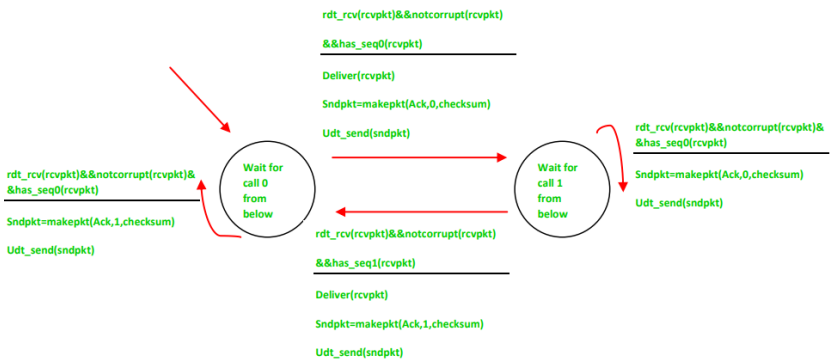
- Nguyên nhân ra đời: Trước RDT 3.0, RDT 2.2 đã được giới thiệu, để giải thích cho kênh có lỗi bit, trong đó lỗi bit cũng có thể xảy ra trong các ACK. Như thiết kế của RDT 2.2 là một giao thức "*Stop and wait*". Nếu có sự cố mạng và xác nhận / gói bị mất. Người gửi chờ đợi nó vô hạn. RDT 3.0 giới thiệu một bộ đếm thời gian ở phía người gửi nếu không nhận được thông báo xác nhận trong một thời gian cụ thể mà người gửi gửi lại gói tin. Phương pháp này giải quyết vấn đề mất gói.
- FSM:
  - Bên gửi:
    - Trạng thái 1 (*Trên cùng bên trái*): Đây là trạng thái Bắt đầu trong FSM của người gửi, được gọi là "*wait for call 0 from above*". Nó đợi cho đến khi nhận được thông báo bắt đầu từ lớp ứng dụng. Ở trạng thái này, datagram được tạo với sequence number "0" trong header và payload dưới dạng một thông báo trong phần thân của nó. Cuối

cùng, gói tin được đẩy vào mạng và việc thực thi chuyển sang trạng thái tiếp theo.

- Trạng thái 2 *(Trên cùng bên phải):* Trạng thái này xác nhận người nhận đã nhận được gói tin hay chưa. kiểm tra trạng thái xem nếu xác nhận nhận được không bị hỏng, có sequence number "1" và đạt được trong thời gian. Nếu hai tiêu chí này được thỏa mãn thì việc thực thi sẽ chuyển sang trạng thái tiếp theo, trạng thái khác thì trạng thái đó sẽ gửi lại gói tin.
- Trạng thái 3 (*Dưới cùng bên phải*): Trạng thái này trong FSM của người gửi được gọi là "*wait for a call 1 from above*". Nó đợi cho đến khi nhận được thông báo bắt đầu từ lớp ứng dụng. Ở trạng thái này, datagram được tạo với sequence number "1" trong header của nó và payload dưới dạng một thông báo trong nội dung của nó. Cuối cùng, gói tin được đẩy vào mạng và việc thực thi chuyển sang trạng thái tiếp theo.
- Trạng thái 4 (*Dưới cùng bên trái*): Trạng thái này xác nhận người nhận đã nhận được gói tin hay chưa. kiểm tra trạng thái xem nếu xác nhận nhận được không bị hỏng, có sequence number "0" và đạt được trong thời gian. Nếu hai tiêu chí này được thỏa mãn thì việc thực thi sẽ chuyển sang trạng thái tiếp theo, trạng thái khác thì trạng thái đó sẽ gửi lại gói tin.



- Bên nhận:
  - Trạng thái 1 (*Bên trái*): Đây là Trạng thái đầu tiên trong FSM của người nhận, được gọi là "*wait for call 0 from below*". Trạng thái này cho biết gói nhận được có sequence number "0" và không bị hỏng hay không. Nếu các điều kiện này thỏa mãn trạng thái này tạo ra một gói báo nhận với seq "0" và đẩy nó vào mạng báo hiệu gói chính xác đã được nhận, việc thực thi sẽ chuyển sang trạng thái tiếp theo khác, nó tạo ra một gói báo nhận với seq "1" và đẩy nó vào mạng báo hiệu không nhận được gói tin chính xác.
  - Trạng thái 2 (*Bên phải*): Đây là Trạng thái đầu tiên trong FSM của người nhận, được gọi là "*wait for call 1 from below*". Trạng thái này cho biết gói nhận được có sequence number "1" và không bị hỏng hay không. nếu các điều kiện này thỏa mãn Trạng thái này tạo ra một gói báo nhận với seq "1" và đẩy nó vào mạng báo hiệu gói đúng đã được nhận, việc thực thi chuyển sang trạng thái tiếp theo khác, nó tạo ra một gói báo nhận với seq "0" và đẩy nó vào mạng báo hiệu không nhận được gói tin chính xác



Tóm tắt các phiên bản RDT

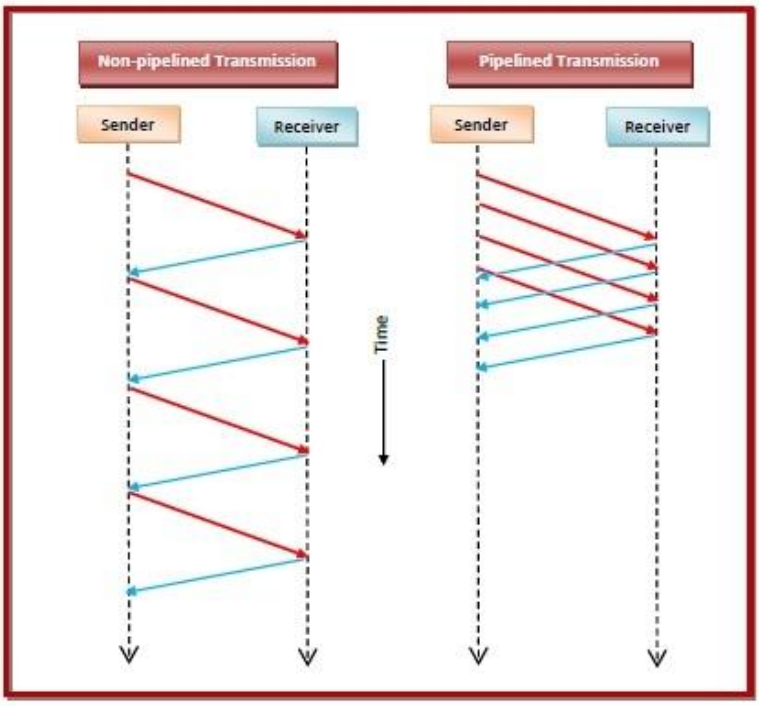
Giả định	Không lỗi bit Không mất gói	Lỗi bit (Không mất gói)	Lỗi ACK,NAK	Lỗi ACK,NAK	Mất gói tin
Phiên bản	1.0	2.0	2.1	2.2	3.0
Cải tiến		Checksum, ACK, NAK	Sequence Number	Bỏ NAK	Timeout

Các giao thức Pipelined

Pipelining: Bên gửi cho phép gửi nhiều gói đồng thời, không cần chờ báo xác nhận ACK

- Dải sequence number phải được tăng lên
- Phải có bộ nhớ đệm tại nơi gửi và/hoặc nhận

Giao thức Sliding Window



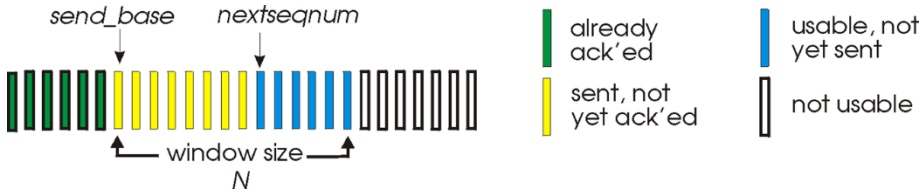
Hai dạng phổ biến của các giao thức pipelined:

- Go-Back-N
- Selective repeat

Go-Back-N

Go Back-N: Bên gửi

- Cho phép gửi nhiều gói tin cùng lúc mà không cần chờ ACK:
  - ACK tích lũy: Nhận được ACK có stt cao nhất là được
  - Định thời gian cho gói tin được gửi
  - Gửi lại tất cả từ gói bị mất

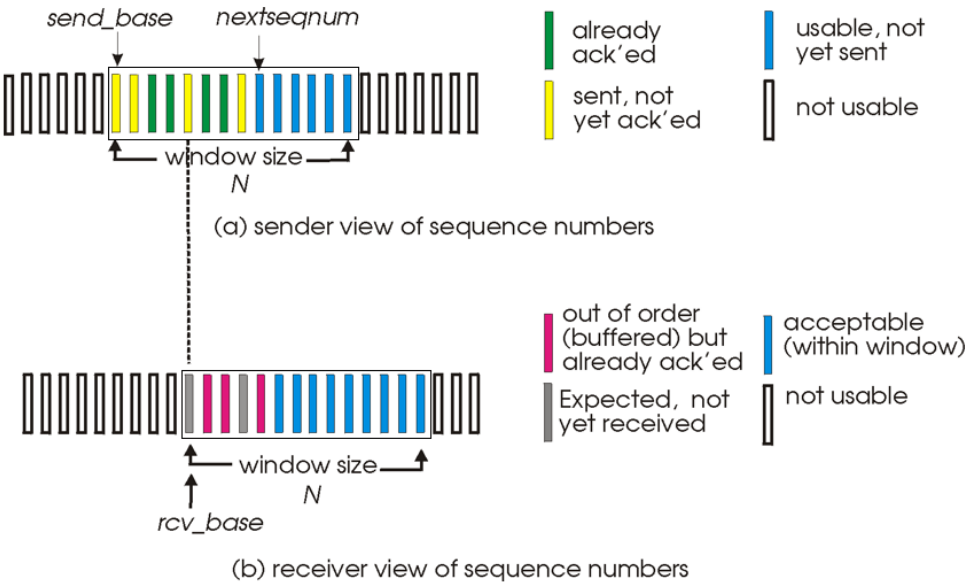


Go Back-N: Bên nhận

- Gửi ACK cho những gói tin được nhận thành công
  - Tăng số ACK khi nhận đúng thứ tự
  - Không tăng số ACK khi phát hiện gói không đúng thứ tự
  - Bỏ các gói không đúng thứ tự

Selective Repeat (Lặp có lựa chọn)

