

# Chương 7: Tầng Vận chuyển

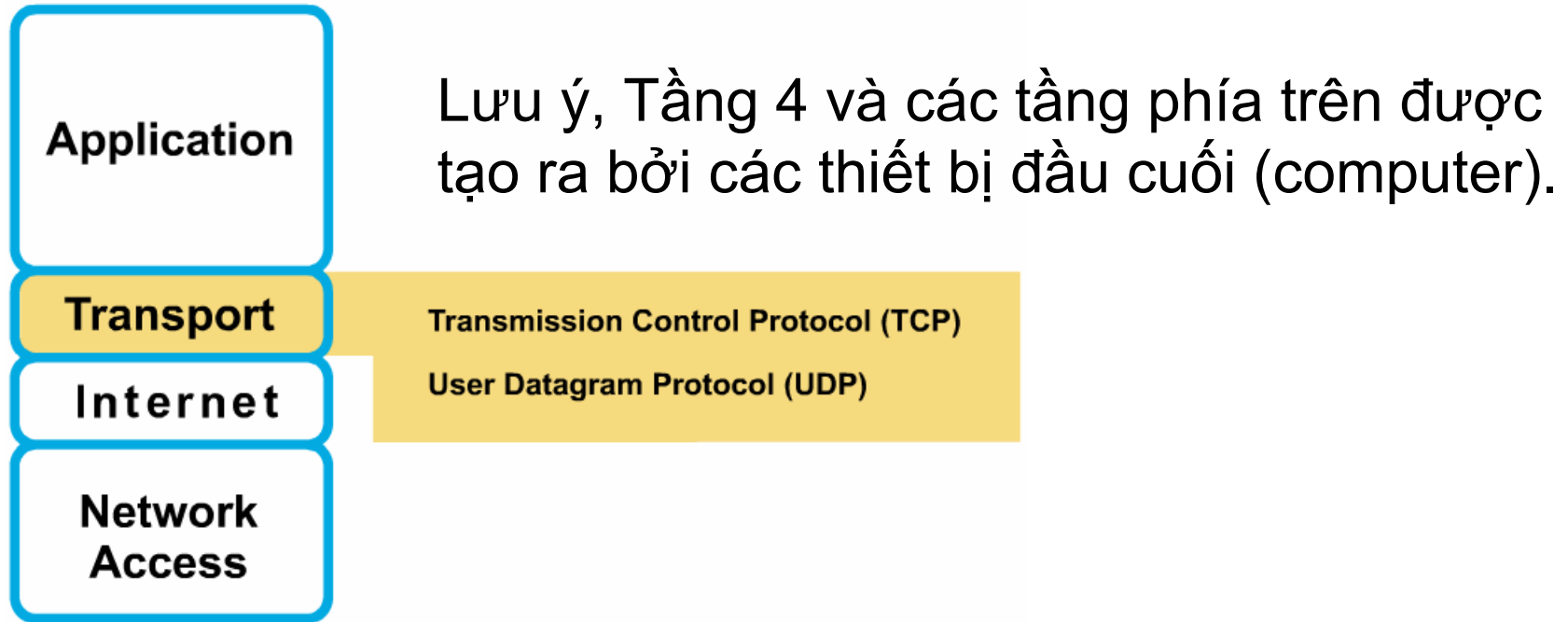
## Mục tiêu:

- ❑ Hiểu rõ các nguyên lý đằng sau các dịch vụ của tầng vận chuyển:
  - multiplexing/demultiplexing
  - vận chuyển dữ liệu tin cậy
  - kiểm soát luồng
  - kiểm soát tắc nghẽn
- ❑ Học về các giao thức tầng vận chuyển trên Internet:
  - UDP: vận chuyển phi kết nối
  - TCP: vận chuyển hướng kết nối
  - Kiểm soát tắc nghẽn trong TCP

# Chương 7 - Nội dung

- ❑ 7.1 Các dịch vụ của tầng Vận chuyển
- ❑ 7.2 Multiplexing và demultiplexing
- ❑ 7.3 Vận chuyển phi kết nối: UDP
- ❑ 7.4 Vận chuyển hướng kết nối: TCP
  - cấu trúc segment
  - quản lý kết nối
  - truyền dữ liệu tin cậy
  - kiểm soát luồng
- ❑ 7.5 Kiểm soát tắc nghẽn trong TCP

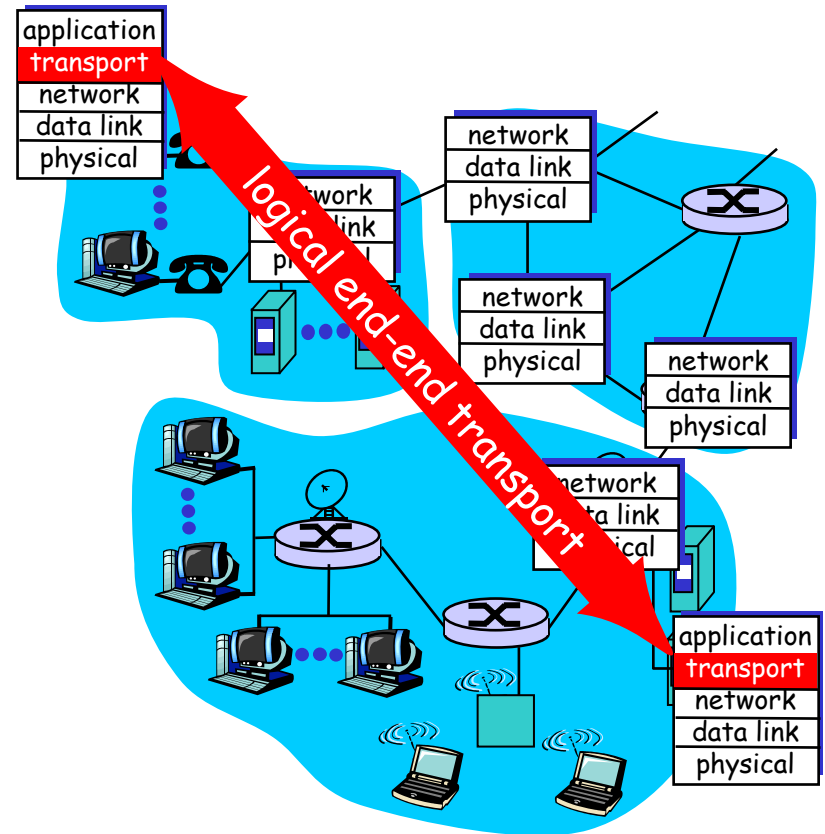
# Transport Layer Overview



- ❑ Tầng Vận chuyển cho phép một thiết bị của người sử dụng phân đoạn dữ liệu của các ứng dụng ở tầng trên để đặt vào cùng dòng dữ liệu tầng 4, và cho phép thiết bị nhận ráp nối lại các đoạn dữ liệu đó để chuyển lên cho tầng trên.
- ❑ Dòng dữ liệu tầng 4 là một kết nối logic giữa các điểm cuối của mạng, và cung cấp các dịch vụ vận chuyển từ một trạm đến một đích nào đó.
- ❑ Điều đó còn được xem như là dịch vụ cuối-đến-cuối.

# Các dịch vụ vận chuyển và giao thức

- ❑ cung cấp *truyền thông logic* giữa các tiến trình ứng dụng chạy trên các trạm khác nhau
- ❑ các giao thức vận chuyển chạy trên các hệ thống đầu cuối
  - bên gửi: chẻ các thông điệp tầng ứng dụng thành *segments*, đưa chúng xuống cho tầng mạng
  - bên nhận: ráp nối các *segments* lại thành các thông điệp, đưa lên cho tầng ứng dụng
- ❑ có nhiều giao thức ở tầng vận chuyển để phục vụ cho tầng ứng dụng
  - Internet: TCP và UDP



# Tầng Vận chuyển so với Tầng mạng

- ❑ *tầng mạng*: truyền thông logic giữa các trạm
- ❑ *tầng vận chuyển*: truyền thông logic giữa các tiến trình trên các trạm
  - dựa vào và nâng cao các dịch vụ và tầng mạng cung cấp

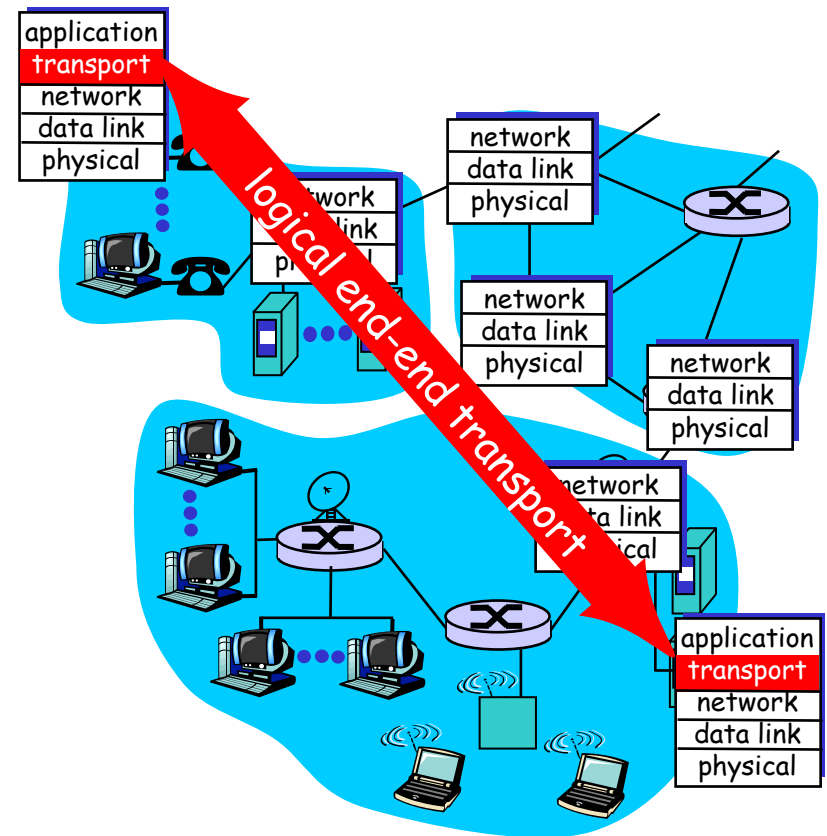
## Tương tự như các hộ gia đình:

*12 kids gửi thư cho 12 kids*

- ❑ tiến trình = kids
- ❑ thông điệp tầng ứng dụng = các bức thư trong bì thư
- ❑ các trạm = các nhà
- ❑ giao thức vận chuyển = Ann và Bill
- ❑ giao thức tầng mạng = dịch vụ bưu điện

# Các giao thức tầng Vận chuyển trên Internet

- ❑ phân phát tin cậy, có thứ tự (TCP)
  - thiết lập kết nối
  - kiểm soát tắc nghẽn
  - kiểm soát luồng
- ❑ phân phát không tin cậy, không thứ tự: UDP
  - không có sự mở rộng đặc biệt nào so với "nỗ lực tối đa" của IP
- ❑ các dịch vụ không sẵn có:
  - đảm bảo về độ trễ
  - đảm bảo về dải thông



# Chương 7 - Nội dung

- ❑ 7.1 Các dịch vụ của tầng Vận chuyển
- ❑ 7.2 Multiplexing và demultiplexing
- ❑ 7.3 Vận chuyển phi kết nối: UDP
- ❑ 7.4 Vận chuyển hướng kết nối: TCP
  - cấu trúc segment
  - quản lý kết nối
  - truyền dữ liệu tin cậy
  - kiểm soát luồng
- ❑ 7.5 Kiểm soát tắc nghẽn trong TCP


# Multiplexing/demultiplexing

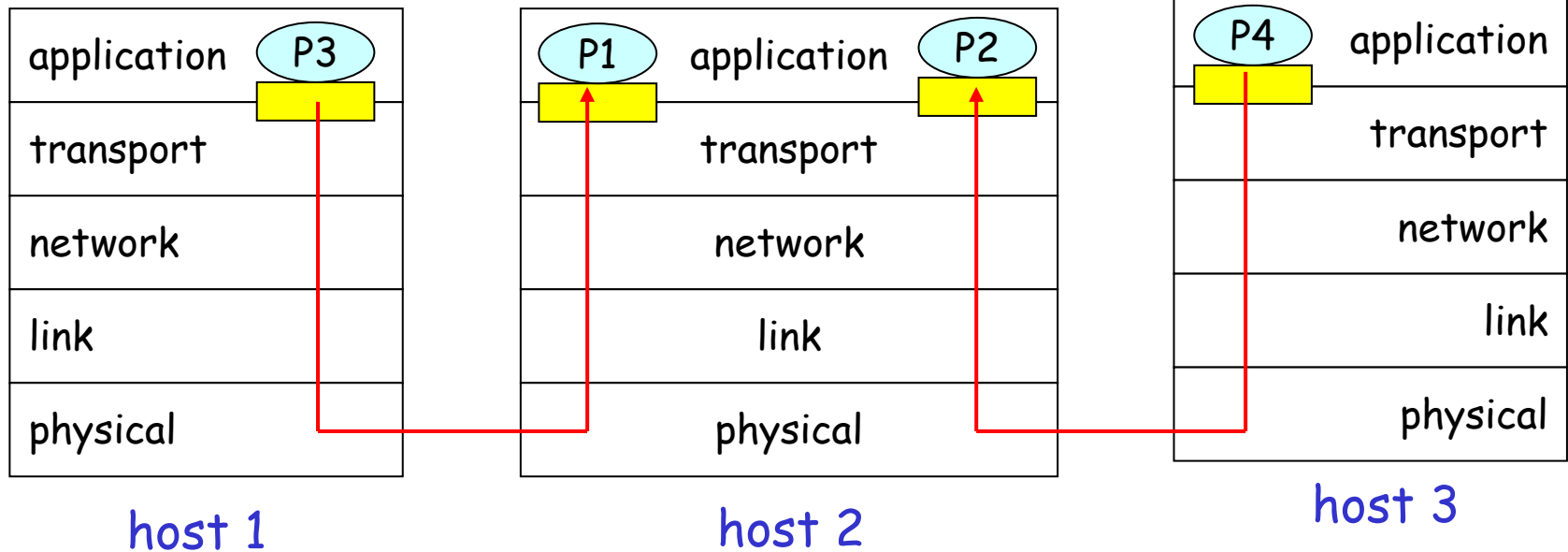
## Multiplexing tại trạm gửi:

tập hợp dữ liệu từ nhiều sockets, bao bọc dữ liệu với thông tin điều khiển (để phục vụ cho demultiplexing sau này)

## Demultiplexing tại trạm nhận:

phân phát các segments nhận được đến đúng socket

 = socket       = process



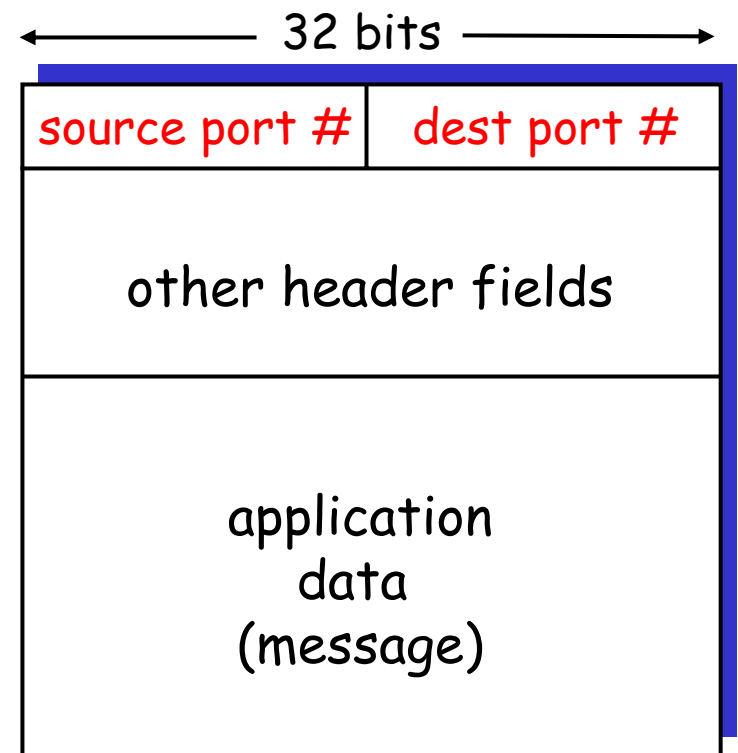


# Demultiplexing làm việc như thế nào

## □ trạm nhận các IP datagrams

- mỗi datagram có địa chỉ IP nguồn, địa chỉ IP đích
- mỗi datagram mang một segment của tầng Vận chuyển
- mỗi segment có số hiệu cổng (port) nguồn và đích

## □ trạm dùng địa chỉ IP và số hiệu cổng để chuyển segment đến socket thích hợp



TCP/UDP segment format

# Demultiplexing trong phi kết nối

- ❑ Tạo sockets với các số hiệu cổng:

```
DatagramSocket mySocket1 = new  
    DatagramSocket(99111);
```

```
DatagramSocket mySocket2 = new  
    DatagramSocket(99222);
```

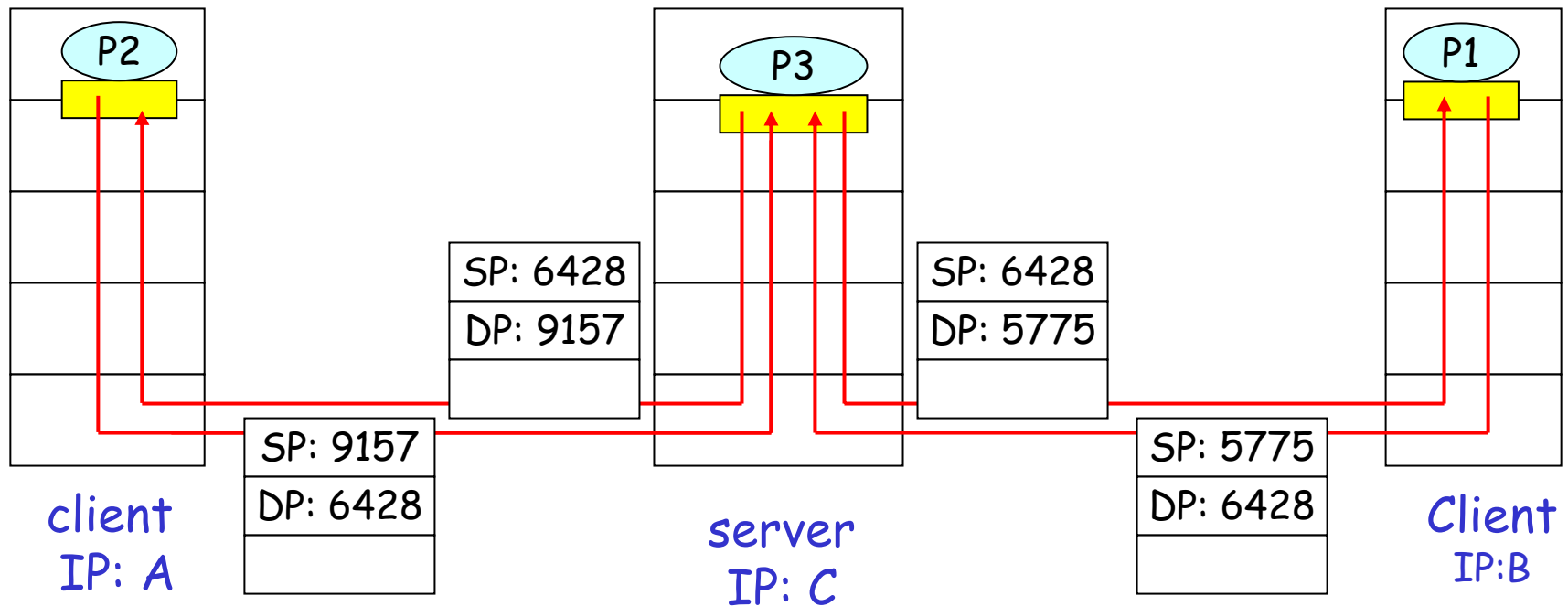
- ❑ UDP socket được định danh bởi bộ hai:

(địa chỉ IP đích, số hiệu cổng đích)

- ❑ Khi trạm nhận UDP segment:
  - kiểm tra giá trị cổng đích trong segment
  - gửi UDP segment đến socket đang mở tại cổng đó
- ❑ IP datagrams với địa chỉ IP nguồn khác nhau và/hoặc số hiệu cổng nguồn khác nhau cũng được gửi đến cùng socket

