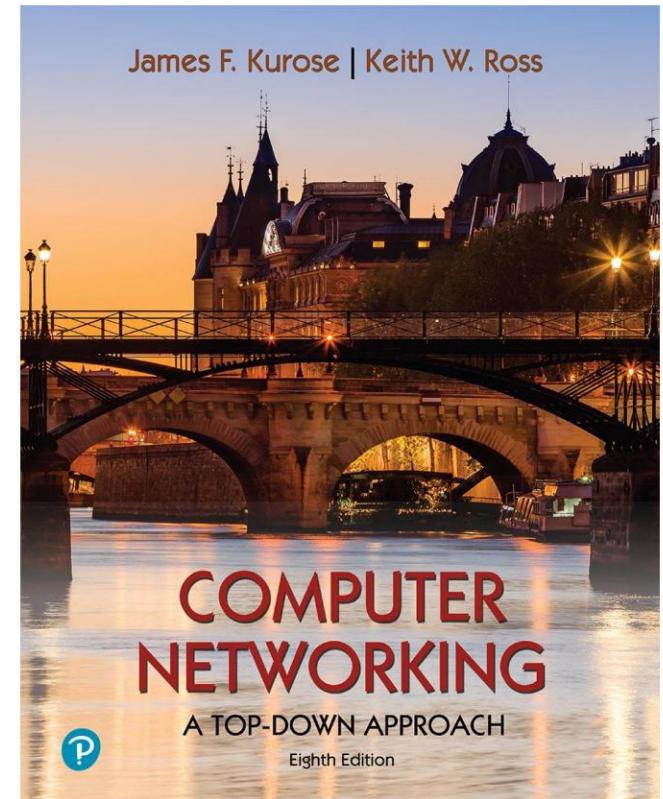


Chương 3

Lớp vận chuyển



Mạng máy tính: A
Cách tiếp cận từ trên xuống
phiên bản thứ 8
Jim KuroseKeith Ross
Pearson, 2020

Lớp vận chuyển: tổng quan

Mục tiêu của

chúng tôi: hiểu các nguyên tắc đằng sau các dịch vụ **tầng vận chuyển** :

- ghép kênh,
- tách kênh •

truyền dữ liệu đáng tin cậy • kiểm soát luồng • kiểm soát tắc nghẽn

tìm hiểu về các giao thức lớp vận chuyển Internet : • **UDP**: vận chuyển không kết nối • **TCP**: vận chuyển đáng tin cậy hướng kết nối • Kiểm soát tắc nghẽn TCP

Lớp vận chuyển: lộ trình

Dịch vụ tầng vận chuyển

Ghép kênh và phân kênh Vận

chuyển không kết nối: UDP Nguyên

tắc truyền dữ liệu đáng tin cậy

Truyền tải hướng kết nối: TCP Nguyên

tắc kiểm soát tắc nghẽn Kiểm soát

tắc nghẽn TCP Sự phát triển của

chức năng tầng vận chuyển



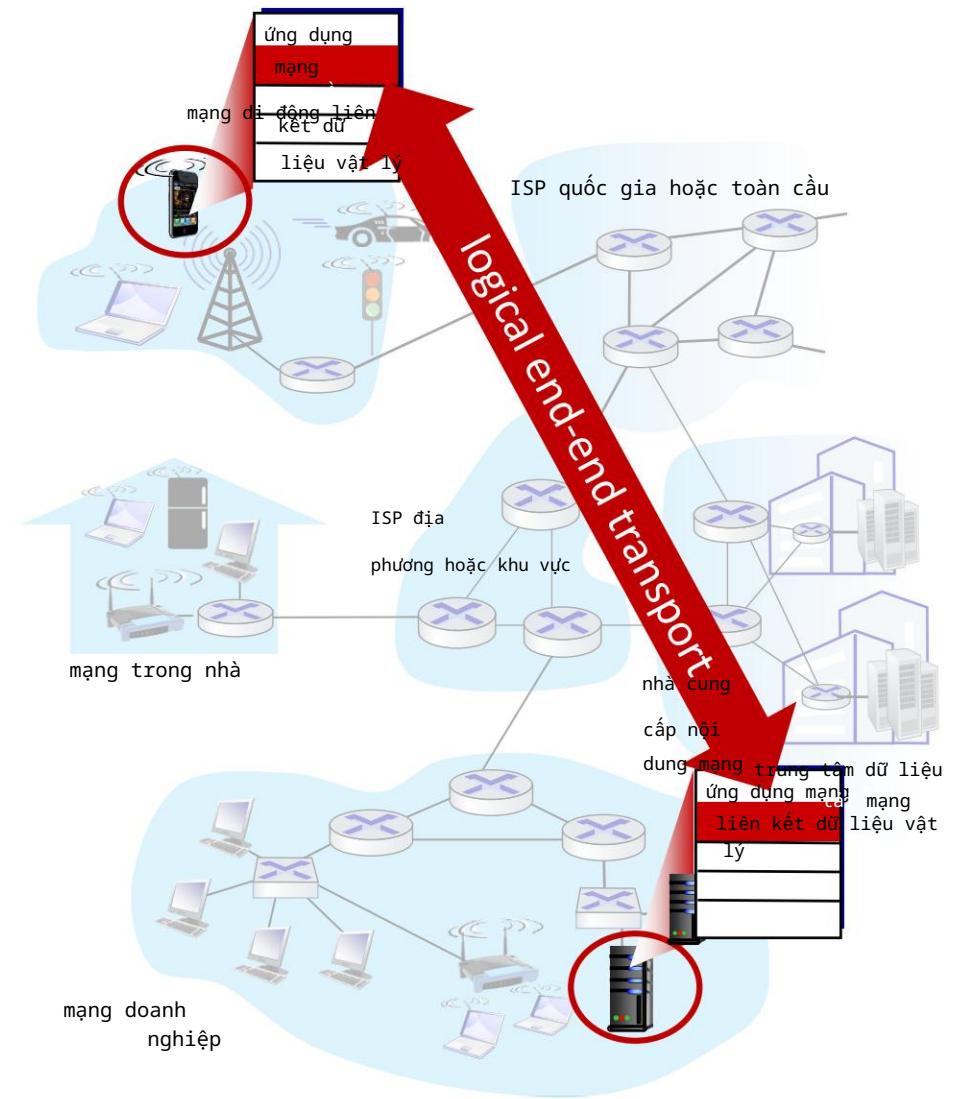
Các dịch vụ và giao thức vận chuyển

cung cấp **giao tiếp hợp lý** giữa các quy trình ứng dụng đang chạy trên các máy chủ khác nhau

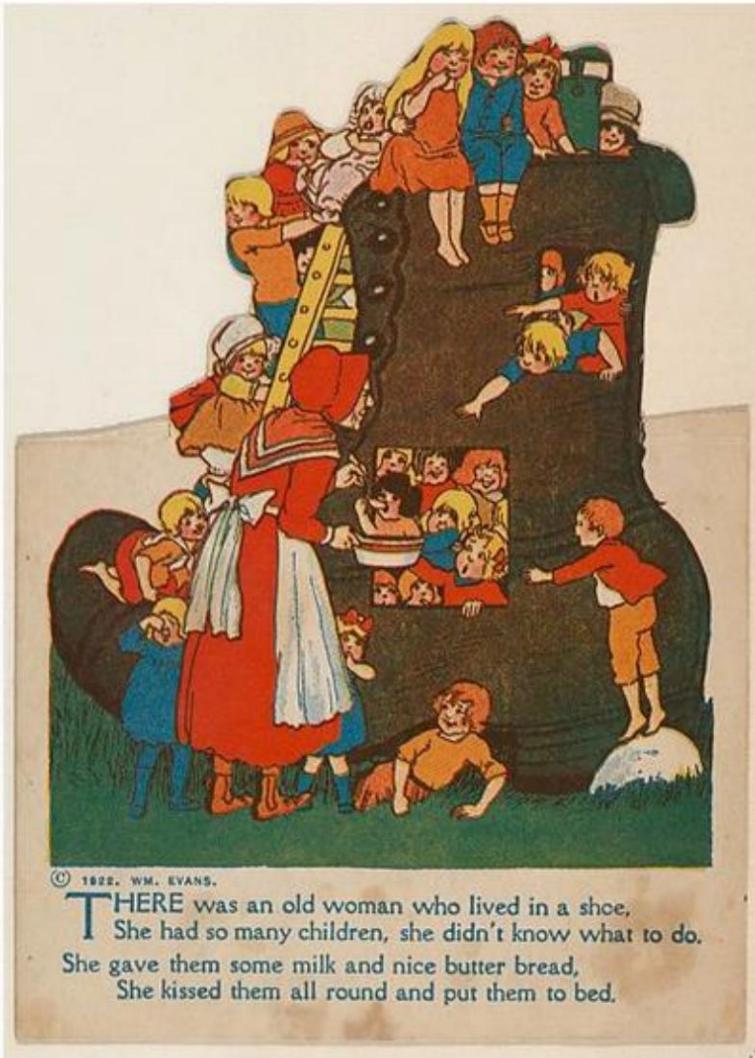
các **hành động của** giao thức truyền tải trong hệ thống đầu cuối : • **người gửi**: ngắt các thông báo của ứng dụng

- thành **các phân đoạn**, chuyển đến lớp mạng
- **người nhận**: tập hợp lại các phân đoạn thành **tin nhắn**, chuyển đến lớp ứng dụng

hai giao **thức** vận chuyển có sẵn để
Ứng dụng Internet • **TCP**,
UDP



Vận chuyển so với các dịch vụ và giao thức lớp mạng



phép loại suy **trong**

gia đình: 12 đứa trẻ nhà Ann
gửi thư cho 12 đứa trẻ nhà
Bill: **chủ nhà** = nhà

quy trình = **tin**

nhắn ứng dụng dành cho trẻ
em = thư trong phong bì

giao thức vận chuyển = Ann và Bill
chuyển giao cho anh chị em trong nhà

giao thức lớp mạng = dịch vụ bưu chính

Vận chuyển so với các dịch vụ và giao thức lớp mạng

lớp mạng: giao tiếp logic giữa các máy chủ

lớp vận chuyển: giao tiếp logic giữa các quy trình • dựa vào, nâng cao, các dịch vụ lớp mạng

phép loại suy trong
gia đình: 12 đứa trẻ nhà Ann
gửi thư cho 12 đứa trẻ nhà Bill:
chủ nhà = nhà

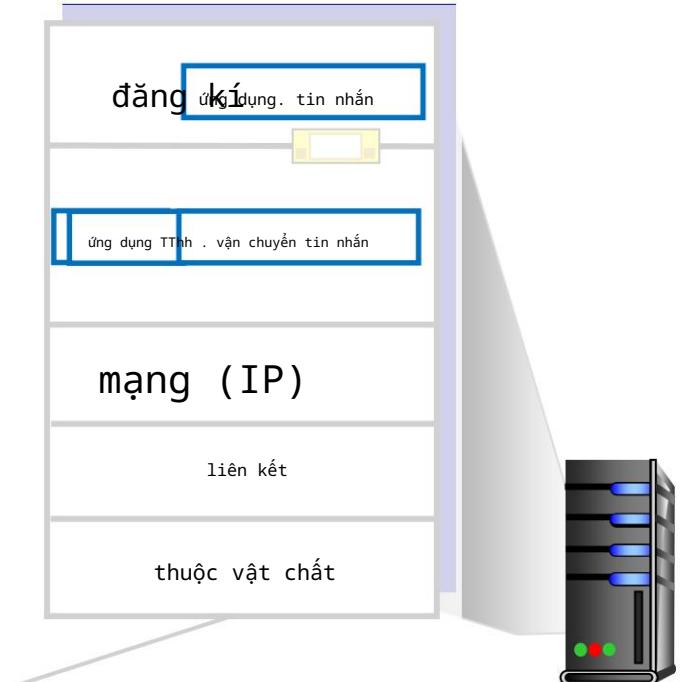
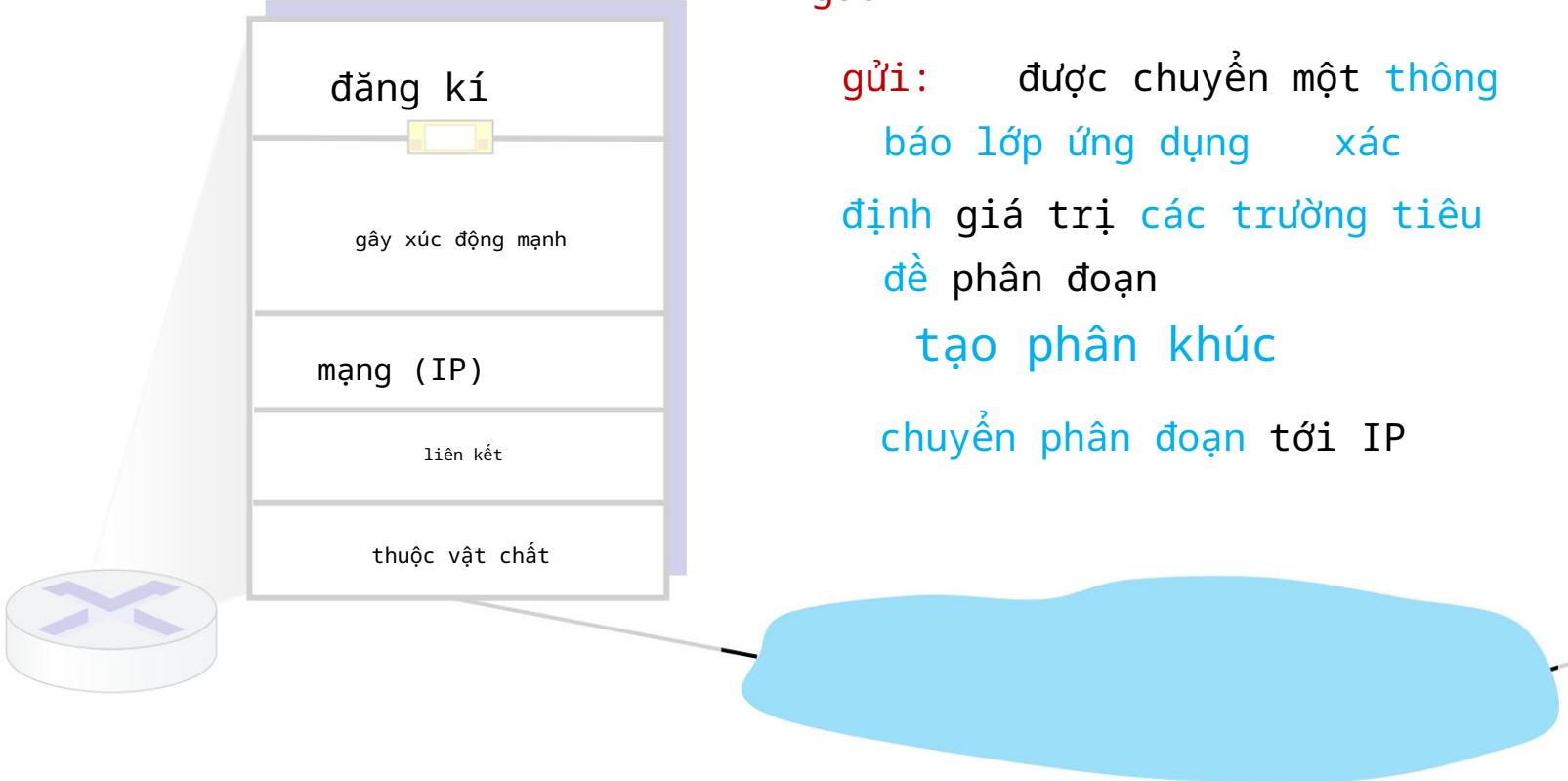
quy trình = tin
nhắn ứng dụng dành cho trẻ em = thư trong phong bì
giao thức vận chuyển = Ann và Bill chuyển giao cho anh chị em trong nhà
giao thức lớp mạng = dịch vụ bưu chính

Hành động tầng vận chuyển

Người

đăng kí
gây xúc động mạnh
mạng (IP)
liên kết
thuộc vật chất

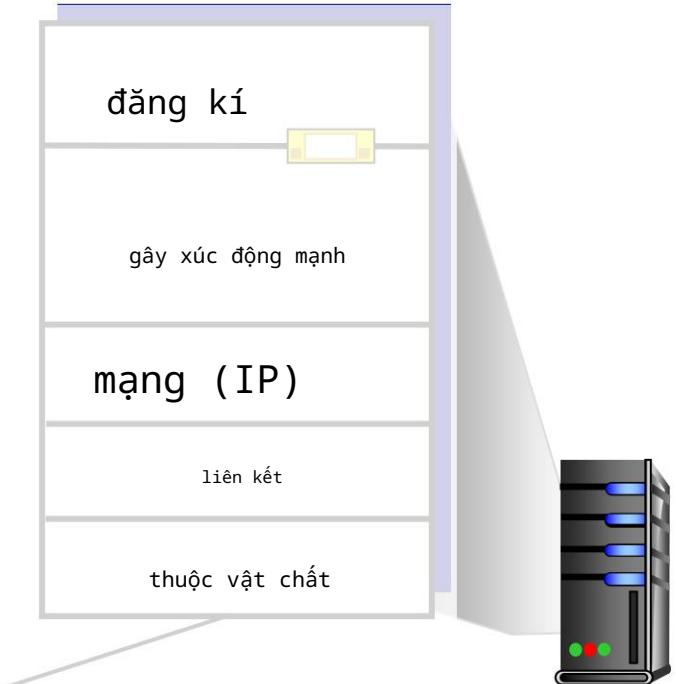
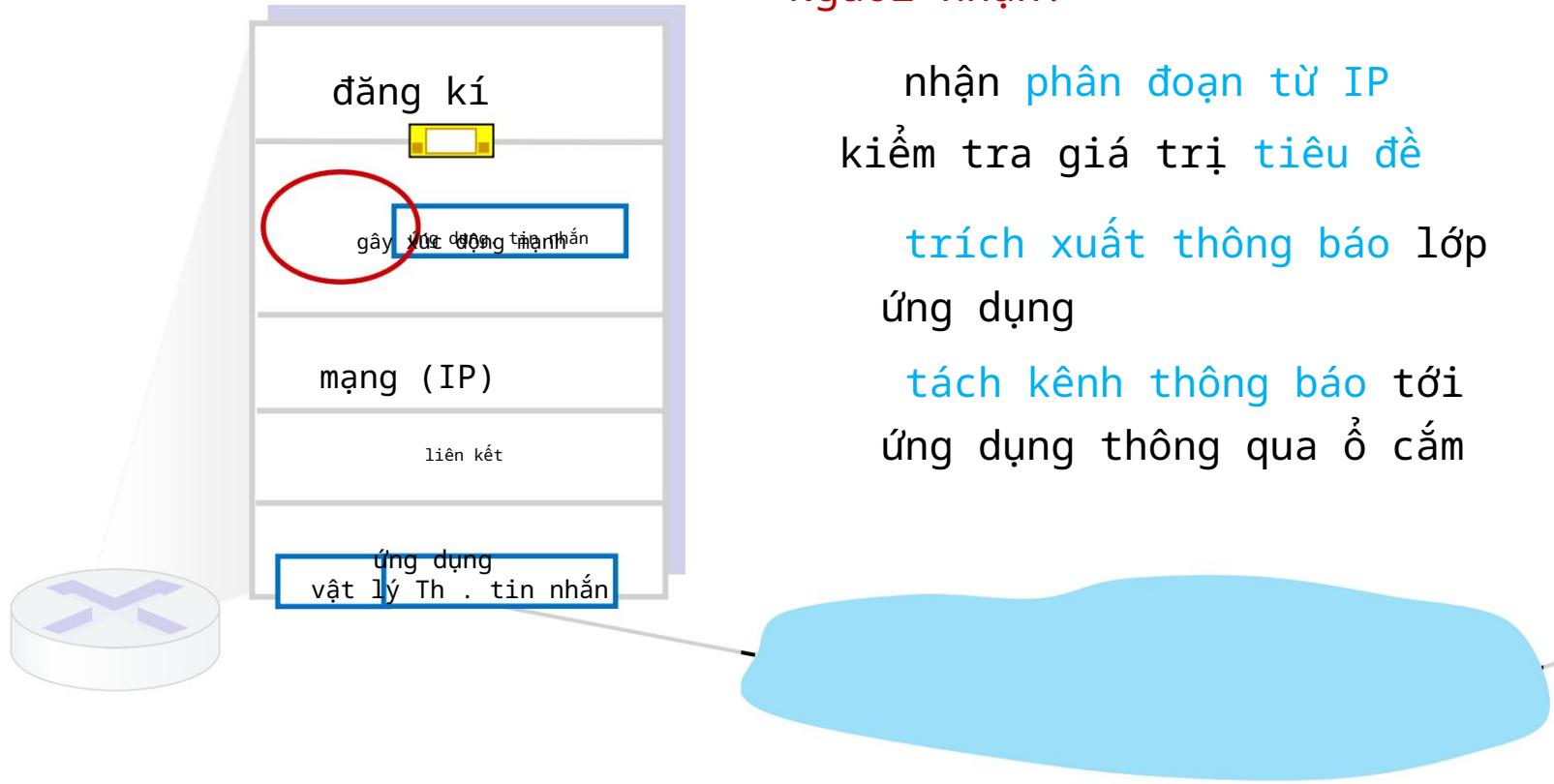
gửi: được chuyển một thông báo lớp ứng dụng xác định giá trị các trường tiêu đề phân đoạn
tạo phân khúc
chuyển phân đoạn tới IP



Hành động tầng vận chuyển

Người nhận:

nhận phân đoạn từ IP
kiểm tra giá trị **tiêu đề**
trích xuất thông báo lớp
ứng dụng
tách kênh thông báo tới
ứng dụng thông qua ổ cắm

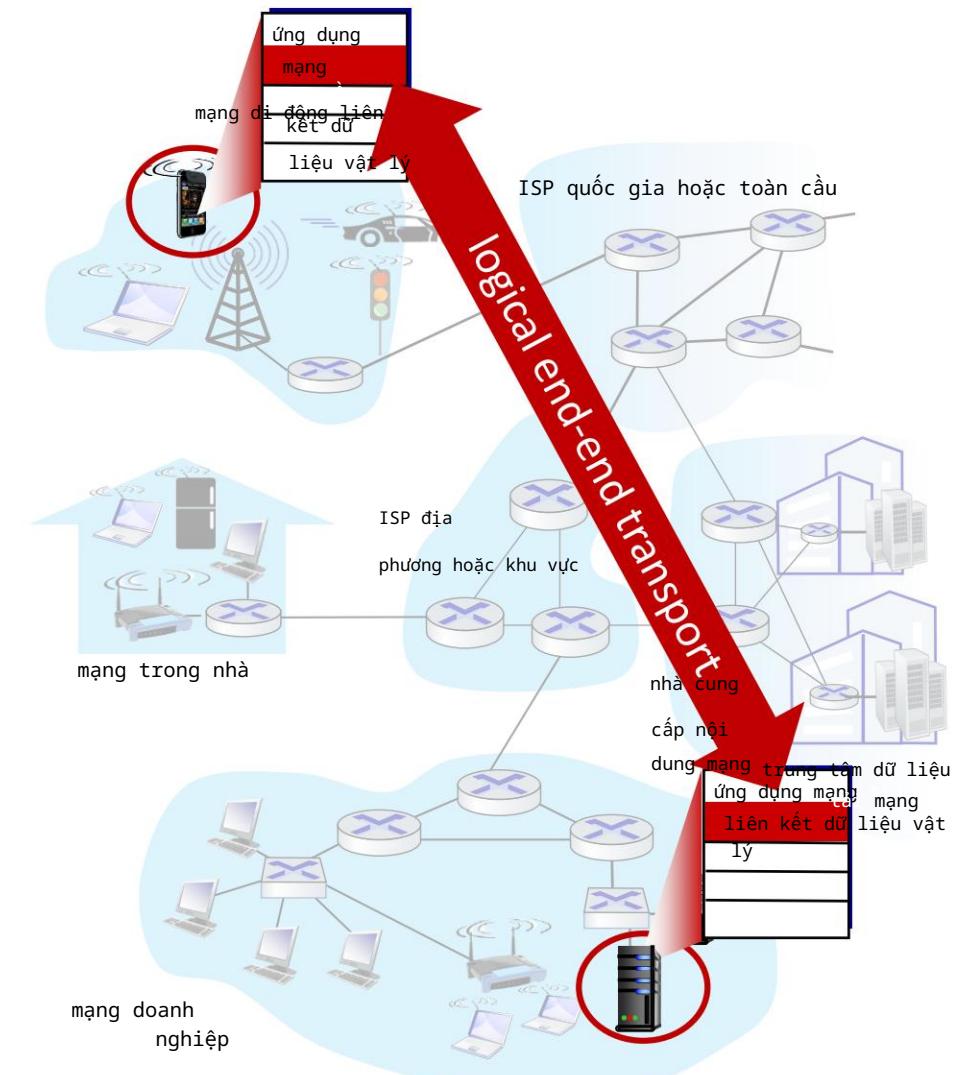


Hai giao thức vận chuyển Internet chính

TCP: Giao thức điều khiển truyền dẫn

- giao hàng đúng thứ tự, đáng tin cậy • kiểm soát tắc nghẽn • kiểm soát luồng
- thiết lập kết nối

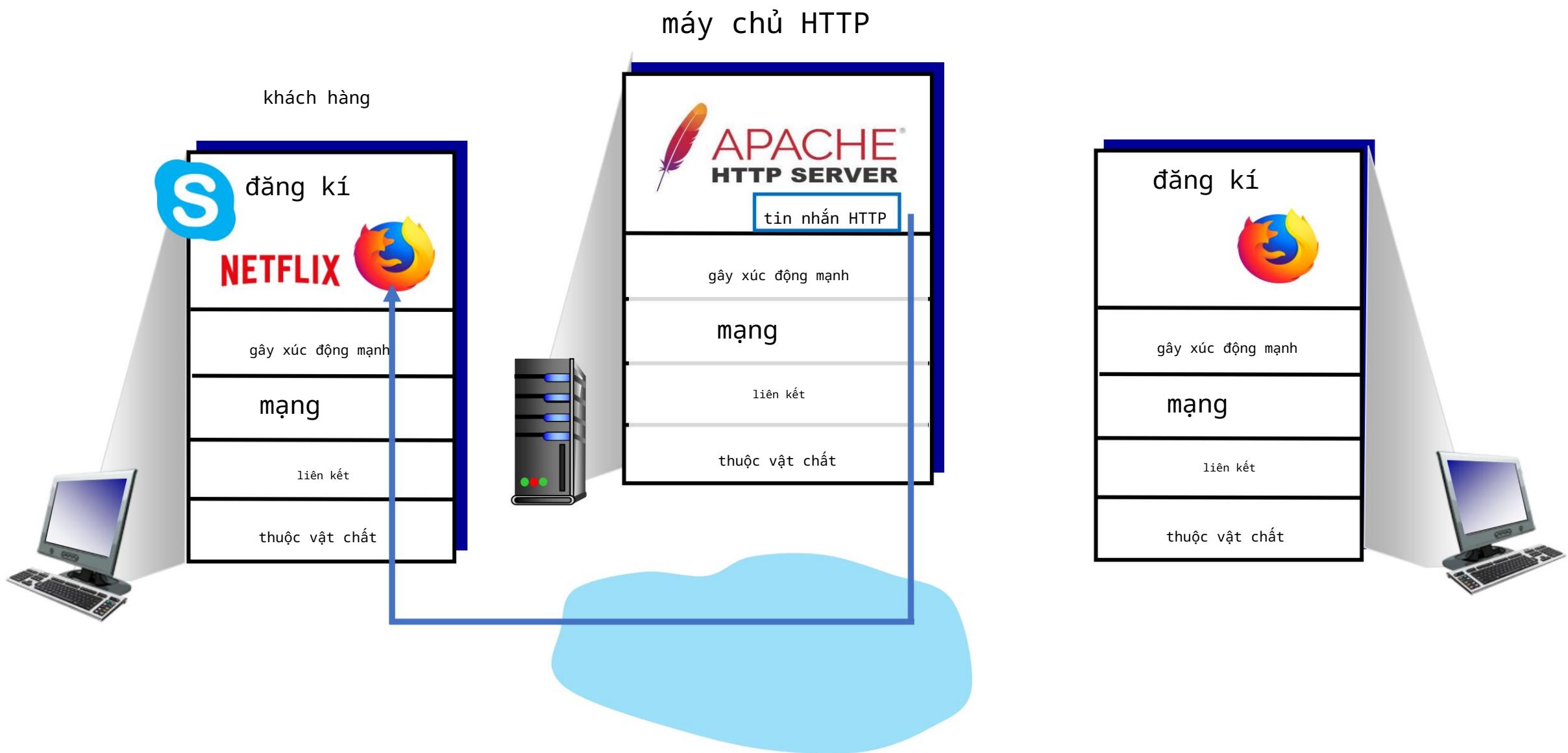
UDP: Giao thức gói dữ liệu người dùng • không đáng tin cậy, phân phối không theo thứ tự • phần mở rộng không rườm rà của IP “nỗ lực cao nhất”
các dịch vụ không khả dụng: • đảm bảo độ trễ • đảm bảo băng thông

