



# Mạng Máy Tính (Computer Networking)

## Tầng vận chuyển (Transport Layer)

TS. Nguyễn Mạnh Cương  
Khoa CNTT, ĐH Nha Trang



# 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 của TCP



# Nội dung

- **Tổng quan về tầng giao vận**
  - Tổng quan về tầng vận chuyển
  - Hướng liên kết vs. Không liên kết
  - Giao thức TCP và UDP
- Dồn kênh/phân kênh (Multiplexing và demultiplexing)
- Vận chuyển phi kết nối: User Datagram Protocol (UDP)
- Vận chuyển hướng kết nối: TCP
- Kiểm soát tắc nghẽn của TCP



# Vị trí trong kiến trúc phân tầng

Application

(HTTP, Mail, ...)

**Transport**

(UDP, TCP ...)

Network

(IP, ICMP...)

Datalink

(Ethernet, ADSL...)

Physical

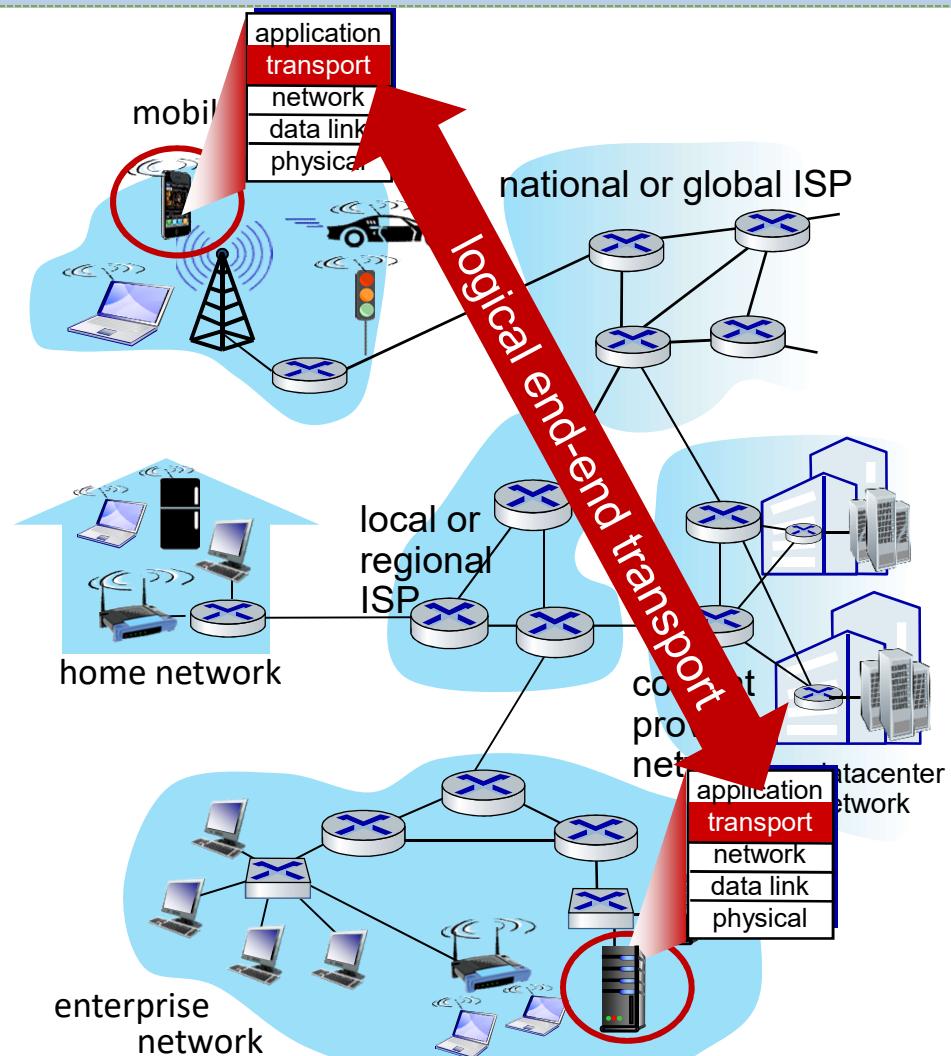
(bits...)

- Hỗ trợ các ứng dụng trên mạng
- Truyền dữ liệu giữa các ứng dụng trên các thiết bị đầu cuối
- Chọn đường và chuyển tiếp gói tin giữa các mạng
- Hỗ trợ việc truyền thông giữa các máy trên cùng 1 mạng
- Truyền và nhận dòng bit trên đường truyền vật lý



# Các dịch vụ vận chuyển và giao thức

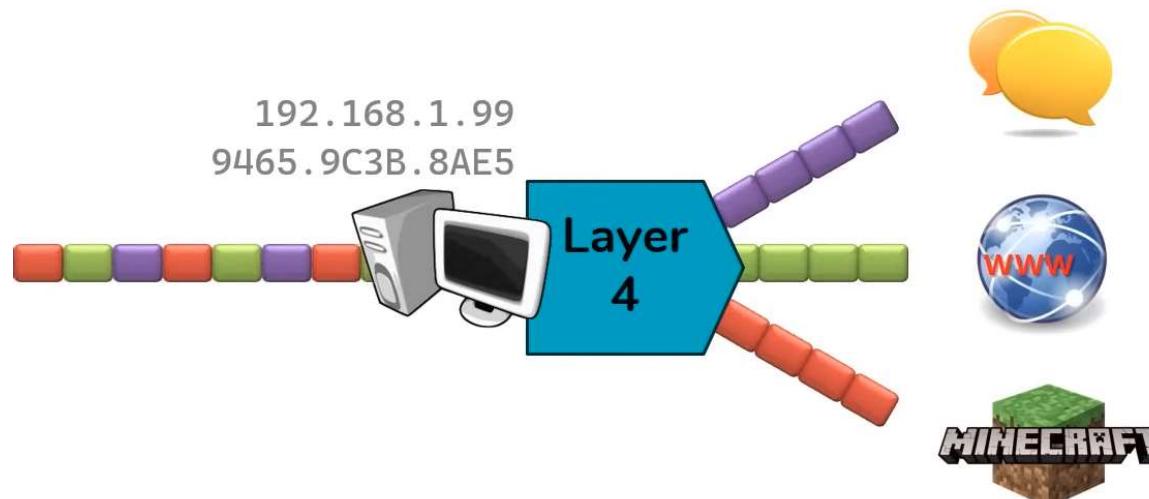
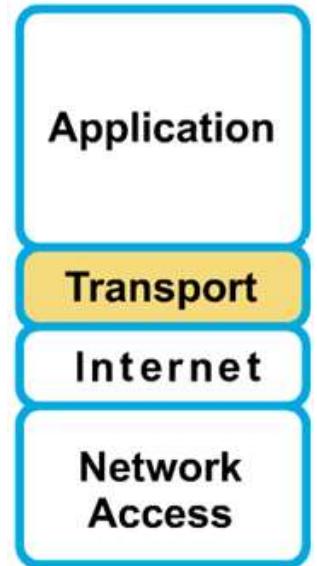
- Cung cấp **truyền thông logic** giữa các tiến trình ứng dụng chạy trên các trạm khác nhau
- Dòng dữ liệu tầng 4 là một **kết nối logic** giữa các điểm đầu-cuối của mạng, và cung cấp các dịch vụ vận chuyển từ một trạm nguồn đến một trạm đích nào đó.
- Vì sao gọi là “**truyền thông logic**” hoặc “**kết nối logic**”?





# Tổng quan về tầng vận chuyển

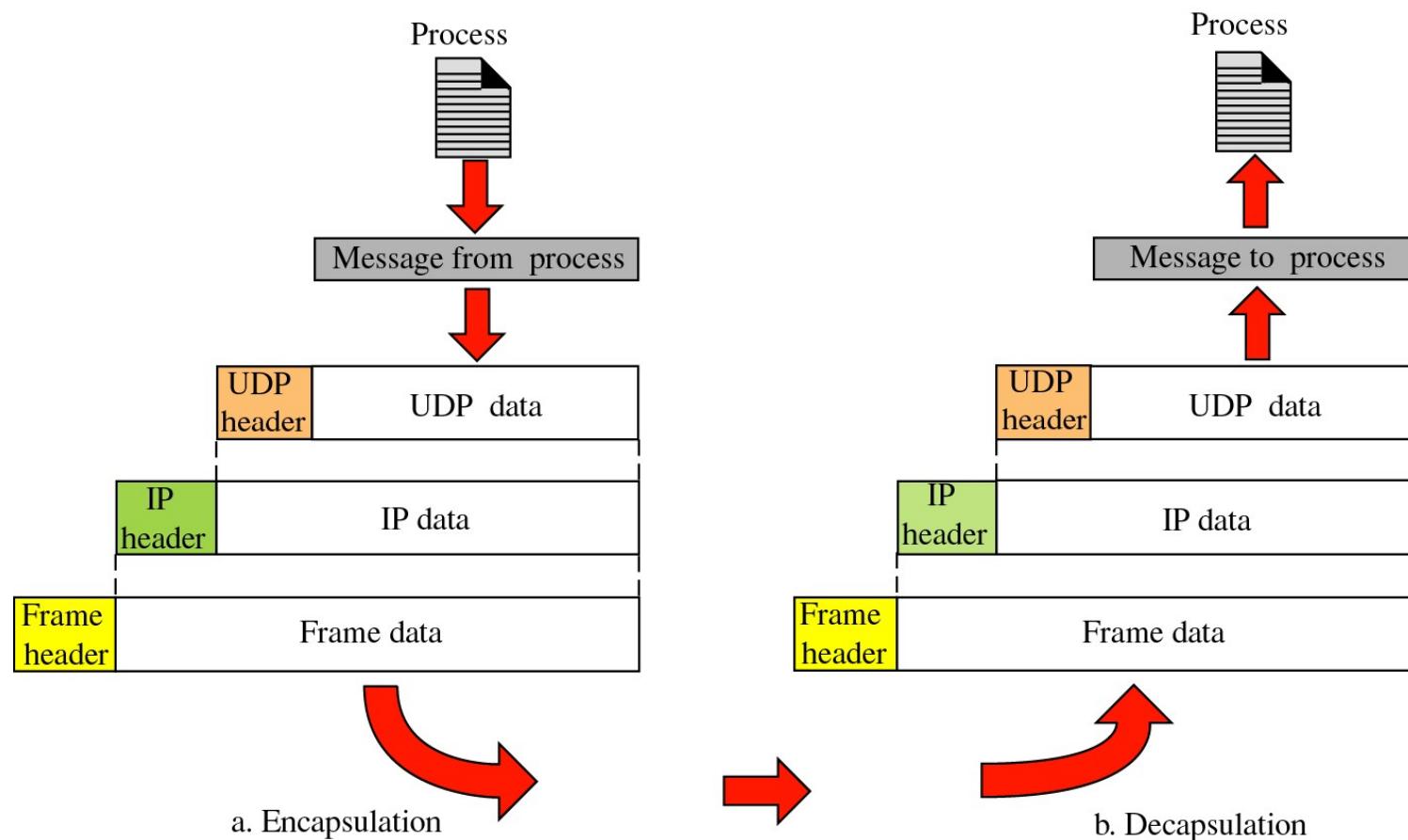
- Tầng Vận chuyển cho phép một thiết bị của người sử dụng **phân đoạn dữ liệu** của các ứng dụng ở tầng trên để đặt vào cùng dòng dữ liệu tầng 4, và cho phép thiết bị nhận ráp nối lại các đoạn dữ liệu đó để chuyển lên cho tầng trên.
- Lưu ý: Tầng 4 và các tầng phía trên được tạo ra bởi các thiết bị đầu cuối (computer).





# Tổng quan về tầng vận chuyển

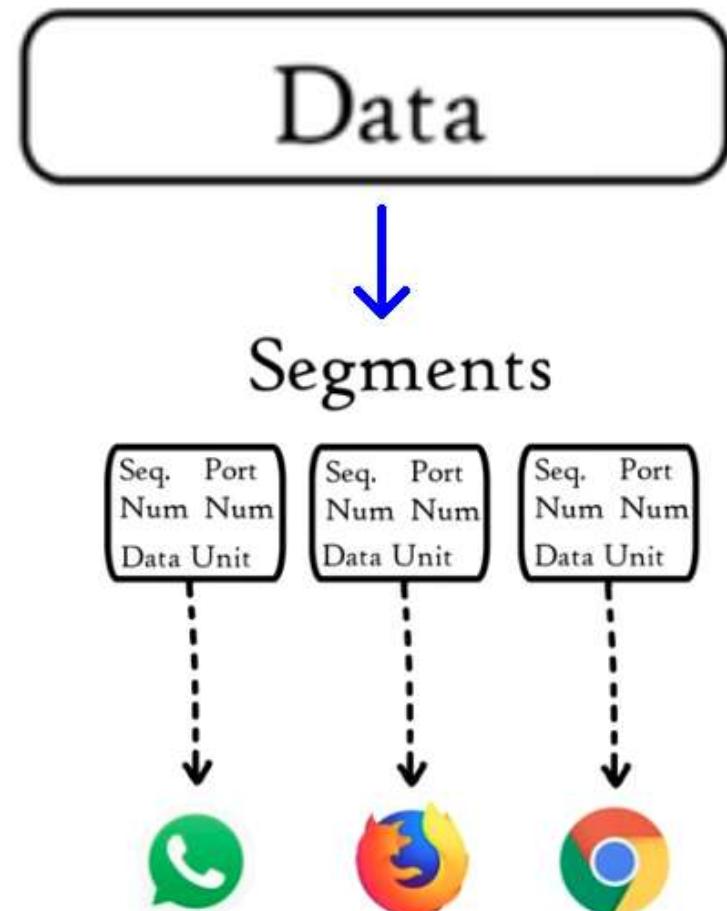
- Dữ liệu tầng ứng dụng được chia tách và đóng gói lại với **TCP/UDP header** rồi chuyển tiếp xuống tầng mạng.





# Các dịch vụ vận chuyển và giao thức

- Các giao thức vận chuyển chạy trên các hệ thống đầu cuối
  - Bên gửi: chẻ các data 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 vs. 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 mà tầng mạng cung cấp

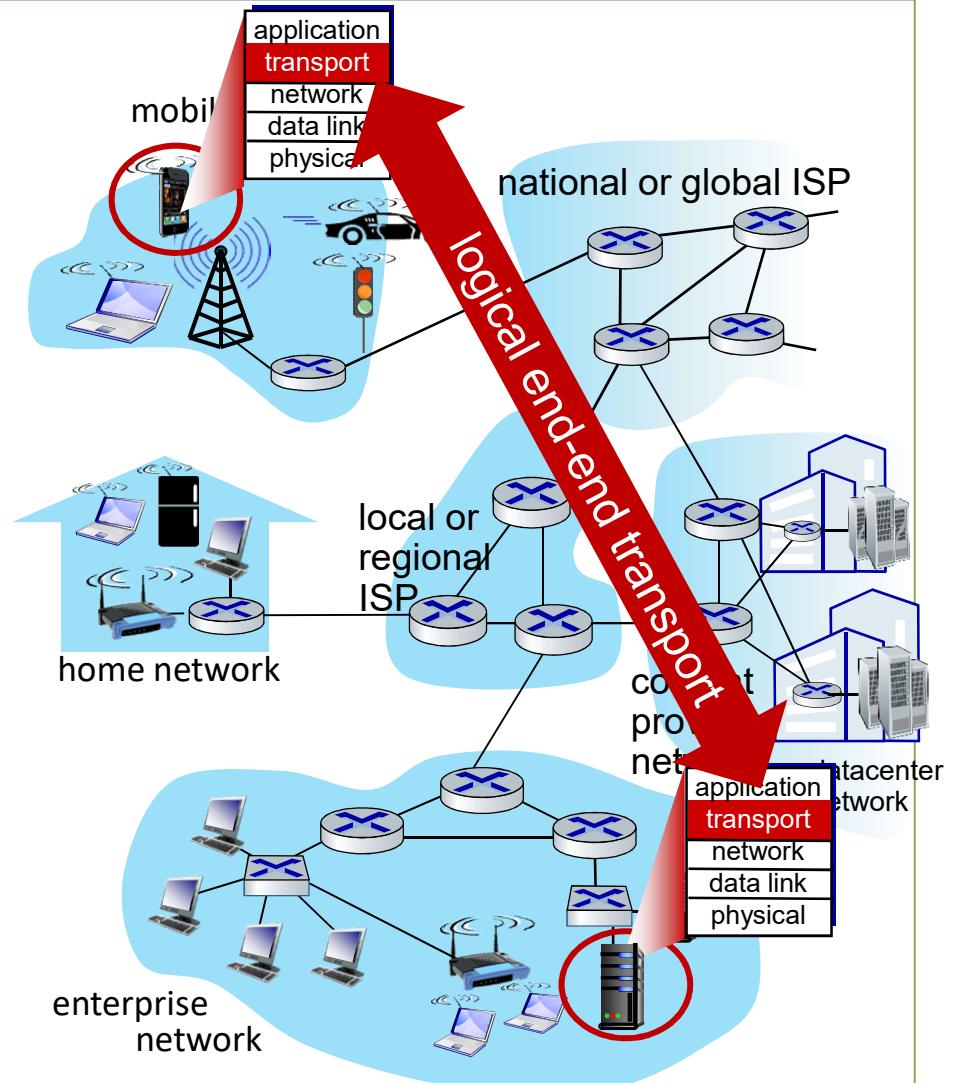
**Tương tự như các hộ gia đình:**  
**12 kids gửi thư cho 12 kids**

- 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)
  - Đơn giản, thực hiện theo “best-effort”
  - Không đảm bảo về các sự trễ
  - Không đảm bảo các thông số về băng thông





# Tại sao lại cần 2 loại dịch vụ?

- Các yêu cầu đến từ tầng ứng dụng là đa dạng. Tùy theo độ ưu tiên của từng ứng dụng:
  - Các ứng dụng cần dịch vụ với 100% độ tin cậy như mail, web, truyền file... → sử dụng dịch vụ của TCP
  - Các ứng dụng cần chuyển dữ liệu nhanh, có khả năng chịu lỗi, e.g. VoIP, Video Streaming → Sử dụng dịch vụ của UDP