# Demultiplexing trong phi kết nối (tiếp)

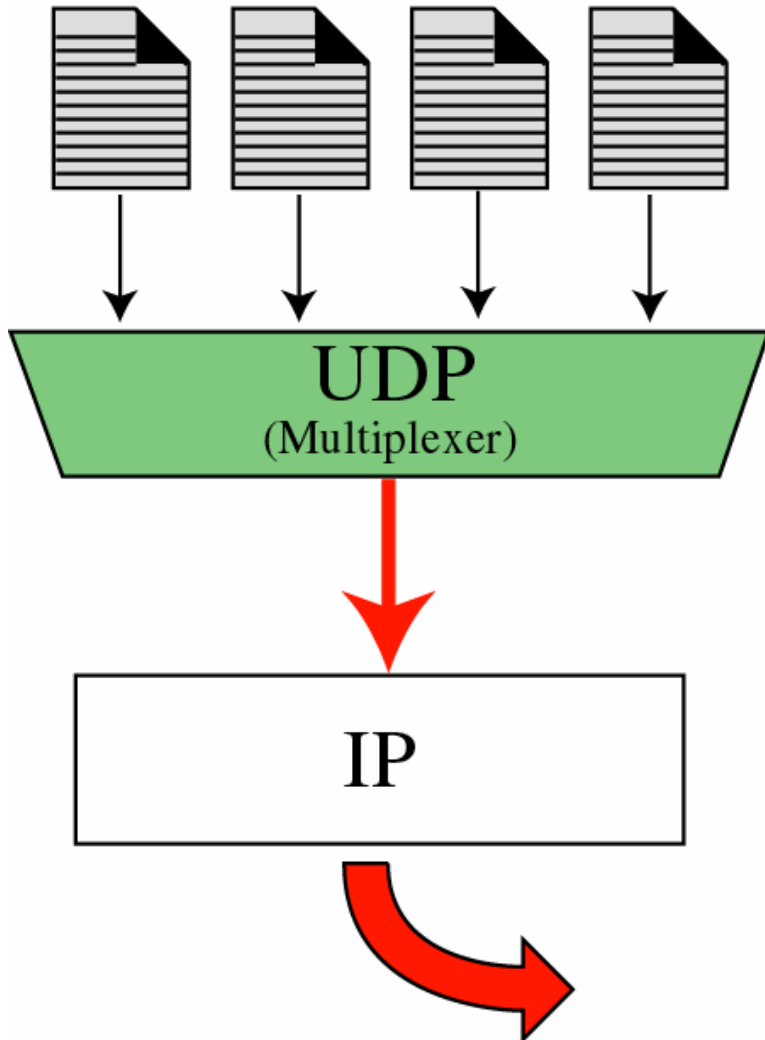
```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



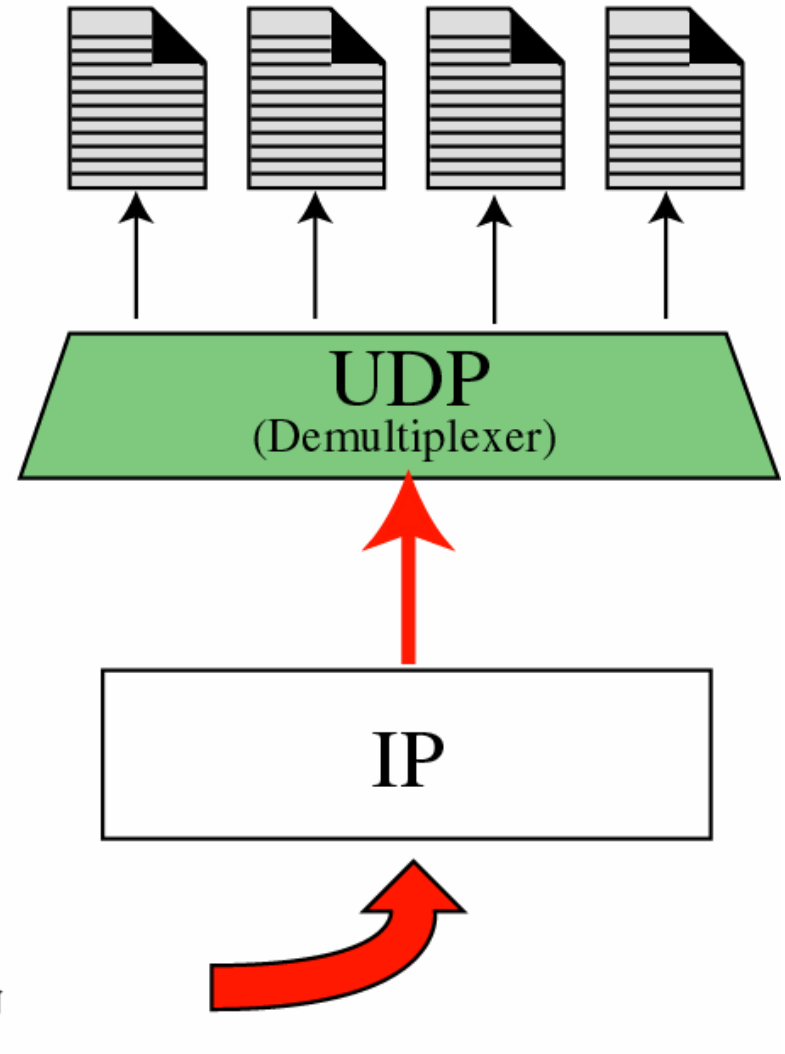
SP cung cấp "địa chỉ trở lại"

# Multiplexing và demultiplexing

Processes



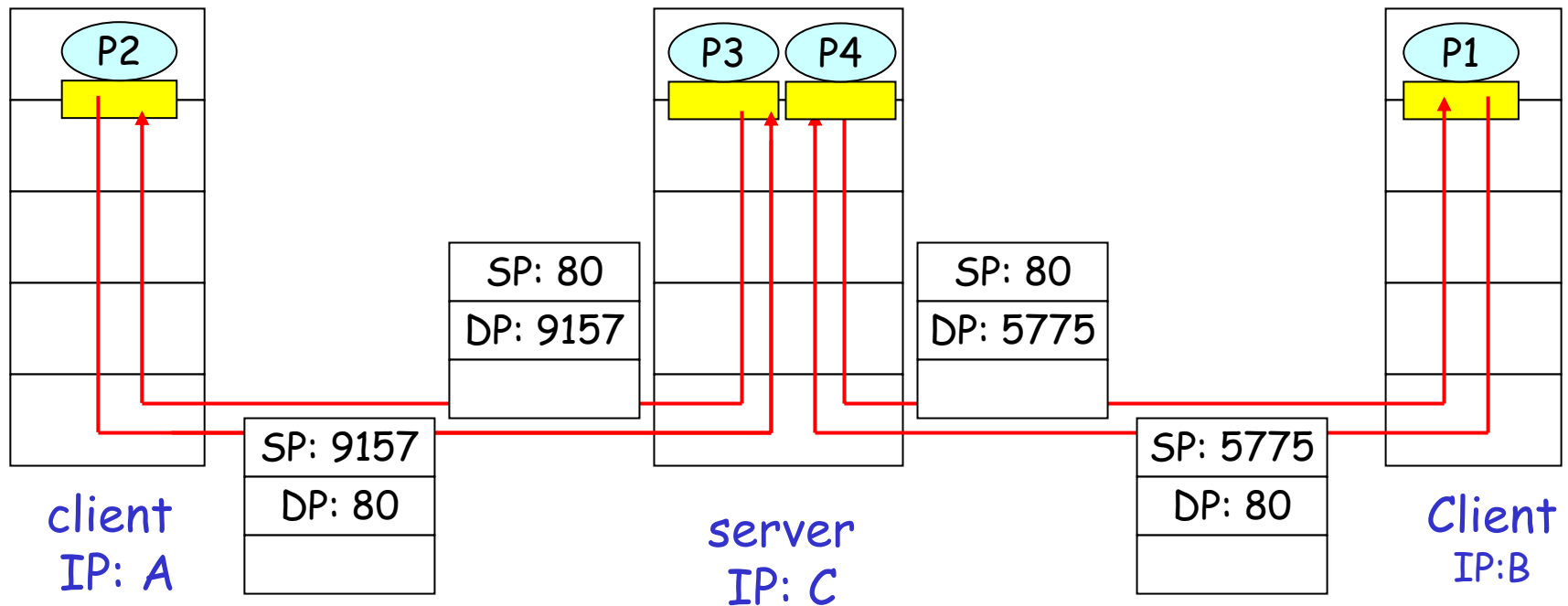
Processes



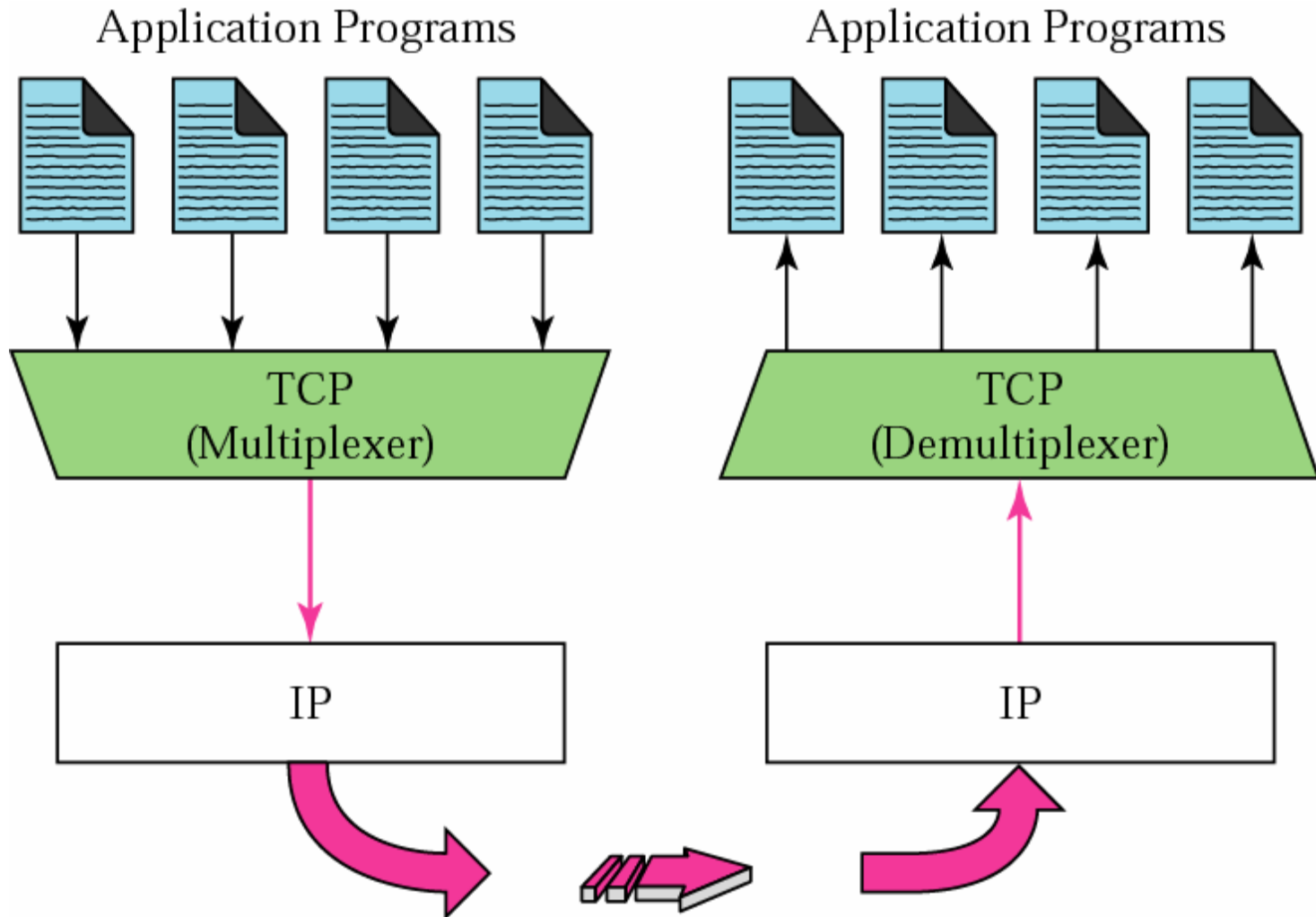
# Demultiplexing trong hướng kết nối

- ❑ TCP socket được định danh bằng bộ 4:
  - địa chỉ IP nguồn
  - số hiệu cổng nguồn
  - địa chỉ IP đích
  - số hiệu cổng đích
- ❑ trạm nhận sử dụng cả bốn giá trị trên để gửi segment đến socket thích hợp
- ❑ Máy chủ có thể hỗ trợ nhiều sockets TCP đồng thời:
  - Mỗi socket được định danh bằng bộ 4 của nó
- ❑ Web servers có các sockets khác nhau cho mỗi client đang kết nối
  - non-persistent HTTP sẽ có socket khác nhau cho mỗi yêu cầu

# Demultiplexing trong hướng kết nối (tiếp)



# Multiplexing và demultiplexing

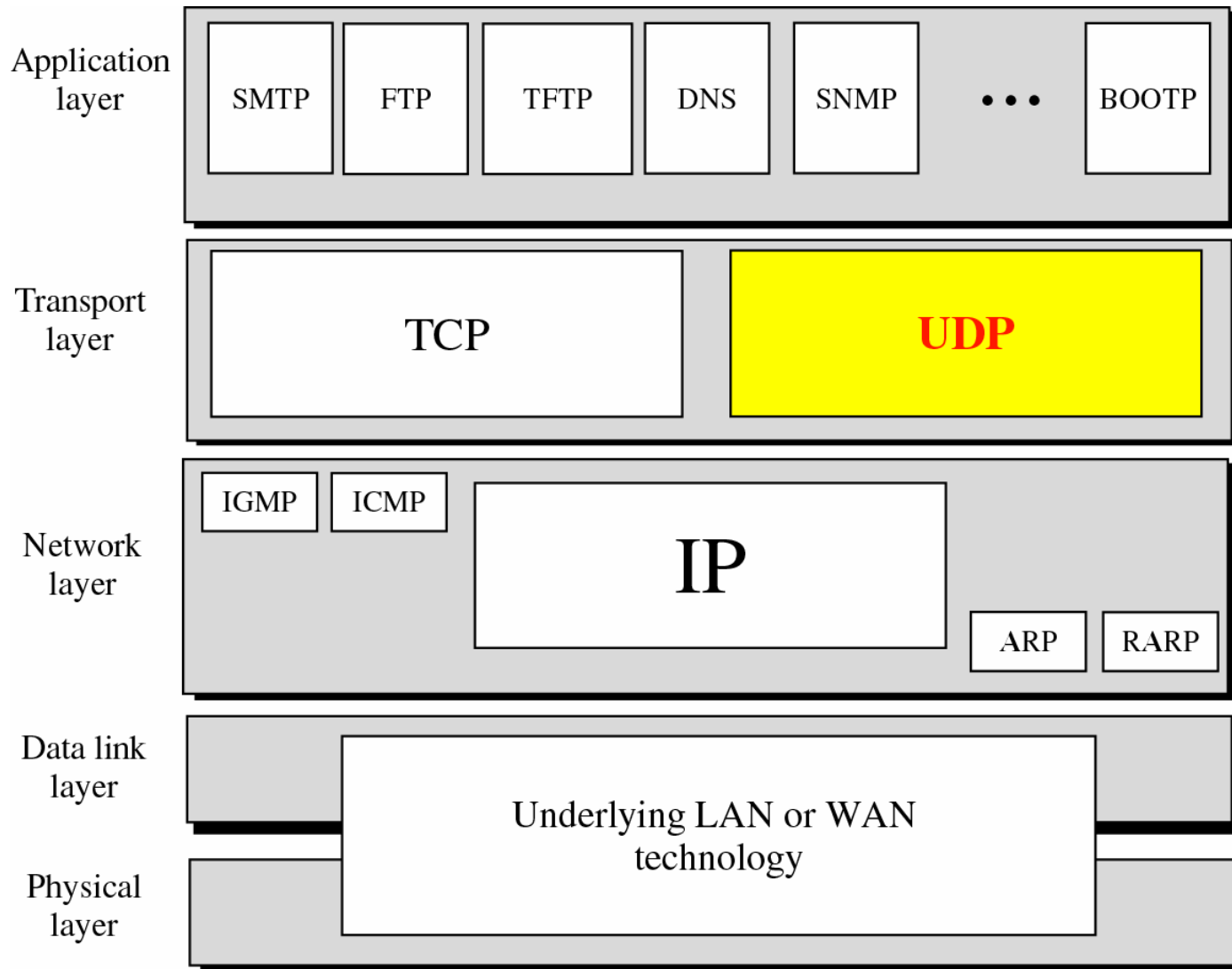


# Chương 7 - Nội dung

- ❑ 7.1 Các dịch vụ của tầng Vận chuyển
- ❑ 7.2 Multiplexing và demultiplexing
- ❑ 7.3 Vận chuyển phi kết nối: UDP
- ❑ 7.4 Vận chuyển hướng kết nối: TCP
  - cấu trúc segment
  - quản lý kết nối
  - truyền dữ liệu tin cậy
  - kiểm soát luồng
- ❑ 7.5 Kiểm soát tắc nghẽn trong TCP



# Vị trí của UDP trong chồng giao thức TCP/IP



# UDP: User Datagram Protocol [RFC 768]

- ❑ Là một giao thức vận chuyển “cơ bản” trên Internet
- ❑ dịch vụ “nỗ lực tối đa” service, UDP segments có thể bị:
  - mất
  - phân phát sai thứ tự đến tầng ứng dụng phía nhận
- ❑ *phi kết nối*:
  - không bắt tay giữa UDP gửi, nhận
  - mỗi UDP segment được xử lý độc lập với các segments khác

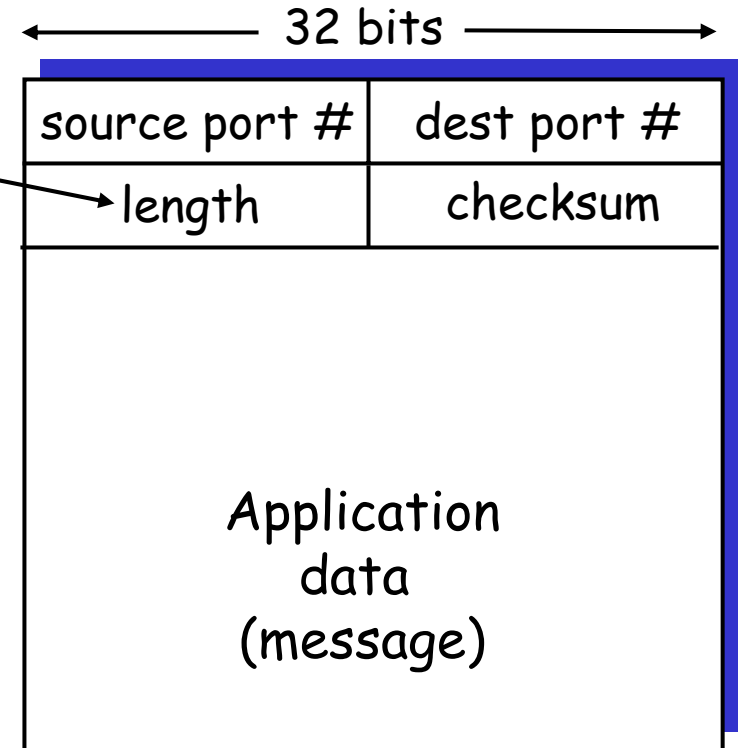
## Tại sao lại cần đến UDP?

- ❑ không thiết lập kết nối (kết nối làm tăng thêm độ trễ)
- ❑ đơn giản: không trạng thái kết nối tại bên gửi và bên nhận
- ❑ thông tin điều khiển của segment nhỏ
- ❑ không kiểm soát tắc nghẽn: UDP có thể đi nhanh nhất trong khả năng

# UDP: tiếp theo

- ❑ Thường được sử dụng cho các ứng dụng streaming multimedia
  - chấp nhận mất mát (loss tolerant)
  - nhạy về tốc độ (rate sensitive)
- ❑ Ví dụ về sử dụng UDP
  - DNS
  - SNMP
- ❑ vận chuyển tin cậy qua UDP: bổ sung sự tin cậy tại tầng ứng dụng
  - phục hồi lỗi tại ứng dụng cụ thể!

Chiều dài của UDP segment tính bằng byte, bao gồm cả header



UDP segment format

# UDP checksum

Mục đích: dò tìm "lỗi" (vd như các bits bị lật) trong các segments được truyền

## Bên gửi:

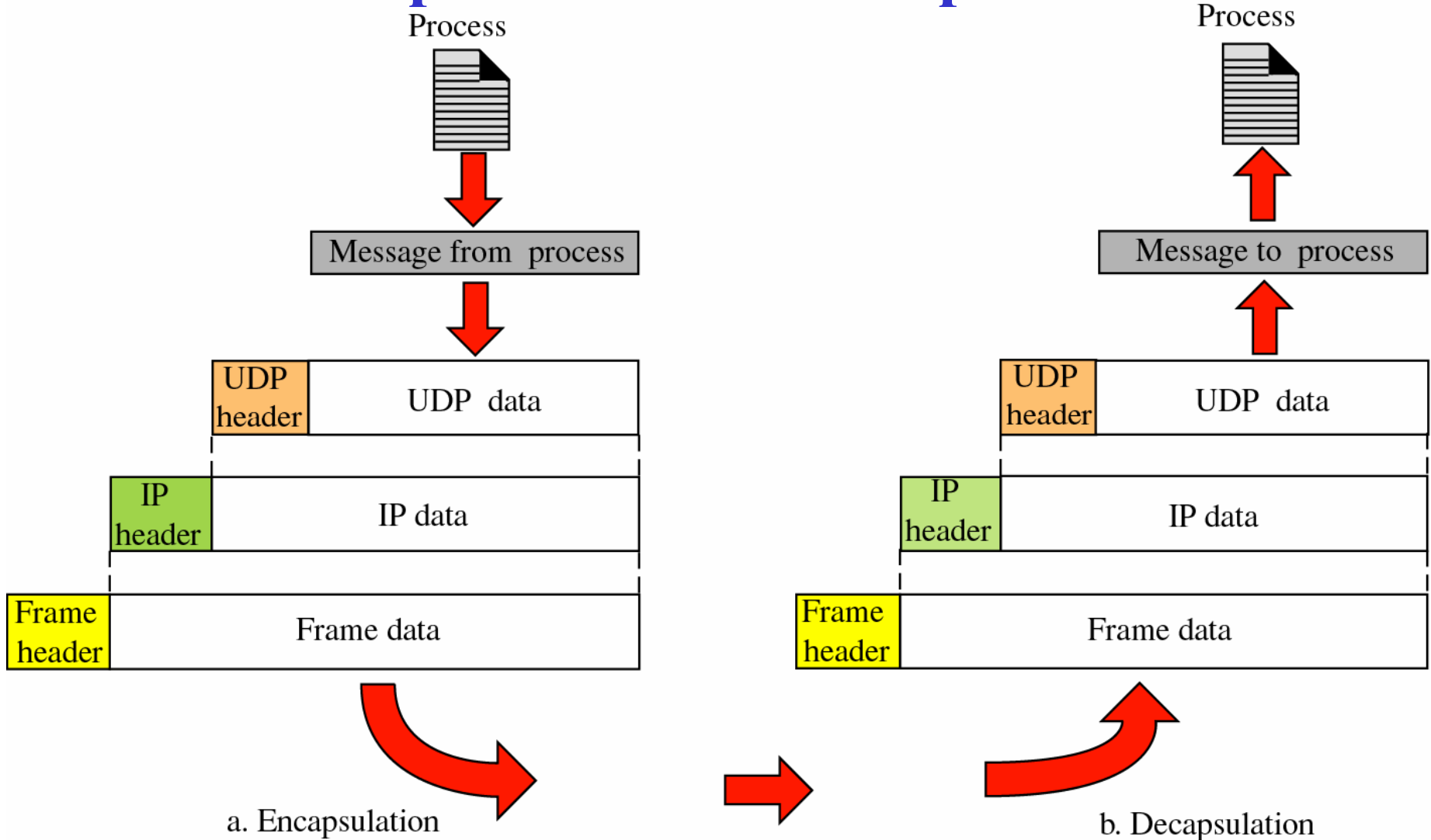
- ❑ xem nội dung của segment như là các số integers 16-bit
- ❑ checksum: tính tổng (tổng phần bù 1) nội dung của segment
- ❑ bên gửi đưa giá trị checksum vào trường checksum của UDP segment

## Bên nhận:

- ❑ tính checksum của segment nhận được
- ❑ kiểm tra xem số tính được có bằng giá trị trong trường checksum hay không:
  - NO - lỗi bị phát hiện
  - YES - không có lỗi bị phát hiện. *Nhưng vẫn có thể có lỗi?*

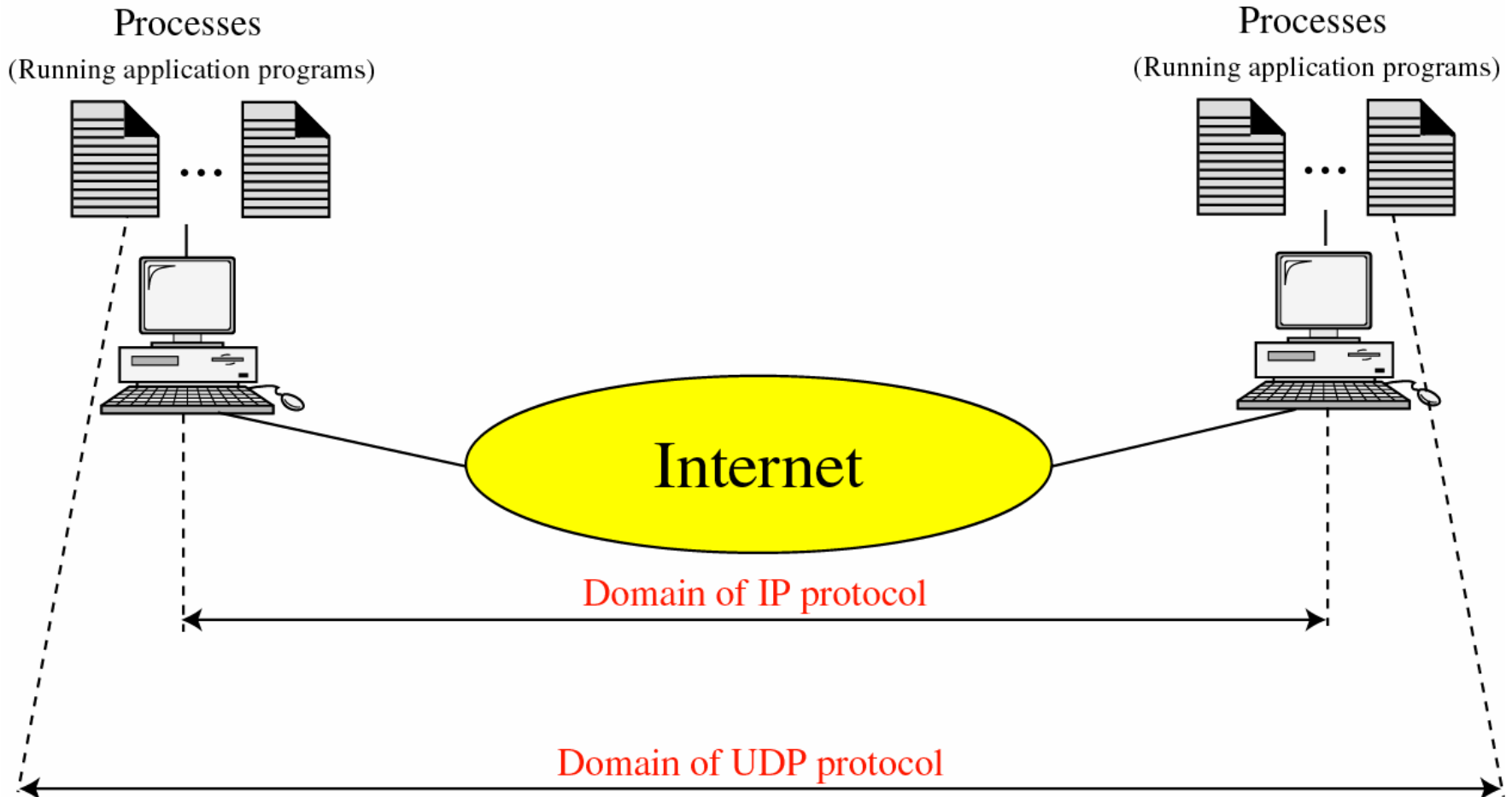
# Encapsulation và decapsulation

## Encapsulation and decapsulation

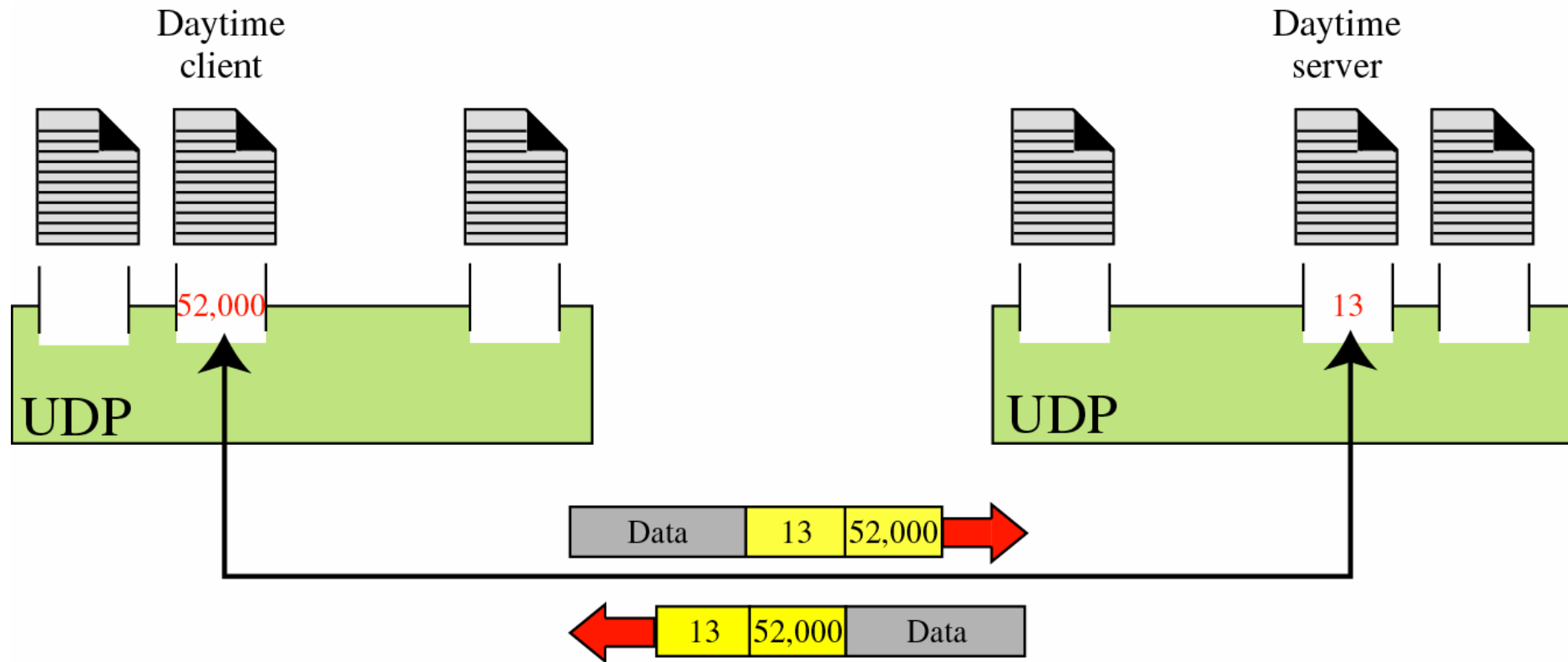


# UDP so với IP

## UDP versus IP

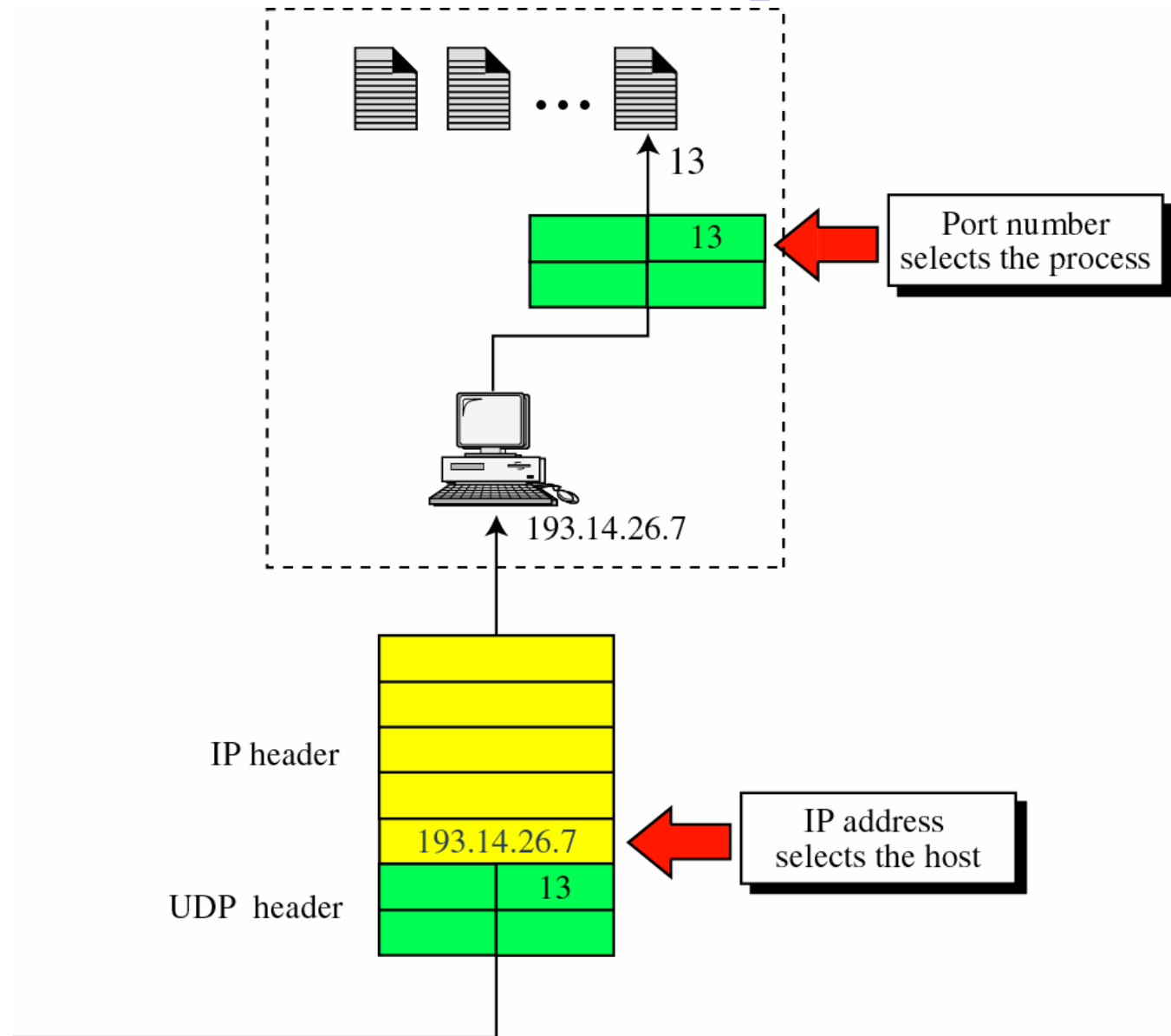


# Số hiệu cổng Port numbers



# Địa chỉ IP so với số hiệu cổng

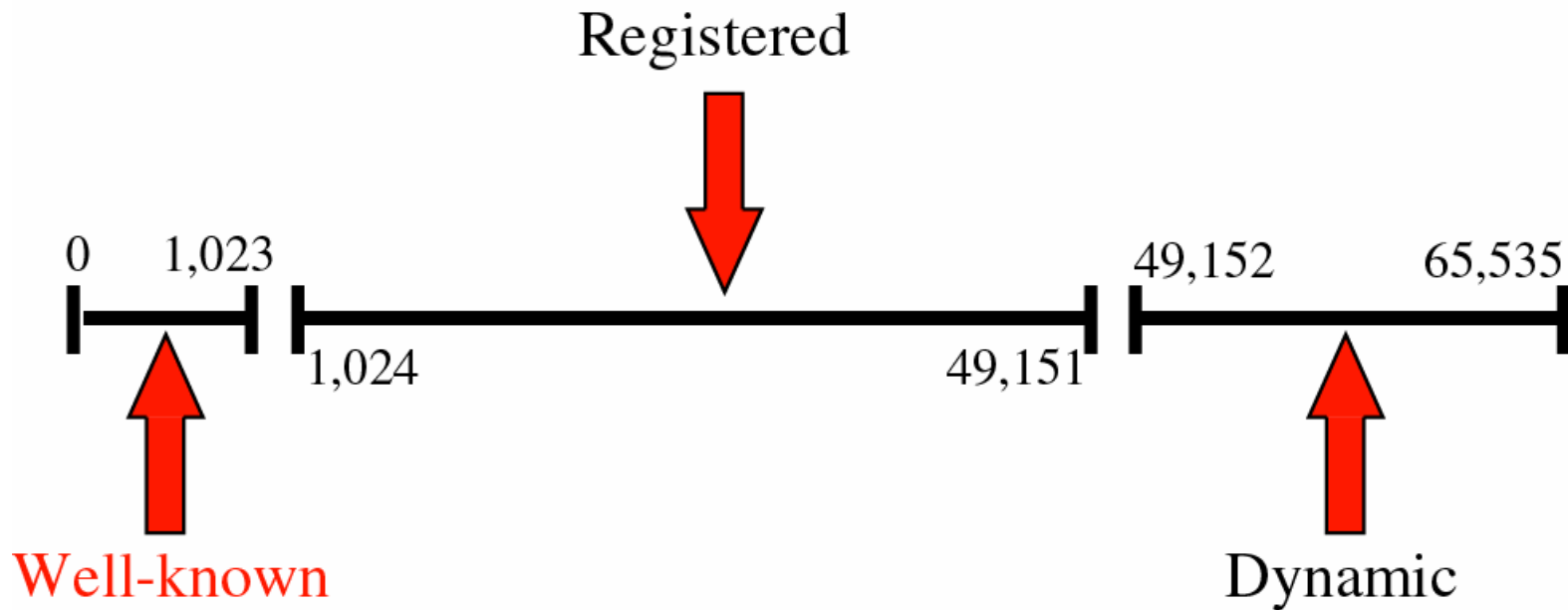
## IP addresses versus port numbers





# Dải số hiệu cổng do IANA quy định

## IANA ranges

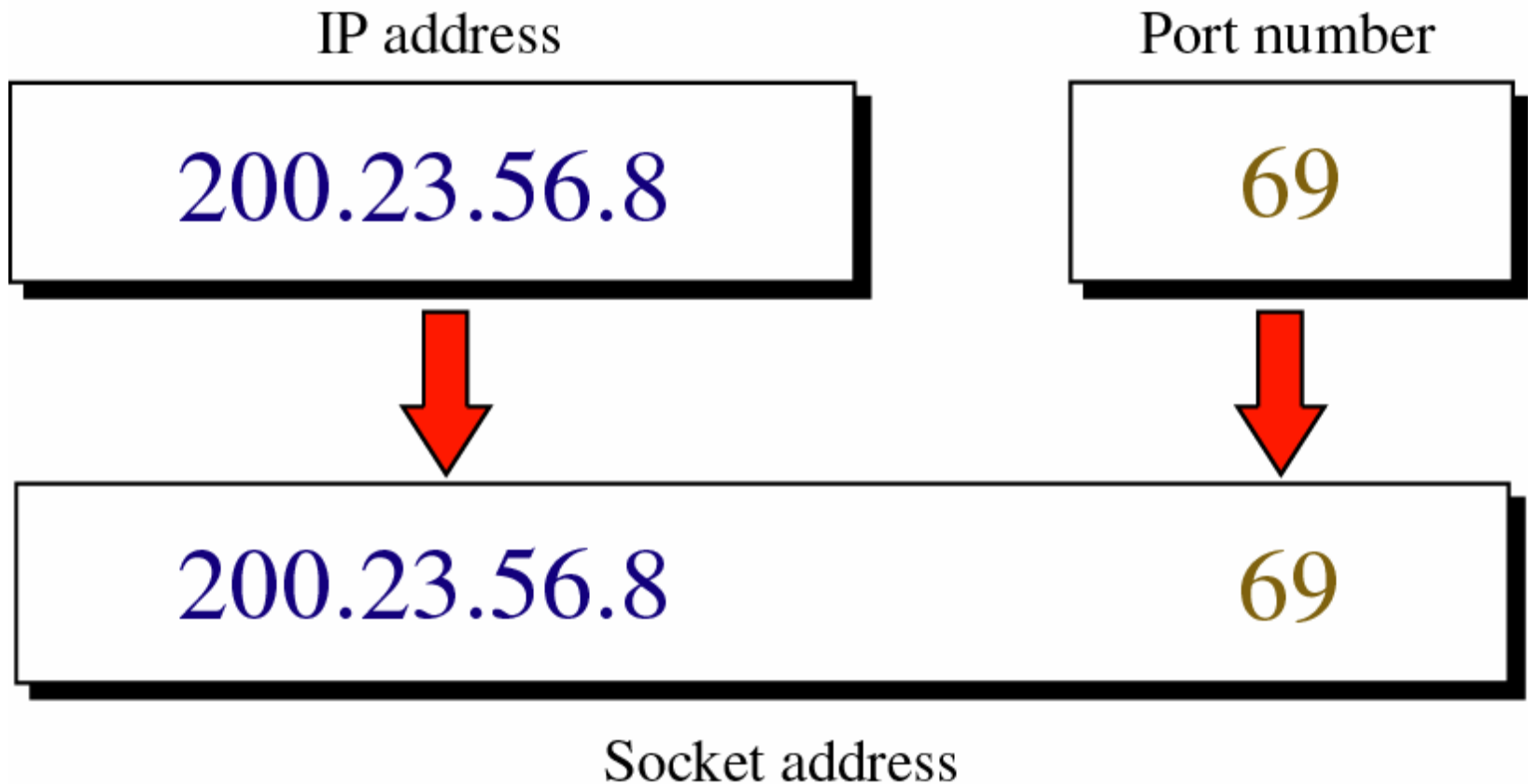


More later!

IANA: Internet Assigned Numbers Authority

# Địa chỉ Socket

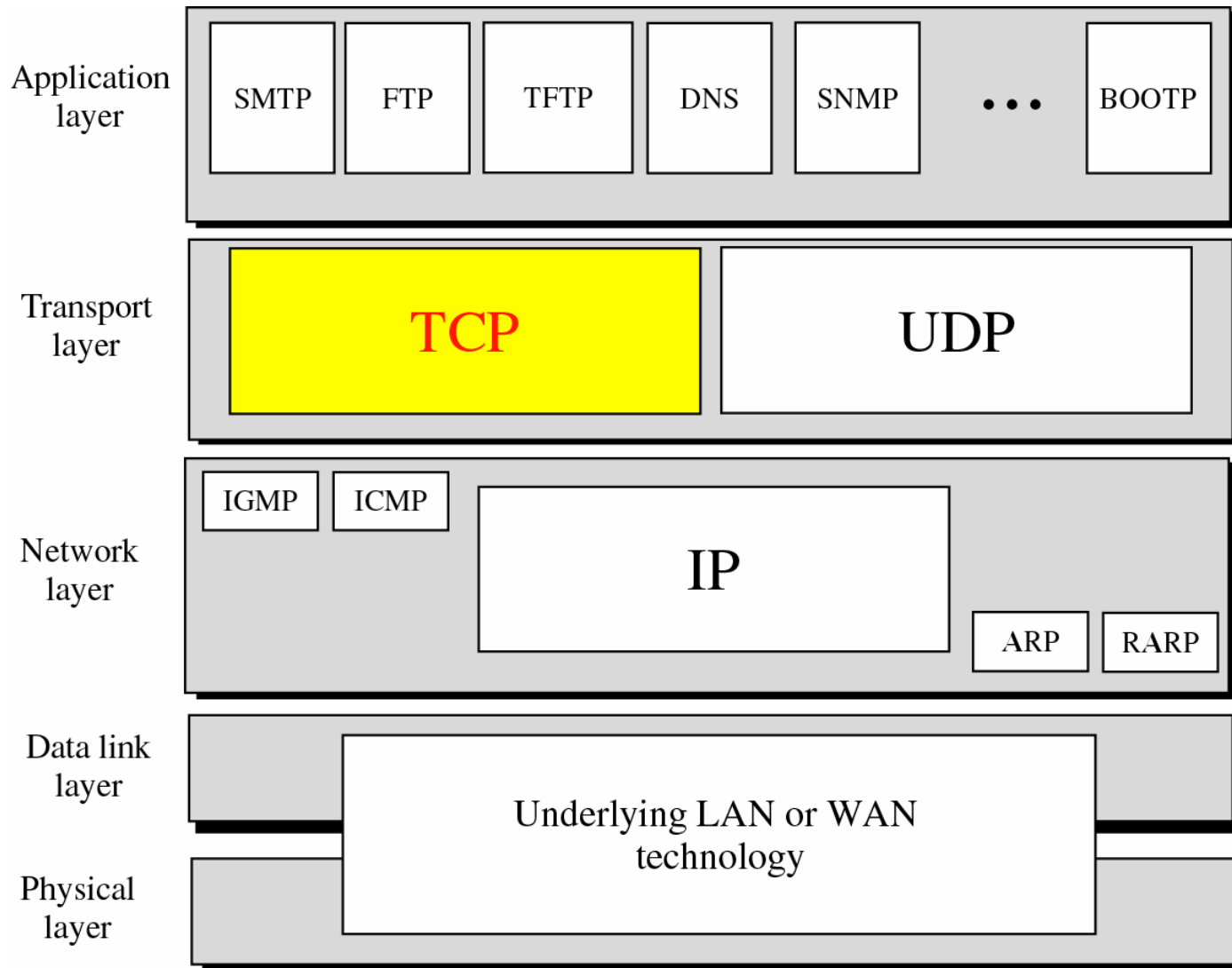
## Socket addresses



# Chương 7 - Nội dung

- ❑ 7.1 Các dịch vụ của tầng Vận chuyển
- ❑ 7.2 Multiplexing và demultiplexing
- ❑ 7.3 Vận chuyển phi kết nối: UDP
- ❑ 7.4 Vận chuyển hướng kết nối: TCP
  - cấu trúc segment
  - quản lý kết nối
  - truyền dữ liệu tin cậy
  - kiểm soát luồng
- ❑ 7.5 Kiểm soát tắc nghẽn trong TCP

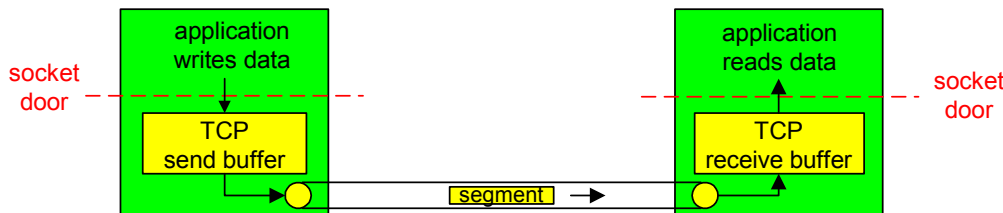
# Vị trí của TCP trong chồng giao thức TCP/IP



# TCP: Tổng quan

RFCs: 793, 1122, 1323, 2018, 2581

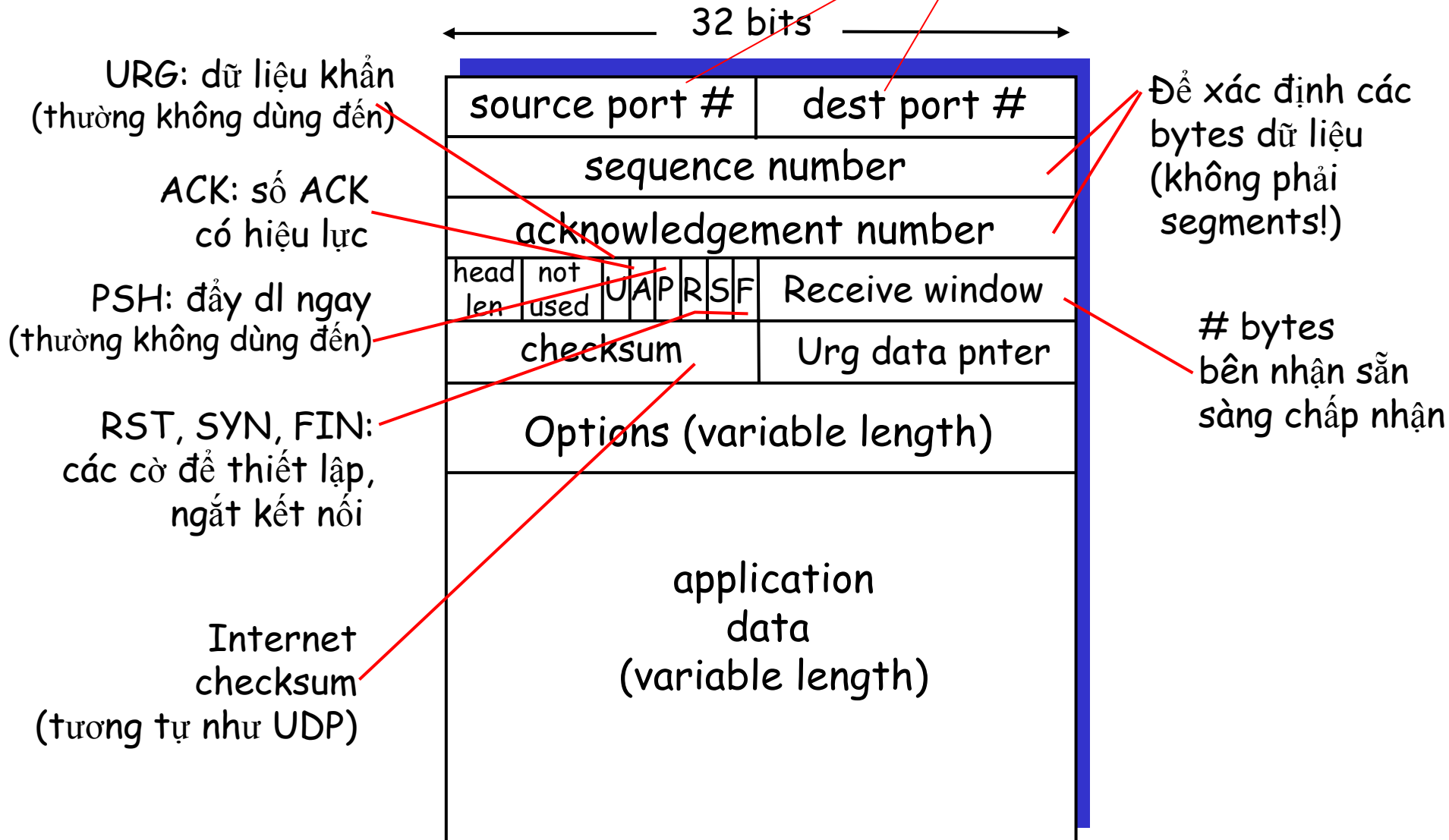
- ❑ **điểm-đến-điểm:**
  - một bên gửi, một bên nhận
- ❑ **tin cậy, dòng byte có thứ tự:**
  - không “ranh giới thông điệp”
- ❑ **đường ống dẫn:**
  - kiểm soát tắc nghẽn trong TCP và kiểm soát luồng đặt kích cỡ cửa sổ
- ❑ **vùng đệm gửi và nhận**



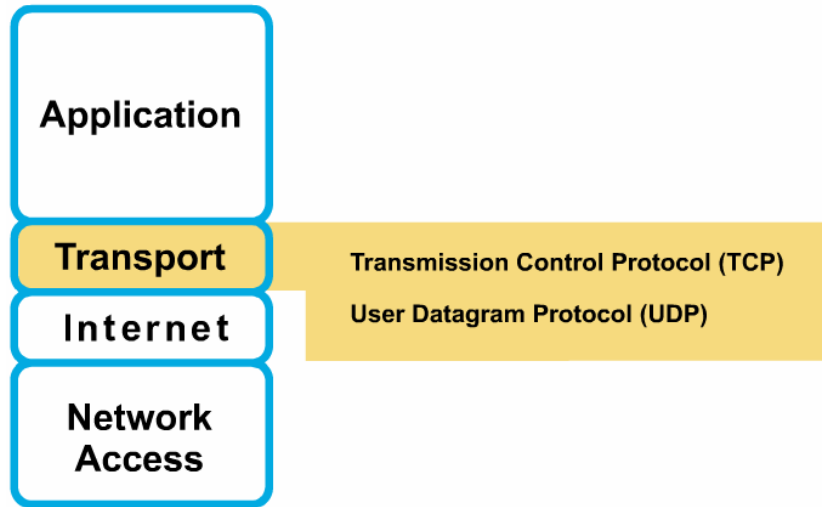
- ❑ **dữ liệu song công hoàn toàn:**
  - Luồng dữ liệu hai hướng trên cùng kết nối
  - MSS: maximum segment size
- ❑ **hướng kết nối:**
  - bắt tay (trao đổi các thông điệp điều khiển) giữa bên gửi và bên nhận trước khi trao đổi dữ liệu
- ❑ **kiểm soát luồng:**
  - bên gửi sẽ không làm ngập/lụt bên nhận

# Cấu trúc TCP segment

giúp định danh các điểm đầu cuối của kết nối



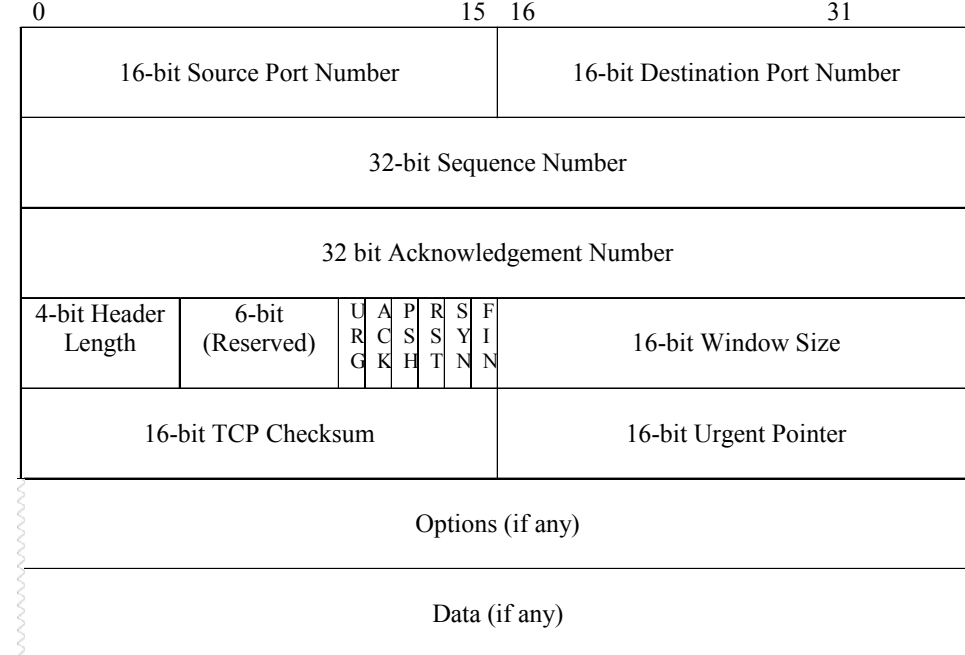
# Transport Layer Overview



0				15				16				31			
16-bit Source Port Number								16-bit Destination Port Number							
32-bit Sequence Number															
32 bit Acknowledgement Number															
4-bit Header Length		6-bit (Reserved)		U	A	P	R	S	F	16-bit Window Size					
				R	C	S	S	Y	I						
				G	K	H	T	N	N						
16-bit TCP Checksum								16-bit Urgent Pointer							
Options (if any)															
Data (if any)															

- ❑ **source port** - số hiệu cổng nguồn (bên gọi)
- ❑ **destination port** - số hiệu cổng đích (bên được gọi)
- ❑ **sequence number** - số chuỗi, được sử dụng để đảm bảo dữ liệu đến đúng theo thứ tự
- ❑ **acknowledgment number** - byte tiếp theo mà bên nhận đang đợi
- ❑ **HLEN** - chiều dài header, tính bằng đơn vị từ (32 bits)
- ❑ **reserved** - được đặt là 0
- ❑ **code bits** - chức năng điều khiển (vd: thiết lập và kết thúc một kết nối)
- ❑ **window** - số octets mà bên nhận đang sẵn sàng tiếp nhận