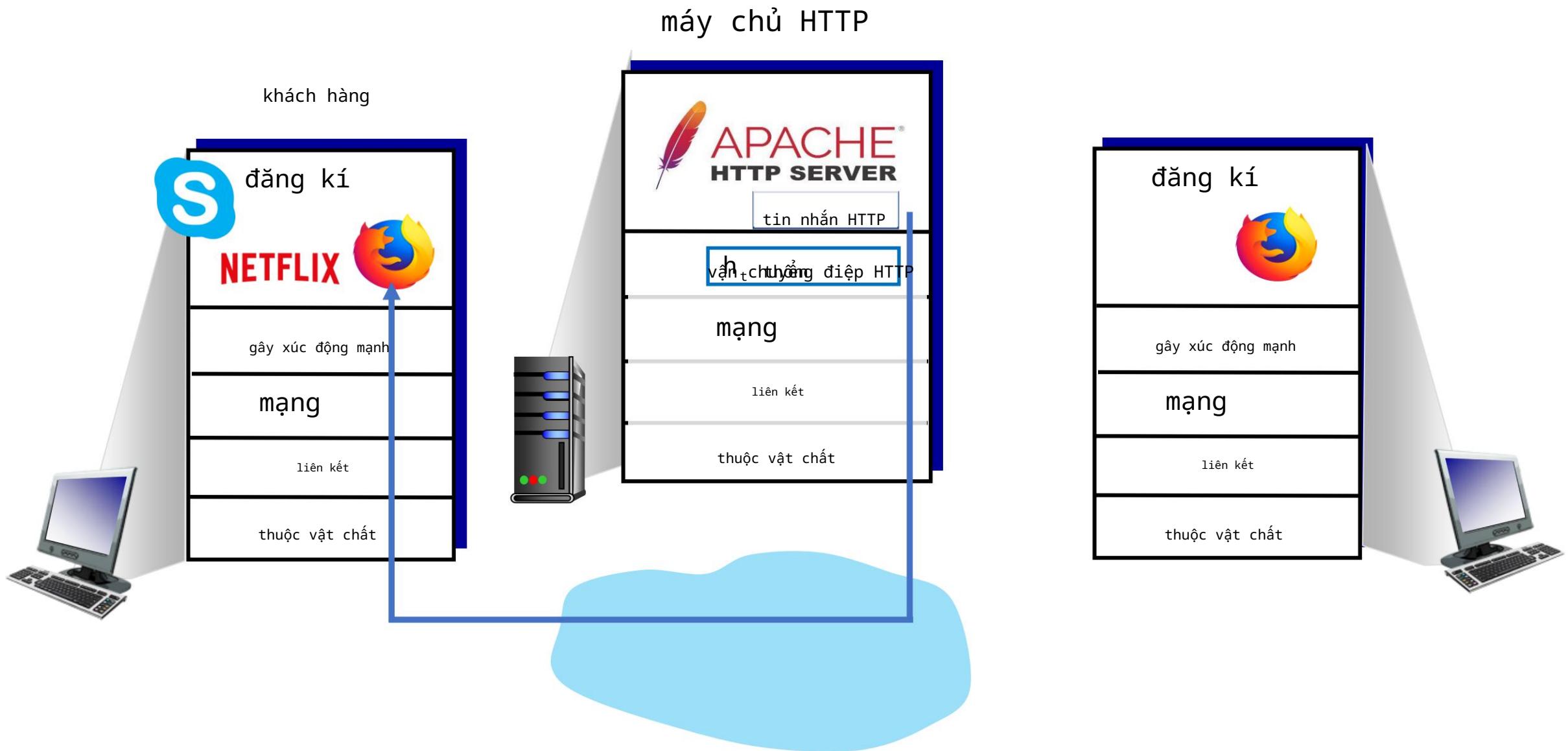


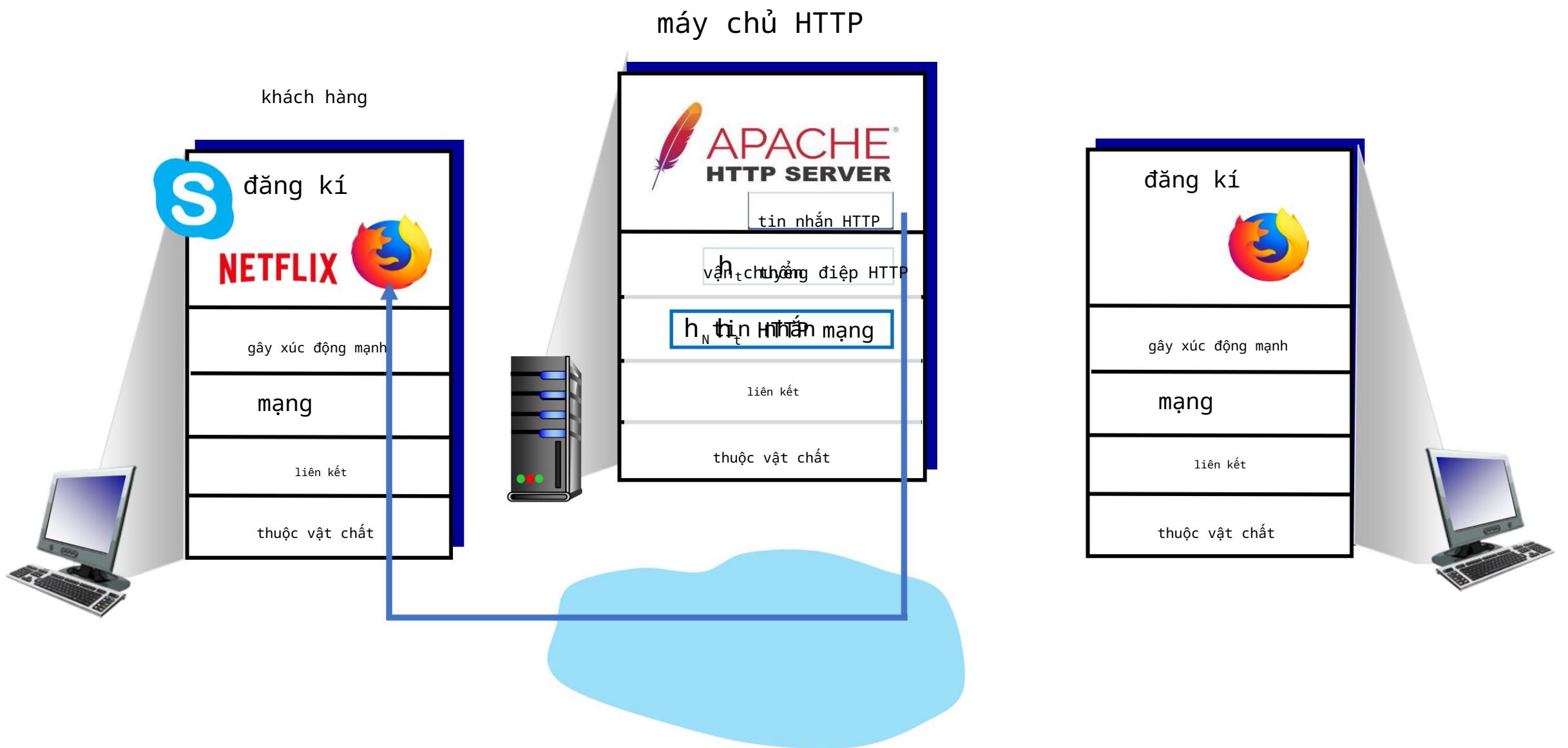
Chương 3: lộ trình

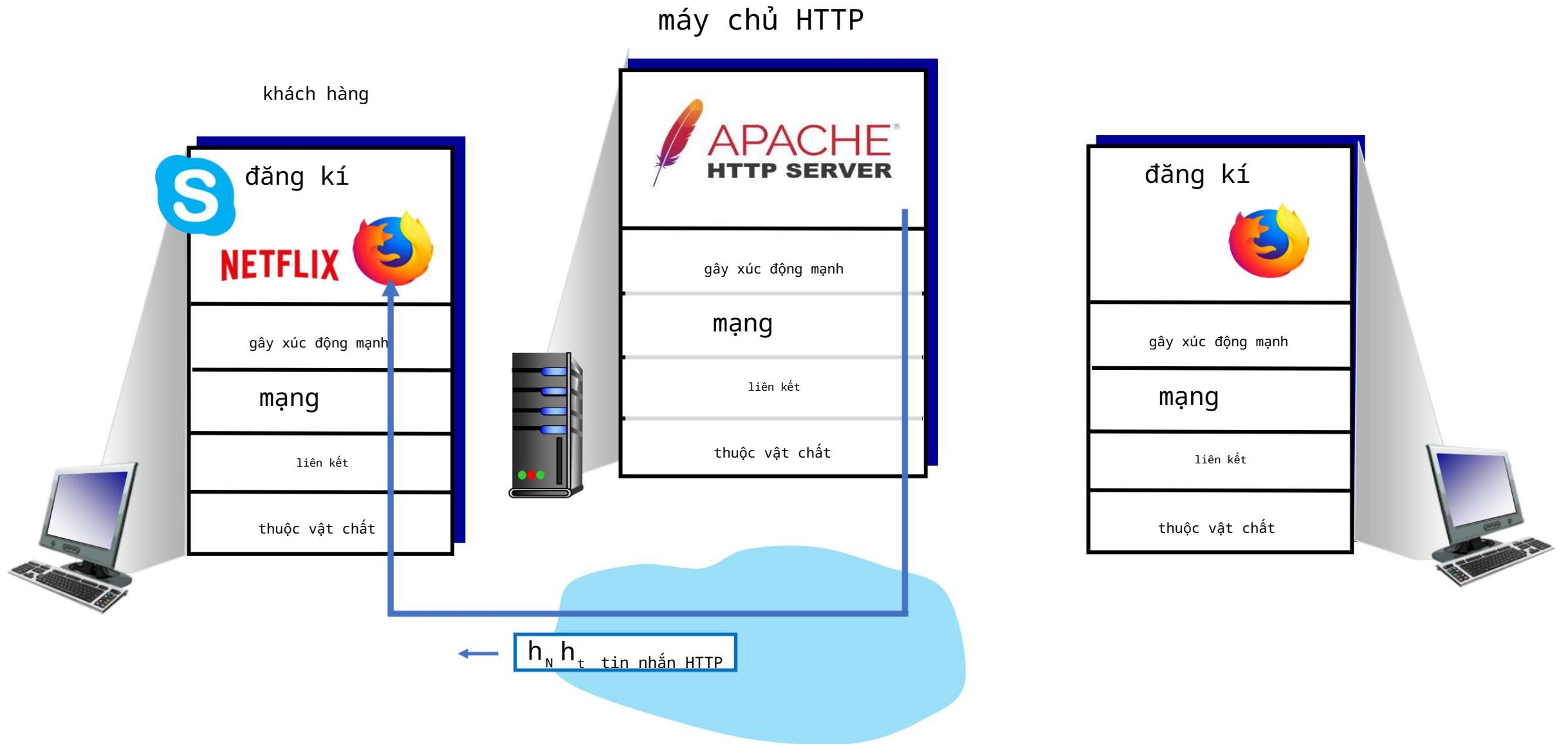
Dịch vụ tầng vận chuyển
Ghép kênh và phân kênh Vận
chuyển không kết nối: UDP Nguyên
tắc truyền dữ liệu đáng tin cậy
Truyền tải hướng kết nối: TCP
Nguyên tắc kiểm soát tắc nghẽn Kiểm
soát tắc nghẽn TCP Sự phát triển
của chức năng tầng vận chuyển

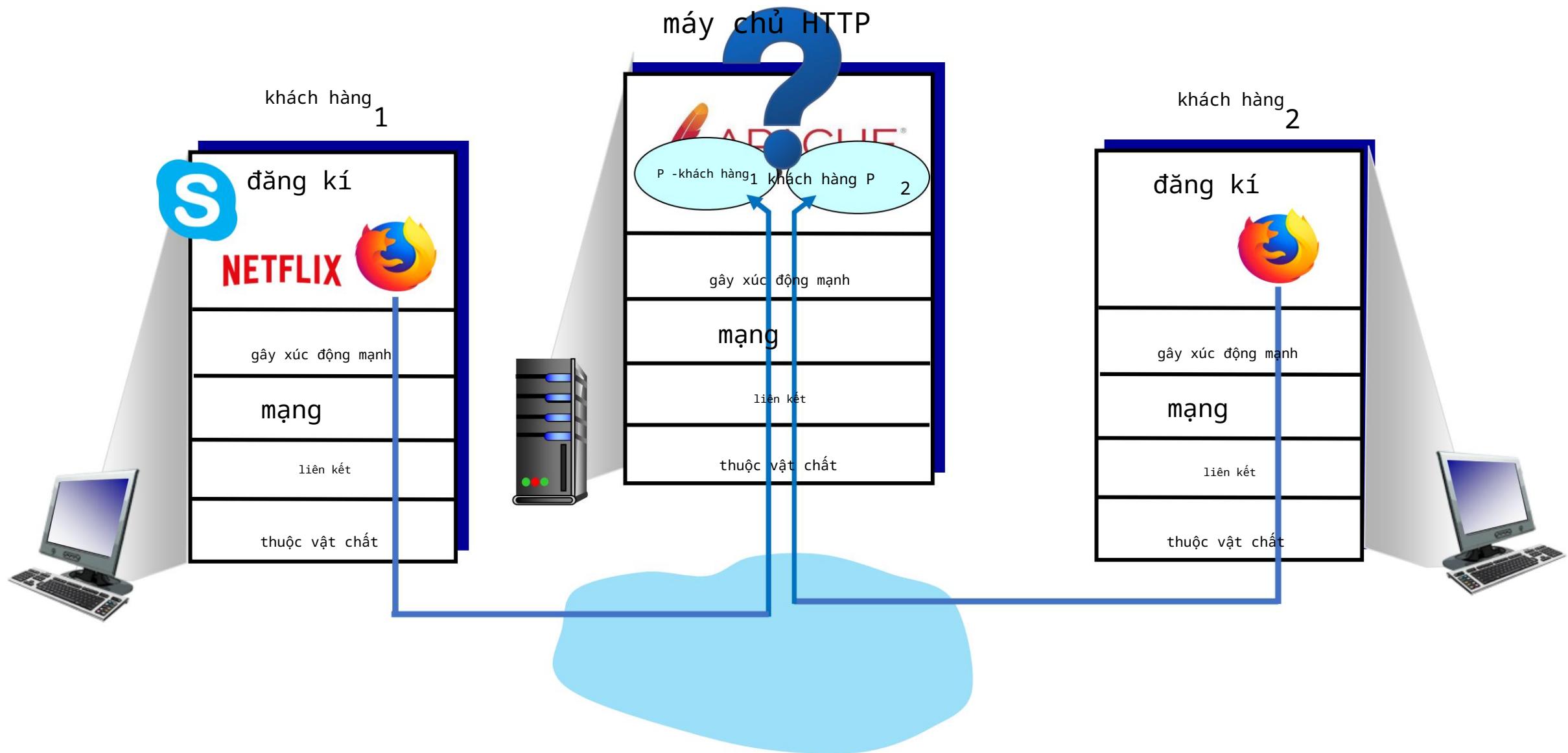












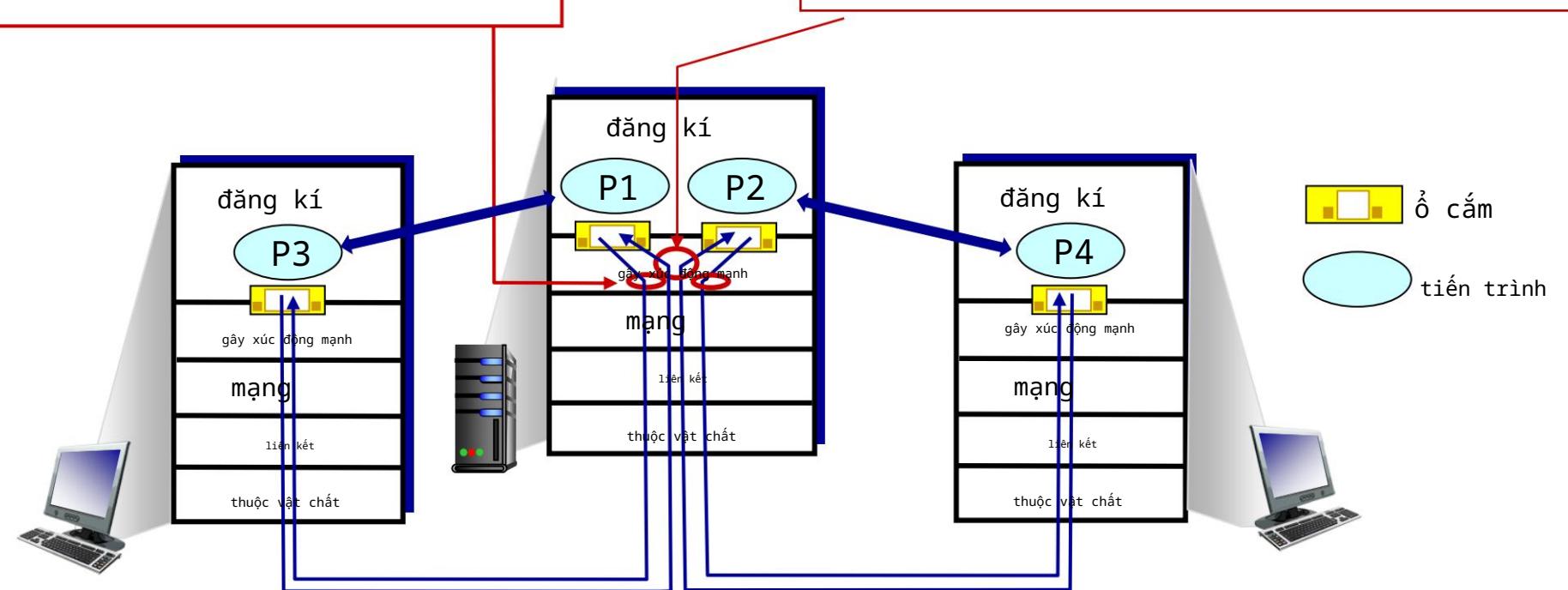
Ghép kênh/tách kênh

ghép kênh tại người gửi:

xử lý dữ liệu từ **nhiều ổ cảm**,
thêm tiêu đề vận chuyển (sau này
được sử dụng để tách kênh)

tách kênh tại máy thu:

sử dụng thông tin tiêu đề để
phân phối các phân đoạn đã nhận **đến**
đúng ổ cảm



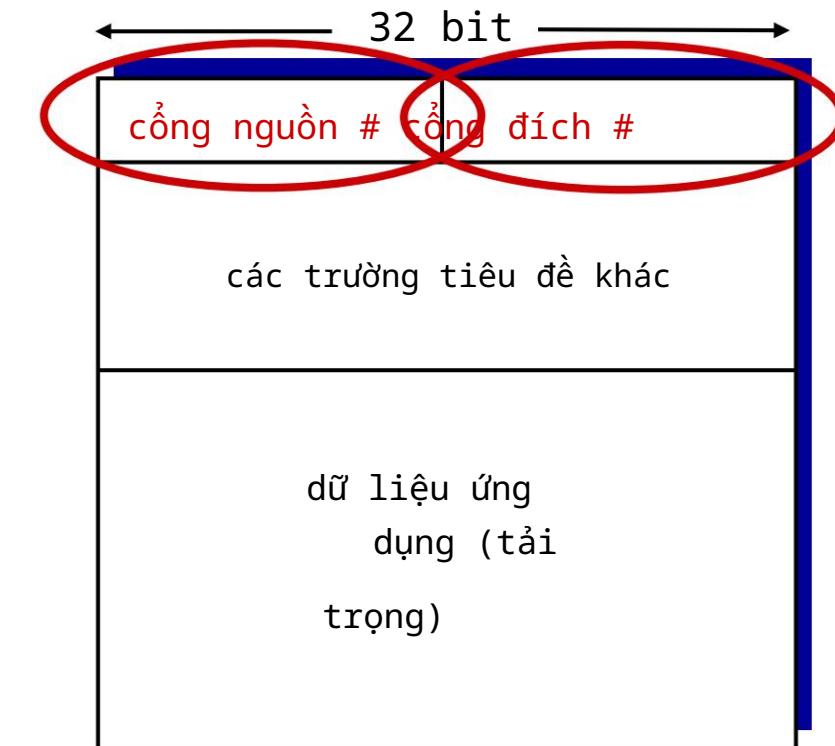
Cách thức hoạt động của phân kênh

máy chủ nhận IP datagram

- mỗi datagram có nguồn IP địa chỉ, địa chỉ IP đích
- mỗi datagram mang một phân đoạn tầng vận

chuyển • mỗi phân đoạn có số cổng nguồn, đích

Máy chủ sử dụng địa chỉ IP & số cổng để hướng phân đoạn tới ổ cắm thích hợp



Định dạng phân đoạn TCP/UDP

tách kênh không kết nối

Nhớ lại:

khi tạo **socket**, host
phải chỉ định **cổng Host-local #**:

```
DatagramSocket mySocket1 =  
DatagramSocket mới(12534);
```

khi tạo **datagram** để gửi
vào UDP socket, host phải
chỉ định • **địa chỉ IP** đích

- **cổng** đích #

khi nhận, máy chủ nhận

Phân đoạn UDP :

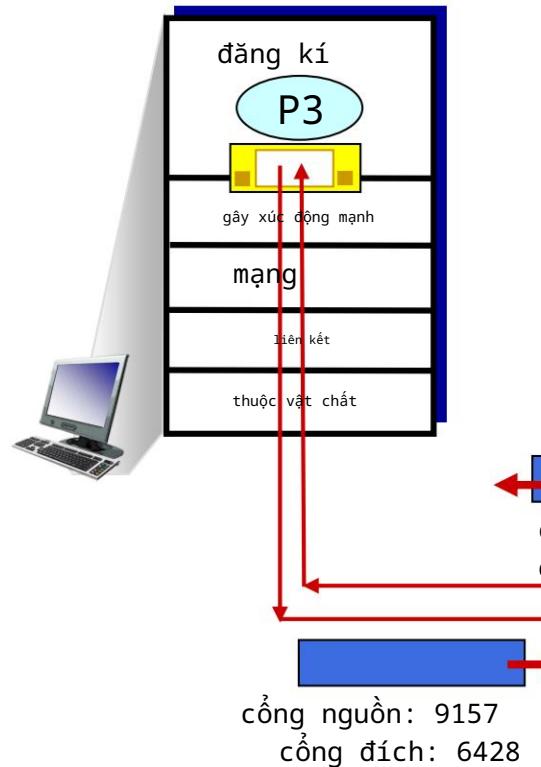
- kiểm tra **cổng đích #** trong phân đoạn
- hướng **phân đoạn UDP** tới **ổ cắm** có cổng đó #



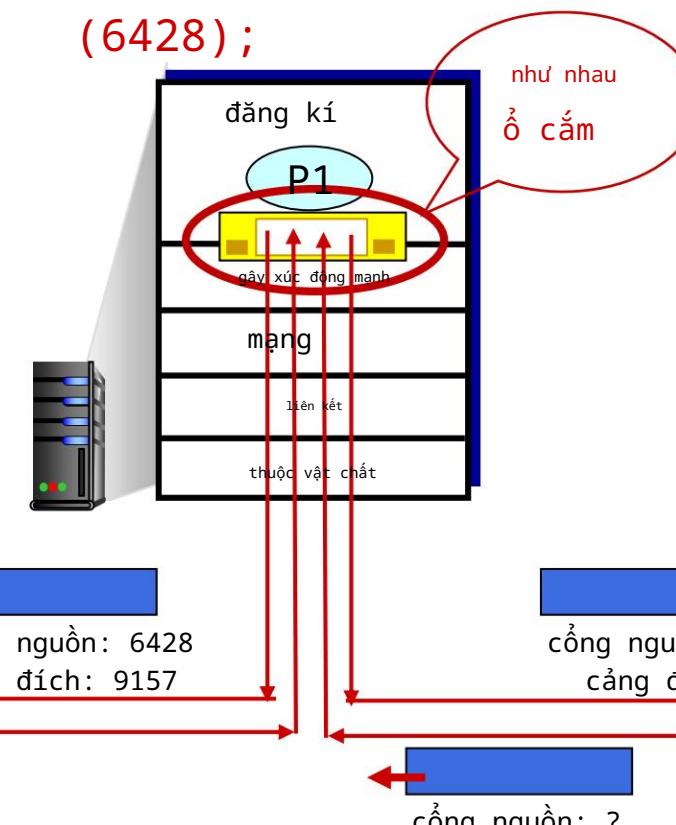
Các datagram IP/UDP có **cùng đích**.
cổng #, nhưng các **địa chỉ IP**
nguồn và/hoặc số cổng nguồn khác
nhau sẽ được chuyển hướng đến **cùng**
một ổ cắm tại máy chủ nhận

Phân kênh không kết nối: một ví dụ

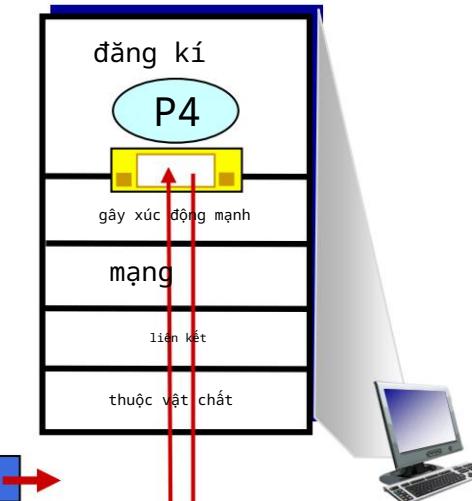
```
DatagramSocket mySocket2 = DatagramSocket  
mới (9157);
```



Máy chủ
DatagramSocketSocket = mới
DatagramSocket
(6428);



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



Phân kênh hướng kết nối

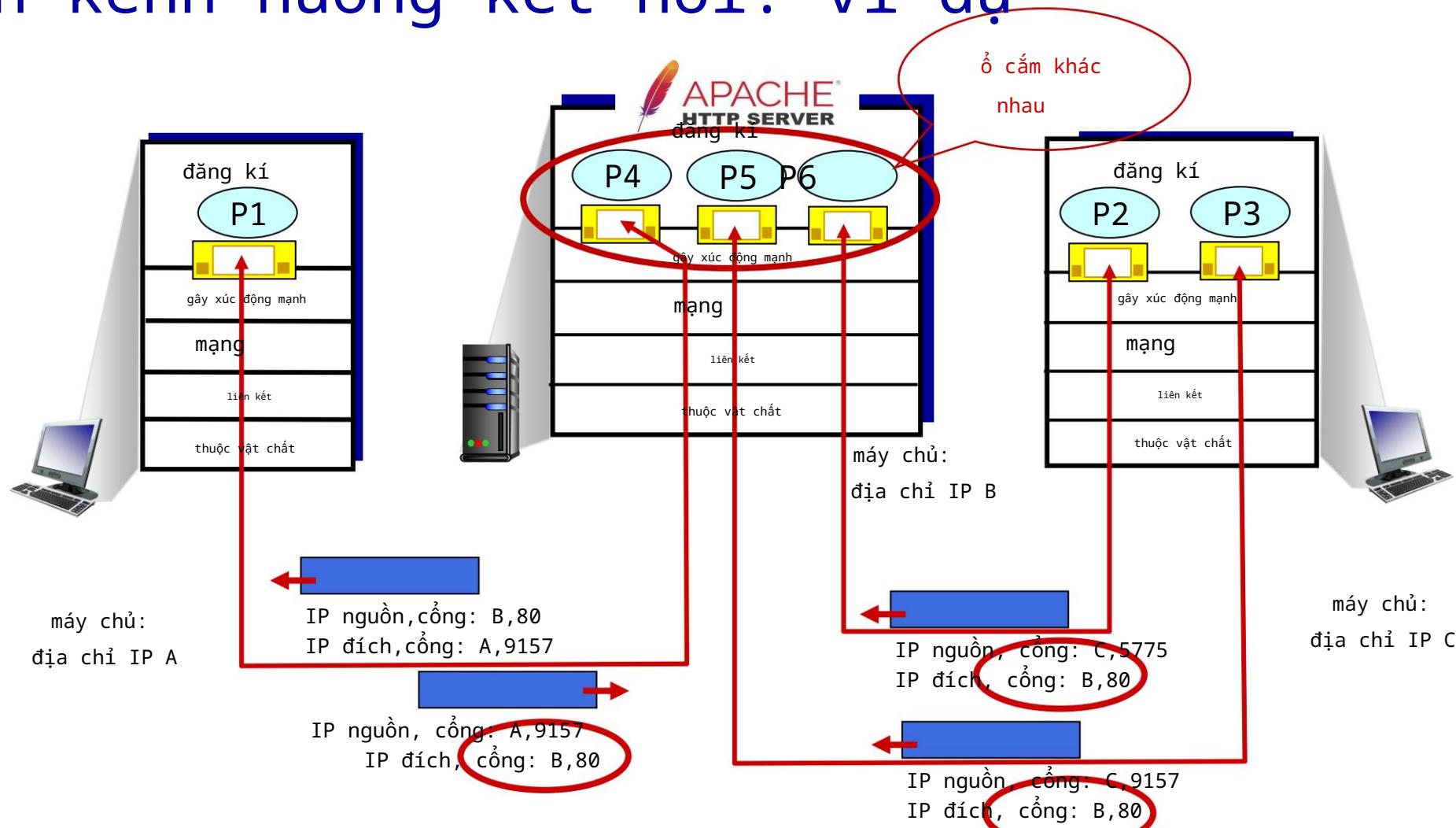
Ô cắm TCP được xác định bởi **4-tuple**: • địa chỉ IP nguồn

• số cổng nguồn •
đích. địa chỉ IP •
đích. số cổng

demux: bộ thu sử dụng **tất cả bốn giá trị** (4-tuple) để hướng phân đoạn tới ô cắm thích hợp

máy chủ có thể hỗ trợ **nhiều** ô cắm TCP đồng thời: • mỗi ô cắm được xác định bởi **4-tuple riêng** của nó • mỗi ô cắm được liên kết với **một máy khách** kết nối khác nhau

Phân kênh hướng kết nối: ví dụ



Ba phân đoạn, tất cả đều được gửi đến địa chỉ IP:

B, dest. cổng: 80 được phân tách thành **các** ô cắm khác nhau

Tóm lược

Ghép kênh, tách kênh: dựa trên giá trị trường **tiêu đề**
đoạn, gói dữ liệu

UDP: tách kênh **sử dụng số cổng đích (chỉ)** TCP: tách
kênh **sử dụng 4-tuple: địa chỉ IP nguồn và đích cũng như số**
cổng

Ghép kênh/tách kênh xảy ra ở tất cả các lớp

Chương 3: lộ trình

Dịch vụ tầng vận chuyển
Ghép kênh và phân kênh Vận
chuyển không kết nối: UDP Nguyên
tắc truyền dữ liệu đáng tin cậy
Truyền tải hướng kết nối: TCP
Nguyên tắc kiểm soát tắc nghẽn Kiểm
soát tắc nghẽn TCP Sự phát triển
của chức năng tầng vận chuyển



UDP: Giao thức gói dữ liệu người dùng

"không kiểu cách," "xương trần"

Giao thức vận chuyển Internet

dịch vụ "**nỗ lực cao nhất**", các
phân đoạn UDP có thể:

- **bị mất**

- giao hàng **không theo thứ tự** cho ứng dụng

không **kết nối**:

- **không bắt tay** giữa người gửi
UDP, người nhận • **mỗi phân đoạn**
UDP được xử lý **độc lập** với những
phân đoạn khác

Tại sao lại có UDP?

không thiết lập

kết nối (có thể thêm **độ trễ RTT**)

đơn giản: không có trạng thái kết
nối ở người gửi, người nhận

phiên: kích thước tiêu đề nhỏ

không kiểm soát tắc nghẽn

UDP có thể chạy **nhanh** như
mong muốn!

có thể hoạt động khi **tắc**
nghẽn

UDP: Giao thức gói dữ liệu người dùng

UDP được sử dụng cho:

phát trực tuyến các ứng dụng đa phương tiện (chịu mất dữ liệu, nhạy cảm với tốc độ) DNS

SNMP

HTTP/3

nếu cần truyền đáng tin cậy qua UDP (ví dụ: HTTP/3): thêm độ tin cậy cần thiết ở lớp ứng dụng thêm kiểm soát tắc nghẽn ở lớp ứng dụng

UDP: Giao thức gói dữ liệu người dùng [RFC 768]

INTERNET STANDARD
J. Postel
ISI
28 August 1980

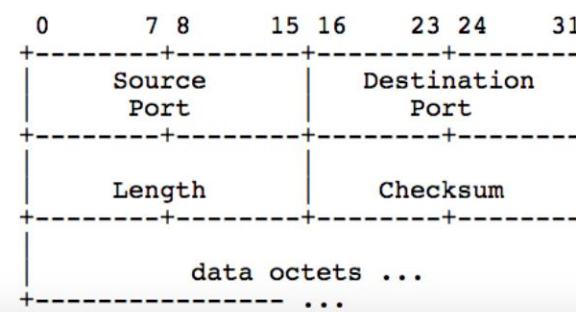
User Datagram Protocol

Introduction

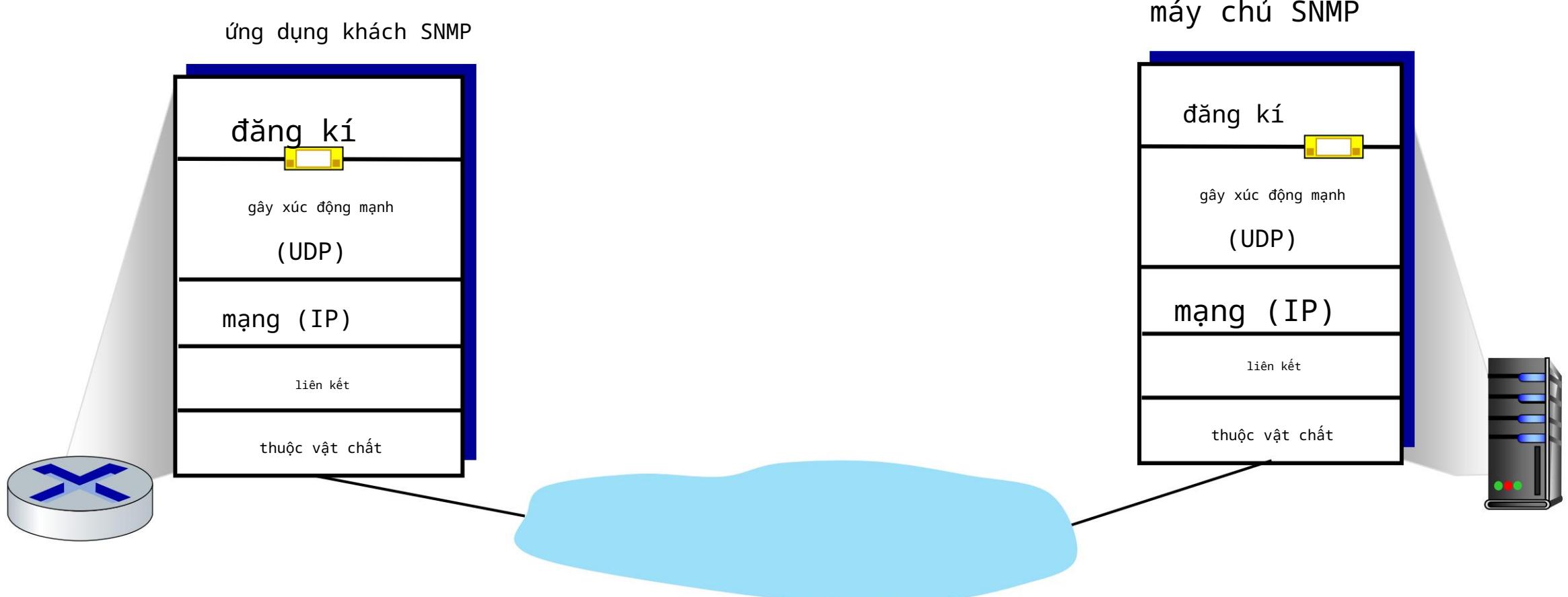
This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format



UDP: Hành động tầng vận chuyển

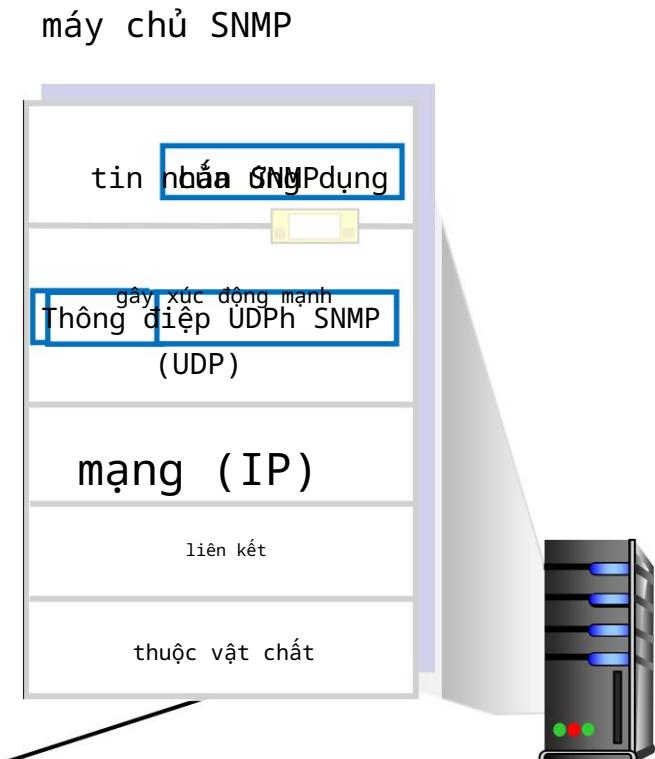


UDP: Hành động tầng vận chuyển

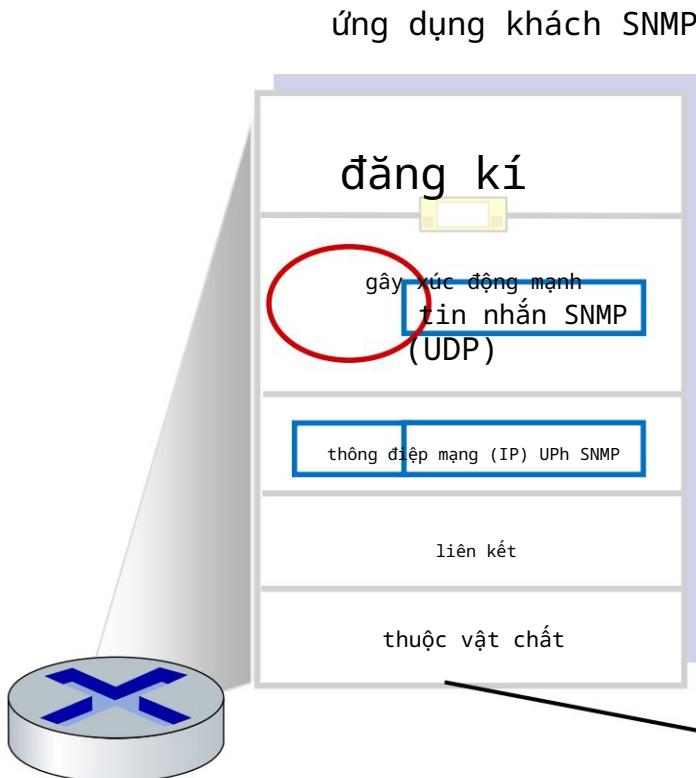


Hành động của **người gửi UDP** :

được **chuyển** một thông báo tầng ứng dụng **xác định** giá trị các trường tiêu đề phân đoạn UDP **tạo** phân đoạn UDP **chuyển** phân đoạn tới IP



UDP: Hành động tầng vận chuyển



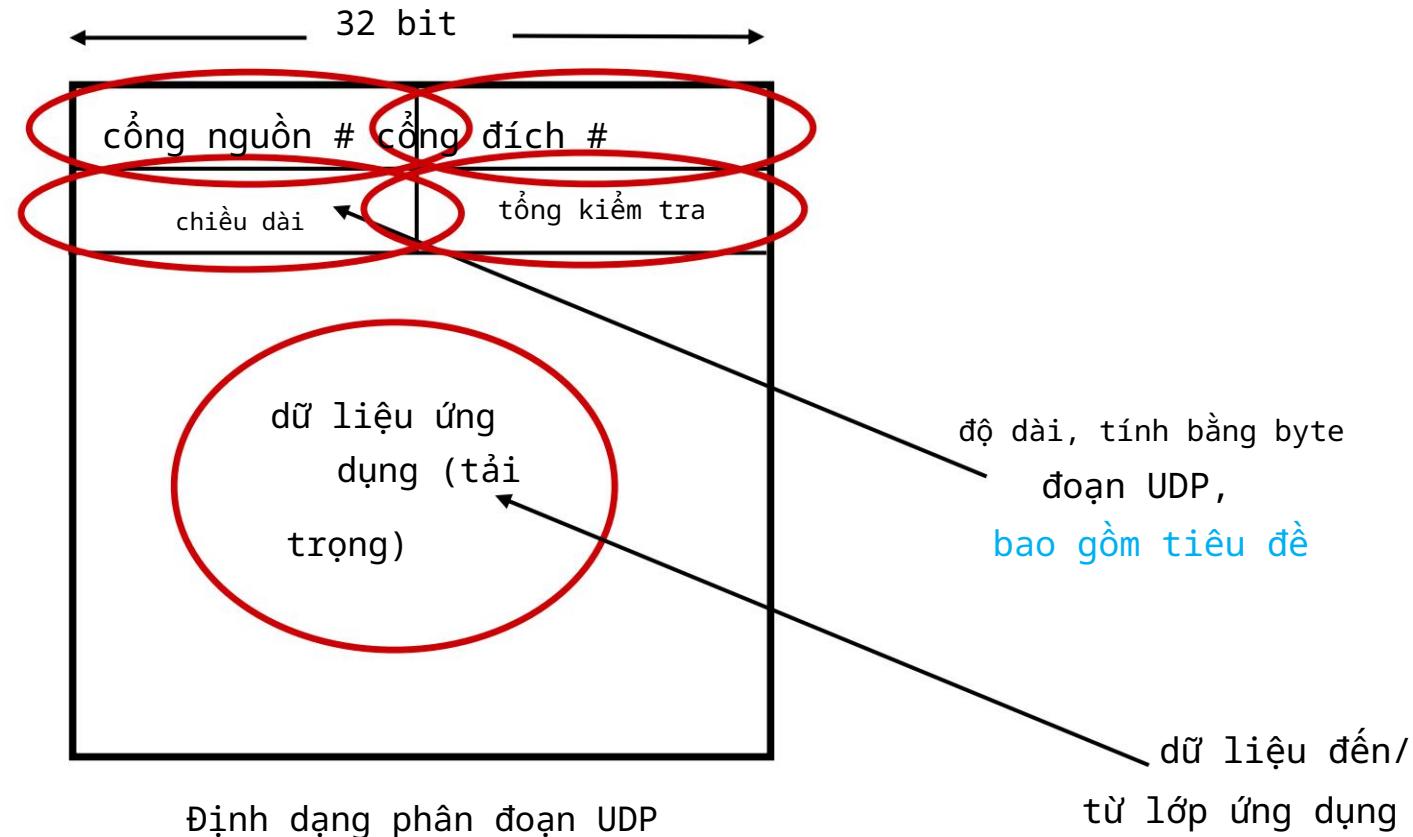
Hành động của **người nhận UDP** :

nhận phân đoạn từ IP
kiểm tra giá trị tiêu đề tổng
kiểm tra UDP
trích xuất thông báo
 tầng ứng dụng **tách**
 thông báo tới ứng dụng
 thông qua socket

máy chủ SNMP

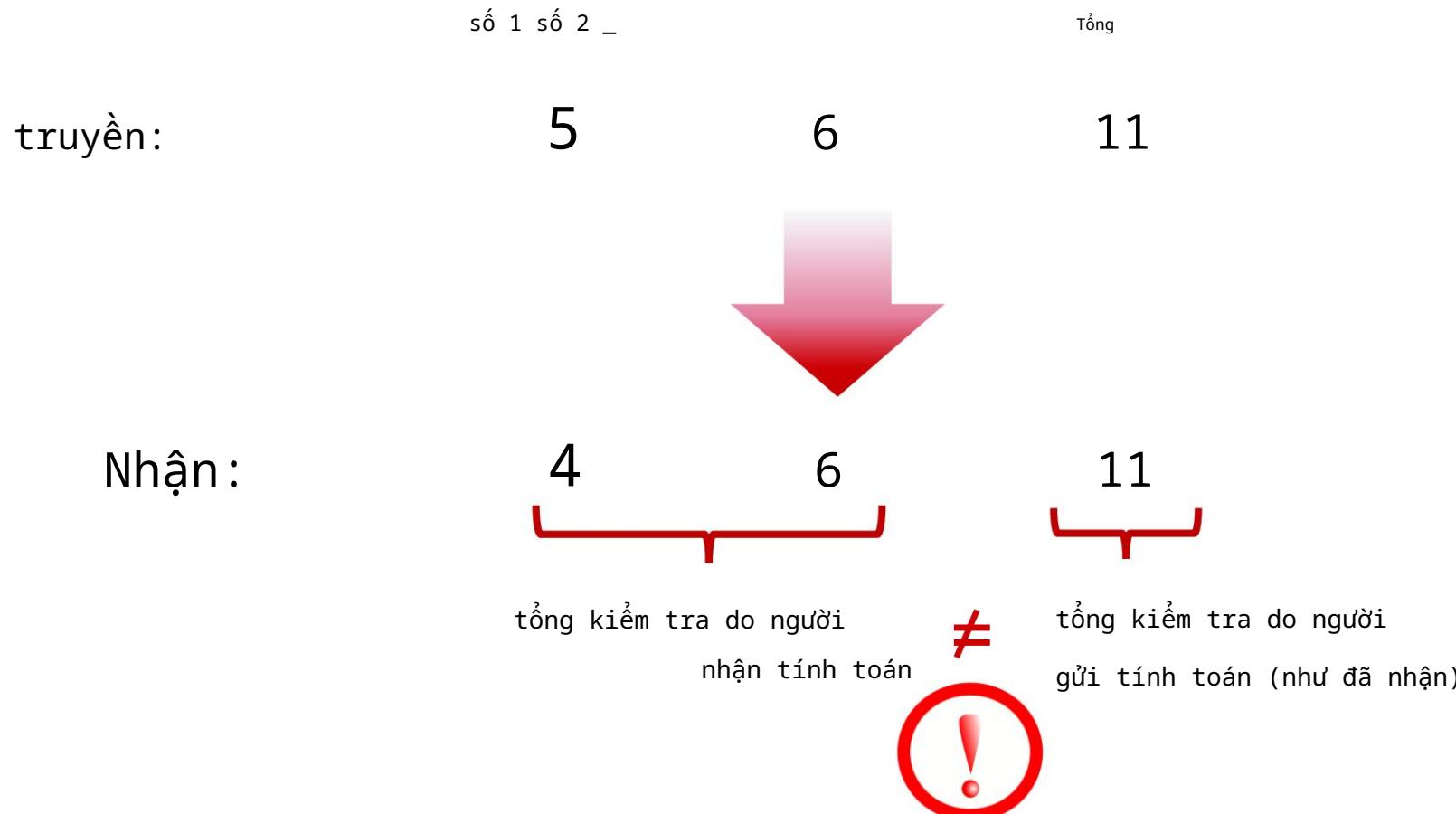


Tiêu đề phân đoạn UDP



tổng kiểm tra UDP

Mục tiêu: phát hiện lỗi (nghĩa là **các bit bị đảo lộn**) trong phân đoạn được truyền



tổng kiểm tra Internet

Mục tiêu: phát hiện lỗi (nghĩa là các bit bị đảo lộn) trong phân đoạn được truyền

người gửi:

xử lý nội dung của phân
đoạn UDP (bao gồm các trường tiêu
đề UDP và địa chỉ IP) dưới dạng
chuỗi các số nguyên 16 bit

tổng kiểm tra: phép cộng (tổng bù
của một người) nội dung phân khúc

giá trị **tổng kiểm** đưa vào
Trường tổng kiểm tra UDP

người nhận:

tính toán tổng kiểm tra của phân đoạn nhận
được

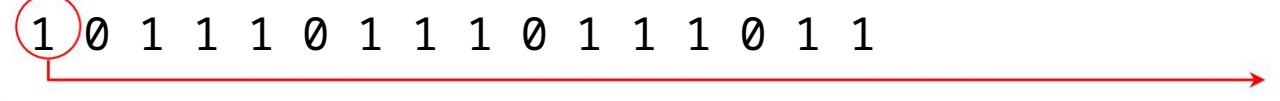
kiểm tra xem tổng kiểm tra được tính có bằng giá trị
trường tổng kiểm tra hay không:

- **không bằng** - phát hiện
lỗi
 - **bằng** - không phát hiện lỗi.
- Nhưng có lẽ lỗi, dù sao? Trễ hơn ..

Tổng kiểm tra Internet: một ví dụ

ví dụ: cộng hai số nguyên 16 bit

$$\begin{array}{r}
 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\
 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\
 \hline
 \text{bao quanh } 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1
 \end{array}$$



 Tổng 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
 tổng kiểm tra 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

Lưu ý: khi thêm số, cần thêm phần thực hiện từ **bit quan trọng nhất** vào kết quả

* Xem các bài tập tương tác trực tuyến để biết thêm ví dụ: http://gaia.cs.umass.edu/kurose_ross/interactive/

Tổng kiểm tra Internet: bảo vệ yếu!

ví dụ: cộng hai số nguyên 16 bit

bao quanh 1 **1** 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1

Tổng 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 0

tổng kiểm tra 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

Mặc dù các số đã thay đổi (lật bit), không có thay đổi trong tổng kiểm tra!

Tổng hợp: UDP

Giao thức “không rườm rà”:

- các phân đoạn có thể bị thất lạc, giao không đúng thứ tự • dịch vụ nỗ lực tối đa : “gửi đi và hy vọng điều tốt nhất”

UDP có những điểm hạn chế:

- không cần thiết lập/bắt tay (không phát sinh RTT) • có thể hoạt động khi dịch vụ mạng bị xâm phạm • trợ giúp về độ tin cậy (tổng kiểm tra) xây dựng chức năng bổ sung trên đầu trang của UDP trong lớp ứng dụng (ví dụ: HTTP/3)

Chương 3: lô trình

Dịch vụ tầng vận chuyển

Ghép kênh và phân kênh Vận
chuyển không kết nối: UDP Nguyên

tắc truyền dữ liệu đáng tin cậy

Truyền tải hướng kết nối: TCP

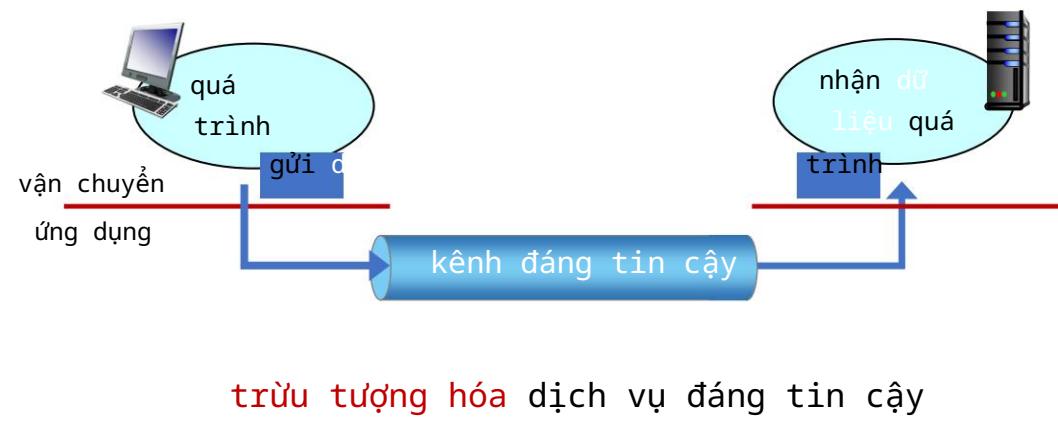
Nguyên tắc kiểm soát tắc nghẽn Kiểm

soát tắc nghẽn TCP Sự phát triển

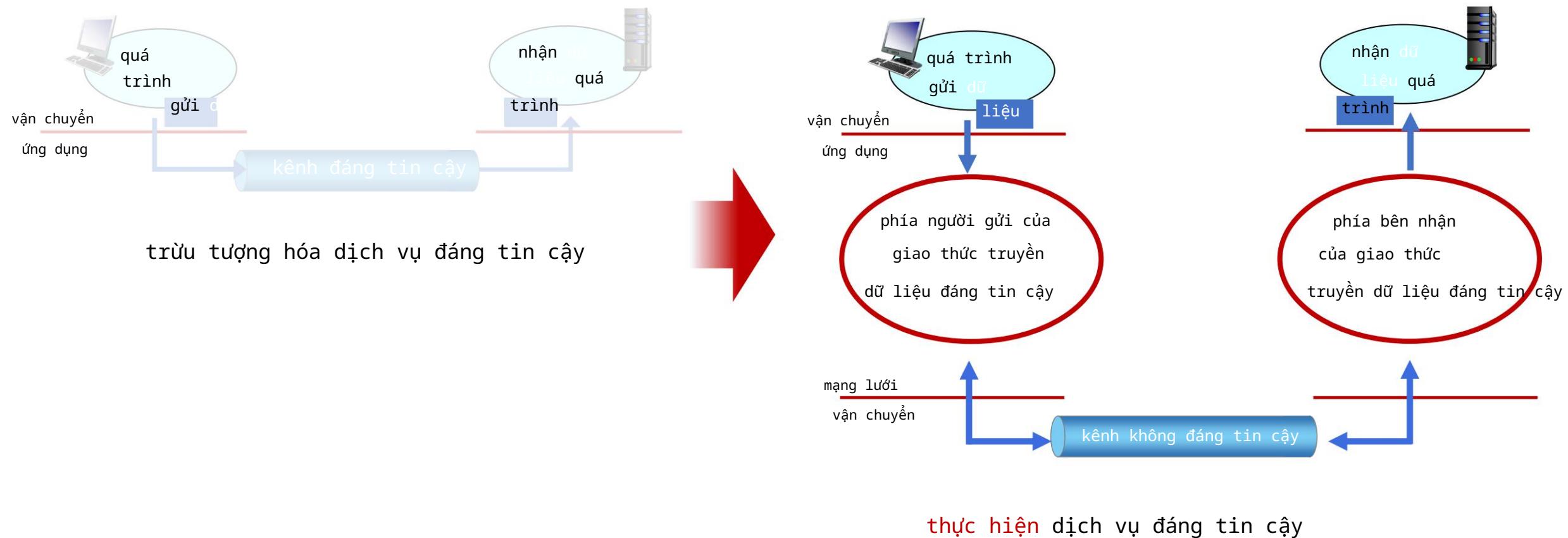
của chức năng tầng vận chuyển



Nguyên tắc truyền dữ liệu đáng tin cậy

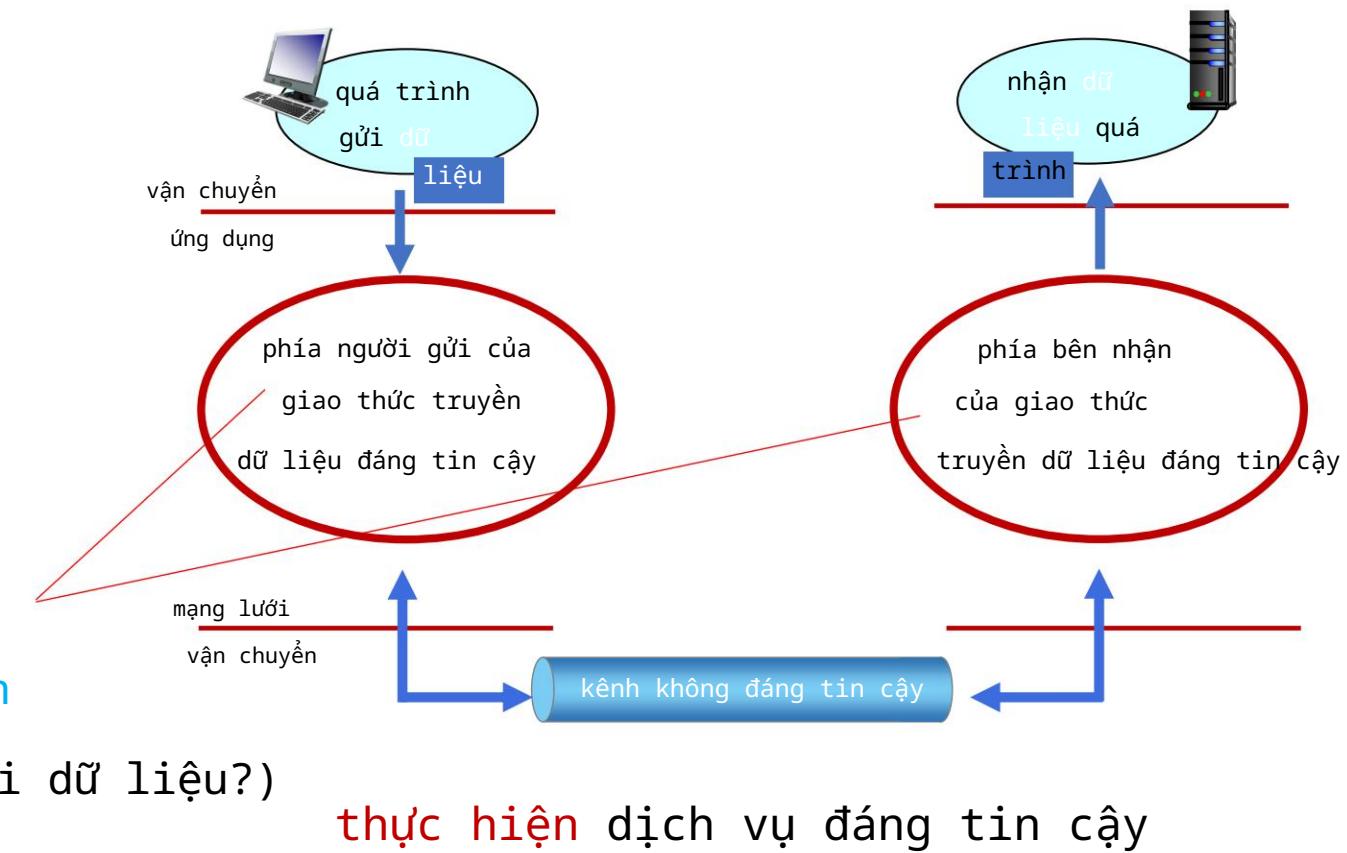


Nguyên tắc truyền dữ liệu đáng tin cậy



Nguyên tắc truyền dữ liệu đáng tin cậy

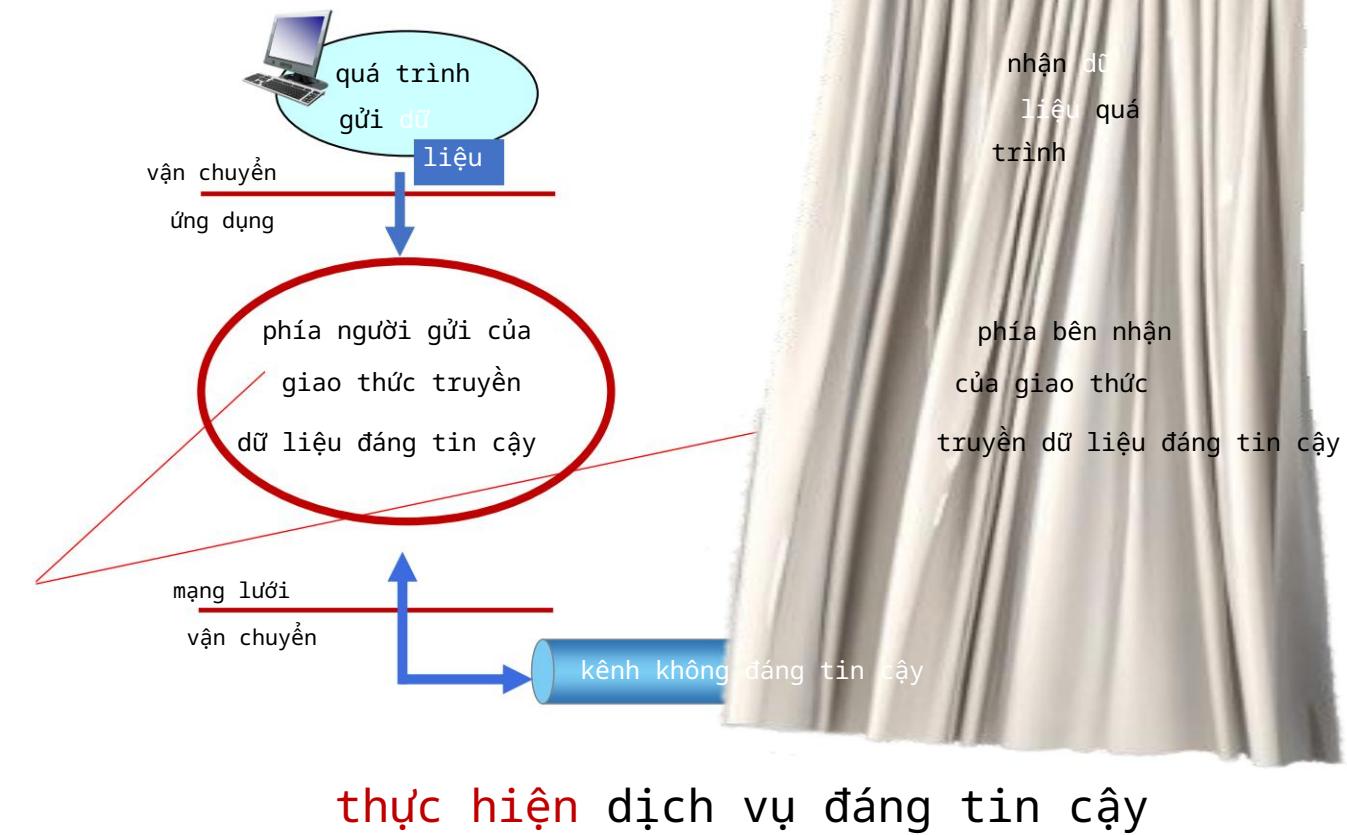
Độ phức tạp của giao thức
truyền dữ liệu đáng tin cậy sẽ
phụ thuộc (mạnh mẽ) vào các đặc
điểm của kênh không đáng tin
cậy (mất, hỏng, sắp xếp lại dữ liệu?)