# Ứng dụng và dịch vụ giao vận

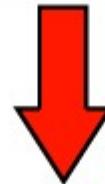
Ứng dụng	Giao thức ứng dụng	Giao thức giao vận
e-mail	SMTP	TCP
remote terminal access	Telnet	TCP
Web	HTTP	TCP
file transfer	FTP	TCP
streaming multimedia	giao thức riêng (e.g. RealNetworks)	TCP or UDP
Internet telephony	giao thức riêng (e.g., Vonage,Dialpad)	thường là UDP



# Địa chỉ Socket

IP address

200.23.56.8



Port number

69



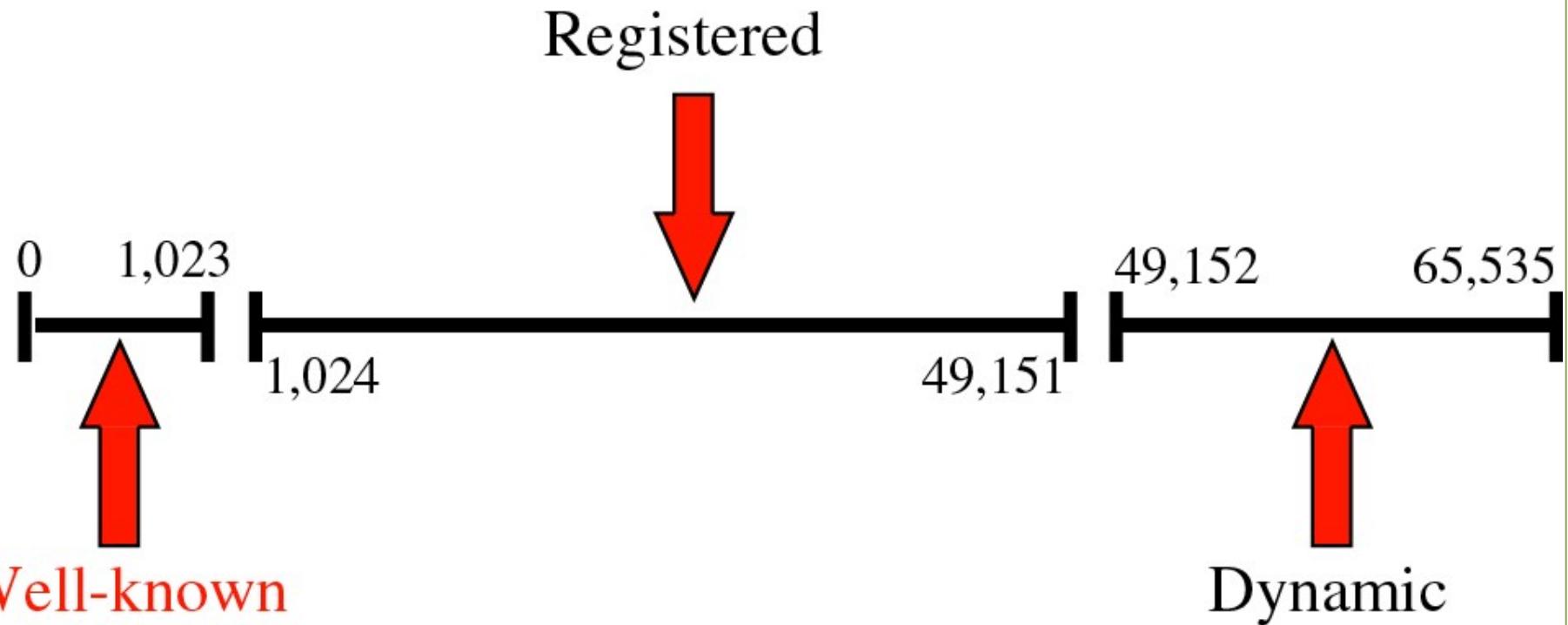
200.23.56.8

69

Socket address

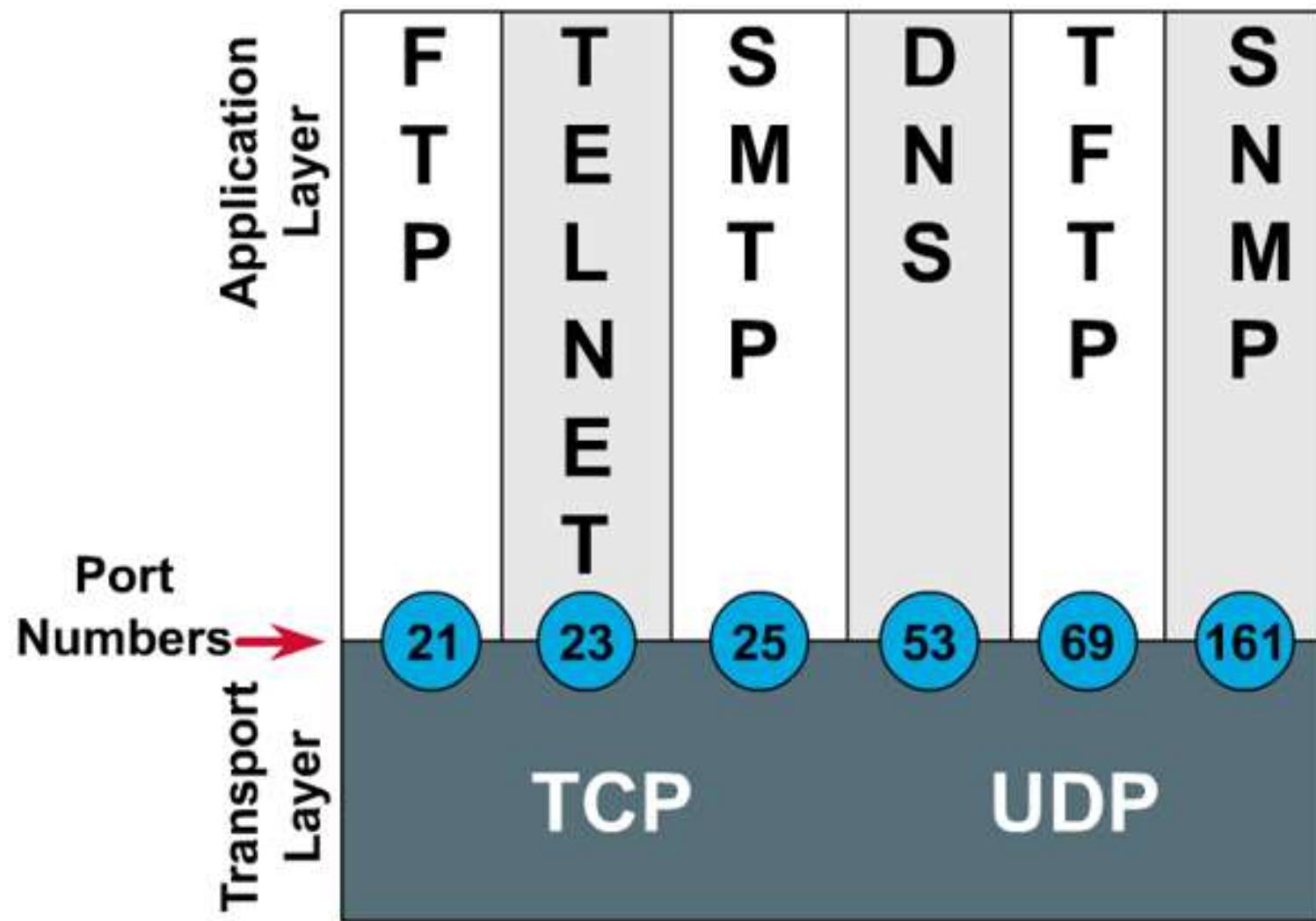


# Dải số hiệu cổng do IANA quy định





# Port Number





# Nội dung

- Tổng quan về tầng giao vận
- Dồn kênh/phân kênh (Multiplexing và demultiplexing)
  - Dồn kênh/phân kênh
  - Kiểm soát lỗi
- Vận chuyển phi kết nối: User Datagram Protocol (UDP)
- Vận chuyển hướng kết nối: TCP
- Kiểm soát tắc nghẽn của TCP



# Dồn kênh/phân kênh

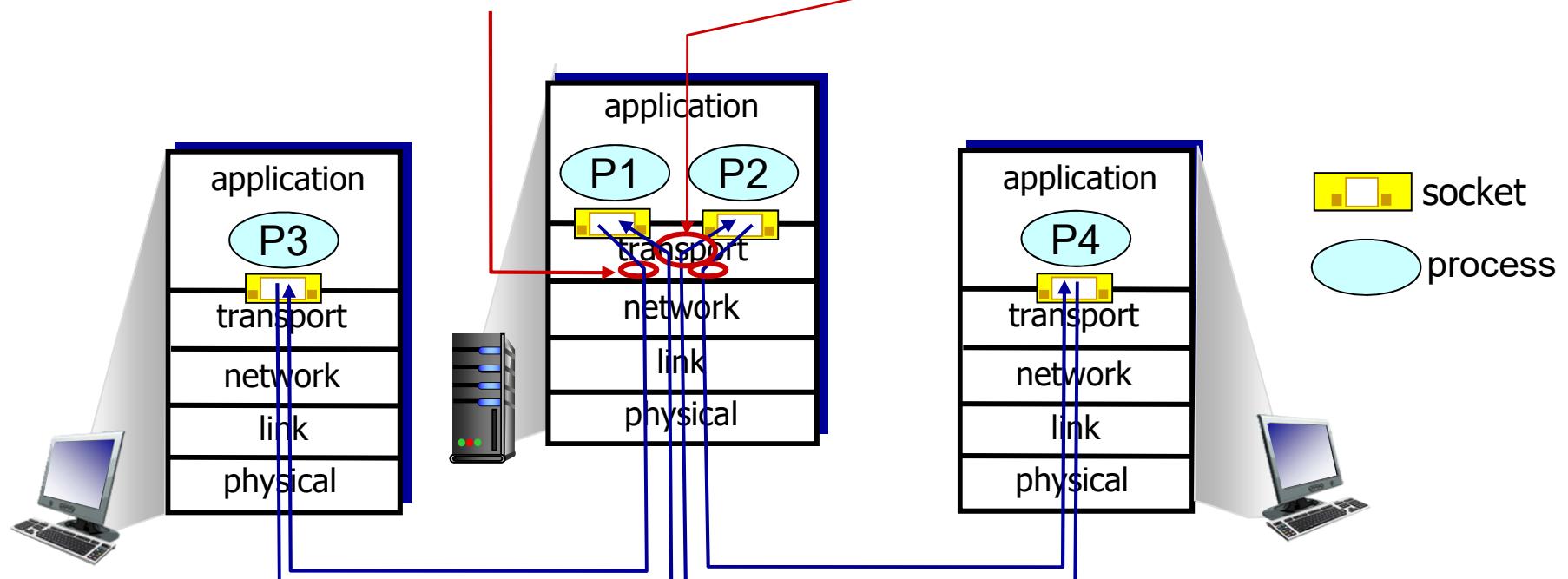
## Multiplexing/demultiplexing

*multiplexing at sender:*

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

*demultiplexing at receiver:*

Sử dụng các thông tin transport header để gửi các segments đến đúng socket

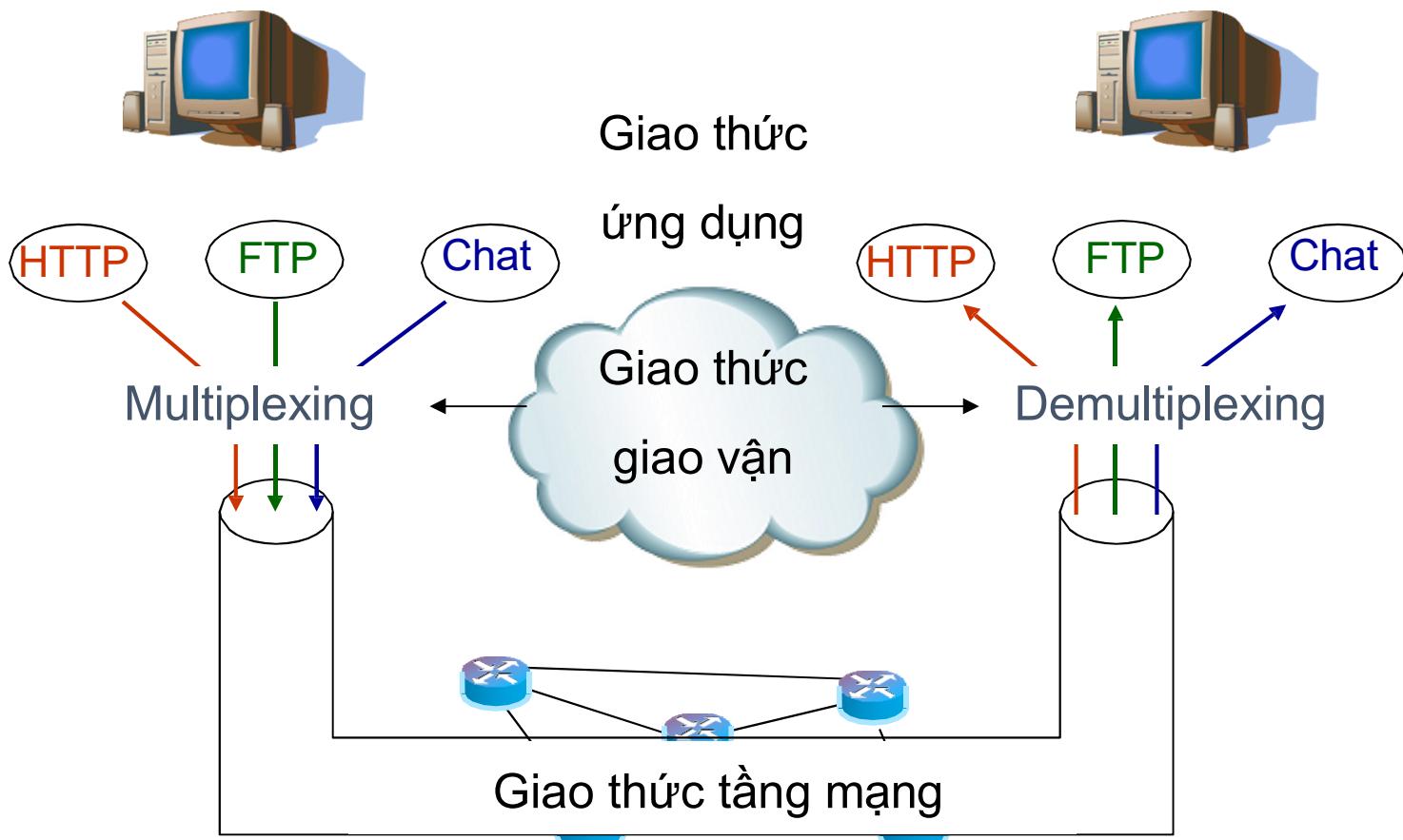


Là các quá trình đóng gói/rút tách thông tin giúp gói tin được gửi đến chính xác tiến trình cần



# Dồn kênh/phân kênh

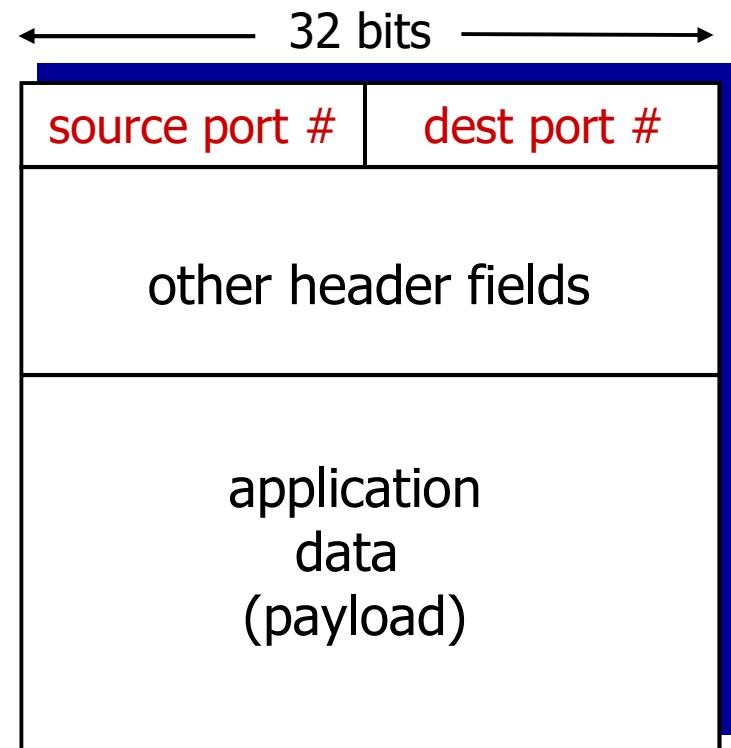
## Multiplexing/demultiplexing





# Demultiplexing làm việc như thế nào?

- Host lấy IP từ các datagrams
  - Mỗi datagram chứa thông tin về **source IP address**, **destination IP address**
  - Mỗi datagram chứa 1 **segment** thông tin transport-layer, segment này chứa số **port** của source, destination.
- Host sử dụng **IP addresses & port numbers** để gửi segment đến chính xác socket cần gửi.



TCP/UDP segment format



# Kiểm soát lỗi

- Sử dụng **CRC** hoặc **Checksum**
- **Checksum**
  - Phát hiện lỗi bit trong các đoạn tin/gói tin
  - Nguyên lý giống như checksum (16 bits) của giao thức IP
- **Nguyên lý checksum**
  - Dữ liệu cần gửi được chia thành các đoạn bằng nhau
  - Các đoạn được tính tổng với nhau, nếu có nhò thì cộng giá trị nhò vào tổng
  - Đảo tổng thu được checksum



# Ví dụ

Sender's End		Receiver's End	
Frame 1:	11001100	Frame 1:	11001100
Frame 2:	+ 10101010	Frame 2:	+ 10101010
Partial Sum:	1 01110110	Partial Sum:	1 01110110
	+ 1		+ 1
	01110111		01110111
Frame 3:	+ 11110000	Frame 3:	+ 11110000
Partial Sum:	1 01100111	Partial Sum:	1 01100111
	+ 1		+ 1
	01101000		01101000
Frame 4:	+ 11000011	Frame 4:	+ 11000011
Partial Sum:	1 00101011	Partial Sum:	1 00101011
	+ 1		+ 1
Sum:	00101100	Sum:	00101100
Checksum:	11010011	Checksum:	11010011
		Sum:	11111111
		Complement:	00000000
		Hence accept frames.	



