☆COMPUTER NETWORKING☆

Chương I: Giới thiệu

I. 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 of 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

II. 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

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

III. 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

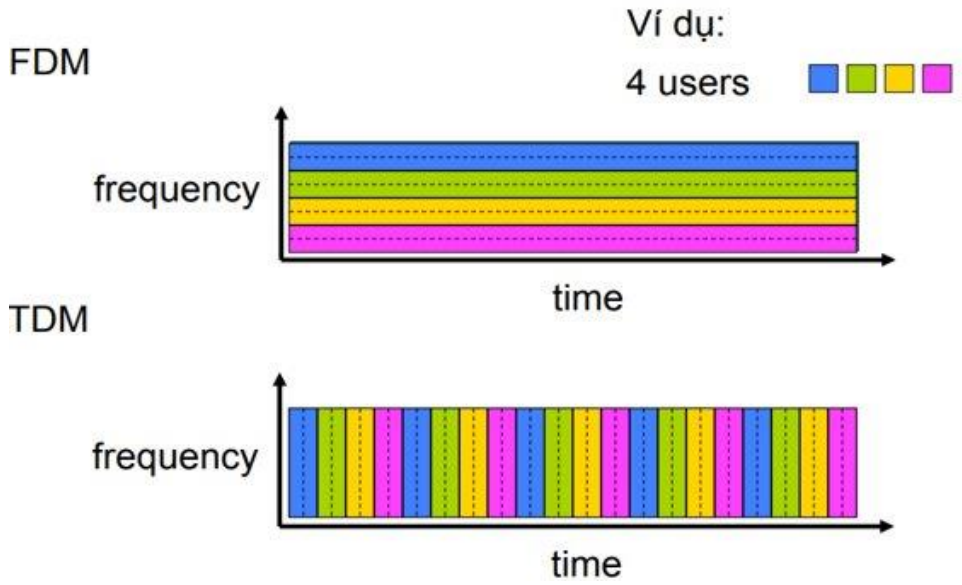
❖ Chuyển mạch gói
- Host chia nhỏ dữ liệu từ tầng ứng dụng thành các packet
- Chuyển tiếp các gói từ bộ định tuyến này đến bộ định tuyến tiếp theo trên đường đi từ nguồn tới đích
- Mỗi packet được truyền với công suất lớn nhất của đường truyền

❖ Độ trễ hàng đợi, sự mất mát: nếu tốc độ đến của đường link cao hơn tốc độ truyền của đường link trong 1 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: 2 điểm đầu cuối muốn trao đổi thông tin sẽ thiết lập 1 kênh truyền riêng (circuit) với băng thông được cấp phát dành riêng cho cả 2 cho tới khi truyền tin kết thúc
- Kênh không được chia sẽ nên sẽ rảnh rỗi lúc không truyền tin

❖ 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 chuyền 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 1 băng tần tín hiệu riêng



Chuyển mạch gói	Chuyển mạch kênh
❖ **Ưu điểm:** Băng thông được sử dụng tốt hơn do không bị giới hạn trên 1 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 2 bên ngắt kết nối và có thể lãng phí băng thông nếu không truyền dữ liệu Thời gian thiết lập kênh truyền quá lâu

IV. 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
  - 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

R: băng thông đường link (bps)

L: độ dài gói tin (bits)

a: tỷ lệ trung bình gói tin đến

Cường độ lưu thông = La/R

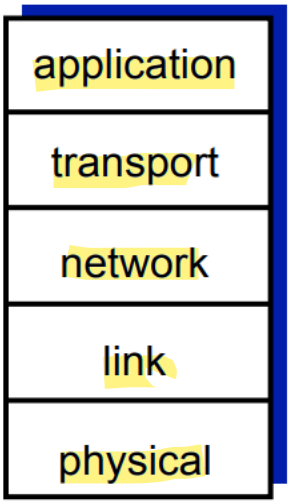
- La/R ~ 0: trễ trung bình nhỏ
- La/R -> 1: trễ trung bình lớn
- La/R > 1: nhiều "việc" đến hơn khả năng phục vụ, trễ trung bình vô hạn!