thực hiện dịch vụ đáng tin cậy

Nguyên tắc truyền dữ liệu đáng tin cậy

Người gửi, người nhận **không biết** “trạng thái” của nhau, vd:
đã nhận được tin nhắn chưa?
trừ khi được **thông báo qua một thông điệp**

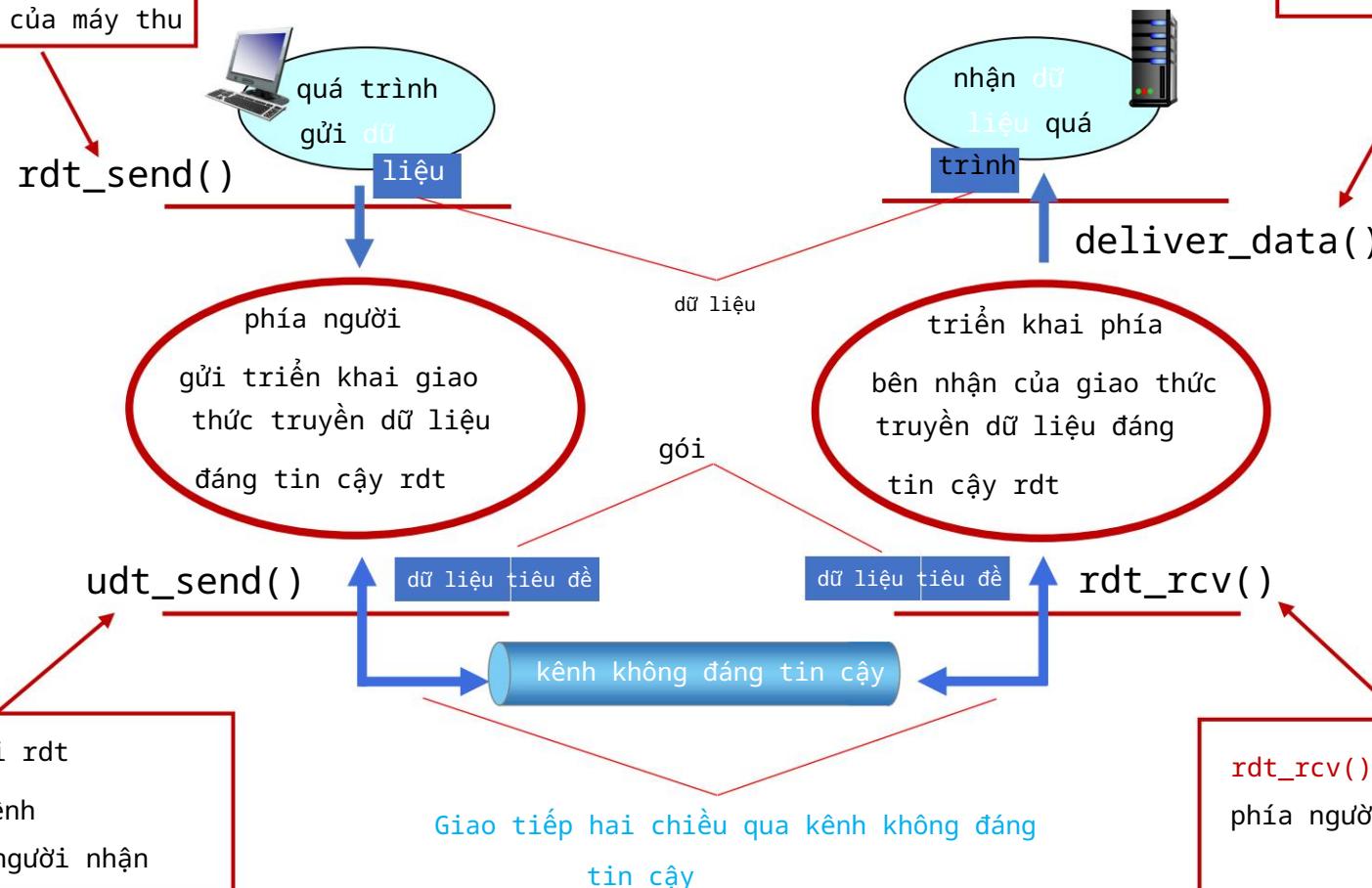


thực hiện dịch vụ đáng tin cậy

Giao thức truyền dữ liệu đáng tin cậy (rdt): giao diện

`rdt_send()`: được gọi từ phía trên, (ví dụ: bởi ứng dụng.). Dữ liệu đã truyền để gửi đến lớp trên của máy thu

`Deliver_data()`: được gọi bởi rdt để chuyển dữ liệu lên tầng trên



Truyền dữ liệu đáng tin cậy: bắt đầu

Chúng tôi

sẽ: từng bước phát triển bên gửi, bên nhận của giao_thức truyền dữ liệu
đáng tin cậy (rdt)

chỉ xem xét truyền dữ liệu một chiều • nhưng
thông tin điều khiển sẽ truyền theo cả hai hướng!

sử dụng máy trạng thái hữu hạn (FSM) để chỉ định người gửi, người nhận



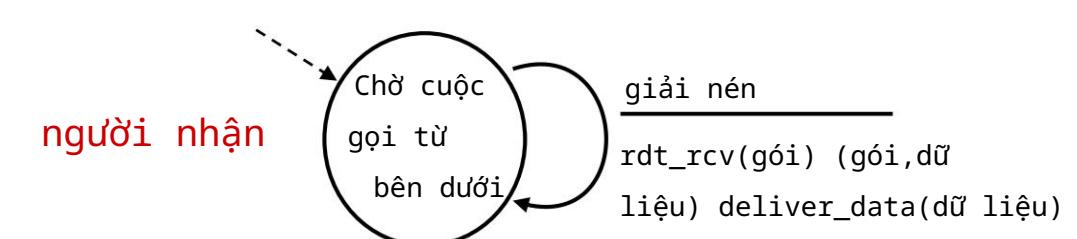
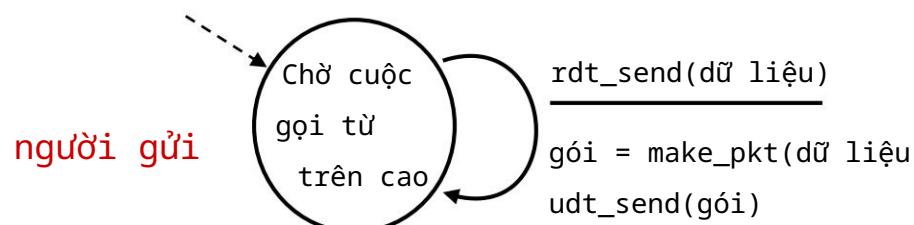
rdt1.0: truyền đáng tin cậy qua một kênh đáng tin cậy

kênh cơ sở hoàn toàn đáng tin cậy

- không có lỗi bit
- không bị mất gói tin

FSM **riêng biệt** cho người gửi, người nhận:

- bên **gửi** gửi dữ liệu vào kênh bên dưới
- bên **nhận** đọc dữ liệu từ kênh bên dưới



rdt2.0: kênh có lỗi bit

kênh bên dưới có thể lật các bit trong gói tin

- tổng kiểm tra (ví dụ: tổng kiểm tra Internet) để phát hiện lỗi bit

câu hỏi: làm thế nào để khôi phục lỗi?

Làm thế nào để con người phục hồi từ "lỗi" trong khi trò chuyện?

rdt2.0: kênh có lỗi bit

kênh bên dưới có thể lật bit trong gói • tổng kiểm tra để phát hiện lỗi bit

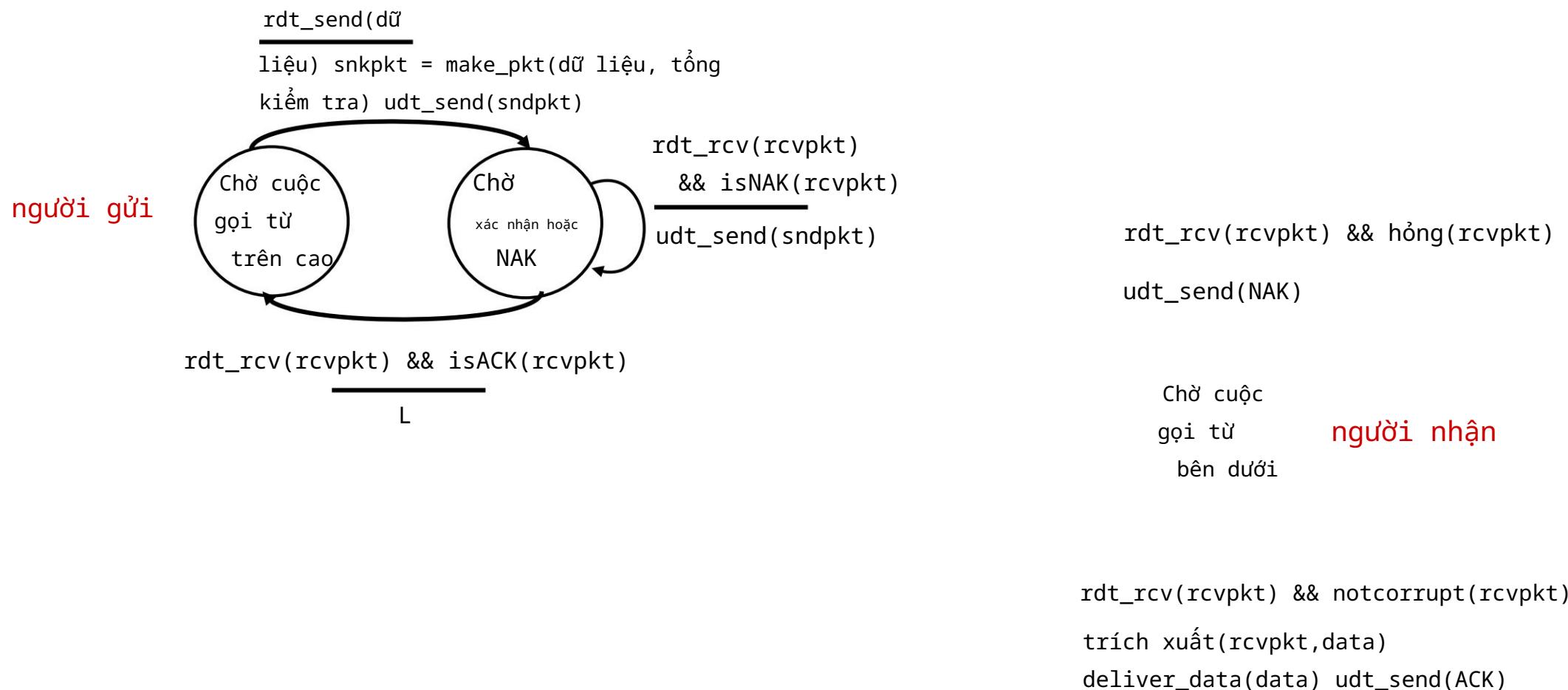
câu hỏi: làm thế nào để khôi phục lỗi?

- **xác nhận (ACK)**: người nhận thông báo rõ ràng cho người gửi rằng pkt nhận được **OK**
- **xác nhận tiêu cực (NAK)**: người nhận thông báo rõ ràng cho người gửi rằng gói có **lỗi** • người gửi **truyền lại** gói khi nhận được NAK

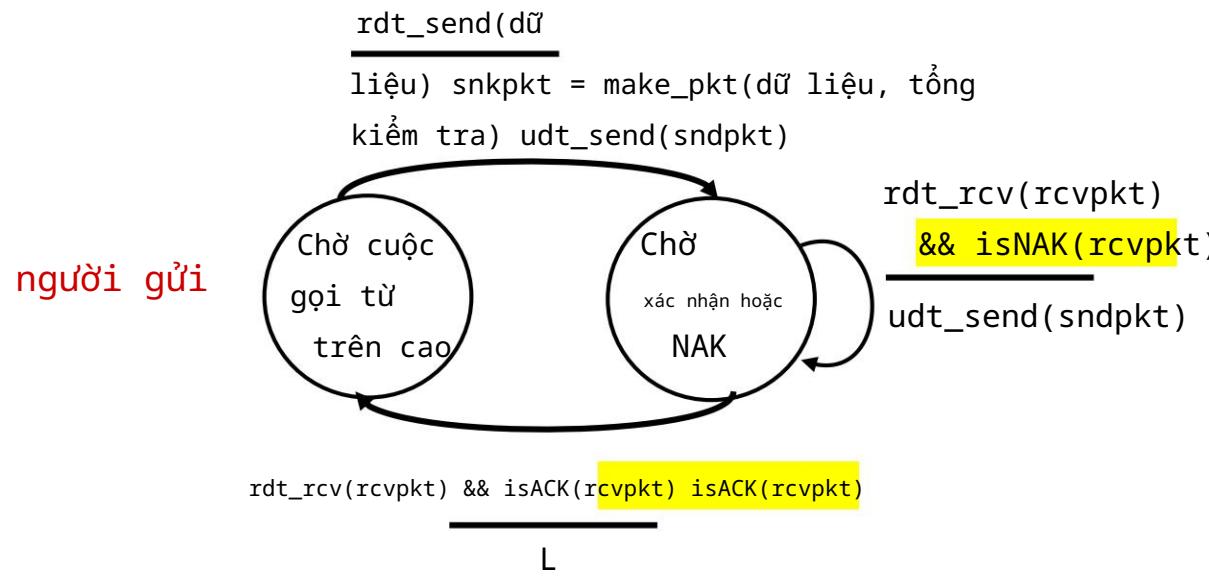
dừng và đợi

người gửi gửi một gói, sau đó đợi phản hồi của người nhận

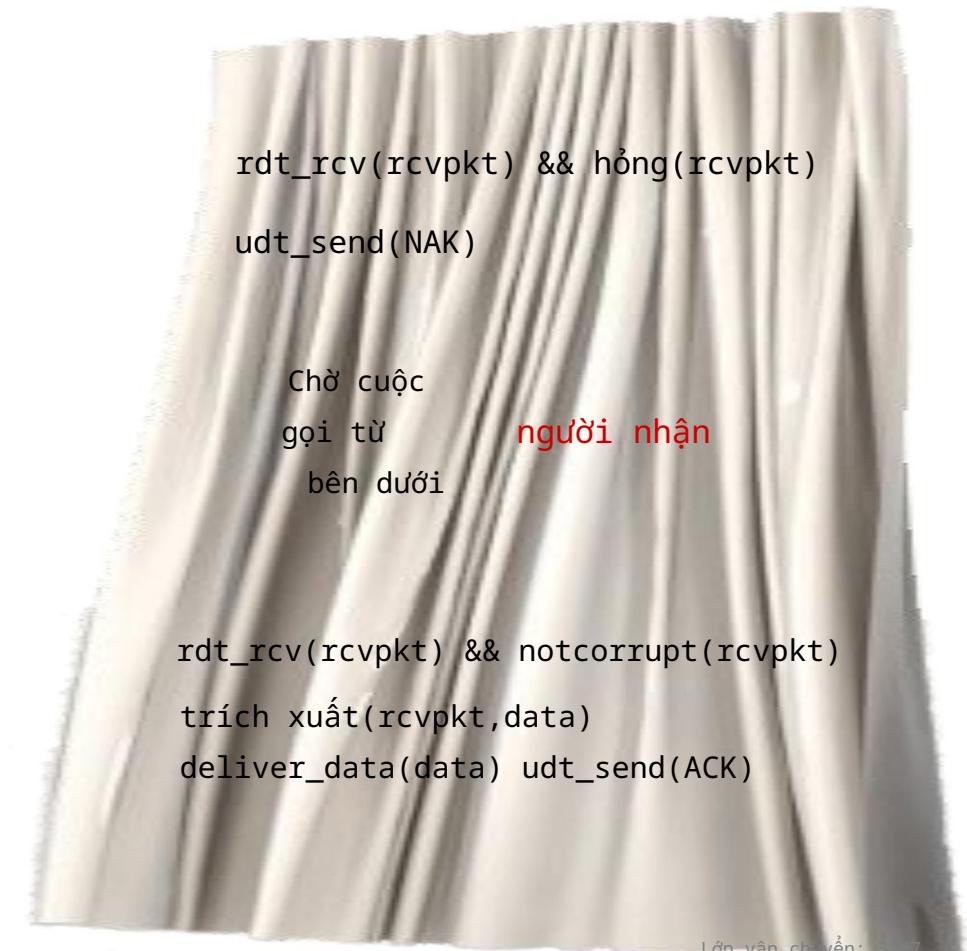
rdt2.0: Thông số kỹ thuật của FSM



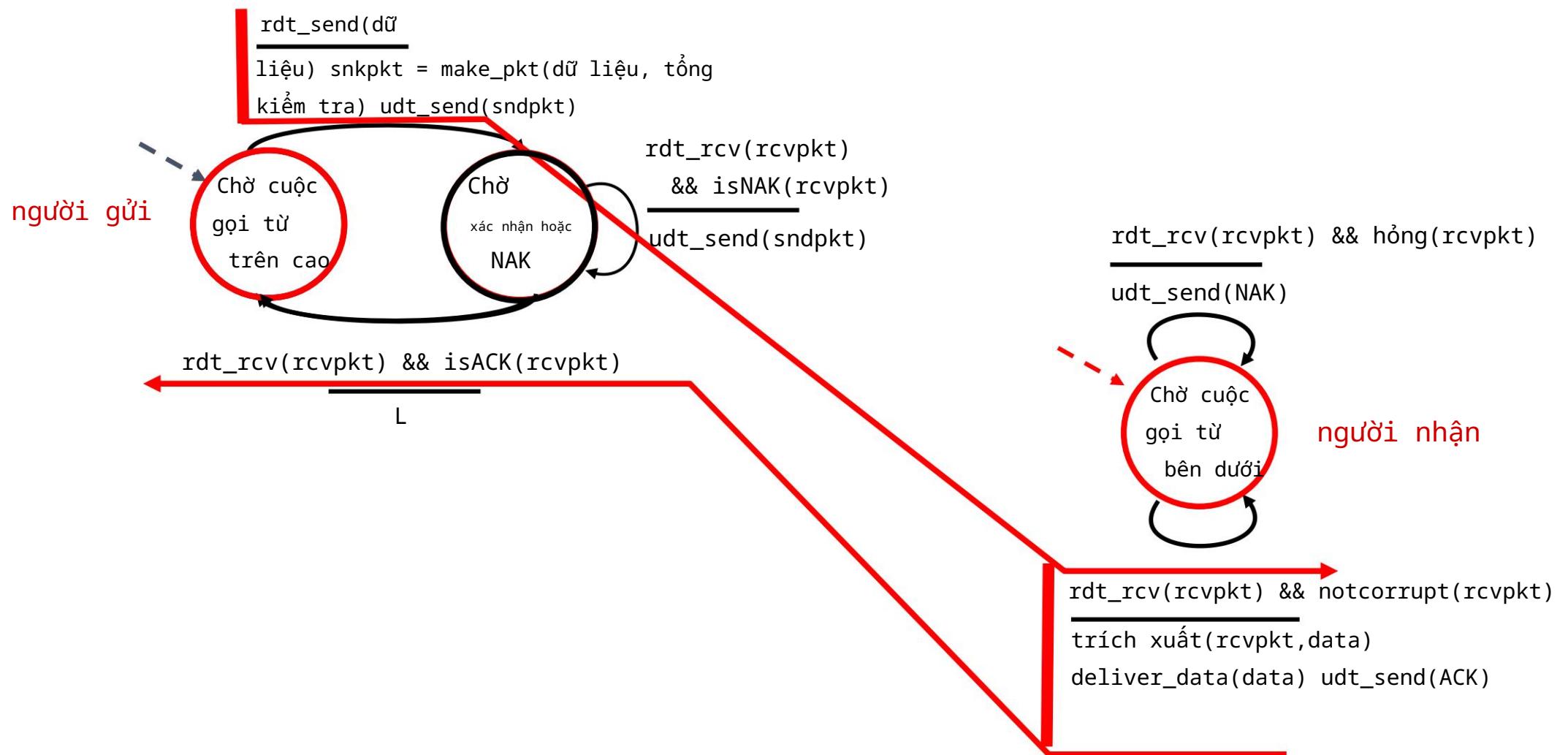
rdt2.0: Thông số FSM



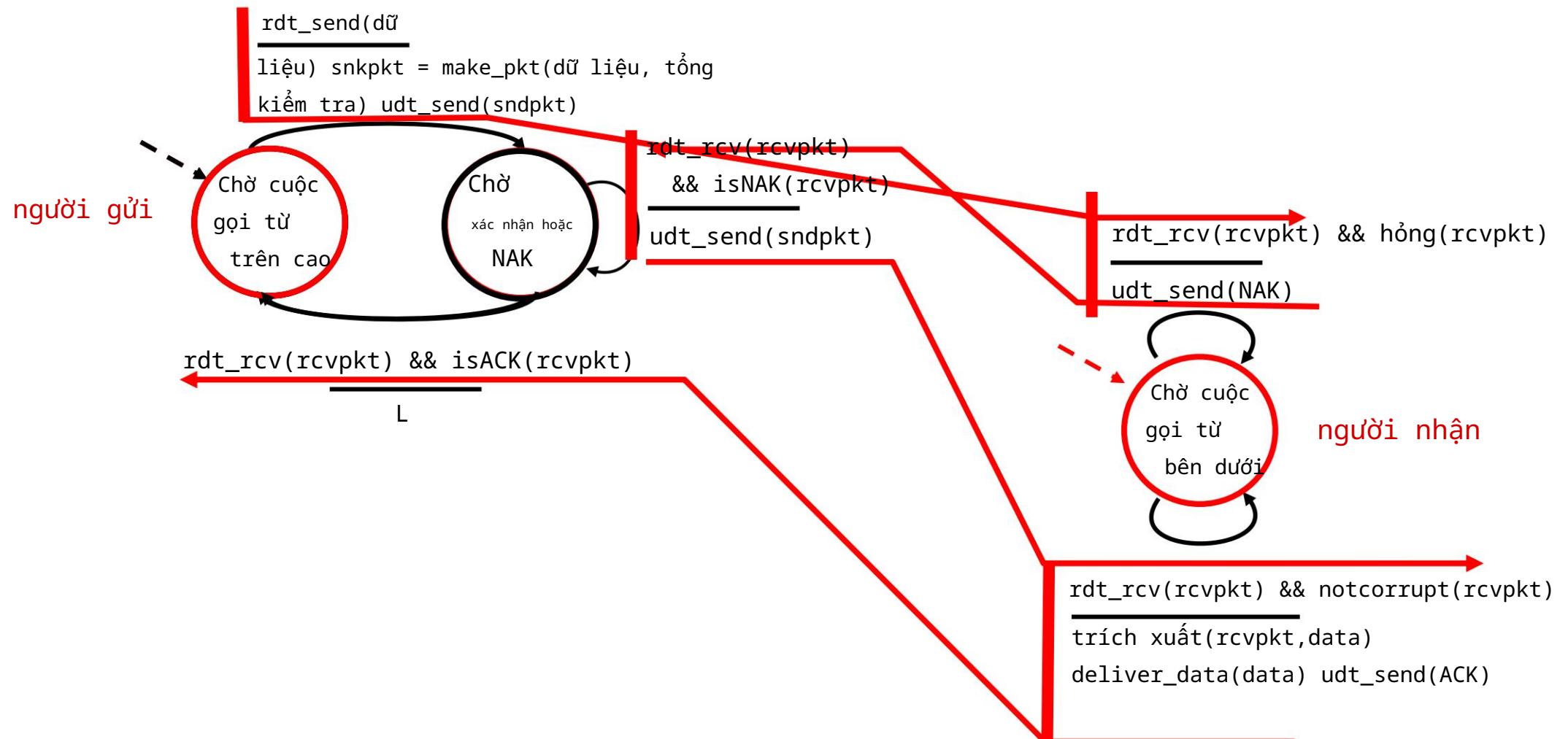
Lưu ý: “**trạng thái**” của người nhận (người nhận có nhận được tin nhắn của tôi chính xác không?) không được người gửi biết trừ khi được **thông báo bằng cách nào đó** từ người nhận đến người gửi đó là lý do tại sao chúng ta cần một giao thức!



rdt2.0: hoạt động không có lỗi



rdt2.0: kịch bản gói bị hỏng



rdt2.0 có một lỗ hỏng chết người!

