



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Chương 5: Tầng giao vận

Đọc trước: Chapter 5-Computer Networks,
Tanenbaum

Tổng quan

- Các tuần trước : Giao thức IP
 - Địa chỉ, gói tin IP
 - ICMP
 - Chọn đường
- Hôm nay: Tầng giao vận
 - Nguyên lý tầng giao vận
 - Giao thức UDP
 - Giao thức TCP



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Các khái niệm cơ bản

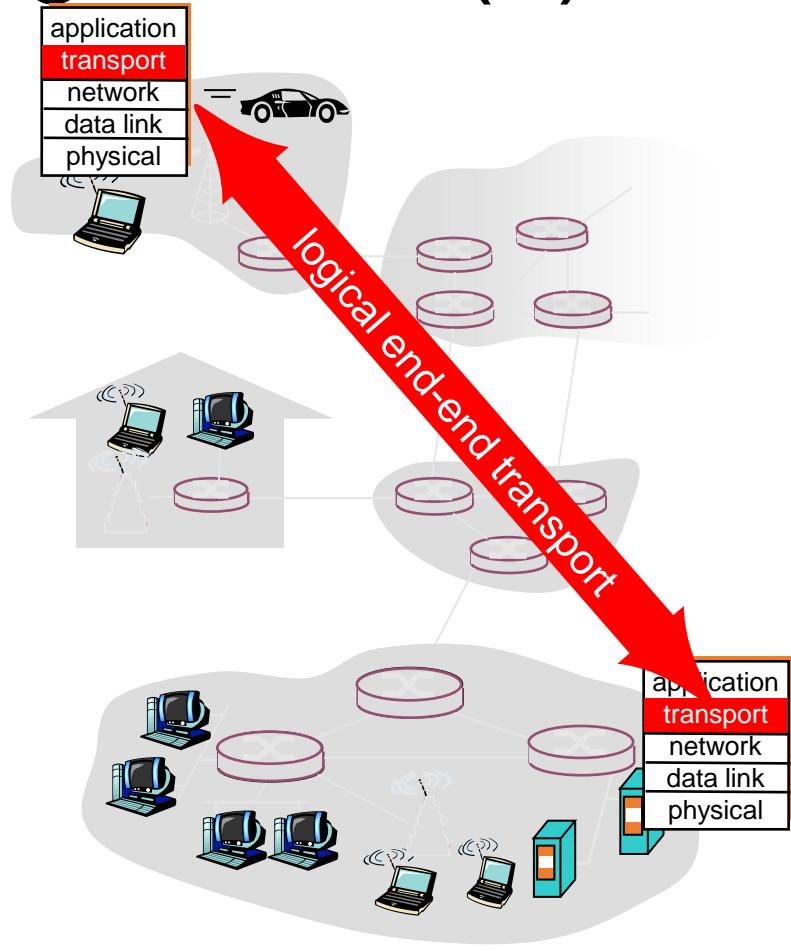
Vị trí trong kiến trúc phân tầng
Hướng liên kết vs. Không liên kết
UDP & TCP

Vị trí trong kiến trúc phân tầng

Application (HTTP, Mail, ...)	Hỗ trợ các ứng dụng trên mạng
Transport (UDP, TCP ...)	Truyền dữ liệu giữa các ứng dụng
Network (IP, ICMP ...)	Chọn đường và chuyển tiếp gói tin giữa các máy, các mạng
Datalink (Ethernet, ADSL...)	Hỗ trợ việc truyền thông cho các thành phần kế tiếp trên cùng 1 mạng
Physical (bits...)	Truyền và nhận dòng bit trên đường truyền vật lý

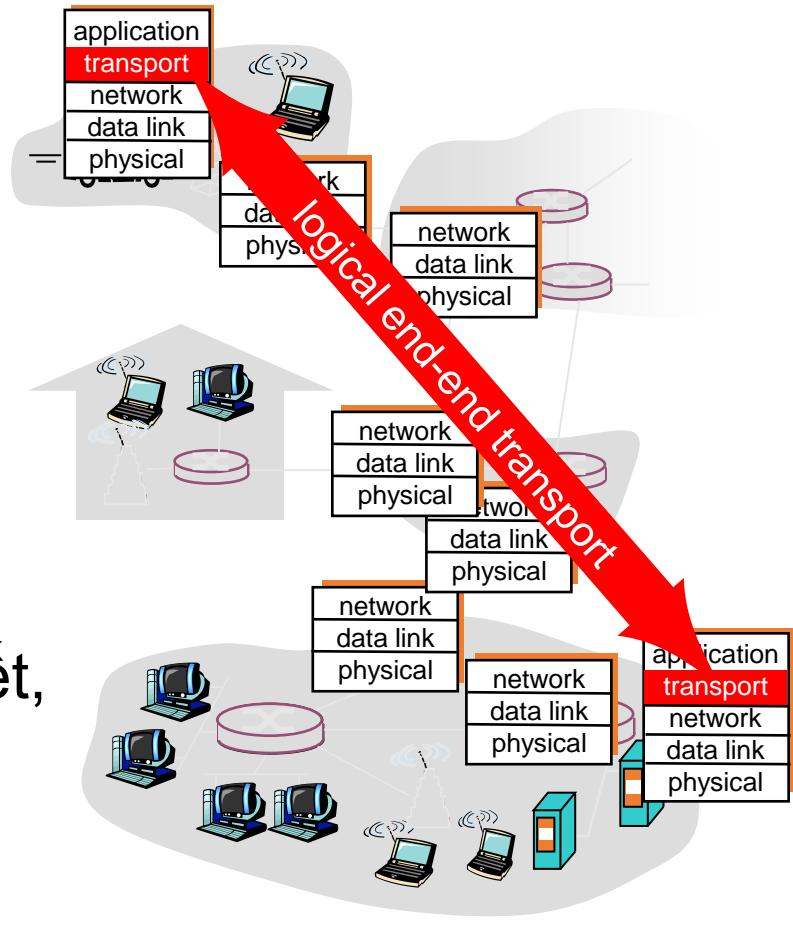
Tổng quan về tầng giao vận (1)

- Cung cấp phương tiện truyền giữa các ứng dụng cuối
 - Các ứng dụng là các tiến trình chạy trên các máy.
- Bên gửi:
 - Nhận dữ liệu từ ứng dụng
 - Đặt dữ liệu vào các đoạn tin và chuyển cho tầng mạng
 - Nếu dữ liệu quá lớn, nó sẽ được chia làm nhiều phần và đặt vào nhiều đoạn tin khác nhau
- Bên nhận:
 - Nhận các đoạn tin từ tầng mạng
 - Tập hợp dữ liệu và chuyển lên cho ứng dụng



Tổng quan về tầng giao vận (2)

- Được cài đặt trên các hệ thống cuối
 - Không cài đặt trên các routers, switches...
- Hai dạng dịch vụ giao vận
 - Tin cậy, hướng liên kết, e.g TCP
 - Không tin cậy, không liên kết, e.g. UDP



Tại sao lại cần 2 loại dịch vụ?

- Các yêu cầu đến từ tầng ứng dụng là đa dạng
- Các ứng dụng cần dịch vụ với 100% độ tin cậy như mail, web...
 - 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

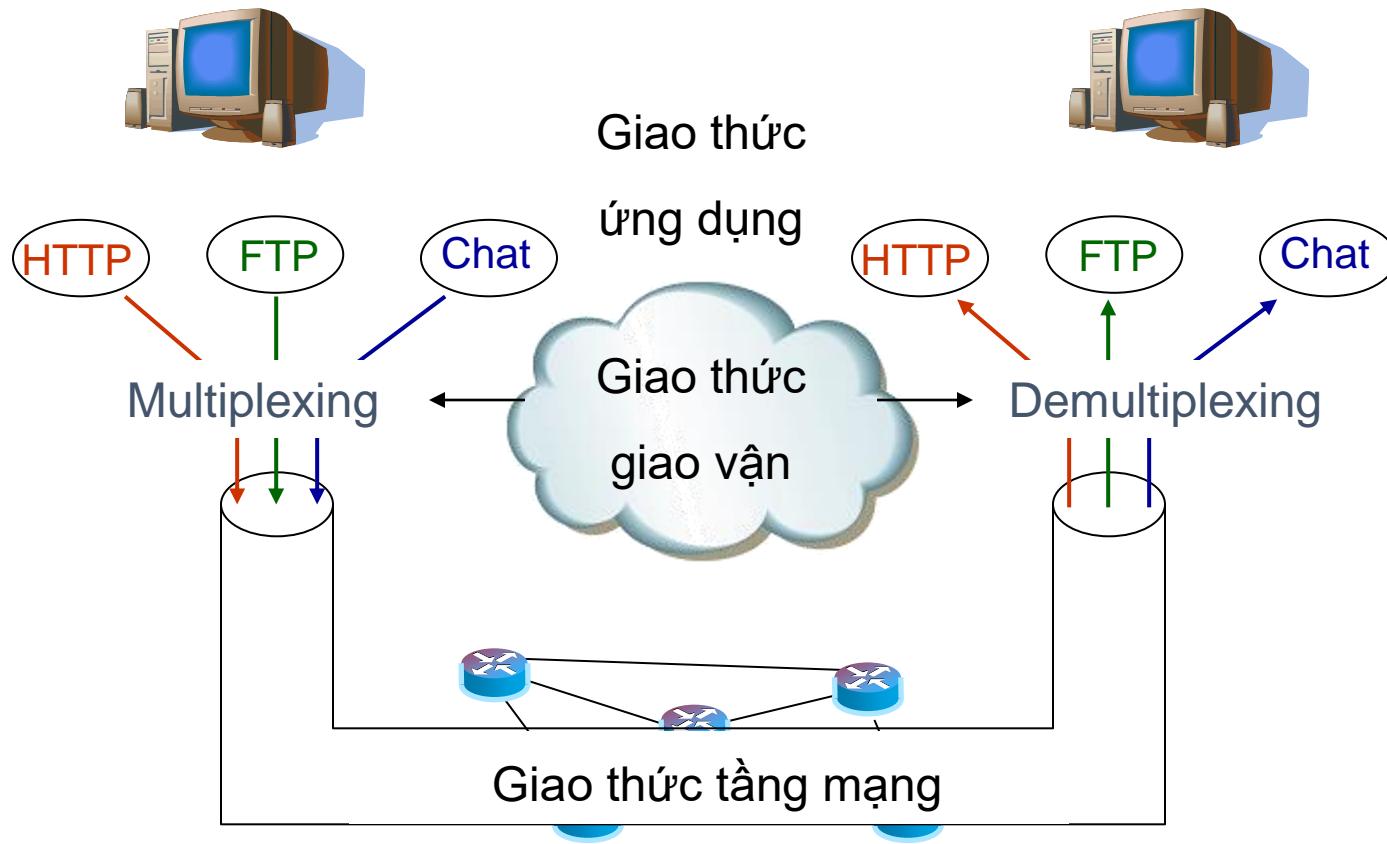


ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Các chức năng chung

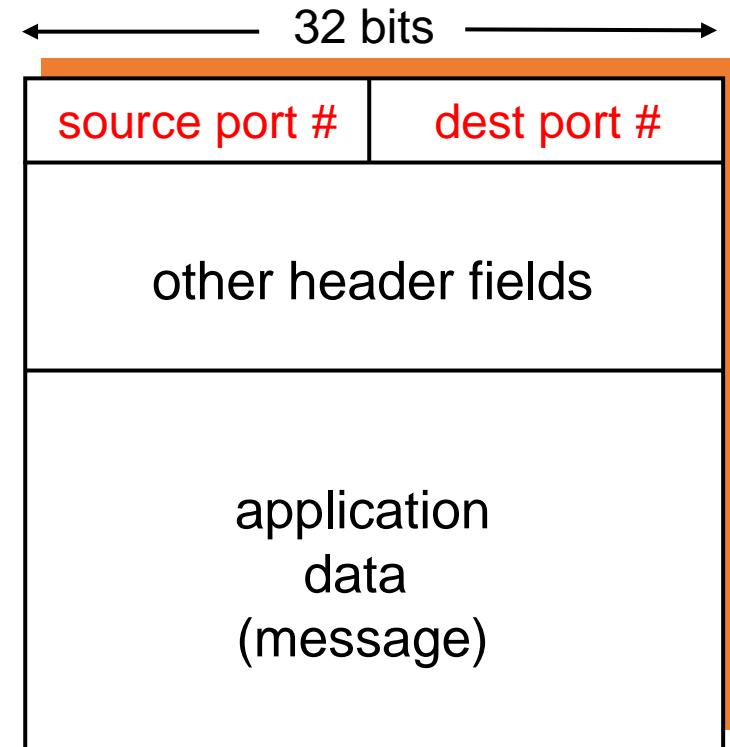
Dồn kênh/phân kênh
Kiểm soát lỗi

Dồn kênh/phân kênh - Mux/Demux



Mux/Demux hoạt động ntn?

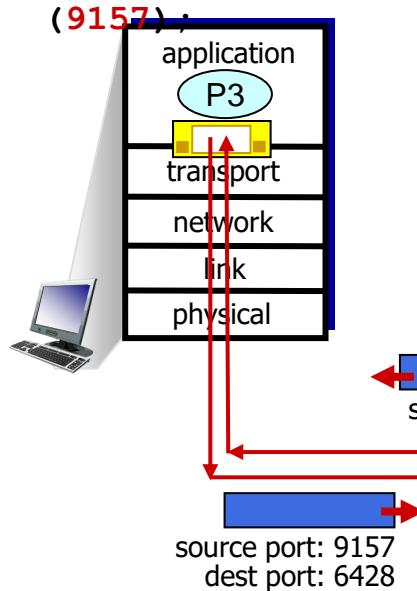
- Tại tầng mạng, gói tin IP được định danh bởi địa chỉ IP
 - Để xác định máy trạm
- Làm thế nào để phân biệt các ứng dụng trên cùng một máy?
 - Sử dụng số hiệu cổng (16 bits)
 - Mỗi trình duyệt ứng dụng được gán 1 cổng
- **Socket:** Một cặp địa chỉ IP và số hiệu cổng



TCP/UDP segment format

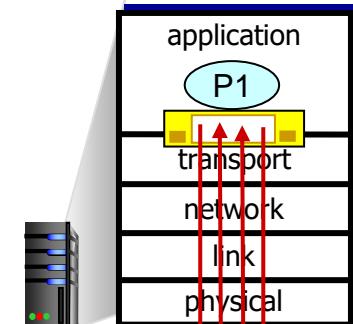
Demultiplexing không kết nối: ví dụ

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

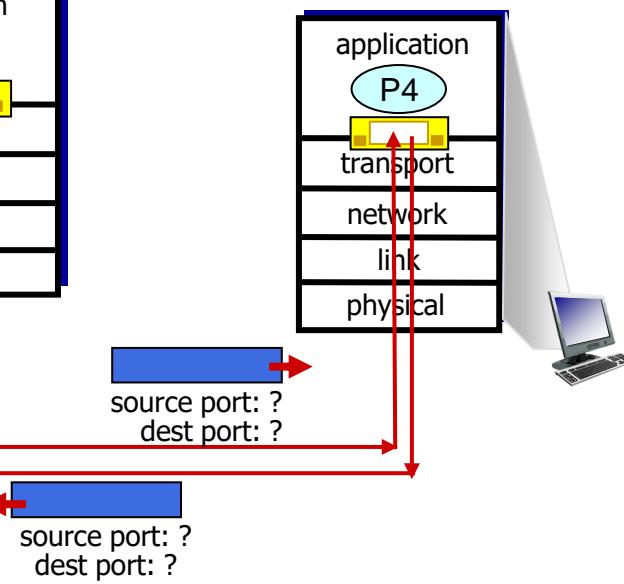


DatagramSocket

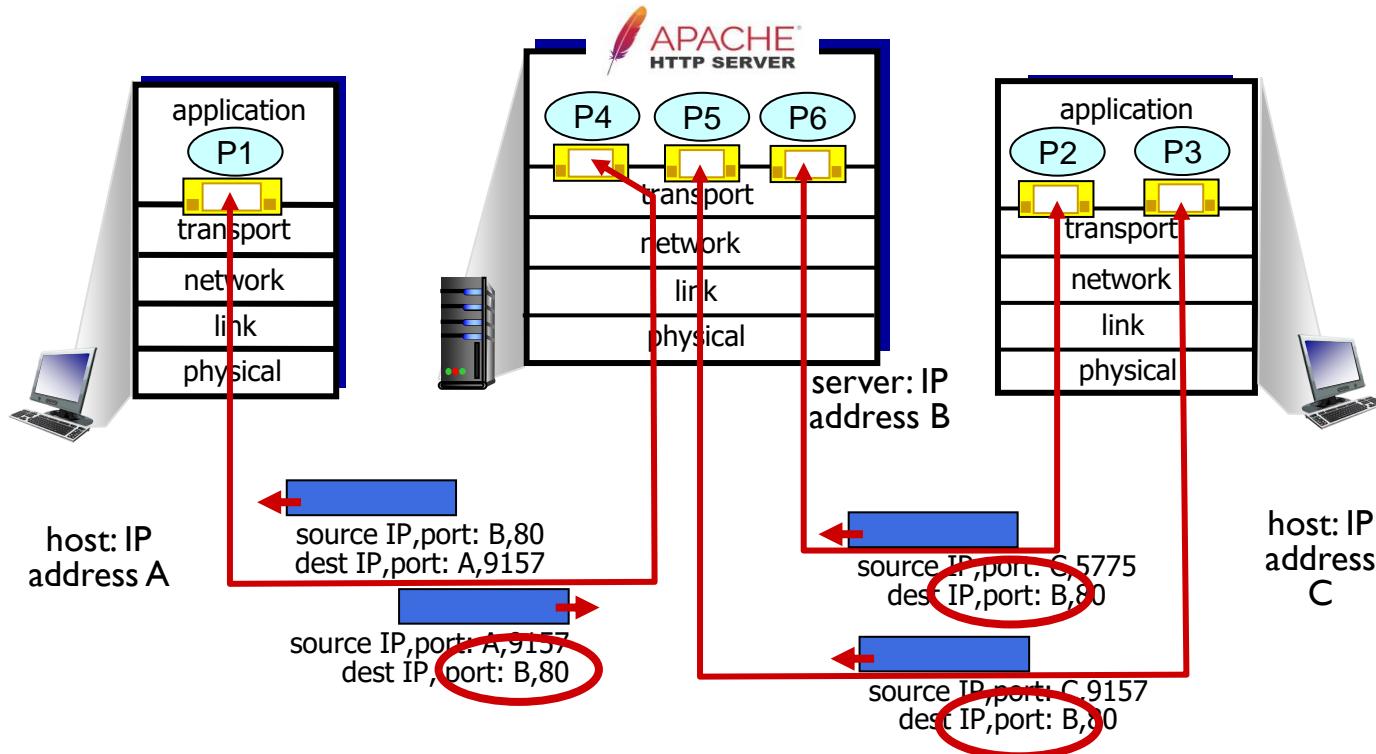
```
serverSocket = new  
DatagramSocket  
(6428);
```