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

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

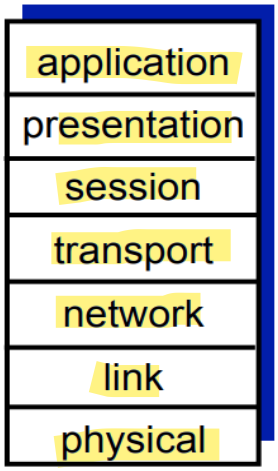
V. Các lớp giao thức

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

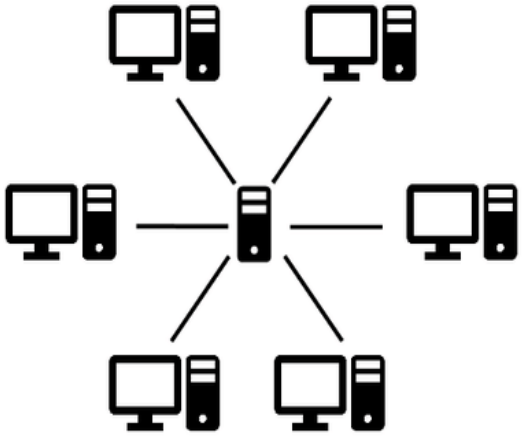


Chương II: Tầng Application

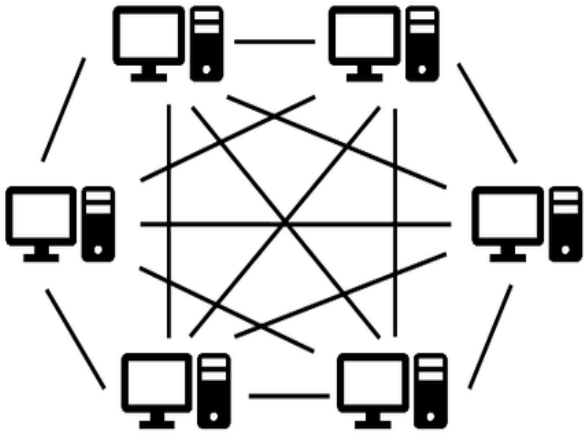
I. Các nguyên lý của các ứng dụng mạng

Các kiến trúc ứng dụng:

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



Client-Server network



P2P network

	Client – Server	P2P
Server	- 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 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).

Client, Server:

Tiến trình Client: tiến trình khởi tạo truyền thông.

Tiến trình Server: tiến trình chờ đợi để được liên lạc.

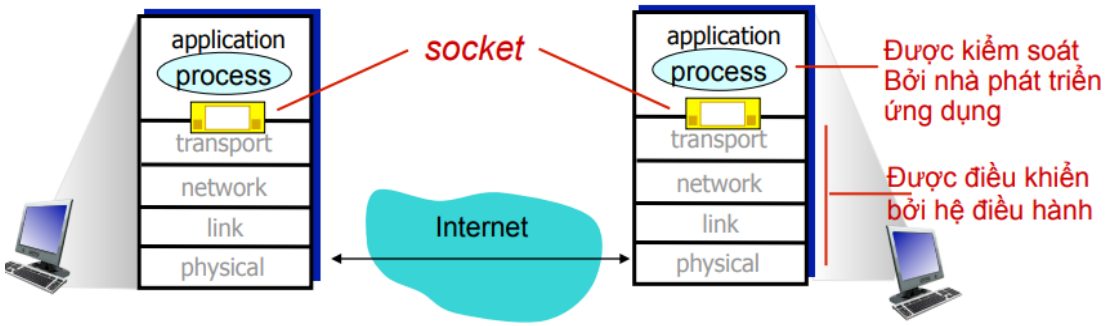
Chú ý: Các ứng dụng với kiến trúc P2P có cả tiến trình client và tiến trình server.

Sockets:

- Socket là giao diện lập trình ứng dụng mạng được dùng để truyền và nhận dữ liệu trên Internet. Giữa hai chương trình chạy trên mạng cần có một liên kết giao tiếp 2 chiều (two-way communication) để kết nối 2 tiến trình trò chuyện với nhau. Điểm cuối (endpoint) của liên kết này được gọi là socket.

=> **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 và số cổng (port number).

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

Một số Port tầng Application:

Protocol	Port number	Protocol	Port number
FTP	20	HTTP	80
SSH	22	Telnet	23
DNS	53	HTTPS	443
LPD	515	TFTP	69
SMTP	25	NFS	2049

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

II. 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

HTTP không bền vững (Nonpersistent HTTP)	HTTP bền vững (Persistent HTTP)
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	Nhiều đối tượng có thể gửi qua một kết nối TCP giữa Client và Server.
HTTP/1.0 (RFC 1945)	HTTP/1.1 (RFC 2616)
Thời gian đáp ứng	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.	Persistent without pipelining Client chỉ gửi request khi đã nhận response trước. 1 RTT cho 1 đối tượng được quan tâm.	Persistent with pipelining Client gửi request liên tục đến các đối tượng được quan tâm. Có thể 1 RTT cho tất cả các đối tượng được quan tâm.
Các phương thức: GET POST HEAD	Các phương thức: GET POST HEAD PUT DELETE	

HTTP Cookie: 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:

1) Cookie header line của thông điệp đáp ứng HTTP
2) Cookie header line trong thông điệp đáp ứng HTTP kế tiếp
3) File cookie được lưu trữ trên host của người dùng, được quản lý bởi trình duyệt của người sử dụng
4) 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).

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.

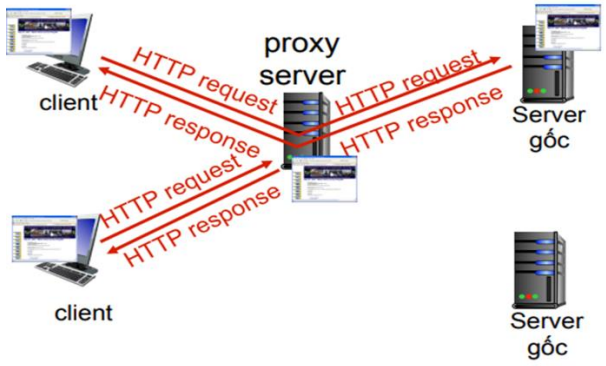
301 Moved Permanently – Đối tượng được yêu cầu đã di chuyển, vị trí mới được xác định sau trong thông điệp này.