- ❑ **checksum** - tổng kiểm tra của phần thông tin điều khiển và dữ liệu
- ❑ **urgent pointer** - chỉ vị trí kết thúc của dữ liệu khẩn
- ❑ **option** - ví dụ về một option được định nghĩa: maximum TCP segment size
- ❑ **data** - dữ liệu giao thức của tầng trên

# TCP Segment Header



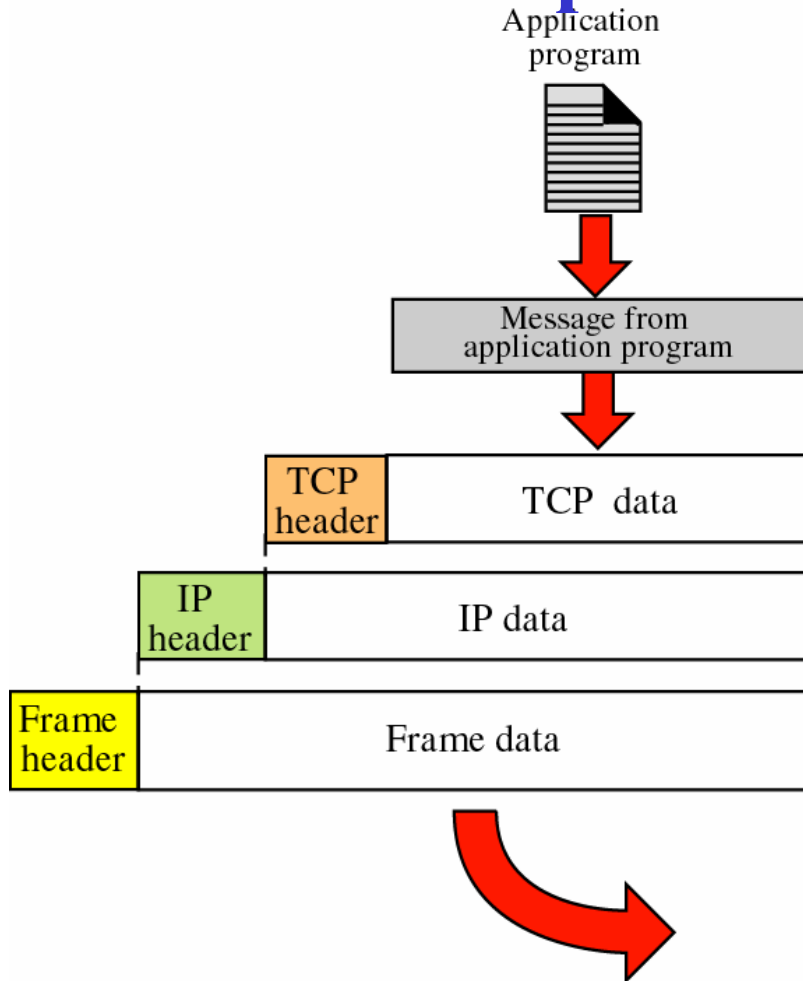
## *TCP (Transmission Control Protocol)*

- ❑ Giao thức hướng kết nối, tin cậy
- ❑ Cung cấp:
  1. **kiểm soát luồng** được cung cấp thông qua cơ chế cửa sổ trượt,
  2. **tin cậy** by được cung cấp thông qua các số chuỗi và cơ chế hồi báo (ACKs).
- ❑ TCP gọi lại bất cứ đoạn dữ liệu nào mà bên kia không nhận được và cung cấp một **kênh ảo** giữa các ứng dụng của các hệ thống đầu cuối.
- ❑ Ưu điểm của TCP là nó cung cấp dịch vụ **phân phát đảm bảo** các segments.

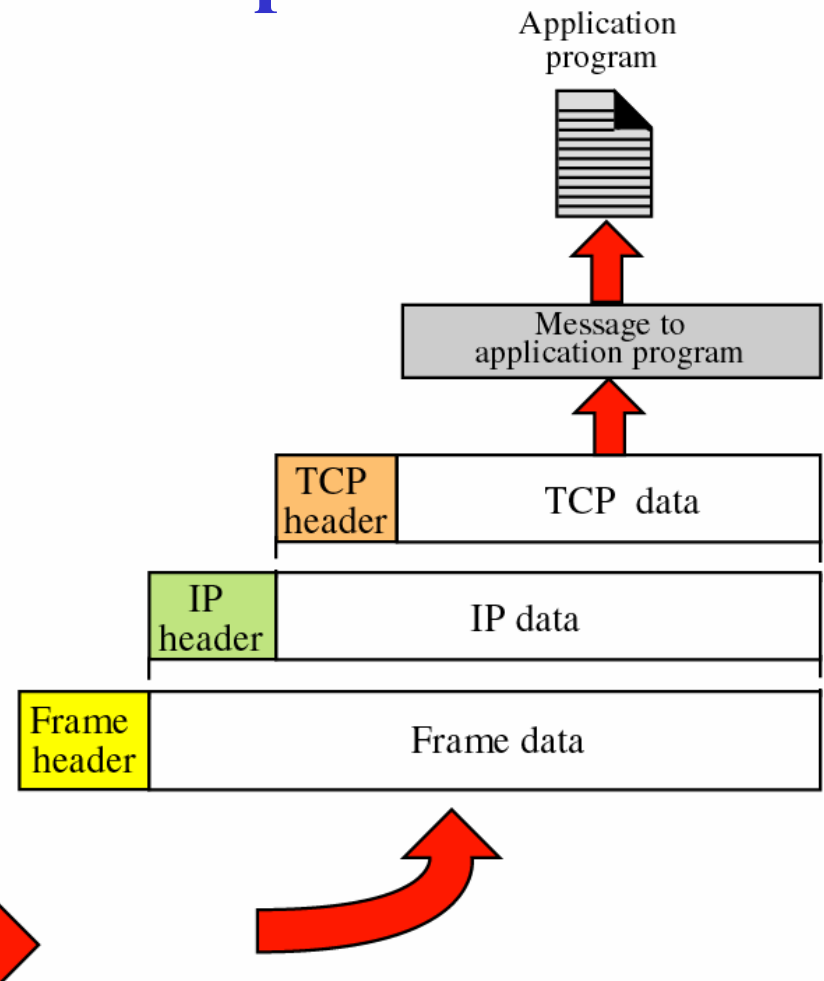


# Encapsulation và decapsulation

## Encapsulation and decapsulation

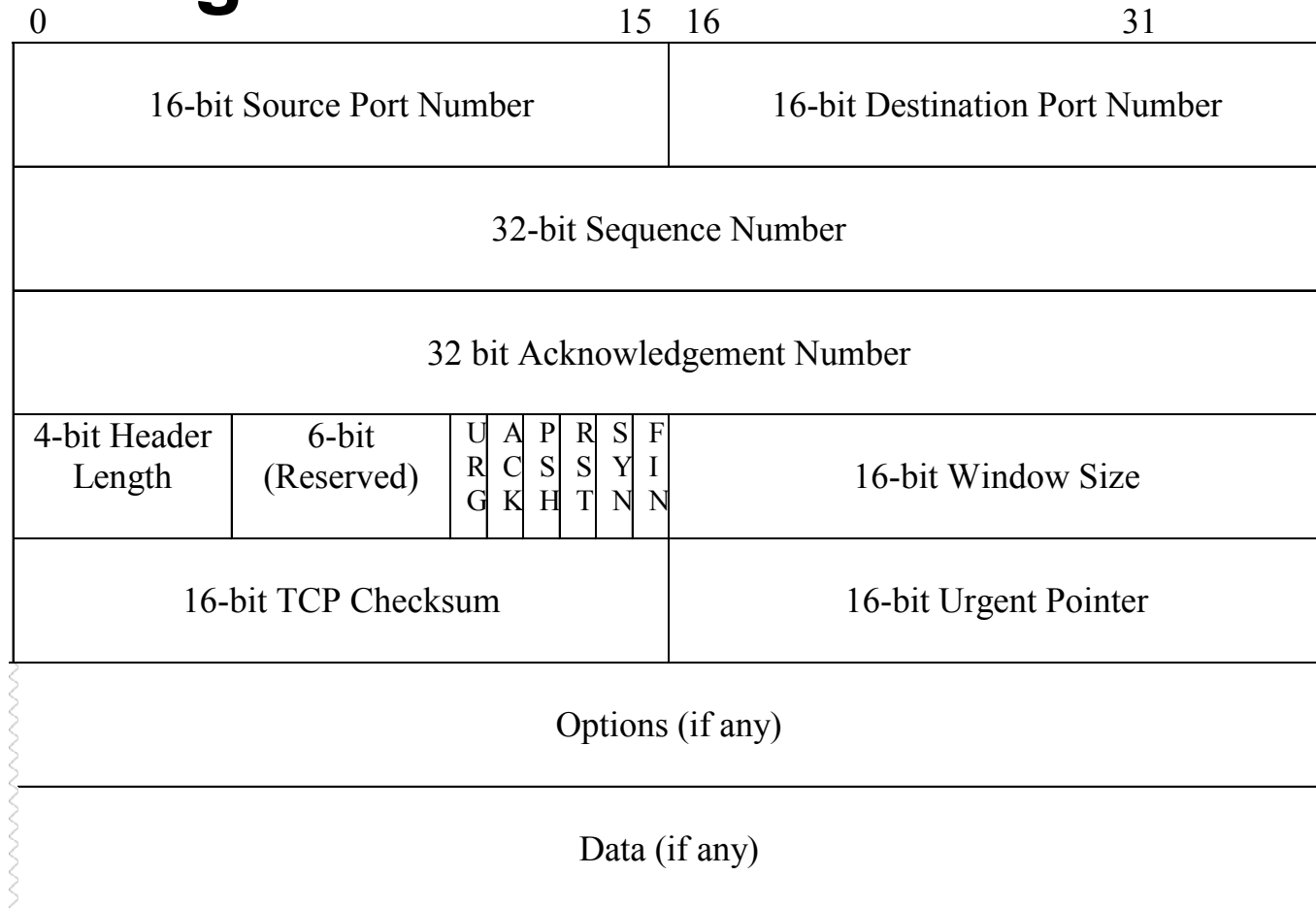


a. Encapsulation



b. Decapsulation

# TCP Segment Header



**Một số giao thức sử dụng TCP ở tầng vận chuyển:**

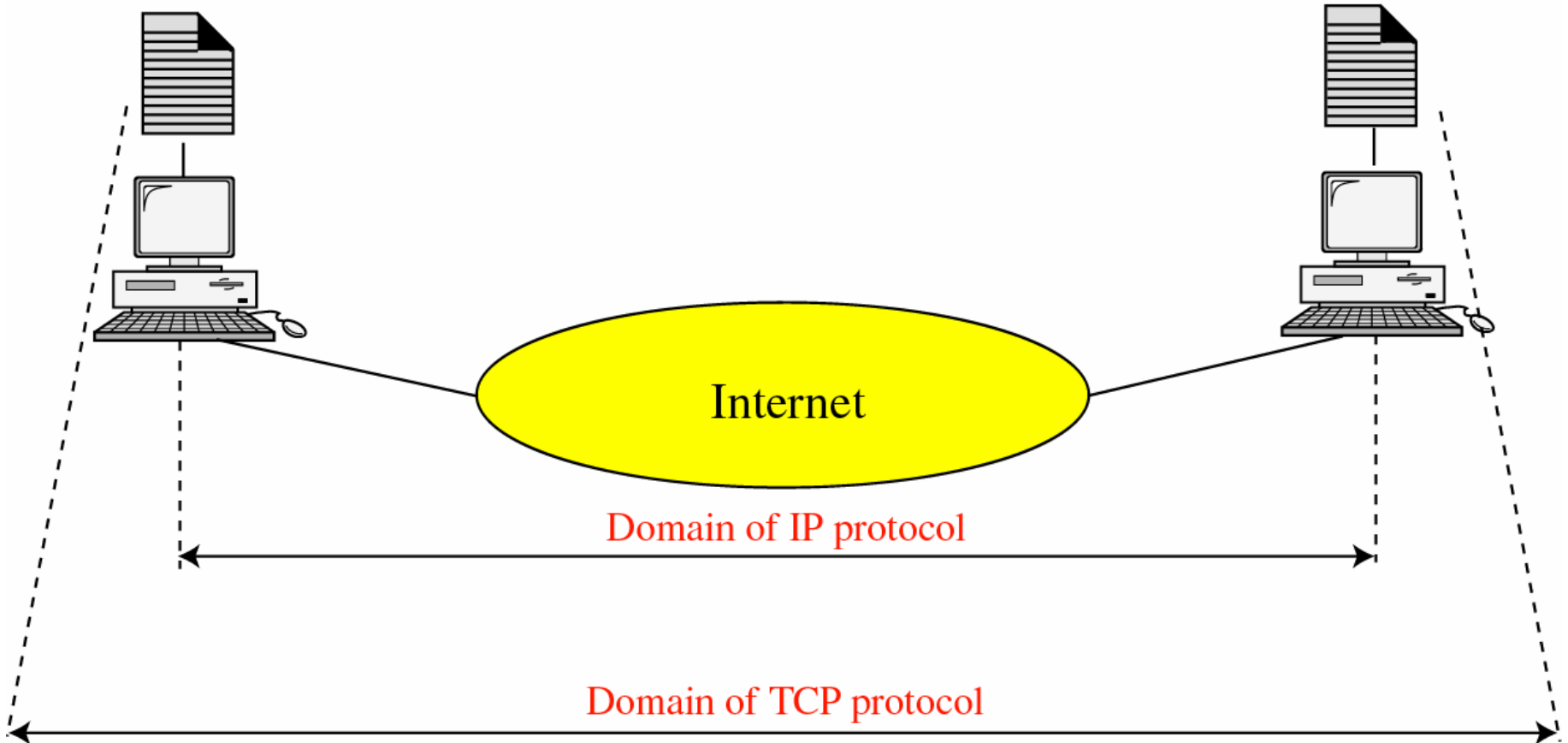
- ☐ HTTP
- ☐ Telnet
- ☐ FTP

# TCP so với IP

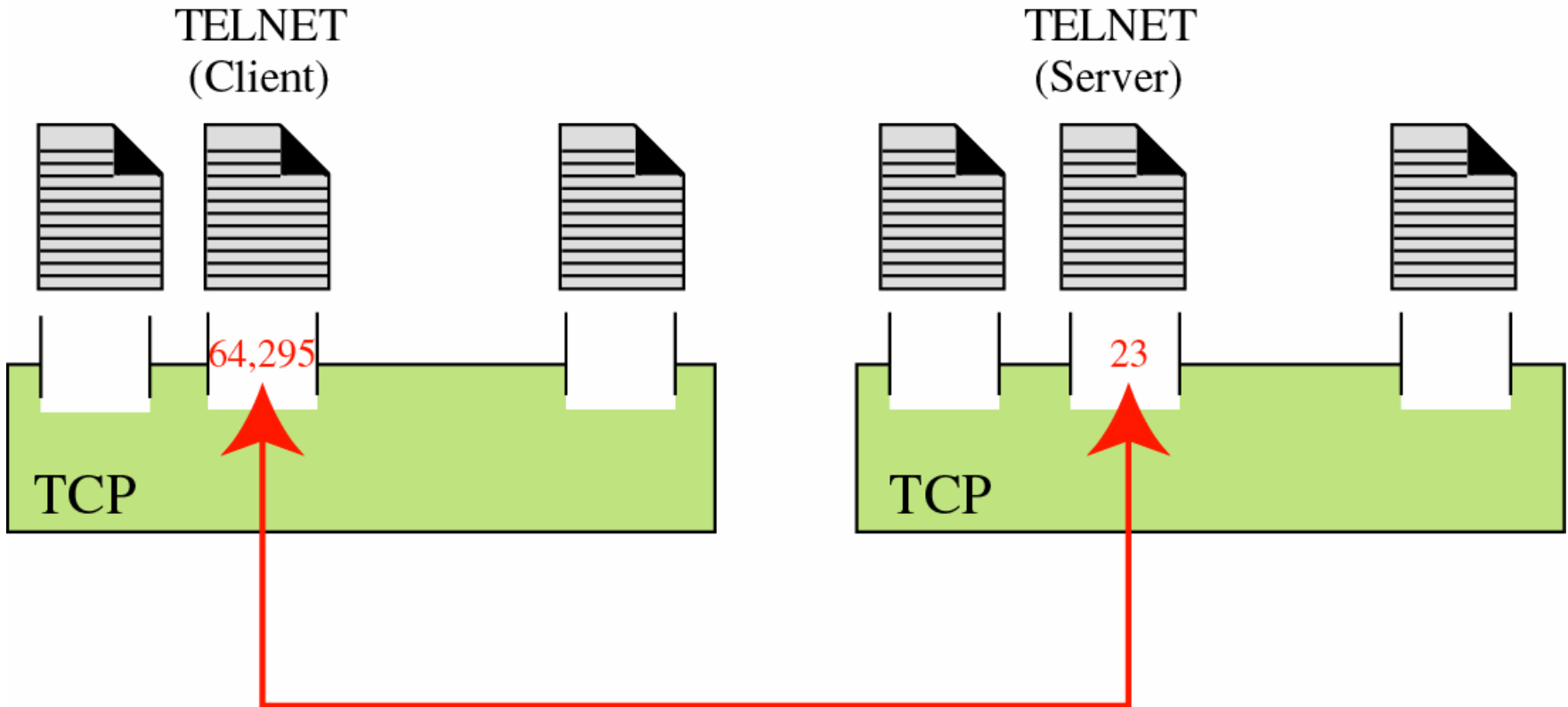
## TCP versus IP

Application program  
(Process)

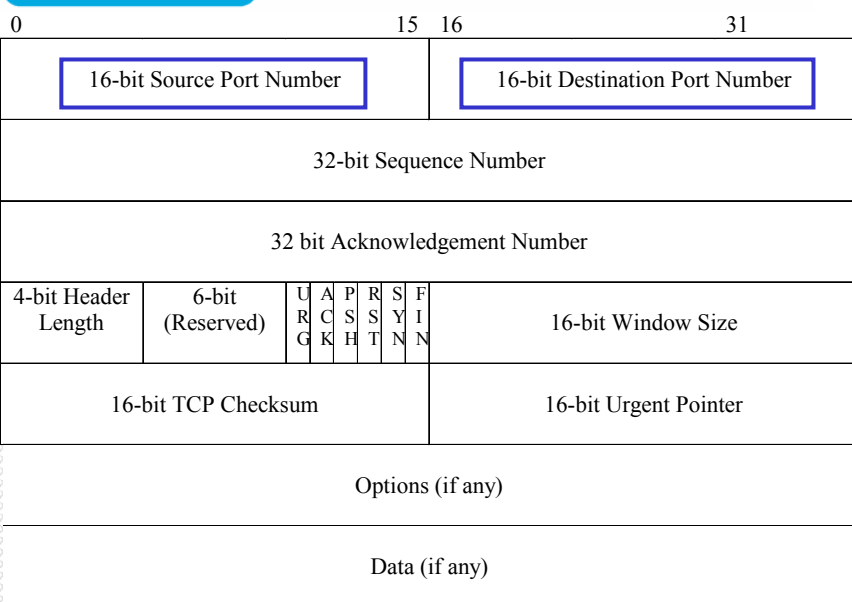
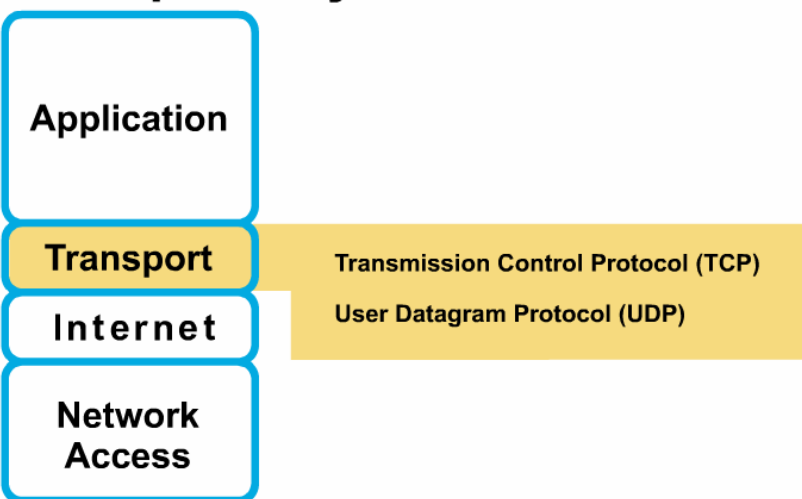
Application program  
(Process)



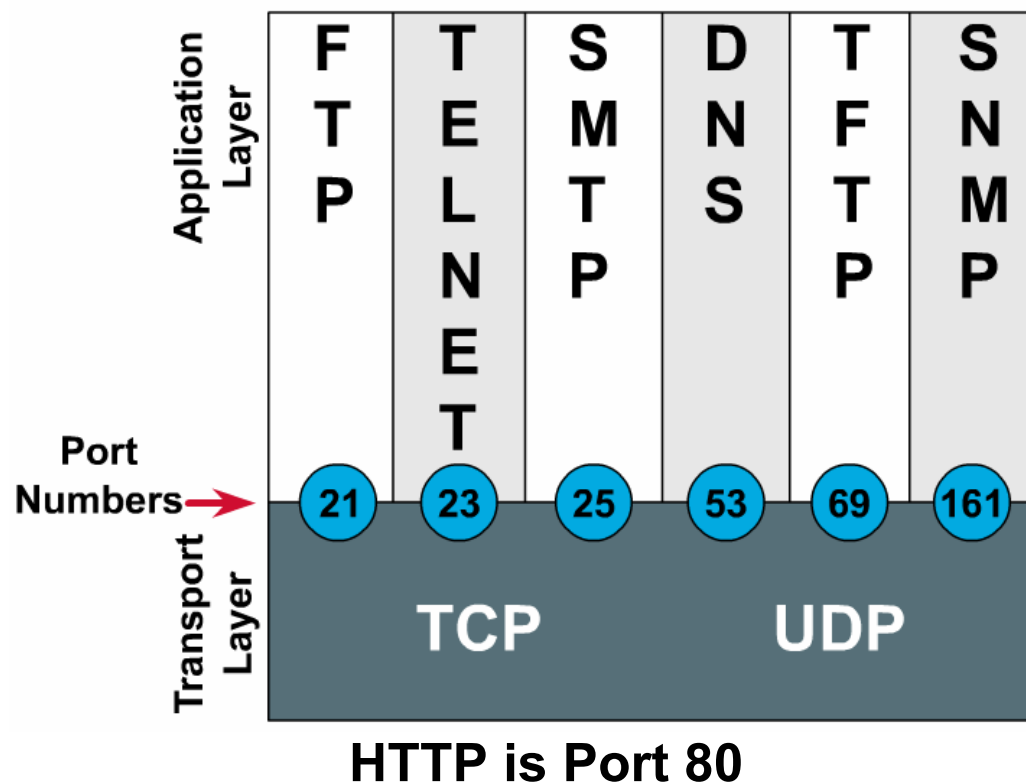
# Số hiệu cổng Port numbers



# Transport Layer Overview

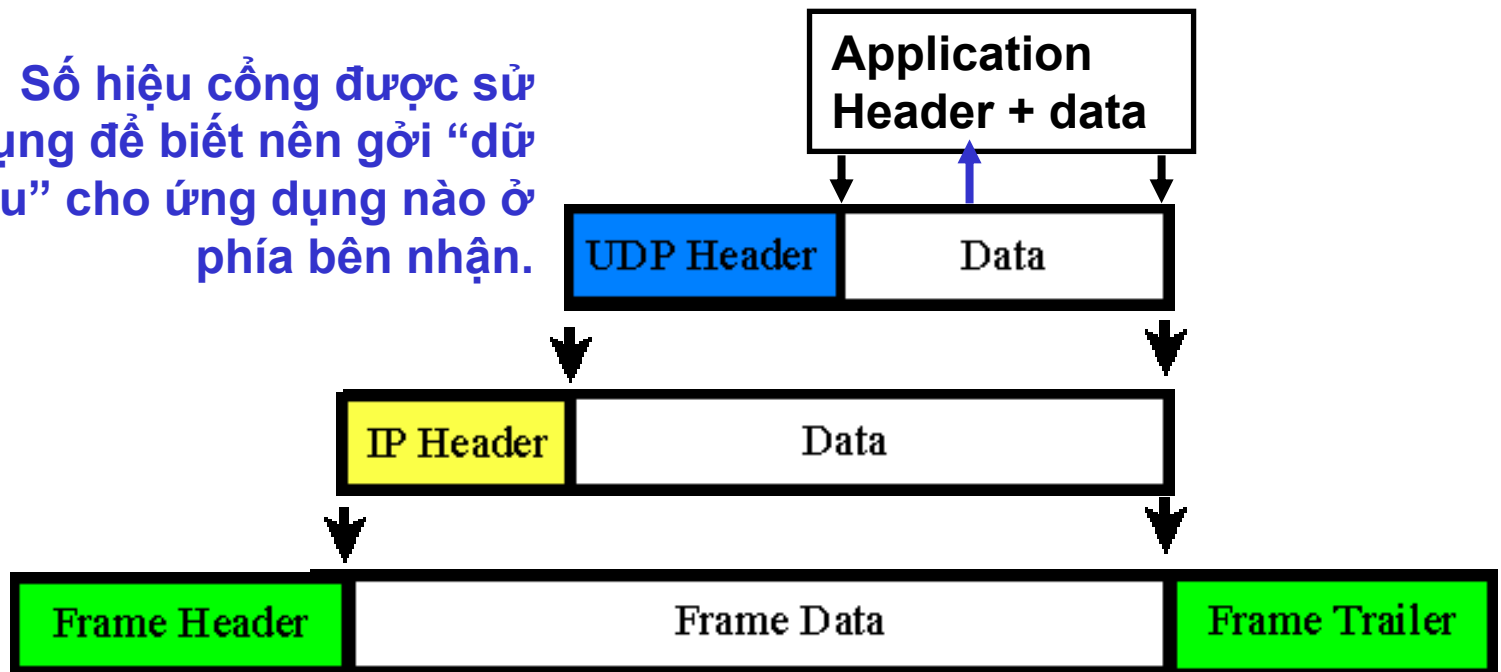


# Port Numbers

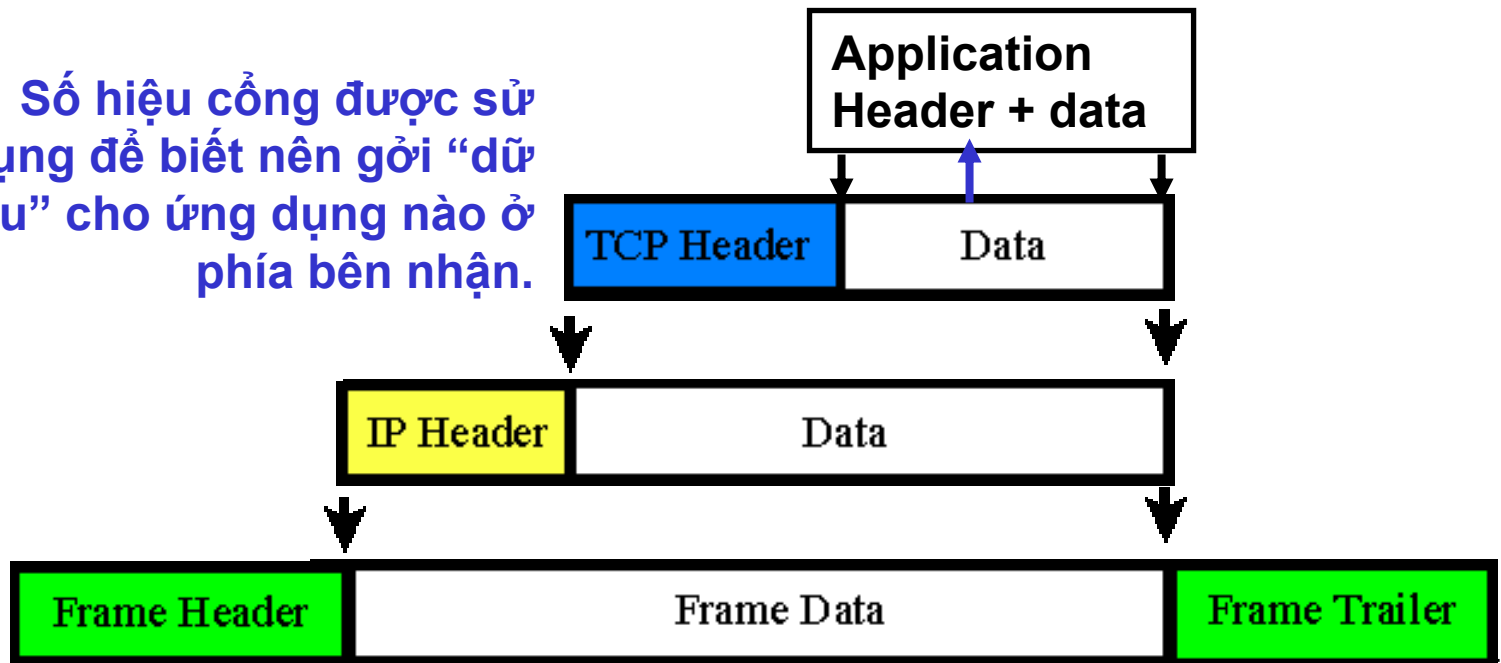


❑ Cả hai TCP và UDP sử dụng số hiệu cổng để chuyển thông tin lên tầng trên.

Số hiệu cổng được sử dụng để biết nên gửi “dữ liệu” cho ứng dụng nào ở phía bên nhận.

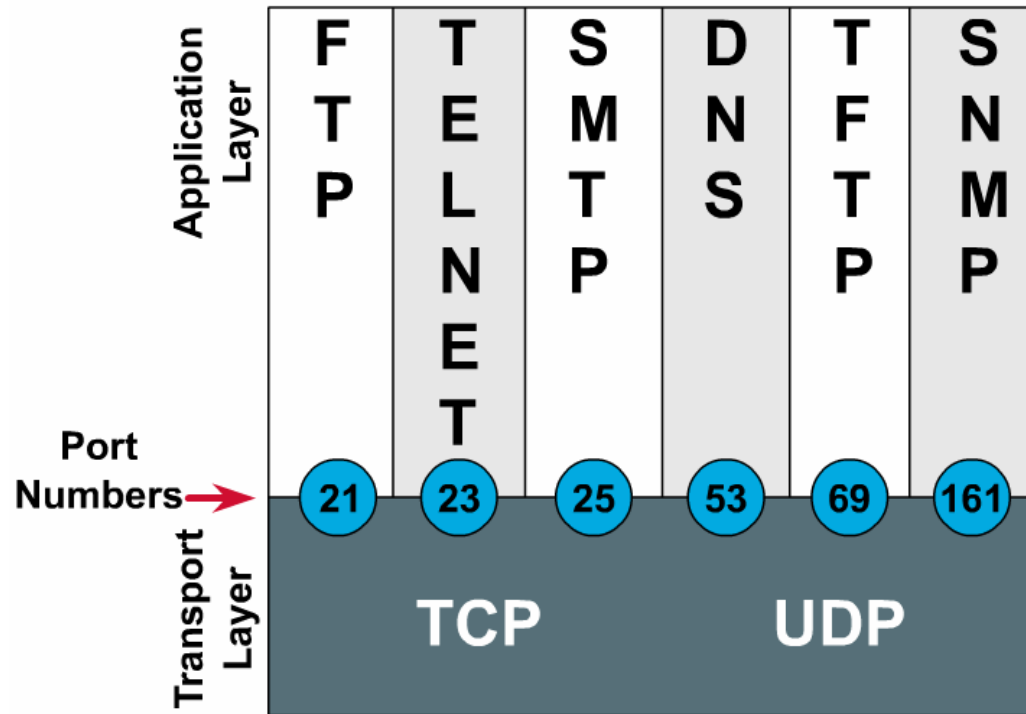
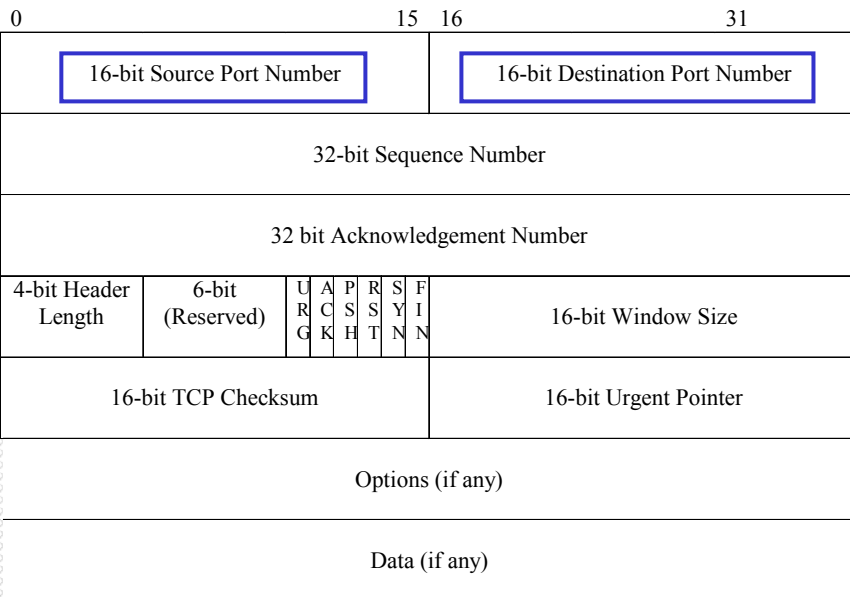


Số hiệu cổng được sử dụng để biết nên gửi “dữ liệu” cho ứng dụng nào ở phía bên nhận.



# Port Numbers

## TCP Header

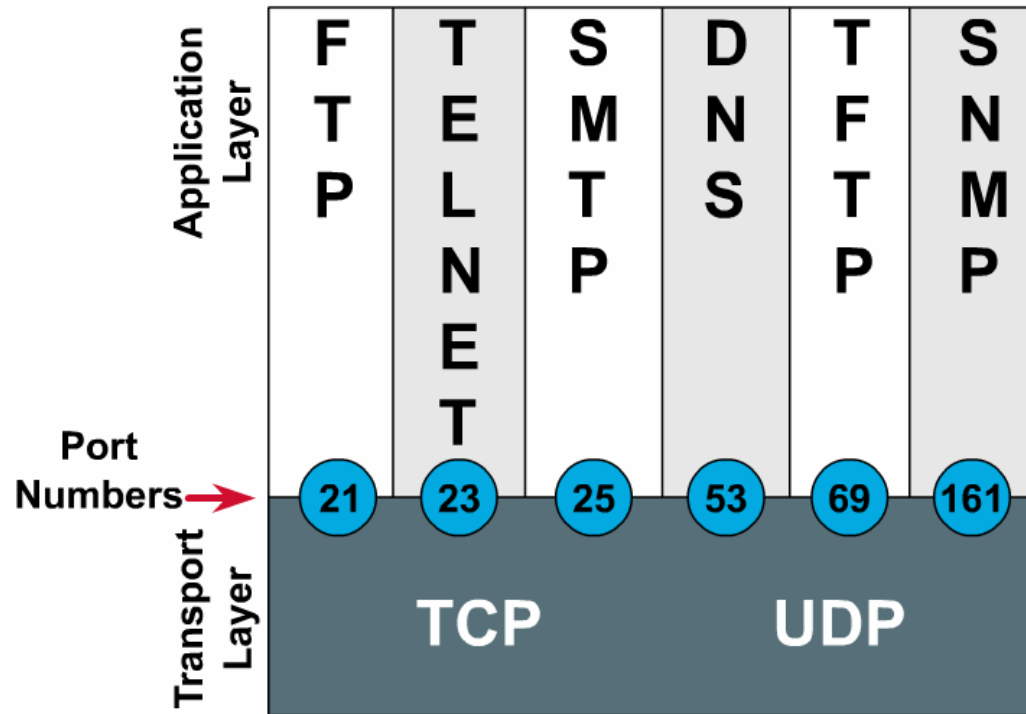
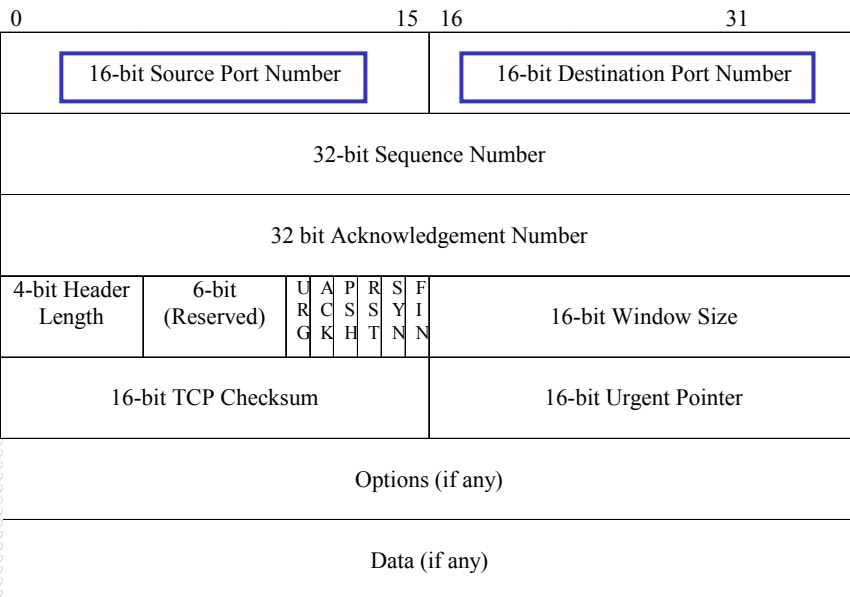


HTTP is Port 80

- ❑ Các nhà phát triển phần mềm ứng dụng đã thống nhất với nhau trong việc sử dụng các **well-known port numbers (cổng thông dụng)** được định nghĩa trong RFC 1700.
- ❑ Ví dụ, bất kỳ hội thoại nào gắn với một ứng dụng **FTP** đều sử dụng cổng chuẩn là **21**.

# Port Numbers

## TCP Header



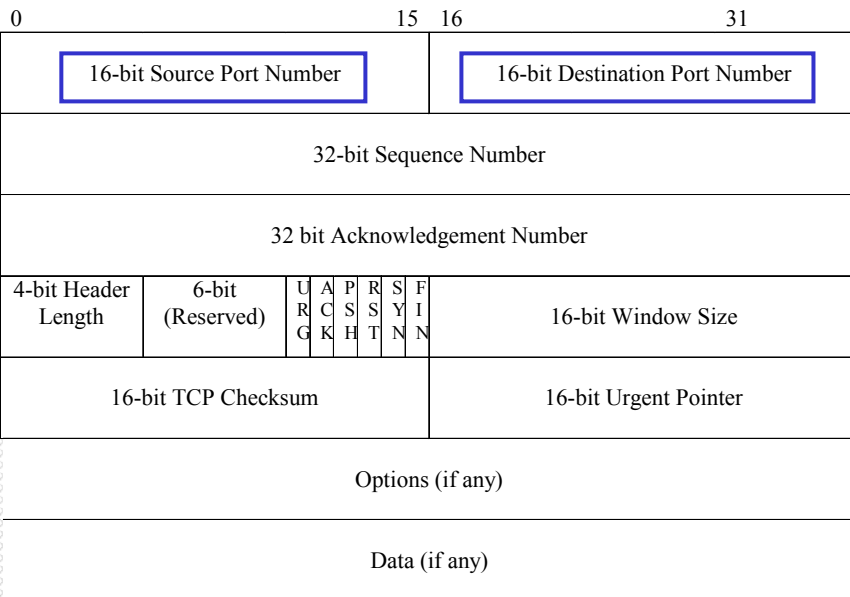
**HTTP is Port 80**

- ❑ Các hội thoại mà không bao gồm một ứng dụng với một well-known port number thì được gán số hiệu cổng ngẫu nhiên, được lựa chọn từ một dải cụ thể.
- ❑ Những số hiệu cổng này được sử dụng như là địa chỉ nguồn và địa chỉ đích trong TCP segment.

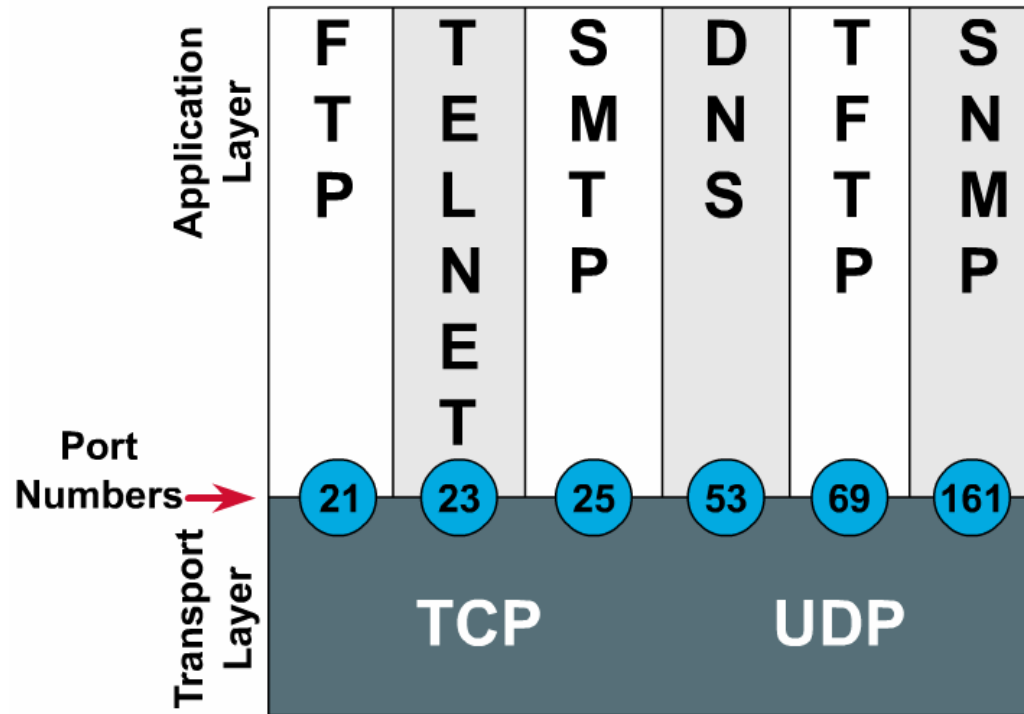


# Port Numbers

## TCP Header



## HTTP is Port 80



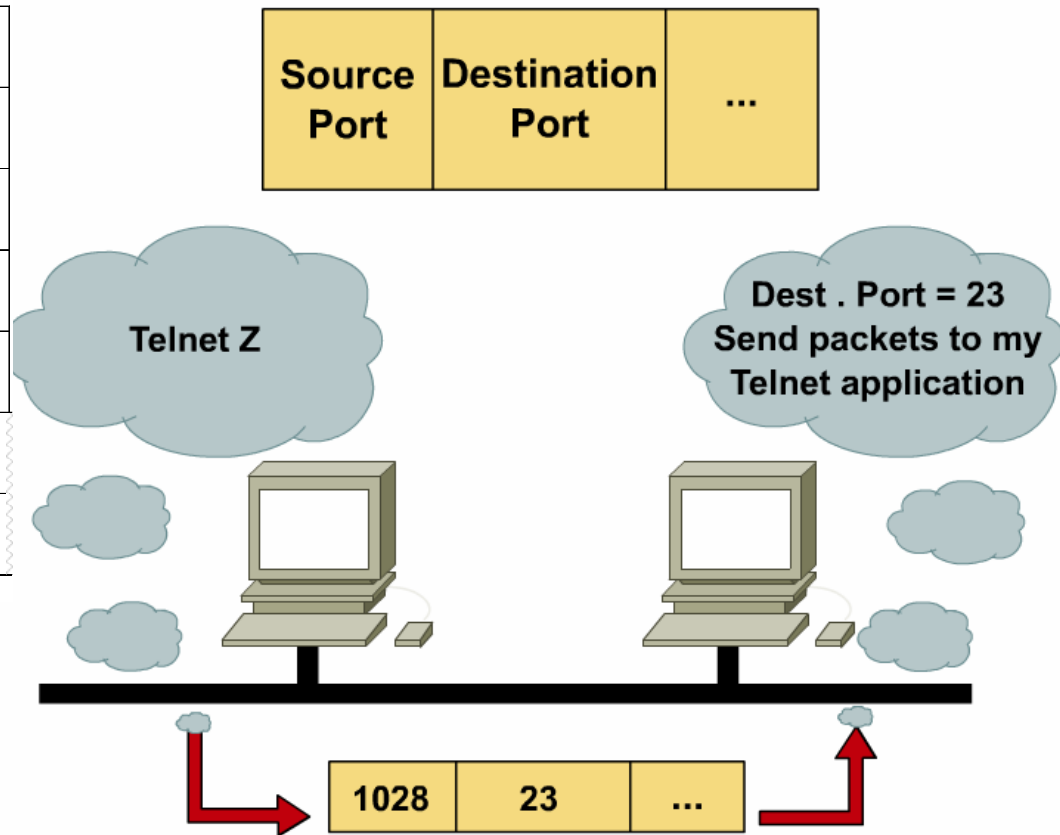