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



Demultiplexing hướng kết nối: ví dụ



Ba gói tin, cùng tới địa chỉ IP B, cổng 80
được chia thành các socket khác nhau

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 nhó thì cộng giá trị nhó vào tổng
 - Đảo tổng thu được checksum

Checksum: Ví dụ

Partial Sum:	$\begin{array}{r} 01110110 \\ + 1 \\ \hline 01110111 \end{array}$
Frame 3:	$\begin{array}{r} + 11110000 \\ \hline \end{array}$
Partial Sum:	$\begin{array}{r} 1 01100111 \\ + 1 \\ \hline \end{array}$
Frame 4:	$\begin{array}{r} + 11000011 \\ \hline \end{array}$
Partial Sum:	$\begin{array}{r} 1 00101011 \\ + 1 \\ \hline \end{array}$
Sum:	$\begin{array}{r} 00101100 \\ + 1 \\ \hline \end{array}$
Checksum:	11010011

Partial Sum:	$\begin{array}{r} 01110110 \\ + 1 \\ \hline 01110111 \end{array}$
Frame 3:	$\begin{array}{r} + 11110000 \\ \hline \end{array}$
Partial Sum:	$\begin{array}{r} 1 01100111 \\ + 1 \\ \hline \end{array}$
Frame 4:	$\begin{array}{r} + 11000011 \\ \hline \end{array}$
Partial Sum:	$\begin{array}{r} 1 00101011 \\ + 1 \\ \hline \end{array}$
Sum:	00101100
Checksum:	11010011



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Truyền thông tin cậy tại tầng giao vận

ARQ

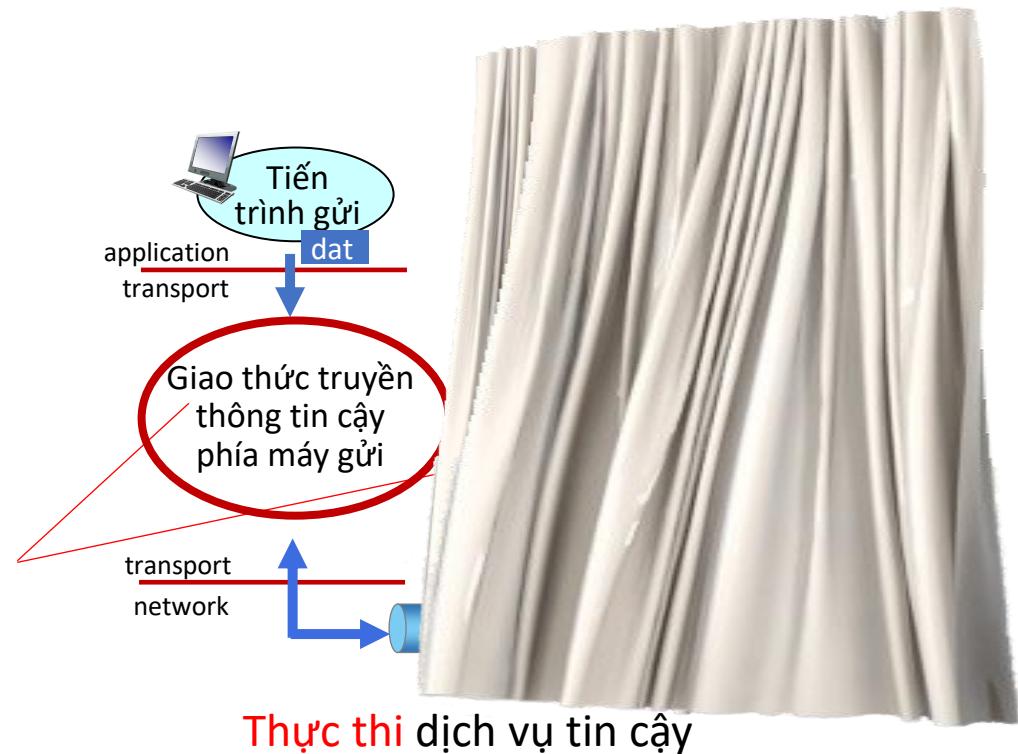
Stop-and-wait

Sliding windows

Nguyên lý của truyền thông tin cậy

Máy gửi và nhận không biết “trạng thái” của nhau (Vd: một gói tin đã nhận được chưa)

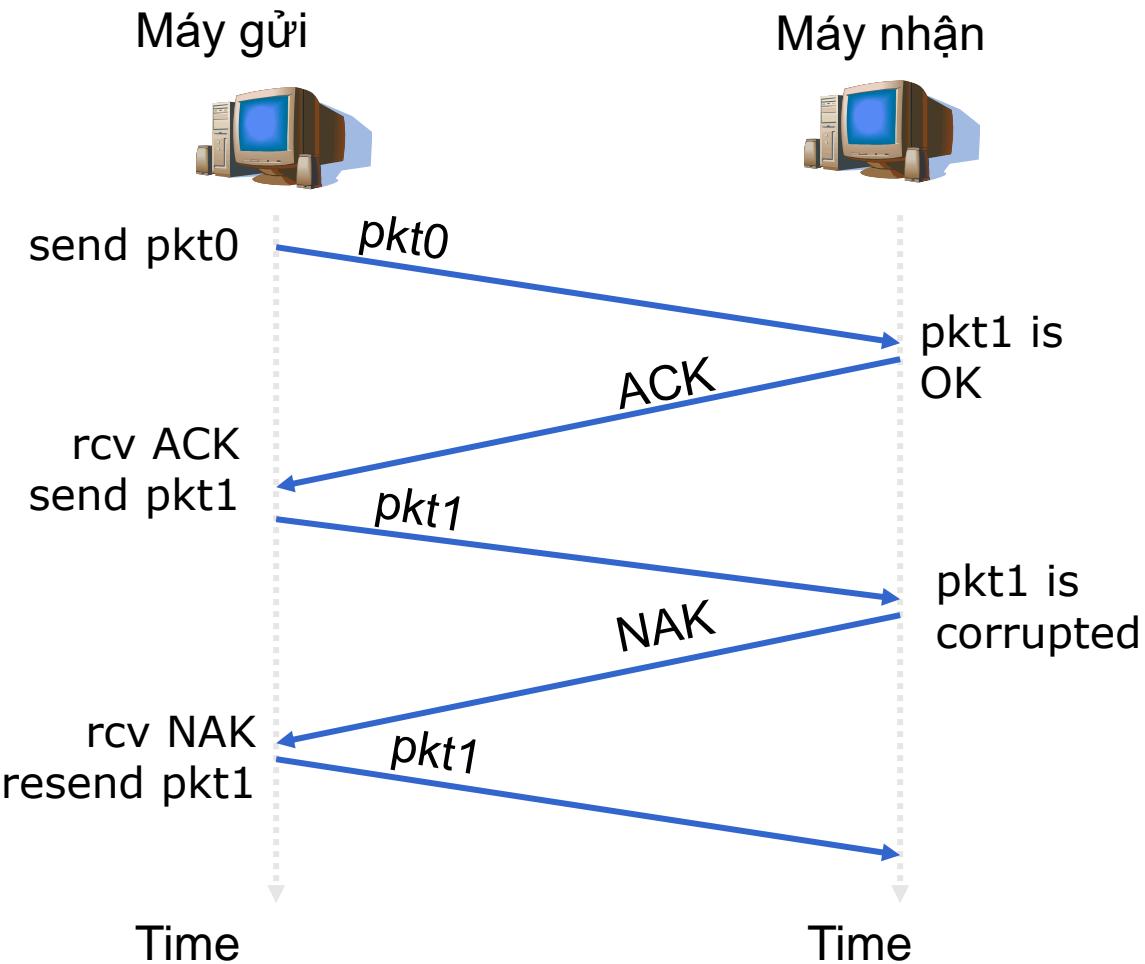
- Trừ khi giao tiếp qua 1 thông điệp



Truyền thông tin cây

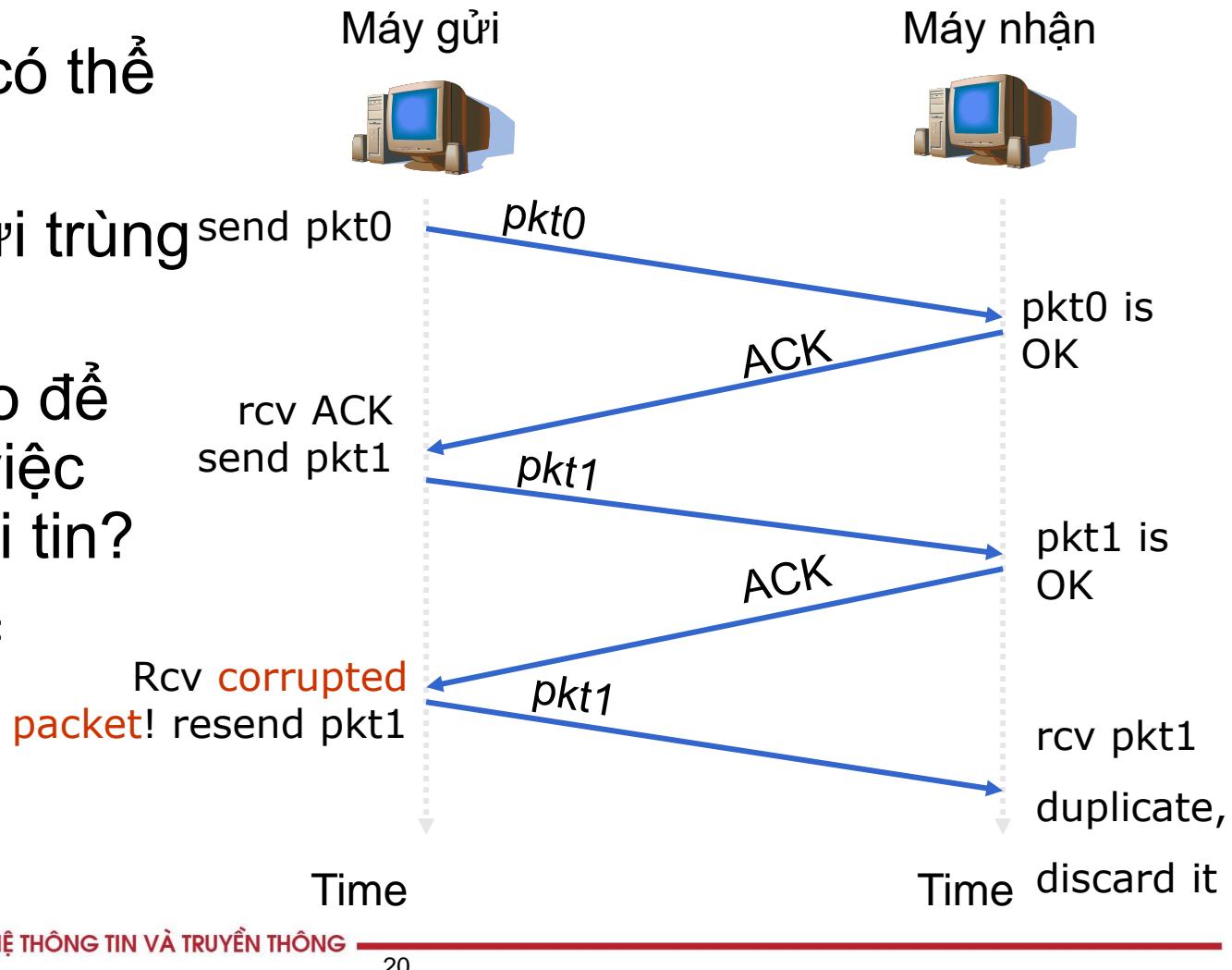
- Làm sao phát hiện lỗi?
 - Checksum
- Làm sao báo cho bên gửi?
 - ACK (*acknowledgements*):
 - NAK (*negative acknowledgements*)
- Phản ứng của bên gửi?
 - Gửi lại gói tin bị lỗi khi nhận được NAK

Điều khiển lỗi

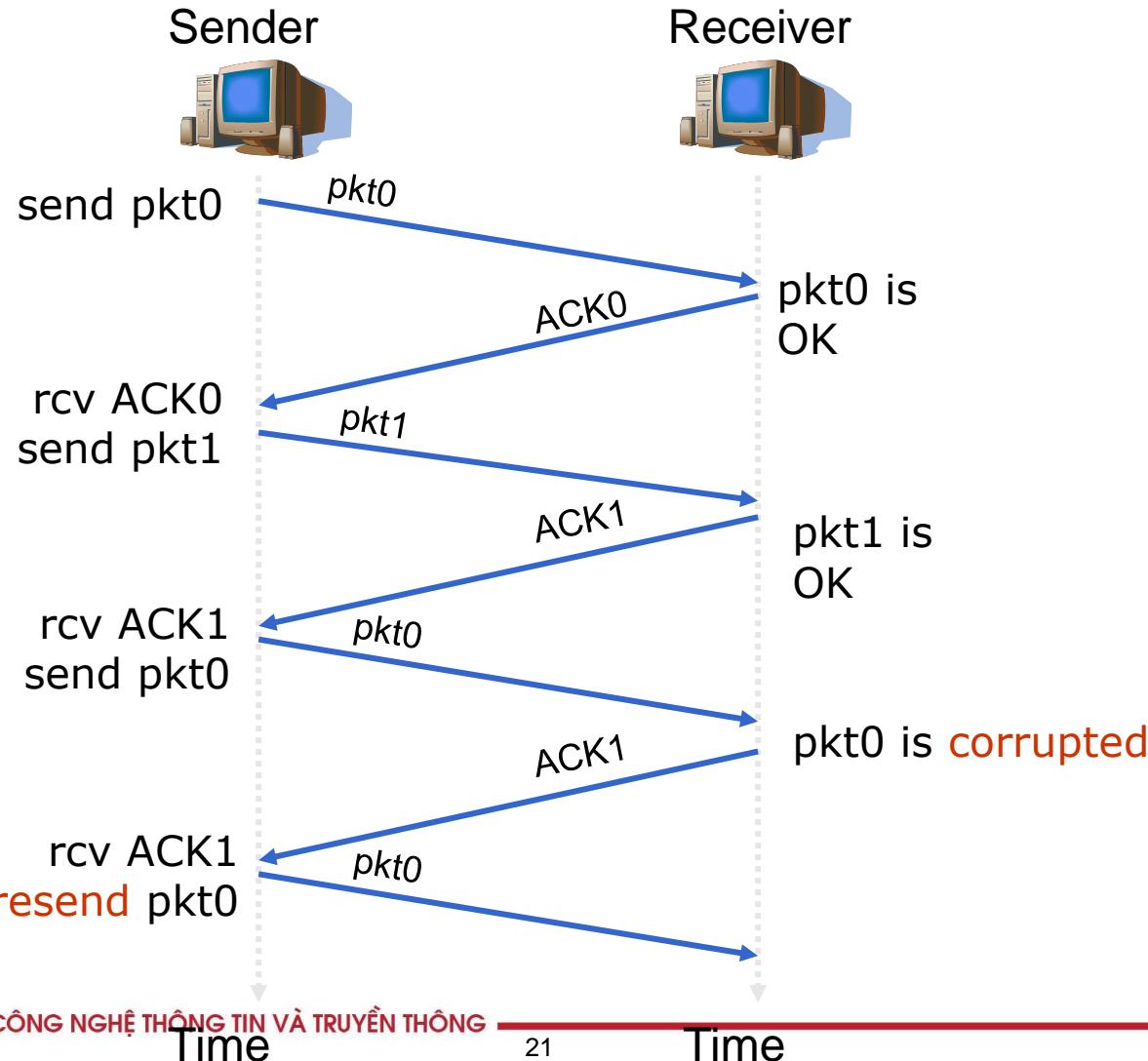


Gói tin ACK/NAK bị lỗi

- ACK/ NAK có thể bị lỗi
- Gói tin bị gửi trùng lặp
- Làm thế nào để khắc phục việc trùng lặp gói tin?
- Dùng Seq.#



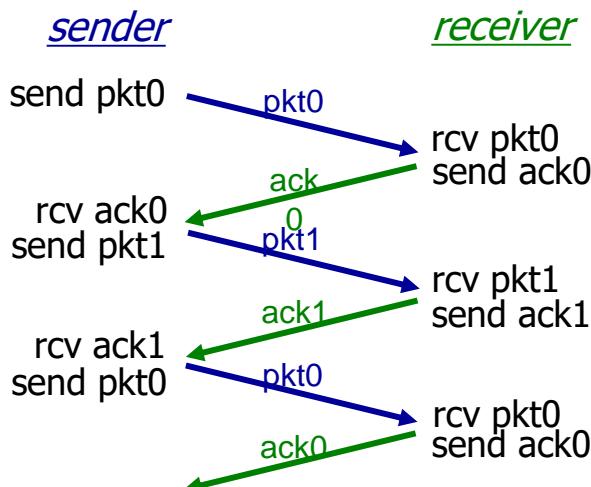
Điều khiển lỗi không cần NAK



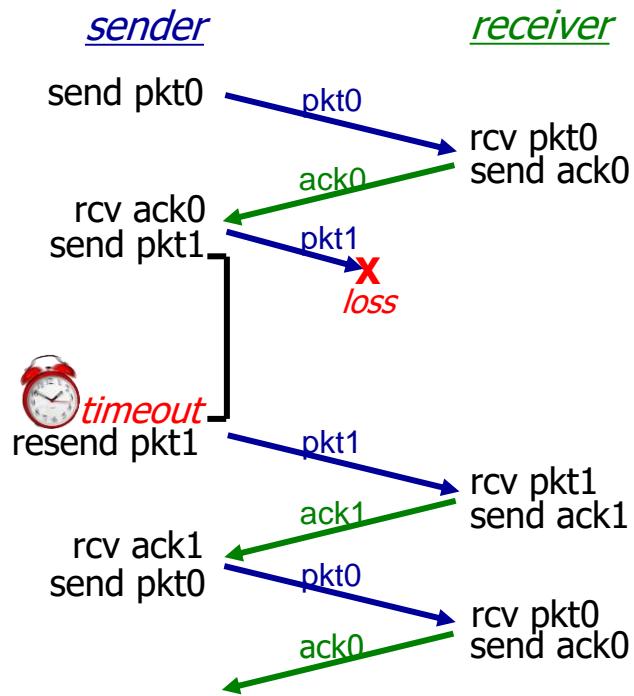
Chanel with error and packet lost

- Dữ liệu và gói tin ACK có thể bị mất trên đường truyền
 - Không nhận được ACK, làm sao máy gửi biết và gửi lại gói tin?
 - Máy gửi đợi ACK trong 1 khoảng thời gian nhất định.
Timeout
- Nên đợi trong bao lâu?
 - Ít nhất 1 RTT (Round Trip Time)
 - Cần bắt đầu bộ đếm thời gian khi gửi 1 gói tin
- Nếu gói tin đã đến và ACK bị mất?
 - Đánh số gói tin