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.

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.

III. 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/lưu trữ mail vào server người nhận => gửi mail

Giao thức truy cập mail: truy cập mail từ server người nhận => nhận mail

IV. 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.

DNS caching, cập nhật các record:

- 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)

1. Type **A** lưu ở **authoritative** (mp3.zing.vn, ...)

- name: tên host

- value: địa chỉ IP

2. Type **NS** lưu ở **TLD** (Zing.vn, ...)

- name: tên miền
- value: tên host của name server có thẩm quyền cho tên miền này

3. 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

4. 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.

V. 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

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

Chương 3: Tầng Transport

I. 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

II. Multiplexing và demultiplexing

1. 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)

2. 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

3. 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

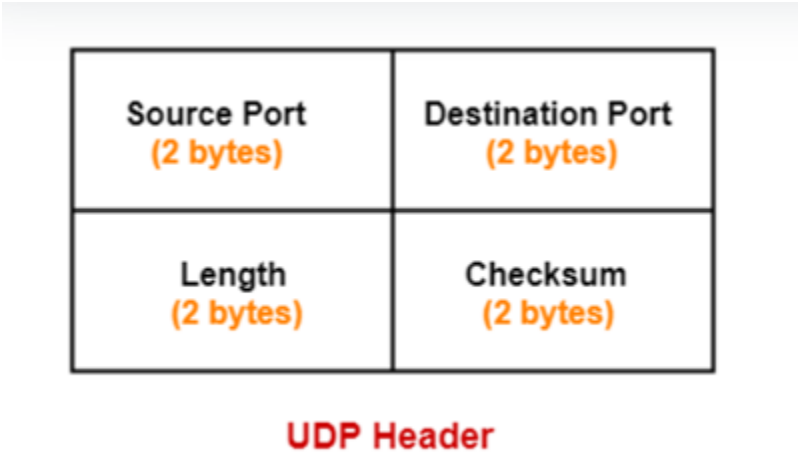
4. demultiplexing hướng kết nối

– 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*

III. Vận chuyển phi kết nối UDP

– UDP (RFC 768): đơn giản, không rườm rà là một giao thức thuộc Transport
– Các segment của UDP: bị mất, vận chuyển không theo thứ tự
– 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 segment header



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

IV. Các nguyên lý truyền dữ liệu tin cậy

1. 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.

2. 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*.

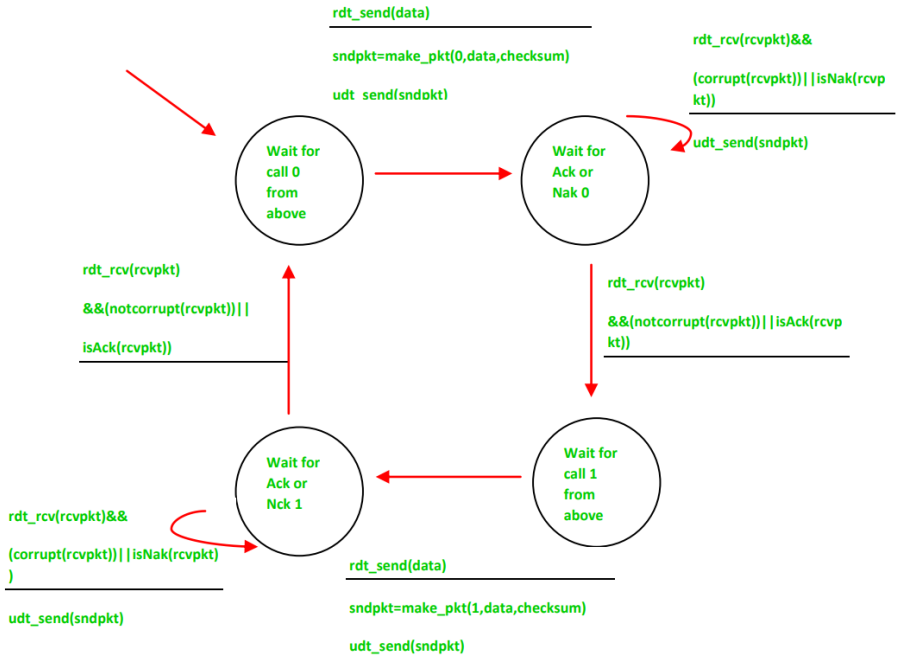
- Bên nhận: Trang web bên nhận có một trạng thái duy nhất, ngay khi packet đến, bên nhận sẽ trả lời bằng ACK hoặc NAK, tùy thuộc vào việc packet nhận được có bị hỏng hay không, tức là bằng cách sử dụng rdt_rcv (rcvpkt) && corrupt (rcvpkt) ở đâu một packet được nhận và được phát hiện là có lỗi hoặc rdt_rcv (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.