# Demultiplexing trong phi kết nối (UDP)

- 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:

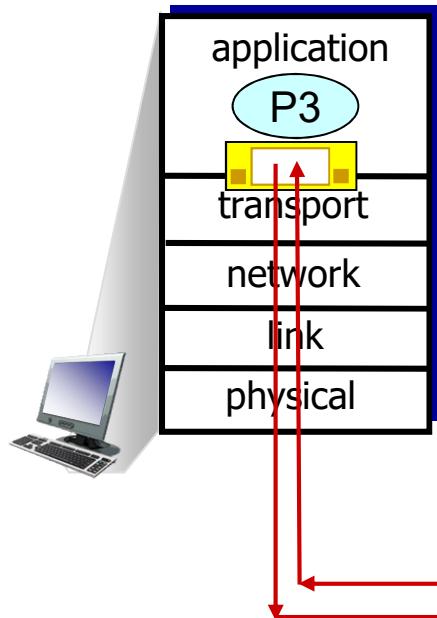
(địa chỉ IP đích, số hiệu cổng đích)

- 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

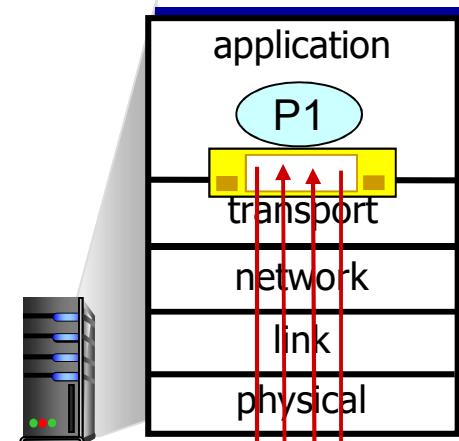


# Ví dụ: Demultiplexing trong phi kết nối

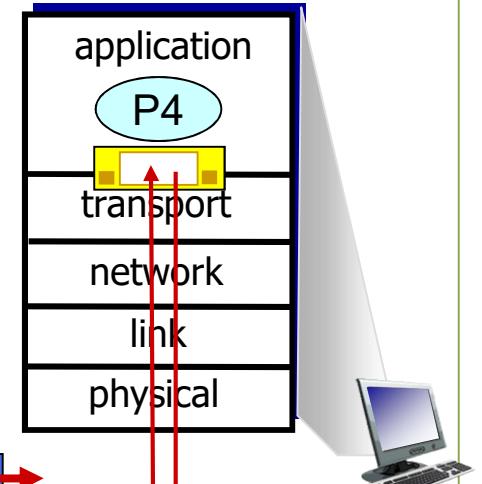
```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```



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