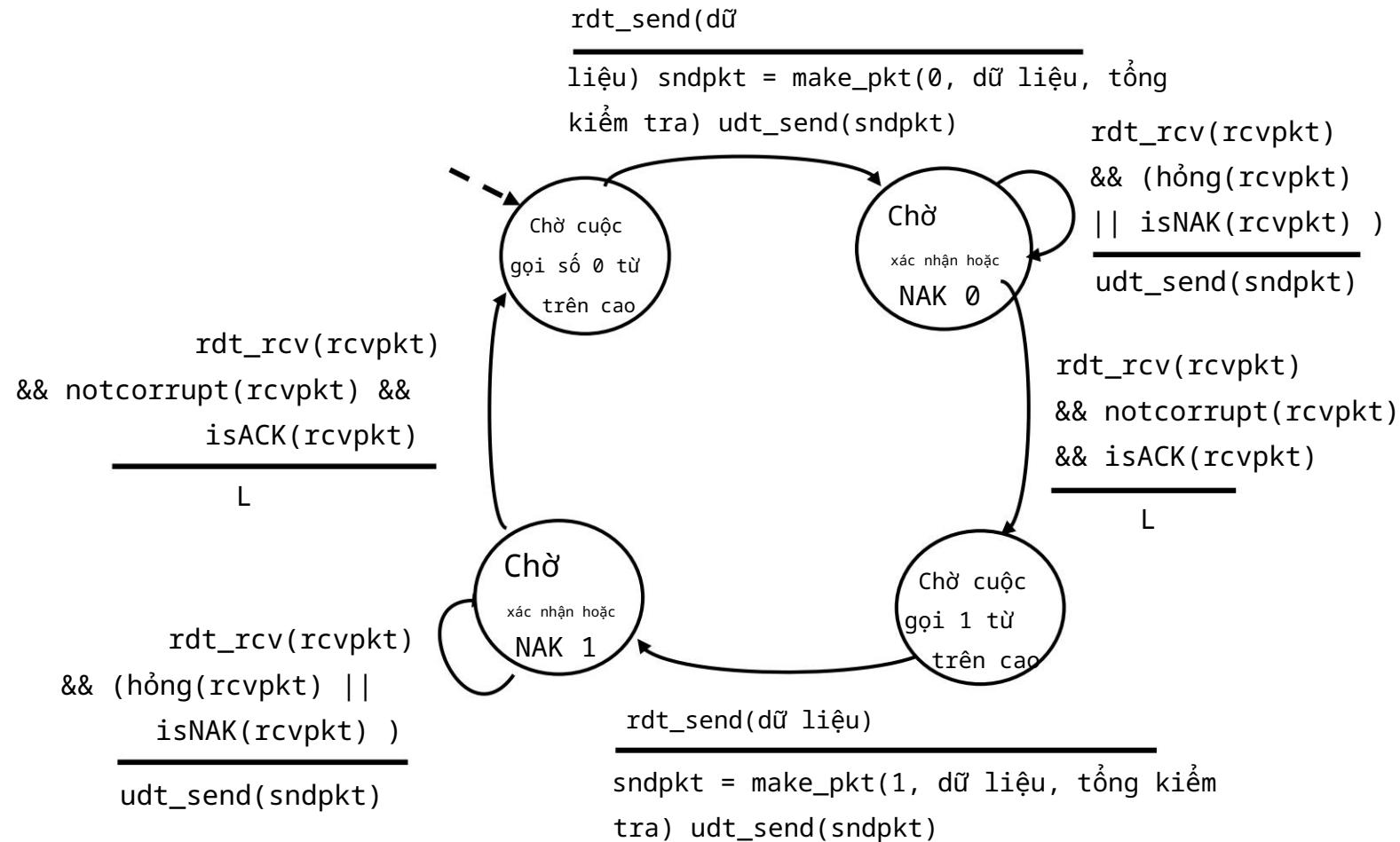
điều gì xảy ra nếu ACK/NAK bị hỏng? Người gửi không biết chuyện gì đã xảy ra ở người nhận! Không thể chỉ truyền lại: có thể trùng lặp

xử lý các gói trùng lặp: bên gửi **truyền lại** gói hiện tại nếu ACK/NAK bị hỏng bên gửi thêm **số thứ tự** vào mỗi gói bên nhận **loại bỏ** (không phân phối) gói trùng lặp

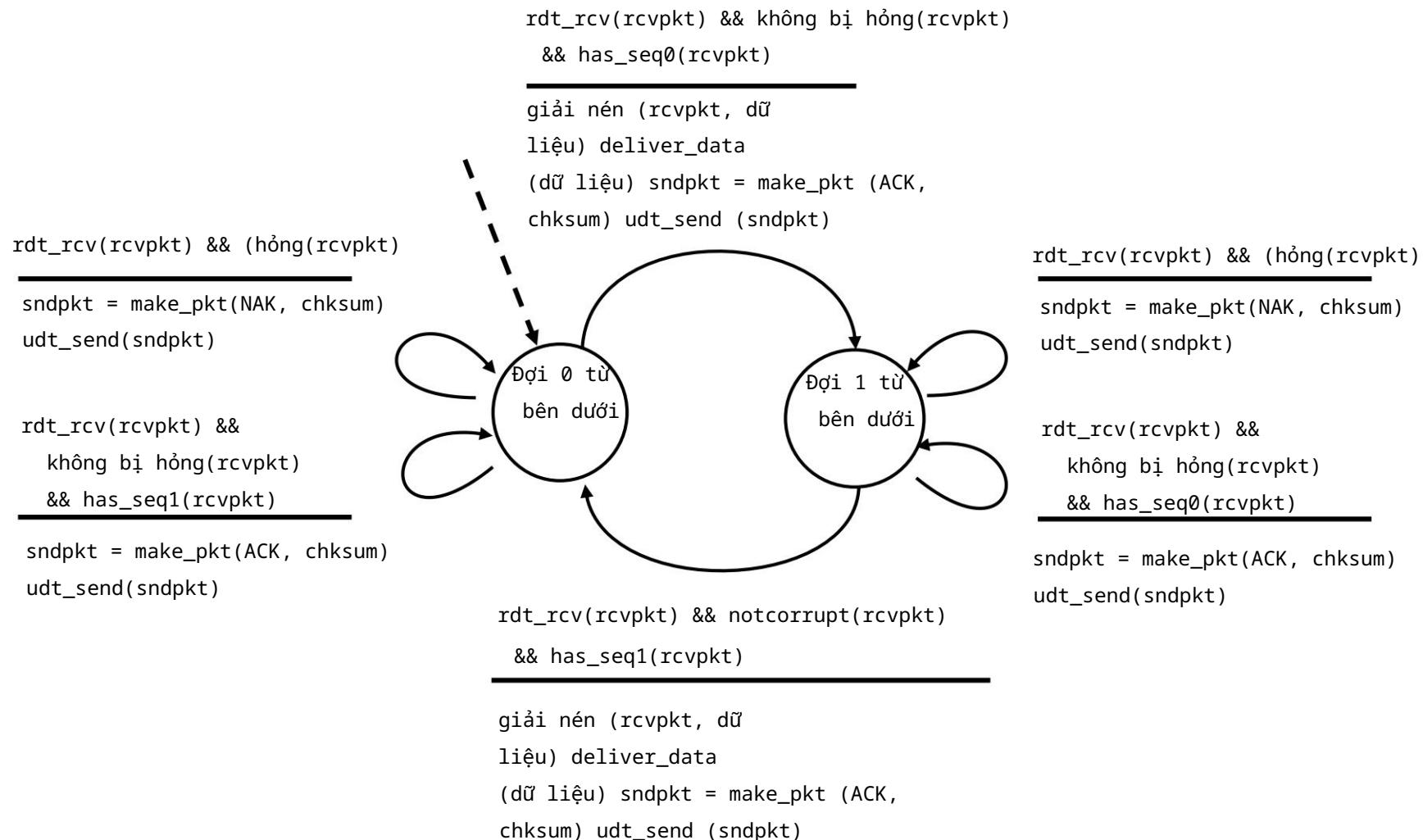
dừng và đợi

người gửi gửi một gói, sau đó đợi phản hồi của người nhận

rdt2.1: người gửi, xử lý ACK/NAK bị cắt xén



rdt2.1: bộ thu, xử lý ACK/NAK bị cắt xén



rdt2.1: thảo luận

người gửi:

thứ tự # được thêm
vào pkt **hai thứ tự.** #s (0,1) là đủ.
Tại sao?

phải kiểm tra xem ACK/NAK nhận được
có bị hỏng không gấp đôi số trạng
thái • **trạng thái** phải “nhớ” liệu gói
“dự kiến” có số thứ tự là 0 hay 1

người nhận:

phải kiểm tra xem gói tin nhận được có
bị trùng lặp không • trạng thái cho biết
liệu 0 hay 1 được mong đợi pkt seq #

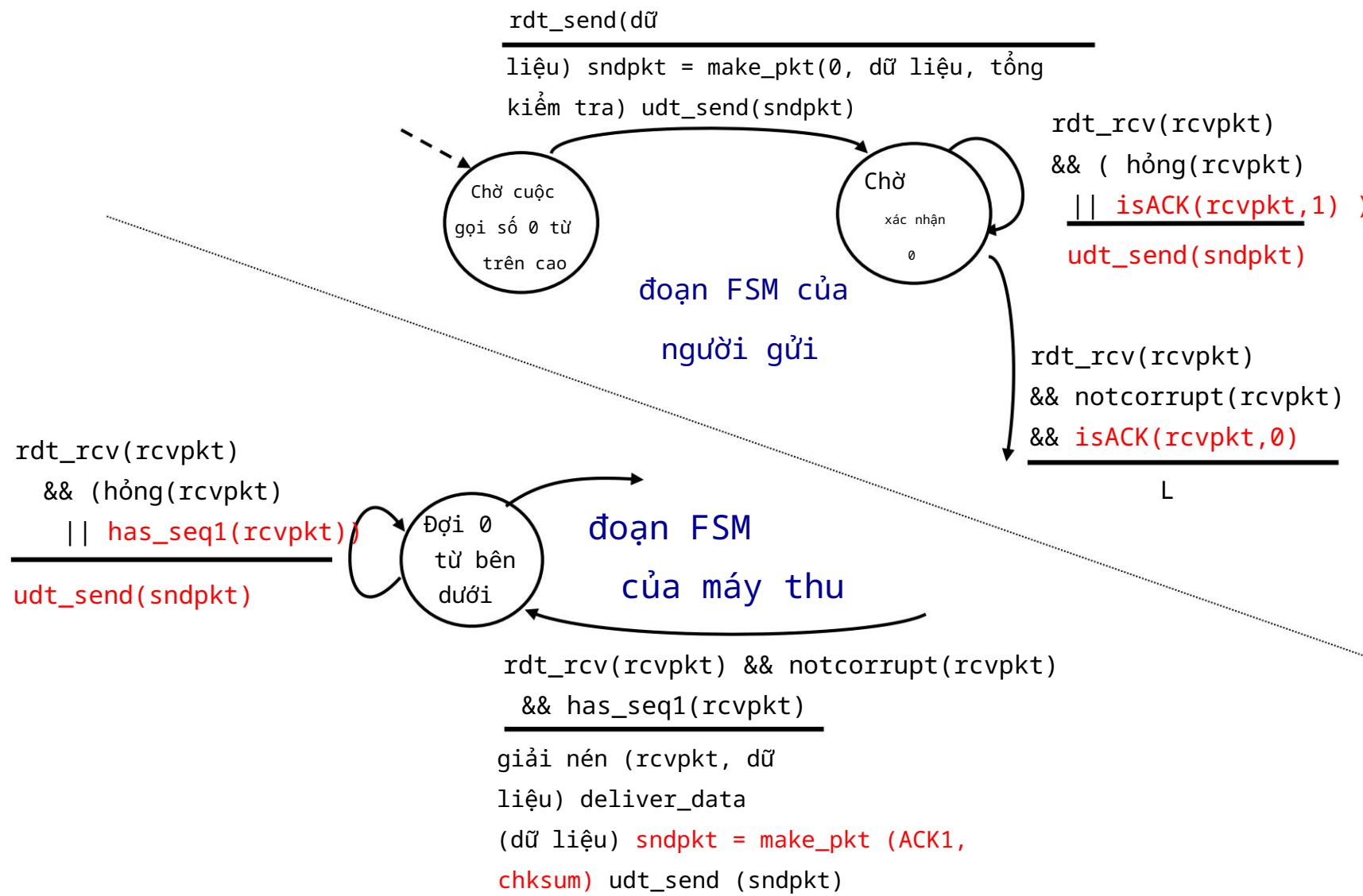
lưu ý: người nhận **không thể biết** liệu ACK/
NAK cuối cùng của mình có nhận được từ người
gửi hay không

rdt2.2: giao thức không có NAK

chức năng tương tự như rdt2.1, chỉ sử dụng ACK
thay vì NAK, người nhận gửi ACK cho gói cuối cùng đã nhận OK • người nhận
phải bao gồm rõ ràng số thứ tự của gói đang được ACK ACK trùng lặp
ở người gửi dẫn đến hành động tương tự như NAK: truyền lại gói hiện tại

Như chúng ta sẽ thấy, TCP sử dụng phương pháp này để không có NAK

rdt2.2: phân đoạn người gửi, người nhận



rdt3.0: các kênh bị lỗi và mất dữ liệu

Giả định kênh mới: kênh bên dưới cũng có thể **mất gói** (dữ liệu, ACK)

- tổng kiểm tra, số thứ tự, ACK, truyền lại sẽ hữu ích . nhưng không đủ

H: Làm thế nào để con người xử lý các từ bị mất từ người gửi đến người nhận trong cuộc trò chuyện?

rdt3.0: các kênh bị lỗi và mất dữ liệu

Cách tiếp cận: người gửi đợi khoảng thời gian “hợp lý” cho ACK

truyền lại nếu không nhận được ACK trong thời gian này nếu pkt (hoặc ACK)

chỉ bị trễ (không bị mất):

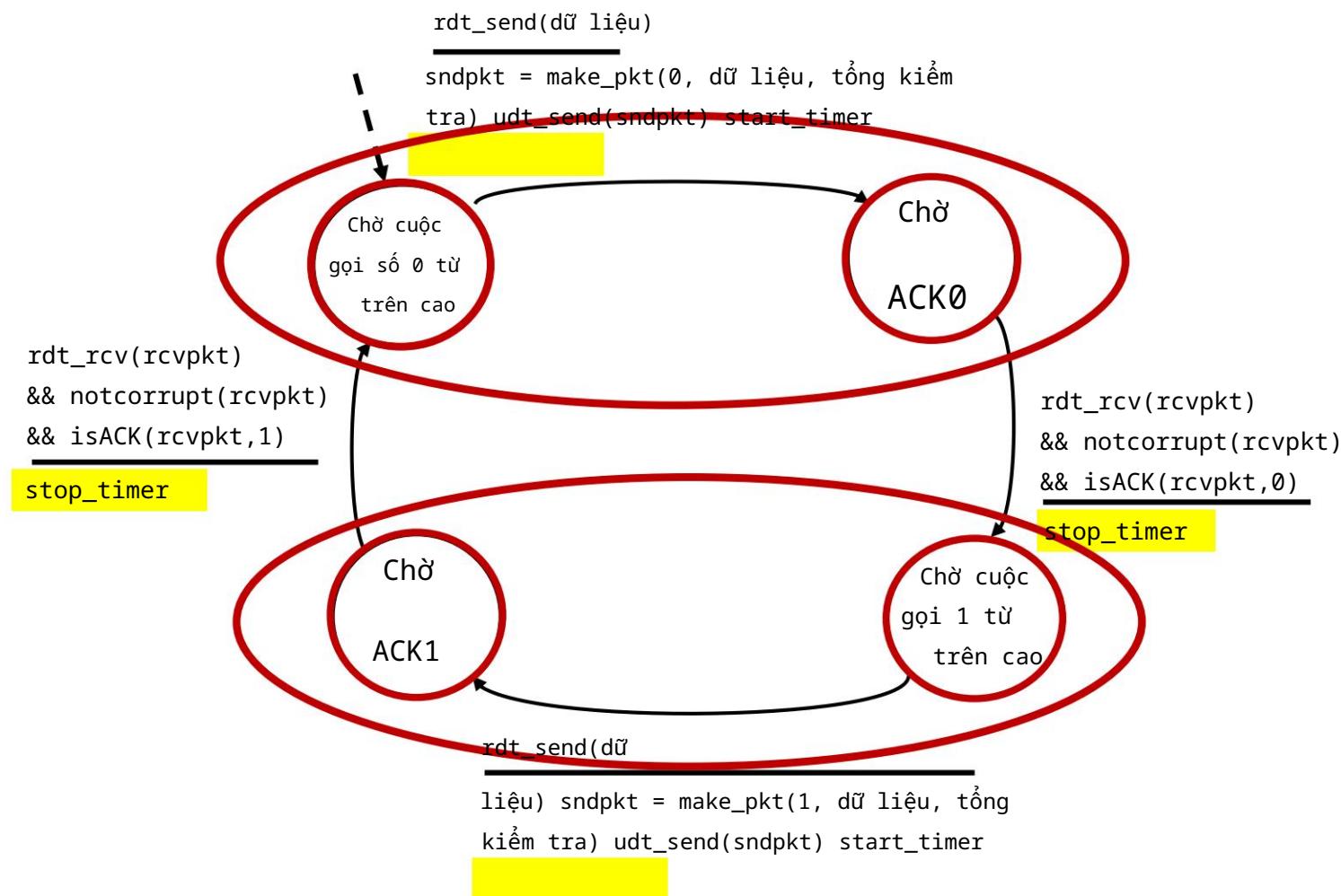
- truyền lại sẽ bị trùng lặp, nhưng seq#s đã xử lý việc này!
- bên nhận phải chỉ định thứ tự # của gói được ACK sử dụng

đồng hồ đếm ngược để ngắt sau khoảng thời gian “hợp lý”

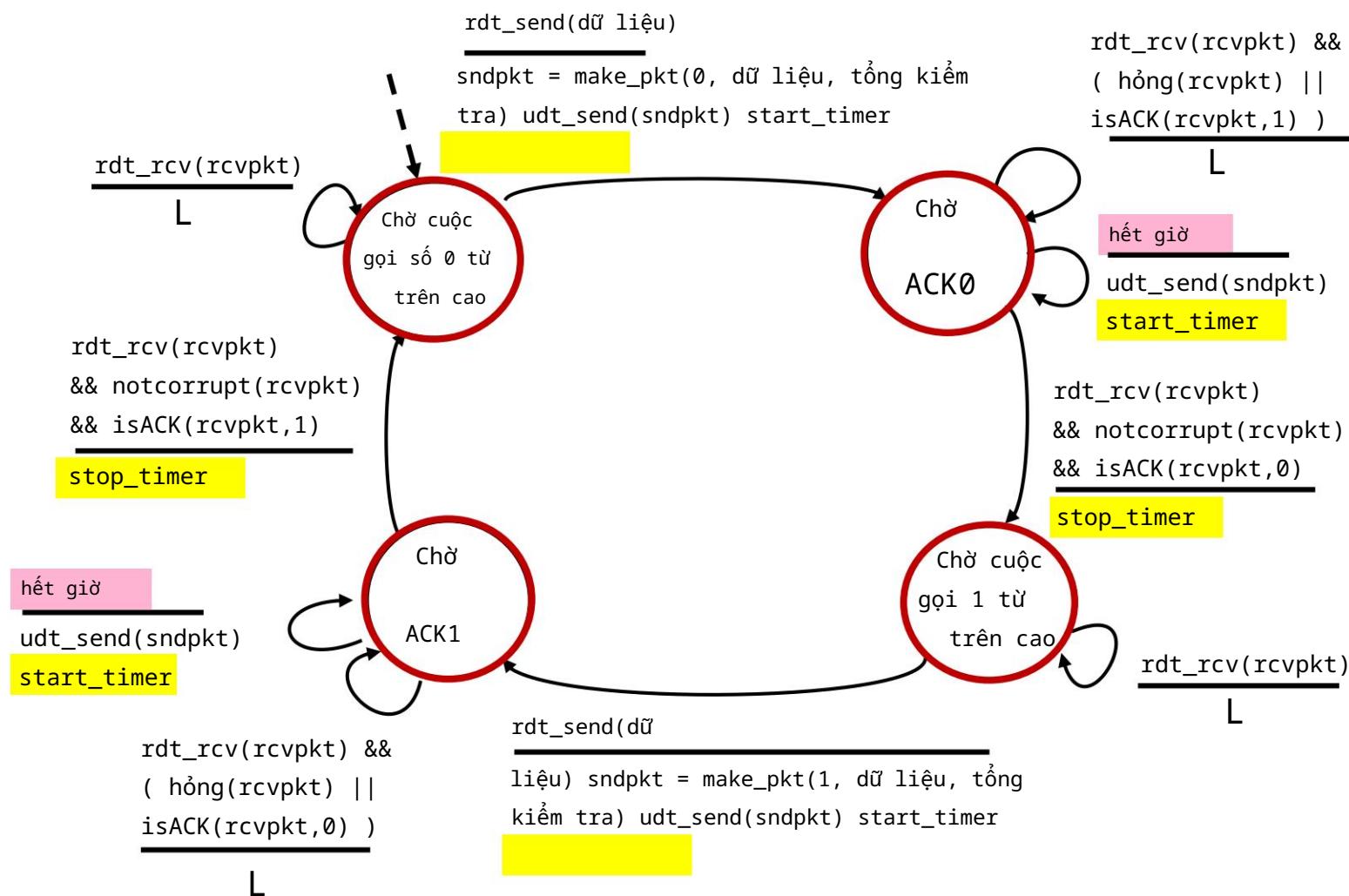
của thời gian



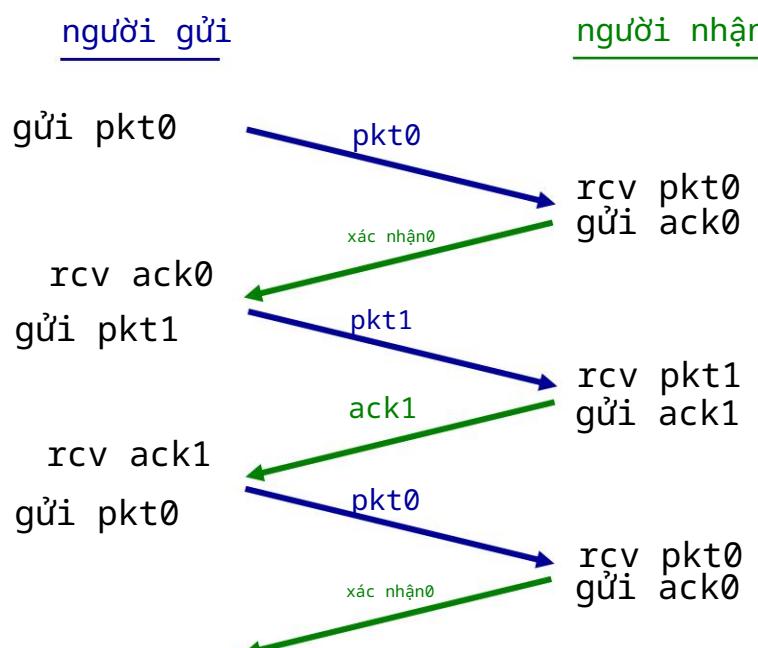
người gửi rdt3.0



người gửi rdt3.0

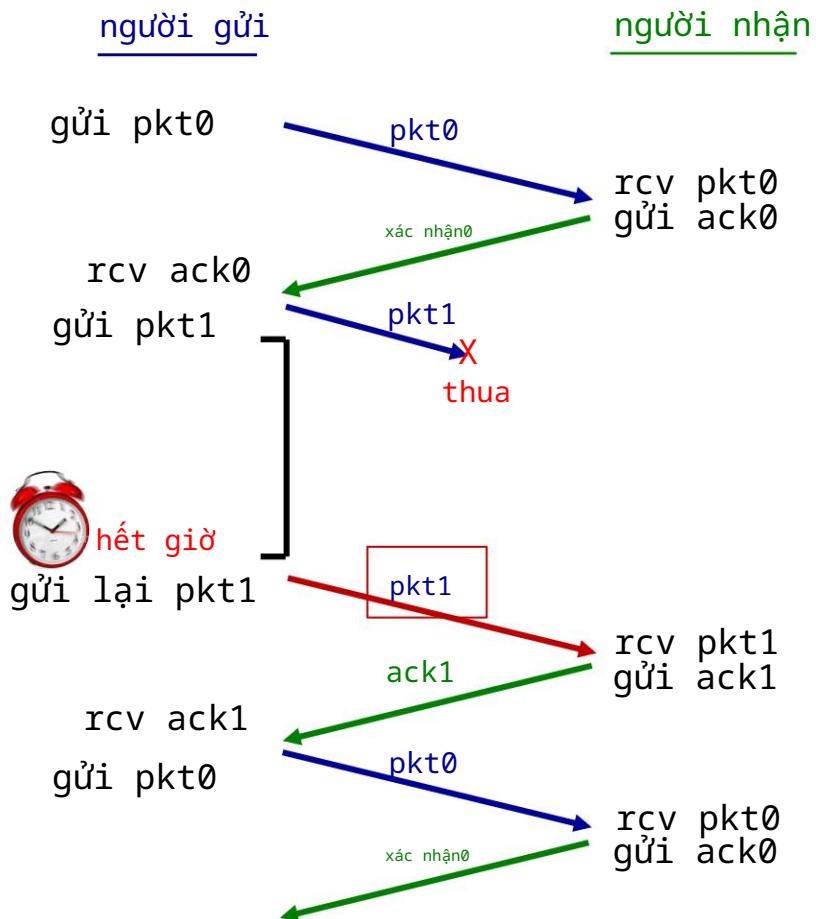


rdt3.0 đang hoạt động



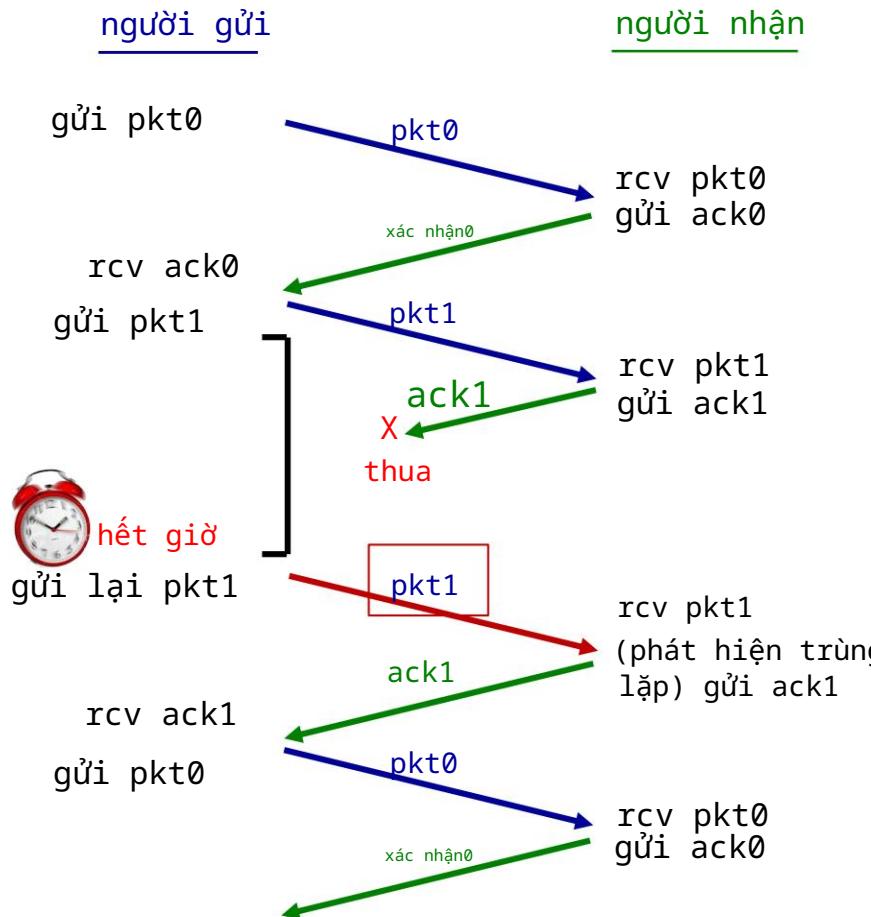
(a) không mất mát

dừng và đợi
người gửi gửi một gói, sau đó
đợi phản hồi của người nhận

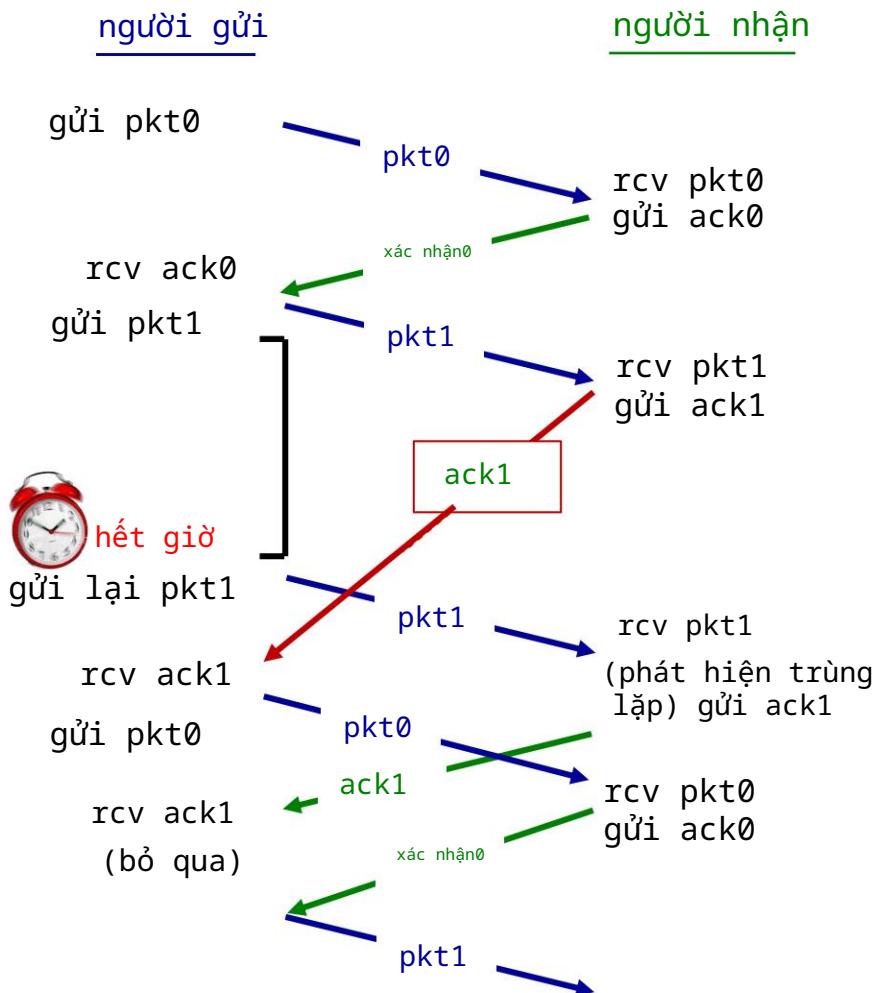


(b) m át gói tin

rdt3.0 đang hoạt động



(c) Mất ACK



(d) thời gian chờ quá sớm/ACK bị trì hoãn

Hiệu suất của rdt3.0 (dừng và chờ)