3. RDT 2.1

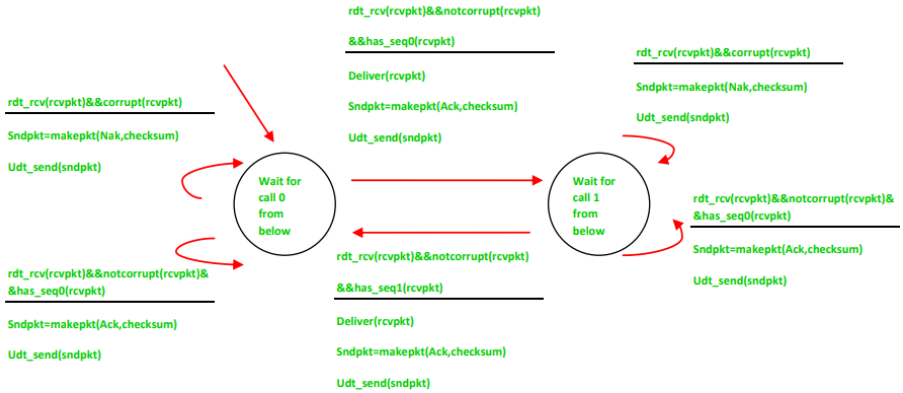
- Nguyên nhân ra đời: Giao thức RDT 2.0 giải thích chức năng trong kênh có lỗi bit. Nó giới thiệu các ACKs và NAKs, nếu người nhận nhận được gói tin bị hỏng máy nhận sẽ gửi thông báo xác nhận NAK và ngược lại trong trường hợp chính xác. Nó không thành công khi ACK bị hỏng. RDT 2.1 giải quyết vấn đề này.
- 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'.

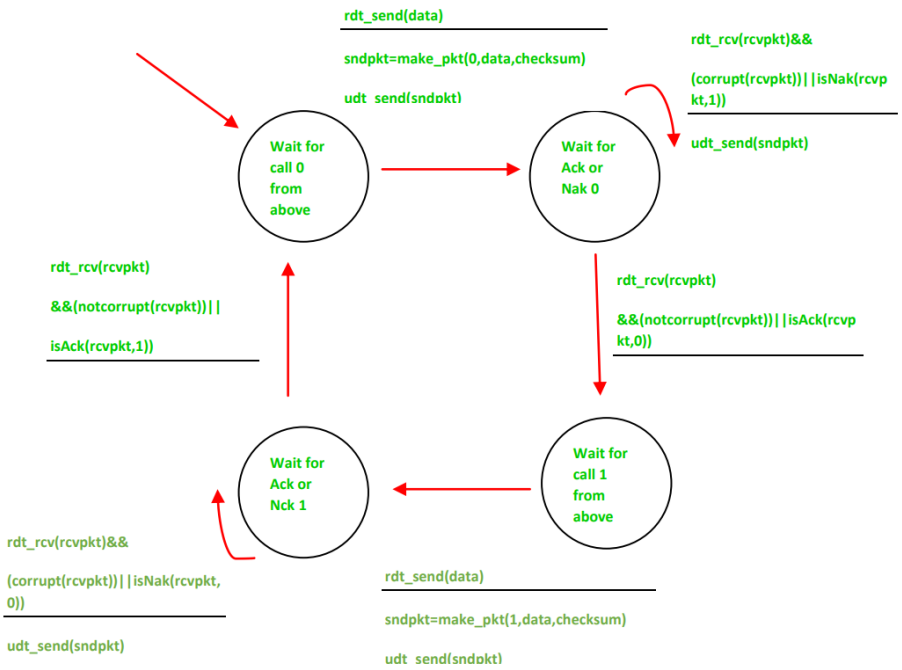
- Bên nhận:



  - 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.

4. RDT 2.2

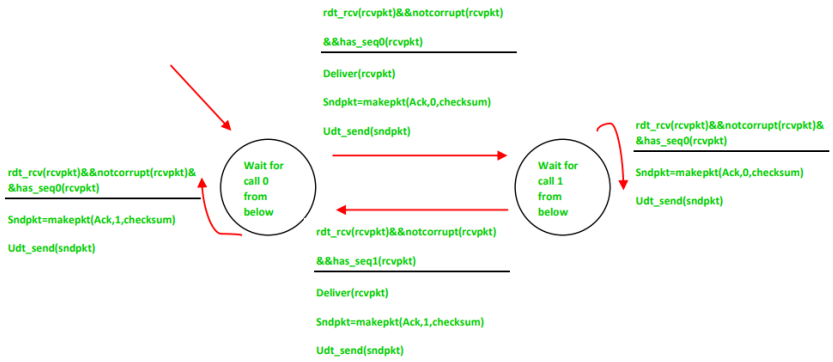
- Thay đổi nổi bật trong RDT 2.2 là loại trừ NAK.
- FSM:
  - Bên gửi:



- 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:



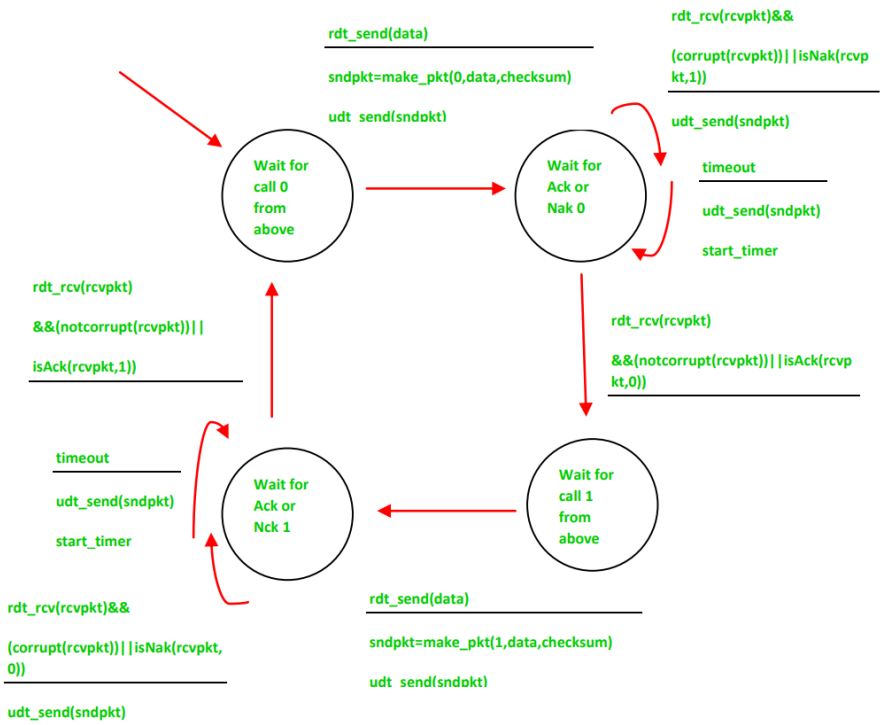
  - Trạng thái 1 (*Bên trái*): Nếu gói nhận được bị hỏng hoặc có seq với '1', nó sẽ gửi xác nhận với sequence number '1' cho nó để cho người gửi biết rằng gói được gửi không theo thứ tự. Nếu gói không bị hỏng và có sequence number '0', máy thu sẽ trích xuất dữ liệu và gửi xác nhận với seq '0'. Nó chuyển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

5. 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.**

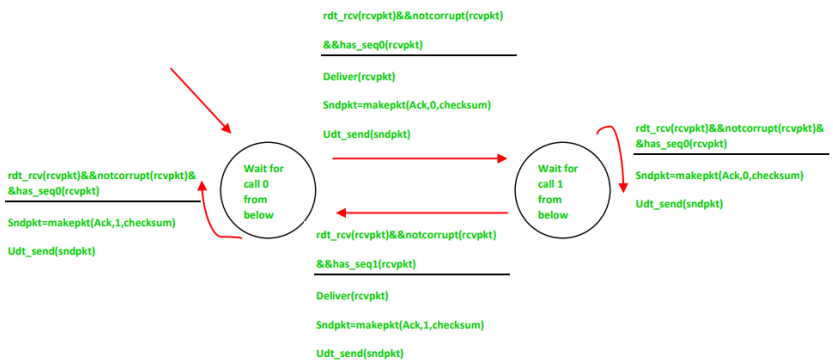
- FSM:
  - Bên gửi:



  - 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:

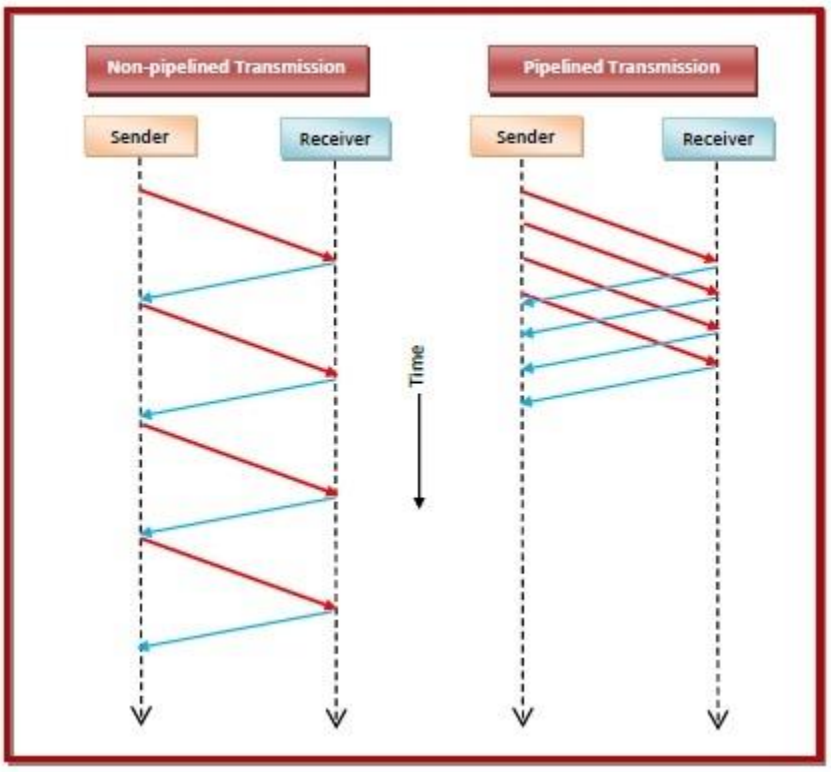


- Trạng thái 1 (*Bên trái*): Đây là Trạng thái đầu tiên trong FSM của người nhận, được gọi là "*wait for call 0 from below*". Trạng thái này cho biết gói nhận được có sequence number "0" và không bị hỏng hay không. Nếu các điều kiện này thỏa mãn trạng thái này tạo ra một gói báo nhận với seq "0" và đẩy 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