```
DatagramSocket mySocket1  
= new DatagramSocket  
(5775);
```



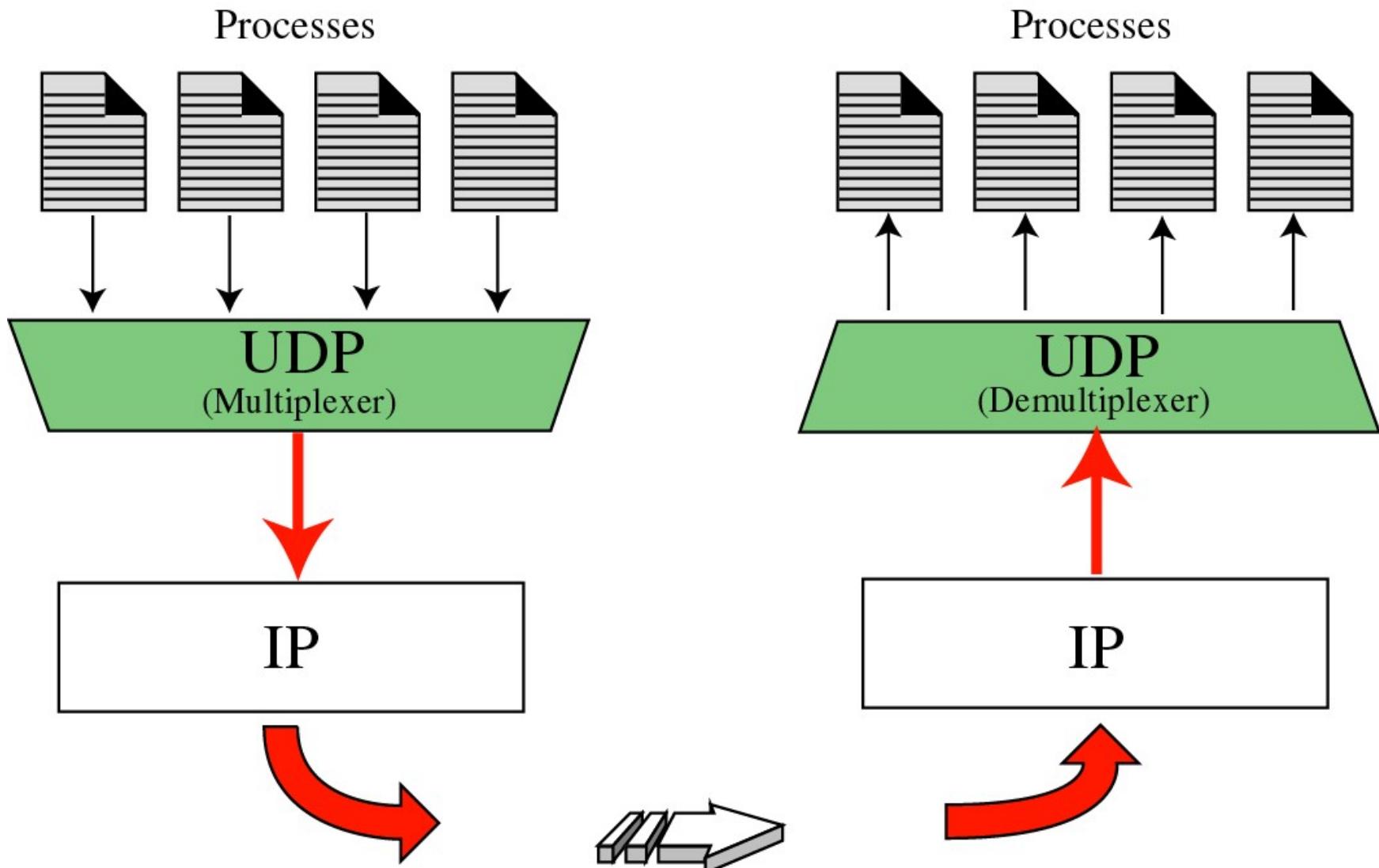
source port: 9157  
dest port: 6428

source port: ?  
dest port: ?



# Dồn kênh/phân kênh

## Multiplexing/demultiplexing



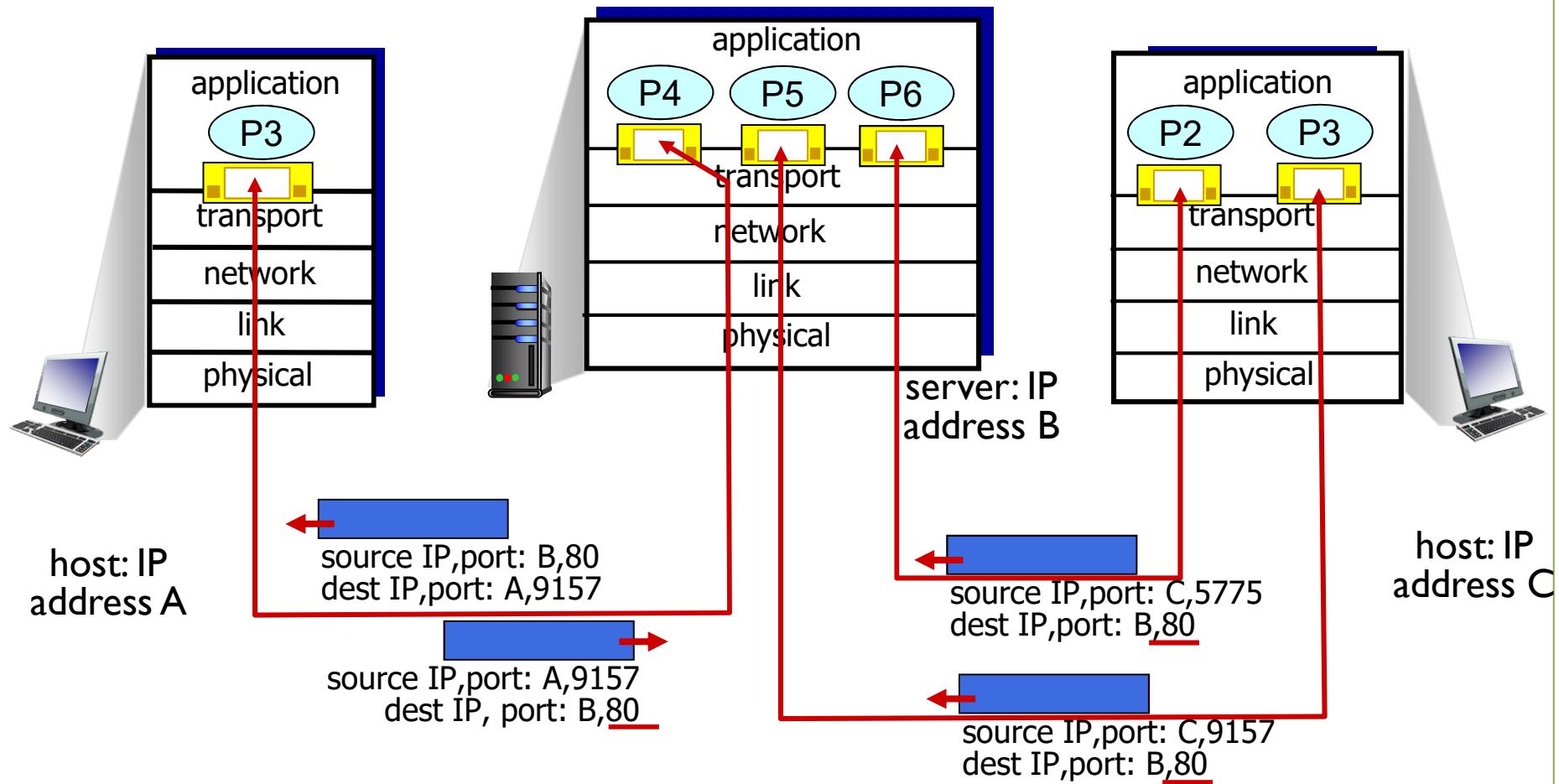


# Demultiplexing trong hướng kết nối (TCP)

- 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: Ví dụ



Three segments, all destined to IP address: B,  
dest port: 80 are demultiplexed to *different* sockets



# Nội dung

- Tổng quan về tầng giao vận
- Dồn kênh/phân kênh (Multiplexing và demultiplexing)
- Vận chuyển phi kết nối: User Datagram Protocol (UDP)
  - Khuôn dạng gói tin
  - UDP checksum
- Vận chuyển hướng kết nối: TCP
- Kiểm soát tắc nghẽn của TCP



# Giao thức dạng “Best effort”

## ■ Dùng UDP khi nào?

- Cung cấp dịch vụ theo hướng “best effort”, các UDP segments có thể:
  - Lost (mất gói)
  - Bất đồng bộ (out-of-order)
- Không có quản lý tắc nghẽn: UDP cứ gửi dữ liệu nhanh nhất, nhiều nhất nếu có thể

## ■ Không cần thiết lập liên kết

- Không tạo kết nối trước khi truyền giữa bên gửi và nhận
- Mỗi UDP segment độc lập, không phụ thuộc vào nhau.



# Giao thức dạng “Best effort”

- UDP có những chức năng cơ bản gì?
  - Dồn kênh/phân kênh
  - Phát hiện lỗi bit bằng checksum
- Ứng dụng UDP
  - Các ứng dụng streaming multimedia (livestream...), DNS
  - SNMP (các giao thức hỗ trợ giám sát mạng)

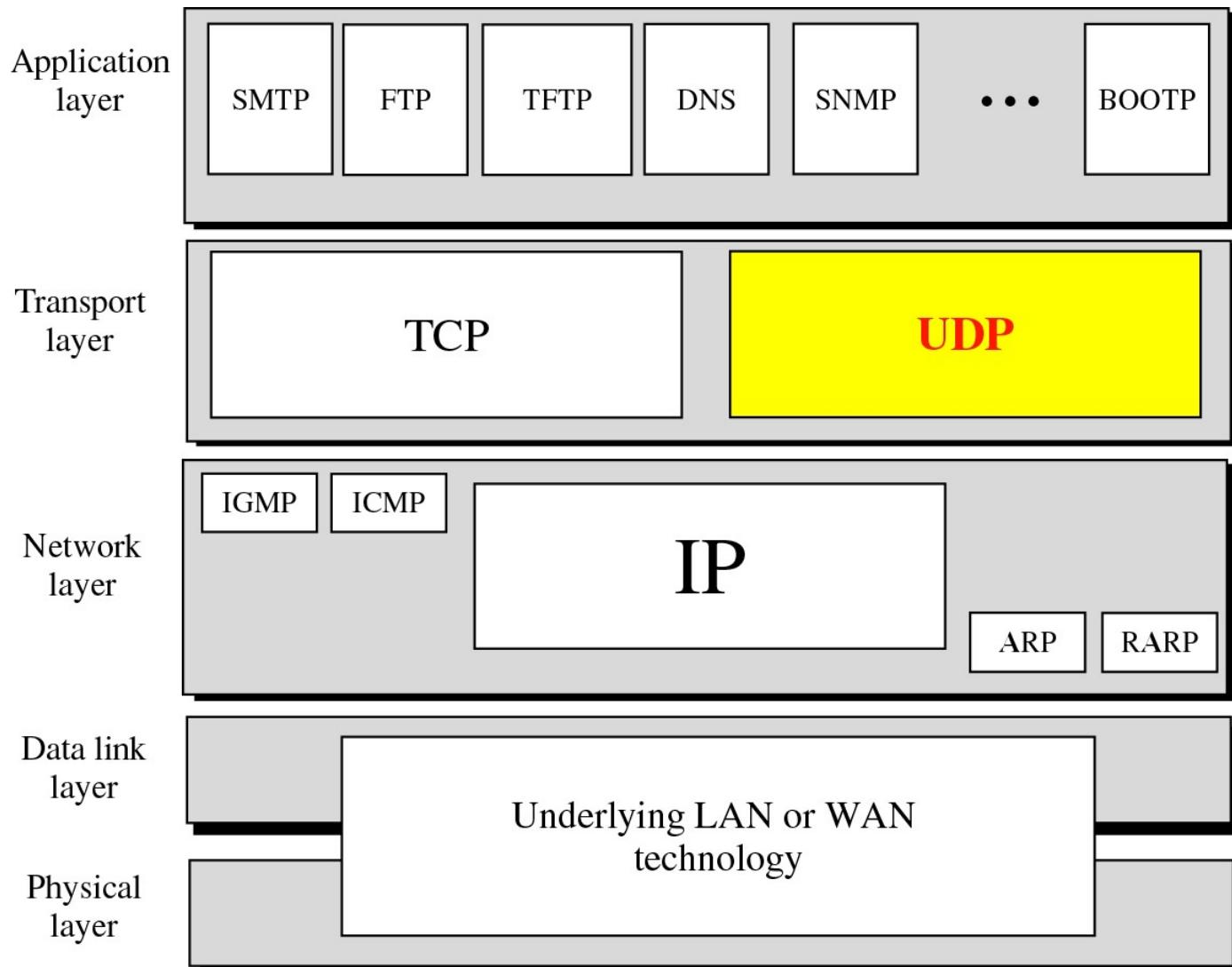


# Tại sao lại cần đến UDP?

- Không thiết lập kết nối (nó có thể 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



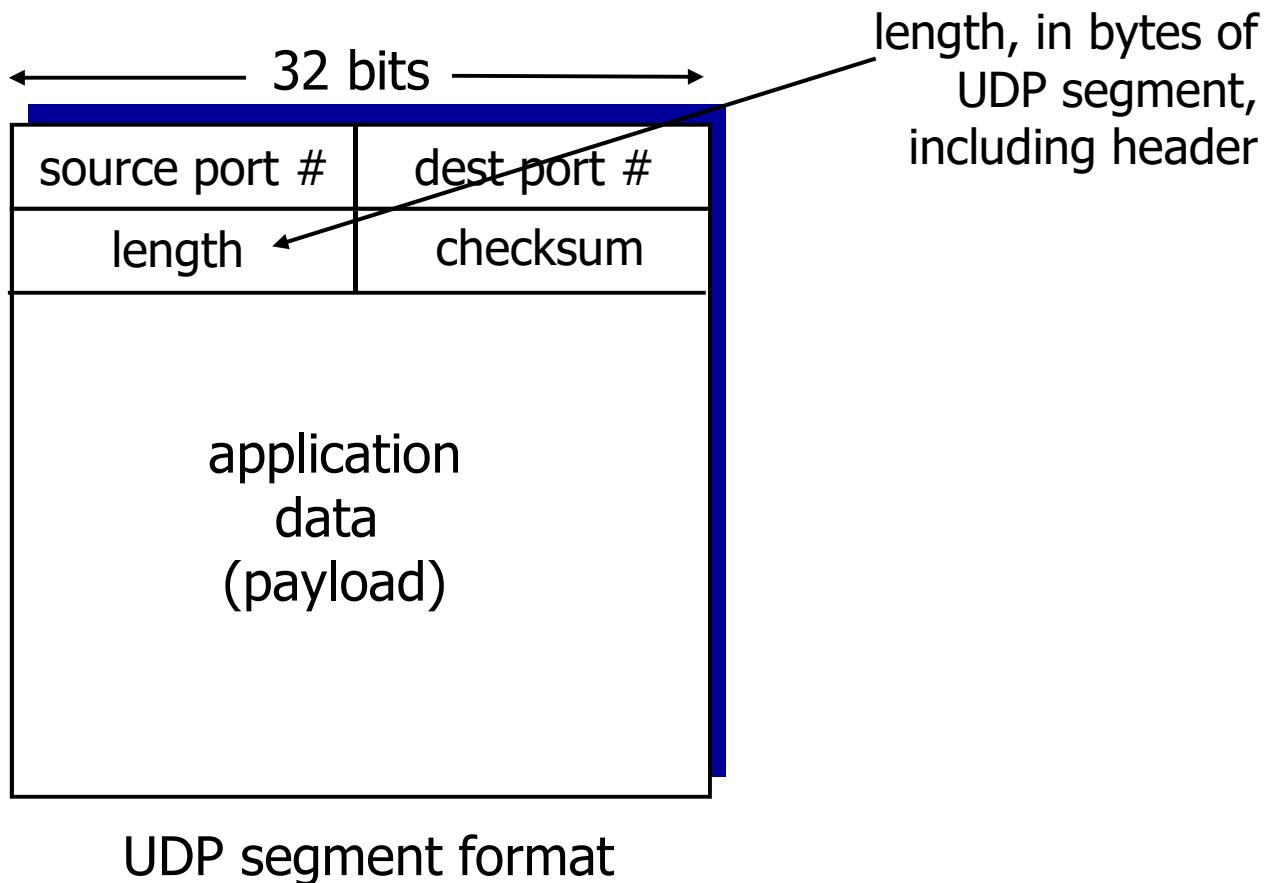
# Vị trí của UDP trong chồòng giao thức TCP/IP





# UDP Segment Header

- UDP sử dụng đơn vị dữ liệu gọi là –datagram (bức tin)





# 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. Những vẫn có thể có lỗi?



# Internet Checksum: example

example: add two 16-bit integers

Giả sử data gồm 2 từ:

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
	<hr/>															
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1



# Các vấn đề của UDP

- Không có kiểm soát tắc nghẽn
- Làm Internet bị quá tải
- Không bảo đảm được độ tin cậy
- Các ứng dụng phải cài đặt cơ chế tự kiểm soát độ tin cậy
- Việc phát triển ứng dụng sẽ phức tạp hơn

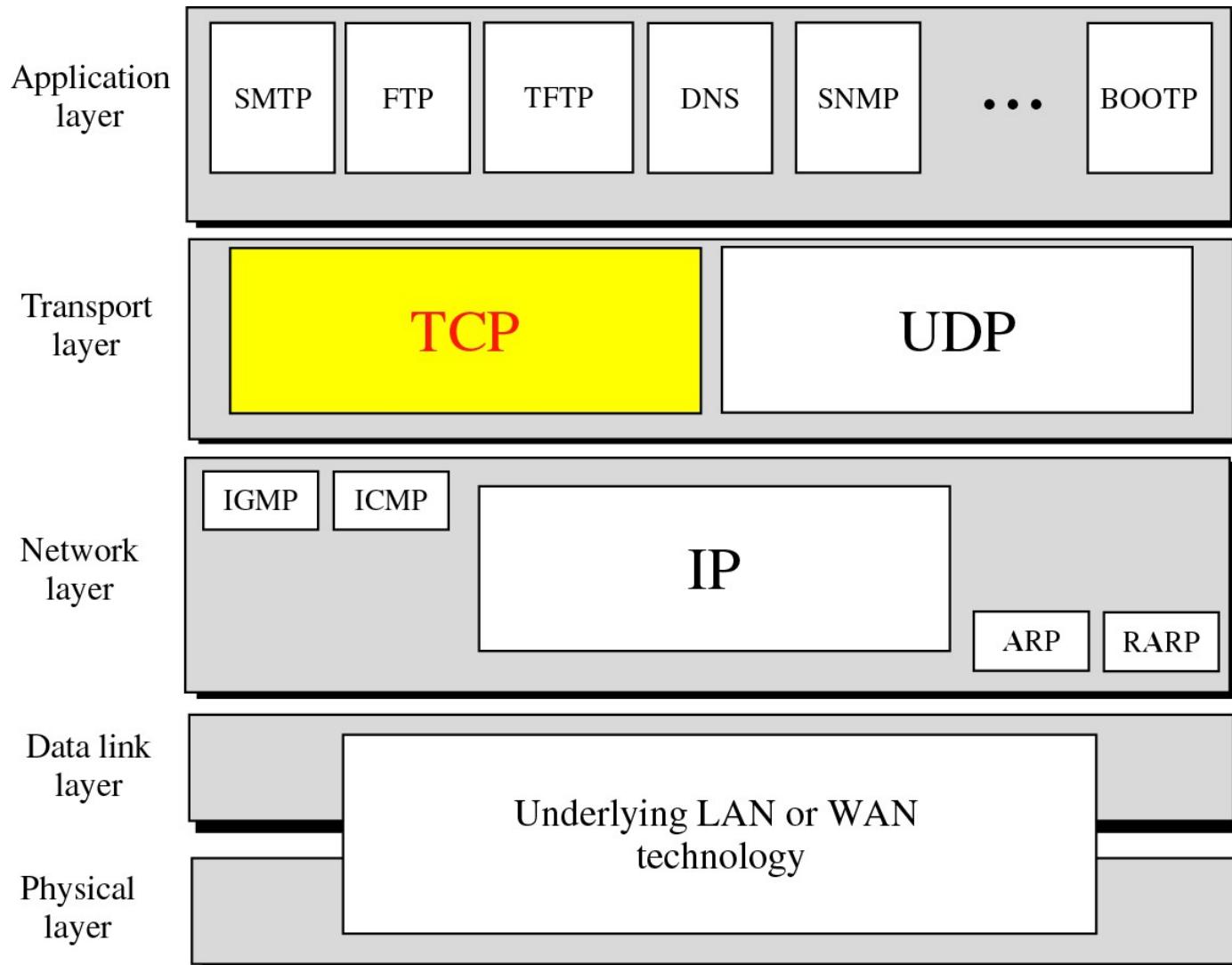


# Nội dung

- Tổng quan về tầng giao vận
- Dồn kênh/phân kênh (Multiplexing và demultiplexing)
- Vận chuyển phi kết nối: User Datagram Protocol (UDP)
- **Vận chuyển hướng kết nối: TCP**
  - Cấu trúc đoạn tin TCP
  - Quản lý kết nối
  - Kiểm soát luồng
  - Kiểm soát tắc nghẽn
- Kiểm soát tắc nghẽn của TCP



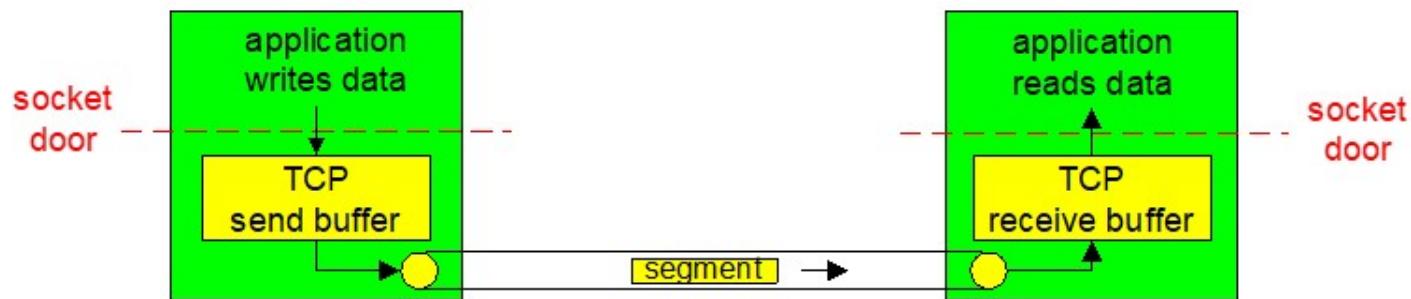
# Vị trí của TCP trong chồòng giao thức TCP/IP





# Tổng quan về TCP

- **Point-to-point**
  - 1 sender, 1 receiver