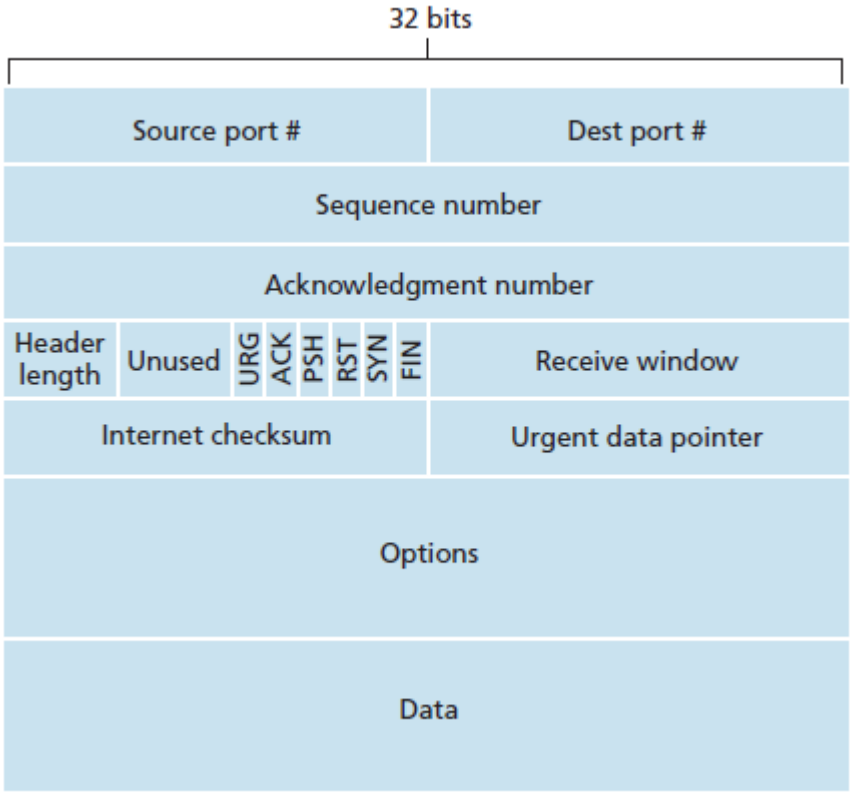
- Gửi ACK riêng biệt cho tất cả các gói tin không bị hỏng (mất,sai thứ tự)
- Lưu các gói tin đã nhận được vào bộ nhớ đệm
- Chỉ gửi lại các gói tin không được ACK(khác biệt lớn so với GBN)
- Định thời gian riêng biệt cho từng gói



3.5 Vận chuyển hướng kết nối TCP

Cấu trúc Segment TCP

- **Source port** và **Destination port** để multiplexing/demultiplexing dữ liệu từ/đến lớp ứng dụng phía trên
- Trường **Sequence number** 32 bit: để đảm bảo dữ liệu được trao cho ứng dụng đích theo đúng thứ tự.
- Trường **ACK**(acknowledgment number) 32 bit: chứa số tiếp theo mà bên nhận mong đợi nhận được..
- Trường **Receive window** nhận 16 bit được sử dụng trong flow control. Receive window TCP là buffer(bộ đệm) ở mỗi bên của kết nối TCP tạm thời giữ dữ liệu đến. Kích thước của cửa sổ (bộ đệm) được đặt trong quá trình bắt tay 3 bước TCP và có thể thay đổi sau đó. Bên gửi chỉ có thể gửi lượng dữ liệu đó trước khi phải đợi thông báo từ phía nhận.



- Như với UDP, Segment TCP bao gồm trường **Internet checksum**.
- Trường **Header length** 4 bit: chỉ định độ dài của header TCP trong 32 bits word. Header TCP có thể có độ dài thay đổi do TCP options field. (Thông thường, options field trống, do đó độ dài của Header TCP điển hình là 20 byte).
- Trường **Options** có độ dài thay đổi và tùy chọn được sử dụng khi người gửi và người nhận thương lượng kích thước phân đoạn tối đa (MSS) hoặc như một hệ số tỷ lệ cửa sổ để sử dụng trong mạng tốc độ cao. Một tùy

chọn đóng dấu thời gian cũng được xác định. Xem RFC 854 và RFC 1323 để biết thêm chi tiết.

- Trường **flag** chứa 6 bit. Bit **ACK** được sử dụng để chỉ ra rằng giá trị được mang trong báo nhận cho một phân đoạn đã được nhận thành công. Các bit **RST**, **SYN** và **FIN** được sử dụng để thiết lập và chia nhỏ kết nối. Việc thiết lập bit **PSH** chỉ ra rằng bên nhận sẽ chuyển dữ liệu lên lớp trên ngay lập tức. Cuối cùng, bit **URG** được sử dụng để chỉ ra rằng có dữ liệu trong phân đoạn này mà thực thể lớp trên bên gửi đã đánh dấu là "khẩn cấp".
- Vị trí của byte cuối cùng của dữ liệu khẩn cấp này được chỉ ra bởi trường **urgent data pointer** 16 bit. TCP phải thông báo cho thực thể lớp trên phía bên nhận khi tồn tại dữ liệu khẩn cấp và chuyển nó đến một con trỏ đến cuối dữ liệu khẩn cấp.
- *Trong thực tế, **PSH**, **URG** và **urgent data pointer** không được sử dụng.*

Truyền dữ liệu tin cậy

- TCP phải khôi phục dữ liệu bị Internet làm hỏng, mất, trùng lặp hoặc gửi không theo yêu cầu. TCP đạt được độ tin cậy này bằng cách gán một sequence number cho mỗi octet mà nó truyền và yêu cầu xác nhận tích cực (ACK) từ TCP nhận. Nếu ACK không được nhận trong khoảng thời gian chờ, dữ liệu sẽ được truyền lại. Giá trị time-out truyền lại TCP được xác định động cho mỗi kết nối, dựa trên round-trip time. Tại bên nhận, các sequence number được sử dụng để sắp xếp chính xác các segment có thể nhận được không theo thứ tự và để loại bỏ các đoạn trùng lặp. Thiệt hại được xử lý bằng cách sử dụng checksum để kiểm tra vào mỗi segment được truyền đi, kiểm tra nó ở bên nhận và loại bỏ các segment bị hỏng

Điều khiển luồng (flow control) kiem soat khong bi tran bo dem cua ben nhan -> mat mat goi tin

- Flow control xử lý lượng dữ liệu được gửi đến phía người nhận mà không nhận được bất kỳ ACK nào. Nó đảm bảo rằng người nhận sẽ không bị quá tải với dữ liệu.
- Là một loại quy trình đồng bộ hóa tốc độ giữa người gửi và người nhận.
- *Ví dụ*: Sinh viên B đang tham gia một buổi training. Giả sử anh ta nắm bắt chậm các khái niệm do trainer trình bày. Mặt khác, người trainer đang trình bày rất nhanh mà không nhận bất kỳ ACK nào từ sinh viên B. Sau một thời gian, mọi lời nói từ trainer tràn ngập đầu B. Do đó, anh ta không hiểu gì cả.

Ở đây, trainer phải có thông tin về số lượng khái niệm mà sinh viên B có thể xử lý cùng một lúc. Sau một thời gian, B yêu cầu trainer giảm tốc độ vì anh ta quá tải với dữ liệu. Người trainer quyết định dạy một số khái niệm trước và sau đó chờ ACK từ sinh viên B trước khi tiếp tục các khái niệm sau.

- **RcvBuffer** = size or TCP Receive Buffer: Kích thước của **RcvBuffer** được thiết đặt thông qua các tùy chọn của socket (thông thường mặc định là 4096 byte). Nhiều hệ điều hành tự động điều chỉnh **RcvBuffer**
- Để kiểm soát lượng dữ liệu được gửi bởi TCP, bên nhận sẽ tạo một buffer còn được gọi là **Receive Window(rwnd)**

Quản lí kết nối

- **Thiết lập kết nối**
  - Để thiết lập kết nối, TCP sử dụng **three-way handshake**. Trước khi máy khách cố gắng kết nối với máy chủ, trước tiên máy chủ phải liên kết và lắng nghe tại một cổng để mở nó cho các kết nối: điều này được gọi là mở thụ động. Sau khi mở bị động được thiết lập, khách hàng có thể bắt đầu mở chủ động. Để thiết lập kết nối, hãy bắt tay ba bước (hoặc 3 bước)
- **Đóng kết nối**
  - Sử dụng **four-way handshake**, với mỗi bên của kết nối chấm dứt độc lập. Khi một điểm cuối muốn dừng một nửa kết nối của nó, nó sẽ truyền một gói **FIN**, mà đầu kia thừa nhận bằng một ACK. Do đó, việc chia nhỏ điển hình yêu cầu một cặp phân đoạn FIN và ACK từ mỗi điểm cuối TCP. Sau khi cả hai trao đổi **FIN/ACK** được kết thúc, bên gửi FIN đầu tiên trước khi nhận một sẽ đợi một khoảng thời gian chờ trước khi cuối cùng đóng kết nối, trong thời gian đó, local port không khả dụng cho các kết nối mới; điều này ngăn ngừa sự nhầm lẫn do các packet bị trễ được gửi trong các kết nối tiếp theo

3.6 Các nguyên lý về điều khiển tắc nghẽn

kiem soat toc do duong truyen qua nhanh -> tac nghen ngay tren duong truyen

- Kiểm soát tắc nghẽn TCP được Van Jacobson đưa vào Internet vào cuối những năm 1980, khoảng tám năm sau khi giao thức TCP/IP đi vào hoạt động. Ngay trước thời điểm này, Internet đã bị sập vì tắc nghẽn — các máy chủ sẽ gửi các gói của họ vào Internet nhanh như cửa sổ được quảng cáo cho phép, tắc nghẽn sẽ xảy ra ở một số router (khiến các packet bị rớt) và

các máy chủ sẽ hết thời gian chờ và truyền lại các gói của chúng, dẫn đến tắc nghẽn nhiều hơn.

- Quá nhiều nguồn gửi quá nhiều dữ liệu với tốc độ quá nhanh vượt quá khả năng xử lý của mạng
- *Khác với flow control*
- Các biểu hiện:
  - Mất gói (tràn bộ đệm tại các router)
  - Độ trễ lớn (xếp hàng trong các bộ đệm của router)
- Phương pháp: end-end, có sự hỗ trợ của mạng (network-assisted),...

3.7 Điều khiển tắc nghẽn TCP

- Bên gửi tăng tốc độ truyền (kích thước window), thăm dò băng thông có thể sử dụng, cho đến khi mất mát gói xảy ra
- **Congestion window (CWND)** là một trong những yếu tố quyết định số lượng byte có thể được gửi đi bất cứ lúc nào
- Một số thuật toán giải quyết Congestion TCP: Tahoe, Reno, New Reno,...
- Giải thích thuật toán TCP Reno qua các giai đoạn
  - **Slow Start**
    - Khi kết nối bắt đầu, tăng tốc độ theo cấp số nhân cho đến sự kiện mất gói đầu tiên xảy ra. Tốc độ ban đầu chậm, nhưng nó sẽ tăng lên theo cấp số nhân
    - Cách nhận biết: cwnd < ssthresh
    - Tham số:
      - $cwnd_{next} = cwnd_{prev} * 2$
      - ssthresh giữ nguyên
  - **Congestion avoidance(CA)**
    - Cách nhận biết: cwnd >= ssthresh
    - Tham số:
      - $cwnd_{next} = cwnd_{prev} + 1MSS$ (đơn vị dữ liệu gửi trong thời gian)
      - ssthresh giữ nguyên
  - **Fast Recovery**
    - TCP Reno có cài đặt thêm thuật toán "Fast-Retransmit" khi gặp trường hợp có 3 gói ACK bị lặp lại. Chuyển nhanh lại gói tin đã bị mất (Fast-Retransmit) và bước vào một pha gọi là Fast Recovery.