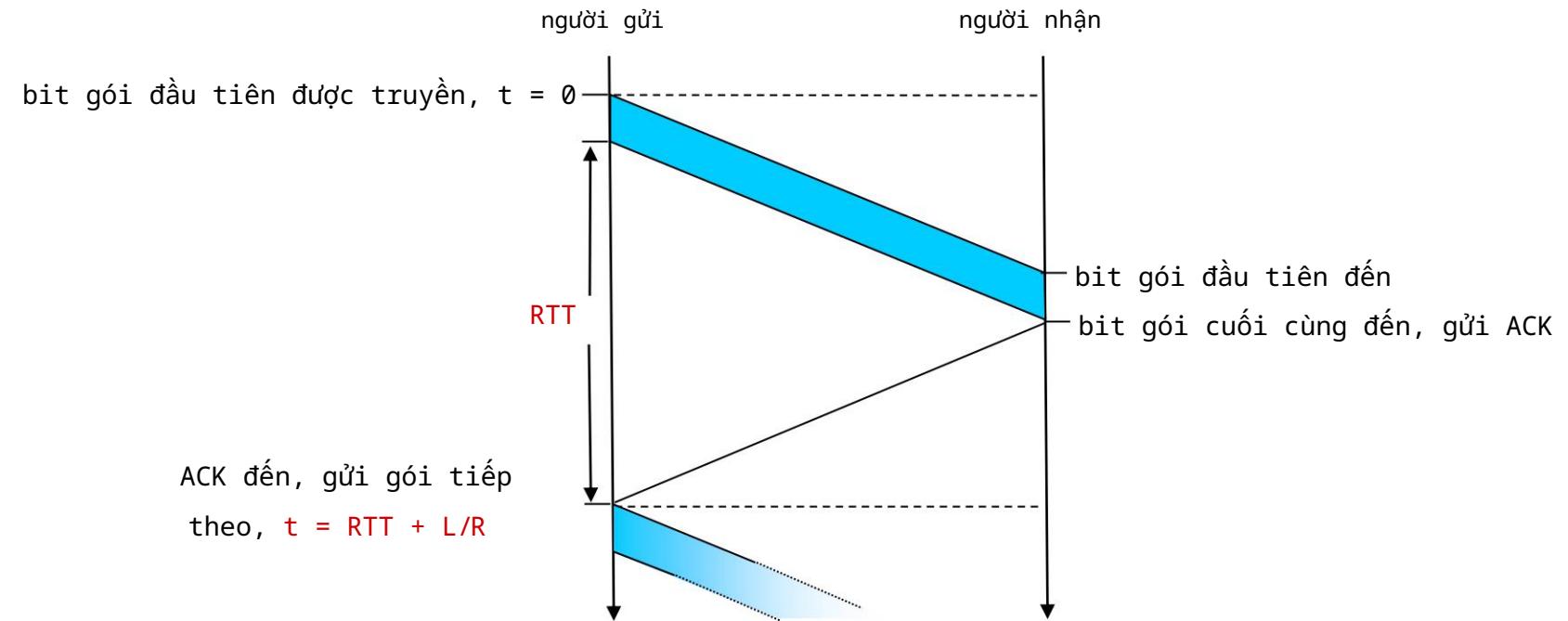
U người gửi: sử dụng - một phần thời gian **người gửi bận** gửi

ví dụ: liên kết 1 Gbps , giá đỡ 15 ms . độ trễ, gói 8000 bit

- thời gian truyền gói vào kênh: 8000

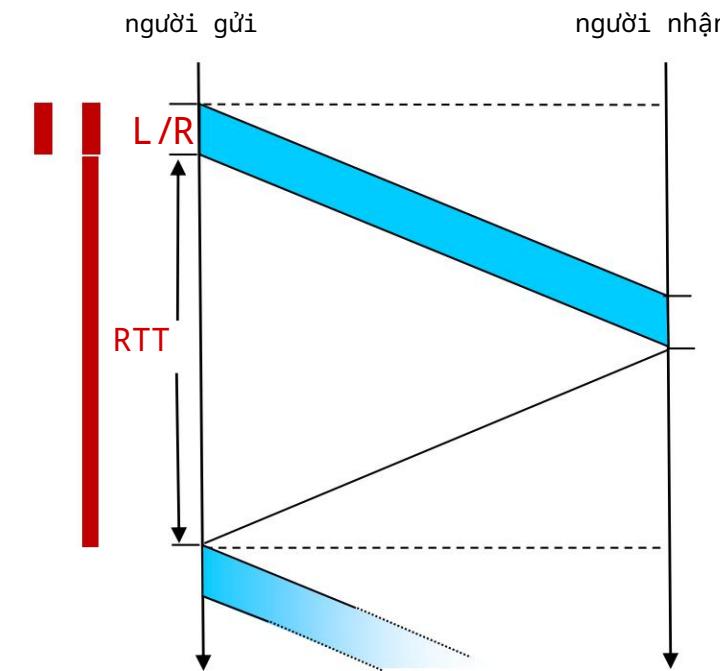
$$d_{trans} = \frac{L}{r} = \frac{\text{bit}}{\text{giây}} = \frac{109}{8 \text{ micro giây}}$$

rdt3.0: hoạt động dừng và chờ



rdt3.0: hoạt động dừng và chờ

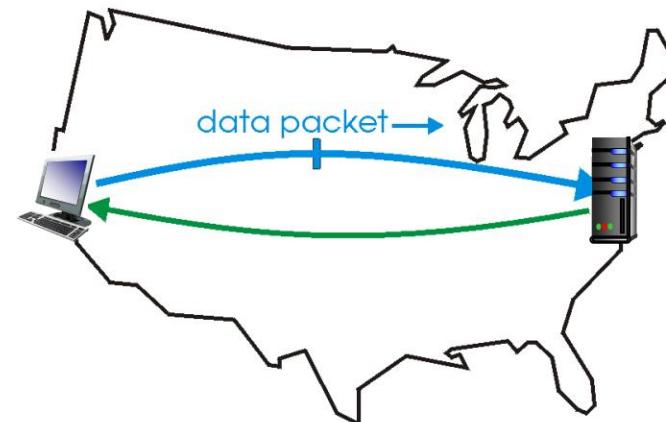
$$\begin{aligned}
 \text{Người dùng} &= \frac{L/R}{RTT + L/R} \\
 &= \frac{.008}{30.008} \\
 &= 0,00027
 \end{aligned}$$



Hiệu suất của giao thức rdt 3.0 rất tệ! Giao thức giới hạn hiệu suất của cơ sở hạ tầng bên dưới (kênh)

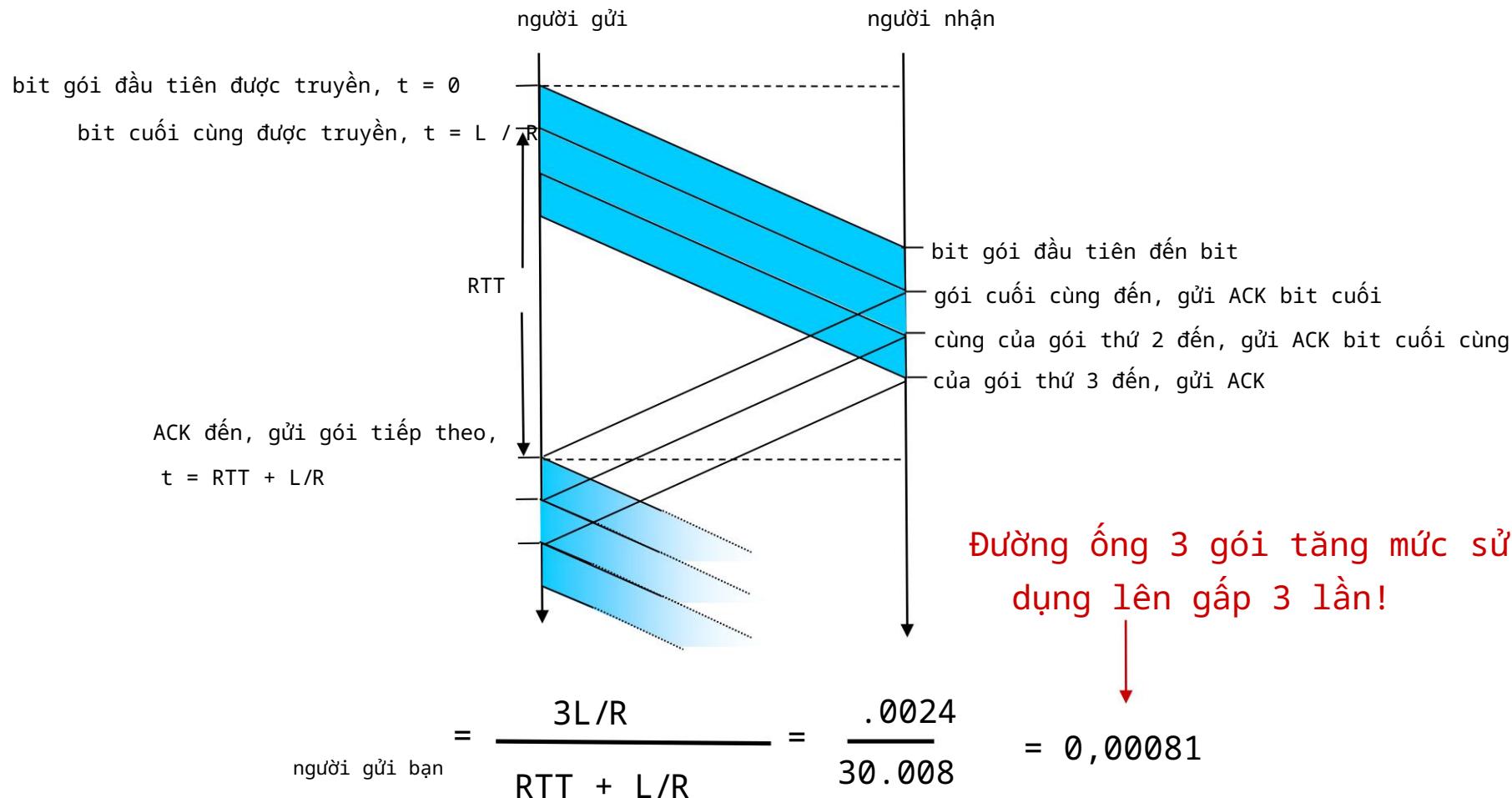
rdt3.0: hoạt động của giao thức đường ống

pipelining: người gửi cho phép nhiều gói, “trong chuyến bay”, các gói chưa được xác nhận, phạm vi số **thứ tự** phải được tăng lên • đệm tại người gửi và/ • hoặc người nhận



(a) a stop-and-wait protocol in operation

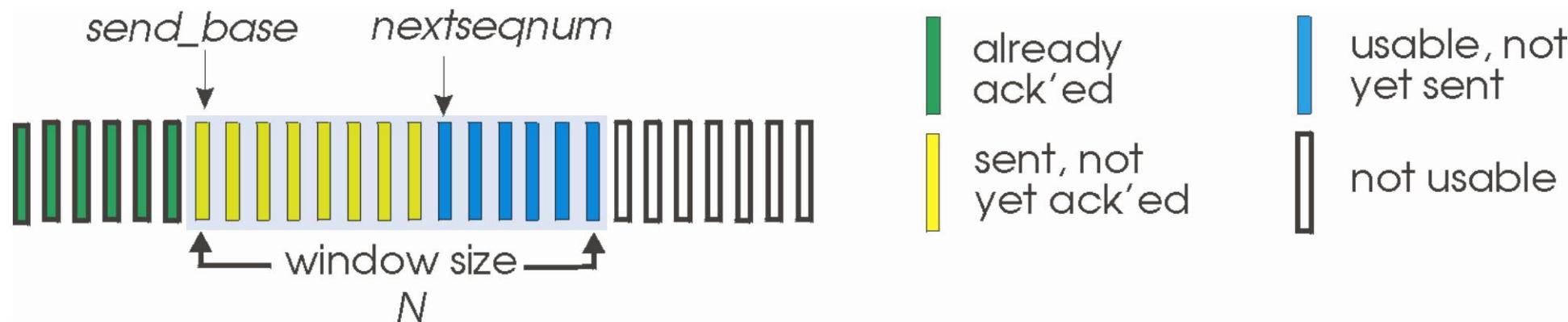
Đường ống: tăng cường sử dụng



Go-Back-N: người gửi

người gửi: “cửa sổ” lên đến N , các gói được truyền liên tiếp nhưng không được ACK

- k-bit seq # trong tiêu đề pkt



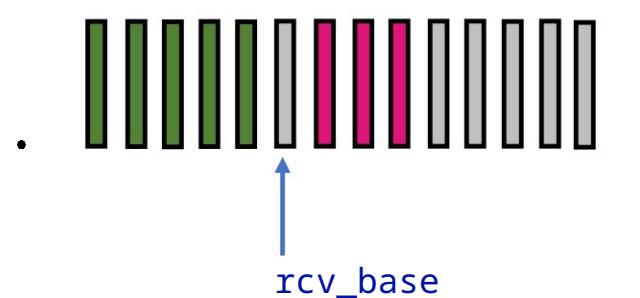
ACK tích lũy: $\text{ACK}(n)$: ACK tất cả các gói lên đến, bao gồm cả thứ tự # n • khi nhận được $\text{ACK}(n)$: di chuyển cửa sổ về phía trước để bắt đầu tại $n+1$ hẹn giờ cho gói trong chuyến bay cũ nhất hết thời gian (n): truyền lại gói n và tất cả các gói có số thứ tự cao hơn trong cửa sổ

Go-Back-N: máy thu

Chỉ ACK: luôn gửi ACK cho gói đã nhận chính xác cho đến nay, với seq # theo thứ tự cao nhất có thể tạo ACK trùng lặp • chỉ cần nhớ `rcv_base`

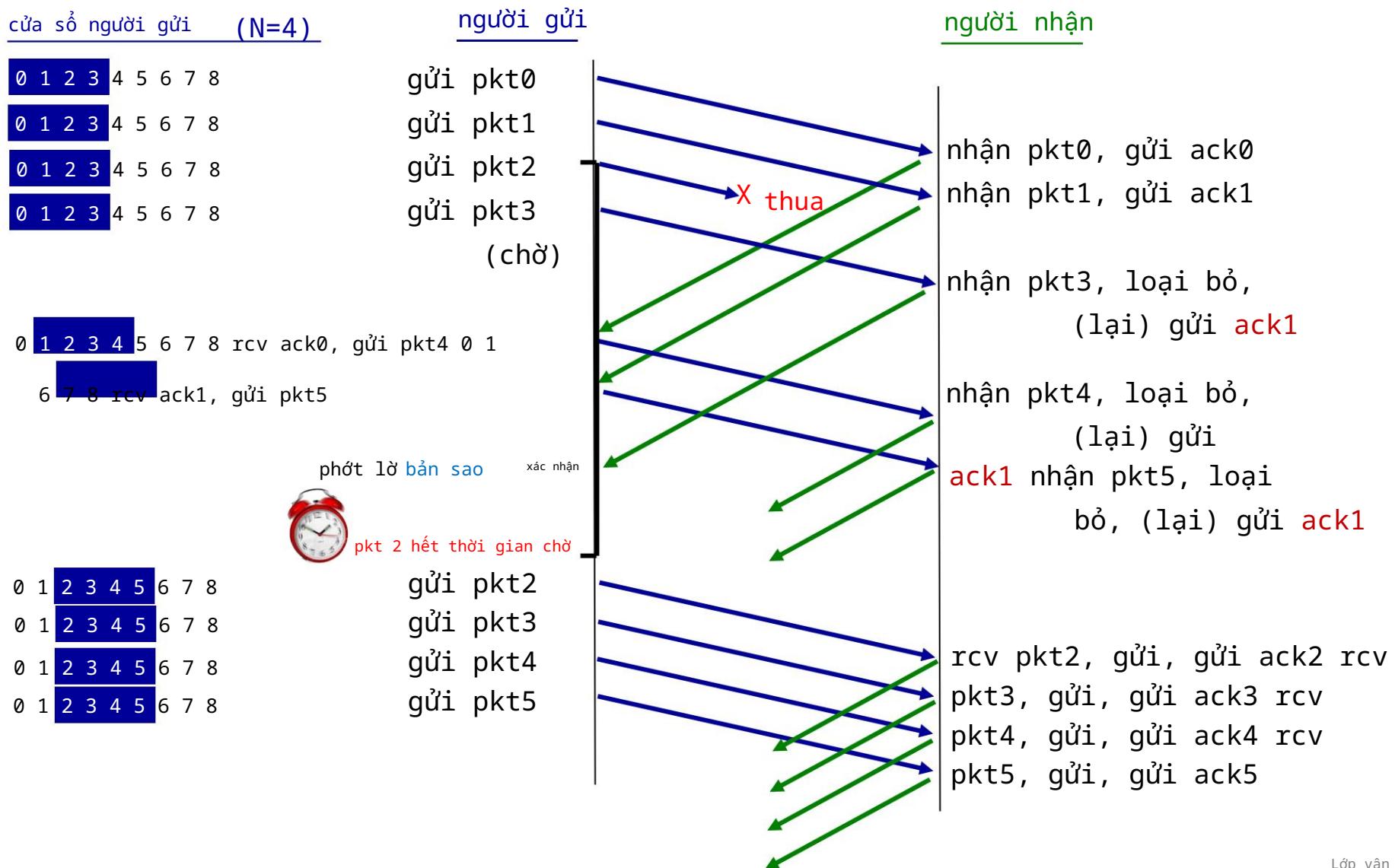
- khi nhận gói không theo thứ tự : • có thể loại bỏ (không đệm) hoặc bộ đệm: một quyết định triển khai • ACK lại pkt với thứ tự cao nhất seq #

Chế độ xem của người nhận về không gian số thứ tự:



| | |
|--|---------------------------------------|
| | đã nhận và ACK |
| | Out-of-order: đã nhận nhưng không ACK |
| | Không nhận |

Go -Back -N trong hành động



lặp lại có chọn lọc

người nhận xác nhận **riêng lẻ** tất cả các gói đã nhận chính xác • **đệm các** gói, nếu cần, **để phân phôi theo thứ tự cuối cùng** cho cấp trên

lớp

sender hết thời gian chờ/truyền lại **riêng lẻ** cho các gói chưa được ACK

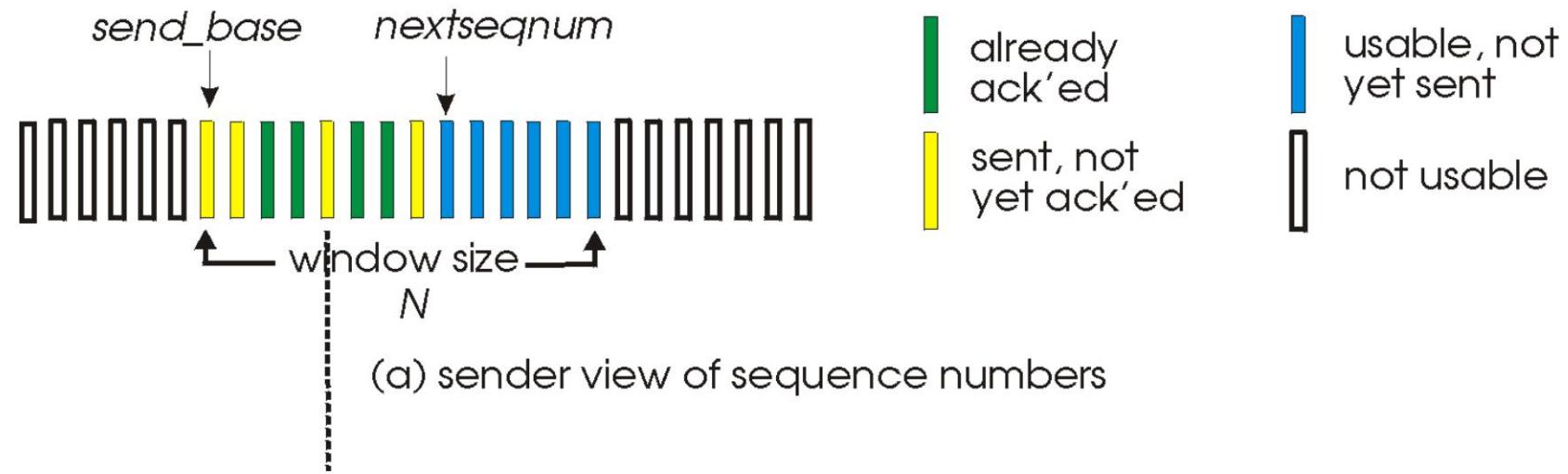
- người gửi duy trì **bộ đếm thời gian cho mỗi** cửa sổ người

gửi pkt chưa được ACK

- **N số** thứ tự liên tiếp •

giới hạn số thứ tự của các gói đã gửi, chưa được ACK

Lặp lại có chọn lọc: cửa sổ người gửi, người nhận



Lặp lại có chọn lọc: người gửi và người nhận

dữ liệu

người gửi từ phía trên:

nếu tiếp theo có sẵn seq #

trong cửa sổ, gửi gói thời gian

chờ (n):

gửi lại gói n, khởi động lại bộ hẹn giờ

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

đánh dấu gói n là đã nhận nếu

n gói chưa được ACK nhỏ nhất, nâng cấp
cơ sở cửa sổ tới chuỗi chưa được ACK

tiếp theo #

người nhận

gói n trong $[rcvbase, rcvbase+N-1]$ gửi ACK(n)

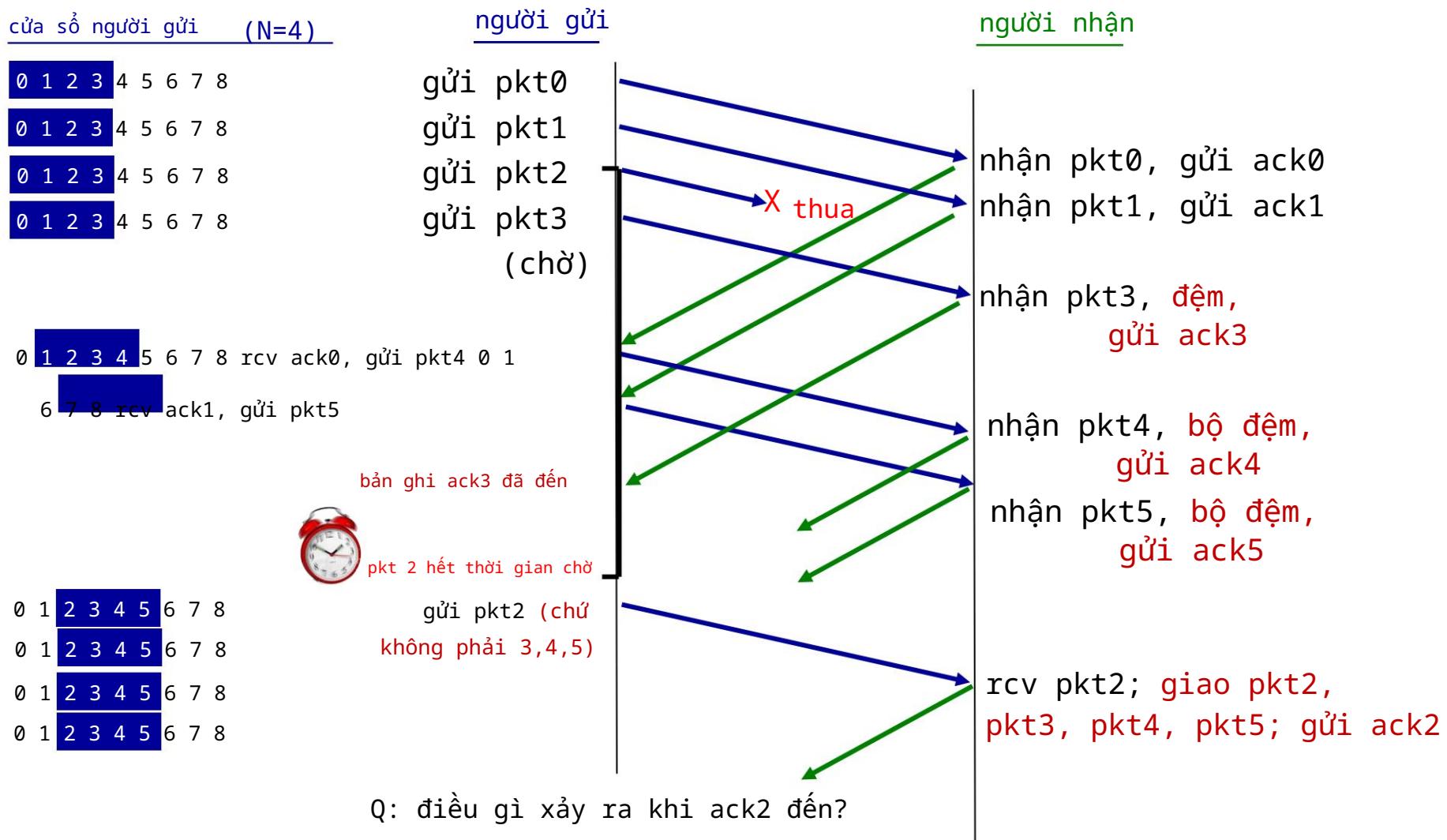
không đúng thứ tự: bộ đệm

theo thứ tự: phân phối (cũng phân phối
các gói được lưu vào bộ đệm, theo thứ
tự), cửa sổ chuyển tiếp tới gói chưa nhận
được tiếp theo

gói n trong $[rcvbase-N, rcvbase-1]$

ACK(n) nếu không: bỏ qua

Lặp lại có chọn lọc trong hành động



Lặp lại có chọn lọc:
tiến thoái lưỡng nan!

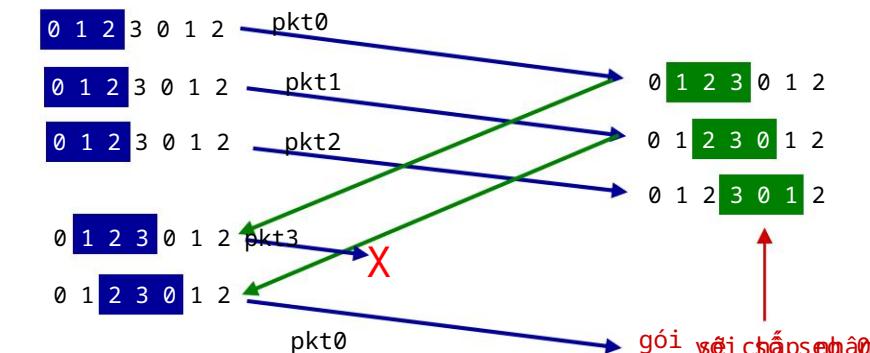
ví dụ:

seq #s: 0, 1, 2, 3 (đếm cơ số 4)

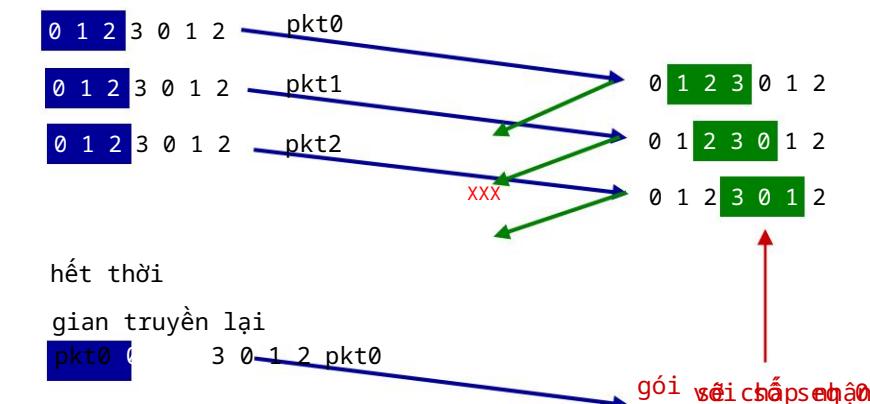
kích thước cửa sổ=3

cửa sổ người
gửi (sau khi nhận)

cửa sổ nhận (sau
khi nhận)



(a) không vấn đề gì



(b) ôi!