Mô tả các kịch bản truyền thông

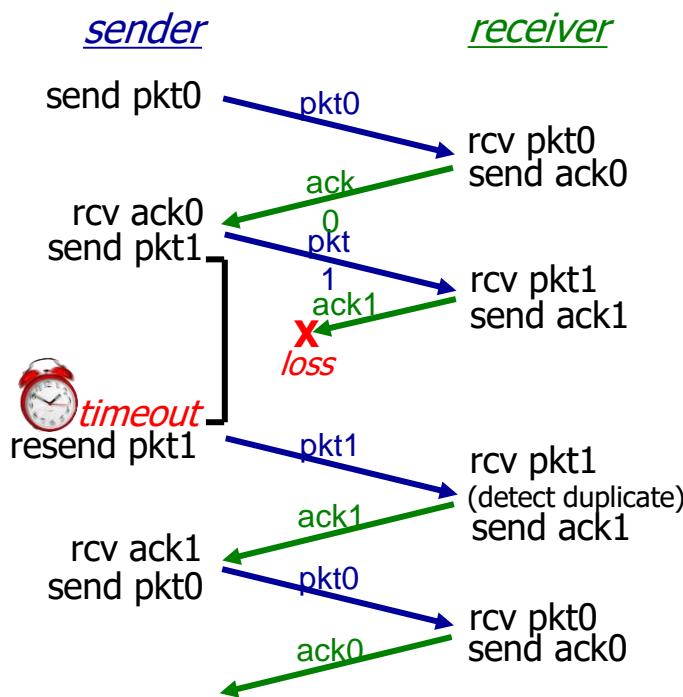


(a) Không mất gói

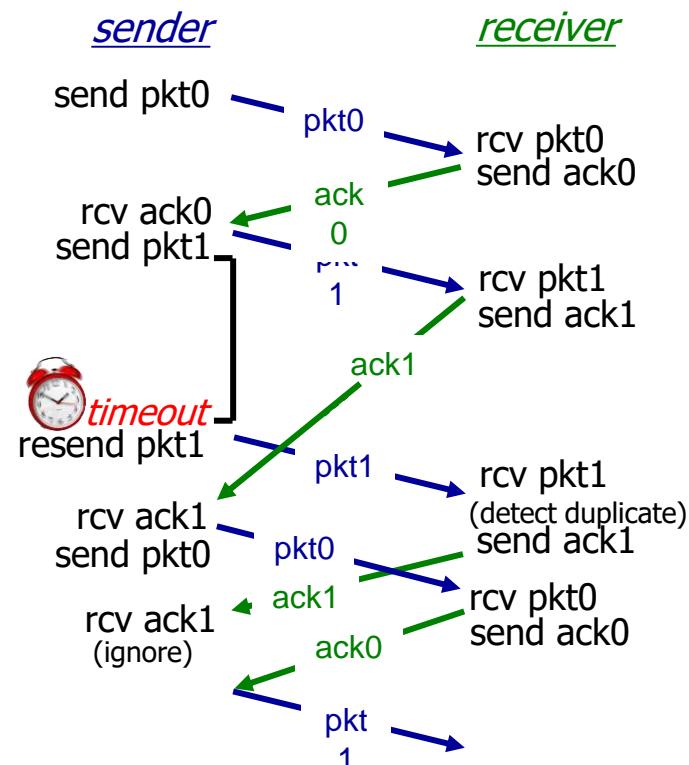


(b) mất gói

Mô tả các kịch bản truyền thông



(c) Mất ACK



(d) ACK bị trễ

Hiệu năng của truyền thông tin cậy (stop-and-wait)

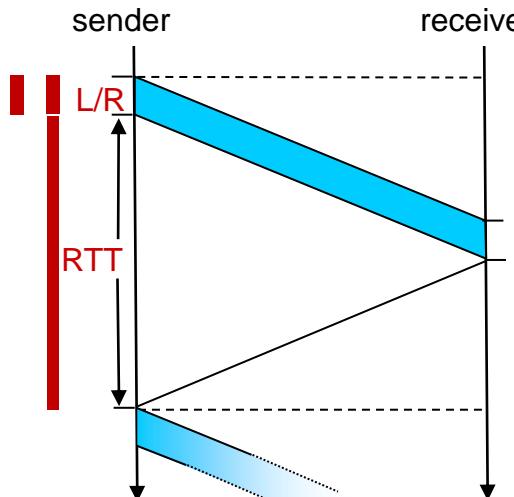
- U_{sender} : *utilization* – fraction of time sender busy sending
- example: 1 Gbps link, 15 ms prop. delay, 8000 bit packet
 - time to transmit packet into channel:

$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

Hiệu năng của truyền thông tin cây (stop-and-wait)

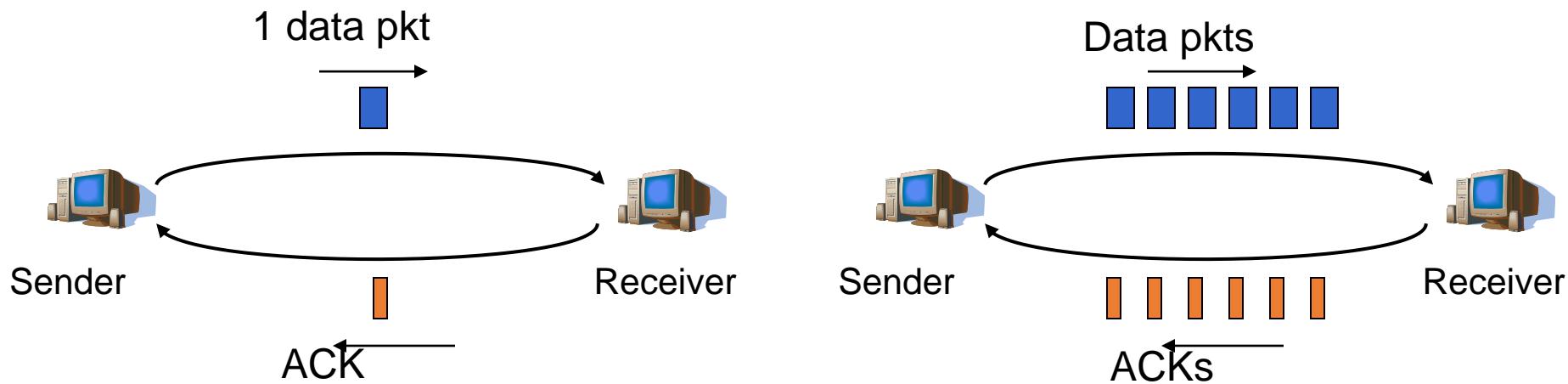
- Ví dụ: Liên kết 1 Gbps, 15 ms trễ lan truyền, gói tin 8000 bit
 - Để đẩy gói tin lên đường truyền: $D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$

$$\begin{aligned}U_{\text{sender}} &= \frac{L / R}{RTT + L / R} \\&= \frac{.008}{30.008} \\&= 0.00027\end{aligned}$$



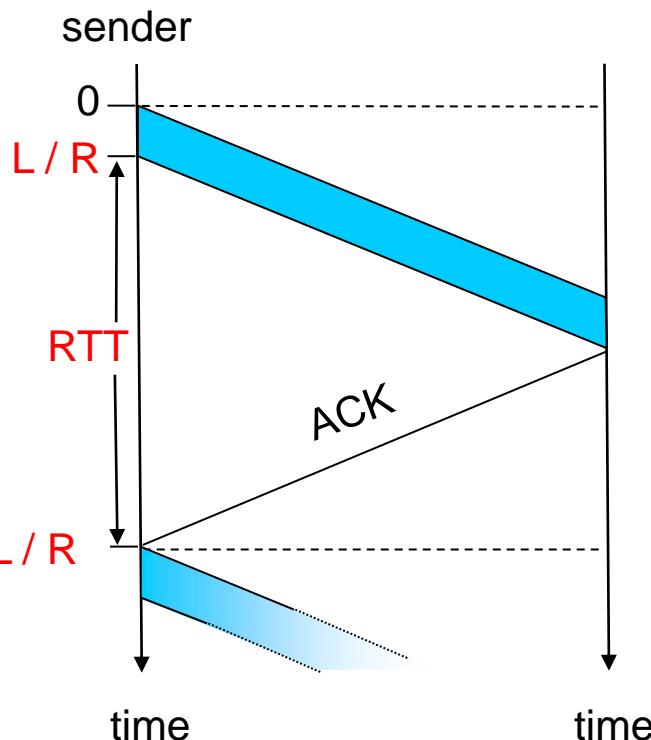
- Hiệu năng rất kém

Truyền thông kiểu đường ống



So sánh hiệu quả

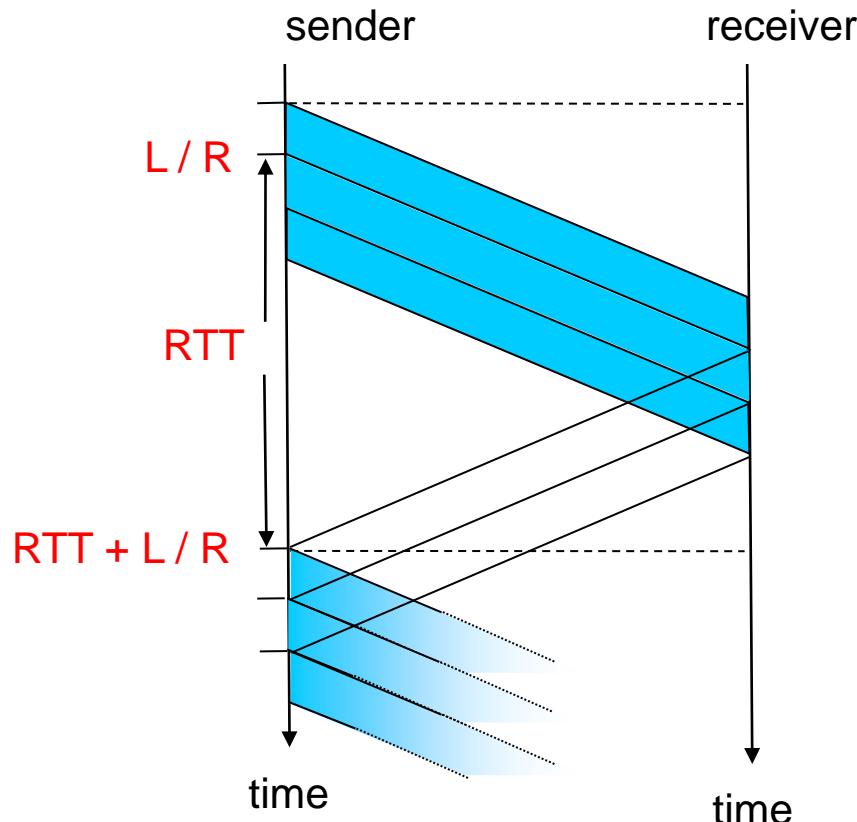
stop-and-wait



L: Size of data pkt
 R: Link bandwidth
 RTT: Round trip time

$$\text{Performance} = \frac{L / R}{RTT + L / R}$$

Pipeline



$$\text{Performance} = \frac{3 * L / R}{RTT + L / R}$$

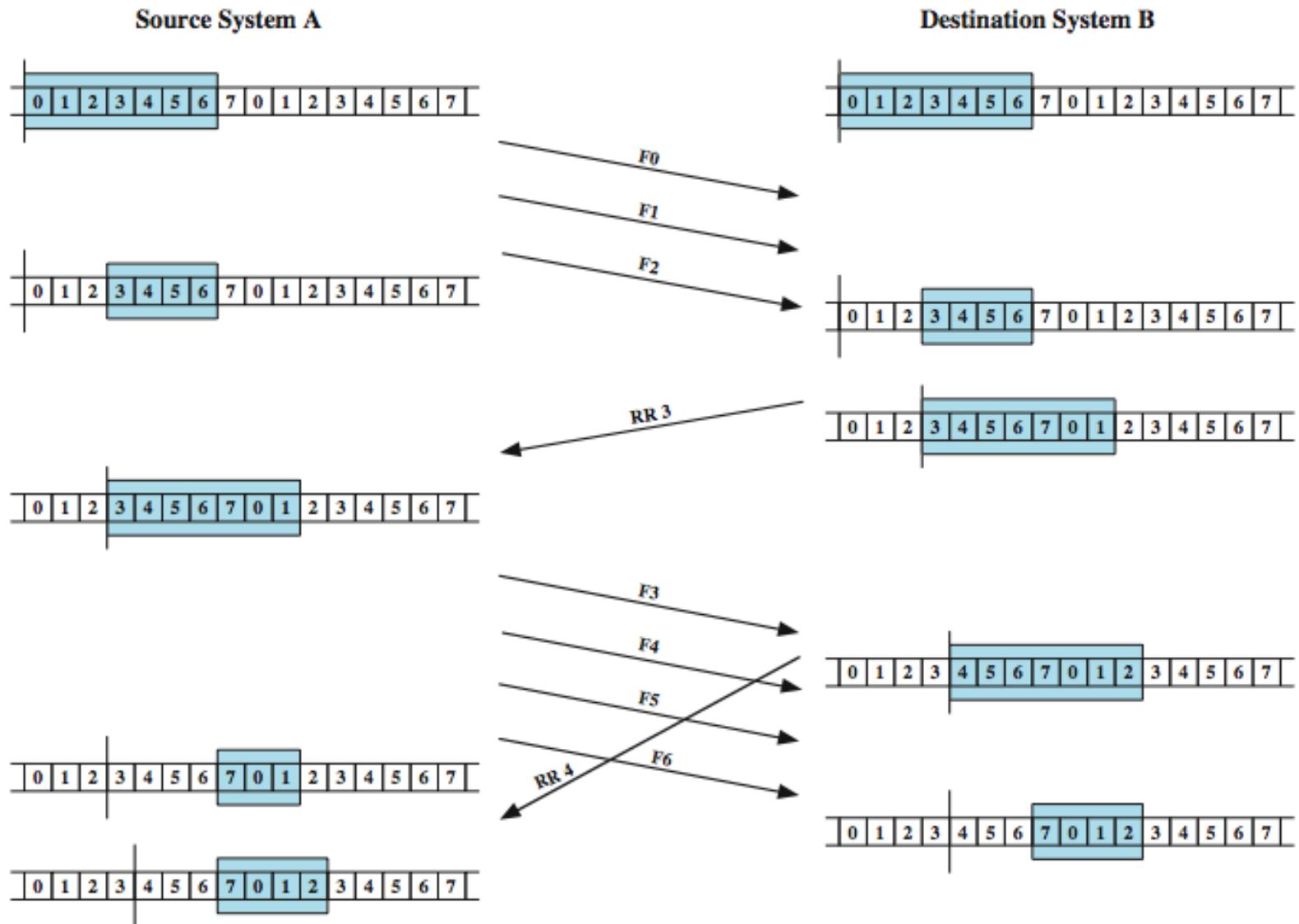
Cơ chế của sổ trượt: nguyên tắc

- Gửi nhiều khung dữ liệu để giảm thời gian chờ.
- Các khung đã gửi đi chưa báo nhận được lưu trữ tạm thời trong bộ nhớ đệm.
- Số khung được truyền đi phụ thuộc vào bộ nhớ đệm.
- Khi nhận được báo nhận
 - giải phóng khung dữ liệu đã truyền thành công khỏi bộ nhớ đệm
 - Truyền tiếp các khung bằng với số khung đã truyền thành công

Cơ chế của sổ trượt: Nguyên tắc

- Xét hai trạm A, B kết nối bằng một đường truyền song công
 - B có bộ nhớ đệm n khung dữ liệu
 - Như vậy B có thể nhận cùng một lúc n khung dữ liệu mà không cần báo nhận
- Báo nhận
 - Để ‘nhớ’ các khung dữ liệu đã báo nhận, cần đánh số các khung dữ liệu
 - B báo nhận một khung bằng cách báo số khung dữ liệu mà B đang chờ nhận, ngầm định đã nhận tất cả các khung trước đó
 - Một báo nhận có thể dùng cho nhiều khung dữ liệu