- ❑ Một số cổng dành riêng cho cả TCP và UDP, mặc dù các ứng dụng có thể không được viết để hỗ trợ chúng.
- ❑ Dải số hiệu cổng được gán được quản lý bởi IANA là 0-1023:  
<http://www.iana.org/assignments/port-numbers>
  - Các cổng thông dụng (The **Well Known Ports**) có số từ **0 đến 1023**. (Được cập nhật cho đến 11-13-2002. Trước đó, 0 - 255 được xem là các cổng thông dụng.)
  - Các cổng được đăng ký (The **Registered Ports**) là những cổng từ **1024 đến 49151** (thường được đăng ký cho các ứng dụng định trước của các nhà cung cấp thiết bị)
  - Các cổng động (The **Dynamic and/or Private Ports**) là những cổng từ **49152 đến 65535**

# TCP Header

31

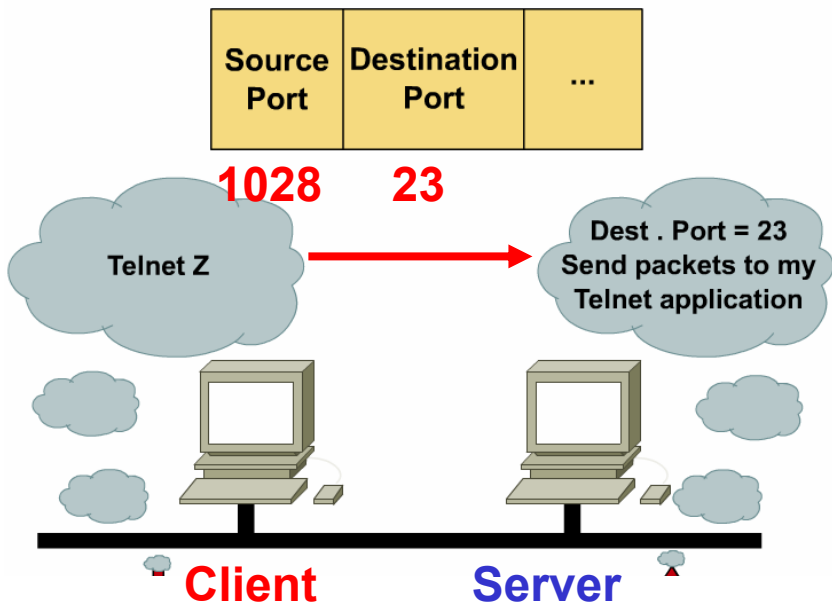
16-bit Source Port Number				16-bit Destination Port Number								
32-bit Sequence Number												
32 bit Acknowledgement Number												
4-bit Header Length		6-bit (Reserved)		U R G	A C K	P S H	R S T	S S Y	S F I	F I N	16-bit Window Size	
16-bit TCP Checksum						16-bit Urgent Pointer						
Options (if any)												
Data (if any)												

# TCP/UDP Port Numbers

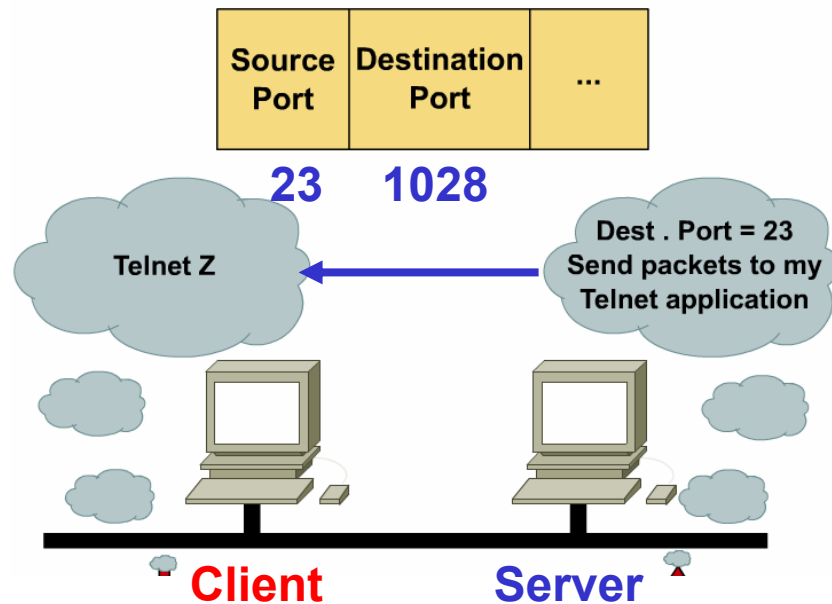


- ❑ Các hệ thống đầu cuối sử dụng các số hiệu cổng để chọn ứng dụng thích hợp.
- ❑ Các số hiệu cổng nguồn khởi nguồn, thường là các số lớn hơn 1023, được gán động bởi trạm nguồn.

## TCP/UDP Port Numbers



## TCP/UDP Port Numbers



Chú ý sự khác nhau trong việc số hiệu cổng nguồn và đích được sử dụng như thế nào với clients và servers:

**Client (khởi tạo dịch vụ Telnet):**

- Cổng đích = 23 (telnet)
- Cổng nguồn = 1028 (được gán động)

**Server (đáp ứng dịch vụ Telnet):**

- Cổng đích = 1028 (cổng nguồn của client)
- Cổng nguồn = 23 (telnet)

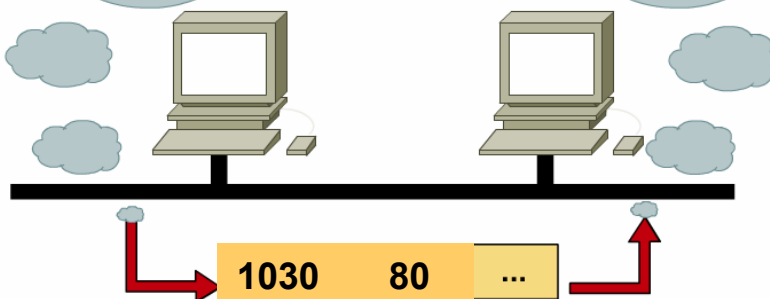
# TCP/UDP Port Numbers

Source Port	Destination Port	...
-------------	------------------	-----

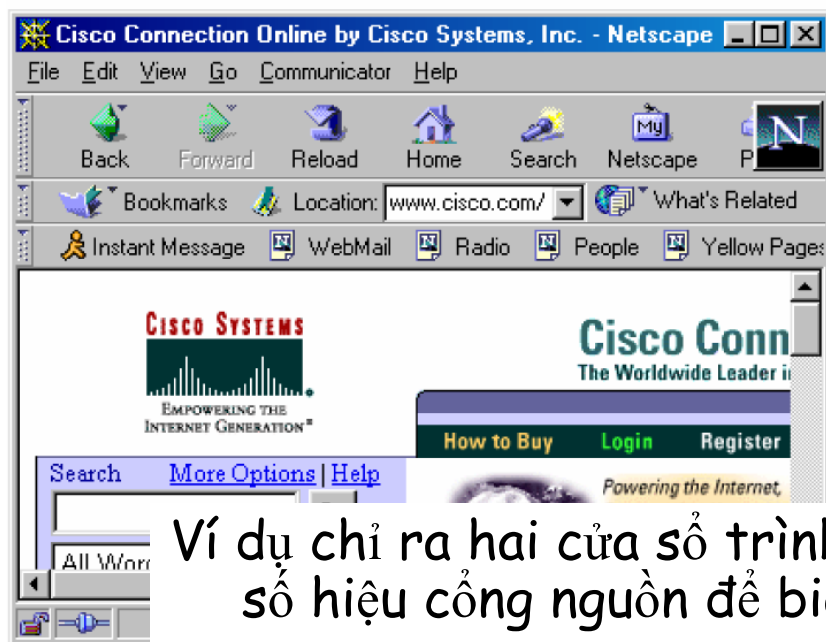
Phiên http thứ hai giữa cùng client và server.  
Giống số cổng đích, nhưng khác số cổng nguồn để định danh duy nhất cho phiên duyệt web này.

http đến  
www.cisco.com

Dest. Port = 80 Gởi  
packets đến ứng  
dụng web server

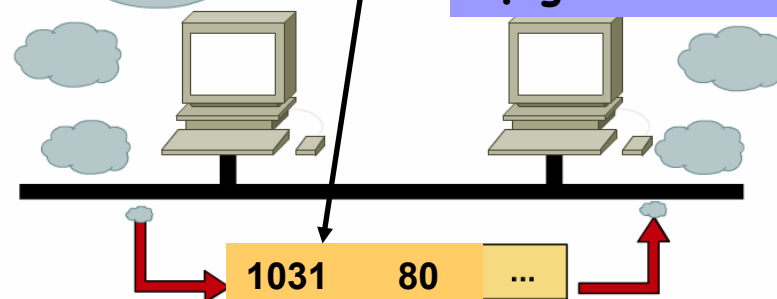


## Netscape Navigator



http đến  
www.cisco.com

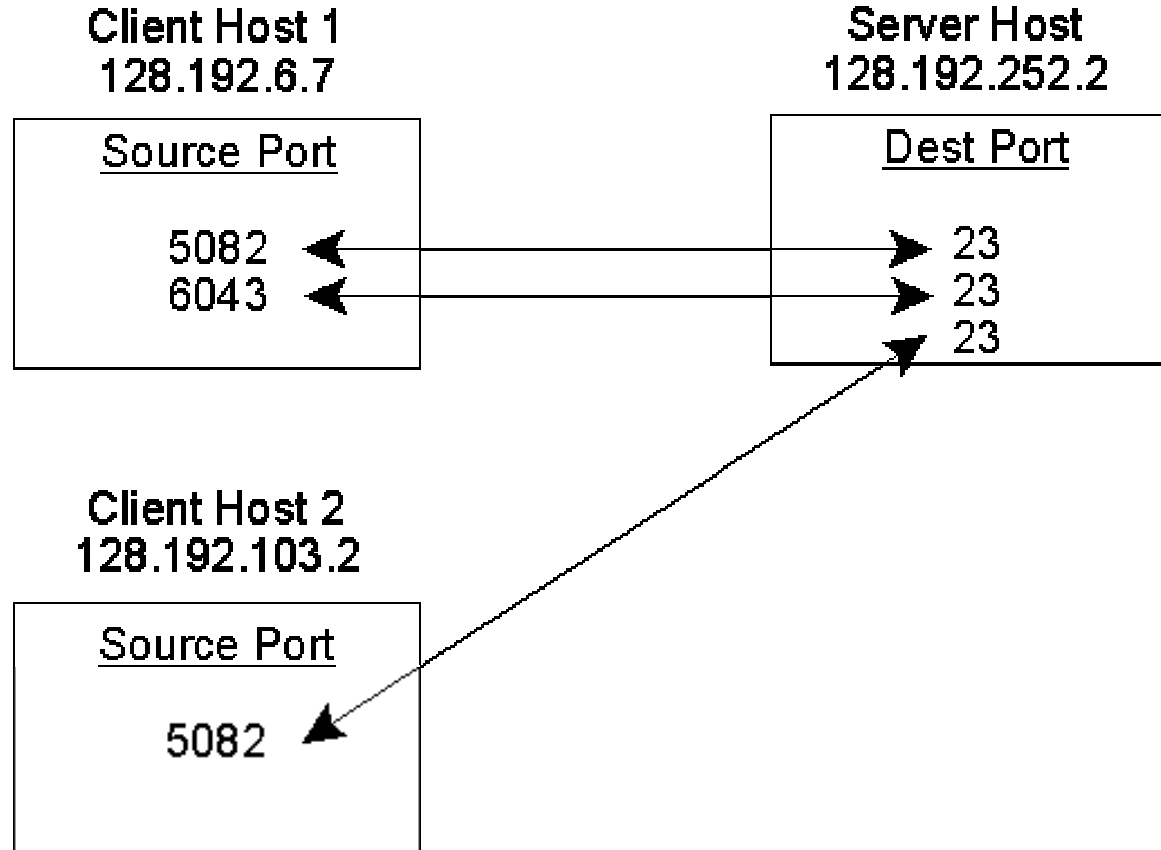
Dest. Port = 80 Gởi  
packets đến ứng  
dụng web server



## Netscape Navigator



Ví dụ chỉ ra hai cửa sổ trình duyệt có cùng URL. TCP/IP sử dụng các số hiệu cổng nguồn để biết thông tin được chuyển đến cửa sổ nào.

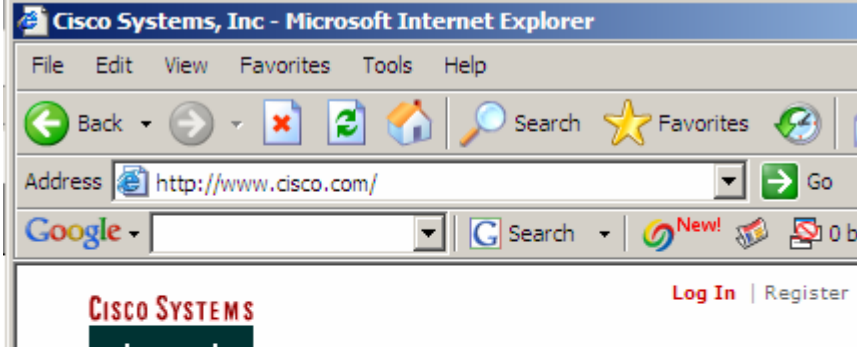
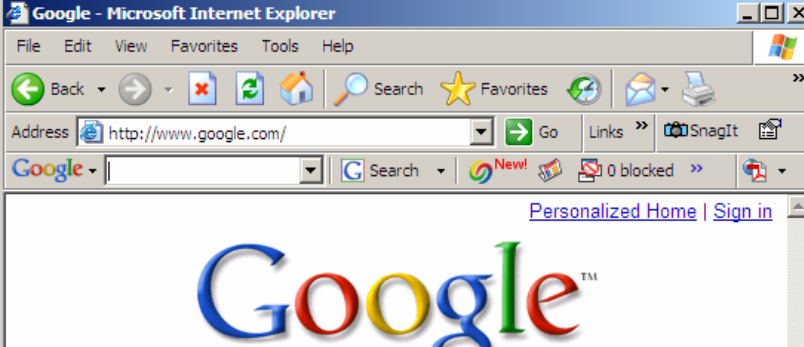


Yếu tố nào xác định tính duy nhất cho mỗi kết nối?

❑ Kết nối được định nghĩa bởi cặp các số:

- Địa chỉ IP nguồn, Số cổng nguồn
- Địa chỉ IP đích, Số cổng đích

❑ Các kết nối khác nhau có thể dùng cùng số hiệu cổng đích trên trạm chủ miễn là các số cổng nguồn hay địa chỉ IP nguồn là khác nhau.



TCP  
or  
UDP