- Cách nhận biết: 3 ACK trùng
- Tham số:
  - cwnd = $\frac{1}{2}$ $cwnd_{prev}$ + 3 (do có 3 gói tin phản hồi ACK trùng)
  - ssthresh = $\frac{1}{2}cwnd_{prev}$

Chương 4: Tầng Network

4.1 Giới thiệu

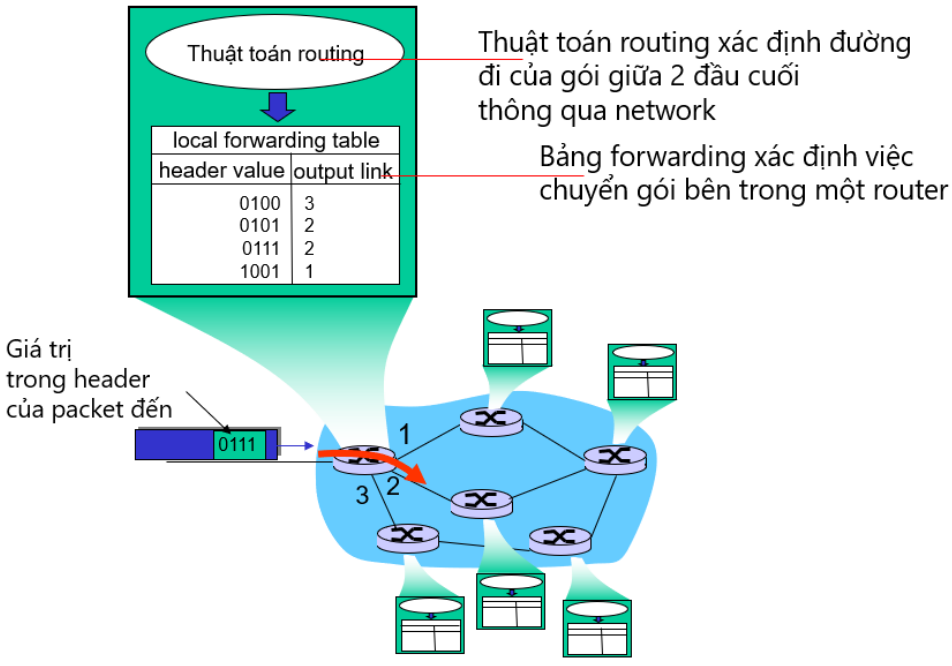
4.1.1 Giao thức và dịch vụ tầng Network

- Đảm bảo việc truyền thông logic giữa các host
  - Bên gửi: Đóng gói các segment vào trong các Datagram
  - Bên nhận: Mở gói Datagram thành các segment và gửi lên tầng Transport
- Hoạt động ở mọi host, router
- Routers
  - Xem xét các trường của header trong tất cả các gói IP datagram đi qua nó
  - Chuyển packet từ cổng vào đến cổng ra phù hợp

4.1.2 Chức năng chính của tầng Network

- Forwarding: Chuyển các gói tin từ đầu vào đến đầu ra thích hợp của Router
- Routing: Xác định đường đi cho các gói từ nguồn đến đích

Tác động qua lại giữa Routing và Forwarding



4.1.3 Thiết lập kết nối

- Sử dụng trong một số kiến trúc mạng: ATM, frame relay, X.25
- Dịch vụ kết nối tầng Transport so với tầng Network
    - Network: Giữa 2 hosts (cũng có thể bao gồm các router trung gian trong trường hợp kết nối ảo)
    - Transport: Giữa 2 tiến trình

4.1.4 Mô hình dịch vụ mạng

Kiến trúc mạng	Mô hình dịch vụ	Đảm bảo băng thông	Đảm bảo không mất dữ liệu	Thứ tự	Đảm bảo thời gian	Báo hiệu nghẽn
Internet	Best effort	Không	Không	Bất kỳ thứ tự nào	Không	Không
ATM	CBR	Tốc độ luôn được giữ vững	Có	Theo thứ tự	Có	Không xảy ra nghẽn
ATM	ABR	Đảm bảo tốc độ thấp nhất	Không	Theo thứ tự	Không	Báo hiệu khi xảy ra nghẽn

4.2 Mạng mạch ảo (virtual circuit) và mạng chuyển gói (datagram)

4.2.1 Dịch vụ connection (hướng kết nối) và connection-less (phi kết nối)

- Datagram network cung cấp dịch vụ connection-less tại tầng network
- Virtual-circuit network cung cấp dịch vụ connection tại tầng network

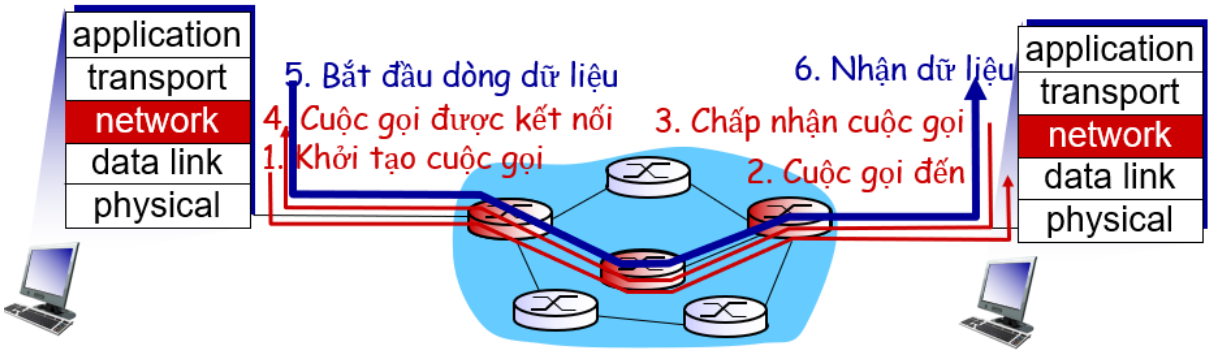
4.2.2 Các mạng mạch ảo (virtual-circuits network)

Thiết lập kết nối ảo. Một kết nối ảo bao gồm:

- Đường đi (path) từ nguồn tới đích
- Số hiệu kết nối ảo (VC numbers), một số cho một kết nối dọc theo đường đi
- Các mục trong các bảng forwarding ở trong các router dọc theo đường đi

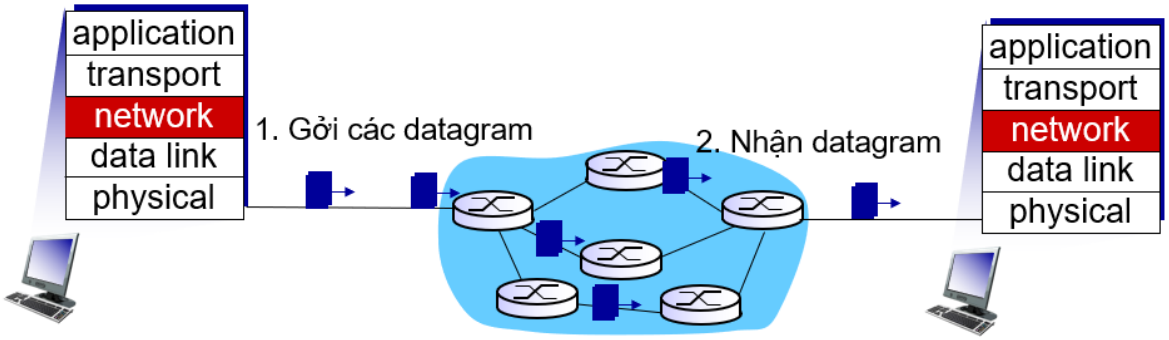
Các giao thức gửi tín hiệu:

- Được dùng để thiết lập, duy trì kết nối ảo
- Được dùng trong ATM, frame-relay, X.25
- Không được sử dụng trong Internet ngày nay



4.2.3 Mạng chuyển gói (datagram network)

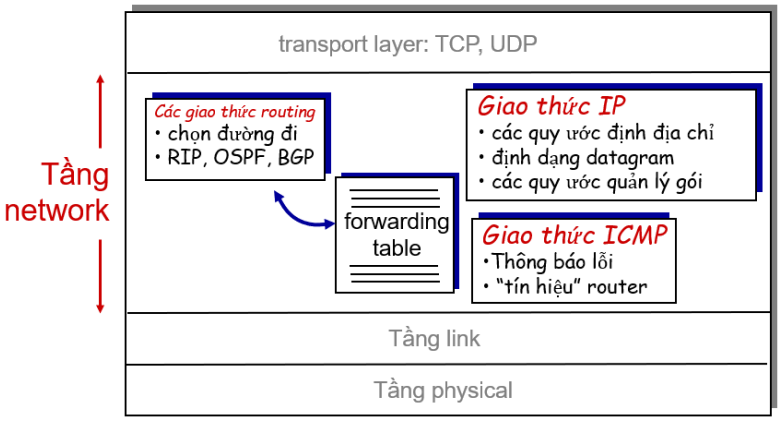
- Không thiết lập cuộc gọi tại tầng Network
- Mỗi một lần bên gửi muốn gửi một gói tin đi, bên gửi sẽ thêm vào gói tin địa chỉ nguồn gửi và địa chỉ đích sau đó sẽ đẩy gói tin đi trên mạng (không duy trì trạng thái mạng ảo nào)
- Mỗi gói tin đều có địa chỉ nguồn và đích, bộ định tuyến nhờ đó mà chuyển gói tin đi