- **Giao thức hướng liên kết**
  - bắt tay (trao đổi các thông điệp điều khiển) giữa bên gửi và bên nhận trước khi trao đổi dữ liệu
- **Giao thức truyền dữ liệu theo dòng byte, tin cậy**
  - Sử dụng vùng đệm gởi & nhận
- **Truyền theo kiểu pipeline, sử dụng sliding windows.**
  - kiểm soát tắc nghẽn và kiểm soát luồng (kích cỡ cửa sổ)





# Tổng quan về TCP

## ■ Kiểm soát luồng

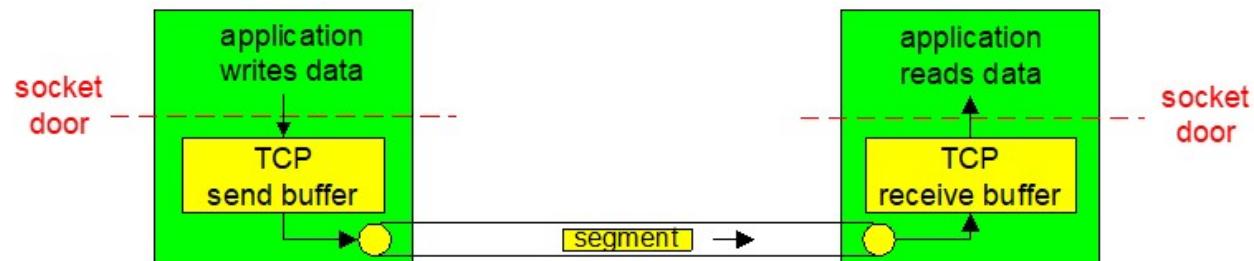
- Bên gửi không làm quá tải bên nhận

## ■ Kiểm soát tắc nghẽn

- Việc truyền dữ liệu không nên làm tắc nghẽn mạng (thực tế: luôn có tắc nghẽn)

## ■ Dữ liệu song công hoàn toàn

- Luồng dữ liệu hai hướng trên cùng kết nối
- MSS: maximum segment size





# Cấu trúc TCP segment

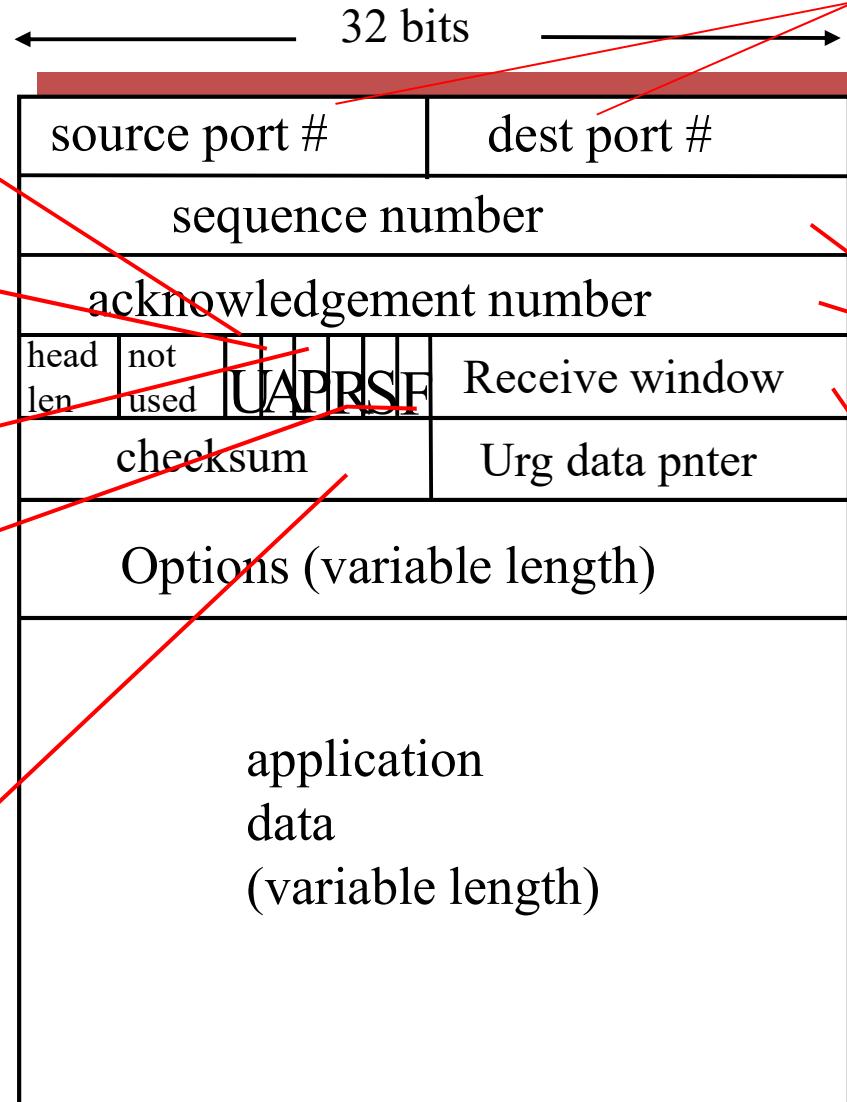
URG: dữ liệu khẩn  
(thường không dùng đến)

ACK: số ACK  
có hiệu lực

PSH: đẩy dl ngay  
(thường không dùng đến)

RST, SYN, FIN:  
các cờ để thiết lập,  
ngắt kết nối

Internet  
checksum  
(tương tự như UDP)

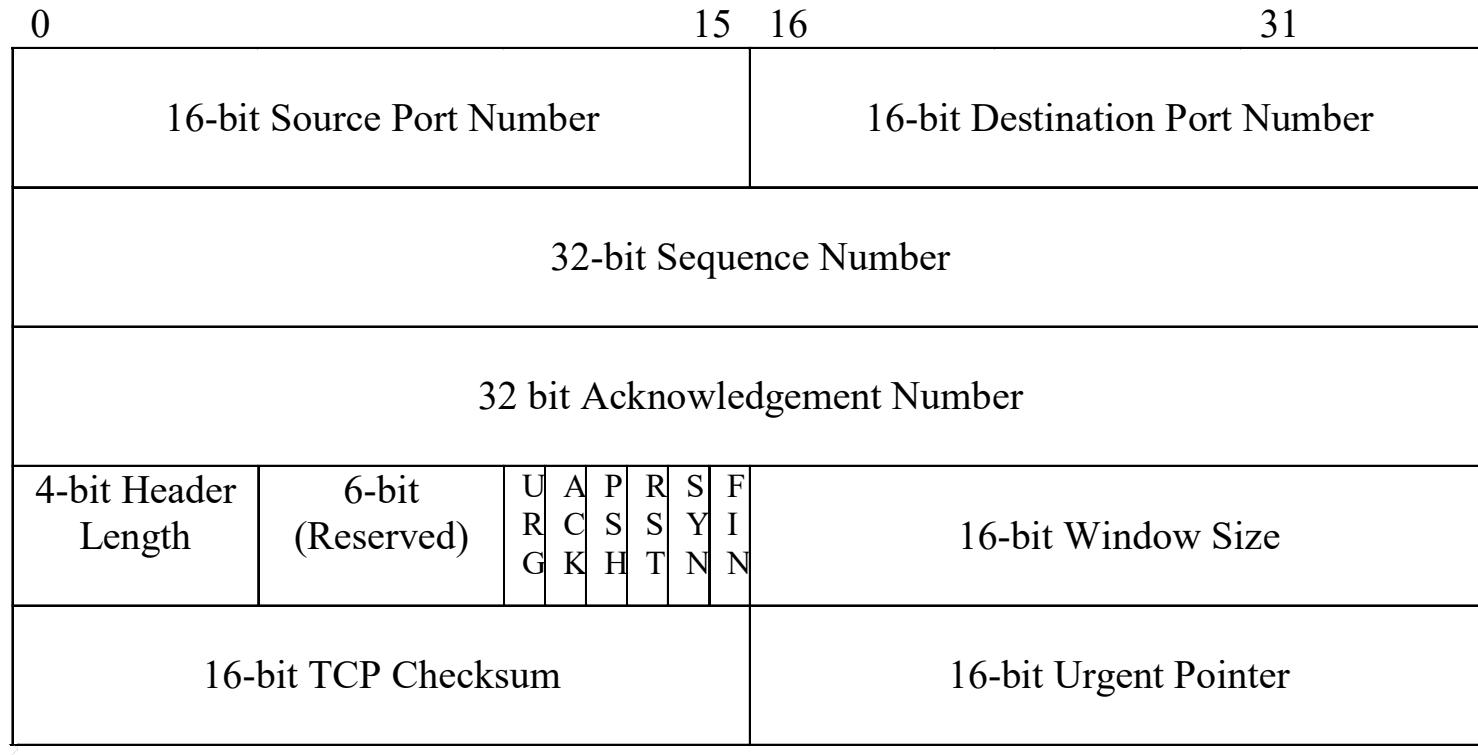


Để xác định các bytes dữ liệu  
(không phải segments!)

# bytes  
bên nhận sẵn  
sàng chấp nhận



# Cấu trúc TCP segment



Options (if any)

Data (if any)

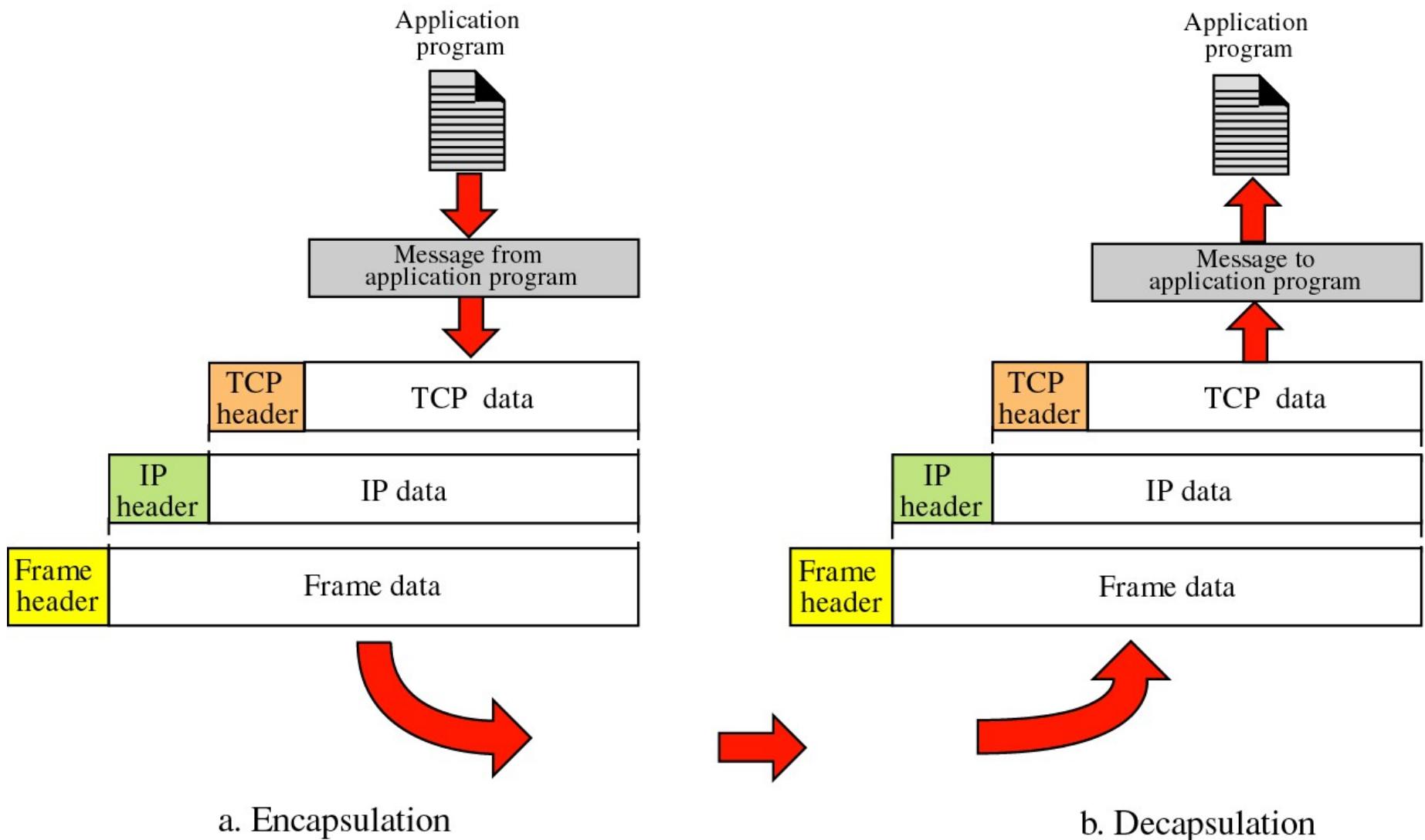


# Cấu trúc TCP segment

- **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



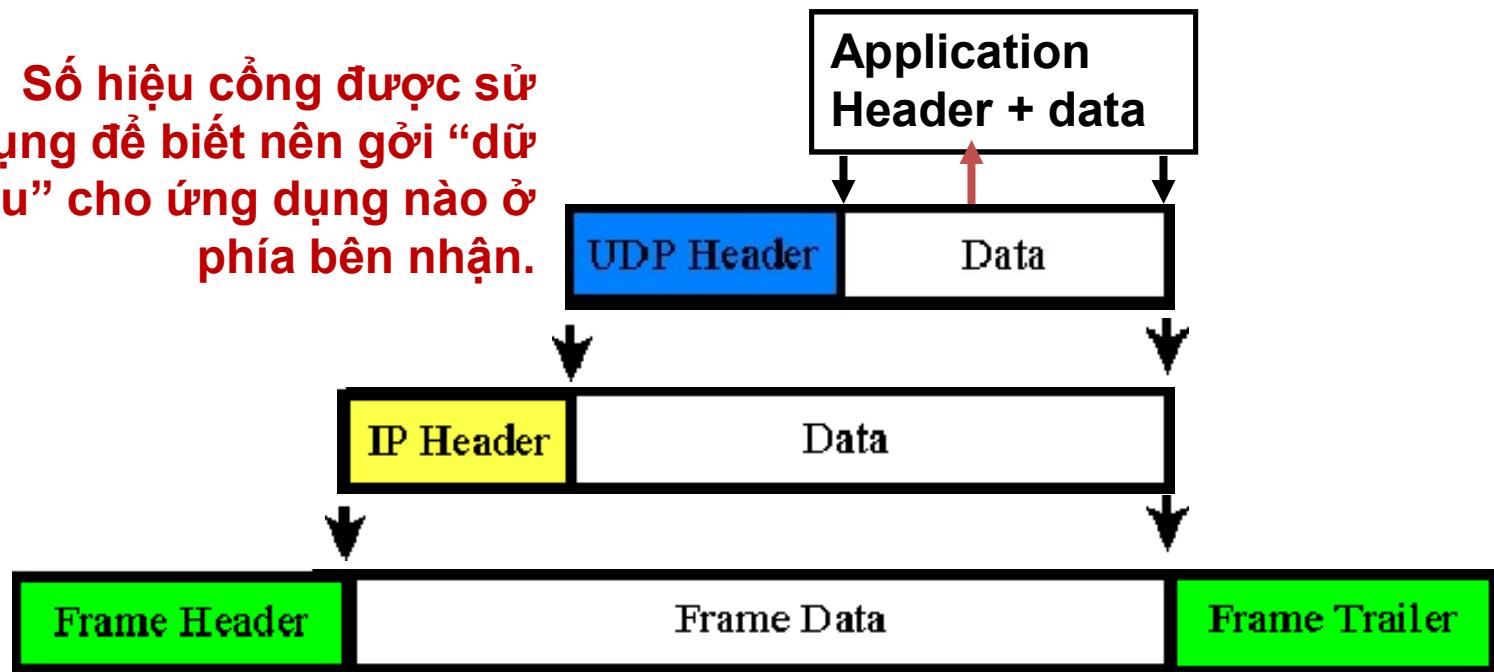
# Encapsulation và decapsulation





# Encapsulation và decapsulation

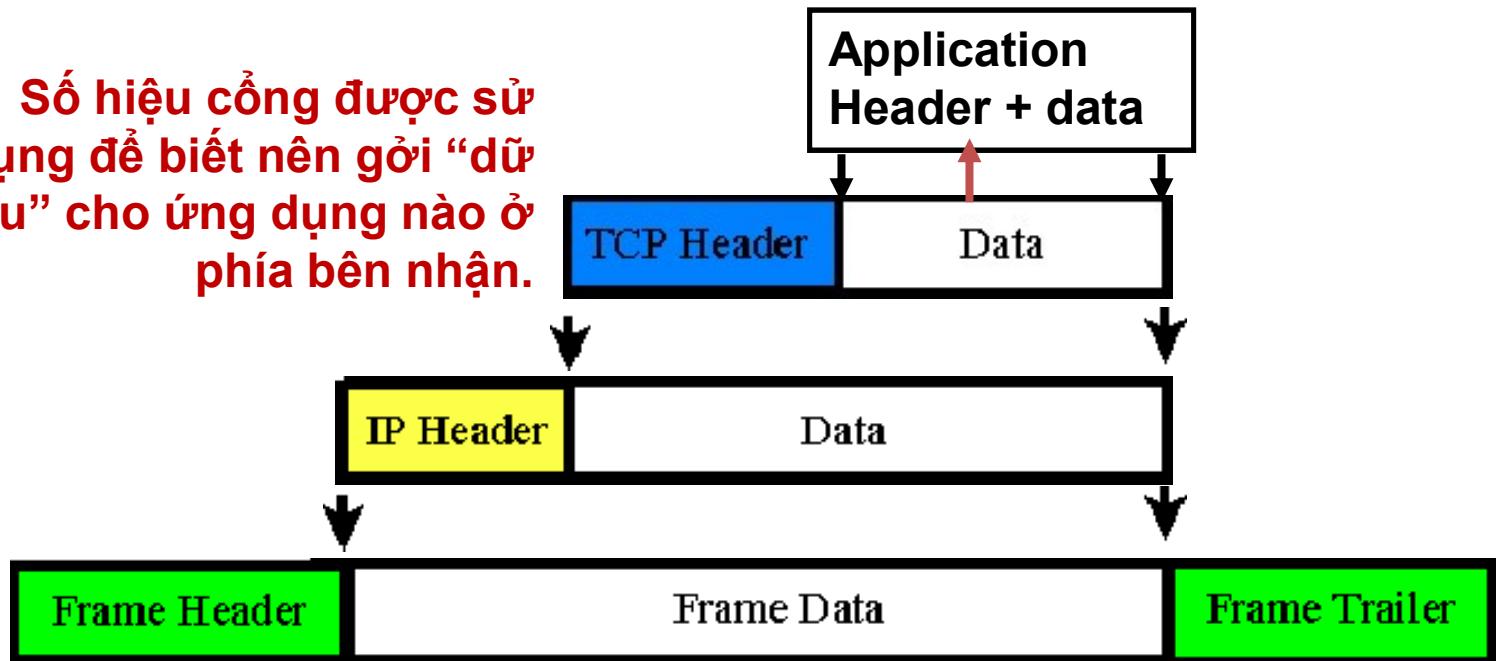
Số hiệu cổng được sử dụng để biết nên gửi “dữ liệu” cho ứng dụng nào ở phía bên nhận.





# Encapsulation và decapsulation

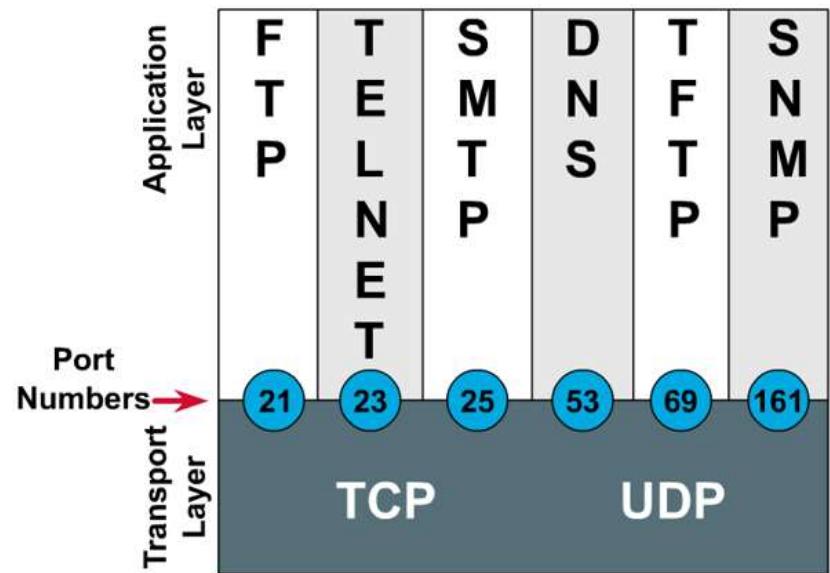
Số hiệu cổng được sử dụng để biết nên gởi “dữ liệu” cho ứng dụng nào ở phía bên nhận.





# Port number

- 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](#).
- Các message 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.





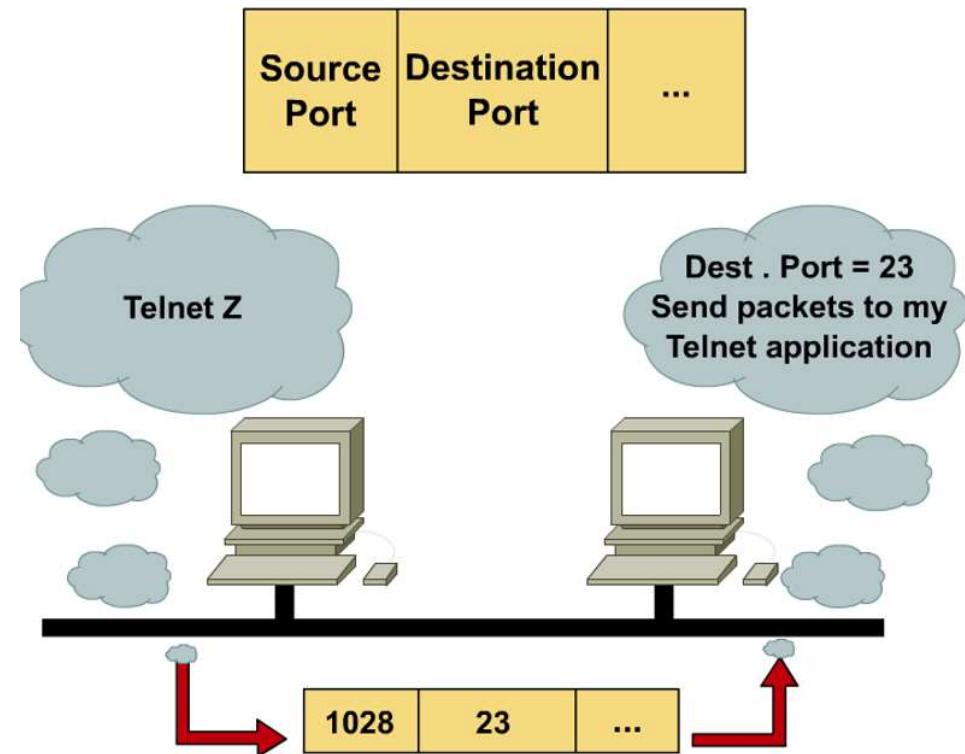
# Port number

- 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](http://www.iana.org/assignments/port-numbers) là 0-1023:  
<http://www.iana.org/assignments/port-numbers>
- Các cổng thông dụng 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](#)

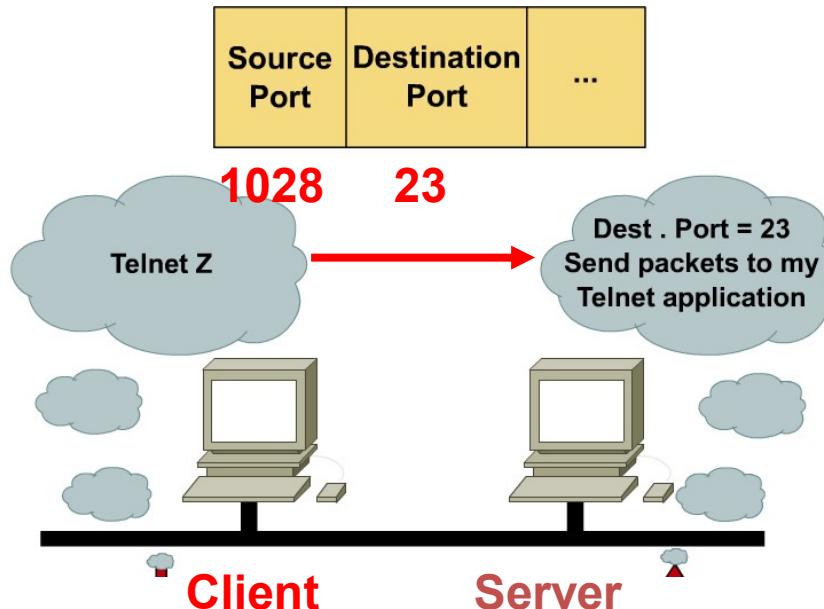


# TCP/UDP port number

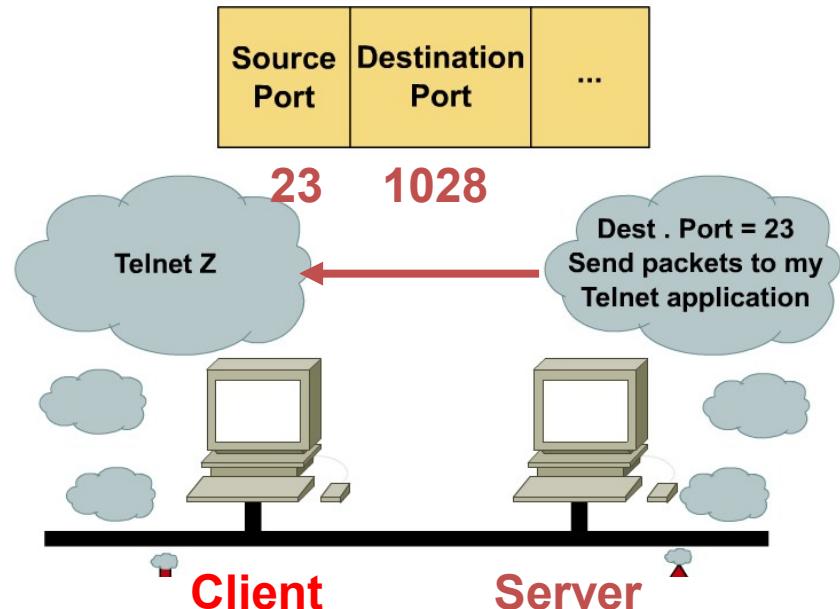
- Các hệ thống đầu cuối sử dụng các số hiệu cổng để chọn ứng dụng thích hợp.
- Các số hiệu cổng nguồn khởi nguồn, thường là các số lớn hơn 1023, được gán động bởi trạm nguồn.



## 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)**

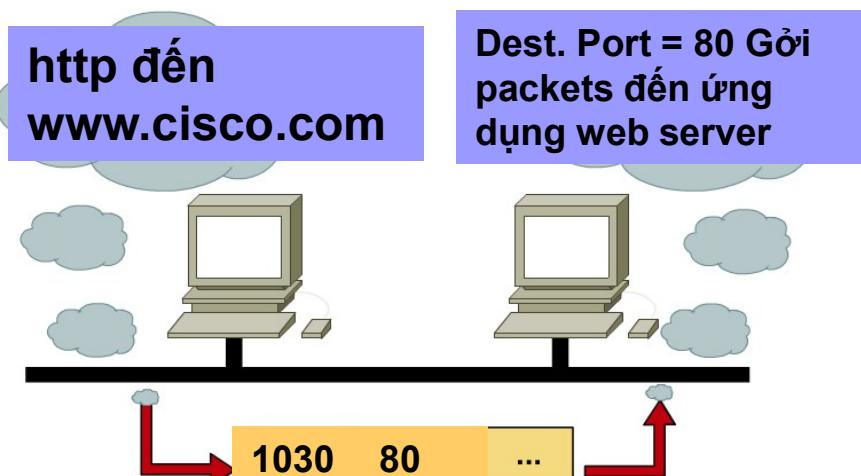
### **Server (đáp ứng dịch vụ Telnet):**

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

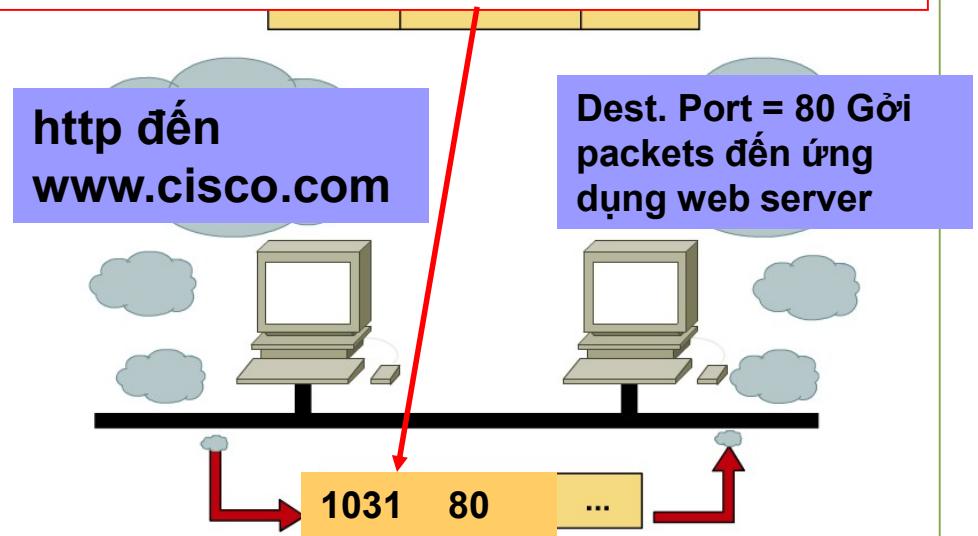
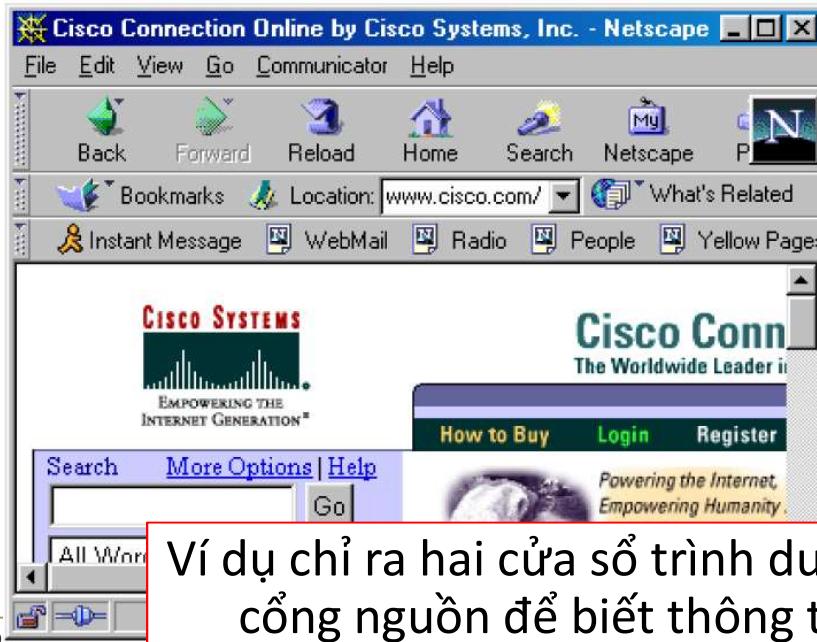
## TCP/UDP Port Numbers



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