```
C:\WINDOWS\system32\cmd.exe

C:\>netstat -n

Active Connections

Proto Local Address Foreign Address State
TCP 172.17.150.112:1033 172.16.1.44:524 ESTABLISHED
TCP 172.17.150.112:1034 172.16.1.44:524 ESTABLISHED
TCP 172.17.150.112:1042 205.188.9.73:5190 ESTABLISHED
TCP 172.17.150.112:1050 64.12.165.95:5190 ESTABLISHED
TCP 172.17.150.112:1069 207.62.185.140:143 ESTABLISHED
TCP 172.17.150.112:1332 198.133.219.25:80 TIME_WAIT
TCP 172.17.150.112:1333 198.133.219.25:80 ESTABLISHED
TCP 172.17.150.112:1334 198.133.219.25:80 ESTABLISHED
TCP 172.17.150.112:1335 64.154.80.254:80 ESTABLISHED
TCP 172.17.150.112:1336 66.102.7.99:80 ESTABLISHED

C:\>
```

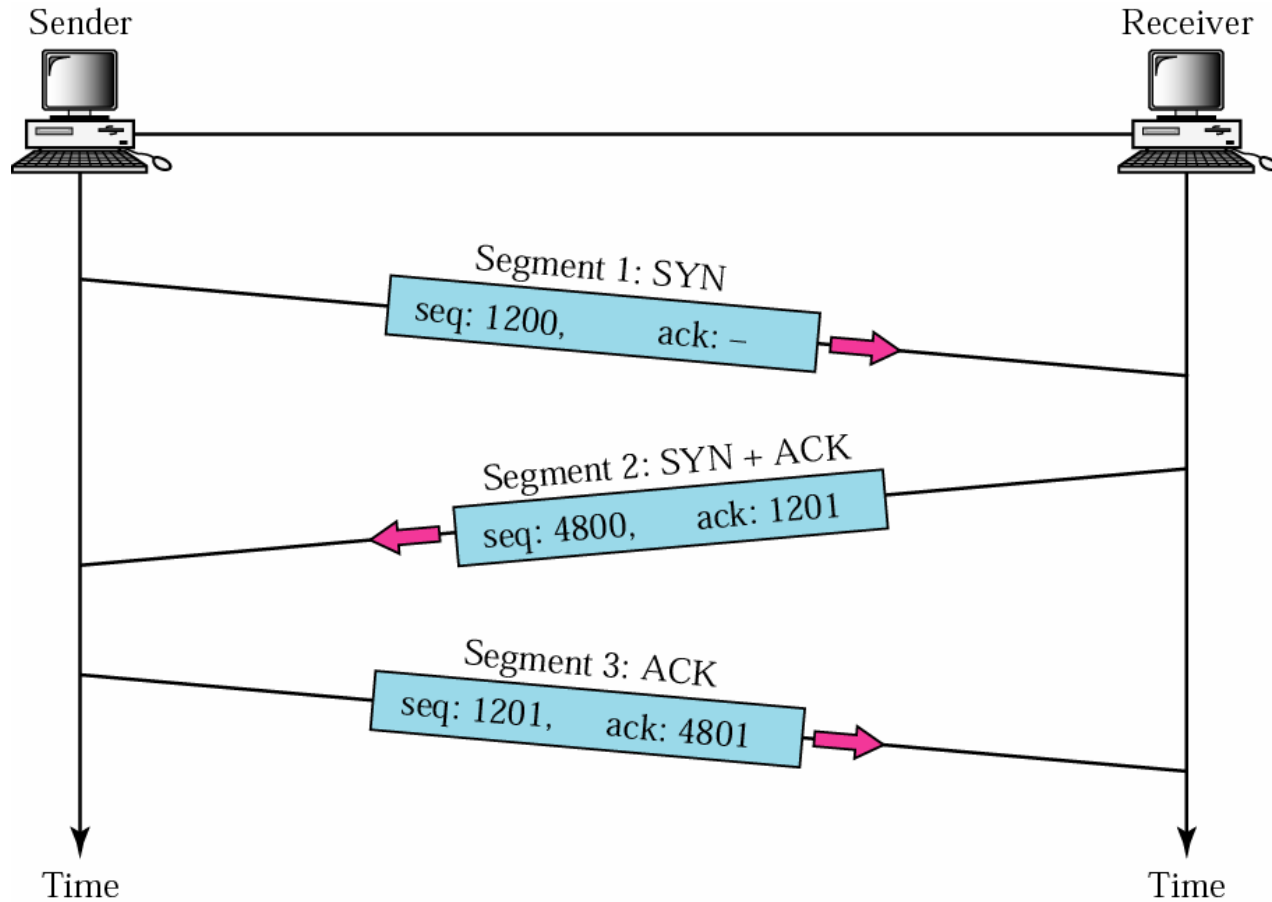
- ❑ Lưu ý: Trên thực tế, khi ta mở một trang html đơn, thông thường có nhiều phiên TCP được tạo ra, chứ không chỉ là một.
- ❑ Hình trên minh họa nhiều kết nối TCP của một phiên http đơn.

# Chương 7 - Nội dung

- ❑ 7.1 Các dịch vụ của tầng Vận chuyển
- ❑ 7.2 Multiplexing và demultiplexing
- ❑ 7.3 Vận chuyển phi kết nối: UDP
- ❑ 7.4 Vận chuyển hướng kết nối: TCP
  - cấu trúc segment
  - quản lý kết nối
  - truyền dữ liệu tin cậy
  - kiểm soát luồng
- ❑ 7.5 Kiểm soát tắc nghẽn trong TCP

# Bắt tay ba bước

## Three-way handshake





# Quản lý kết nối của TCP

**Nhắc lại:** Bên gửi và nhận (sử dụng TCP) thiết lập "kết nối" trước khi trao đổi các segments dữ liệu

- ❑ Khởi tạo các tham số cho TCP:
  - số chuỗi (sequence number)
  - thông tin về bộ đệm, kiểm soát luồng (vd: cửa sổ nhận)

❑ *client*: bên khởi tạo kết nối

```
Socket clientSocket = new  
Socket("hostname", "port  
number");
```

❑ *server*: được liên hệ bởi client

```
Socket connectionSocket =  
welcomeSocket.accept();
```

## **Bắt tay ba bước:**

**Bước 1:** client gửi SYN TCP segment đến server

- định rõ số chuỗi khởi tạo của client
- không chứa dữ liệu

**Step 2:** server nhận SYN, trả lời với SYNACK segment

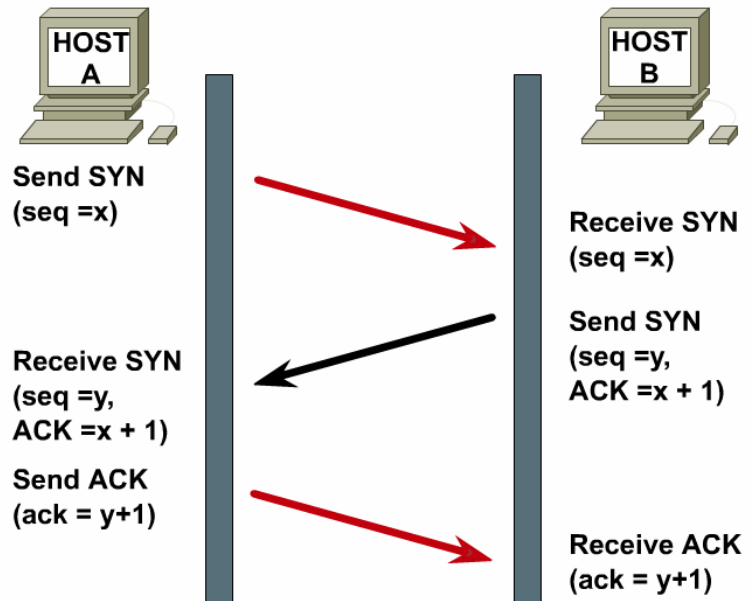
- server cấp phát bộ đệm
- định rõ số chuỗi khởi tạo của server

**Step 3:** client nhận SYNACK, trả lời với ACK segment, nó có thể chứa dữ liệu

# TCP Header

0				15				16				31			
16-bit Source Port Number								16-bit Destination Port Number							
32-bit Sequence Number															
32 bit Acknowledgement Number															
4-bit Header Length				6-bit (Reserved)				U A P R S F R C S S Y I G K H T N N				16-bit Window Size			
16-bit TCP Checksum								16-bit Urgent Pointer							
Options (if any)															
Data (if any)															

## TCP Three-Way Handshake/ Open Connection



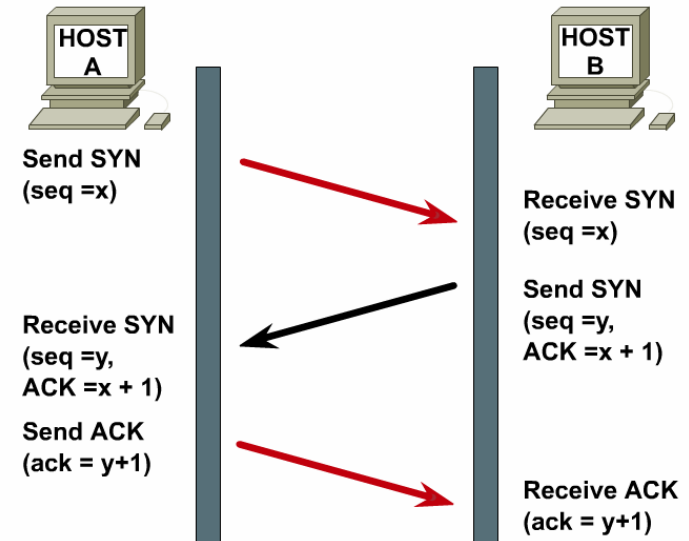
- ❑ Để một kết nối được thiết lập, hai trạm cuối phải đồng bộ với nhau về số chuỗi khởi tạo của TCP (ISNs).
- ❑ Số chuỗi được sử dụng để theo dõi thứ tự của các gói và để đảm bảo rằng không có gói nào bị mất trong quá trình truyền.
- ❑ Số chuỗi khởi tạo là con số bắt đầu khi một kết nối TCP được thiết lập.
- ❑ Trao đổi số chuỗi bắt đầu trong suốt quá trình kết nối đảm bảo rằng dữ liệu bị mất có thể được phục hồi.

# TCP Header

31

16-bit Source Port Number				16-bit Destination Port Number			
32-bit Sequence Number							
32 bit Acknowledgement Number							
4-bit Header Length	6-bit (Reserved)	URG	ACK	PUSH	RESET	SYN	FIN
				16-bit Window Size			
16-bit TCP Checksum				16-bit Urgent Pointer			
Options (if any)							
Data (if any)							

## TCP Three-Way Handshake/ Open Connection



- ❑ **Sự đồng bộ** được thiết lập bằng cách trao đổi các segments có mang các số chuỗi khởi tạo và bit điều khiển SYN được bật, có nghĩa là *synchronize* (đồng bộ). (Các segments mang bit SYN còn được gọi là SYNs.)
- ❑ Kết nối thành công đòi hỏi một cơ chế phù hợp để chọn một số chuỗi khởi tạo và một cách liên quan đến việc "bắt tay" để trao đổi số chuỗi khởi tạo (ISNs).
- ❑ Sự đồng bộ yêu cầu rằng mỗi bên gửi số chuỗi khởi tạo (ISN) của nó và nhận một sự xác nhận và ISN từ phía bên kia của kết nối.
- ❑ Mỗi bên phải nhận ISN của phía bên kia và gửi báo nhận (ACK) trong một thứ tự cụ thể.

# TCP Header

31

16-bit Source Port Number				16-bit Destination Port Number						
32-bit Sequence Number										
32 bit Acknowledgement Number										
4-bit Header Length		6-bit (Reserved)		U R G	A C K	P R H	S S T	F I N	16-bit Window Size	
16-bit TCP Checksum					16-bit Urgent Pointer					
Options (if any)										
Data (if any)										

# TCP Three-Way Handshake/ Open Connection



Send SYN  
(seq = x)

Receive SYN  
(seq = y,  
ACK = x + 1)

Send ACK  
(ack = y+1)



Receive SYN  
(seq = x)

Send SYN  
(seq = y,  
ACK = x + 1)

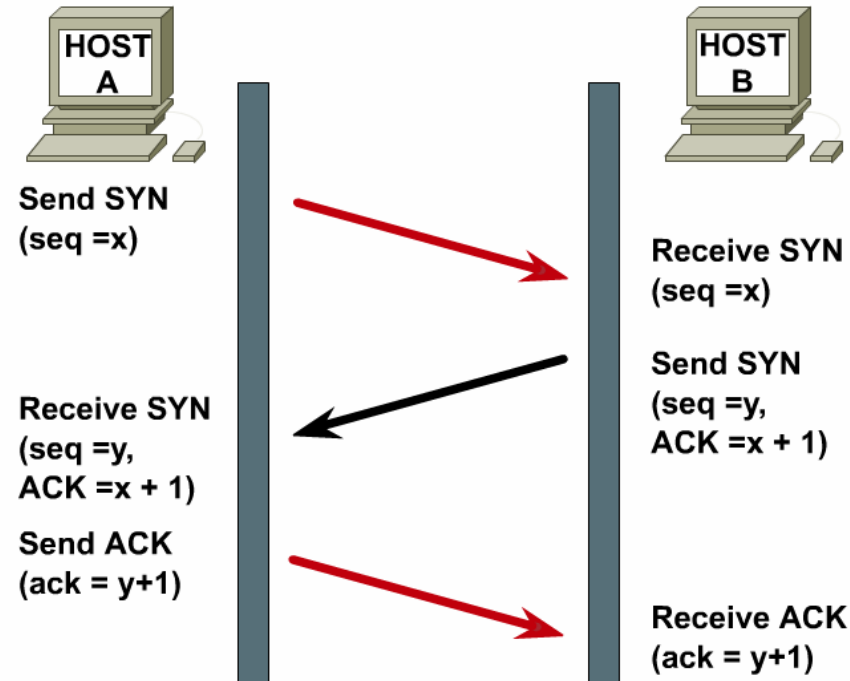
Receive ACK  
(ack = y+1)

- ❑ Để thiết lập kết nối, ba packets được gửi giữa hai trạm đầu cuối.
- ❑ Bắt tay ba bước là cần thiết vì TCPs (trên các hệ thống đầu cuối) có thể sử dụng các cơ chế khác nhau để chọn ra một số chuỗi khởi tạo (ISN).
- ❑ Người nhận gói SYN đầu tiên không có cách nào để biết được nó có phải là một segment cũ bị trễ hay không trừ khi nó nhớ được số chuỗi cuối cùng được sử dụng cho kết nối đó, điều này không phải lúc nào cũng có thể thực hiện được, và nó phải yêu cầu người gửi xác minh lại gói SYN này

# TCP Header

0															15																31																																												
16-bit Source Port Number															16-bit Destination Port Number																																																												
32-bit Sequence Number																																																																											
32 bit Acknowledgement Number																																																																											
4-bit Header Length				6-bit (Reserved)				<table><tr><td>U</td><td>A</td><td>P</td><td>R</td><td>S</td><td>F</td></tr><tr><td>R</td><td>C</td><td>S</td><td>S</td><td>Y</td><td>I</td></tr><tr><td>G</td><td>K</td><td>H</td><td>T</td><td>N</td><td></td></tr></table>								U	A	P	R	S	F	R	C	S	S	Y	I	G	K	H	T	N		16-bit Window Size																																									
U	A	P	R	S	F																																																																						
R	C	S	S	Y	I																																																																						
G	K	H	T	N																																																																							
16-bit TCP Checksum																																16-bit Urgent Pointer																																											
Options (if any)																																																																											
Data (if any)																																																																											

## TCP Three-Way Handshake/ Open Connection



- Tại thời điểm này, cả hai bên đều có thể bắt đầu truyền thông, và cả hai đều có thể ngắt mọi liên kết truyền thông này vì TCP là phương thức truyền thông đồng đẳng (cân bằng).

# Quản lý kết nối của TCP (tiếp theo)

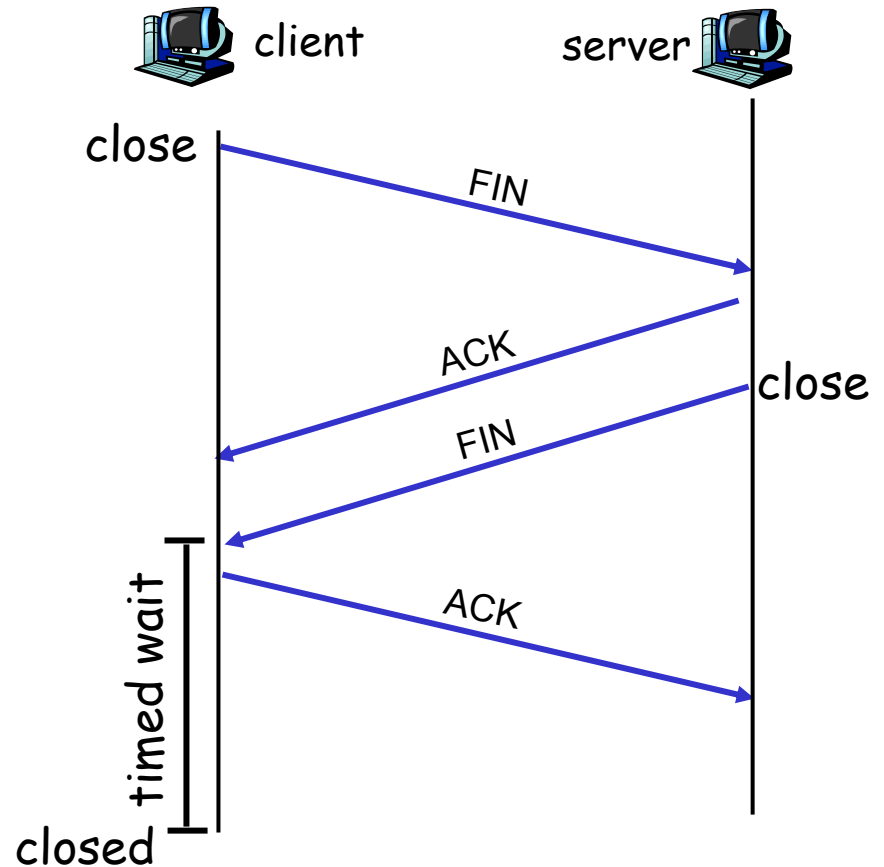
## Đóng một kết nối:

đóng client socket:

```
clientSocket.close();
```

**Bước 1:** client gửi segment điều khiển TCP FIN đến server

**Bước 2:** server nhận FIN, trả lời với ACK. Đóng kết nối, gửi FIN.

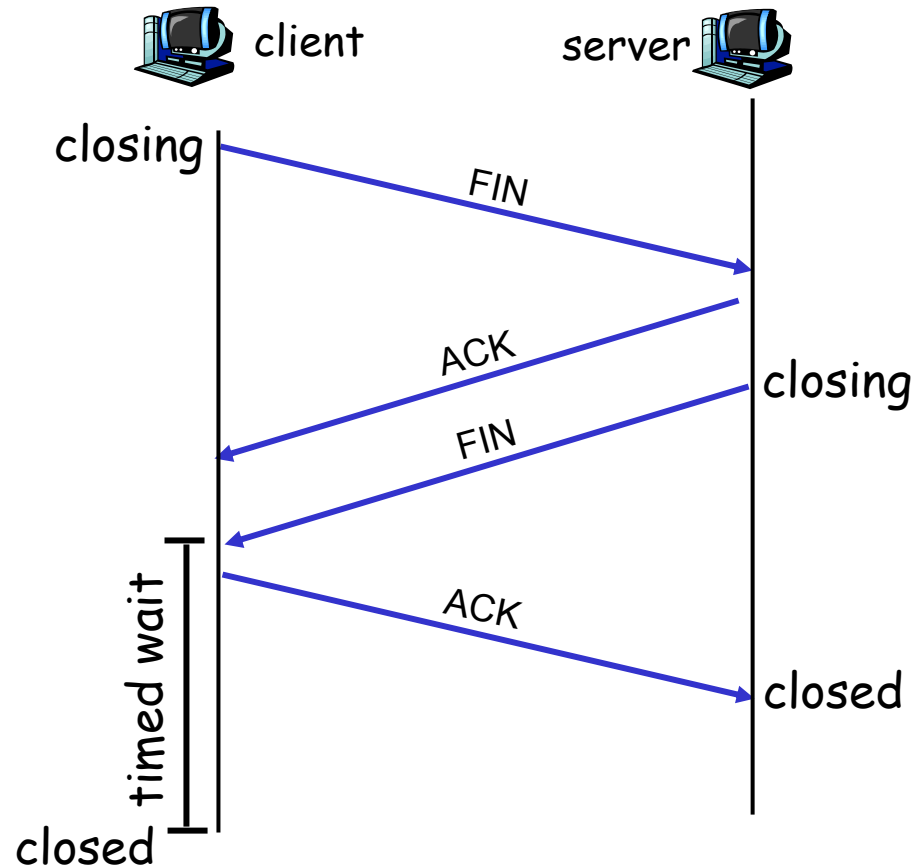


# Quản lý kết nối của TCP (tiếp theo)

**Bước 3:** client nhận FIN, trả lời với ACK.

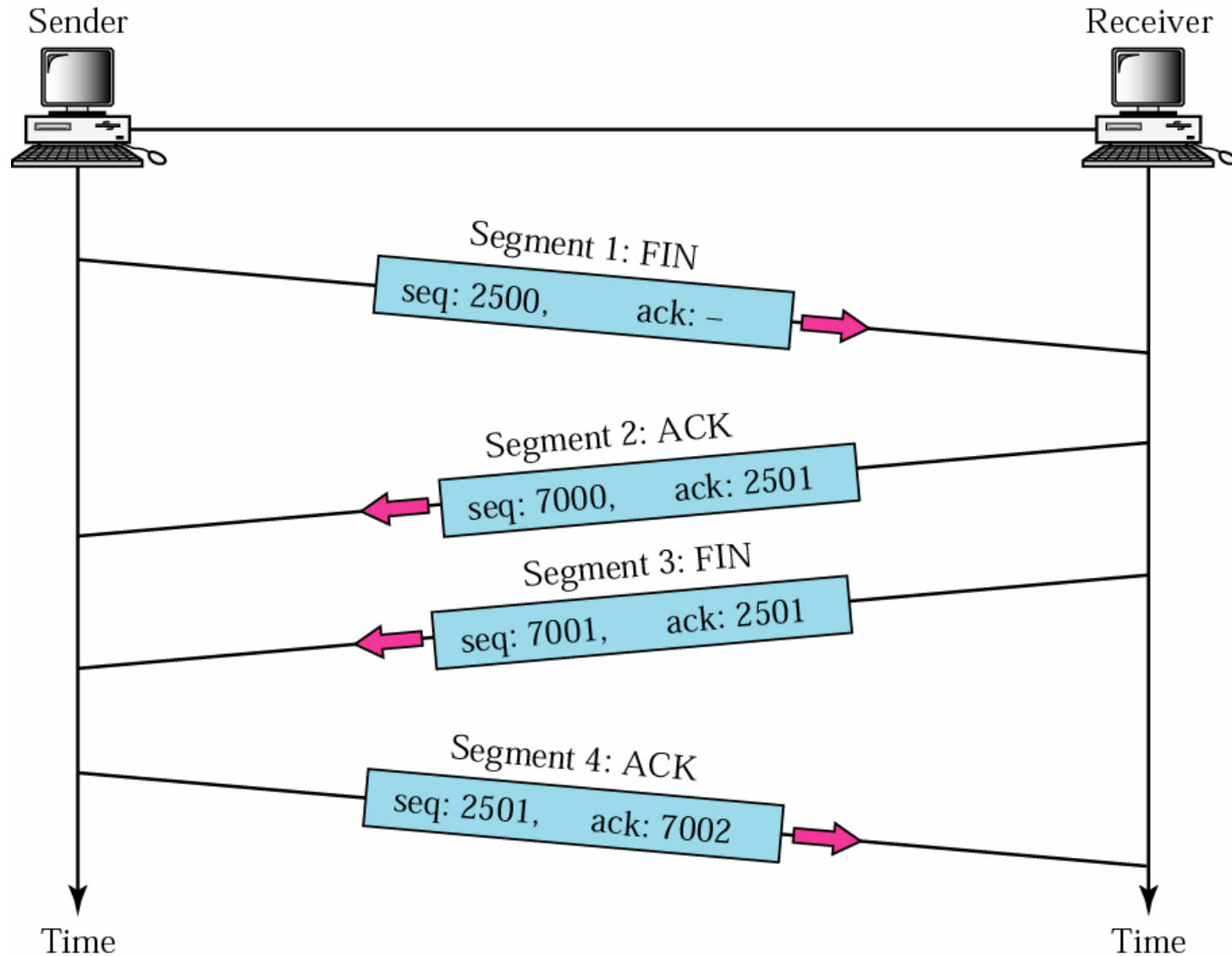
- Vào trạng thái "chờ đợi một thời gian" - sẽ trả lời với ACK cho FINs nhận được

**Bước 4:** server, nhận ACK.  
Kết nối đã được đóng.



# Bắt tay Bốn bước

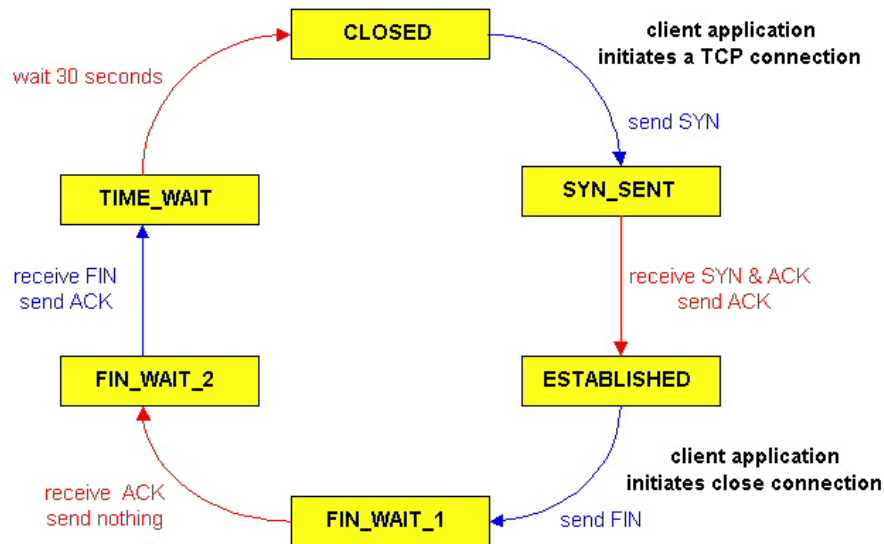
## Four-way handshake



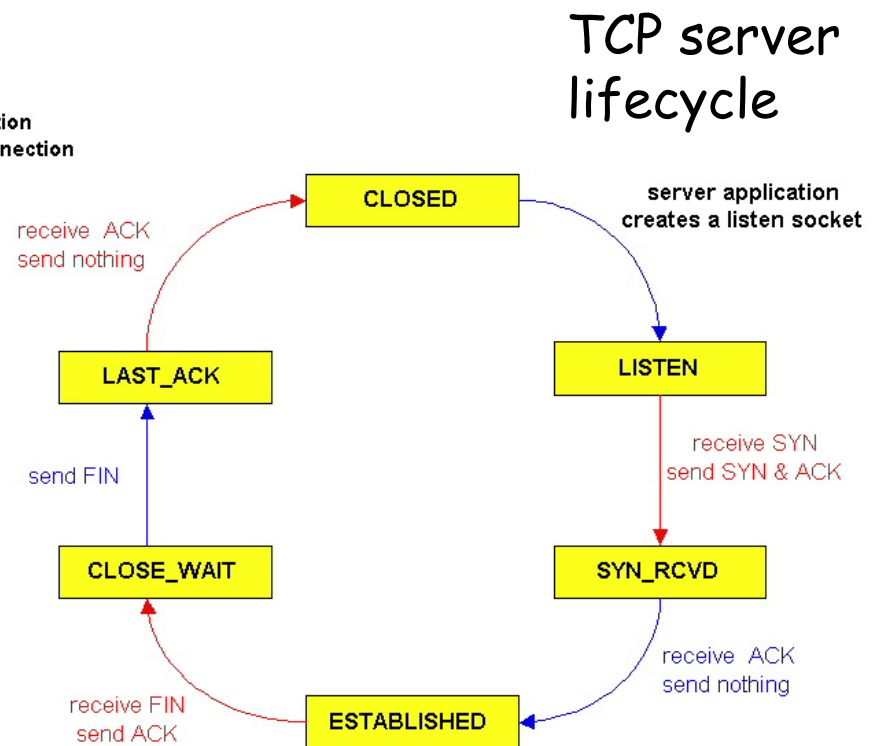


# Quản lý kết nối của TCP (tiếp theo)

## TCP Connection Management (cont.)



TCP client lifecycle



# Chương 7 - Nội dung

- ❑ 7.1 Các dịch vụ của tầng Vận chuyển
- ❑ 7.2 Multiplexing và demultiplexing
- ❑ 7.3 Vận chuyển phi kết nối: UDP
- ❑ 7.4 Vận chuyển hướng kết nối: TCP
  - cấu trúc segment
  - quản lý kết nối
  - truyền dữ liệu tin cậy & kiểm soát luồng
- ❑ 7.5 Kiểm soát tắc nghẽn trong TCP

## TCP Simple Acknowledgment

The diagram shows a communication between a **Sender** and a **Receiver**. The process is as follows:

- Sender** sends 1 (red arrow).
- Receiver** receives 1 and sends ACK 2 (black arrow).
- Sender** sends 2 (red arrow).
- Receiver** receives 2 and sends ACK 3 (black arrow).
- Sender** sends 3 (red arrow).
- Receiver** receives 3 and sends ACK 4 (black arrow).

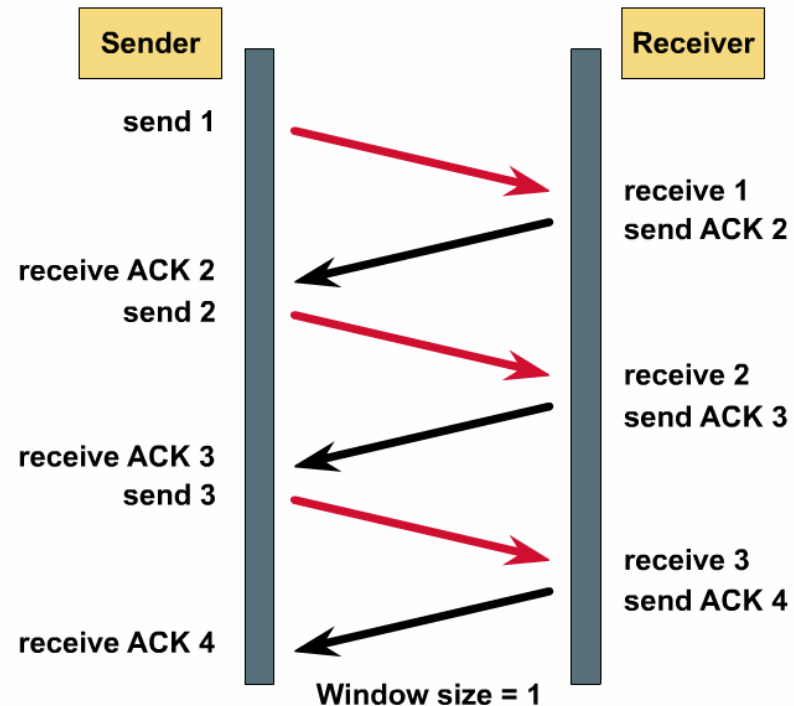
At the bottom, it states **Window size = 1**.

- ❑ Để kiểm soát luồng dữ liệu giữa các thiết bị, TCP dùng một cơ chế kiểm soát luồng đồng đẳng (peer-to-peer).
- ❑ Tầng TCP của trạm nhận thông báo một kích thước cửa sổ cho tầng TCP của trạm gửi.
- ❑ **Kích thước cửa sổ này định rõ số bytes, bắt đầu từ số báo nhận (ACK number), mà tầng TCP ở bên nhận hiện tại đang chuẩn bị để nhận.**

# TCP Header

				31			
16-bit Source Port Number				16-bit Destination Port Number			
32-bit Sequence Number							
32 bit Acknowledgement Number							
4-bit Header Length	6-bit (Reserved)	U R G	A C K	P S H	R S T	S Y N	F I N
				16-bit Window Size			
16-bit TCP Checksum				16-bit Urgent Pointer			
Options (if any)							
Data (if any)							

# TCP Simple Acknowledgment

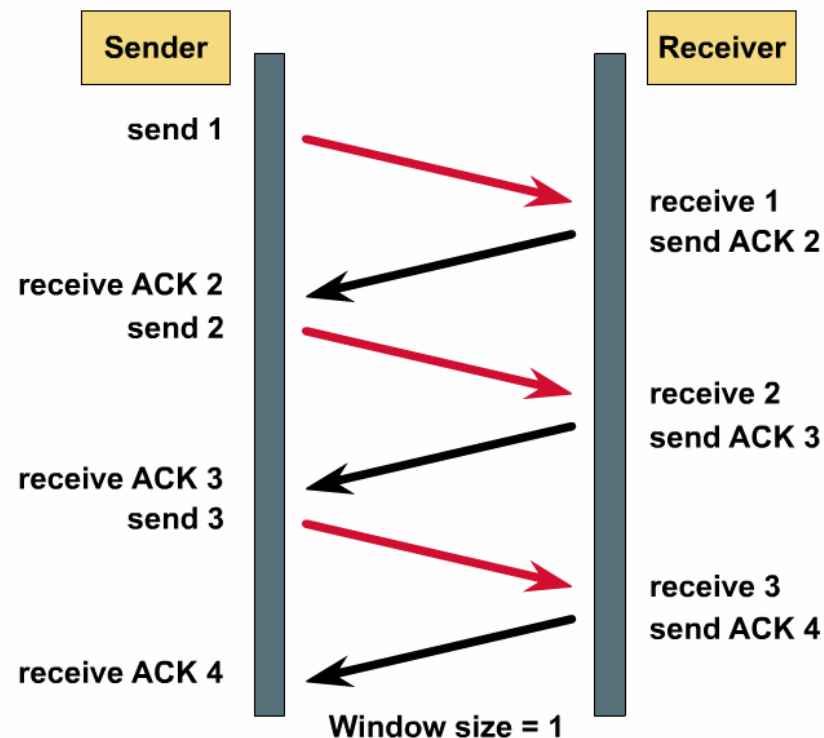


- ❑ TCP - một giao thức hướng kết nối, tin cậy; cung cấp kiểm soát luồng qua việc cung cấp các cửa sổ trượt, và tin cậy bằng cách cung cấp các số chuỗi và hồi báo.
- ❑ TCP gởi lại bất kỳ đoạn nào không được nhận và cung cấp một kênh ảo "TCP" giữa các ứng dụng của người dùng đầu cuối.
- ❑ Sự thuận lợi của TCP đó là nó cung cấp dịch vụ phân phát bảo đảm các segment.

# TCP Header

				31			
16-bit Source Port Number				16-bit Destination Port Number			
32-bit Sequence Number							
32 bit Acknowledgement Number							
4-bit Header Length	6-bit (Reserved)	U R G	A C K	P S H	R S T	S Y N	F I N
				16-bit Window Size			
16-bit TCP Checksum				16-bit Urgent Pointer			
Options (if any)							
Data (if any)							

# TCP Simple Acknowledgment

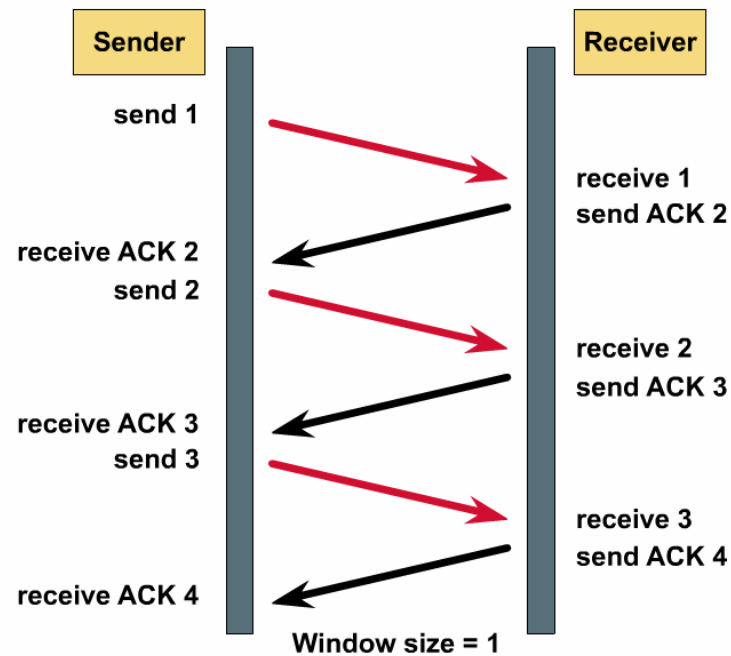


- ❑ Kích cỡ cửa sổ nói đến số bytes có thể được truyền đi trước khi nhận một hồi báo (ACK).
- ❑ Sau khi một trạm truyền một số bytes bằng của kích cỡ cửa sổ, nó phải nhận một hồi báo trước khi thêm bất kỳ dữ liệu nào có thể được gửi.
- ❑ Kích cỡ cửa sổ xác định bao nhiêu dữ liệu mà trạm nhận có thể tiếp nhận tại một thời điểm.

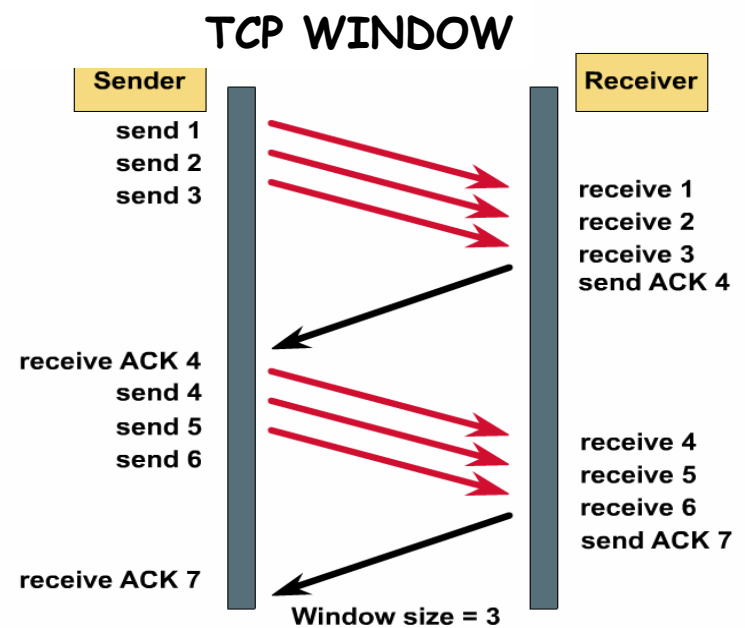
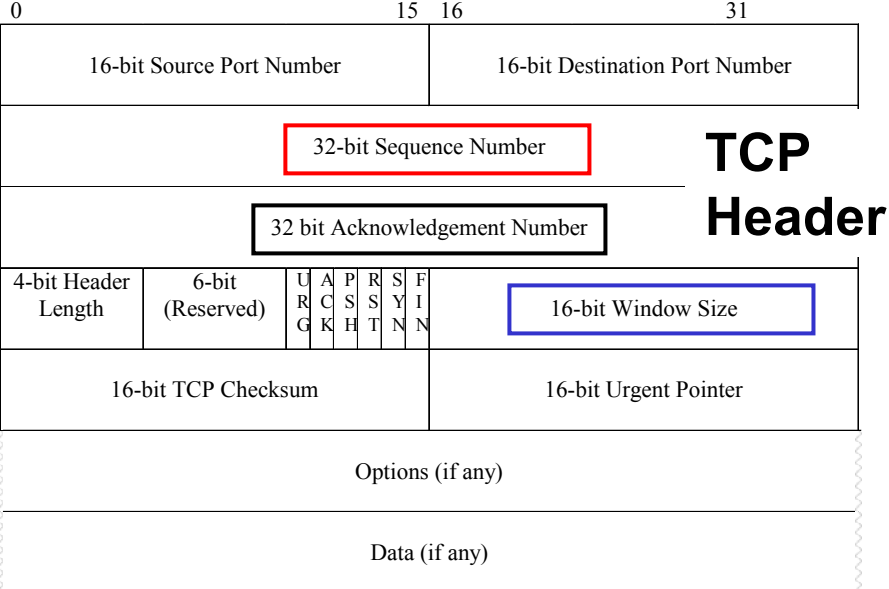
# TCP Header

				31			
16-bit Source Port Number				16-bit Destination Port Number			
32-bit Sequence Number							
32 bit Acknowledgement Number							
4-bit Header Length	6-bit (Reserved)	U R G	A P K	P R H	S S T	F I N	16-bit Window Size
16-bit TCP Checksum				16-bit Urgent Pointer			
Options (if any)							
Data (if any)							

# TCP Simple Acknowledgment



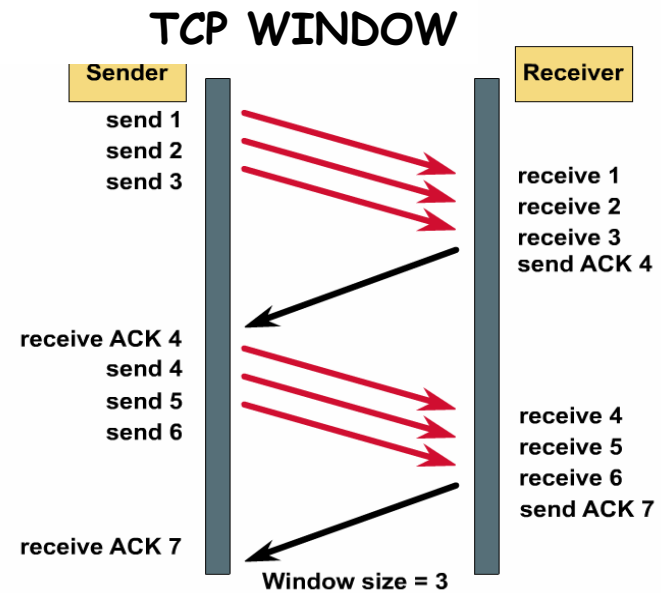
- ❑ Với kích thước cửa sổ bằng 1, mỗi segment chỉ mang một byte dữ liệu và phải được hồi báo trước khi segment khác được truyền.
- ❑ Điều này làm cho việc sử dụng giải thông của trạm không hiệu quả.
- ❑ Mục đích của cửa sổ là để cải tiến kiểm soát luồng và sự tin cậy.
- ❑ Nhưng với kích thước cửa sổ là 1, ta thấy việc sử dụng giải thông là rất không hiệu quả.



## Kích thước cửa sổ TCP (TCP Window Size)

- ❑ TCP sử dụng kích thước cửa sổ, số bytes, mà bên nhận sẵn sàng để tiếp nhận, và thông thường được quản lý bởi tiến trình nhận.
- ❑ TCP sử dụng **hồi báo mong đợi (expectational acknowledgments)**, nghĩa là số hồi báo nói đến byte tiếp theo mà người gửi hồi báo muốn nhận.
- ❑ Kích thước cửa sổ càng lớn cho phép càng nhiều dữ liệu được truyền trước khi phải chờ hồi báo.
- ❑ Lưu ý: Số chuỗi đang được gửi chính là số chuỗi của ra byte đầu tiên trong *segment* đó.

0		15		16		31					
16-bit Source Port Number				16-bit Destination Port Number							
32-bit Sequence Number						TCP Header					
32 bit Acknowledgement Number											
4-bit Header Length		6-bit (Reserved)		U R G	A C K	P S H	R S T	S Y N	F I N	16-bit Window Size	
16-bit TCP Checksum						16-bit Urgent Pointer					
Options (if any)											
Data (if any)											



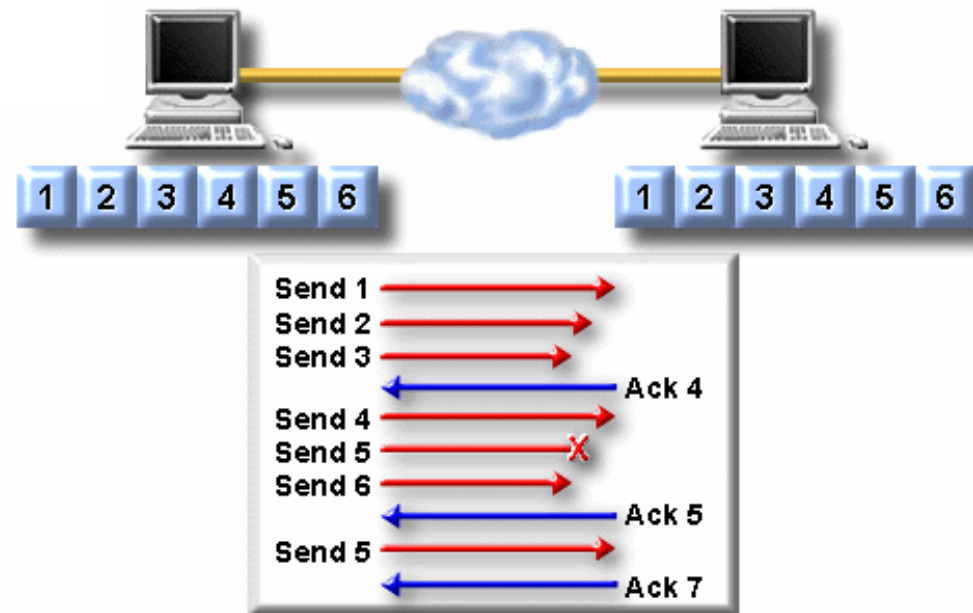
## Kích thước cửa sổ TCP (TCP Window Size)

- ❑ TCP cung cấp dịch vụ song công hoàn toàn, nghĩa là dữ liệu có thể chạy trên mỗi hướng, độc lập với hướng còn lại.
- ❑ Kích thước cửa sổ, các số chuỗi và số hồi báo là độc lập cho mỗi luồng dữ liệu.
- ❑ Bên nhận gửi kích thước cửa sổ chấp nhận được đến người gửi trong mỗi segment được truyền đi (kiểm soát luồng)
  - nếu quá nhiều dữ liệu đang được gửi, kích thước cửa sổ chấp nhận được bị giảm xuống
  - nếu có thể xử lý nhiều dữ liệu, kích thước cửa sổ được tăng lên
- ❑ Kỹ thuật này được biết đến như là một giao thức cửa sổ **Dừng-và-Đợi (Stop-and-Wait)**.



015															16																31																																																																																								
16-bit Source Port Number																														16-bit Destination Port Number																																																																																									
32-bit Sequence Number																																																												TCP Header																																																											
32 bit Acknowledgement Number																																																																																																																							
4-bit Header Length															6-bit (Reserved)															U R G A C K R S T S Y N F I N															16-bit Window Size																																																																										
16-bit TCP Checksum																														16-bit Urgent Pointer																																																																																									
Options (if any)																																																																																																																							
Data (if any)																																																																																																																							

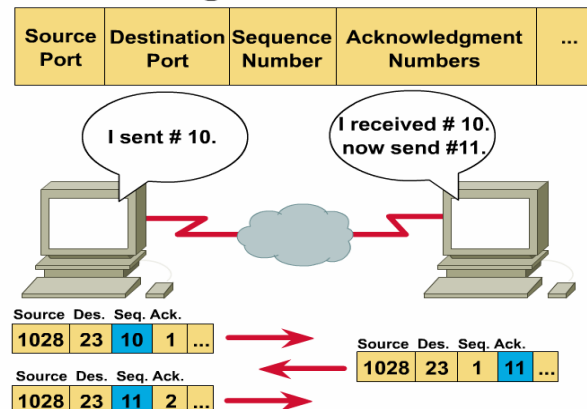
## An Acknowledgment Technique



- ❑ Packets có thể bị bỏ dọc đường, hết thời gian, hay bị sai lệch.
- ❑ TCP cung cấp các **hồi báo lũy tích (cumulative acknowledgments)**.
- ❑ Hiện tại không có cách để hồi báo các mảnh được chọn lọc của dòng dữ liệu (data stream).
- ❑ Nếu octets 4, 5, và 6 đã được gửi, nhưng 5 bị mất, bên nhận chỉ hồi báo cho đến 4 bằng cách gửi một Ack là 5.
- ❑ Bên gửi gửi lại 5 và đợi để nghe từ phía nhận nó nên bắt đầu lại từ đâu.
- ❑ Bên nhận gửi Ack 7, do đó bên gửi biết nó có thể bắt đầu gửi tiếp từ octet 7.

Chỉ một octet được gửi tại mỗi thời điểm, nhưng nếu nhiều bytes được gửi (thông thường) thì sao?

## TCP Sequence and Acknowledgment Numbers



## Lưu ý

- Bên gửi: Giá trị của số chuỗi là của byte đầu tiên trong dòng dữ liệu.
- Làm thế nào để bên nhận biết bao nhiêu dữ liệu được gửi, gửi giá trị hồi báo là bao nhiêu?
- Bên nhận: Sử dụng gói IP của bên gửi và thông tin của TCP segment, giá trị của ACK là::

IP Length: (IP header) Total length - Header length

- TCP header length (TCP header): Header length