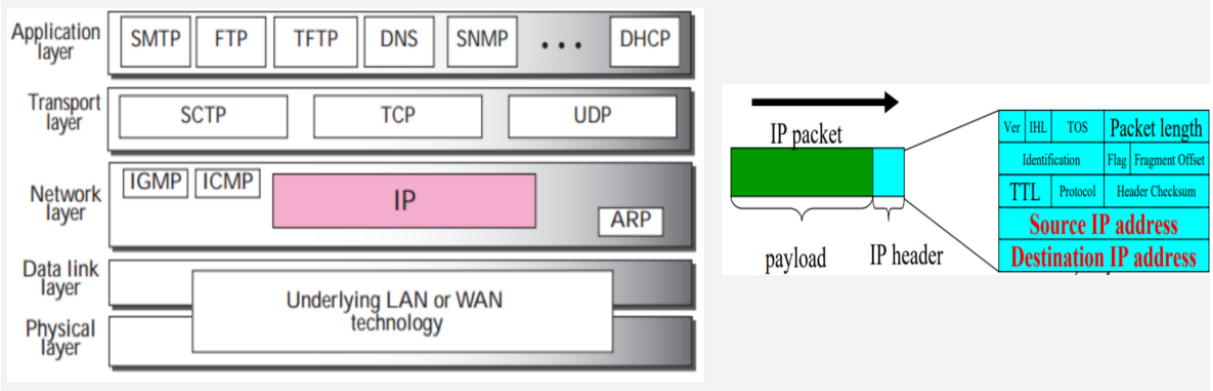
4.3 IP: Internet Protocol

4.3.1 Tầng Internet network

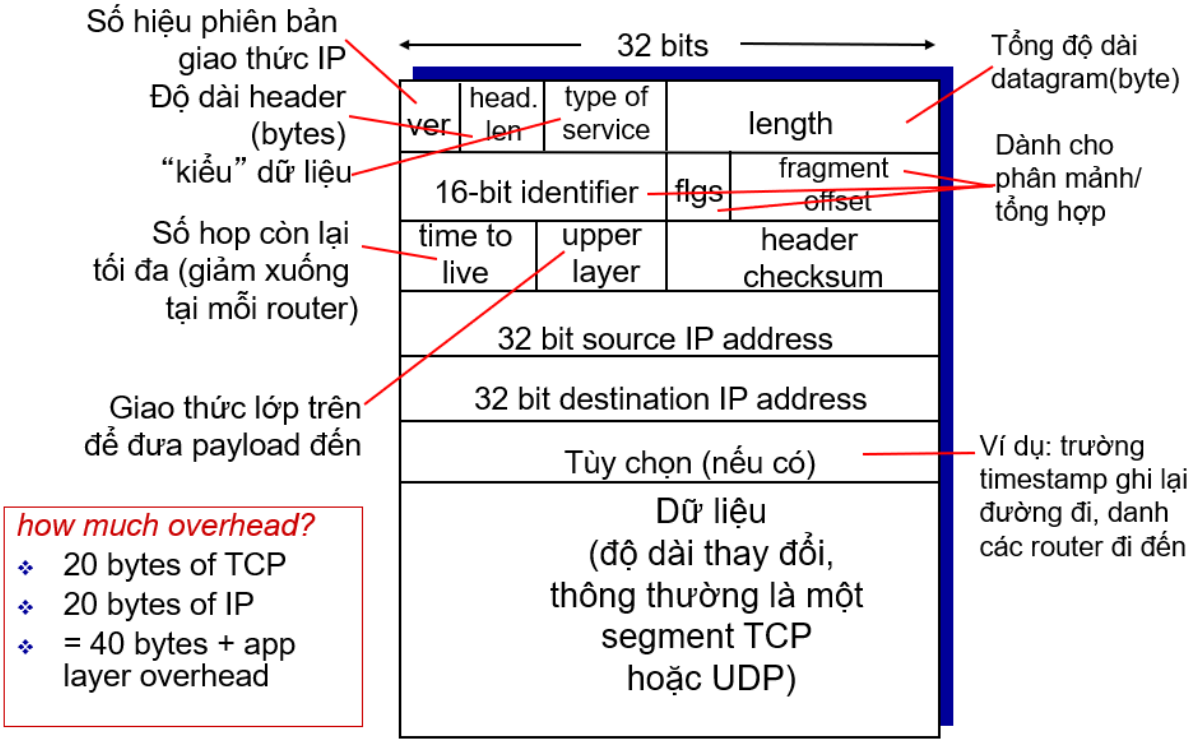
- Các chức năng tầng network của host và router:



- Vị trí của IP:



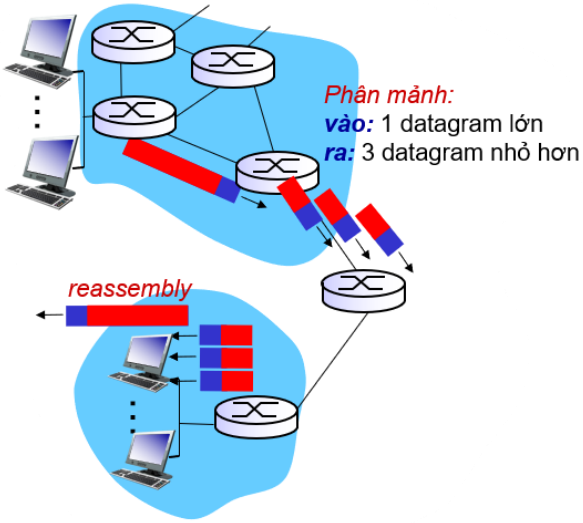
4.3.2 Định dạng IP Datagram



4.3.3 Phân mảnh và tổng hợp IP

- Các đoạn kết nối mạng có MTU (Max Transfer Size)-frame lớn nhất có thể truyền trên kết nối
  - Các kiểu kết nối khác nhau có các MTU khác nhau
- Các gói IP datagram lớn được chia ("fragmented") bên trong mạng
  - 1 datagram thành một vài datagram

- "Tổng hợp" chỉ được thực hiện ở đích cuối cùng
- Các bit của IP header được sử dụng để xác định, xếp thứ tự các fragment liên quan

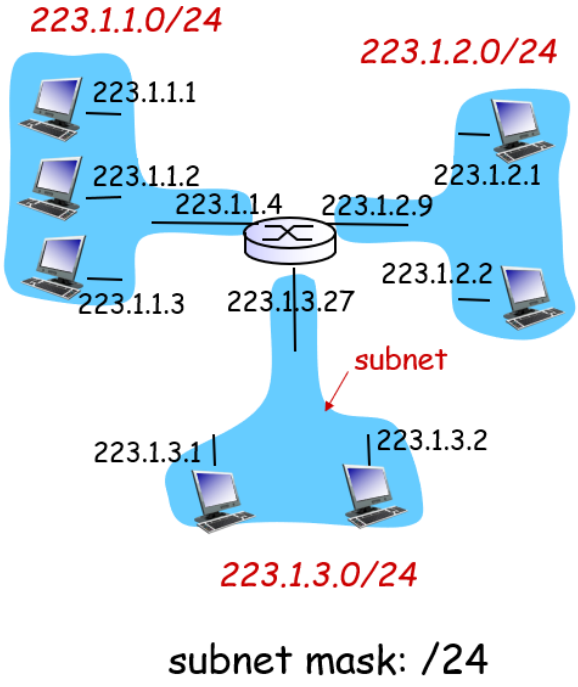


4.3.4 IPv4 addressing

- Địa chỉ IP: 32-bit nhận dạng cho host, router interface
- Interface: Kết nối giữa host/router và đường kết nối vật lý
  - Router thường có nhiều interface
  - Host thường có 1 hoặc 2 interface
- Mỗi địa chỉ IP được liên kết với mỗi interface

4.3.5 Các subnet (mạng con)

- Địa chỉ IP
  - Phần subnet: Các bit có trọng số cao
  - Phần host: Các bit có trọng số thấp
- Subnet là gì ?
  - Các interface của thiết bị có phần subnet của địa chỉ IP giống nhau
  - Có thể giao tiếp vật lý với nhau mà không cần router trung gian can thiệp
- Phương pháp
  - Để xác định các subnet, tách mỗi interface từ host hoặc router của nó, tạo vùng các mạng độc lập
  - Mỗi mạng độc lập được gọi là một subnet



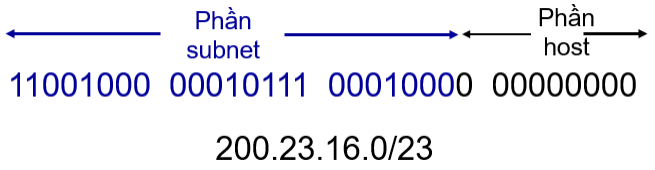
subnet mask: /24

4.3.6 Phân lớp địa chỉ IPv4

Lớp	Octet thứ 1 hệ thập phân	Network/Host (N = Network, H = Host)	Subnet Mask mặc định	Số mạng	Số mạng mỗi mạng
A	1 - 126	N.H.H.H	255.0.0.0	126 ($2^7 - 2$)	16,777,214 ($2^{24} - 2$)
B	128 - 191	N.N.H.H	255.255.0.0	16,382 ($2^{14} - 2$)	65,534 ($2^{16} - 2$)
C	192 - 223	N.N.N.H	255.255.255.0	2,097,150 ($2^{21} - 2$)	254 ($2^8 - 2$)
D	224 - 239	Multicast			
E	240 - 254	Nghiên cứu (Không sử dụng)			

4.3.7 Định địa chỉ IP: CIDR(Classless InterDomain Routing)

- Phần subnet: Độ dài bất kỳ
- Định dạng địa chỉ: a.b.c.d/x, trong đó x là số các bits trong phần subnet của địa chỉ



4.3.8 Phân loại địa chỉ IP

Phân loại theo phạm vi hoạt động

- Private IP: Sử dụng trong mạng LAN, có thể sử dụng lặp lại ở các mạng LAN khác nhau
  - Lớp A: từ 10.0.0.0 -> 10.255.255.255
  - Lớp B: từ 172.16.0.0 -> 172.31.255.255
  - Lớp C: từ 192.168.0.0 -> 192.168.255.255
- Public IP: Sử dụng trong mạng WAN, dùng để định tuyến trên Internet, và là duy nhất cho mỗi host tham gia vào Internet
- Loopback IP
  - Dải địa chỉ: 127.0.0.1 -> 127.255.255.254

Phân loại trong quá trình truyền thông

- Địa chỉ mạng (network): Tất cả bit HostID = 0
- Địa chỉ quảng bá (broadcast): Tất cả bit HostID = 1
- Địa chỉ dùng cho host: Trường hợp còn lại

4.3.9 Chia mạng con

Thực hiện 3 bước:

- Bước 1: Xác định class và subnet mask mặc nhiên của địa chỉ
- Bước 2: Xác định số bit cần mượn và subnet mask mới, tính số lượng mạng con, số host thực sự có được
- Bước 3: Xác định các vùng địa chỉ host và chọn mạng con muốn dùng

Subnet mask: Tất cả bit HostID = 0, các phần còn lại = 1

Ví dụ: Cho địa chỉ IP sau: 172.16.0.0/16. Hãy chia thành 8 mạng con và có 1000 host trên mỗi mạng con

Giải:

Bước 1: Xác định class và subnet mask mặc nhiên

$$172.16.0.0_{10} = 10101100.00010000.00000000.00000000_2$$

Octect thứ 1: 172 => Lớp B

Subnet mask mặc nhiên: 16 bit => 225.225.0.0

Bước 2: Xác định số bit cần mượn …

Số bit cần mượn:

N = 3 vì:

- Số mạng con có thể $2^3 = 8$
- Số host của mỗi mạng con có thể: $2^{16-3} - 2 > 1000$

Subnet mask mới: 11111111.11111111.11100000.00000000 (hay 255.255.224.0)

Bước 3: Xác định vùng địa chỉ host

STT	SubnetID	Vùng HostID	Broadcast
1	172.16.0.0	172.16.0.1 - 172.16.31.254	172.16.31.255
2	172.16.32.0	172.16.32.1 - 172.16.63.254	172.16.63.255
…	…	…	…
7	172.16.192.0	172.16.192.1 – 172.16.223.254	172.16.223.255
8	172.16.224.0	172.16.224.1 – 172.16.255.254	172.16.255.255

4.3.10 DHCP (Dynamic Host Configuration Protocol)

Để 1 host lấy được địa chỉ IP:

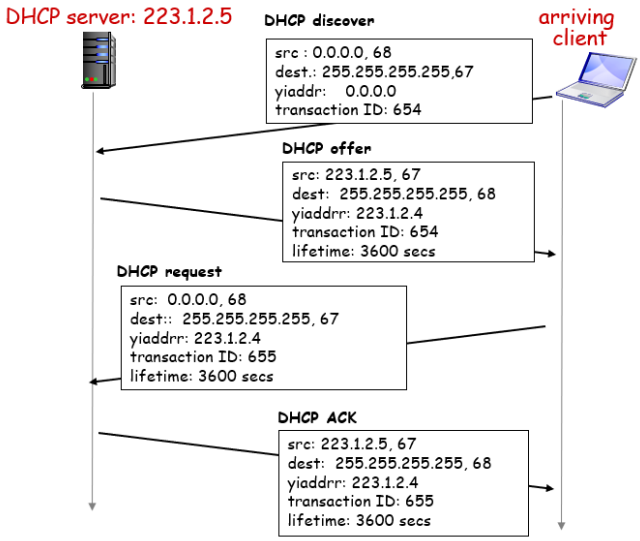
- Người quản trị hệ thống lưu địa chỉ trong cấu hình hệ thống
- DHCP: tự động lấy địa chỉ IP từ server
    - Plug and Play

Mục tiêu:

- Cho phép host (máy) tự động lấy địa chỉ IP của nó từ server trong mạng khi host đó tham gia vào mạng
- Có thể gia hạn địa chỉ IP mà host đó vừa được cấp
- Cho phép tái sử dụng các địa chỉ IP (chỉ giữ địa chỉ trong khi được kết nối)
- Hỗ trợ cho người dùng di động muốn tham gia vào mạng(trong thời gian ngắn)

Tổng quan về DHCP:

- Host quảng bá (broadcasts) thông điệp "DHCP discover" [tùy chọn]
- DHCP server đáp ứng bằng thông điệp "DHCP offer"[tùy chọn]
- Host yêu cầu địa chỉ IP: "DHCP request" msg
- DHCP server gởi địa chỉ: "DHCP ack" msg



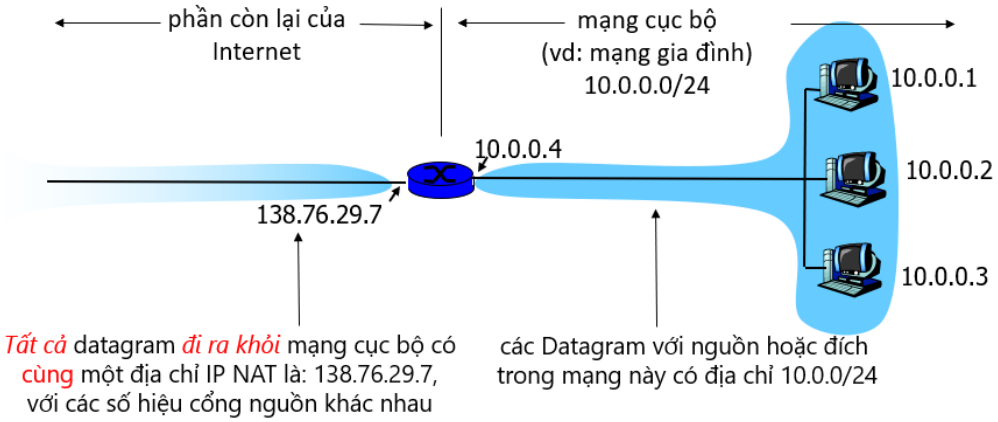
Ngữ cảnh DHCP client-server

DHCP cung cấp nhiều thông tin: DHCP không chỉ trả về địa chỉ IP, mà nó có thể trả về nhiều thông tin như sau:

- Địa chỉ của router ở cửa ngõ kết nối ra ngoài mạng của client (default gateway)
- Tên và địa chỉ IP của DNS server
- Network mask (cho biết phần của mạng và phần host của địa chỉ IP)

4.3.11 NAT (Network Address Translation)

- Được thiết kế để tiết kiệm địa chỉ IP
- Cho phép mạng nội bộ sử dụng địa chỉ IP riêng
- Địa chỉ IP riêng sẽ được chuyển đổi sang địa chỉ công cộng định tuyến được
- Mạng riêng được tách biệt và giấu kín IP nội bộ
- Thường sử dụng trên router biên của mạng một cửa



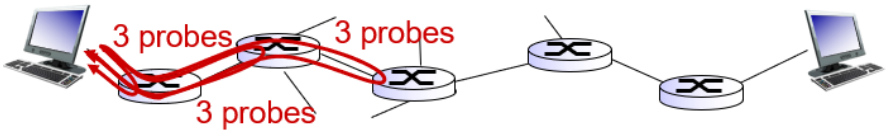
4.3.12 ICMP (Internet Control Message Protocol)

- Được sử dụng bởi các host và router để truyền thông tin tầng Mạng
- Tầng Mạng "trên" IP
  - Các thông điệp ICMP được gửi trong các IP datagram

Loại	mã	Mô tả
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

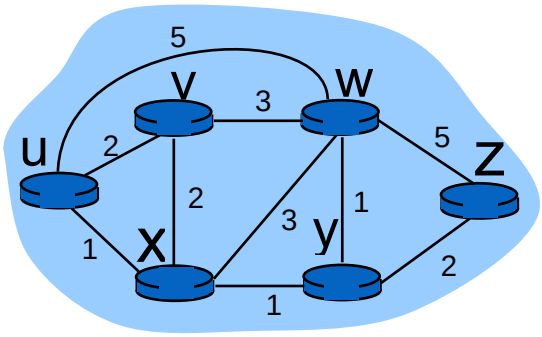
Traceroute và ICMP

- Nguồn gửi một chuỗi các segment UDP đến đích
- Khi datagram thứ n đến router thứ n:
  - Router hủy datagram
  - Và gửi đến nguồn một thông điệp ICMP (loại 11, mã 0)
  - Thông điệp ICMP bao gồm tên và địa chỉ IP của router



4.4 Các thuật toán Routing

4.4.1 Mô hình đồ thị: chi phí



Đồ thị G = (N,E)

- N = Tập hợp các router = {u,v,w,x,y,z}
- E = Tập hợp các kết nối = {(u,v), (u,x),...}

Chi phí đường đi $(x_1, x_2, x_3,..., x_p) = c(x_1,x_2) + c(x_2,x_3) + ... + c(x_{p-1},x_p)$

4.4.2 Phân loại thuật toán Routing

Toàn cục	Phân cấp
- Tất cả các router có toàn bộ thông tin về chi phí kết nối, cấu trúc mạng ⇨ Thuật toán "link state"	- Router biết các router được kết nối vật lý trực tiếp với nó (neighbor), và chi phí kết nối đến neighbor đó - Lặp lại qua quá trình tính toán, trao đổi thông tin với các neighbor ⇨ Thuật toán "distance vector"

Tĩnh	Động
- Các đường đi được cập nhật chậm theo thời gian	- Các đường đi thay đổi nhanh ○ Cập nhật theo chu kỳ ○ Cập nhật khi có những thay đổi về chi phí kết nối

4.4.3 Link State:

Thuật toán Dijkstra

- Biết chi phí kết nối, cấu trúc mạng của tất cả các node
  - Được thực hiện thông qua "link state broadcast"
  - Tất cả các nodes có cùng thông tin với nhau
- Tính toán đường đi có chi phí thấp nhất từ một node("nguồn") đến tất cả các node khác
  - Cho trước bảng forwarding của node đó

- Lặp lại: sau k lần lặp lại, biết được đường đi có chi phí thấp nhất của k đích

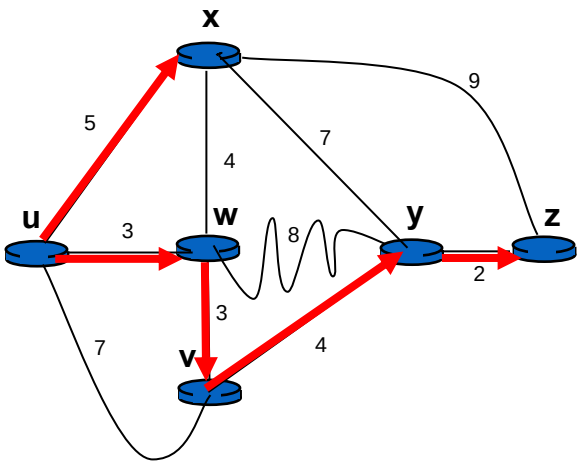
```
1  Khởi tạo:
2    N' = {u}
3    for all nodes v
4      if v adjacent to u
5          then D(v) = c(u,v)
6      else D(v) = ∞
7
8  Lặp
9     find w not in N' such that D(w) is a minimum
10    add w to N'
11    update D(v) for all v adjacent to w and not in N' :
12       D(v) = min( D(v), D(w) + c(w,v) )
13/* chi phí mới đến v là chính nó hoặc chi phí đường đi ngắn nhất
14cộng với chi phí từ w đến v*/
15  until all nodes in N'
```

Ký hiệu:

- c(x,y): chi phí kết nối từ node x đến y; = ∞ nếu không kết nối trực tiếp đến neighbor
- D(v): giá trị chi phí hiện tại của đường đi từ nguồn tới đích v
- p(v): node trước nằm trên đường đi từ nguồn tới v
- N': tập các node mà chi phí đường đi thấp nhất đã được xác định

Ví dụ:

Bước	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



4.4.4 Thuật toán Distance vector(DV)

Ý tưởng:

- Mỗi node định kỳ gởi ước lượng distance vector của nó cho các neighbor
- Sau khi nhận DV mới, nó cập nhật lại DV cũ dùng công thức Bellman-Ford:

$$d_x(y) = min\{c(x,v) + d_v(y)\}$$

Với:

- $d_x(y)$: Chi phí đường đi nhỏ nhất từ x đến y
- $c(x,v)$: Chi phí từ x đến v (lân cận của x)
- $d_v(y)$: Chi phí đường đi nhỏ nhất từ v đến y(lân cận của v)