6. Các giao thức Pipelined

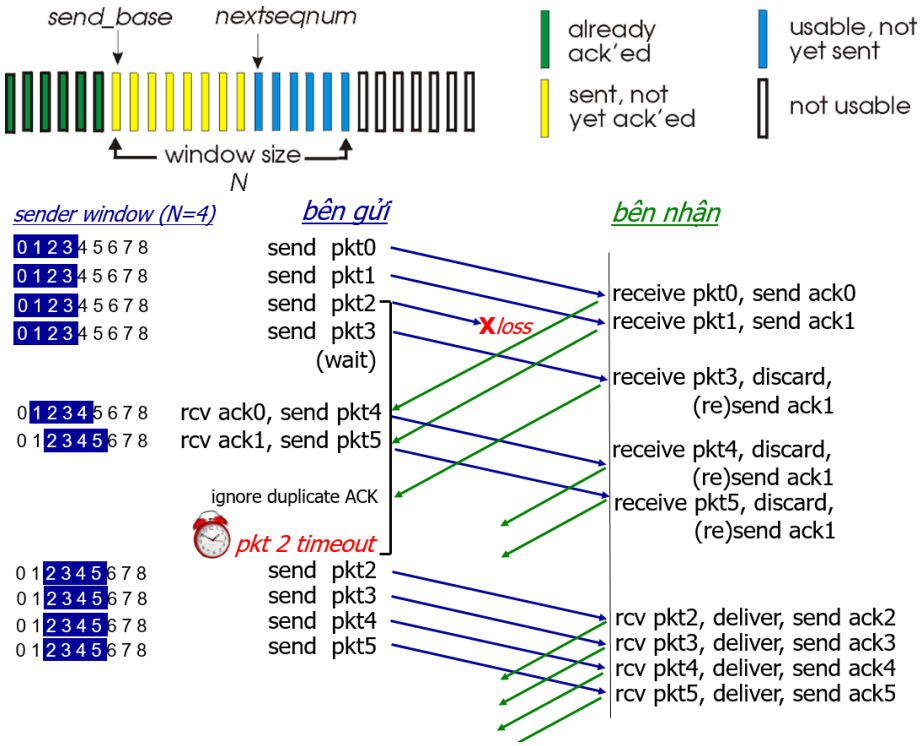
- Bên gửi cho phép gửi nhiều gói đồng thời, không cần chờ báo xác nhận ACK.
- Dải số thứ tự phải được tăng lên.
- Phải có bộ nhớ đệm tại nơi gửi và/hoặc nhận
- Kỹ thuật này có lợi khi lượng dữ liệu cần truyền rất lớn, và gửi dữ liệu bằng cách chia chúng thành nhiều phần khác nhau. Những phần dữ liệu này có thể được pipelined và gửi đến người nhận qua kênh. Trong pipelining, chúng tôi không đợi các gói dữ liệu đã gửi ACK. Chúng tôi tiếp tục gửi các gói dữ liệu liên tục mà không cần bận tâm về các ACK.
- Giao thức Sliding Window