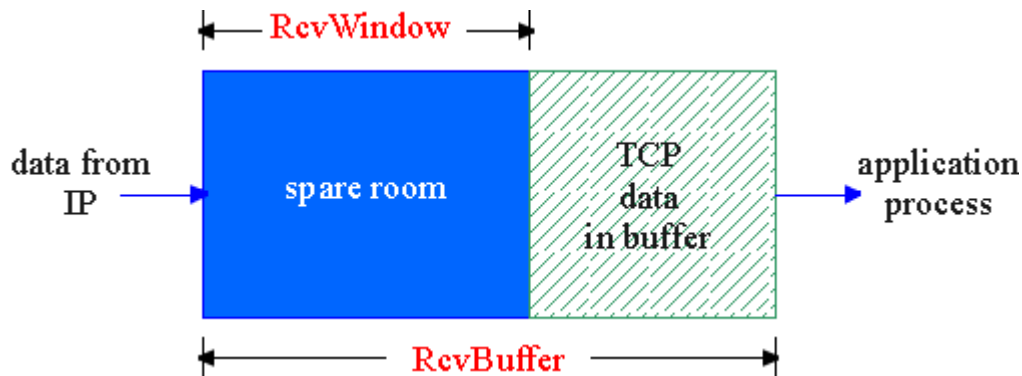
-----  
Chiều dài của dữ liệu trong TCP segment (Length of data in TCP segment)

ACK = Số chuỗi cuối cùng đã hồi báo + Chiều dài của dữ liệu trong TCP (được tính ở trên)

- Kiểm tra số chuỗi để phát hiện các segments bị mất và để sắp xếp lại các segments đến sai thứ tự.
- Giá trị của ACK là cho số chuỗi của byte mà bạn (receiver) muốn nhận. Khi ta ACK 101, điều đó nói rằng bạn đã nhận được tất cả các bytes cho đến 100.

# Kiểm soát luồng của TCP

- bên nhận của kết nối TCP có một bộ đệm nhận dữ liệu:



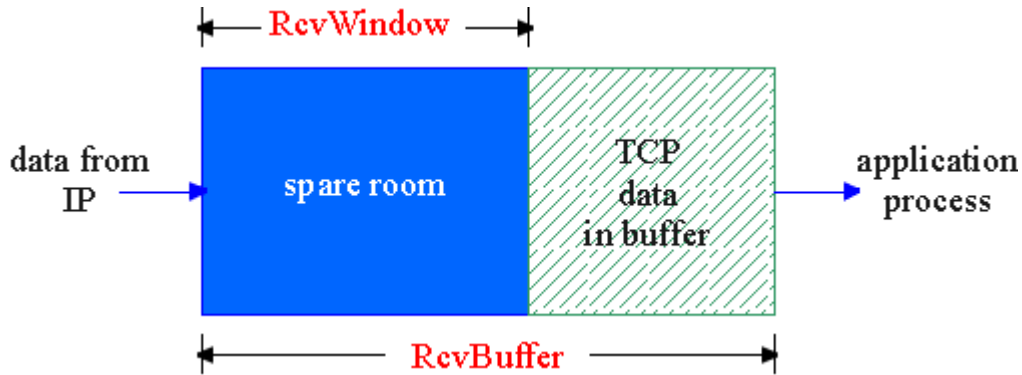
- tiến trình ứng dụng có thể bị chậm lúc đọc dữ liệu từ vùng đệm

## kiểm soát luồng

bên gửi sẽ không làm ngập bộ đệm của bên nhận bằng cách truyền quá nhiều, quá nhanh

- dịch vụ tốc độ thích hợp: điều chỉnh tốc độ gửi phù hợp với khả năng của ứng dụng bên nhận

# Kiểm soát luồng của TCP: hoạt động như thế nào



(Giả sử bên nhận của kết nối TCP loại bỏ các segment sai thứ tự)

- chỗ còn trống trong vùng đệm

= RcvWindow

= RcvBuffer - [LastByteRcvd - LastByteRead]

- Bên nhận thông báo chỗ còn trống bằng cách bao gồm giá trị của RcvWindow trong các segments
- Bên gửi giới hạn các dữ liệu chưa được ACK đến RcvWindow
  - đảm bảo rằng bộ đệm của bên nhận không bị tràn

# Cửa sổ trượt

**Kích thước cửa sổ khởi tạo**

**Cửa sổ có thể sử dụng**

**Có thể gửi ngay**

**Kích thước cửa sổ hoạt động**

**Đã gửi  
chưa ACK**

**Có thể sử dụng  
Có thể gửi ngay**

## **Giao thức cửa sổ trượt**

- ❑ Giải thuật cửa sổ trượt là một phương pháp kiểm soát luồng cho việc truyền dữ liệu trên mạng sử dụng kích cỡ cửa sổ (Window size).
- ❑ Bên gửi tính cửa sổ có thể sử dụng được, nó cho biết bao nhiêu byte dữ liệu có thể được gửi ngay.
- ❑ Theo thời gian, cửa sổ trượt này di chuyển sang bên phải, khi bên nhận hồi báo cho dữ liệu đã nhận đúng.
- ❑ Bên nhận gửi hồi báo khi bộ đệm nhận TCP của nó trống.
- ❑ Ngôn ngữ được sử dụng để mô tả sự dịch chuyển của gờ trái và gờ phải của cửa sổ trượt đó là:
  1. Gờ trái đóng (di chuyển sang bên phải) khi dữ liệu được gửi và được hồi báo.
  2. Gờ phải mở (di chuyển sang phải) cho phép nhiều dữ liệu hơn được gửi. Điều này xảy ra khi bên nhận hồi báo đã nhận một số byte nào đó.
  3. Gờ giữa mở (di chuyển sang phải) khi dữ liệu đã được gửi, nhưng chưa được hồi báo.

## Host A - Sender

1	2	3	4	5	6	7	8	9	10	11	12	13
---	---	---	---	---	---	---	---	---	----	----	----	----

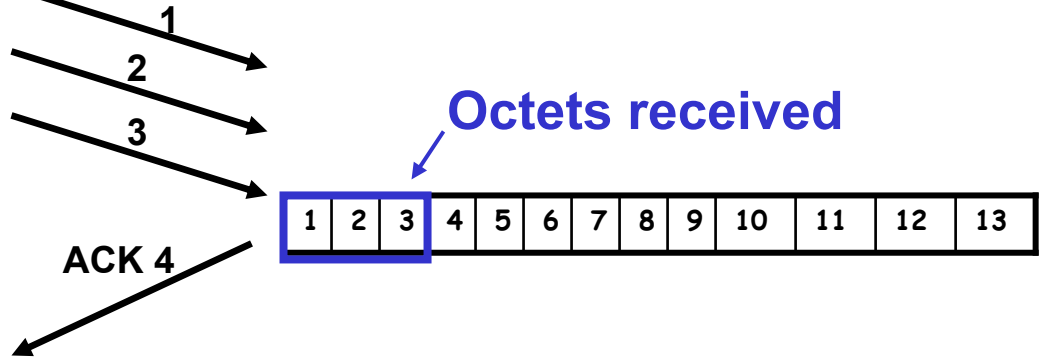
1	2	3	4	5	6	7	8	9	10	11	12	13
---	---	---	---	---	---	---	---	---	----	----	----	----

Window size = 6

Octets sent	Usable Window
Not ACKed	Can send ASAP

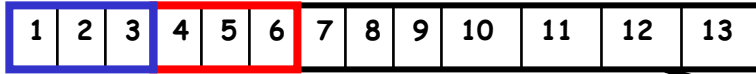
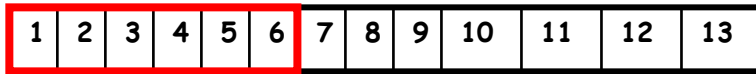
## Host B - Receiver

1	2	3	4	5	6	7	8	9	10	11	12	13
---	---	---	---	---	---	---	---	---	----	----	----	----

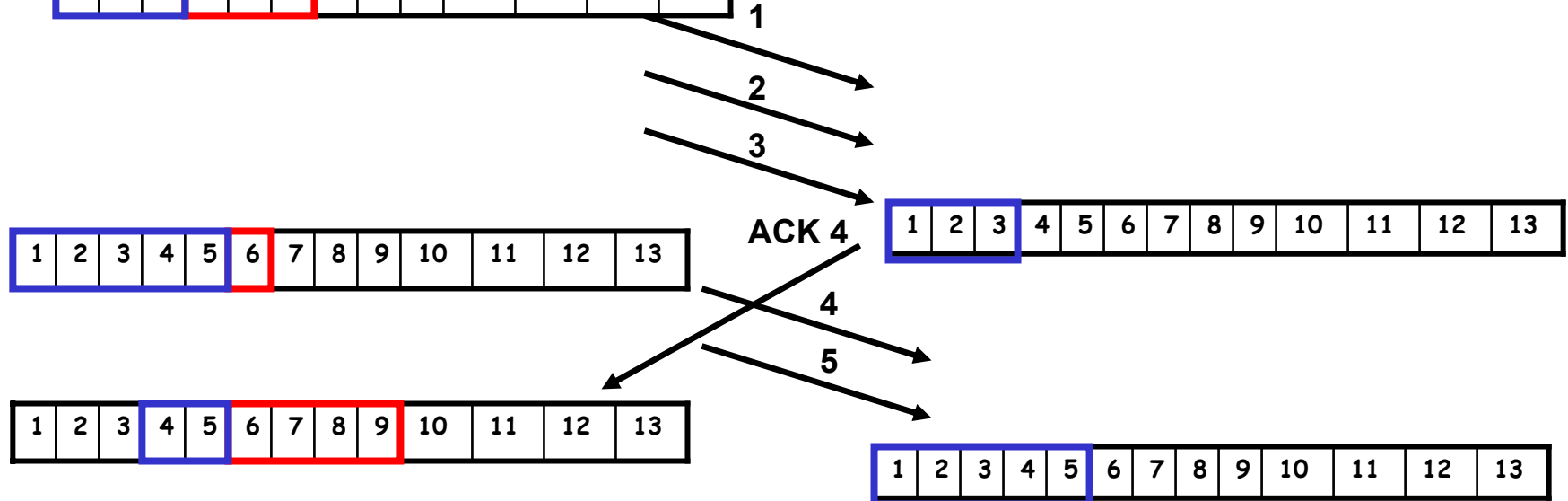
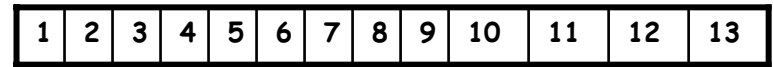


- Host B đưa cho Host A một cửa sổ kích thước là 6 (octets).
- Host A bắt đầu bằng cách gửi các octets đến Host B: octets 1, 2, và 3 và trượt cửa sổ của nó (gờ giữa) để chỉ 3 octets đã được gửi.
- Host A sẽ không tăng kích thước cửa sổ có thể sử dụng (Usable Window) của nó lên 3, cho đến khi nó nhận một hồi báo từ Host B rằng nó đã nhận được một số hoặc tất cả các octets.
- Host B, không đợi tất cả 6 octets đến, sau khi nhận octet thứ 3 gửi một ACK mong nhận có giá trị là 4 đến Host A.

## Host A - Sender



## Host B - Receiver



Window size = 6

Octets sent

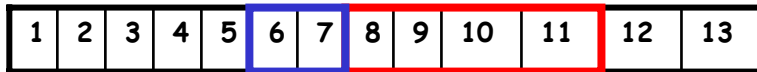
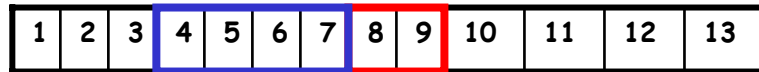
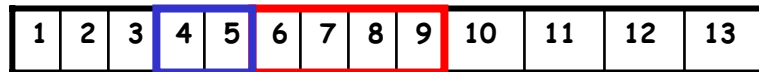
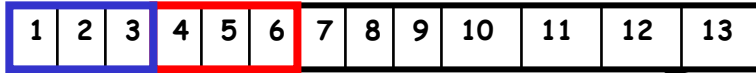
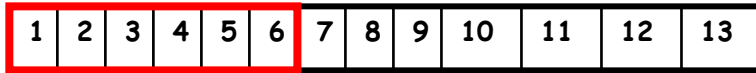
Usable Window

Not ACKed

Can send ASAP

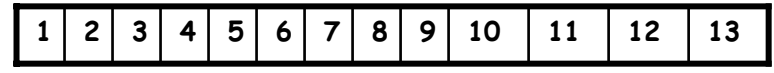
- Host A không cần phải đợi một hồi báo từ Host B để tiếp tục gửi dữ liệu, cho đến khi kích thước cửa sổ đạt đến 6, do đó nó gửi octet 4 và 5.
- Host A nhận hồi báo ACK 4 và bây giờ có thể **trượt** cửa sổ của nó cho đủ 6 octets, 3 octets đã được gửi – chưa được hồi báo cộng với 3 octets có thể được gửi ngay lập tức.

## Host A - Sender



Các cửa sổ trượt  
(tiếp theo)

## Host B - Receiver



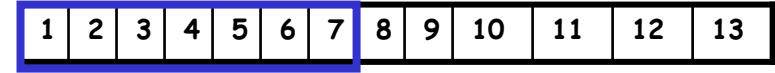
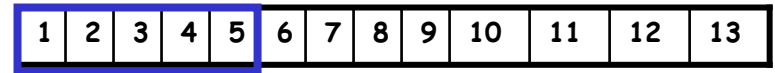
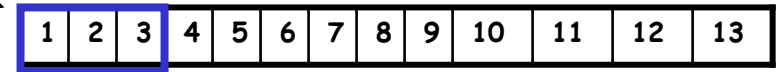
Window size = 6

Octets sent

Not ACKed

Usable Window

Can send ASAP



Transport Layer 7-72



# Chương 7 - Nội dung

- ❑ 7.1 Các dịch vụ của tầng Vận chuyển
- ❑ 7.2 Multiplexing và demultiplexing
- ❑ 7.3 Vận chuyển phi kết nối: UDP
- ❑ 7.4 Vận chuyển hướng kết nối: TCP
  - cấu trúc segment
  - quản lý kết nối
  - truyền dữ liệu tin cậy & kiểm soát luồng
- ❑ 7.5 Kiểm soát tắc nghẽn trong TCP

# Kiểm soát tắc nghẽn trong TCP

- ❑ Tiếp cận kiểm soát tắc nghẽn theo hướng end-to-end (không có sự hỗ trợ của tầng mạng)
- ❑ Bên gửi giới hạn số lượng dữ liệu chưa được báo nhận:

**$\text{LastByteSent} - \text{LastByteAcked}$**

**$\leq \min\{\text{CongWin}, \text{RcvWindow}\}$**

- ❑ Tốc độ xấp xỉ của bên gửi:

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- ❑ **CongWin** được thay đổi khi nhận thấy có tắc nghẽn trên mạng

**Bên gửi nhận thấy tắc nghẽn như thế nào?**

- ❑ Sự kiện mất mát = timeout hoặc 3 lần acks giống nhau
- ❑ Bên TCP gửi giảm tốc độ (CongWin) sau sự kiện mất mát

**Ba thành phần chính của giải thuật kiểm soát tắc nghẽn trong TCP:**

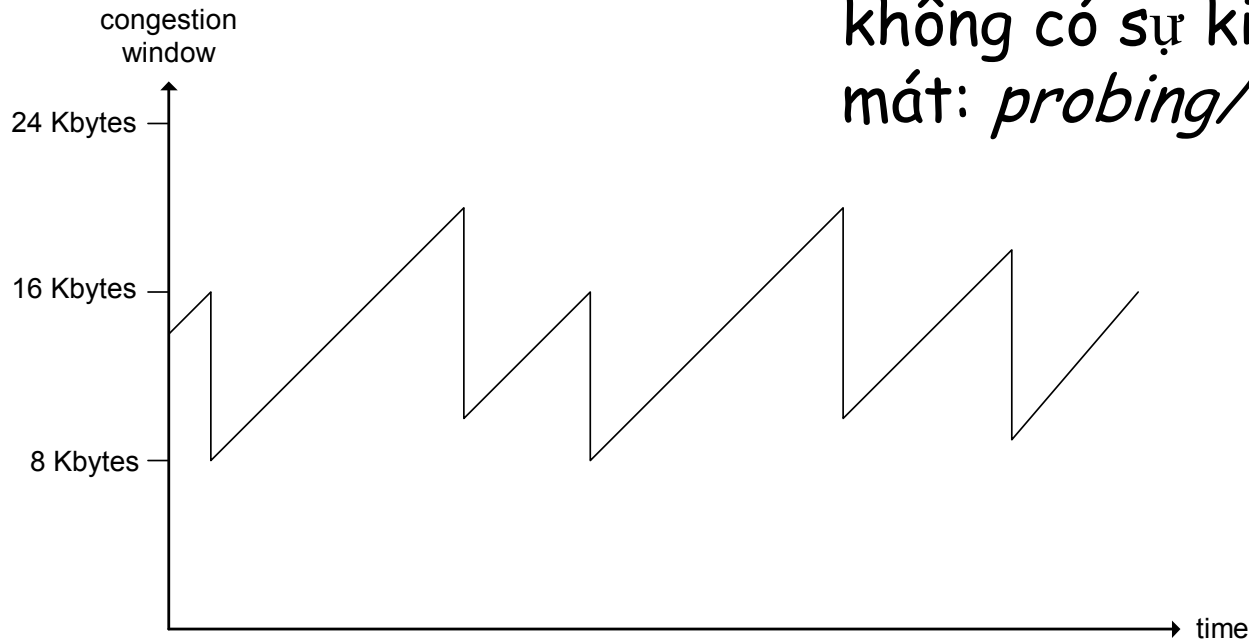
- AIMD
- slow start/bắt đầu chậm
- thận trọng/dè dặt sau các sự kiện timeout

# TCP AIMD (Tăng cộng, giảm nhân)

## multiplicative decrease:

giảm CongWin xuống một nửa sau khi có sự kiện mất mát

additive increase: tăng CongWin lên 1 MSS (Maximum Segment Size) mỗi RTT khi không có sự kiện mất mát: *probing/thăm dò*



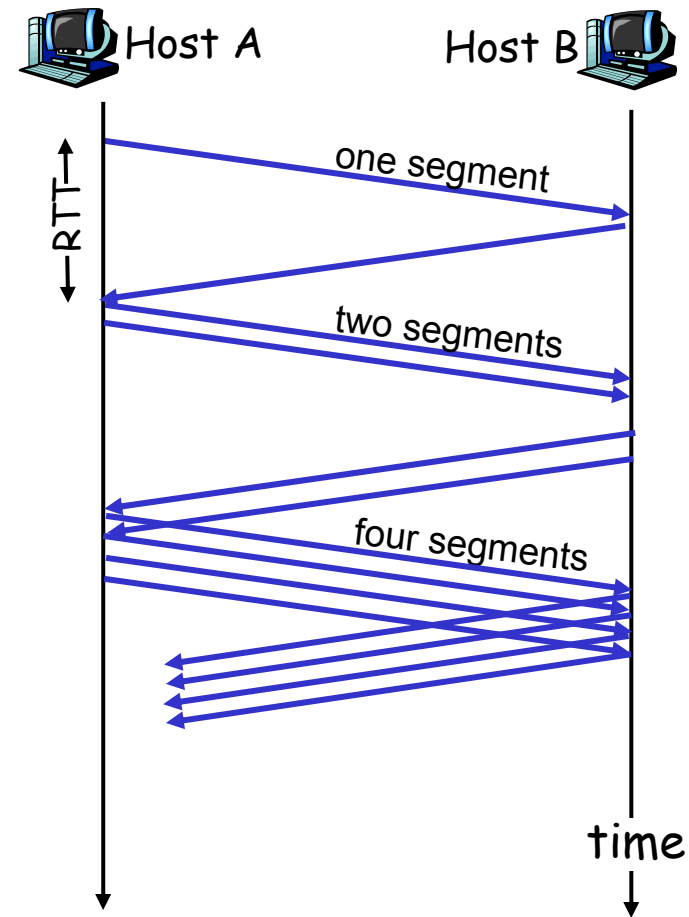
Long-lived TCP connection

# TCP Slow Start

- ❑ Khi kết nối bắt đầu,  
**CongWin = 1 MSS**
  - Ví dụ: MSS = 500 bytes & RTT = 200 msec
  - tốc độ khởi tạo = 20 kbps
- ❑ Băng thông sẵn có có thể lớn hơn rất nhiều so với MSS/RTT
  - cần thiết phải nhanh chóng nhảy lên đến tốc độ đáng kể chứ không nên tăng từ từ
- ❑ Khi kết nối bắt đầu, tăng tốc độ nhanh chóng theo hàm mũ cho đến khi gặp sự kiện mất mát đầu tiên.

# TCP Slow Start (tiếp theo)

- Khi kết nối bắt đầu, tăng tốc độ theo hàm mũ cho đến khi gặp sự mất mát đầu tiên:
  - gấp đôi CongWin sau mỗi RTT
  - được thực hiện bằng việc tăng CongWin cho mỗi ACK nhận được
- Tóm tắt: tốc độ khởi tạo thì chậm nhưng nhảy lên thật nhanh theo hàm mũ



# Sự cải tiến

- ❑ Sau 3 ACKs trùng lặp:
  - CongWin được giảm một nửa
  - sau đó nó được tăng lên tuyến tính
- ❑ Nhưng sau sự kiện timeout:
  - CongWin được đặt về 1 MSS;
  - sau đó tăng nó lên theo hàm mũ
  - khi đến một threshold (ngưỡng), tăng lên tuyến tính

## Triết lý:

- 3 ACKs trùng lặp biểu thị mạng có khả năng phân phát một số segments
- timeout trước khi 3 ACKs trùng lặp nghĩa là tình trạng tắc nghẽn được "báo động nhiều hơn"

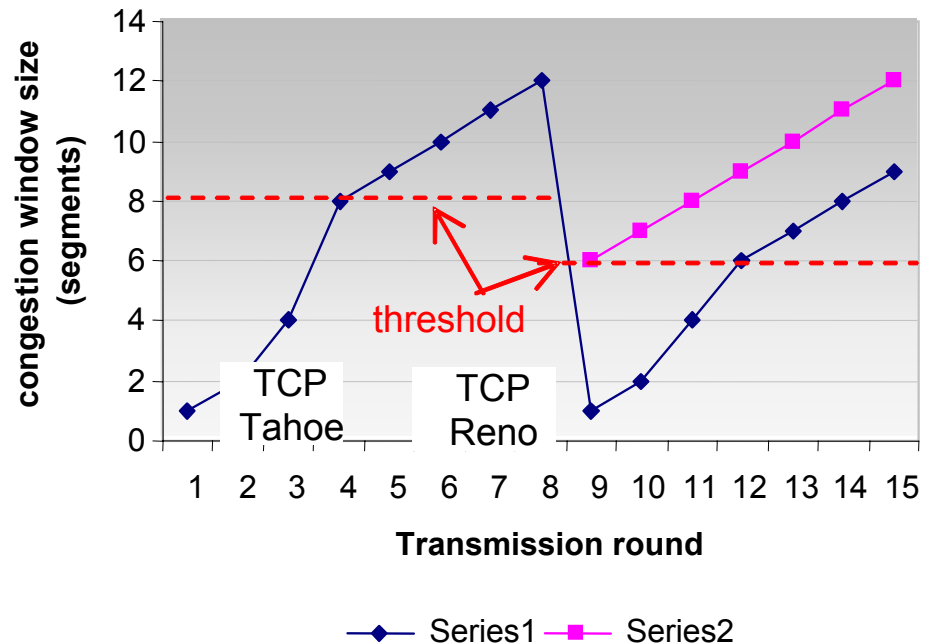
# Sự cải tiến (tiếp theo)

**Q:** Khi nào thì chuyển từ việc tăng theo hàm mũ sang tăng tuyến tính?

**A:** Khi CongWin đạt được 1/2 giá trị mà nó đã có được trước khi có sự kiện timeout.

## Sự thi hành/cài đặt:

- ❑ Sử dụng biến Threshold
- ❑ Khi có sự kiện timeout, Threshold được đặt bằng 1/2 giá trị của CongWin ngay trước khi sự kiện mất mát xảy ra



# Tóm tắt về kiểm soát tắc nghẽn trong TCP

- ❑ Khi CongWin dưới ngưỡng/Threshold, bên gửi hoạt động ở pha **slow-start**, cửa sổ được tăng theo hàm mũ.
- ❑ Khi CongWin trên ngưỡng/Threshold, bên gửi hoạt động ở pha **congestion-avoidance**, cửa sổ tăng tuyến tính.
- ❑ Khi một **ACK trùng lặp ba lần** xuất hiện, Threshold được đặt về CongWin/2 và CongWin được đặt về ngưỡng/Threshold.
- ❑ Khi **timeout** xuất hiện, Threshold được đặt về CongWin/2 và CongWin được đặt về 1 MSS.



# Chương 7: Tóm tắt

- ❑ Các nguyên lý đằng sau các dịch vụ của tầng vận chuyển:
  - multiplexing, demultiplexing
  - chuyển giao dữ liệu tin cậy
  - kiểm soát luồng
  - kiểm soát tắc nghẽn
- ❑ Thuyết minh bằng ví dụ cụ thể và sự cài đặt/sự thi hành các dịch vụ đã nêu trên môi trường Internet
  - UDP
  - TCP