- Mỗi node thông báo đến các neighbor chỉ khi DV của nó thay đổi
- Các node lặp đi lặp lại quá trình này cho đến khi không còn thông tin trao đổi giữa các neighbor

4.5 Routing trong Internet

4.5.1 Định tuyến Intra-AS

Còn gọi là interior gateway protocols(IGP)

Các giao thức định tuyến intra-AS phổ biến:

- RIP: Routing Information Protocol
- OSPF: Open Shortest Path First
- IGRP: Interior Gateway Routing Protocol(độc quyền của Cisco)

4.5.2 RIP

- Công bố vào năm 1982 trong BSD-UNIX
- Thuật toán Distance Vector
    - Metric khoảng cách: số lượng hop (max = 15 hops), mỗi link có giá trị là 1
    - Các DV được trao đổi giữa các neighbors mỗi 30 giây trong thông điệp phản hồi (còn gọi là advertisement)
    - Mỗi advertisement: danh sách lên đến 25 subnet đích

4.5.3 OSPF

- "Open": Công khai cho mọi đối tượng sử dụng
- Dùng thuật toán Link State
    - Phổ biến packet LS
    - Bản đồ cấu trúc mạng tại mỗi node
    - Tính toán đường đi dùng thuật toán Dijkstra
- Thông điệp quảng bá OSPF chứa 1 mục thông tin cho mỗi router lân cận
- Các thông điệp này được phát tán đến toàn bộ AS
- Giao thức định tuyến IS-IS: gần giống với OSPF

Chương 5: Tầng link

5.1 Một vài thuật ngữ

- Node: Là hosts và routers
- Links: Là các kênh truyền thông kết nối các nút kề nhau theo đường truyền
  - Wired links: Kết nối có dây
  - Wireless links: Kết nối không dây
  - LANs
- Frame: Là các gói tin ở tầng liên kết – chứa datagram

5.2 Các dịch vụ của tầng liên kết

- Đóng gói, truy cập kênh truyền
  - Đóng gói datagram vào các frame, bổ sung header, trailer
  - Truy cập kênh truyền nếu môi trường truyền dẫn là dùng chung
  - Địa chỉ MAC được sử dụng trong các frame header để xác nguồn gửi, đích nhận
    - Khác với địa chỉ IP
- Truyền tin cậy giữa các node lân cận(adjacent nodes)
  - Đã đề cập trong chương 3 (giao thức rdt)
  - Ít khi được sử dụng trên các đường truyền ít lỗi bit (cáp quang, cáp xoắn đôi)
  - Kết nối không dây: Tỷ lệ lỗi cao
- Điều khiển luồng(flow control): Điều khiển tốc độ truyền giữa các node gửi và nhận liền kề nhau
- Phát hiện lỗi(error detection):
- Sửa lỗi(error correction)
- Half-duplex và full-duplex
  - Với half-duplex, các nút ở đầu liên kết có thể truyền cho nhau, nhưng không thể cùng lúc

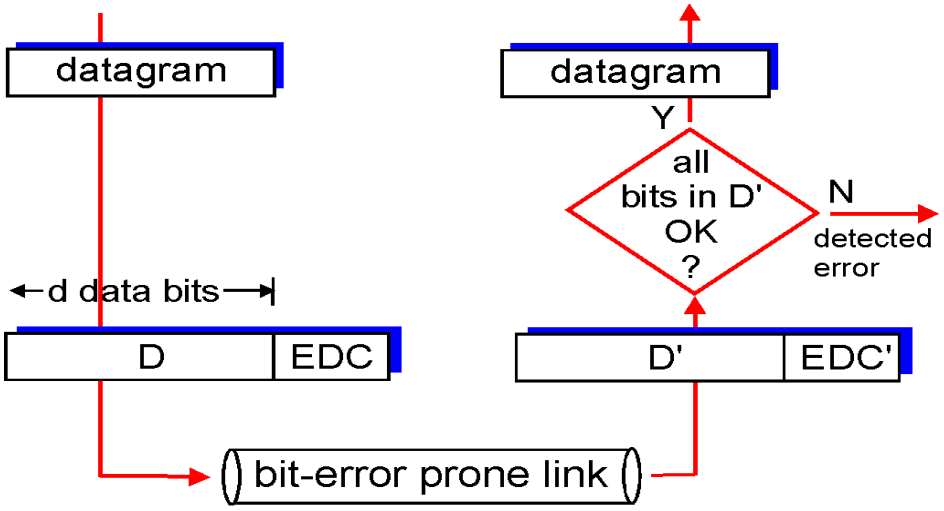
5.3 Phát hiện và sửa lỗi

EDC: Error Detection and Correction bits (redundancy)

D = Data protected by error checking, many include header fields

Chức năng phát hiện lỗi không hoàn toàn đáng tin cậy 100%

- Dù rất hiếm, nhưng giao thức vẫn có thể sót lỗi
- Field EDC càng lớn thì càng dễ phát hiện và sửa lỗi hơn



Kiểm tra chẵn lẻ (parity checking)

Là kỹ thuật dùng thêm một số bit để đánh dấu tính chẵn lẻ

- Parity 1 chiều: Thể hiện dưới dạng 1 ma trận có kích thước 1xM
- Parity 2 chiều: Thể hiện dưới dạng 1 ma trận có kích thước MxN

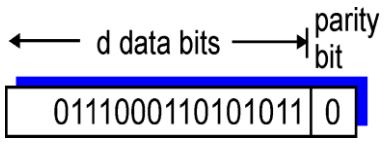
Parity bit 1 chiều

- Số bit parity: 1 bit
- Có 2 mô hình cho parity bit:
  - Mô hình chẵn: Số bit 1 trong dữ liệu gửi đi là một số chẵn
  - Mô hình lẻ: Số bit 1 trong dữ liệu gửi đi là một số lẻ

Bên gửi	Bên nhận
1. Xác định mô hình parity 2. Thêm 1 bit parity vào dữ liệu cần gửi đi 3. Tiến hành gửi dữ liệu đi	1. Nhận D' có (d+1) bits 2. Đếm số bits 1 trong D' 3. Đưa ra kết luận cho dữ liệu trên

Ví dụ: Parity bit 1 chiều:

- Bên gửi: d = 0111000110101011 theo mô hình parity lẻ
- Bên nhận: Nhận D' = 01110001101010110 => accept

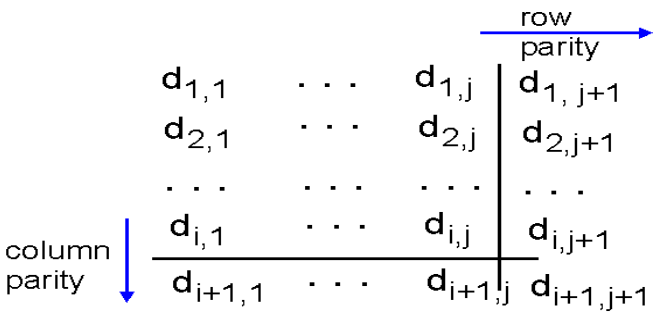


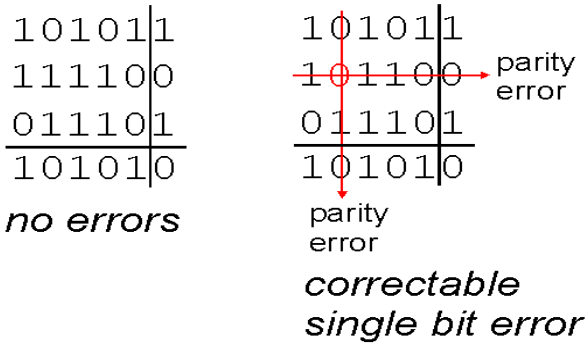
Parity bit 2 chiều

- Số bit parity: (N+M+1)
- Có hai mô hình cho parity bit
  - Mô hình chẵn: Số bit 1 trong dữ liệu gửi đi là một số chẵn
  - Mô hình lẻ: Số bit 1 tron dữ liệu gửi đi là một số lẻ

Bên gửi	Bên nhận
1. Xác định mô hình parity 2. Biểu diễn dữ liệu cần gửi thành ma trận MxN 3. Tính giá trị bit parity trên từng dòng, từng cột. Sau đó gửi dữ liệu	1. Biểu diễn dữ liệu nhận được thành ma trận (M+1)(N+1) 2. Kiểm tra tính đúng đắn của từng dòng và cột. Và đánh dấu các dòng,cột bị lỗi 3. Đưa ra kết luận cho dữ liệu trên

Ví dụ:





Internet checksum

Mục tiêu: Phát hiện "các lỗi" (ví dụ, các bit bị đảo) trong packet được truyền

- *Chỉ được dùng tại tầng vận chuyển*

Bên gửi	Bên nhận
- Xử lý các nội dung của segment như một chuỗi các số nguyên 16-bit **- Checksum: thêm(tổng bù 1) vào các nội dung của segment** - Bên gửi đặt các giá trị checksum vào trong trường checksum của UDP	- Tính toán checksum của segment vừa nhận - Kiểm tra xem giá trị của checksum vừa được tính có bằng với giá trị của trường checksum: + Không: phát hiện lỗi + Có: Không có lỗi phát hiện. Nhưng có thể còn có lỗi khác không ?

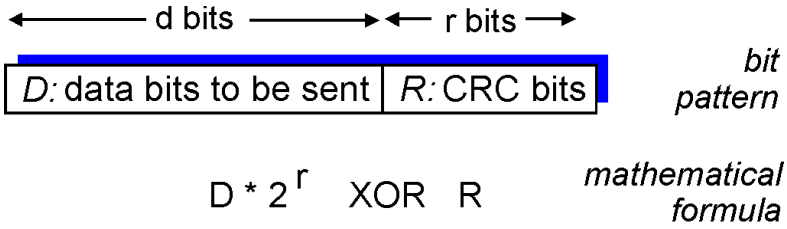
CRC (Cyclic Redundancy Check)

Khái niệm: Là một phương pháp để phát hiện lỗi bit bằng cách gắn thêm 1 khối bit phía sau khối dữ liệu

Khối bit CRC: Thể hiện các bit bổ sung vào sau khối dữ liệu

Mã CRC: Thể hiện phần dư của phép chia nhị phân không nhớ

Mô hình:



Bên gửi	Bên nhận
1. Chọn mẫu dài r+1 làm bộ sinh (hay còn gọi là G). Thống nhất với bên nhận 2. Chọn r bit CRC, gọi là R sao cho cụm <D,R> chia G dư 0 theo module 2 3. Tính giá trị dữ liệu cần gửi và gửi dữ liệu đã tính toán đi	1. Tiếp nhận dữ liệu 2. Tiến hành lấy dữ liệu đã nhận và thực hiện phép chia cho G 3. Dựa vào kết quả ở bước 2, đưa ra kết luận cho dữ liệu trên

5.4 Giao thức đa truy cập

Xét môi trường truyền quảng bá/dùng chung:

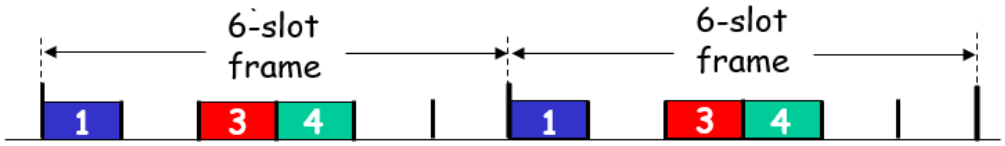
- Hai hay nhiều cuộc truyền đồng thời từ các nút: tín hiệu từ các frame bị trộn lẫn, khôn thể tách rời => Ta nói hiện tượng va chạm [collision]
- Vấn đề quan trọng ở tầng link: Làm thế nào để điều khiển việc nhiều nút cùng gửi và cùng nhận trên một môi trường truyền chung (multiple access problem)
  - Cần phải có quy tắc cho việc này, chính là giao thức đa truy cập (multiple access protocol)

Phân loại các giao thức MAC:

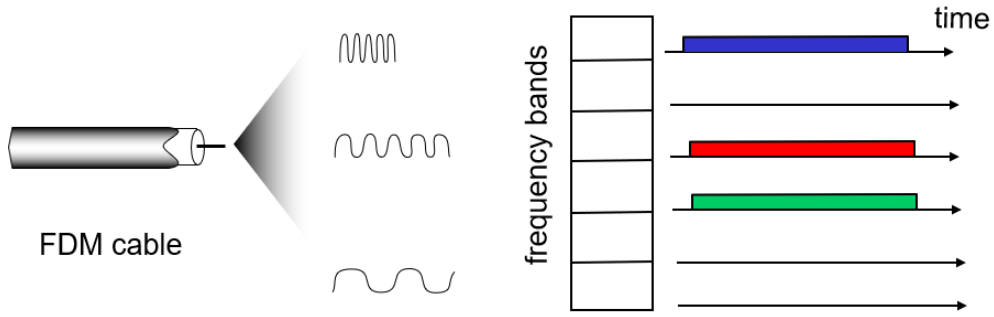
- Phân loại theo kênh truyền
  - Chia kênh truyền ra thành những mảnh nhỏ hơn (time slots, frequency, code)
  - Cấp phát mảnh này cho node để sử dụng độc quyền
- Truy cập ngẫu nhiên
  - Kênh truyền không được chia, cho phép đụng độ
  - "Phục hồi" đụng độ
- "Xoay vòng"
  - Các node thay phiên nhau, nhưng các node có quyền nhiều hơn có thể giữ phiên truyền lâu hơn

Phân loại theo kênh truyền:

- TDMA(Time division multiple access)
  - Truy cập đến kênh truyền theo hình thức "xoay vòng"
  - Mỗi trạm(station) có slot với độ dài cố định(độ dài = thời gian truyền packet) trong mỗi vòng (round)
  - Các slot khôn sử dụng sẽ nhàn rỗi
  - Ví dụ: LAN có 6 trạm, 1,3,4 có gói được gửi, các slot 2,5,6 sẽ nhàn rỗi



- FDMA(Frequency division multiple access)
  - Phổ kênh truyền được chia thành các dải tần số
  - Mỗi trạm được gán một dải tần số cố định
  - Thời gian truyền không được sử dụng trong dải tần số sẽ nhàn rỗi
  - Ví dụ: LAN có 6 trạm, 1,3,4 có packet truyền, các dải tần số 2,5,6 nhàn rỗi



Các giao thức truy cập ngẫu nhiên

- Khi node có packet cần gửi
  - Truyền dữ liệu với trọn tốc độ của kênh dữ liệu R
  - Không có sự ưu tiên giữa các node
- Hai hoặc nhiều node truyền => "đụng độ"
- Giao thức truy cập ngẫu nhiên MAC xác định
  - Cách để phát hiện đụng độ
  - Cách để giải quyết đụng độ (vd: truyền lại sau đó)
  - Một vài giao thức MAC: Slotted ALOHA, Pure ALOHA, CSMA,CSMA/CD,CSMA/CA

Slotted ALOHA

Gia thuyết

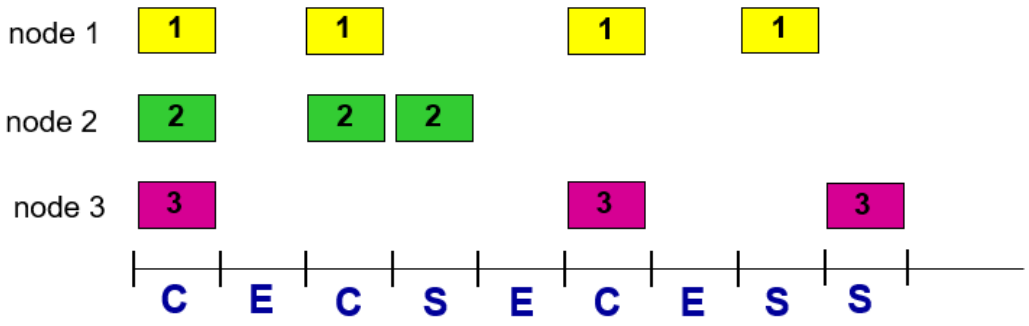
- Tất cả các frame có cùng kích thước
- Thời gian được chia thành các slot có kích thước bằng nhau(thời gian để truyền một frame)
- Các node bắt đầu truyền chỉ ngay tại lúc bắt đầu slot
- Các node được đồng bộ hóa
- Nếu 2 hoặc nhiều node truyền trong slot, thì tất cả các node đều phát hiện đụng độ

Hoạt động: Khi node có được frame mới, nó sẽ truyền trong slot kế tiếp

- Nếu không có đụng độ: node có thể gửi frame mới trong slot kế tiếp
- Nếu có đụng độ: node truyền lại frame trong mỗi slot tiếp theo với xác suất p cho đến khi thành công

Mô hình giả định:

- C: collision slot
- E: empty slot
- S: successful slot



Ưu điểm:

- Node đơn kích hoạt có thể truyền liên tục với tốc độ tối đa của kênh
- Phân cấp cao: chỉ có các slot trong các node cần được đồng bộ
- Đơn giản

Nhược điểm:

- Đụng độ, lãng phí slot
- Các slot nhàn rỗi
- Node phải phát hiện ra hiện tượng va chạm (nếu có) với thời giạn ngắn hơn thời gian truyền
- Đồng bộ hóa

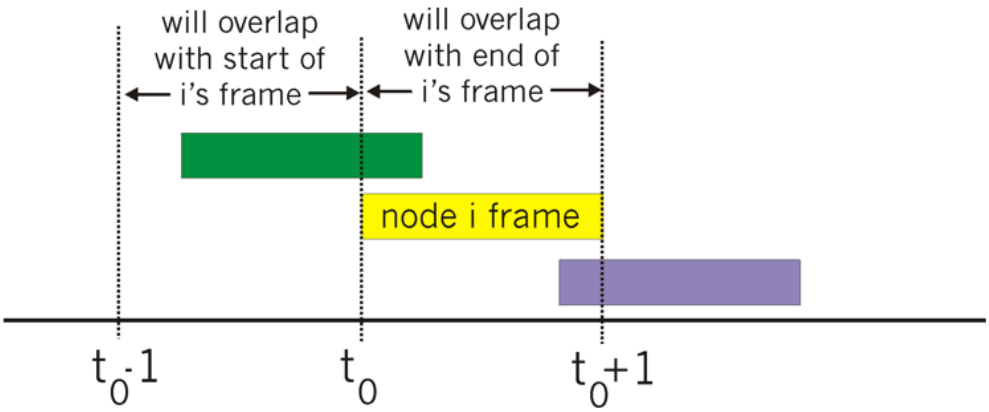
Hiệu suất

- Độ hiệu quả: % slot thành công so với tổng số slot, khi có nhiều node cùng truyền một thời gian dài, mỗi node có một số lượng lớn các frame cần truyền
- Hiệu suất cực đại: 1/e, xấp xỉ 37%

Pure(unslotted) ALOHA

Mô hình giả định

- Frame được truyền tại thời điểm t0 va chạm với các frame được gửi trong khoảng thời gian [t0-1,t0+1]



Ưu điểm:

- Đơn giản hơn Slotted ALOHA
- Không cần đồng bộ hóa
- Truyền đi ngay lần đầu các frame đến

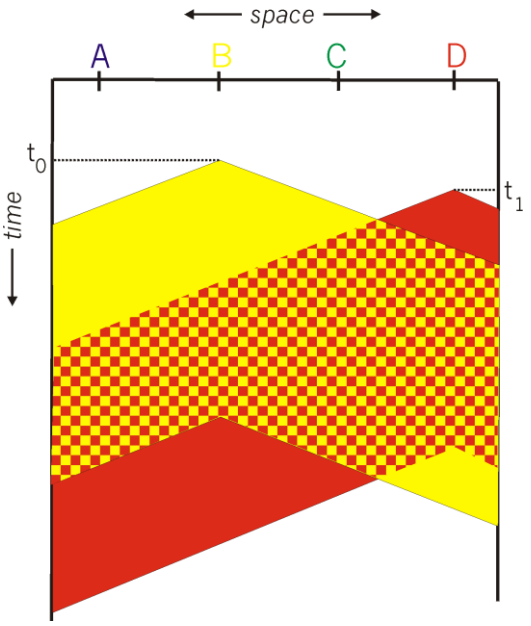
Nhược điểm:

- Khả năng va chạm tang hơn so với Slotted ALOHA
- Hiệu suất = 1/(2e) tức thấp hơn một nữa Slotted ALOHA

CSMA

Lắng nghe trước khi truyền:

- Nếu kênh nhàn rỗi: truyền toàn bộ frame
- Nếu kênh truyền bận: trì hoãn truyền



Đụng độ có thể vẫn xảy ra: trễ lan truyền nghĩa là hai node không thể nghe thấy quá trình truyền lẫn nhau

Đụng độ: Toàn bộ thời gian truyền packet bị lãng phí

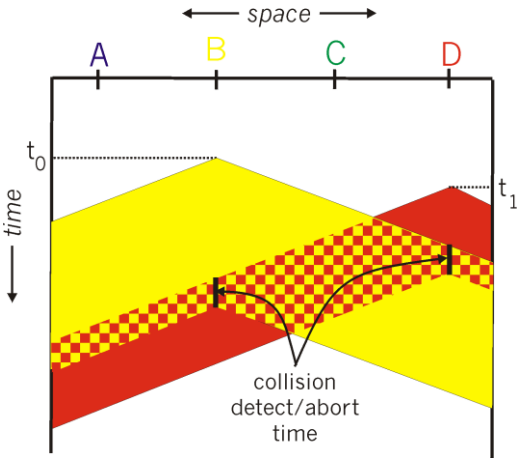
Lưu ý: Vai trò của khoảng cách và thời gian lan truyền có ảnh hưởng đến khả năng va chạm