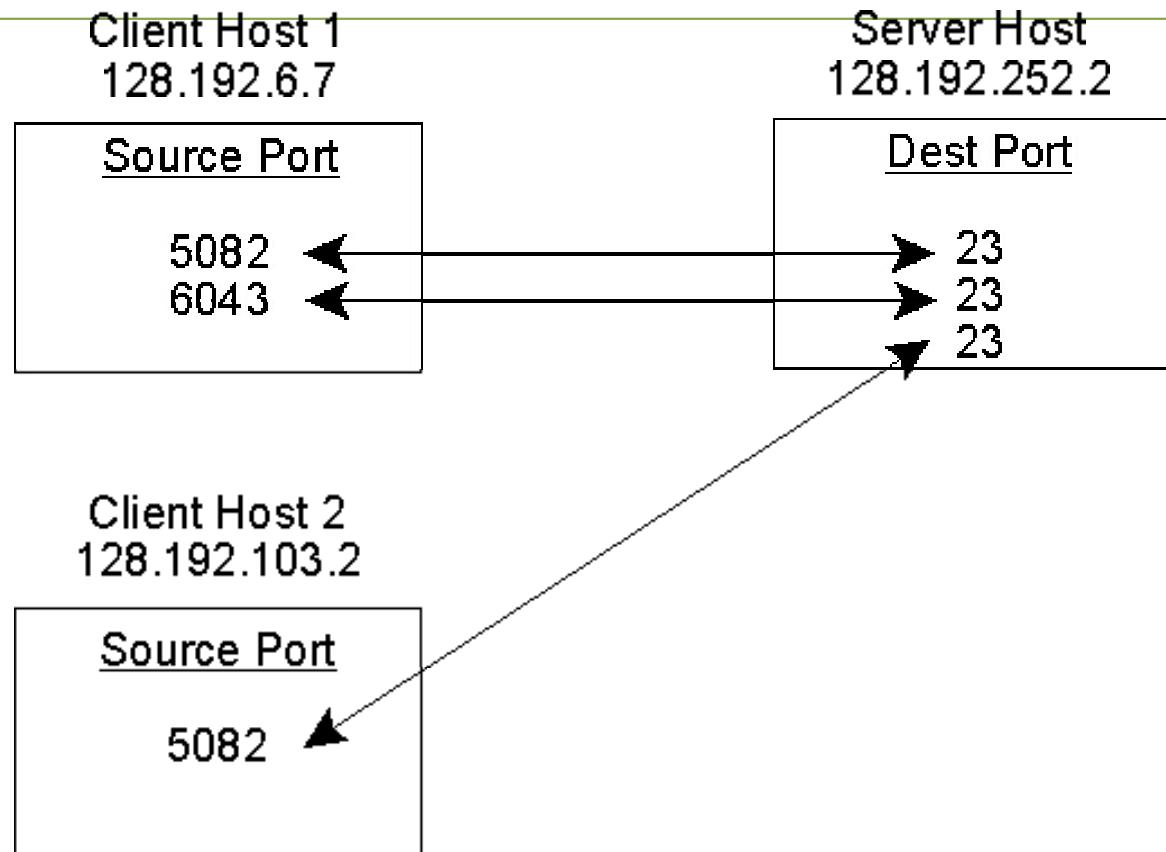
Netscape Navigator



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**

Proto	Local Address	Source IP	Source Port	Destination IP	Destination Port	Connection 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

- 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.



# Nội dung

- Tổng quan về tầng giao vận
- Dồn kênh/phân kênh (multiplexing và demultiplexing)
- Vận chuyển phi kết nối: User Datagram Protocol (UDP)
- Vận chuyển hướng kết nối: TCP
  - Cấu trúc đoạn tin TCP
  - Quản lý kết nối
  - Kiểm soát luồng
  - Kiểm soát tắc nghẽn
- Kiểm soát tắc nghẽn của TCP



# TCP cung cấp dịch vụ tin cậy như thế nào?

- Kiểm soát lỗi dữ liệu: **checksum**
- Kiểm soát mất gói tin: **phát lại** khi có time-out
- Kiểm soát dữ liệu đã được nhận chưa:
  - Seq. #
  - Ack

} Cơ chế báo nhận
- Chu trình làm việc của TCP:
  - Thiết lập liên kết: **bắt tay 3 bước**
  - Truyền/nhận dữ liệu: có thể thực hiện đồng thời(duplex) trên liên kết
  - Đóng liên kết



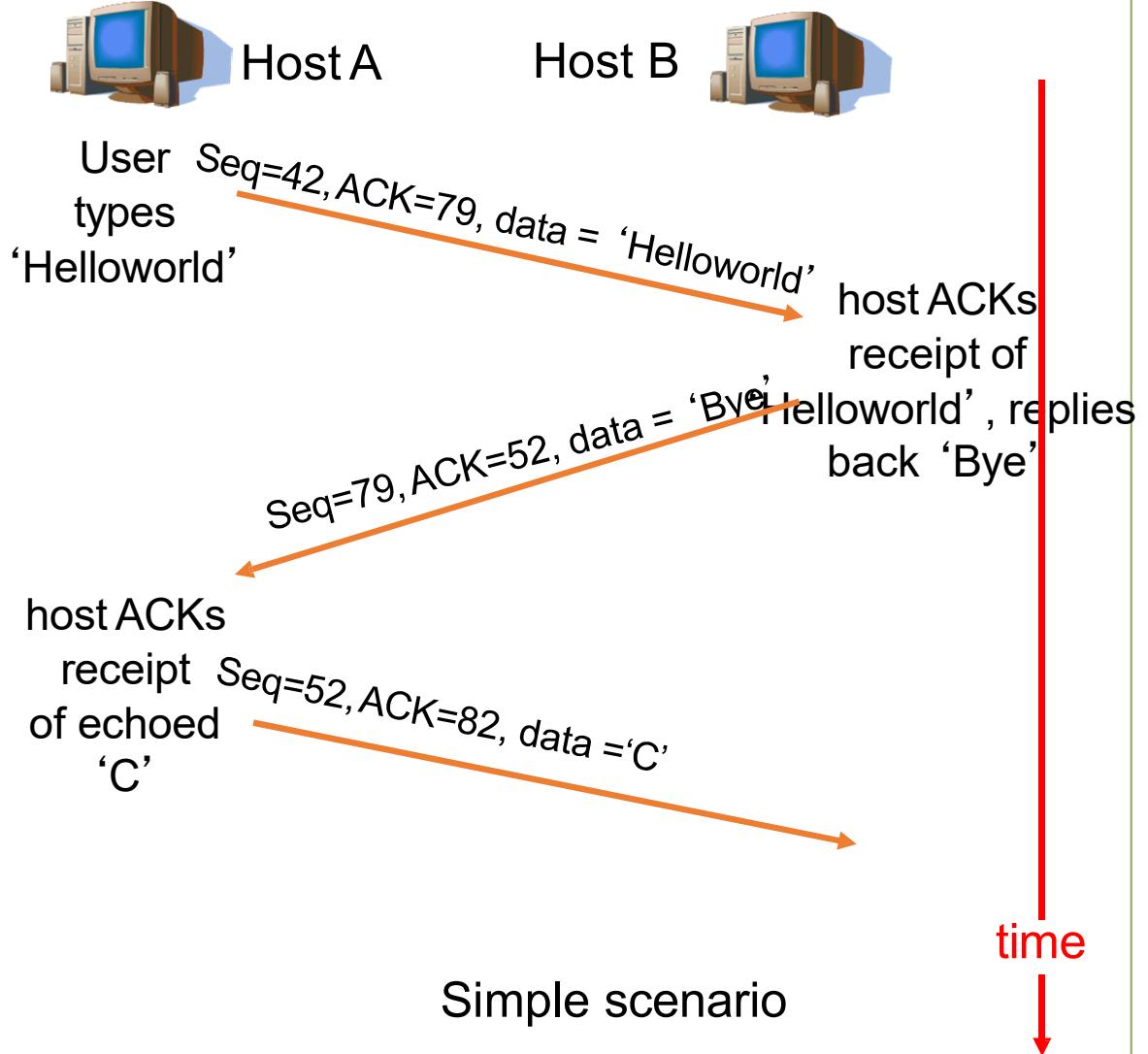
# Bắt tay ba bước (Three-way handshaking)

## ■ Seq. #:

- Số hiệu của byte đầu tiên của gói tin trong dòng dữ liệu

## ■ ACK:

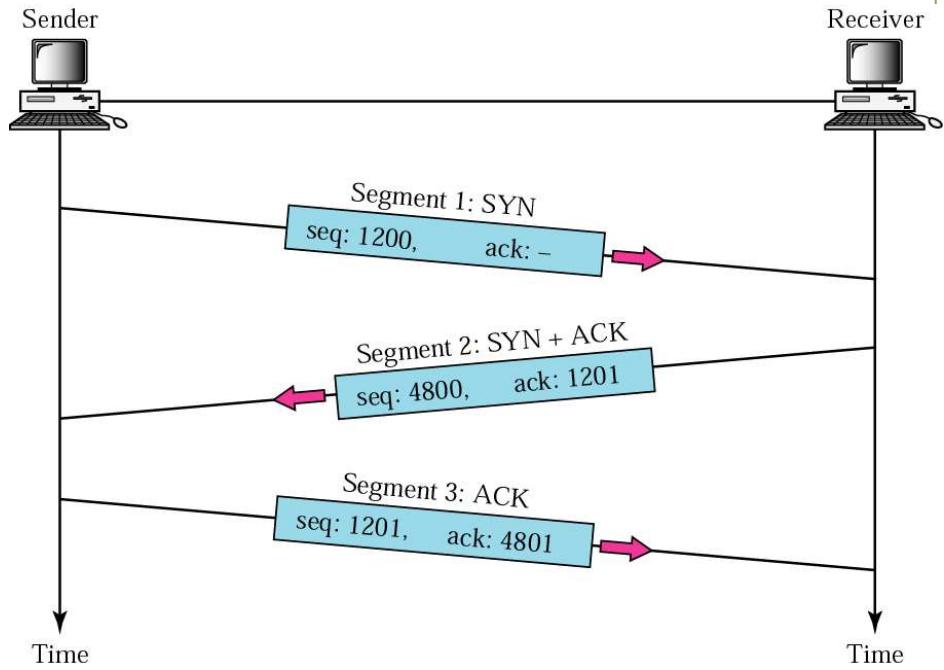
- Số hiệu byte mong muốn nhận từ đối tác
- Ngầm xác nhận đã nhận tốt các byte trước đó.





# Thiết lập liên kết TCP: Giao thức bắt tay 3 bước

- B1: Sender gửi **SYN** cho Receiver
  - chỉ ra giá trị khởi tạo **seq #** của Sender
  - không có dữ liệu
- B2: Receiver nhận SYN, trả lời bằng **SYN/ACK**
  - Receiver khởi tạo vùng đệm
  - chỉ ra giá trị khởi tạo **seq. #** của Receiver
- B3: Sender nhận SYN/ACK, trả lời ACK, có thể kèm theo dữ liệu

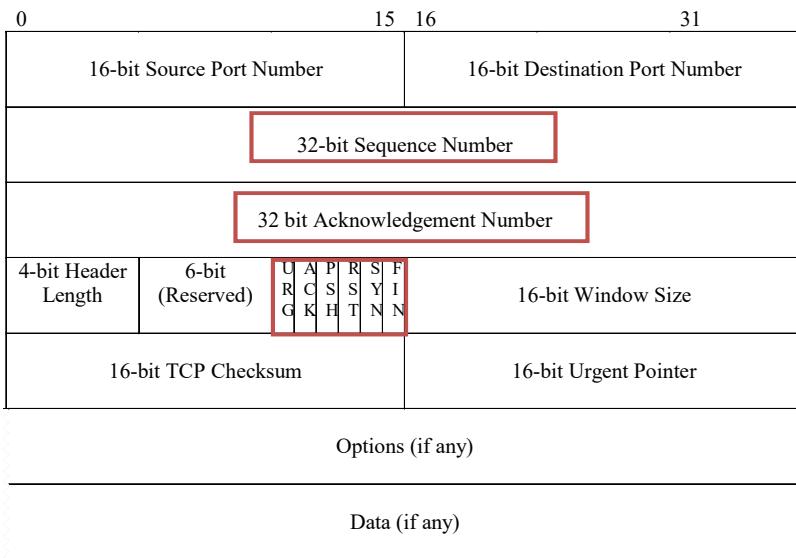


Bên gửi và nhận (sử dụng TCP) thiết lập “kết nối” trước khi trao đổi các segments dữ liệu

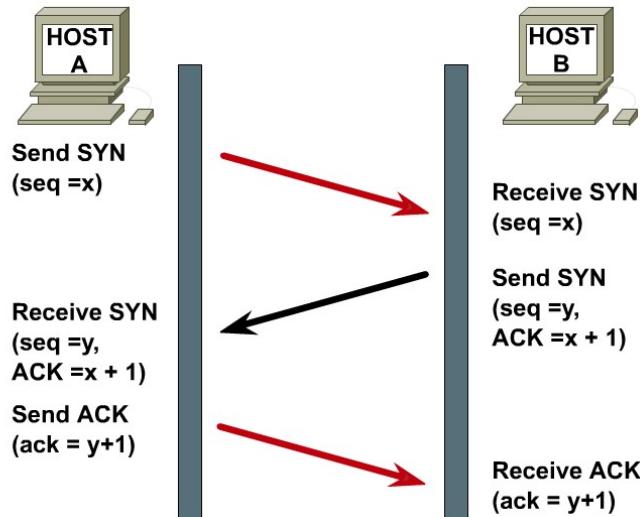
**Khởi tạo các tham số cho TCP:**

- số chuỗi (sequence number)
- thông tin về bộ đếm, kiểm soát luồng (cửa sổ nhận)

## TCP Header

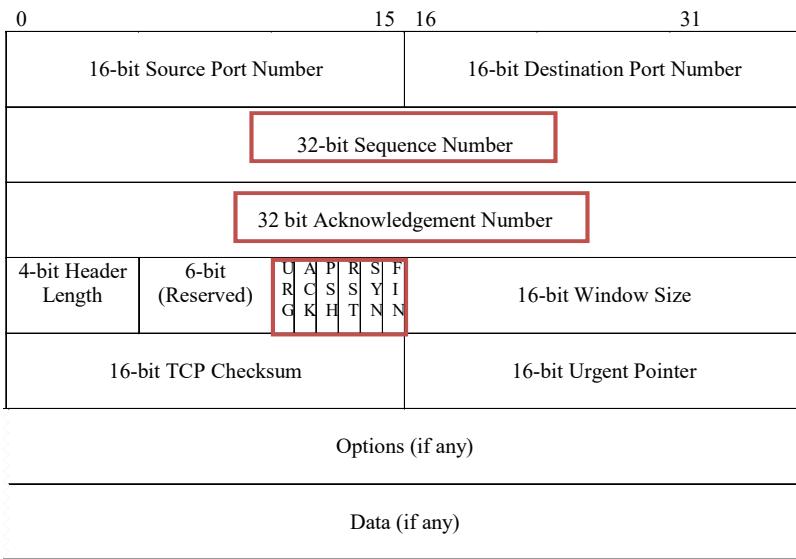


## TCP Three-Way Handshake/ Open Connection

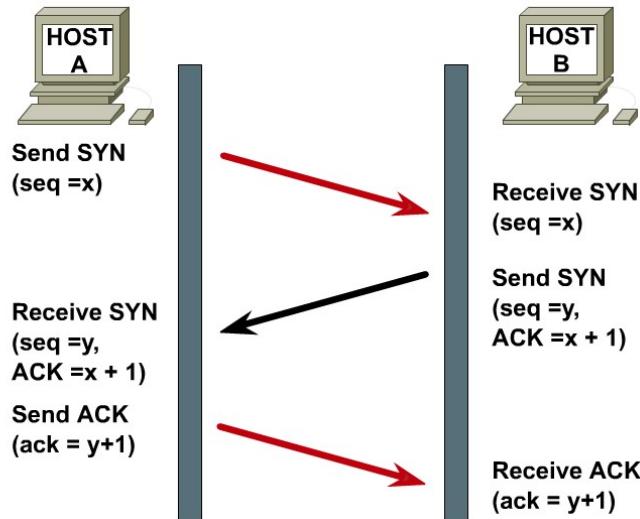


- Để một kết nối được thiết lập, hai trạm cuối phải đồng bộ với nhau về số **init sequence number** (ISNs) (chuỗi khởi tạo) của TCP.
- ISN** được sử dụng để theo dõi thứ tự của các gói và để đảm bảo rằng không có gói nào bị mất trong quá trình truyền.
- ISN** là con số bắt đầu khi một kết nối TCP được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

## TCP Header

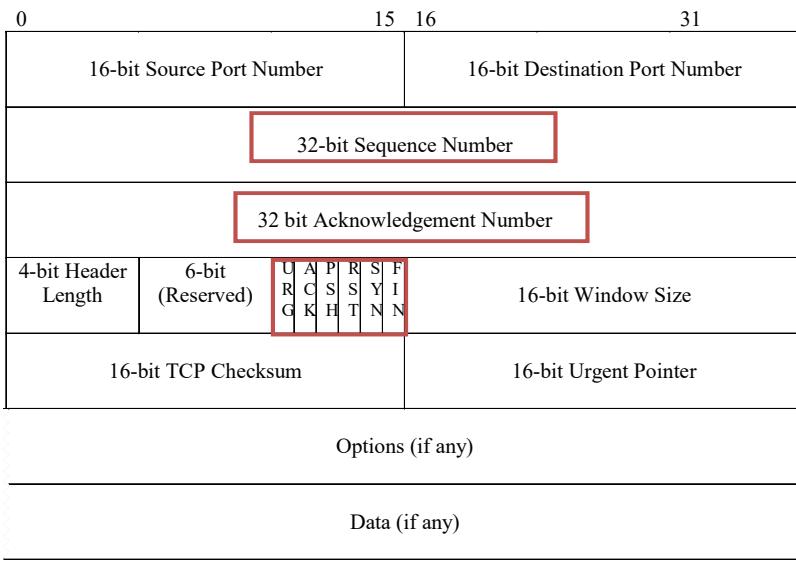


## TCP Three-Way Handshake/ Open Connection

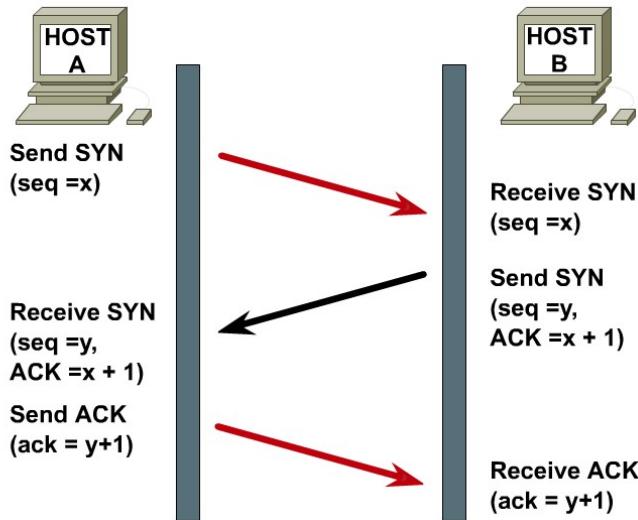


- Sự đồng bộ được thiết lập bằng cách trao đổi các **segments** có mang các **initial sequence numbers (ISNs)** (**chuỗi khởi tạo**) khởi tạo và bit điều khiển **SYN** được bật, có nghĩa là synchronize (đồng bộ). (Các segments mang bit SYN còn được gọi là **SYNs**.)