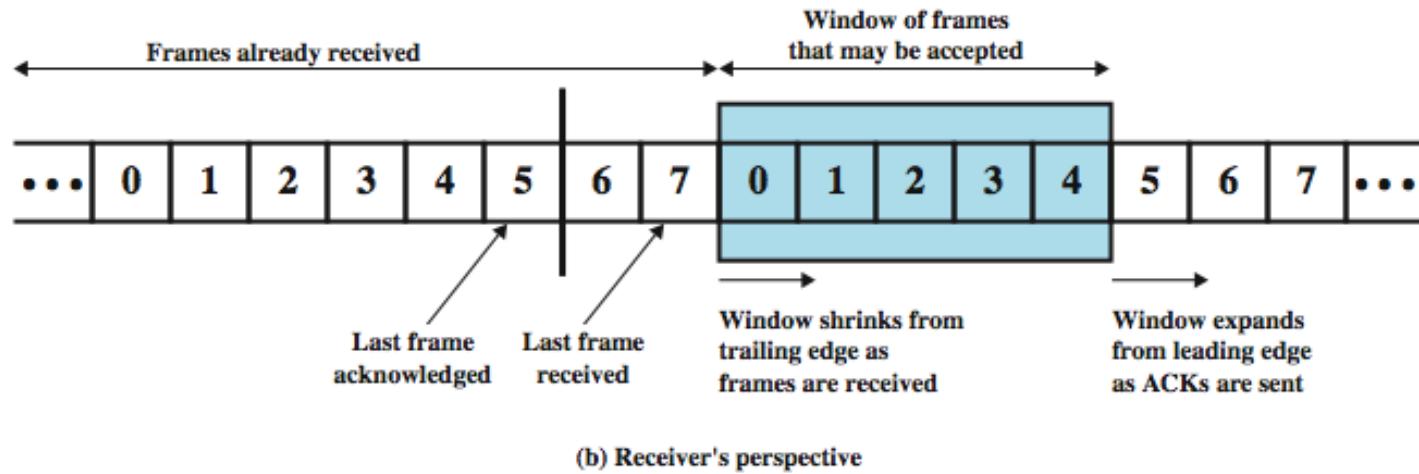
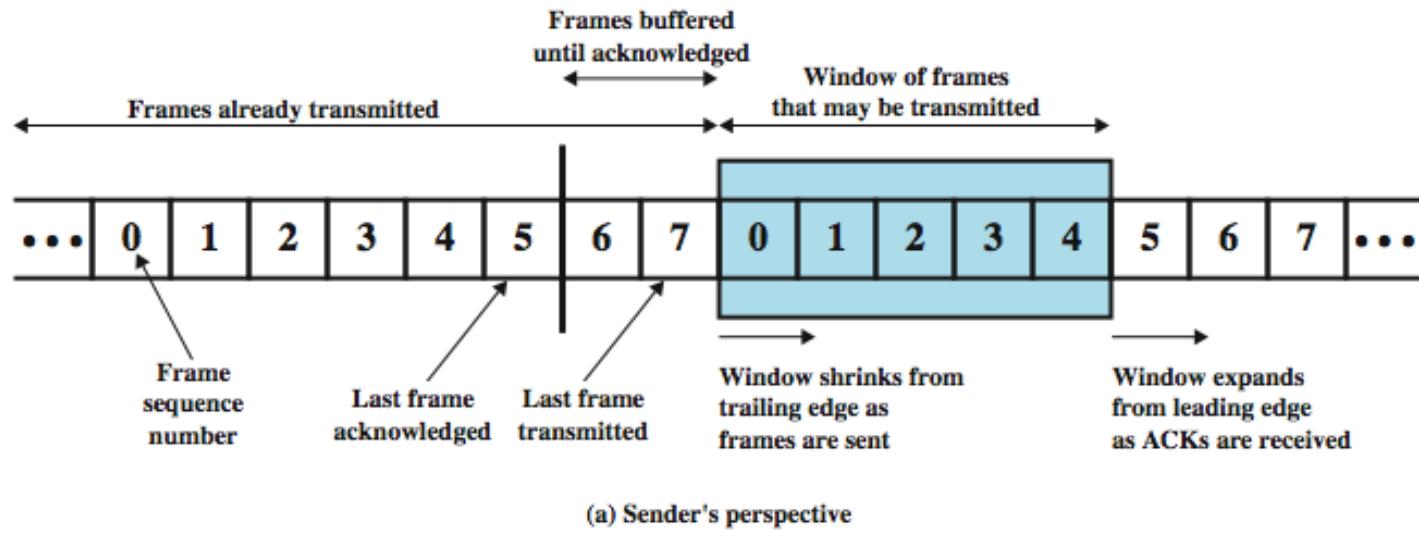
Cơ chế cửa sổ trượt



Trong cửa sổ là các khung sẽ phát

Trong cửa sổ là các khung chờ nhận

Cơ chế cửa sổ trượt



Cơ chế cửa sổ trượt

- Các khung dữ liệu đang được gửi đi được đánh số. Số thứ tự phải lớn hơn hoặc bằng kích thước cửa sổ
- Các khung dữ liệu được báo nhận 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.. đã nhận được

Cơ chế cửa sổ trượt

- Nguồn quản lý
 - Các khung đã gửi đi thành công
 - Các khung đã gửi đi chưa báo nhận
 - Các khung có thể gửi đi ngay
 - Các khung chưa thể gửi đi ngay
- Đích quản lý
 - Các khung đã nhận được
 - Các khung đang chờ nhận

Các kỹ thuật báo nhận, phát lại

- **Kỹ thuật tự động truyền lại (ARQ automatic repeat request).**
- Có 3 phiên bản chuẩn hóa
 - Dừng và chờ (Stop and Wait) ARQ
 - Đã trình bày
 - Quay lại N (Go Back N) ARQ
 - Loại bỏ chọn lọc (Selective Reject) ARQ

Kỹ thuật tự động truyền lại ARQ

- Quay lại N (Go Back N) ARQ
 - Phiên bản đơn giản của Sliding windows.
 - Phía gửi phát liên tục các gói tin (max N gói)
 - Phía nhận phát liên tục các ACK nếu gói được nhận tốt
 - Phía gửi đặt timeout phải nhận ACK cho từng gói tin
 - Khi gặp timeout phía gửi **tự động phát lại tất cả các gói tin bắt đầu từ gói đầu tiên không nhận được ACK**
- Loại bỏ chọn lọc (Selective Reject) ARQ
 - Phía gửi phát liên tục các gói tin (max N gói)
 - Phía nhận phát liên tục các ACK nếu gói được nhận tốt
 - Phía gửi đặt timeout phải nhận ACK cho từng gói tin
 - Phía gửi **tự động phát lại chỉ gói tin không nhận được ACK sau timeout**

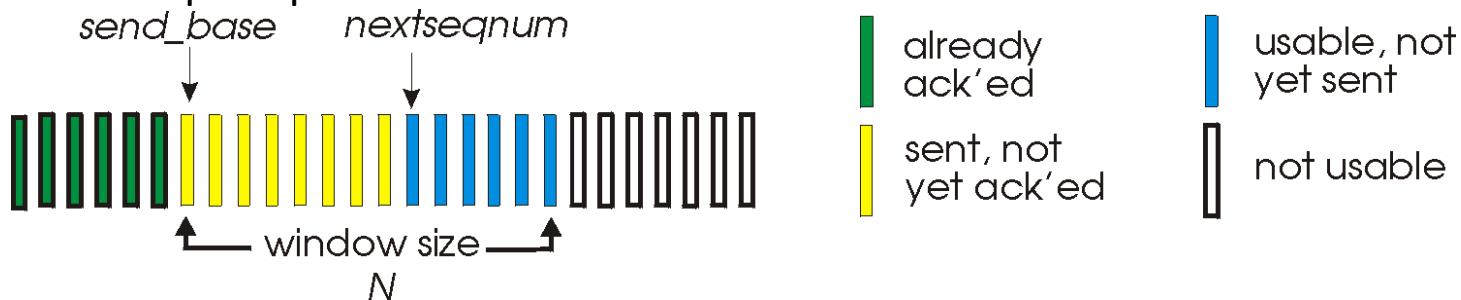
Kỹ thuật tự động truyền lại ARQ

- Các kỹ thuật ARQ được sử dụng trong cả kiểm soát luồng và kiểm soát lỗi
 - Khi gói tin đến đích bị lỗi cần phát lại
 - Khi gói tin đến đích không đúng thứ tự cần phát lại
 - Khi gói tin bị mất hoặc ACK bị mất.

Go-Back-N: máy gửi

- Máy gửi: “cửa sổ” tối đa N , các gói tin đã được gửi nhưng chưa nhận được ACK

- k-bit seq # in pkt header

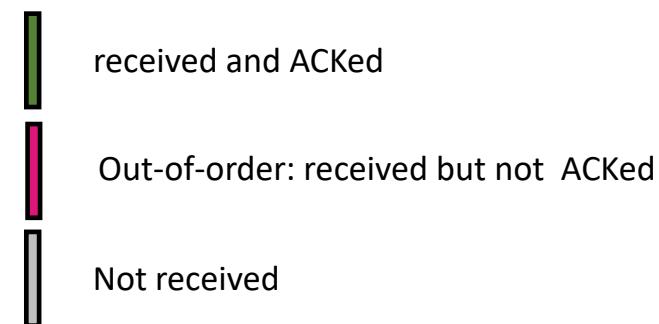
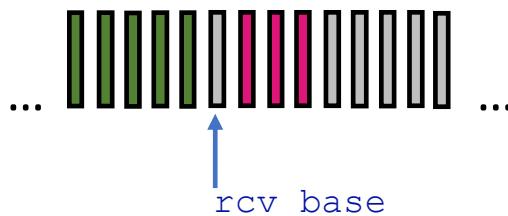


- ACK tích lũy:*** $ACK(n)$: ACKs tới gói tin có số thứ tự seq # n
- Khi nhận được $ACK(n)$: dịch cửa sổ lên phía trước 1 vị trí tại $n+1$
 - Bộ đếm thời gian cho gói tin được gửi sớm nhất
- timeout(n):*** gửi lại gói tin n và những gói khác với seq# lớn hơn trong cửa sổ

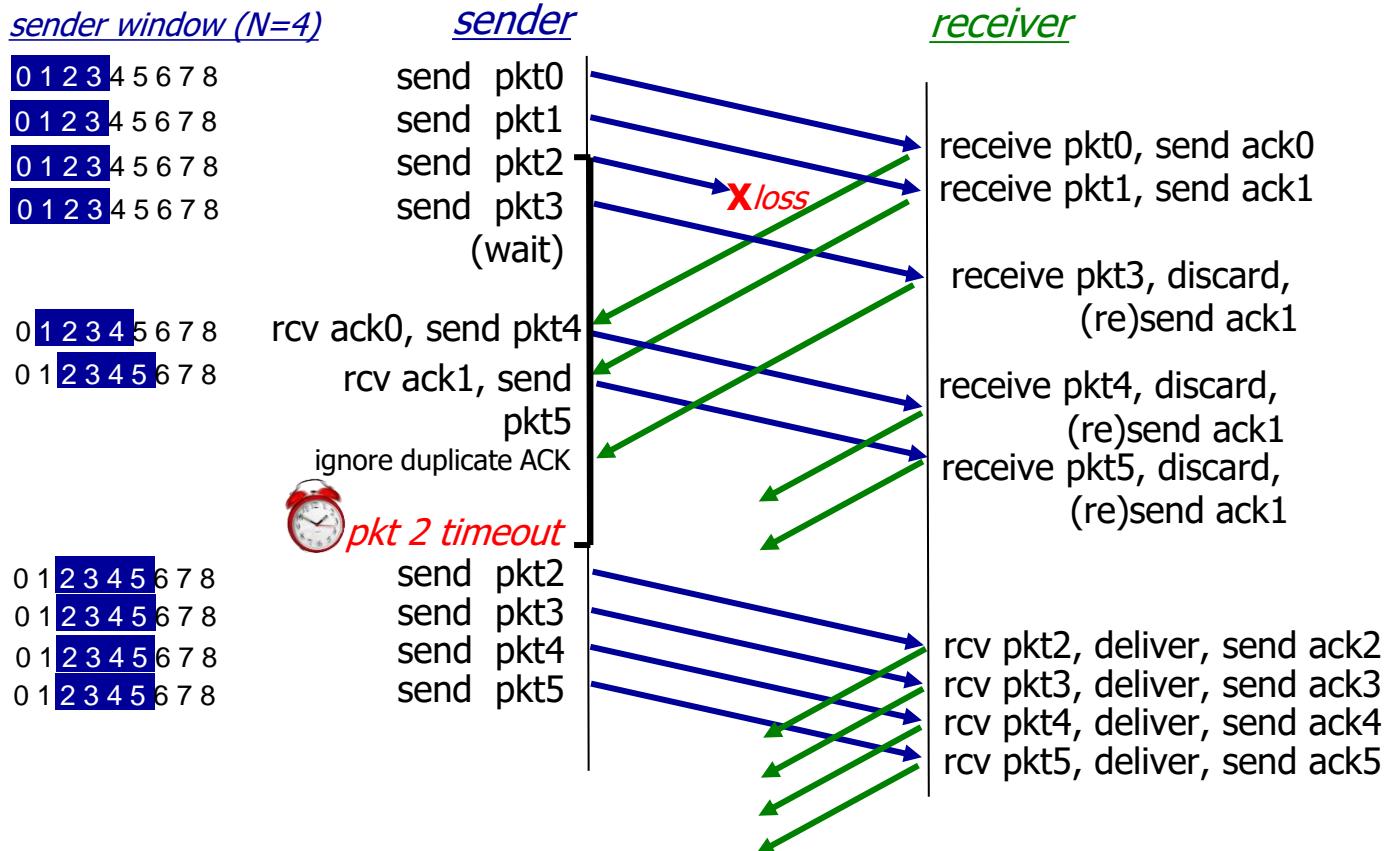
Go-Back-N: máy nhận

- Chỉ-ACK: luôn gửi ACK cho những gói đến đúng thứ tự, với seq# lớn nhất **theo thứ tự**
 - Có thể tạo ACK lặp
 - Chỉ cần nhớ `rcv_base`
- Khi nhận được gói tin không đúng thứ tự:
 - Loại bỏ (không dùng buffer) hoặc buffer: tùy khi thực thi
 - Gửi lại ACK với thứ tự seq# lớn nhất đúng thứ tự

Receiver view of sequence number space:



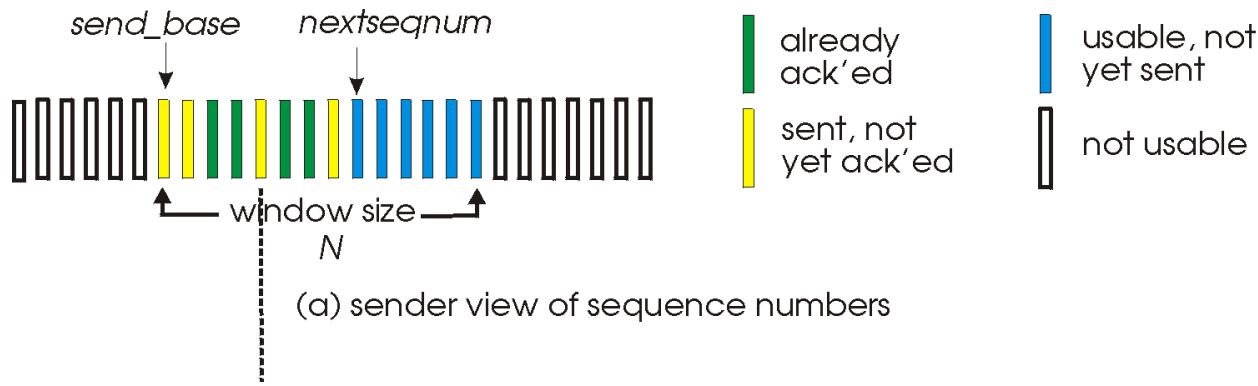
Go-Back-N khi thực hiện



Selective repeat

- Máy nhận đánh dấu từng gói tin nhận được thành công
- Máy nhận đánh dấu và gửi (ACK) từng gói tin nhận được thành công
 - Lưu gói tin bằng buffer để có thể chuyển lên tầng trên theo đúng thứ tự
- Máy gửi gửi lại những gói tin chưa có ACK đơn lẻ khi timeout
 - Máy gửi cần duy trì bộ đếm cho từng gói tin đã gửi (nhưng chưa nhận được ACK)
- Cửa sổ máy gửi
 - *N* gói tin liên tiếp
 - Giới hạn số gói tin được gửi mà chưa nhận được ACK

Selective repeat: cửa sổ máy gửi và nhận



Selective repeat: máy gửi và nhận

sender

data từ tầng trên:

- Nếu seq# tiếp theo trong cửa sổ sẵn sàng, gửi gói tin

timeout(n):

- Gửi lại gói tin n , reset bộ đếm

ACK(n) trong [sendbase, sendbase+N]:

- Đánh dấu gói tin n là “đã nhận”
- Nếu n là gói tin ACK “nhỏ nhất”, dịch cửa sổ 1 vị trí tới seq# tiếp theo (chưa nhận được ACK)

receiver

Gói tin n trong [rcvbase, rcvbase+N-1]

- Gửi ACK(n)
- Không đúng thứ tự: buffer
- Đúng thứ tự: chuyển toàn bộ gói tin (bao gồm cả buffer) lên tầng trên, dịch cửa sổ 1 vị trí

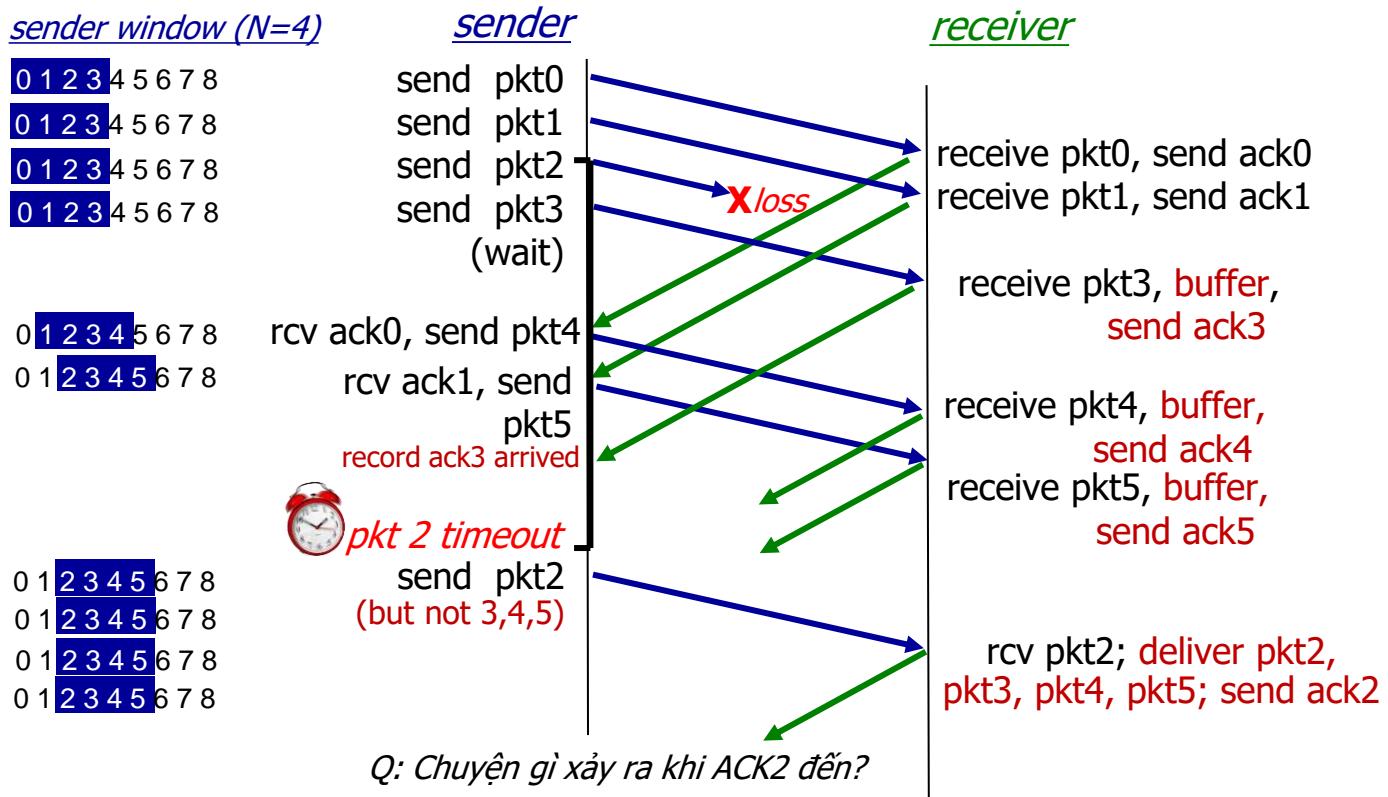
Gói tin n trong [rcvbase-N, rcvbase-1]