CSMA/CD (collision detection)

Đụng độ được phát hiện trong khoảng thời gian ngắn. Việc truyền đụng độ được bỏ qua, giảm lãng phí kênh truyền

Phát hiện đụng độ:

- Dễ dàng trong các mạng LAN hữu tuyến: Đo cường độ tín hiệu, so sánh với các tín hiệu đã được truyền và nhận
- Khó thực hiện trong mạng LAN vô tuyến: cường độ tín hiệu được nhận bị áp đảo bởi cường độ truyền cục bộ



Thuật toán Ethernet CSMA/CD

- NIC nhận datagram từ tầng network, tạo frame
- Nếu NIC cảm nhận được kênh rỗi, nó sẽ bắt đầu việc truyền frame. Nếu NIC cảm nhận kênh bận, đợi cho đến khi kênh rãnh, sau đó mới truyền
- Nếu NIC truyền toàn bộ frame mà không phát hiện việc truyền khác, NIC được truyền toàn bộ frame đó!
- Nếu NIC phát hiện có sự truyền khác trong khi đang truyền, thì nó sẽ hủy bỏ truyền và phát tín hiệu tắt nghẽn
- Sau khi hủy bỏ truyền, NIC thực hiện *binary (exponential) backoff:*
  - Sau lần đụng độ thứ *m*, NIC chọn ngẫu nhiên số *K* trong khoảng {0,1,2, …, 2^m-1}. NIC sẽ đợi K·512 bit lần, sau đó trở lại bước 2- tức là NIC sẽ tiếp tục dò kênh
  - Đụng độ nhiều thì sẽ có khoảng thời gian backoff dài hơn

Độ hiệu quả

- T_{prop} : độ trễ lan truyền lớn nhất giữa 2 node trong mạng LAN
- T_{trans}: thời gian để truyền frame có kích thước lớn nhất
- Công thức: $\text{efficiency} = \frac{1}{1 + \frac{5T_{prop}}{T_{trans}}}$
- Hiệu suất tiến tới 1
  - Khi T_{prop} tiến tới 0
  - Khi T_{trans} tiến tới vô cùng
- Hiệu suất tốt hơn ALOHA: đơn giản, chi phí thấp và điều khiển phân tán

Các giao thức MAC – "xoay vòng"

polling:

- Node chủ (master node) "mời" các node con (slave node) truyền lần lượt
- Thường được sử dụng với các thiết bị con "đần độn"
- Quan tâm:
  - polling overhead
  - latency
  - Chỉ có 1 điểm chịu lỗi (master)

Chuyển token:

- Điều hànhtoken được chuyển từ 1 node đến node kế tiếp theo tuần tự.
- Thông điệp token
- Quan tâm:
  - token overhead
  - latency
  - Chỉ có 1 điểm chịu lỗi (master)

5.5 Địa chỉ MAC, Ethernet, Switch

Địa chỉ MAC hay LAN, Ethernet:

- Chức năng: được sử dụng "cục bộ" để chuyển frame từ 1 interface này đến 1 interface được kết nối vật lý với nhau (cùng mạng, trong ý nghĩa địa chỉ IP)
- Địa chỉ MAC 48 bit (cho hầu hết các mạng LAN) được ghi vào trong NIC ROM, đôi khi cũng trong phần mềm
- Mỗi card mạng đều có một địa chỉ MAC duy nhất
- Sự phân bổ địa chỉ MAC được quản lý bởi IEEE
- Ví dụ: 1A-2F-BB-76-09-AD

ARP (address resolution protocol)

- Mỗi node IP (host,router) trên mạng LAN đều có một bảng ARP của nó
  - Địa chỉ IP/MAC ánh xạ cho các node trong mạng LAN:<địa chỉ IP,địa chỉ MAC>

- TTL(Time to live): thời gian sau đó địa chỉ ánh xạ sẽ bị lãng quên (thông thường là 20 phút)

ARP- cách thức hoạt động (cùng mạng LAN)

A muốn gửi datagram tới B

- TH1: Địa chỉ MAC B nằm trong bảng ARP của A => gửi gói tin đi
- TH2: Địa chỉ MAC B không nằm trong bảng ARP của A
  - A sẽ quảng bá (broadcasts) ARP query packet có chứa địa chỉ IP của B
    - Địa chỉ MAC đích là FF-FF-FF-FF-FF-FF
    - Tất cả các node trên mạng LAN sẽ nhận ARP query này
  - B nhận ARP packet, trả lời tới A với địa chỉ MAC của B
    - Frame được gửi tới địa chỉ MAC của A(unicast)

A sẽ lưu lại cặp địa chỉ IP-MAC trong bảng ARP của nó cho tới khi thông tin này trở nên cũ(quá TTL)

ARP là giao thức "plug and play"

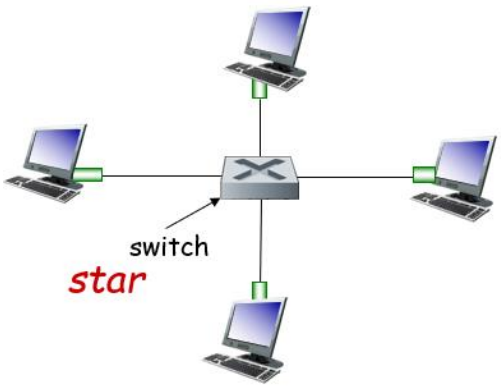
- Các nodes tạo bảng ARP của nó không cần sự can thiệp của người quản trị mạng

Ethernet

Cấu trúc vật lý

- Bus: phổ biến trong giữa thập niên 90
  - Tất cả các node trong cùng collision domain(có thể đụng độ lẫn nhau)
- Star: chiếm ưu thế ngày nay
  - Switch hoạt động ở trung tâm
  - Mỗi chặng kết nối Ethernet hoạt động riêng biệt (các node không đụng độ lẫn nhau)

bus: cáp đồng trục



Cấu trúc frame Ethernet

preamble

- 7 byte với mẫu 10101010 được theo sau bởi 1 byte với mẫu 10101011
- Được sử dụng để đồng bộ tốc độ đồng hồ của người gửi và nhận

addresses: 6 byte địa chỉ MAC nguồn, đích

- Nếu adapter nhận frame với địa chỉ đích đúng là của nó, hoặc với địa chỉ broadcast (như là ARP packet), thì nó sẽ chuyển dữ liệu trong frame tới giao thức tầng network
- Ngược lại, adapter sẽ hủy frame

type: chỉ ra giao thức tầng cao hơn (thường là IP nhưng cũng có thể là những cái khác như là Novell IPX, AppleTalk)

CRC: cyclic redundancy check tại bên nhận

- Lỗi được phát hiện: frame bị bỏ

Ethernet: không tin cậy, không kết nối

- Connectionless (không kết nối): không bắt tay giữa các NIC gửi và nhận
- Unreliable(không tin cậy): NIC nhận sẽ không gửi thông báo nhận thành công (acks) hoặc không thành công (nacks) đến các NIC gửi
  - Dữ liệu trong các frame bị bỏ sẽ được khôi phục lại chỉ khi nếu bên gửi dùng dịch vụ tin cậy của tầng cao hơn (như là TCP) còn không thì dữ liệu mà đã bị bỏ sẽ mất luôn
- Giao thức MAC của Ethernet: unslotted CSMA/CD với binary backoff

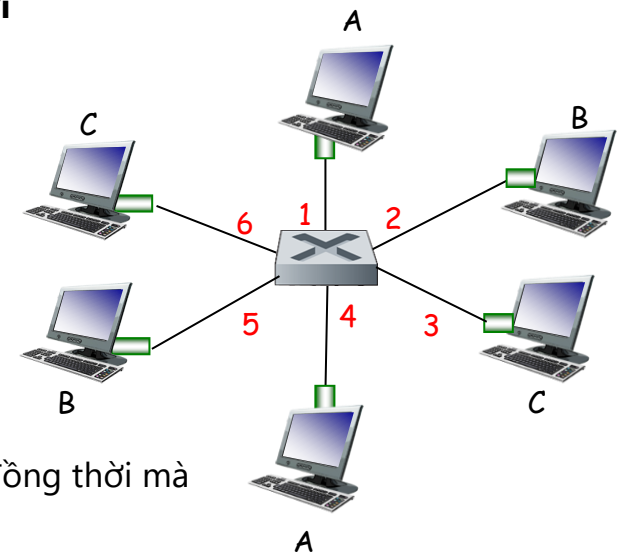
Switch

- Thiết bị tầng link: giữ vai trò tích cực
  - Lưu và chuyển tiếp các frame Ethernet

- Xem xét địa chỉ MAC của frame đến, chọn lựa chuyển tiếp frame tới 1 hay nhiều đường link đi ra khi frame được chuyển tiếp vào segment, dùng CSMA/CD để truy nhập segment
- Transparent(trong suốt)
  - Các host không nhận thức được sự hiện diện của các switch
- Plug and Play
  - Các switch không cần được cấu hình

Switch - cho phép nhiều đường truyền đồng thời

- Các host kết nối trực tiếp tới switch
- Switch lưu tạm các packet
- Giao thức Ethernet được sử dụng trên mỗi đường kết nối vào, nhưng không có đụng độ;cho phép dữ liệu truyền theo hai chiều đồng thời [full duplex]
  - Mỗi đường kết nối là 1 collision domain của riêng nó
- switching: A-tới-A′ và B-tới-B′ có thể truyền đồng thời mà không có đụng độ xảy ra



switch với 6 interface

(1,2,3,4,5,6)

Switch – tự học

- Switch học các host có thể tới được thông qua các interface kết nối với các host đó
  - Khi frame được nhận, switch học vị trí của bên gửi: incoming LAN segment
  - Switch ghi nhận lại cặp thông tin về host gửi (MAC addr) và nhánh mạng chứa host gửi (interface)
- Switch khi nhận được frame
  - Ghi lại đường kết nối vào, địa chỉ MAC của host gửi
  - Ghi vào mục lục bảng switch với địa chỉ MAC đích
  - **Nếu** tìm thấy thông tin đích đến

    Thì {

    Nếu đích đến nằm trên phân đoạn mạng từ cái mà frame đã đến

Thì bỏ frame

Ngược lại, chuyển tiếp frame trên interface được chỉ định bởi thông tin trong bảng switch

} **ngược lại**, flood (chuyển tiếp trên tất cả interface ngoại trừ interface nhận frame đó)

So sánh giữa switch và router

Giống nhau:

- Đều có thể lưu (store) và chuyển tiếp (forwarding)
- Đều có bảng định tuyến

Khác nhau:

Switch	Router
Thiết bị của tầng liên kết Tự học bảng forwarding bằng cơ chế flooding Địa chỉ MAC	Thiết bị của tầng liên mạng Sử dụng các thuật toán định tuyến Địa chỉ IP