- Kết nối thành công đòi hỏi một cơ chế phù hợp để chọn một **ISN** và một cách liên quan đến việc “**bắt tay**” để trao đổi **ISNs**.
- Sự đồng bộ yêu cầu rằng mỗi bên gửi **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



## TCP Three-Way Handshake/ Open Connection

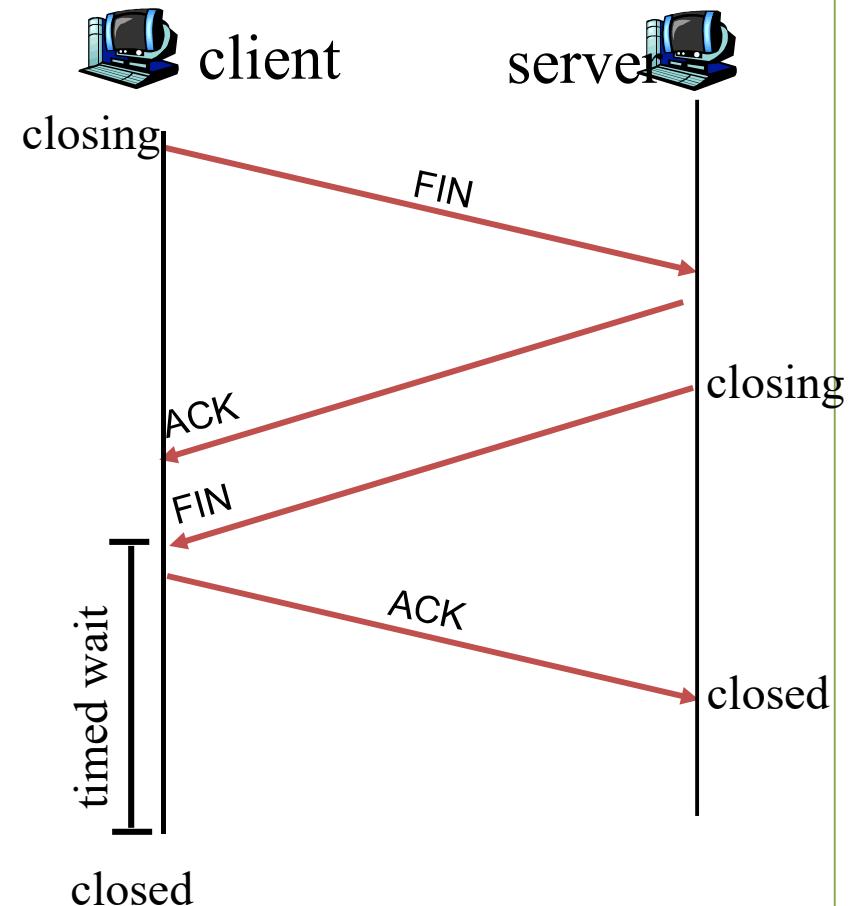


- Người nhận gói SYN đầu tiên không có cách nào để biết được nó có phải là một segment cũ bị trễ hay không trừ khi nó nhớ được số chuỗi cuối cùng được sử dụng cho kết nối đó, điều này không phải lúc nào cũng có thể thực hiện được, và nó phải yêu cầu người gửi xác minh lại gói SYN này
- Tại thời điểm này, cả hai bên đều có thể bắt đầu truyền thông, và cả hai đều có thể ngắt mối liên kết truyền thông này vì TCP là phương thức truyền thông đồng đẳng (cân bằng).



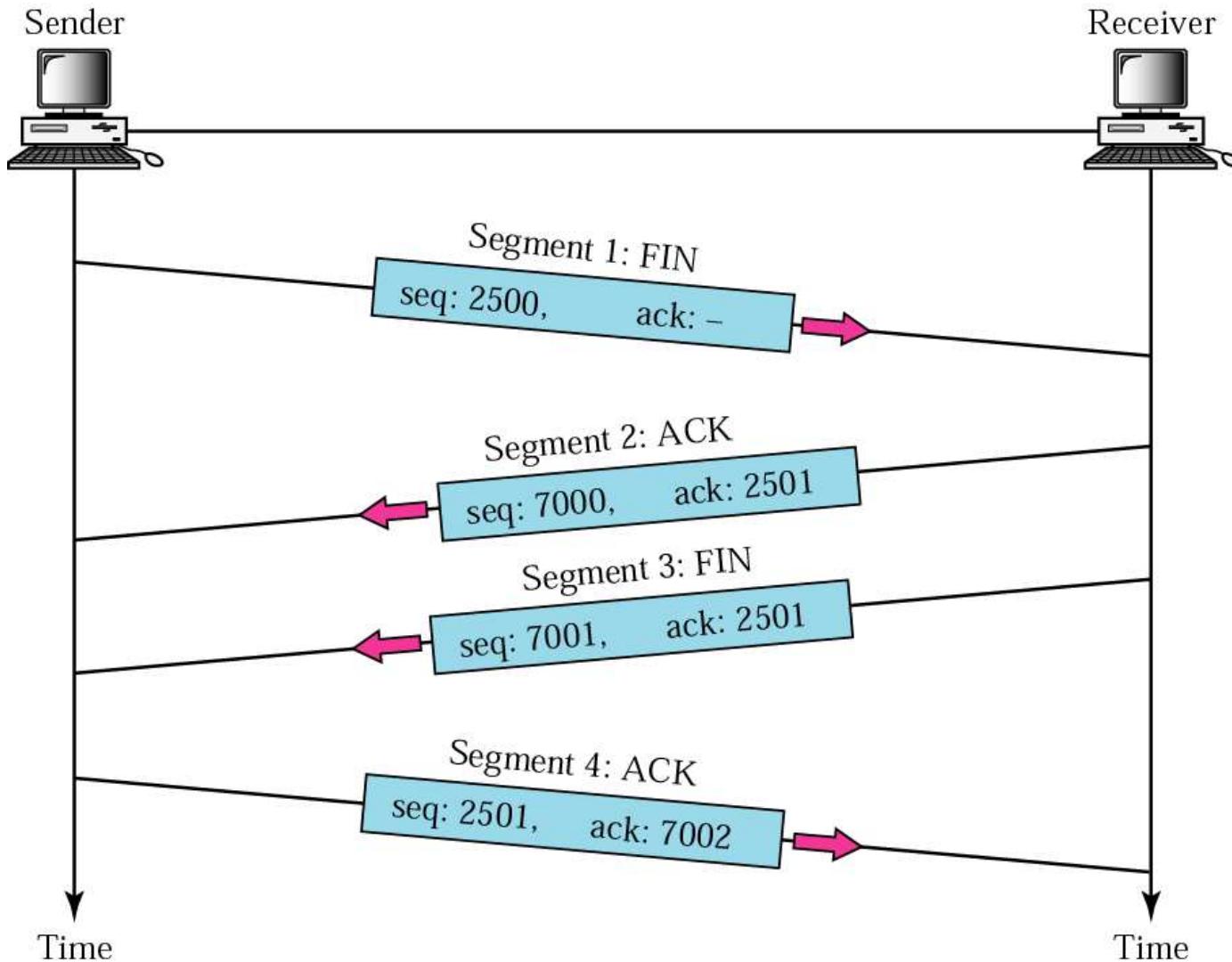
# Đóng kết nối TCP

- **B1:** client gửi segment điều khiển TCP FIN đến server
- **B2:** server nhận FIN, trả lời với ACK. Đóng kết nối, gửi FIN.
- **B3:** client nhận FIN, trả lời với ACK.
  - Vào trạng thái “chờ đợi một thời gian” – sẽ trả lời với ACK cho FINs nhận được
- **B4:** server, nhận ACK. Kết nối đã được đóng.



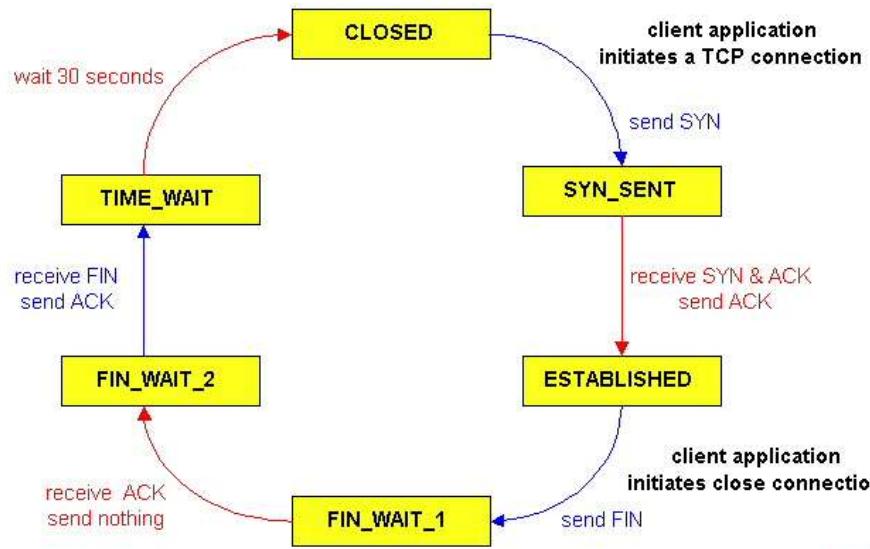


# Bắt tay bốn bước



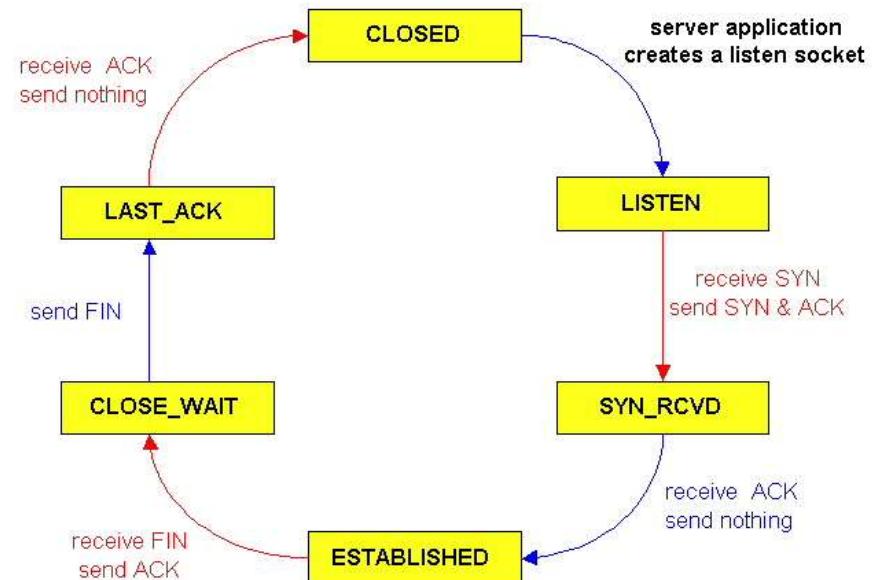


# Quản lý kết nối của TCP



TCP client  
lifecycle

TCP server  
lifecycle





# Nội dung

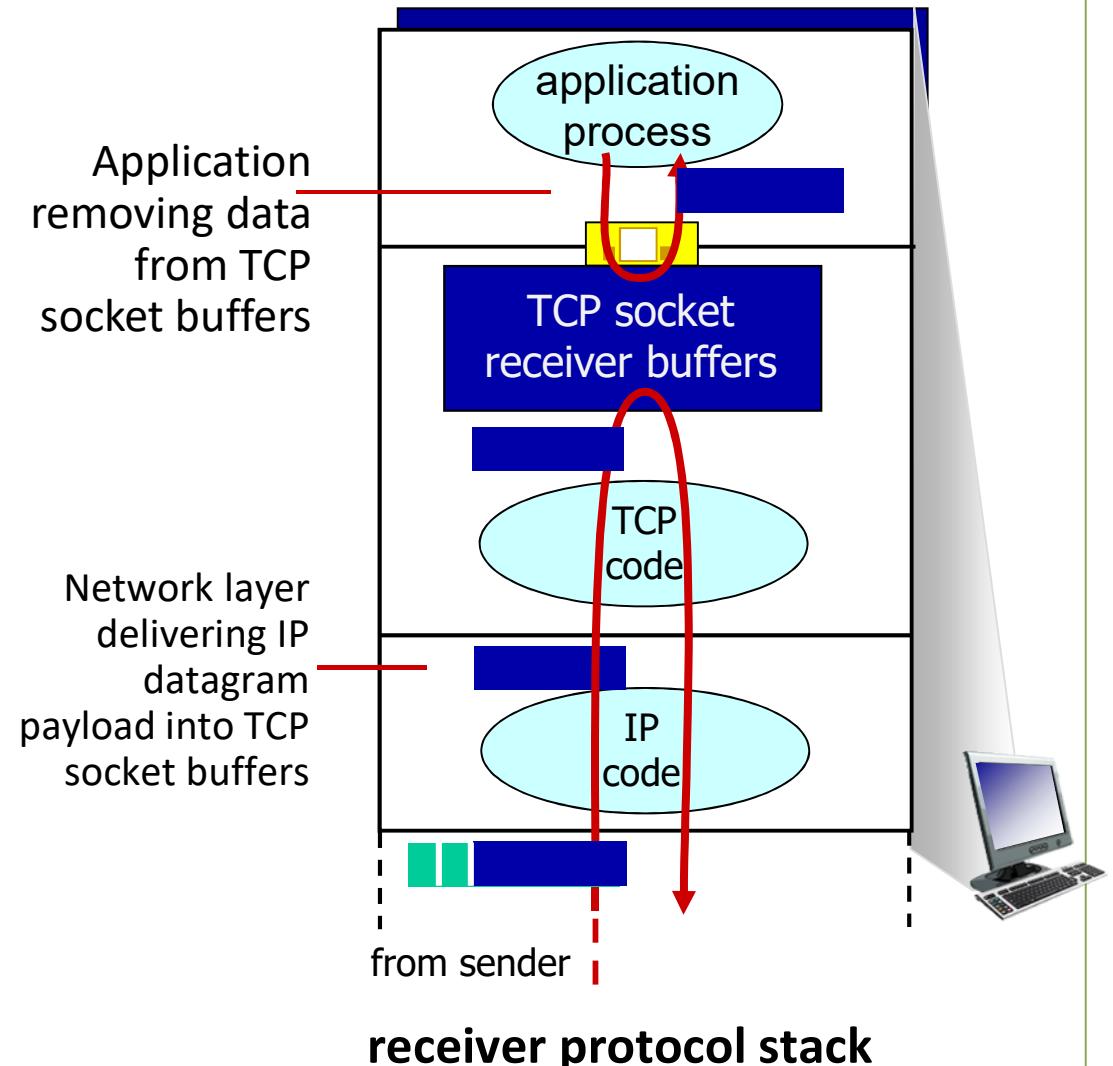
- Tổng quan về tầng giao vận
- Dồn kênh/phân kênh (Multiplexing và demultiplexing)
- Vận chuyển phi kết nối: User Datagram Protocol (UDP)
- Vận chuyển hướng kết nối: TCP
  - Cấu trúc đoạn tin TCP
  - Quản lý kết nối
    - **Kiểm soát luồng**
    - Kiểm soát tắc nghẽn
  - **Kiểm soát tắc nghẽn** của TCP

# Kiểm soát luồng (flow control)

- Q: Chuyện gì xảy ra nếu tầng network phía gửi, gửi data nhanh hơn tầng application (bên nhận) sử dụng data từ bộ đệm của socket?

## flow control

receiver kiểm soát việc gửi data của sender, vì thế sender sẽ không overflow receiver's buffer qua việc gửi quá nhiều và nhanh

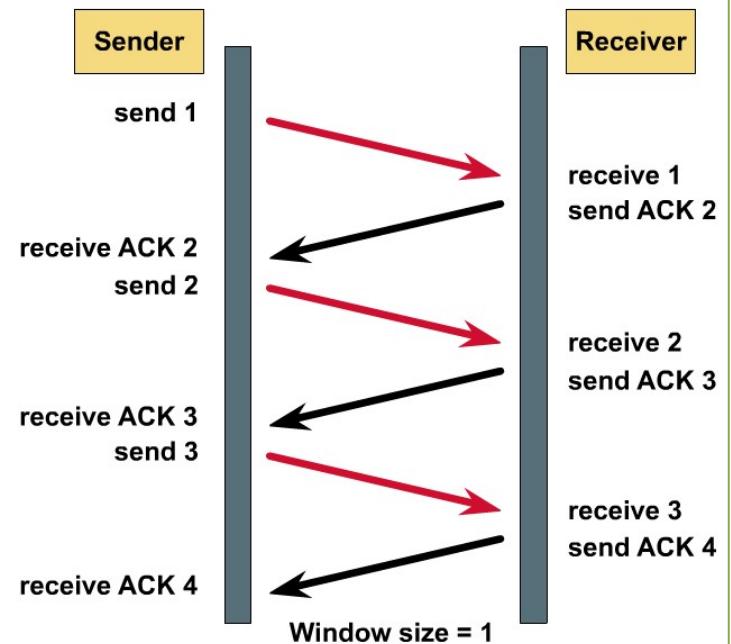




# Kiểm soát luồng trong TCP

- Điều khiển lượng dữ liệu được gửi đi
  - Bảo đảm rằng hiệu quả là tốt
  - Không làm quá tải các bên
- Sử dụng cơ chế cửa sổ trượt.
- Các bên sẽ có cửa sổ kiểm soát
  - **Rwnd**: Cửa sổ nhận
  - **CWnd**: Cửa sổ kiểm soát tắc nghẽn
- Lượng dữ liệu gửi đi phải nhỏ hơn  $\min(\text{Rwnd}, \text{Cwnd})$
- Sau khi một trạm truyền một số bytes bằng của kích cỡ cửa sổ, nó phải nhận được một hồi báo (**ACK**) trước khi gửi tiếp.
- Kích cỡ cửa sổ xác định bao nhiêu dữ liệu mà trạm nhận có thể tiếp nhận tại một thời điểm.

## TCP Simple Acknowledgment



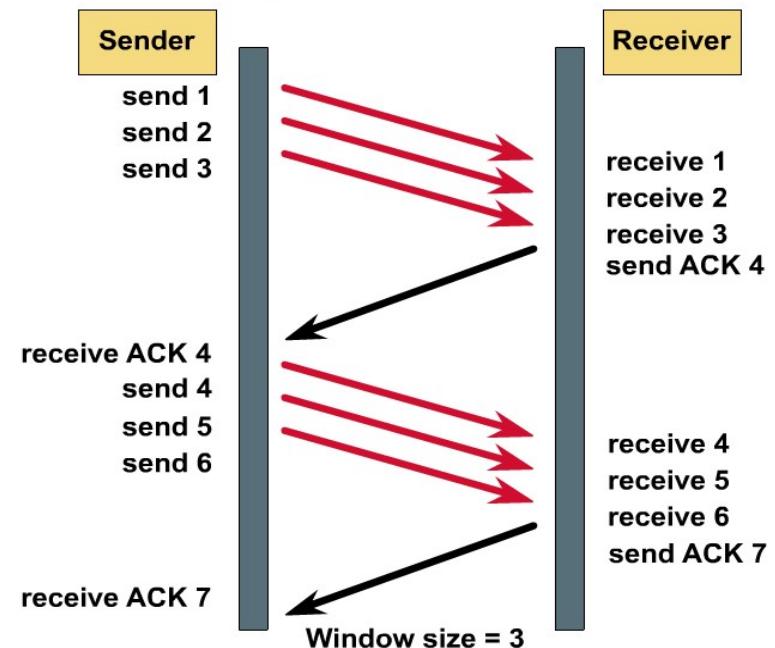