- ACK(n)

Trái lại:

- Bỏ qua

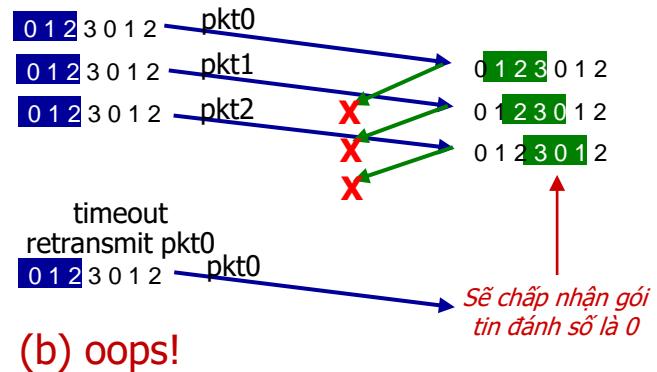
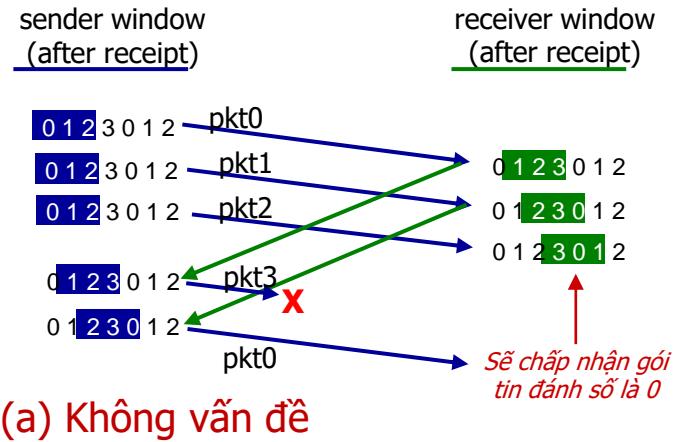
Selective Repeat khi thực hiện



Selective repeat: vẫn đê!

Ví dụ:

- seq #s: 0, 1, 2, 3 (bộ đếm cơ số 4)
- Kích thước cửa sổ=3

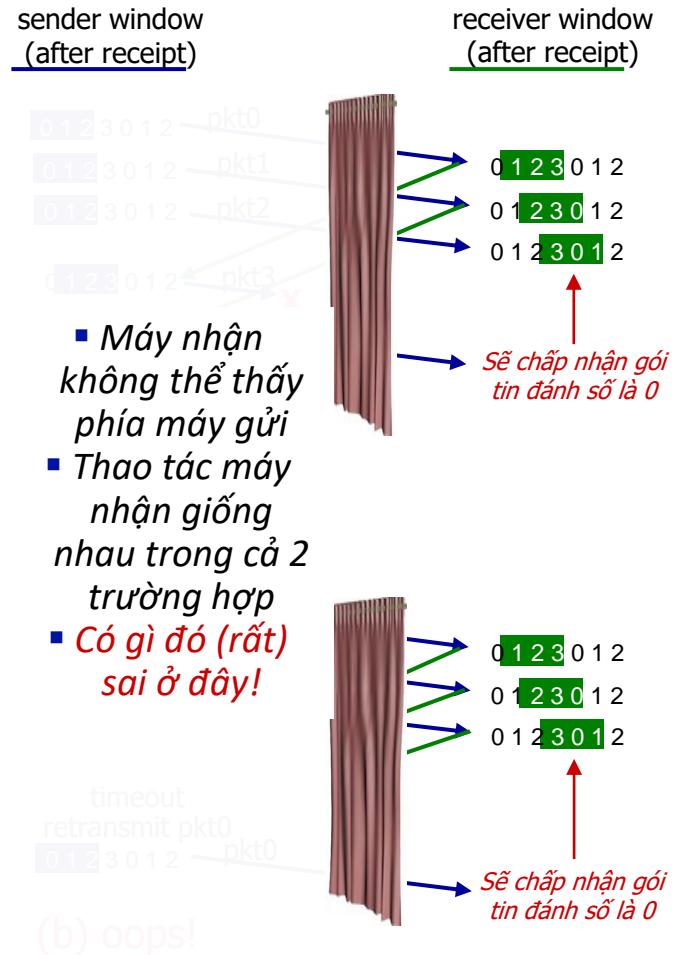


Selective repeat: vẫn đê!

Ví dụ:

- seq #: 0, 1, 2, 3 (bộ đếm cơ số 4)
- Kích thước cửa sổ=3

Q: Mỗi quan hệ giữa kích thước seq# và kích thước cửa sổ là bao nhiêu để tránh kịch bản này?





ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

UDP User Datagram Protocol

Tổng quan

Khuôn dạng gói tin

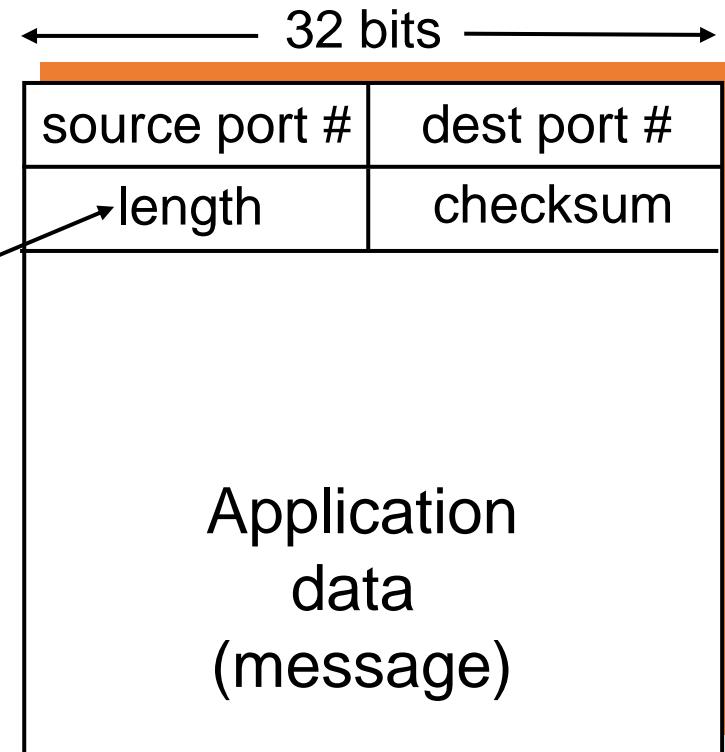
Giao thức dạng “Best effort”

- Dùng UDP khi nào?
 - Không cần thiết lập liên kết (tăng độ trễ)
 - Đơn giản: Không cần lưu lại trạng thái liên kết ở bên gửi và bên nhận
 - Phàn đầu đoạn tin nhỏ
 - Không có quản lý tắc nghẽn: UDP cứ gửi dữ liệu nhanh nhất, nhiều nhất nếu có thể
- UDP có những chức năng cơ bản gì?
 - Dồn kênh/phân kênh
 - Phát hiện lỗi bit bằng checksum

Khuôn dạng bức tin (datagram)

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

Độ dài toàn bộ bức tin tính theo byte



Khuôn dạng đơn vị dữ liệu của UDP

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



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

TCP

Transmission Control Protocol

Cấu trúc đoạn tin TCP

Quản lý liên kết

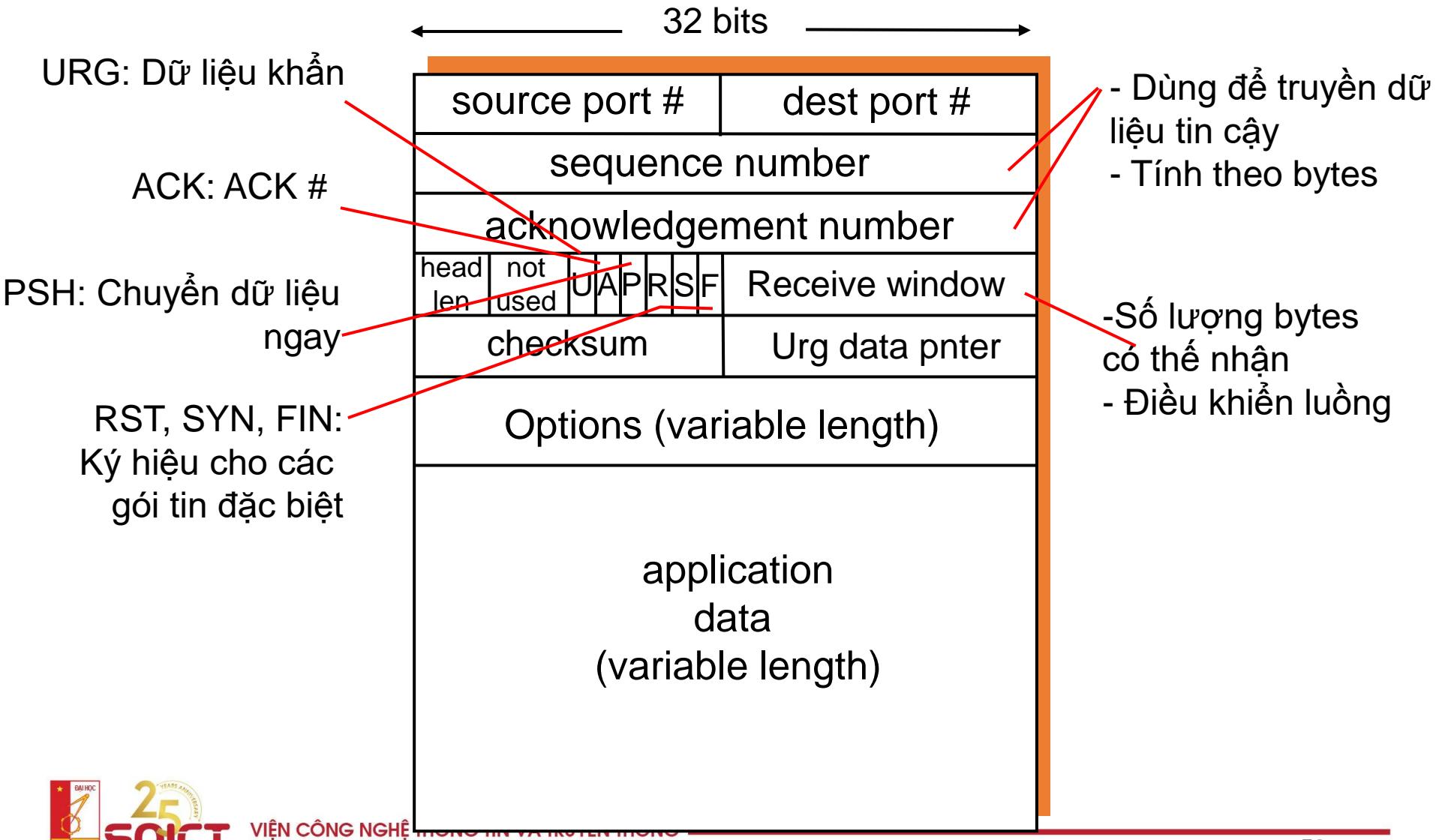
Kiểm soát luồng

Kiểm soát tắc nghẽn

Tổng quan về TCP

- Giao thức hướng liên kết
 - Bắt tay ba bước
- Giao thức truyền dữ liệu theo dòng byte, tin cậy
 - Sử dụng vùng đệm
- Truyền theo kiểu pipeline, sử dụng sliding windows.
 - Tăng hiệu quả
- Kiểm soát luồng
 - Bên gửi không làm quá tải bên nhận (thực tế: quá tải)
- Kiểm soát tắc nghẽn
 - Việc truyền dữ liệu không nên làm tắc nghẽn mạng (thực tế: luôn có tắc nghẽn)

Khuôn dạng đoạn tin - TCP segment



TCP cung cấp dịch vụ tin cậy ntn?

- Kiểm soát dữ liệu đã được nhận chưa:
 - Seq. #
 - Ack
- Chu trình làm việc của TCP:
 - Thiết lập liên kết
 - Bắt tay ba bước
 - Truyền/nhận dữ liệu
 - Đóng liên kết

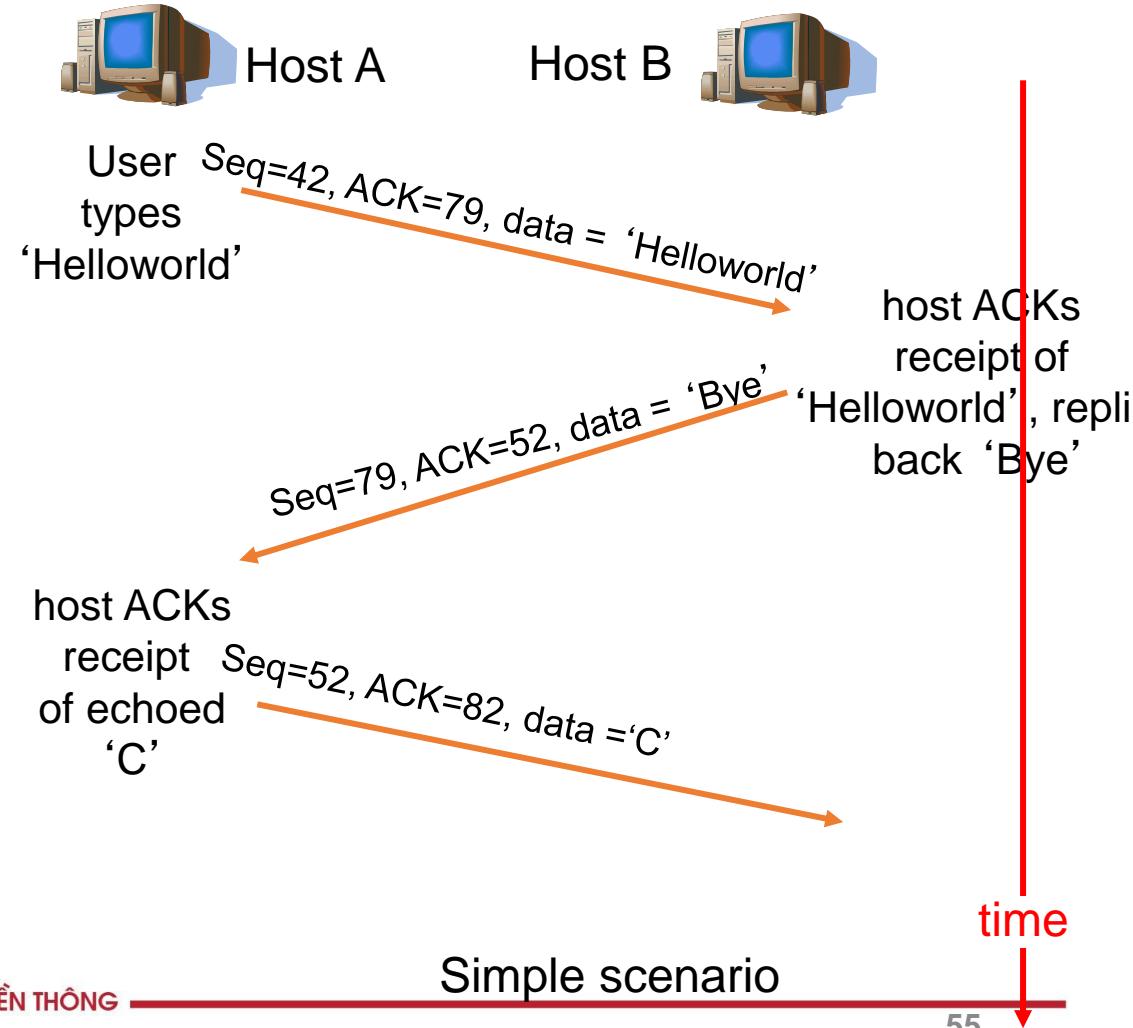
Cơ chế báo nhận trong TCP

Seq. #:

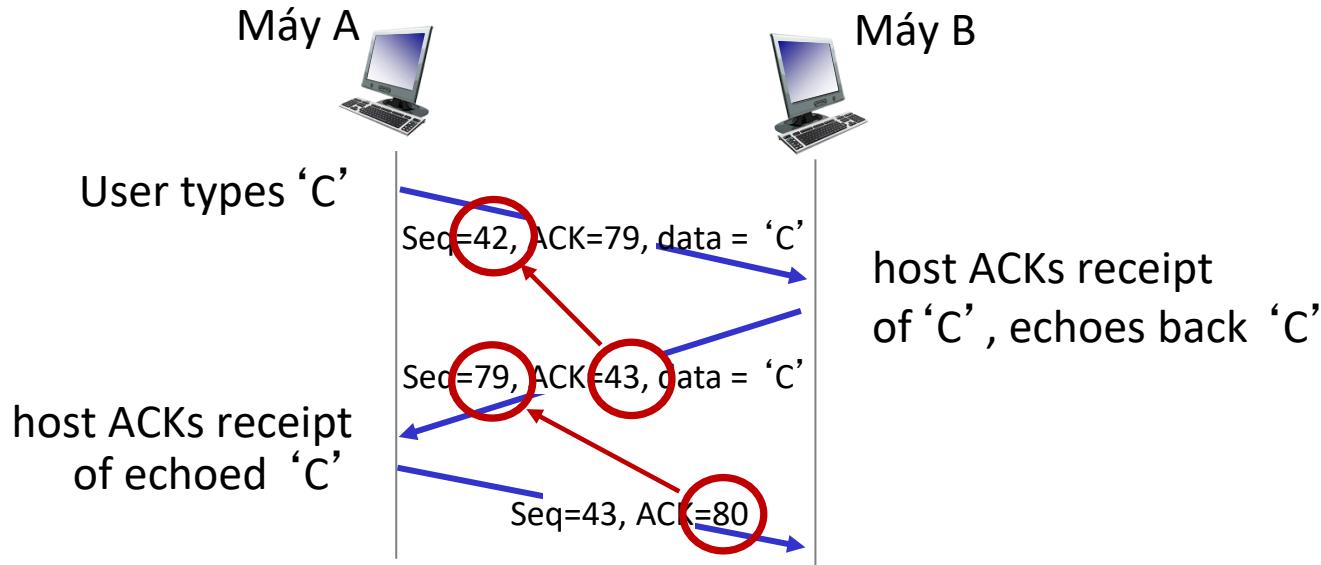
- Số hiệu của byte đầu tiên của đoạn tin trong dòng dữ liệu

ACK:

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

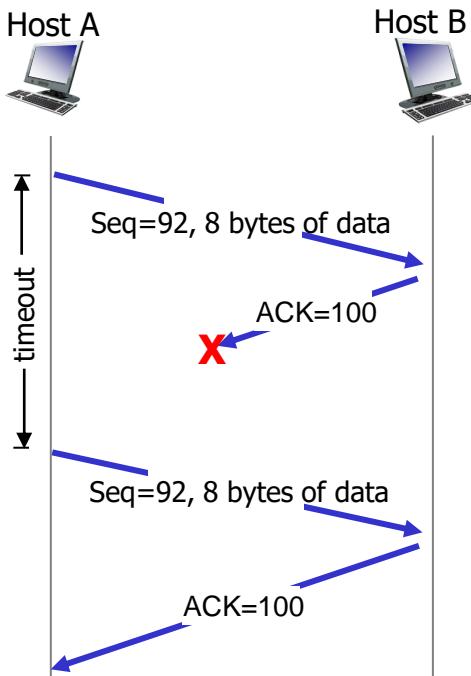


Cơ chế báo nhận trong TCP (ACK)

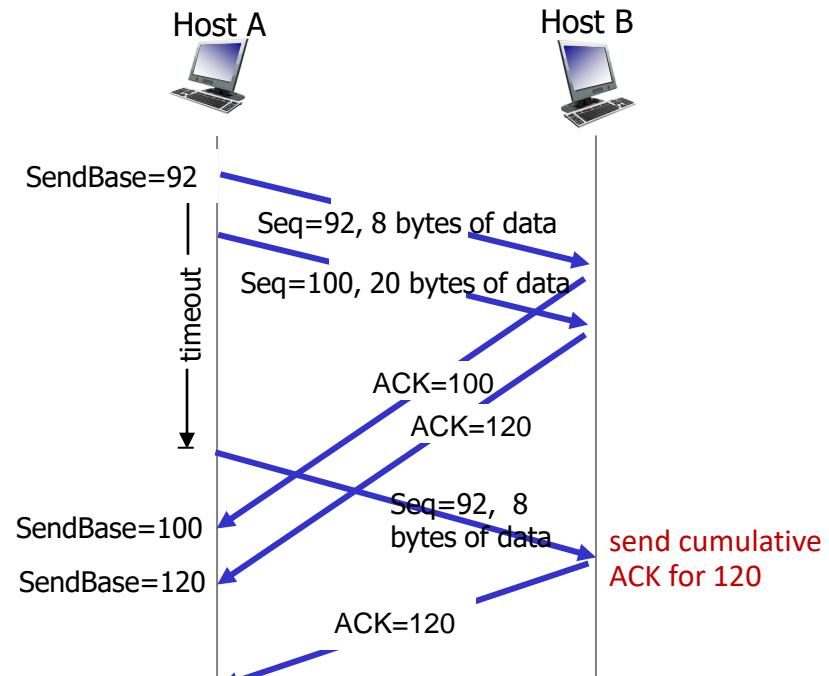


Kịch bản telnet đơn giản

TCP: kịch bản gửi lại

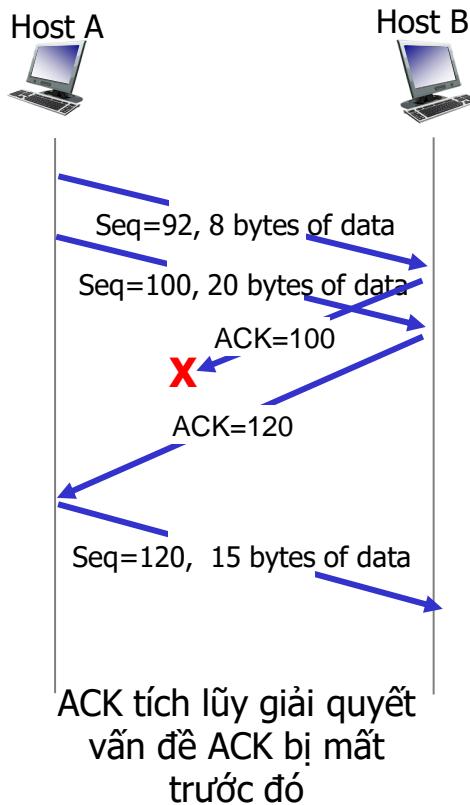


Mất ACK



Quá thời gian (ACK trễ)

TCP: kịch bản gửi lại



Gửi lại nhanh trong TCP

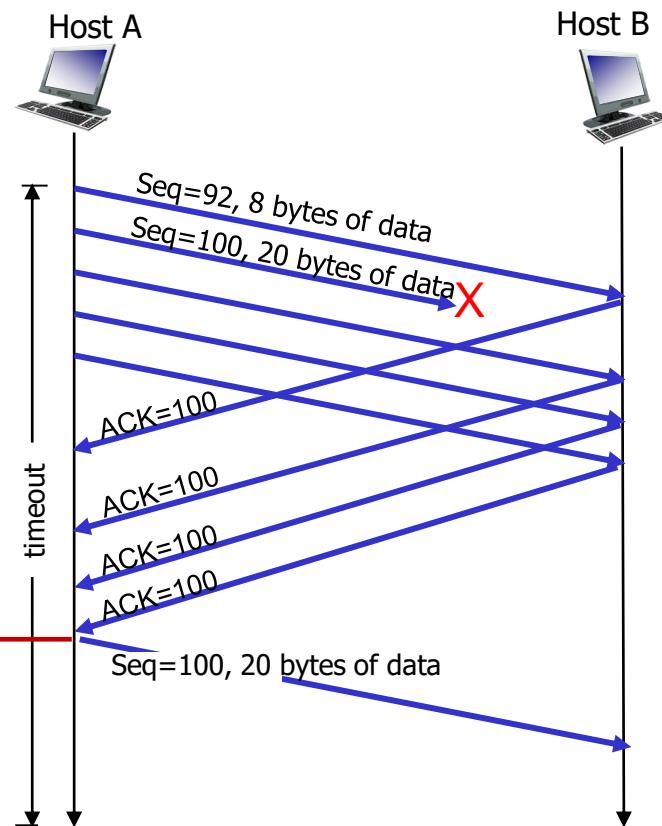
Gửi lại nhanh trong TCP

Nếu máy gửi nhận được 3 gói tin ACK cùng dữ liệu, thì gửi lại gói tin (chưa ACK) với seq# nhỏ nhất

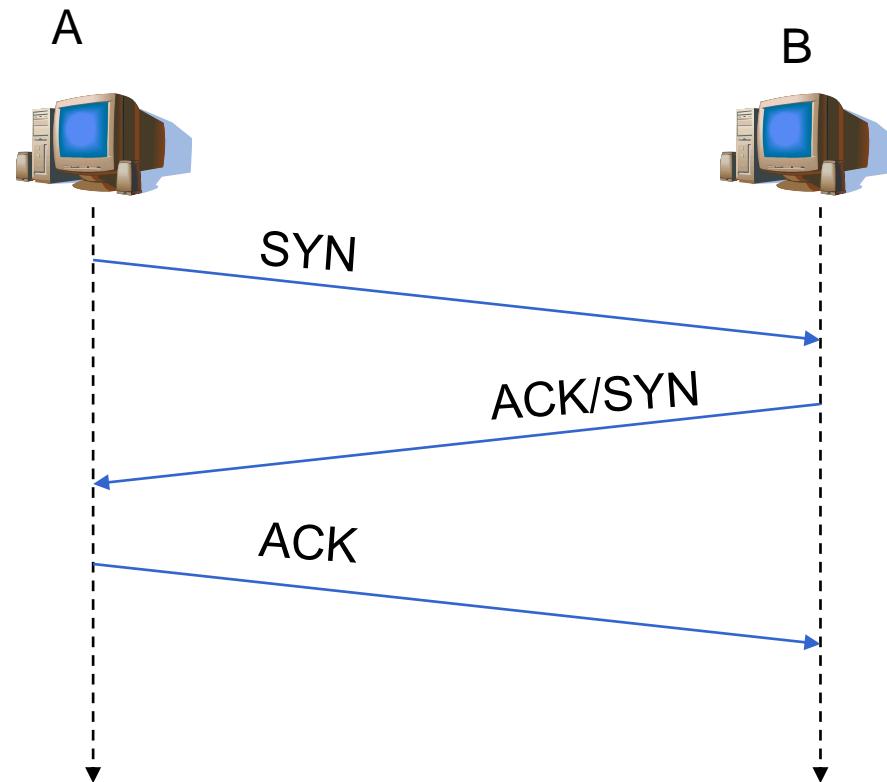
- Nhiều khả năng gói tin đó đã mất, không cần đợi timeout



Nhận 3 ACK giống nhau,
đoán gói tin đã mất, gửi lại
ngay lập tức



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

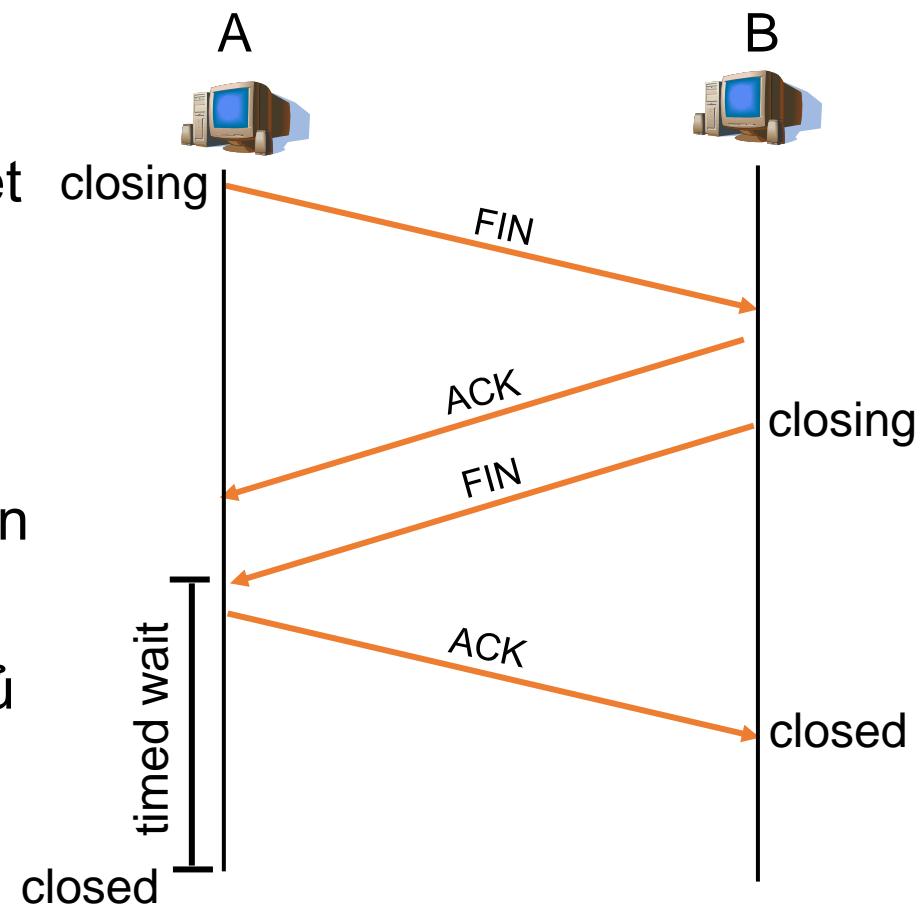


- **Bước 1:** A gửi SYN cho B
 - chỉ ra giá trị khởi tạo seq # của A
 - không có dữ liệu
- **Bước 2:** B nhận SYN, trả lời bằng SYNACK
 - B khởi tạo vùng đệm
 - chỉ ra giá trị khởi tạo seq. # của B
- **Bước 3:** A nhận SYNACK, trả lời ACK, có thể kèm theo dữ liệu

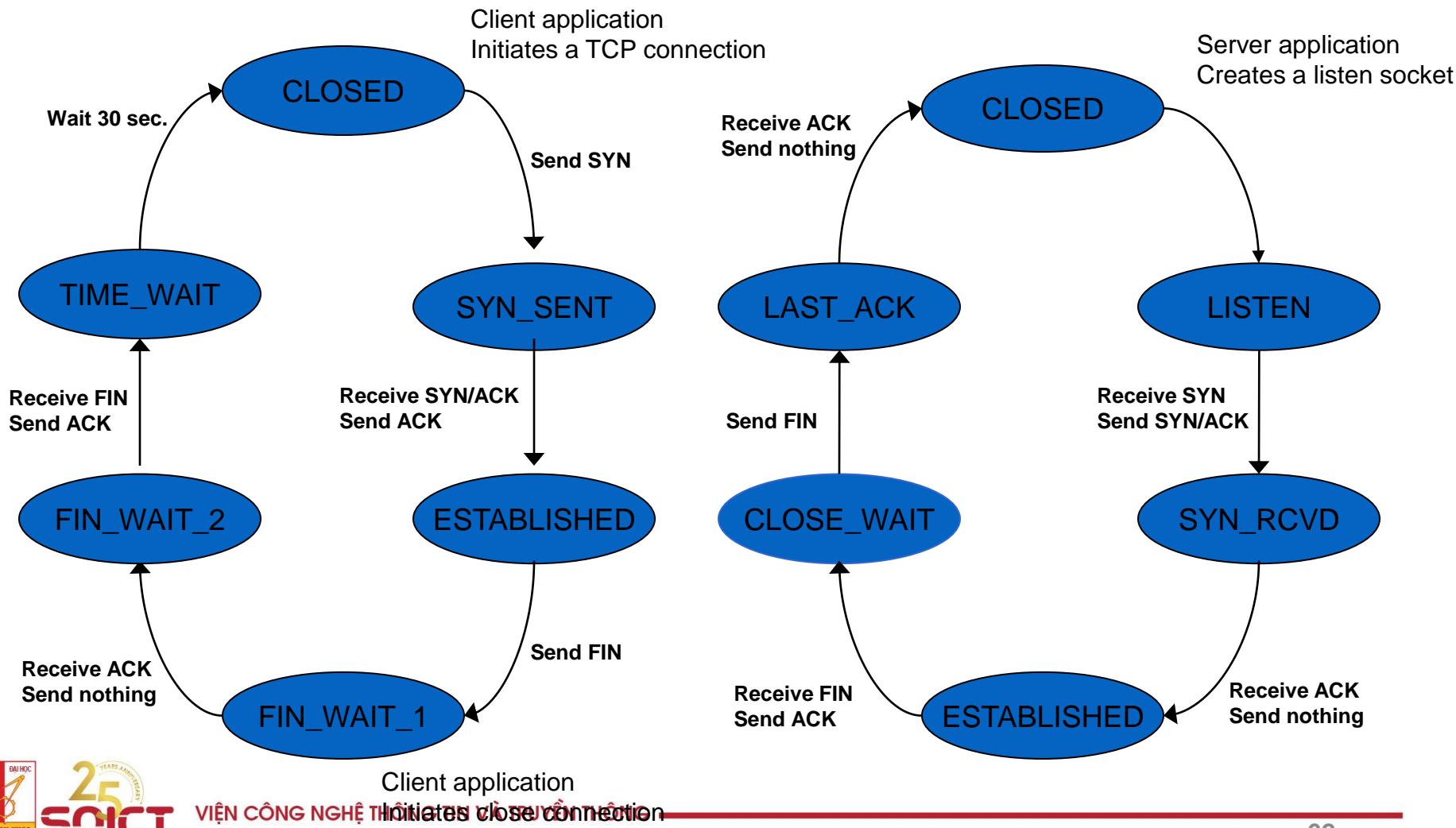
Ví dụ về việc đóng liên kết

- Bước 1: Gửi FIN cho B
- Bước 2: B nhận được FIN, trả lời ACK, đồng thời đóng liên kết closing và gửi FIN.
- Bước 3: A nhận FIN, trả lời ACK, vào trạng thái “chờ”.
- Bước 4: B nhận ACK. đóng liên kết.