Lặp lại có chọn lọc: tiến thoái lưỡng nan!

ví dụ:

seq #s: 0, 1, 2, 3 (đếm cơ số 4)

kích thước cửa sổ=3

Câu hỏi: Cần có mối quan hệ nào
giữa **kích thước trình tự #** và
kích thước **cửa sổ** để tránh sự cố
trong kịch bản (b)?

cửa sổ người
gửi (sau khi nhận)

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 3 | 0 | 1 | 2 | |
| 0 | 1 | 2 | 3 | 0 | 1 | 2 |
| 0 | 1 | 2 | 3 | 0 | 1 | 2 |
| 0 | 1 | 2 | 3 | 0 | 1 | 2 |

pkt0
pkt1
pkt2
pkt3

người nhận không thẻ pkt0
nhìn thấy phía người gửi (a)

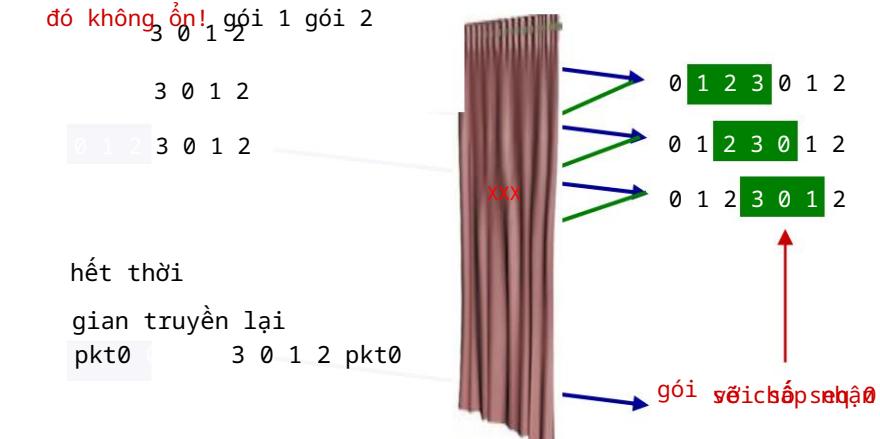
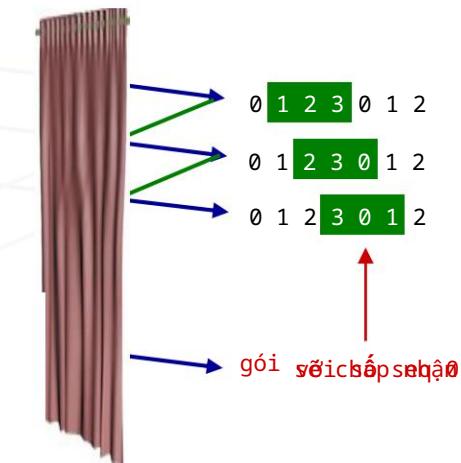
không vấn đề gì hành vi
của người nhận giống hệt
nhau trong cả hai trường
hợp! pkt0 (rất) có gì
đó không ổn! gói 1 gói 2

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 0 | 1 | 2 | | | |
| 0 | 1 | 2 | 3 | 0 | 1 | 2 |

hết thời
gian truyền lại
pkt0 3 0 1 2 pkt0

(b) ôi!

cửa sổ nhận (sau
khi nhận)



Chương 3: lộ trình

Dịch vụ tầng vận chuyển

Ghép kênh và phân kênh Vận
chuyển không kết nối: UDP Nguyên
tắc truyền dữ liệu đáng tin cậy

Vận chuyển hướng kết nối: TCP

- cấu trúc phân khúc •
truyền dữ liệu đáng tin cậy
- kiểm soát dòng chảy
- quản lý kết nối

Nguyên tắc kiểm soát tắc nghẽn

Kiểm soát tắc nghẽn TCP



TCP: tổng quan RFC: 793, 1122, 2018, 5681, 7323

điểm-điểm: • một

người gửi, một người nhận

luồng byte theo thứ tự,
đáng tin cậy :

- không có "ranh giới thông báo"

dữ liệu song công hoàn toàn: • luồng
dữ liệu hai chiều trong
cùng một kết nối

- **MSS:** kích thước đoạn tối đa

ACK tích lũy

đường ống: •

Kích thước cửa sổ đặt kiểm soát luồng
và tắc nghẽn TCP

hướng kết nối: • bắt tay (trao

đổi bản tin điều khiển) khởi tạo trạng thái
người gửi, người nhận trước khi trao đổi
dữ liệu

kiểm soát dòng chảy:

- người gửi sẽ không lấn át người nhận

Cấu trúc phân đoạn TCP

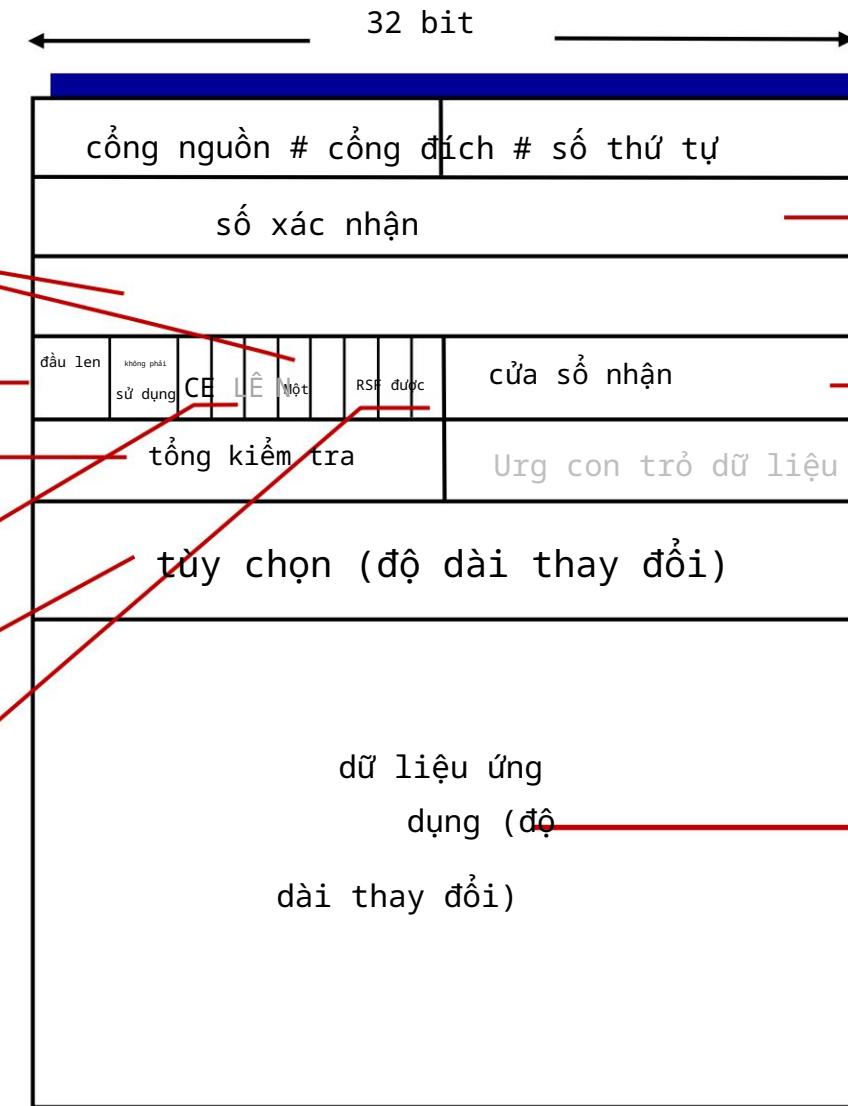
F: Hoàn thiện

ACK: thứ tự # của byte dự kiến tiếp theo; Một chút: đây là ACK

chiều dài (của tiêu đề TCP)
tổng kiểm tra Internet

C, E: thông báo tắc nghẽn

tùy chọn TCP
RST, SYN, FIN: quản lý kết nối



Seq # phân đoạn : đếm byte dữ liệu vào bytestream (không phải phân đoạn!)

điều khiển luồng: # byte người nhận sẵn sàng chấp nhận

dữ liệu được gửi bởi ứng dụng vào ổ cắm TCP

Số thứ tự TCP, ACK

Số thứ tự: • Dòng byte “số”

của byte đầu tiên trong dữ liệu của phân đoạn

Mỗi byte được đánh số thứ tự \rightarrow suy ra thứ tự của byte. . .

Lời cảm ơn: thứ tự # của -

- byte tiếp theo được mong đợi từ phía bên kia
- **ACK tích lũy** (gửi đúng những gì bên nhận muốn)

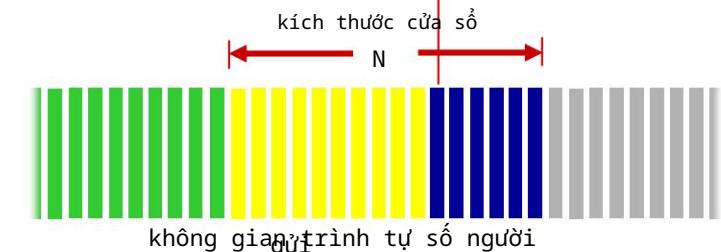
Hỏi: cách người nhận xử lý các phân đoạn

không theo thứ tự • A: Thông số kỹ thuật

TCP không nói, - tùy thuộc vào người triển khai

phân đoạn gửi đi từ người gửi

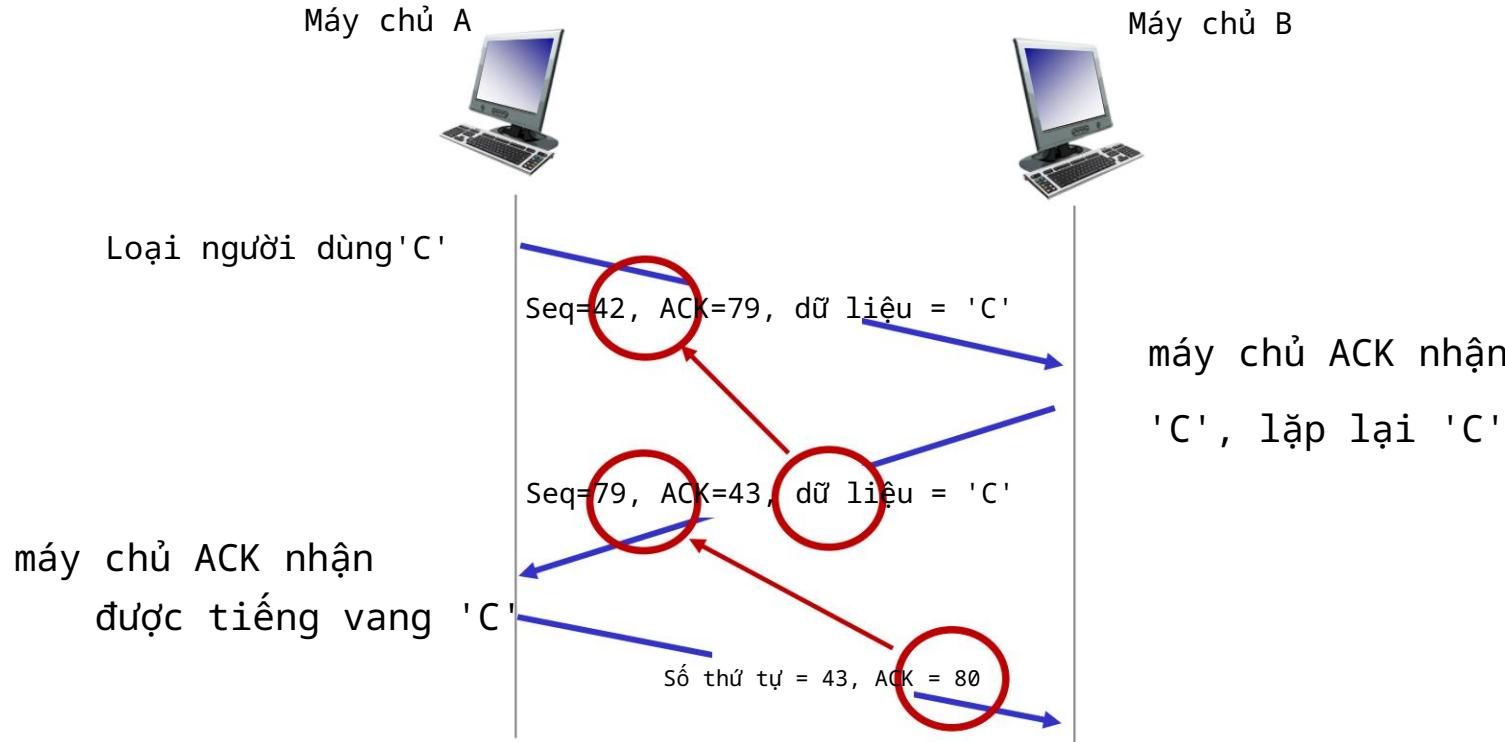
| | |
|------------------|------------------|
| cổng nguồn # | cảng đích # |
| số thứ tự số xác | |
| nhận rwnd | |
| | |
| tổng kiểm tra | con trỏ khẩn cấp |



phân đoạn đi từ máy thu

| | |
|---------------|------------------|
| cổng nguồn # | cảng đích # |
| số thứ tự số | |
| nhận rwnd | |
| | |
| tổng kiểm tra | con trỏ khẩn cấp |

Số thứ tự TCP, ACK



kịch bản telnet đơn giản

Thời gian khứ hồi TCP, thời gian chờ

Q: làm cách nào để đặt giá trị **thời gian chờ** TCP ?

dài hơn RTT, nhưng RTT khác nhau! **quá ngắn:** thời gian chờ quá sớm, truyền lại không cần thiết

quá dài: phản ứng chậm đối với mất đoạn

Q: làm thế nào để **ước tính RTT?**

SampleRTT: thời gian được đo từ khi truyền phân đoạn cho đến khi nhận được ACK • bỏ qua các lần truyền lại **SampleRTT** sẽ thay đổi, muốn RTT ước tính “mượt” hơn

- **trung bình** một số phép đo gần đây , không chỉ hiện tại **mẫuRTT**

Thời gian khứ hồi TCP, thời gian chờ

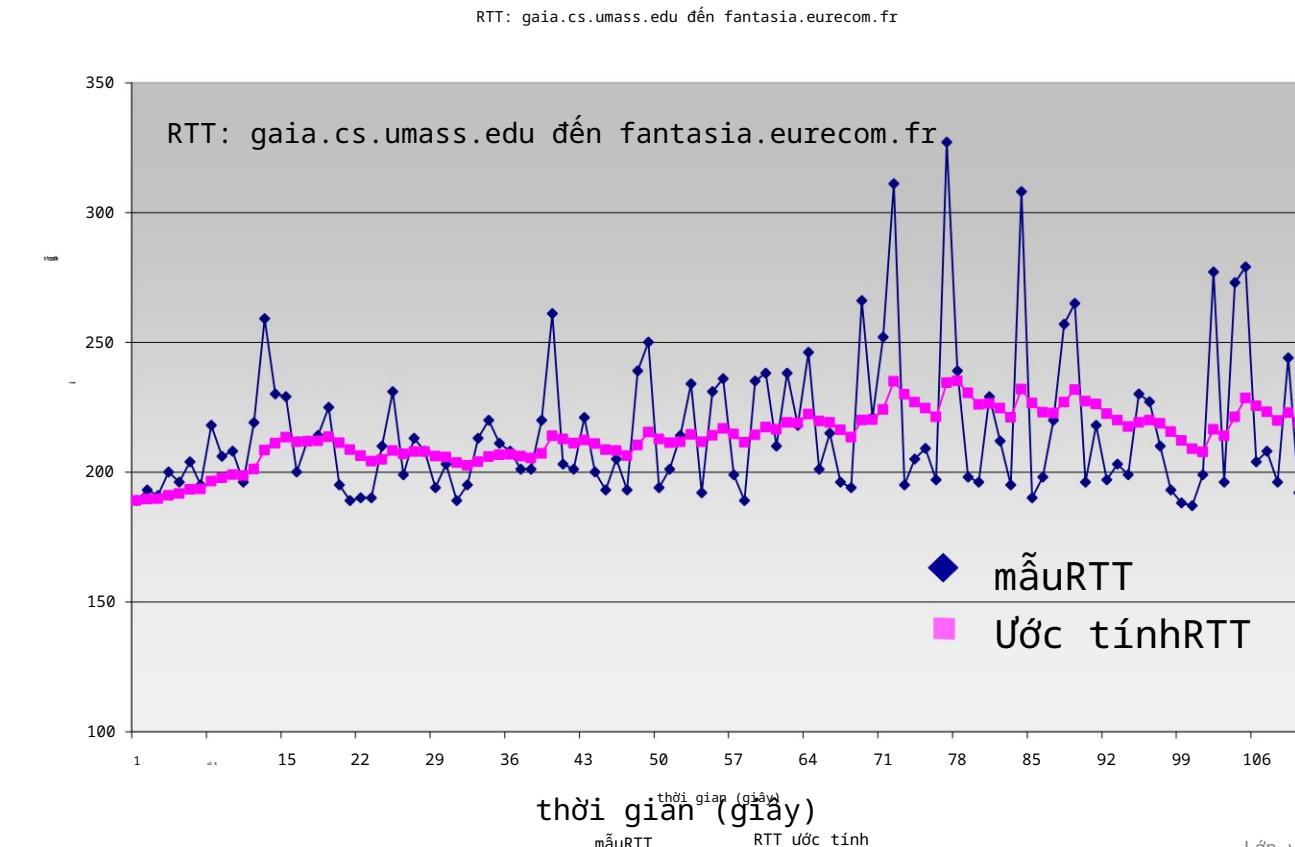
$$\text{Ước tínhRTT} = (1 - \alpha) * \text{Ước tínhRTT} + \alpha * \text{SampleRTT}$$

Đường trung bình động gia quyền theo cấp số nhân (EWMA)

Ảnh hưởng của mẫu trong quá khứ giảm nhanh theo cấp số nhân

Giá trị tiêu biểu: $\alpha = 0,125$

$\alpha = 0 \rightarrow e = \text{ước lượng}$
 $\alpha = 1 \rightarrow e = \text{reality}$



Thời gian khứ hồi TCP, thời gian chờ

hiện thư trong kernel hệ thống điều hành.

Cách khác: Thực thi TCP ở tầng ứng dụng. Được

Muốn thay đổi TCP -> có thể truy cập vào hệ điều hành -> Không chỉ thay đổi máy mình, phải thay đổi cả thế giới.

khoảng thời gian chờ: Ước tínhRTT cộng với “bờ an toàn”

- sự khác biệt lớn trong Ước tínhRTT: muốn có biên độ an toàn lớn hơn

$$\text{TimeOutInterval} = \text{Ước tínhRTT} + 4 * \text{DevRTT}$$



RTT ước tính

“bờ an toàn”

An toàn biên độ.

DevRTT: EWMA của SampleRTT sai lệch so với RTT ước tính:

$$\text{DevRTT} = (1 - \alpha) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(thông thường, $\alpha = 0,25$)

$\alpha = 0,25$, 0.125 đc chọn nhờ kinh nghiệm.

* Xem các bài tập tương tác trực tuyến để biết thêm ví dụ: http://gaia.cs.umass.edu/kurose_ross/interactive/

Người gửi TCP (đơn giản hóa)

sự kiện: dữ liệu nhận được
từ ứng dụng tạo phân
đoạn với seq #

seq # là số luồng byte của byte dữ
liệu đầu tiên trong phân đoạn **khởi**
động bộ đếm thời gian nếu chưa chạy • coi bộ
đếm thời gian như đối với phân đoạn chưa
được ACK **cũ nhất** • khoảng thời gian hết
hạn:

TimeOutInterval

sự kiện: hết giờ
truyền lại đoạn gây ra
thời gian chờ
hẹn giờ khởi động lại

sự kiện: Đã nhận được ACK
nếu ACK xác nhận các phân
đoạn chưa được ACK trước đó • **cập nhật**
những gì được biết là
đã xác nhận
• **bắt đầu hẹn giờ** nếu vẫn còn phân
đoạn chưa được ACK

Bộ thu TCP: Tạo ACK [RFC 5681]

Sự kiện tại người nhận

Hành động nhận TCP

sự xuất hiện của phân đoạn theo thứ tự với số thứ tự dự kiến. Tất cả dữ liệu cho đến thứ tự dự kiến # đã được ACK

ACK bị trì hoãn. Đợi tối đa 500 mili giây cho phân đoạn tiếp theo. Nếu không có phân đoạn tiếp theo, hãy gửi ACK

sự xuất hiện của phân đoạn theo thứ tự với số thứ tự dự kiến. Một phân khúc khác có ACK đang chờ xử lý

ngay lập tức gửi tích lũy duy nhất ACK, ACKing cả hai phân đoạn theo thứ tự

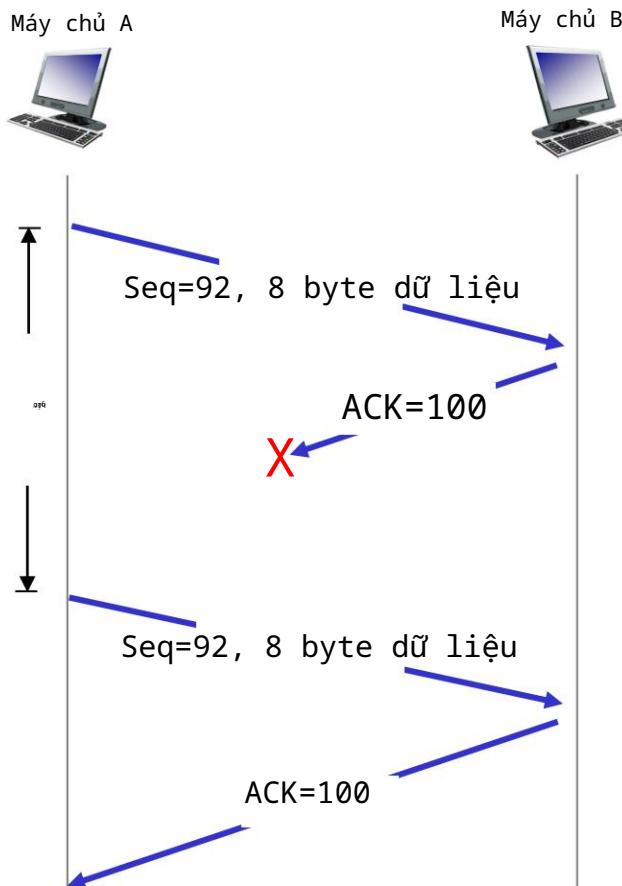
sự xuất hiện của phân khúc không theo thứ tự cao hơn mong đợi seq. # Phát hiện lỗi hỏng

ngay lập tức gửi **ACK trùng lặp**, cho biết seq. # byte dự kiến tiếp theo

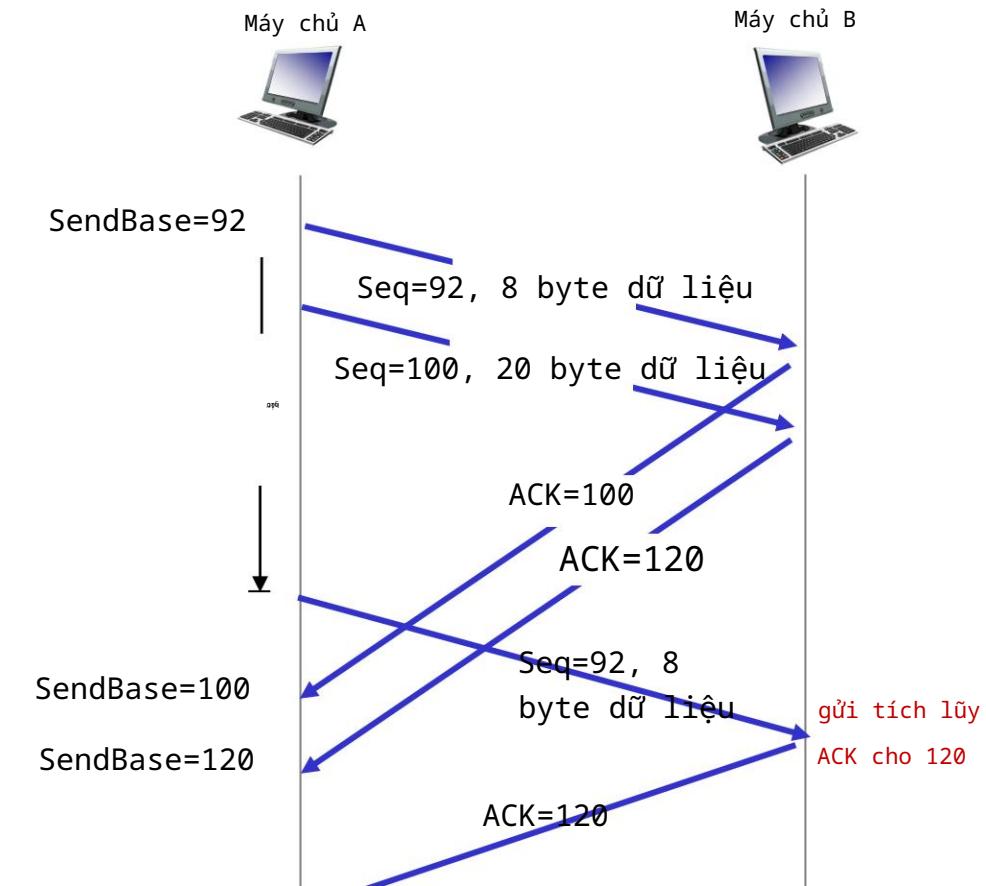
sự xuất hiện của phân khúc lấp đầy một phần hoặc hoàn toàn khoảng trống

gửi ACK ngay lập tức, với điều kiện là đoạn đó bắt đầu ở đầu thấp hơn của khoảng trống

TCP: kịch bản truyềnlại

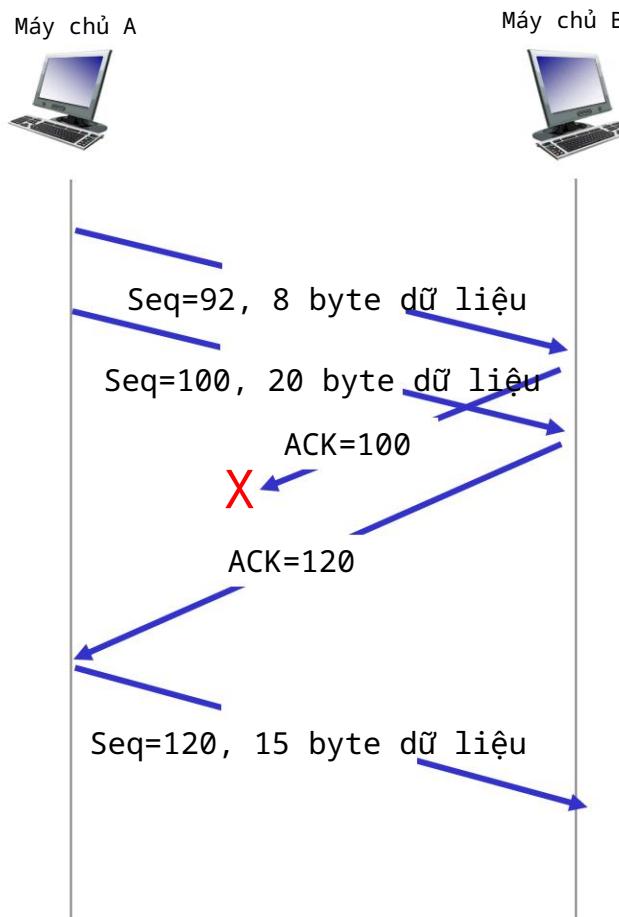


kịch bản mất ACK



thời gian chờ quá sớm

TCP: kịch bản truyềnlại



ACK tích lũy bao gồm ACK
bị mất trước đó

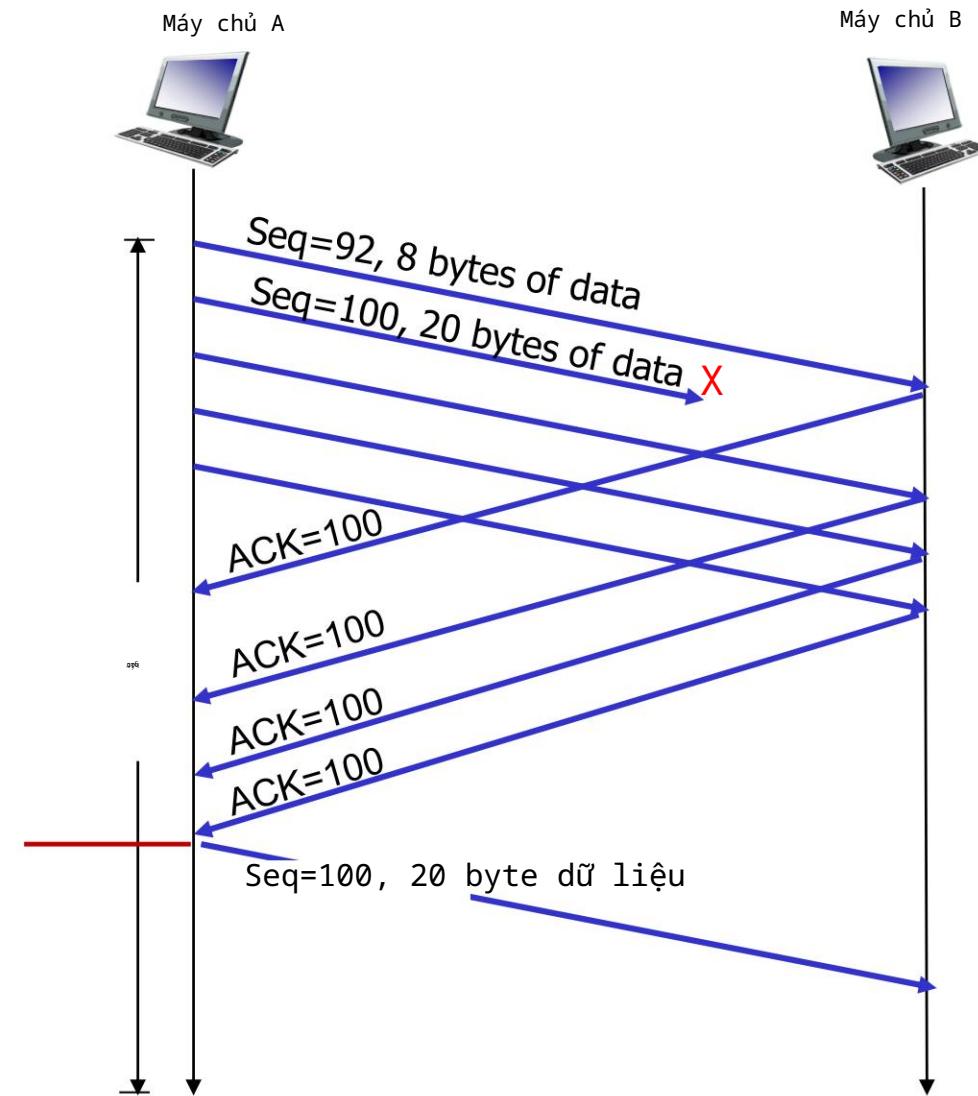
TCP truyền lại nhanh

truyền Nhập TCP

nếu người gửi nhận được **thêm 3 ACK** cho cùng một dữ liệu ("ba ACK trùng lặp"), gửi lại phân đoạn chưa được ACK **với số thứ tự nhỏ nhất** có khả năng phân đoạn chưa được ACK bị mất, vì vậy **đừng đợi hết thời gian chờ**



Việc nhận ba ACK trùng lặp cho biết đã nhận được 3 phân đoạn sau một phân đoạn bị thiếu - có khả năng bị mất phân đoạn. Vì vậy, **truyền lại!**



Chương 3: lô trình

Dịch vụ tầng vận chuyển

Ghép kênh và phân kênh Vận
chuyển không kết nối: UDP Nguyên
tắc truyền dữ liệu đáng tin cậy

Vận chuyển hướng kết nối: TCP

- cấu trúc phân khúc •
truyền dữ liệu đáng tin cậy

- kiểm soát dòng chảy
- quản lý kết nối

Nguyên tắc kiểm soát tắc nghẽn

Kiểm soát tắc nghẽn TCP

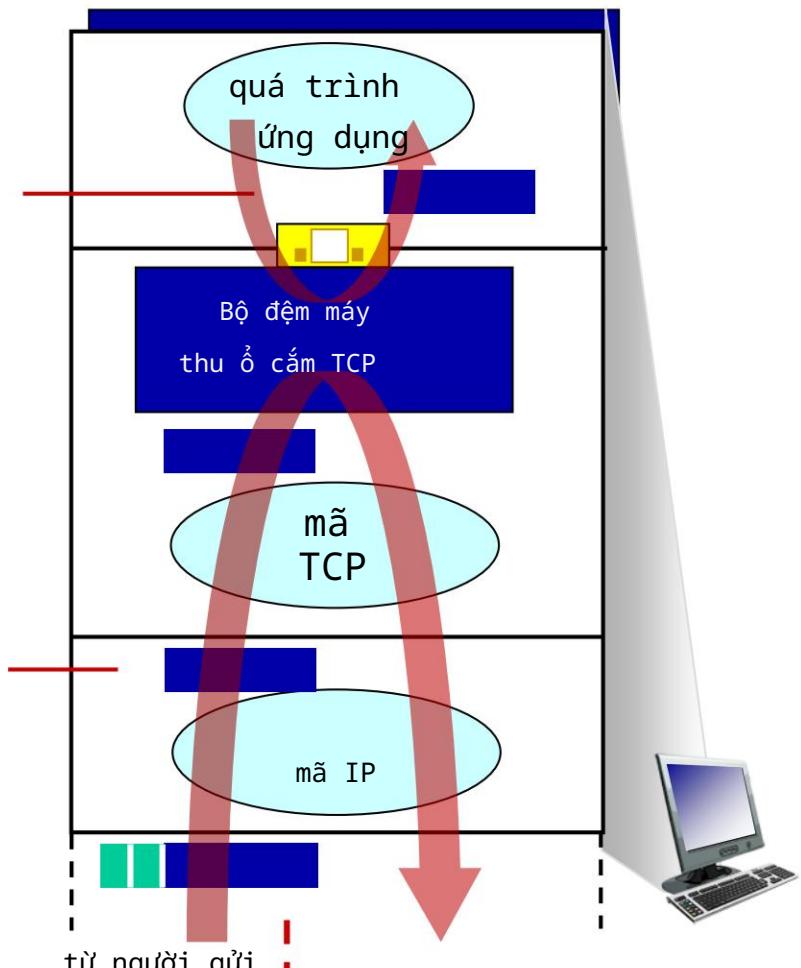


Kiểm soát luồng TCP

Hỏi: Điều gì xảy ra nếu lớp mạng cung cấp dữ liệu nhanh hơn lớp ứng dụng xóa dữ liệu khỏi ổ cắm bộ đệm?