Với kích thước cửa sổ bằng 1, mỗi segment chỉ mang một byte dữ liệu và phải được hồi báo trước khi segment khác được truyền → ko hiệu quả.



# 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*).

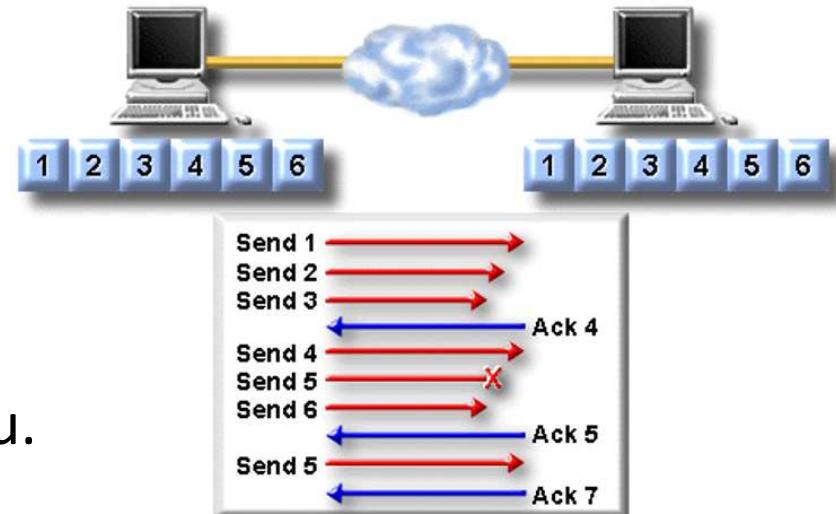
## TCP Sliding Window





# Kích thước cửa sổ TCP (TCP Window Size)

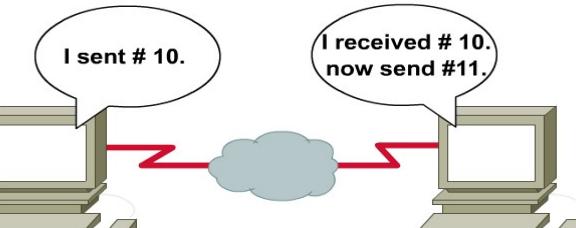
- Packets có thể bị mất dọc đường, hết thời gian, hay bị sai lệch.
- TCP cung cấp các hồi báo lũy tích (**cumulative acknowledgments**).
- Hiện tại không có cách để hồi báo các mảnh được chọn lọc của dòng dữ liệu (data stream).
- Nếu octets **4, 5**, và **6** đã được gửi, nhưng **5** bị mất, bên nhận 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 đầu.
- 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

Source Port	Destination Port	Sequence Number	Acknowledgment Numbers	...
-------------	------------------	-----------------	------------------------	-----



Source Des. Seq. Ack.	1028	23	10	1	...
Source Des. Seq. Ack.	1028	23	1	11	...

#### 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.



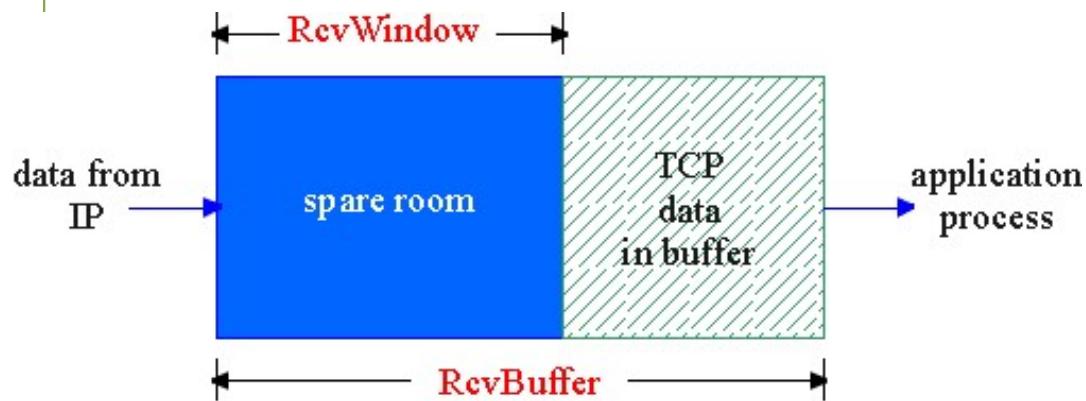
# Cơ chế cửa sổ trượt (Sliding window)

- Các khung dữ liệu gửi đi được đánh số. Số thứ tự phải lớn hơn hoặc bằng kích thước cửa sổ
- Các khung dữ liệu được báo nhận (ACK) bằng thông báo có đánh số
- Được báo gộp. Nếu 1,2,3,4 được nhận thành công, chỉ gửi báo nhận 4
- Khi đã nhận được thông báo nhận được khung **k**, có nghĩa là tất cả các khung **k-1, k-2.., k-1** đã nhận được



# 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

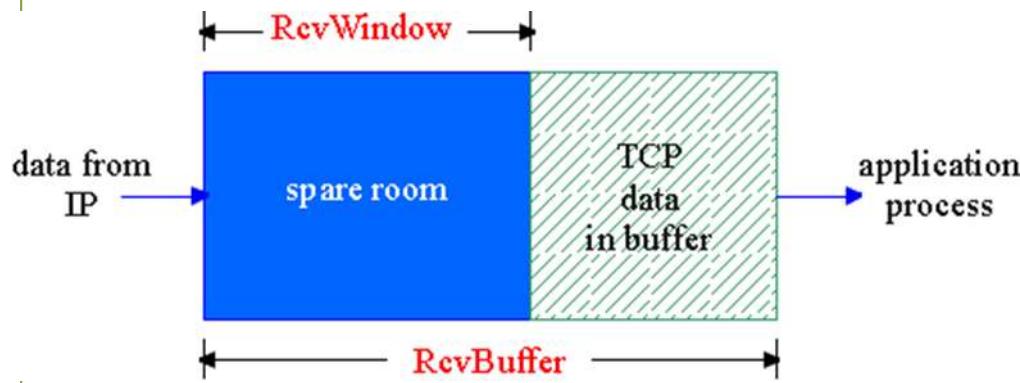
## Kiểm soát luồng

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 (Sliding window)

- Bên gởi tính window size có thể sử dụng được, nó cho biết bao nhiêu byte dữ liệu có thể được gởi ngay.
- Theo thời gian, cửa sổ trượt này di chuyển sang bên phải, khi bên nhận hồi báo (ACK) cho dữ liệu đã nhận đúng.
- Bên nhận gởi 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 gởi và được hồi báo.
  2. Gờ phải mở (di chuyển sang phải) cho phép nhiều dữ liệu hơn được gởi. 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 gởi, nhưng chưa được hồi báo.

Kích thước cửa sổ khởi tạo

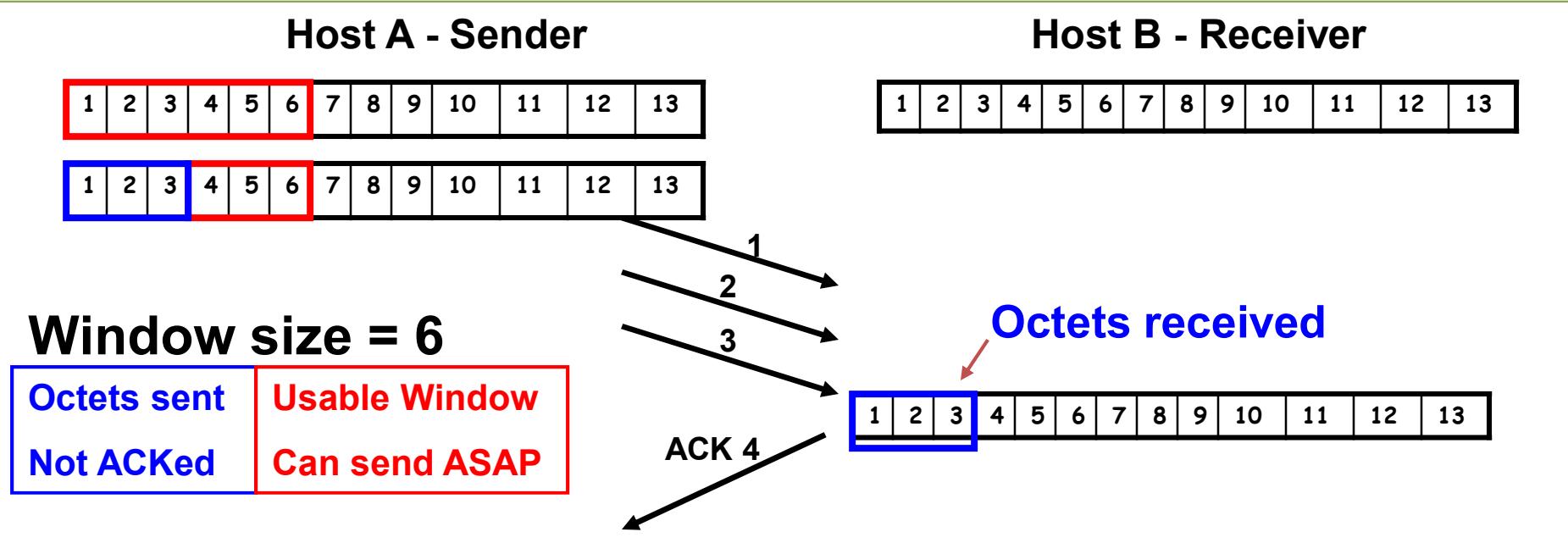
Cửa sổ có thể sử dụng

Có thể gởi ngay

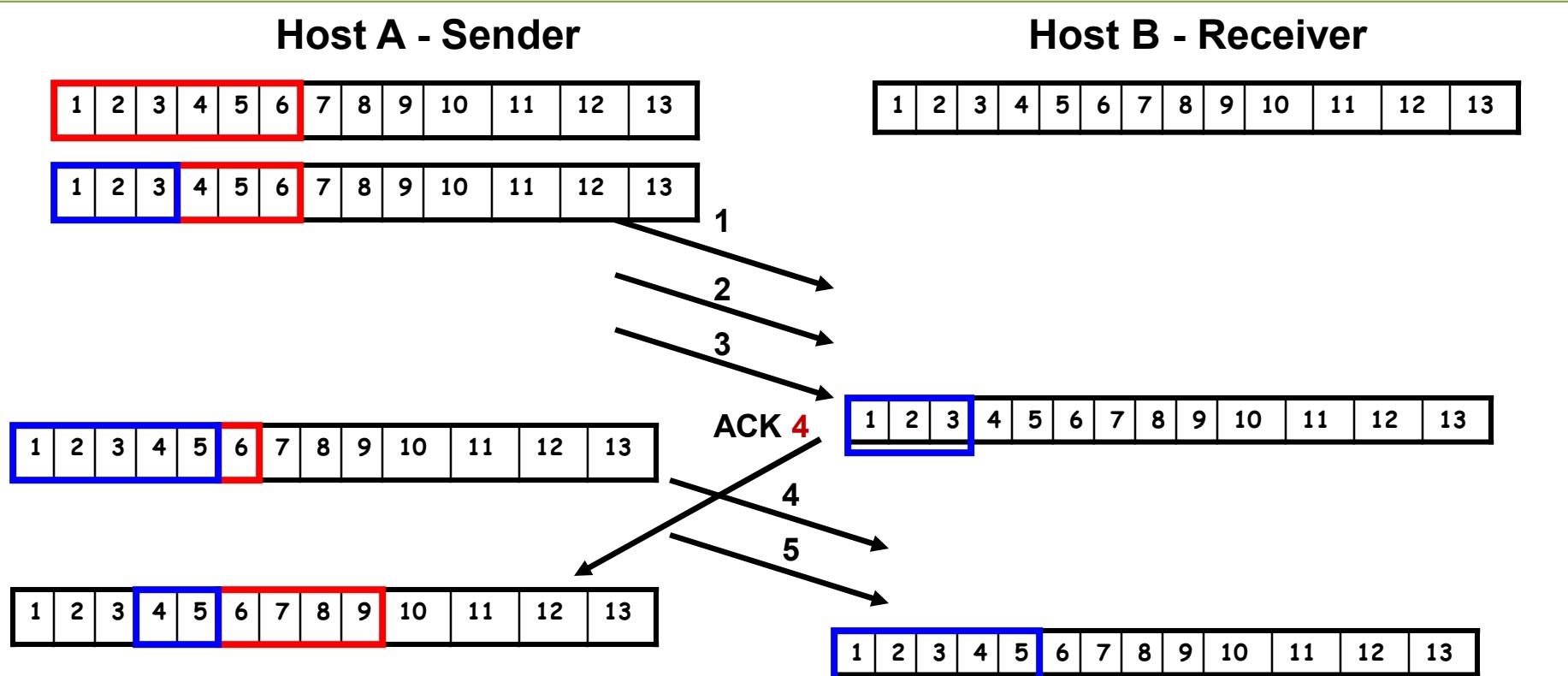
Kích thước cửa sổ hoạt động

Đã gởi  
chưa ACK

Có thể sử dụng  
Có thể gởi ngay



- Host B gửi 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 Host B gửi một ACK có là 4 đến Host A (Host B mong nhận tiếp từ octet **4**).



**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.



# Tham khảo

- Jim Kurose, Keith Ross, Pearson, 2020, Computer Networking: A Top-Down Approach 8th edition
- Mạng Máy Tính-Phan Văn Nam- ĐH Nha Trang
- Mạng Máy Tính-Trương Diệu Linh-Bách Khoa Hà Nội