Lưu ý: Cả hai bên đều có thể chủ động đóng liên kết



Chu trình sống của TCP (đơn giản hóa)





ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

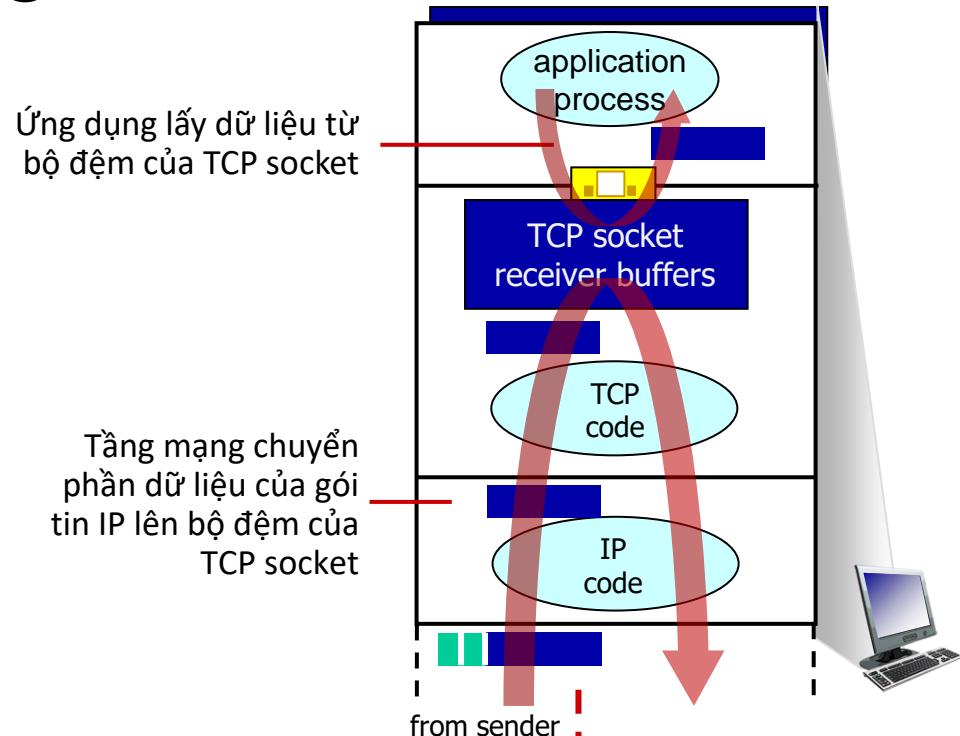
Kiểm soát luồng trong TCP

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
- 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(Rwnd, Cwnd)

Điều khiển luồng của TCP

Q: Chuyện gì xảy ra nếu tầng mạng chuyển dữ liệu lớn hơn khả năng lấy dữ liệu của tầng ứng dụng?



Tầng mạng chuyển
phần dữ liệu của gói
tin IP lên bộ đệm của
TCP socket

protocol stack của máy nhận

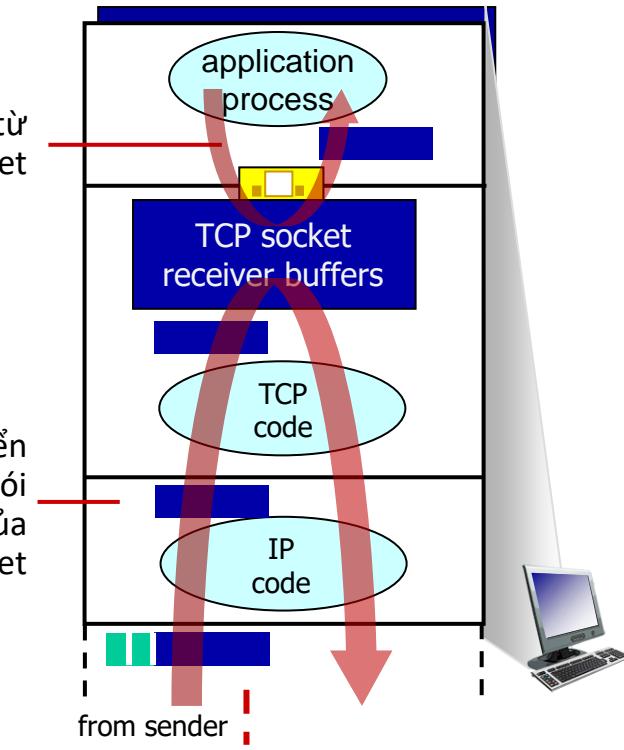
Điều khiển luồng của TCP

Q: Chuyện gì xảy ra nếu tầng mạng chuyển dữ liệu lớn hơn khả năng lấy dữ liệu của tầng ứng dụng?



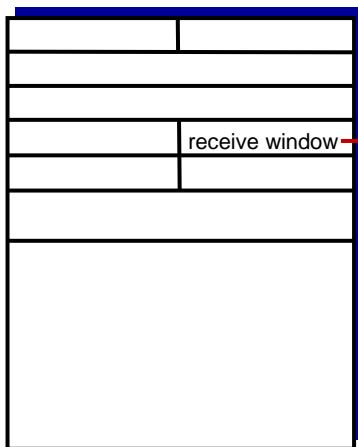
Ứng dụng lấy dữ liệu từ bộ đệm của TCP socket

Tầng mạng chuyển phần dữ liệu của gói tin IP lên bộ đệm của TCP socket



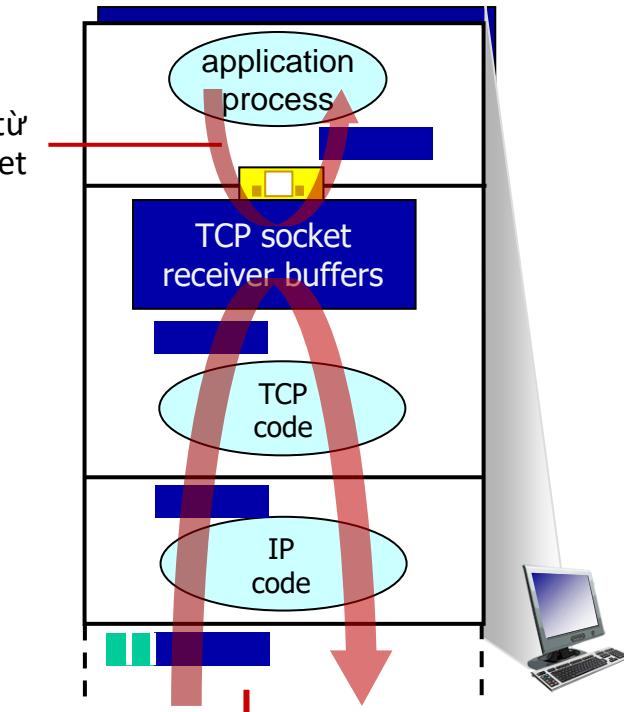
Điều khiển luồng của TCP

Q: Chuyện gì xảy ra nếu tầng mạng chuyển dữ liệu lớn hơn khả năng lấy dữ liệu của tầng ứng dụng?



flow control: # bytes máy gửi muốn nhận

Ứng dụng lấy dữ liệu từ bộ đệm của TCP socket



protocol stack của máy nhận

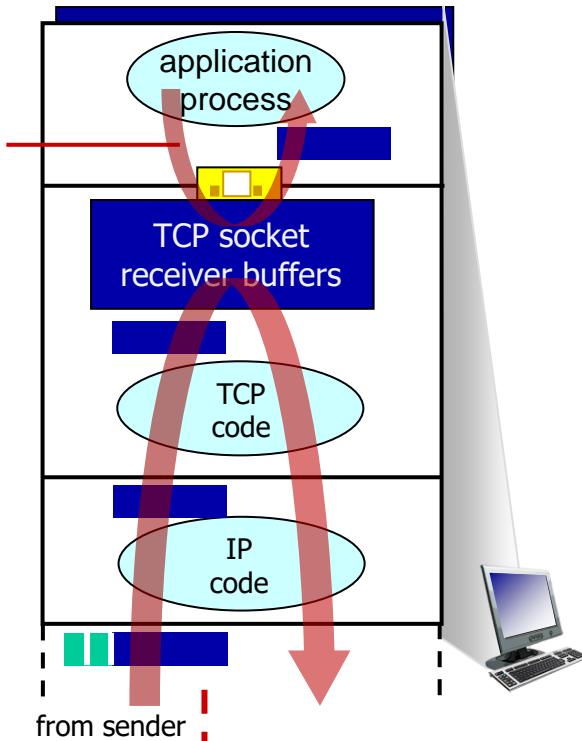
TCP flow control

Q: Chuyện gì xảy ra nếu tầng mạng chuyển dữ liệu lớn hơn khả năng lấy dữ liệu của tầng ứng dụng?

Điều khiển luồng

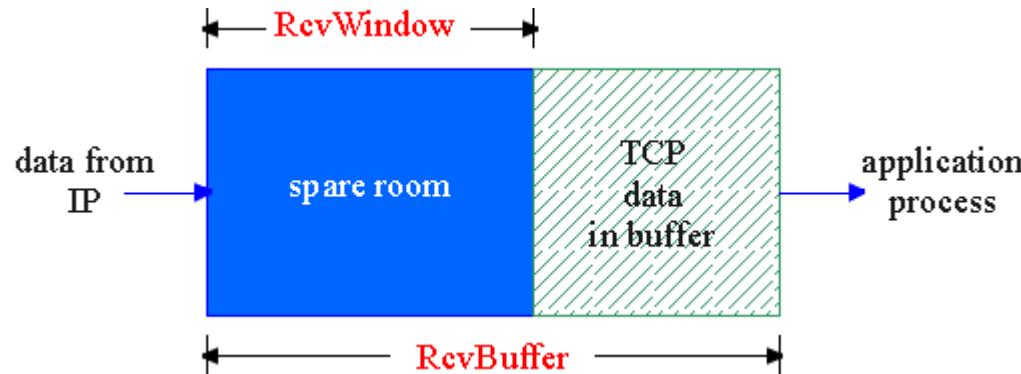
Máy nhận điều khiển máy gửi, để máy gửi không gửi quá khả năng nhận của máy nhận

Ứng dụng lấy dữ liệu từ bộ đệm của TCP socket



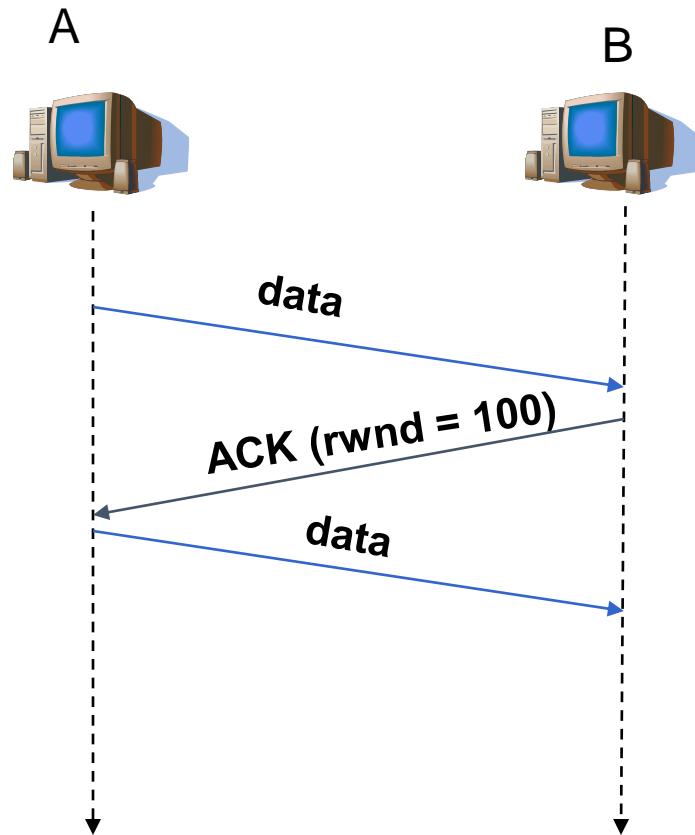
protocol stack của máy nhận

Kiểm soát luồng trong TCP



- Kích thước vùng đệm trống
 - = Rwnd
 - = RcvBuffer - [LastByteRcvd - LastByteRead]

Trao đổi thông tin về Rwnd



- Bên nhận sẽ báo cho bên gửi biết Rwnd trong các đoạn tin
- Bên gửi đặt kích thước cửa sổ gửi theo Rwnd



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Điều khiển tắc nghẽn trong TCP

Nguyên lý kiểm soát tắc nghẽn

Tắc nghẽn:

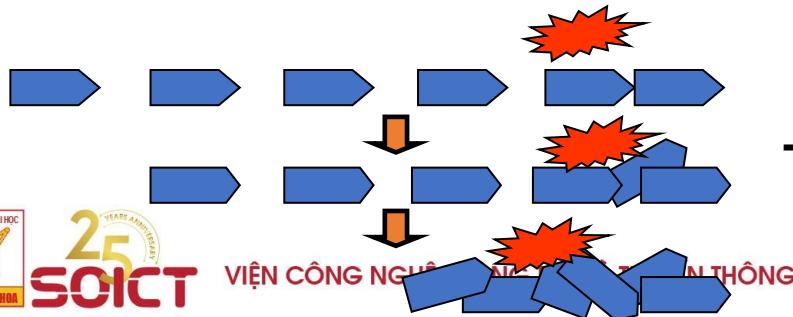
- Không chính thức: “quá nhiều nguồn gửi quá nhiều dữ liệu quá nhanh để **mạng** có thể xử lý”
- Hậu quả:
 - Thông lượng giảm, độ trễ tăng (hàng đợi ở router)
 - Mất gói (quá khả năng bộ đệm của router)
- Vấn đề khác kiểm soát luồng!
- top-10 vấn đề mạng!



Tắc nghẽn: quá
nhiều máy gửi, gửi quá
nhanh



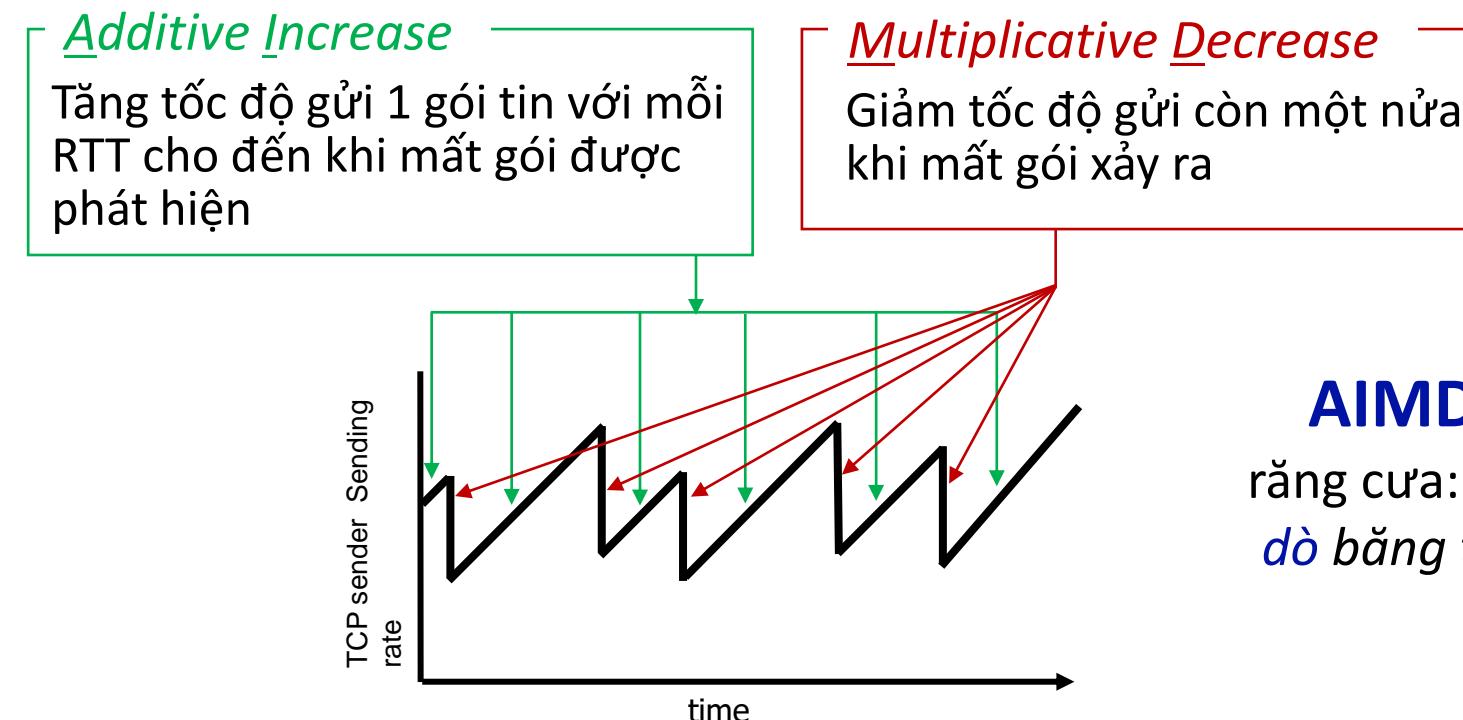
Kiểm soát luồng: một
máy gửi cho 1 máy nhận



**Tắc nghẽn
xảy ra**

Điều khiển tắc nghẽn của TCP: AIMD

- **Cách tiếp cận:** máy gửi tăng tốc độ gửi cho đến khi mất gói xảy ra (tắc nghẽn), sau đó giảm tốc độ gửi khi xảy ra tắc nghẽn



TCP AIMD (tiếp)

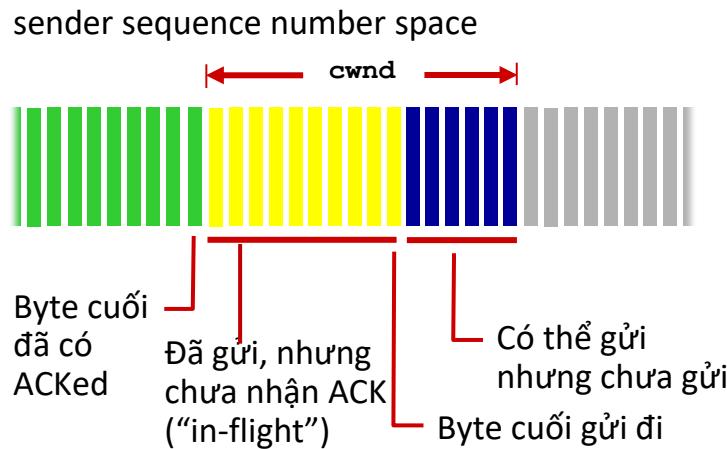
Chi tiết Giảm cấp số nhân (Multiplicative decrease) : tốc độ gửi sẽ

- Giảm một nửa khi nhận được 3 gói ACK trùng lặp (TCP Reno)
- Giảm về 1 MSS (maximum segment size) khi timeout (TCP Tahoe)

Tại sao dùng AIMD?