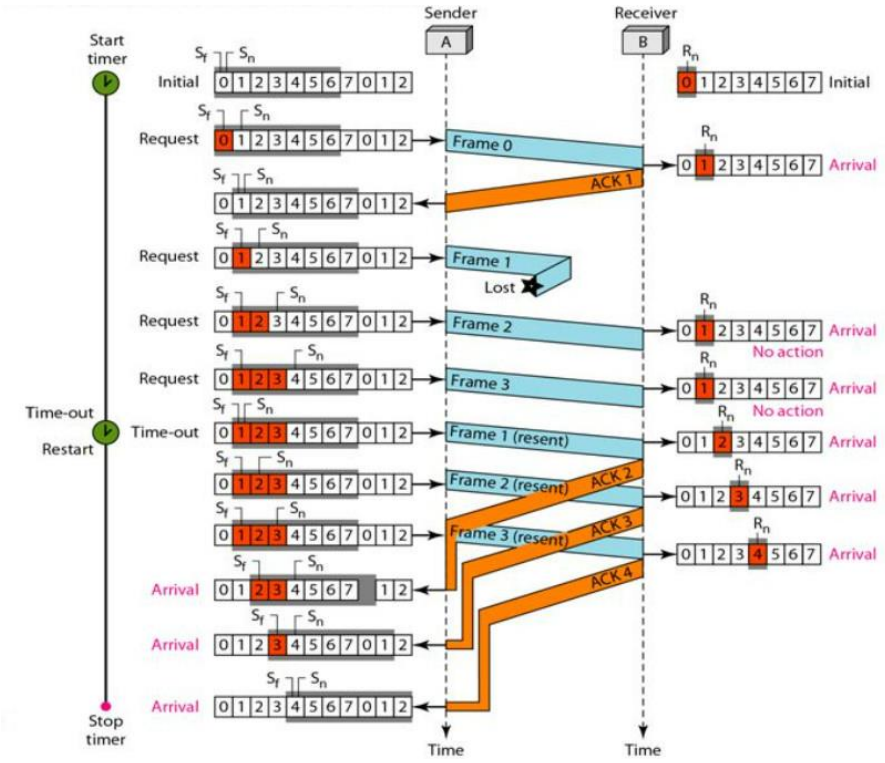


a. Go-Back-N

- Người gửi gửi N gói tin có kích thước với số lượng gói tin bằng kích thước window. Sau khi toàn bộ window được gửi đi, người gửi sẽ đợi ACK tích lũy để gửi thêm gói tin. Ở phía đầu thu, nó chỉ nhận các gói theo thứ tự và loại bỏ các gói không theo thứ tự. Như trong trường hợp mất gói, toàn bộ window sẽ được truyền lại
- Trong Go-Back-N, kích thước window người gửi là N và kích thước window người nhận luôn là 1.
- Go-Back-N sử dụng các ACK tích lũy: Bên nhận duy trì một bộ đếm thời gian báo nhận. Mỗi khi bên nhận nhận được một frame mới, nó sẽ bắt đầu một bộ đếm thời gian báo nhận mới. Sau khi bộ đếm thời gian hết hạn, bộ thu sẽ gửi ACK tích lũy cho tất cả các frame không được xác nhận tại thời điểm đó.
- Go-Back-N không chấp nhận các frame bị hỏng và âm thầm loại bỏ chúng:
    - Nếu bên nhận nhận được một frame bị hỏng, thì nó sẽ âm thầm loại bỏ frame đó. Khung chính xác được người gửi truyền lại sau khi hết thời gian chờ.
    - Loại bỏ frame một cách âm thầm có nghĩa là- "Chỉ cần từ chối frame và không thực hiện bất kỳ hành động nào"
- Go-Back-N dẫn đến việc truyền lại toàn bộ window nếu đối với bất kỳ frame nào, người gửi không nhận được ACK:
    - Bên nhận âm thầm loại bỏ frame nếu nó làm cho frame bị hỏng hoặc không theo thứ tự. Nó không gửi bất kỳ ACK nào cho frame như vậy. Nó cũng âm thầm loại bỏ các frame sau.
    - Nếu đối với bất kỳ frame cụ thể nào, người gửi không nhận được bất kỳ ACK nào, thì nó hiểu rằng cùng với frame đó, tất cả các frame sau cũng phải bị người nhận loại bỏ. Vì vậy, người gửi cũng phải truyền lại tất cả các frame sau cùng với frame cụ thể đó. Do đó, nó dẫn đến việc truyền lại toàn bộ window. Đó là lý do tại sao, giao thức đã được đặt tên là "Go-Back-N".

- Go-Back-N dẫn đến việc truyền lại các frame bị mất sau khi hết thời gian chờ:
    - Xem xét một frame được gửi đến người nhận bị mất trên đường đi.
    - Sau đó, nó chỉ được truyền lại sau khi hết thời gian chờ cho frame đó ở phía người gửi.





- Nhược điểm:
  - Nếu không nhận được ACK cho một frame, toàn bộ window của các frame sẽ được truyền lại thay vì chỉ frame bị hỏng. Điều này làm cho giao thức Go-Back-N không hiệu quả.
  - Truyền lại tất cả các fame khi phát hiện một frame bị hỏng làm tăng tắc nghẽn kênh và cũng làm tăng yêu cầu băng thông.
  - Tốn thời gian hơn vì trong khi truyền lại các frame trên phát hiện frame bị lỗi thì các frame không bị lỗi cũng được truyền theo.

b. Selective Repeat (Lặp có lựa chọn)

- Trong giao thức Selective Repeat, kích thước window bên gửi luôn giống với kích thước window bên nhận.
- Giao thức Selective Repeat, chỉ sử dụng các ACK độc lập:
  - Bên nhận từng frame một cách độc lập.
  - Khi bên nhận nhận được một frame mới từ người gửi, nó sẽ gửi ACK của nó.
- Giao thức Selective Repeat không chấp nhận các frame bị hỏng nhưng không âm thầm loại bỏ chúng:

  - Nếu bên nhận nhận được một frame bị hỏng, thì nó sẽ không âm thầm loại bỏ frame đó.
  - Người nhận xử lý tình huống một cách hiệu quả bằng cách gửi một NAK, NAK cho phép truyền lại sớm frame bị hỏng. Nó cũng tránh việc chờ đợi hết thời gian chờ ở phía người gửi để truyền lại frame.
- Giao thức Selective Repeat yêu cầu tìm kiếm ở bên gửi:
  - Bên nhận không từ chối các frame bên ngoài.
  - Bên nhận chấp nhận các frame không theo thứ tự và sắp xếp chúng sau. Do đó, chỉ có frame bị thiếu mới được gửi bởi bên gửi.
  - Để gửi khung bị thiếu, bên gửi thực hiện tìm kiếm và tìm khung bị thiếu. Sau đó, bên gửi lặp lại khung đó một cách có chọn lọc. Do đó, chỉ frame đã chọn được lặp lại chứ không phải toàn bộ window.
  - Đó là lý do tại sao, giao thức được đặt tên là "Selective Repeat"
- Giao thức Selective Repeat chấp nhận các frame không theo thứ tự:
  - Xem xét bên nhận nhận được một frame có sequence number không phải là thứ mà máy thu mong đợi.
  - Sau đó, nó không loại bỏ frame đó mà chấp nhận và giữ trong cửa sổ của nó
- Giao thức Selective Repeat dẫn đến việc truyền lại các frame bị mất sau khi hết thời gian chờ.

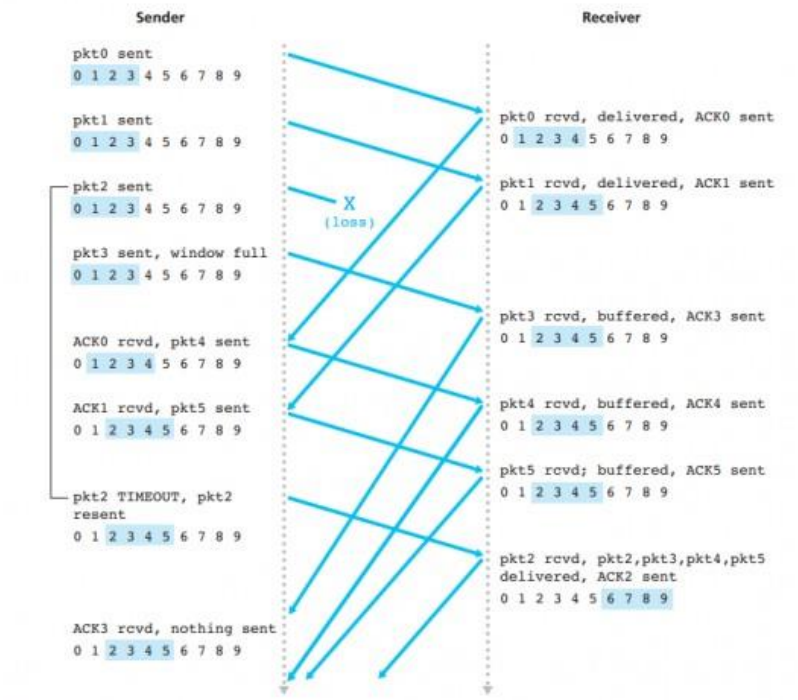
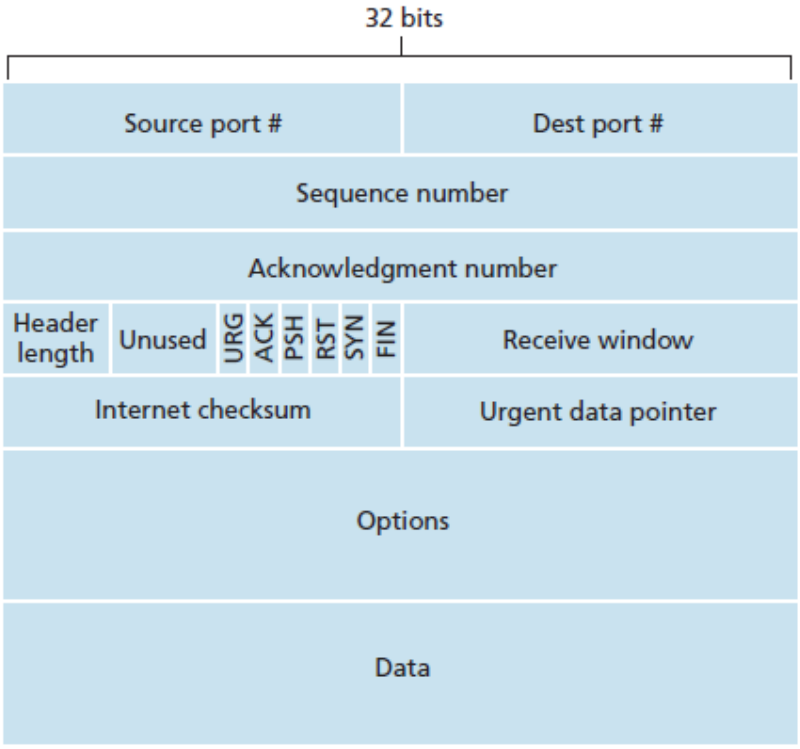


Figure 3.26 ♦ SR operation

V. Vận chuyển hướng kết nối TCP

1. 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.*

2. 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.

3. **Điều khiển luồng (flow control)**

- 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)**.

4. **Quản lí kết nối**

   a. **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)

   b. **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.

VI. Các nguyên lý về điều khiển tắc nghẽn

- 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),...

VII. 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

1. **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

2. **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

3. **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 = $\frac{1}{2}cwnd_{prev}$