- AIMD – thuật toán phân tán, bất đồng bộ – giúp :
 - Tối ưu tốc độ luồng trên diện rộng
 - Có những thuộc tính ổn định

Kiểm soát tắc nghẽn TCP: chi tiết



Hành vi gửi TCP:

- Gửi cwnd bytes, chờ RTT các ACK, rồi gửi nhiều byte hơn

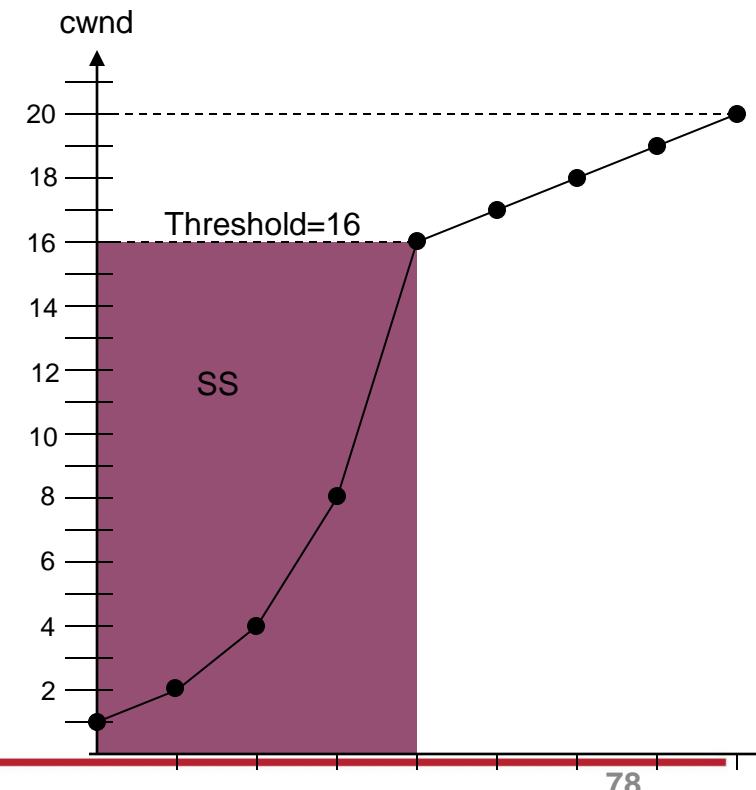
$$\text{Tốc độ TCP} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/s}$$

- Giới hạn gửi của TCP:
- cwnd thay đổi động tùy thuộc vào mức độ tắc nghẽn của mạng (thực thi điều khiển tắc nghẽn TCP)

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$$

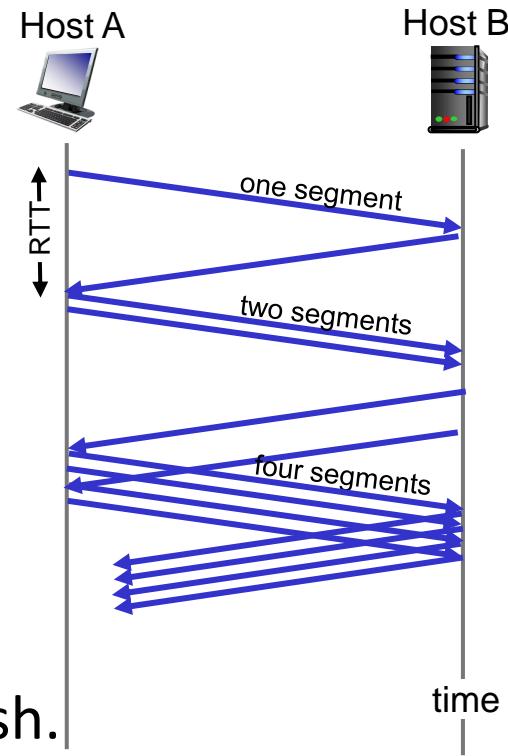
Nguyên lý kiểm soát tắc nghẽn

- Slow-start
 - Tăng tốc độ theo hàm số mũ
 - Tiếp tục tăng đến một ngưỡng nào đó
- Tránh tắc nghẽn
 - Tăng dần tốc độ theo hàm tuyến tính cho đến khi phát hiện tắc nghẽn
- Phát hiện tắc nghẽn
 - Nếu gói tin bị mất



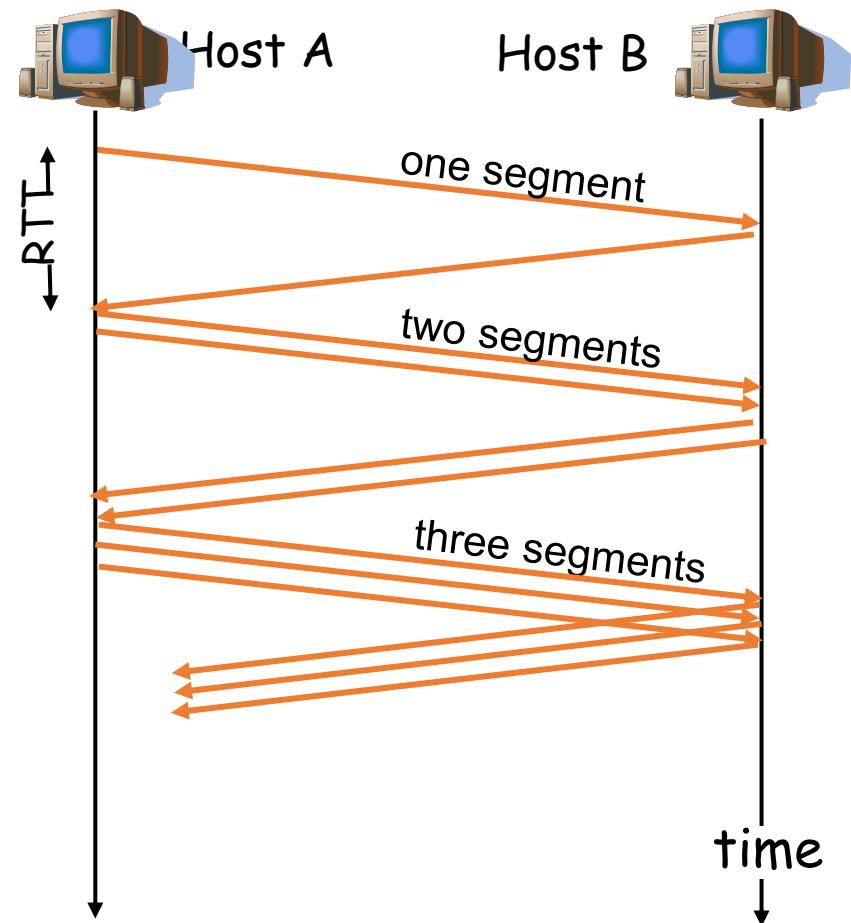
TCP slow start (khởi động chậm)

- Khi bắt đầu kết nối, tăng tốc độ gửi theo cấp số nhân đến khi mất gói hay đến ngưỡng :
 - Bắt đầu **cwnd** = 1 MSS
 - Gấp đôi **cwnd** mỗi RTT
 - Thực hiện tăng cwnd cho mỗi ACK nhận được
- **Tổng kết:** bắt đầu chậm nhưng tăng tốc nhanh
- Tăng cho đến một ngưỡng: ssthresh. Sau đó, TCP chuyển sang trạng thái tránh tắc nghẽn



Tránh tắc nghẽn - Congestion avoidance

- Sau mỗi RTT tăng cwnd thêm 1 MSS
- ➔ Tăng cwnd theo cấp số cộng sau khi nó đạt tới ssthresh



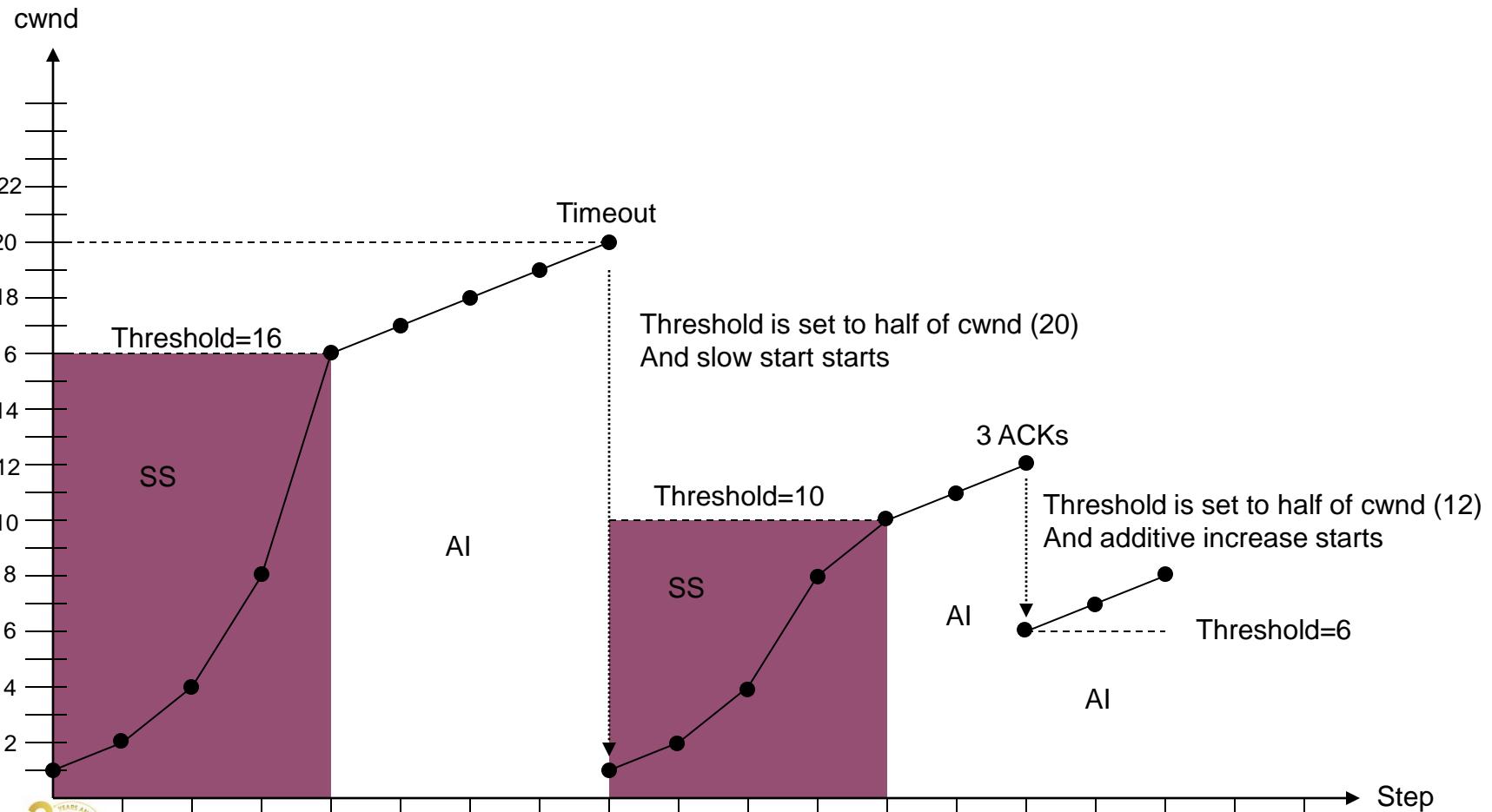
Phản ứng của TCP (1)

- Giảm tốc độ gửi
- Phát hiện tắc nghẽn?
 - Nếu như phải truyền lại
 - Có thể suy ra là mạng “tắc nghẽn”
- Khi nào thì phải truyền lại?
 - Timeout!
 - Cùng một gói tin số hiệu gói tin trong ACK

Phản ứng của TCP (2)

- Khi có timeout của bên gửi
 - TCP đặt ngưỡng ssthresh xuống còn một nửa giá trị hiện tại của cwnd
 - TCP đặt cwnd về 1 MSS
 - TCP chuyển về slow start
- Nếu nhận được 3 ACK giống nhau
 - TCP đặt ngưỡng xuống còn một nửa giá trị hiện tại của cwnd
 - TCP chuyển trạng thái “congestion avoidance”

Kiểm soát tắc nghẽn – minh họa



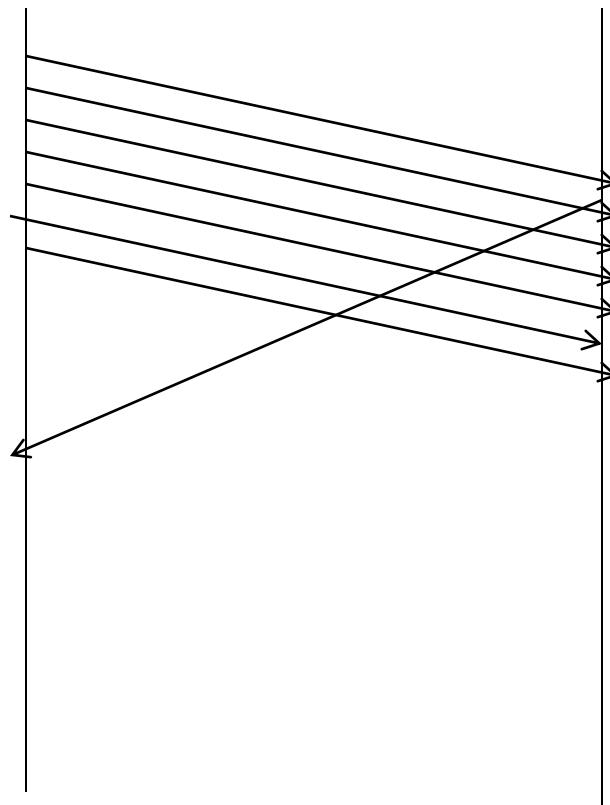
Tổng kết

- Còn rất nhiều chi tiết về TCP!
- Có hai dạng giao thức giao vận
 - UDP và TCP
 - Best effort vs. reliable transport protocol
- Các cơ chế bảo đảm độ tin cậy
 - Báo nhận
 - Truyền lại
 - Kiểm soát luồng và kiểm soát tắc nghẽn

Bài tập

- Giả sử cần truyền 1 file
 - Kích thước $O = 100\text{KB}$ trên kết nối TCP
 - S là kích thước mỗi gói TCP, $S = 536 \text{ byte}$
 - $RTT = 3 \text{ ms.}$
- Giả sử cửa sổ nhận của TCP là cố định với kích thước W .
 - Thời gian truyền với phương pháp Stop-and-wait
 - Thời gian truyền với phương pháp cửa sổ trượt với kích thước cửa sổ =7
 - Hỏi kích thước cửa sổ nhận là bao nhiêu để thời gian chờ đợi ít nhất? Nếu tốc độ đường truyền là
 - $R = 20 \text{ Mbit/s}; R= 100 \text{ Mbits/s.}$

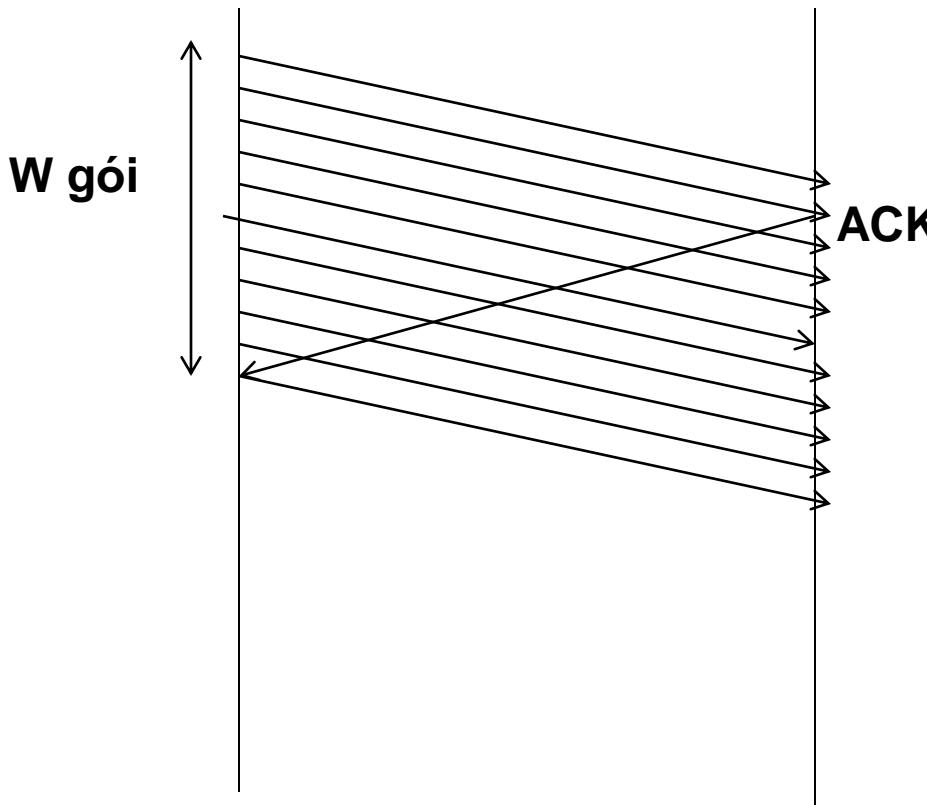
Cơ chế cửa sổ trượt



Thời gian truyền với cửa sổ 7

- T truyền nhanh nhất = (T phát 7 gói + chờ) *số lần.
- 1 lần Chờ = (T phát 1 gói + RTT) – T phát 7 gói
- Số lần chờ = số gói /7

Thời gian truyền nhanh nhất với cửa sổ trượt



- Truyền nhanh nhất đạt được khi nguồn phát xong gói cuối của cửa sổ thì đã nhận được ACK của gói đầu tiên.
- Kích thước cửa sổ W
- $T_{phát} (W \text{ gói}) \geq T_{phát \text{ gói đầu}} + RTT$

Thời gian truyền nhanh nhất với cửa sổ trượt (cont.)

- $T_{\text{phát}} (\text{W gói}) = W * S/R$
- Phát ngắn nhất khi không phải chờ:
- $\Rightarrow (W-1)*S/R \geq RTT$
- $\Rightarrow W \geq RTT * R/S + 1$
- Thời gian phát hết dữ liệu $L = L/R + RTT$
- $R=100 \text{ Mbps}$
 - $W \geq 3\text{ms} * 100 \text{ Mbps} / (536*8) + 1$

Tuần tới: Application Layer

- Application service model
 - Client-server vs. P2P
- Typical applications and protocols
 - HTTP
 - Mail
 - FTP
 - P2P file sharing
 -
 - and your applications?

Acknowledgment

- Bài giảng có sử dụng các hình vẽ từ
 - Tài liệu của trường đại học Keio và Ritsumeikan
 - Tài liệu “Computer Network, a top down approach” của J.F Kurose và K.W. Ross