Ứng dụng xóa dữ liệu khỏi bộ đệm ổ cắm TCP

Lớp mạng cung cấp tải trọng gói dữ liệu IP vào bộ đệm ổ cắm TCP



ngăn xếp giao thức máy thu

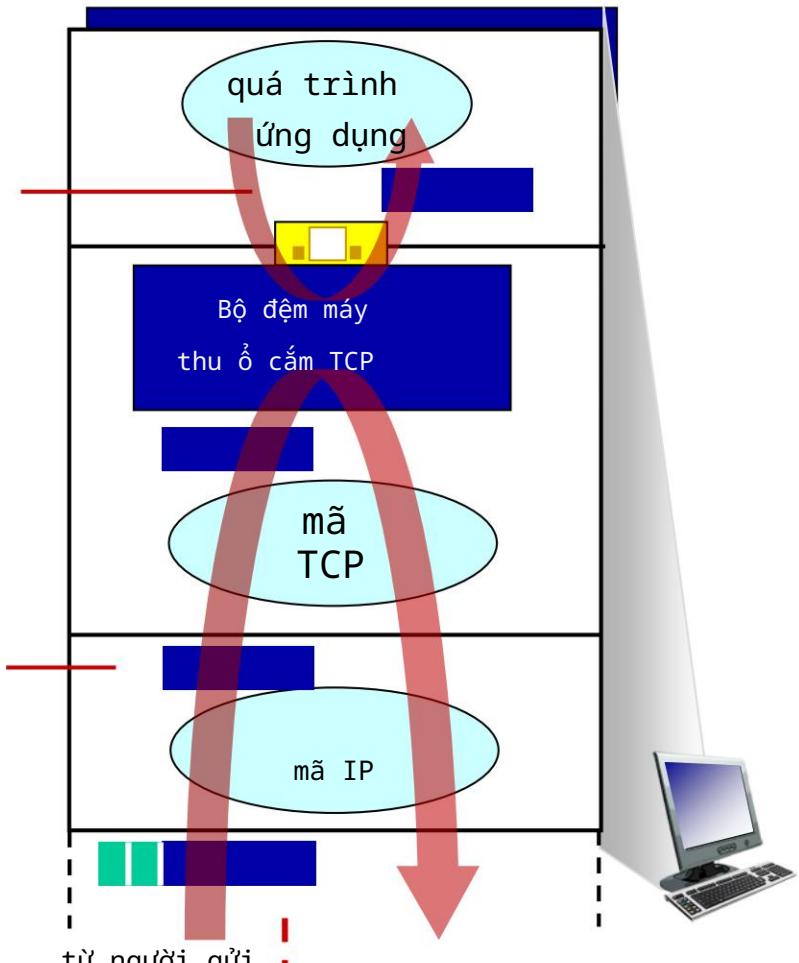
Kiểm soát luồng TCP

Câu hỏi: Điều gì xảy ra nếu lớp mạng cung cấp dữ liệu nhanh hơn lớp ứng dụng xóa dữ liệu khỏi bộ đệm ổ cắm?



Ứng dụng xóa dữ liệu
khỏi bộ đệm ổ cắm TCP

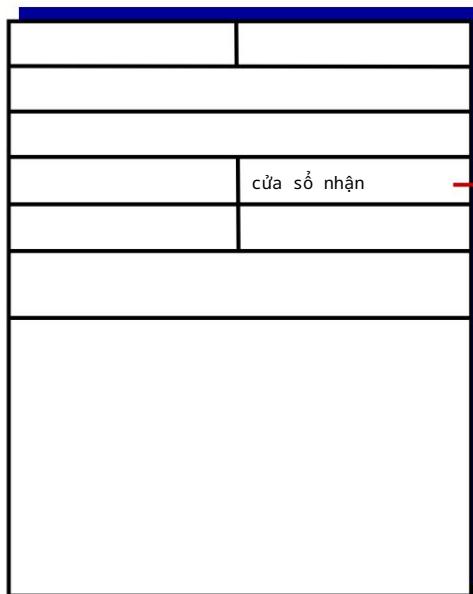
Lớp mạng cung
cấp tải trọng gói dữ
liệu IP vào bộ
đệm ổ cắm TCP



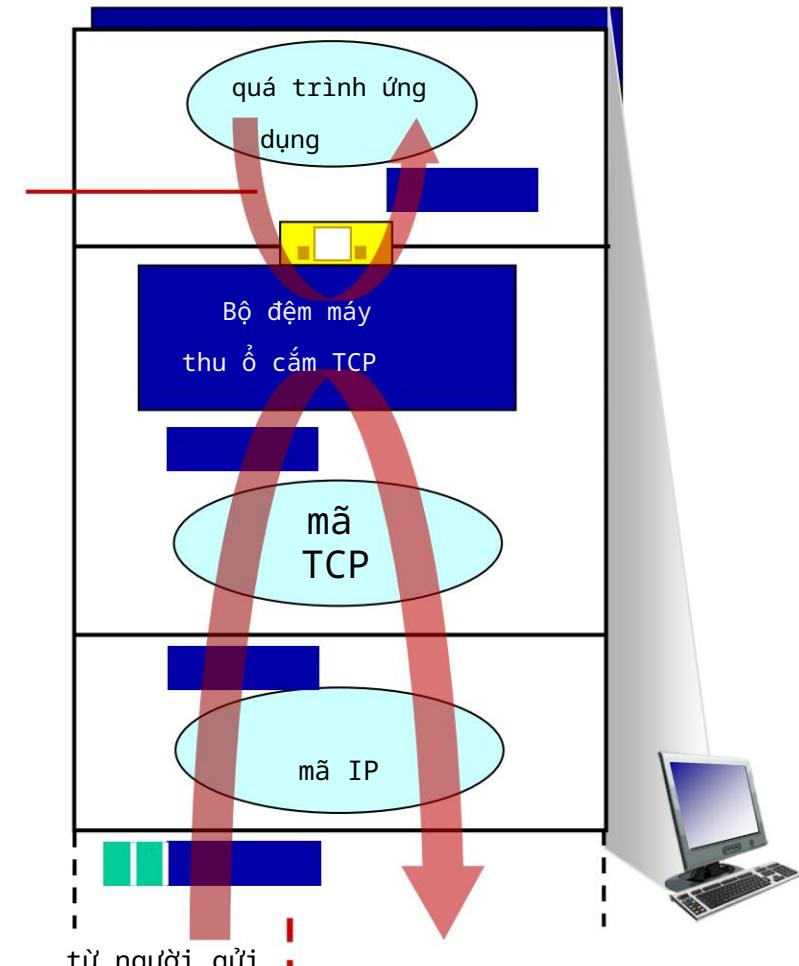
ngăn xếp giao thức máy thu

Kiểm soát luồng TCP

Câu hỏi: Điều gì xảy ra nếu lớp mạng cung cấp dữ liệu nhanh hơn lớp ứng dụng xóa dữ liệu khỏi bộ đệm ổ cắm?



Ứng dụng xóa dữ liệu
khỏi bộ đệm ổ cắm TCP



ngăn xếp giao thức máy thu

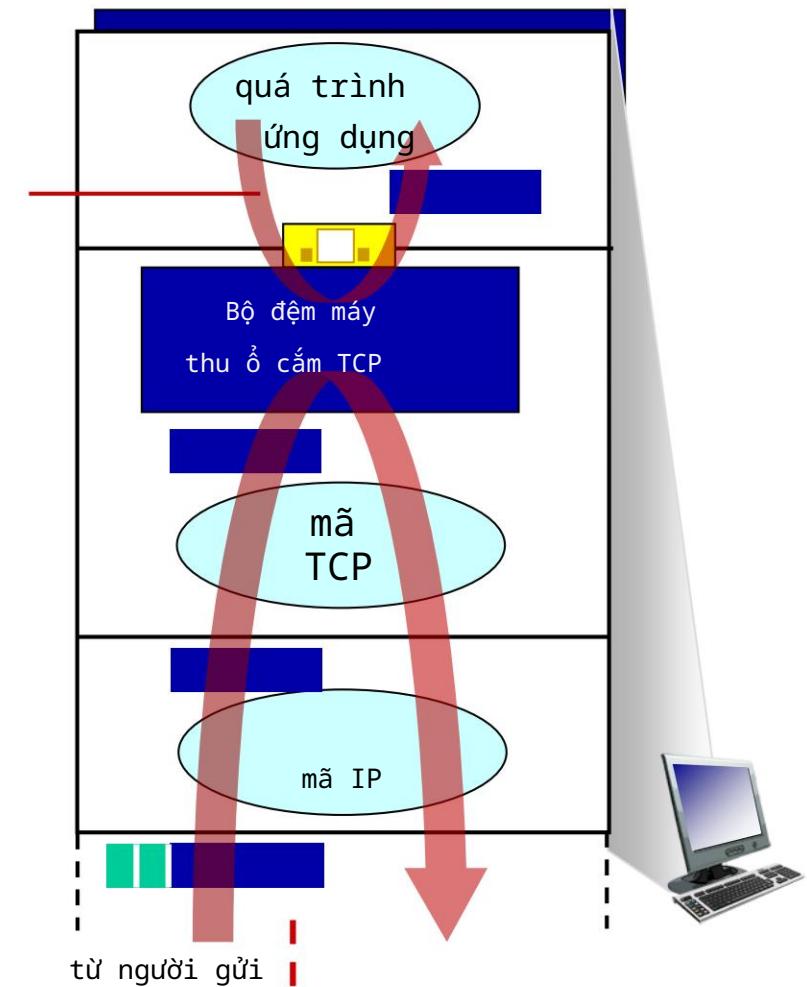
Kiểm soát luồng TCP

Câu hỏi: Điều gì xảy ra nếu lớp mạng cung cấp dữ liệu nhanh hơn lớp ứng dụng xóa dữ liệu khỏi bộ đệm ổ cắm?

kiểm soát luồng

bộ nhận kiểm soát bộ gửi, do đó, bộ gửi sẽ không làm tràn bộ nhớ đệm của bộ nhận do truyền quá nhiều, quá nhanh

Ứng dụng xóa dữ liệu khỏi bộ đệm ổ cắm TCP



ngăn xếp giao thức máy thu

Kiểm soát luồng TCP

Cơ chế: Biết trước kích thước của điểm đến để truyền dữ liệu.

Bộ thu TCP “quảng cáo” không gian bộ đệm trống

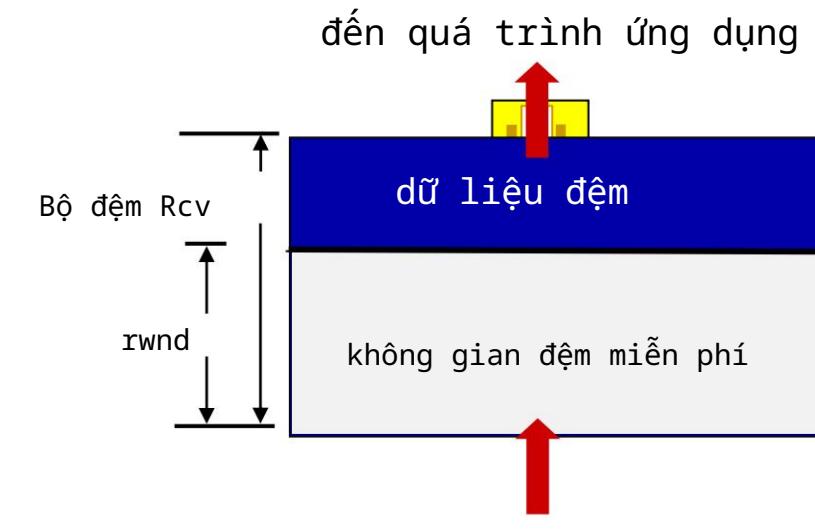
trong trường rwnd trong tiêu đề TCP

- Kích thước RcvBuffer được đặt qua ô cắm tùy chọn (mặc định điển hình là 4096 byte) nhiều hệ điều hành tự động điều chỉnh Bộ đệm Rcv

người gửi **giới hạn** số lượng không được ACK

dữ liệu (“trong chuyến bay”) đến rwnd đã nhận

đảm bảo bộ đệm nhận không bị tràn



Bộ đệm phía máy thu TCP

Tải trọng phân đoạn TCP

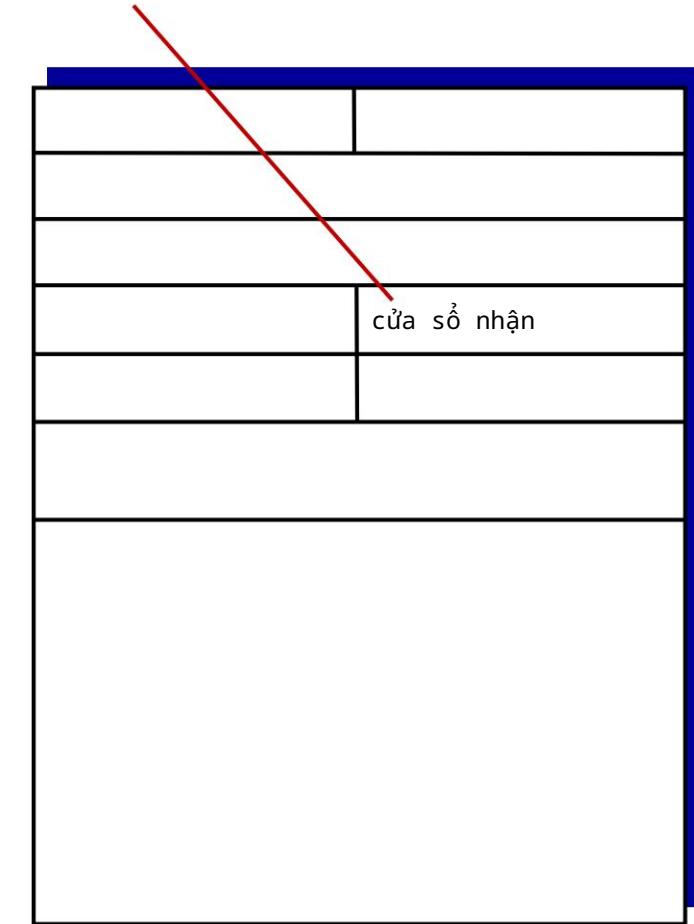
Kiểm soát luồng TCP

Bộ thu TCP “quảng cáo” không gian bộ đệm trống trong trường rwnd trong tiêu đề TCP

- Kích thước RcvBuffer được đặt qua ổ cắm tùy chọn (mặc định điển hình là 4096 byte)
- Nhiều hệ điều hành tự động điều chỉnh Bộ đệm Rcv

người gửi giới hạn số lượng không được ACK dữ liệu (“trong chuyến bay”) đến rwnd đã nhận đảm bảo bộ đệm nhận không bị tràn

điều khiển luồng: # byte người nhận sẵn sàng chấp nhận

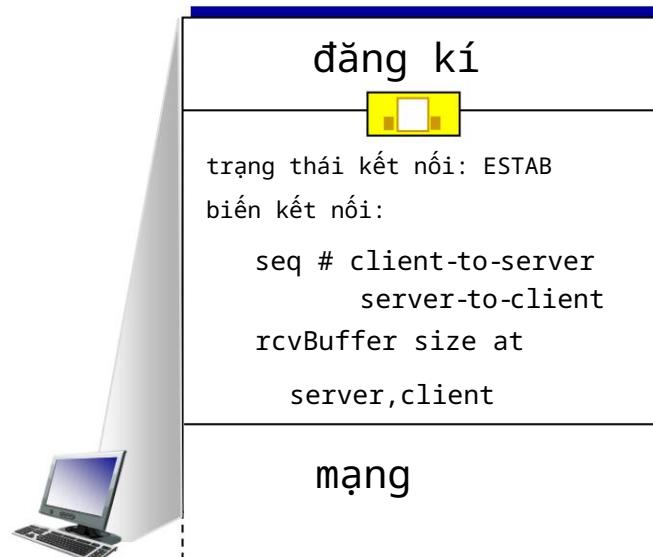


Định dạng phân đoạn TCP

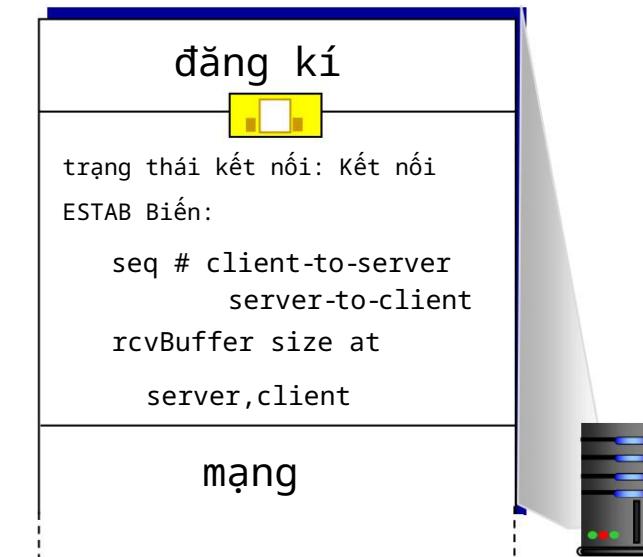
Quản lý kết nối TCP

trước khi trao đổi dữ liệu, người gửi/người nhận “[bắt tay](#)”:

đồng ý [thiết lập kết nối](#) (mỗi bên đều biết bên kia sẵn sàng thiết lập kết nối) đồng ý về [các tham số kết nối](#) (ví dụ: bắt đầu seq #s)



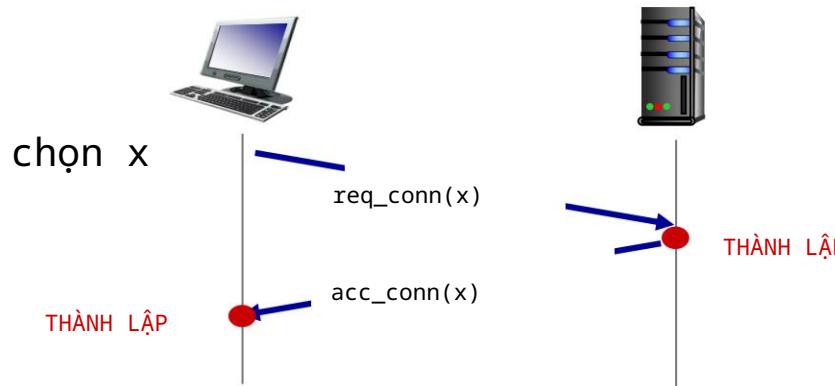
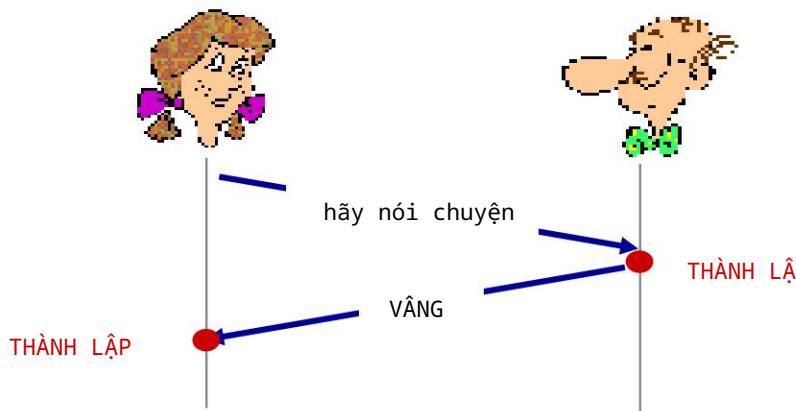
```
ở cắm clientSocket =
newSocket("tên máy chủ", "số cổng");
```



```
Kết nối ở cắmSocket =
welcomeSocket.accept();
```

Đồng ý thiết lập kết nối

Bắt tay 2 chiều:



Q: Bắt tay 2 chiều có luôn hoạt động trong mạng không?

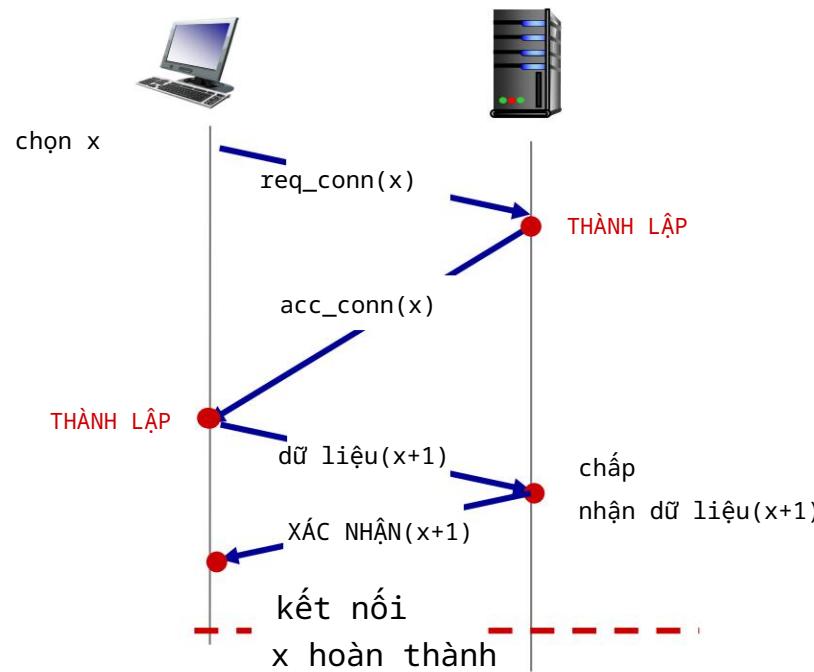
độ trễ có thể thay

đổi thông báo được truyền lại (ví dụ:
`req_conn(x)`) do mất tin nhắn

sắp xếp lại tin nhắn

không thể “nhìn thấy” phía bên kia

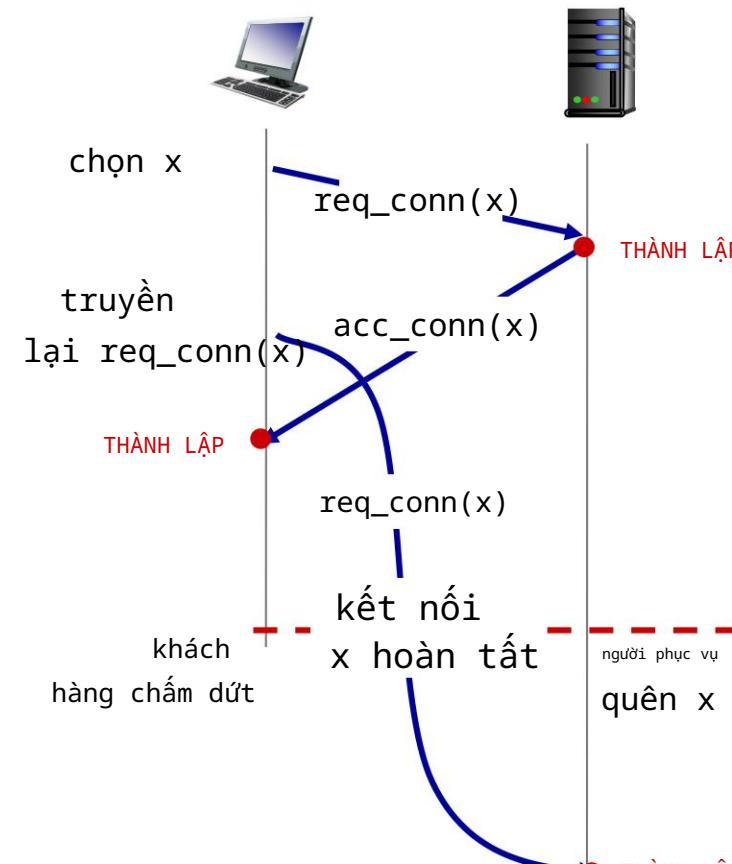
Kịch bản bắt tay 2 chiều



Không vấn đề gì!

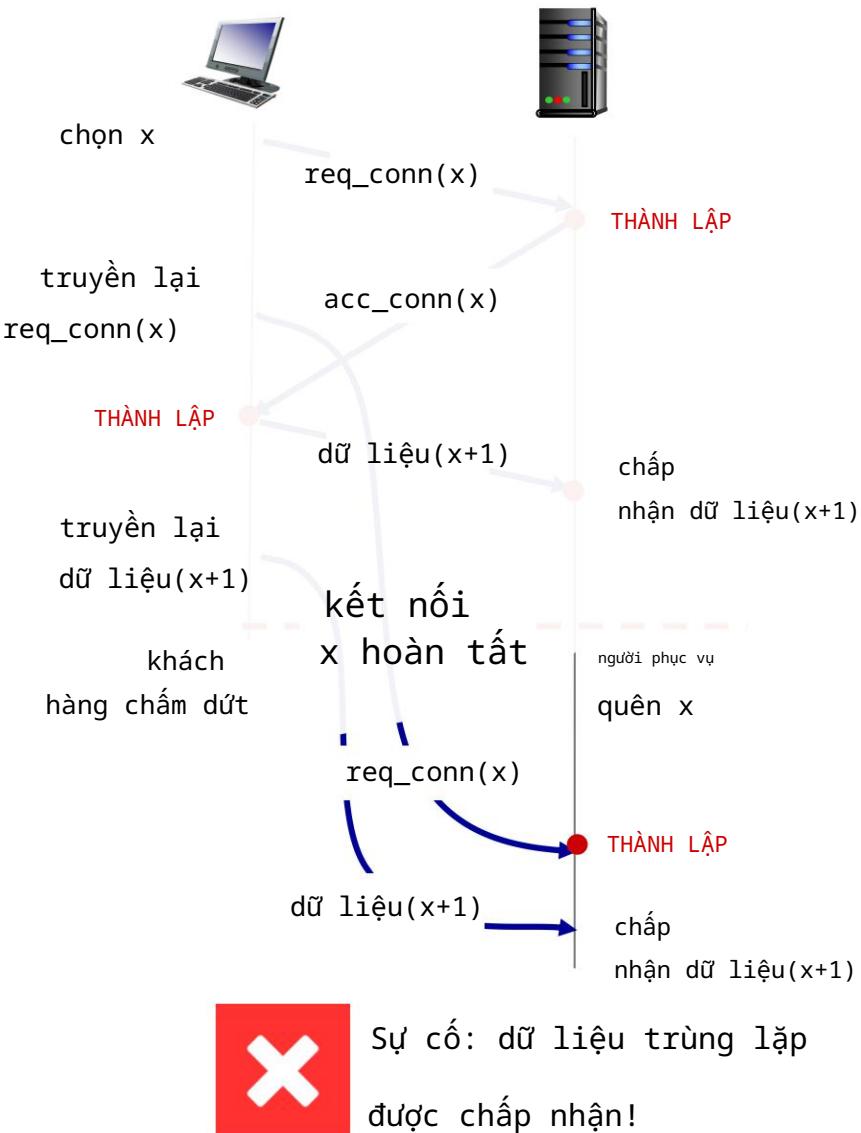


Kịch bản bắt tay 2 chiều



Vấn đề: `kết nối` mở một
nửa! (không có khách hàng)

Kịch bản bắt tay 2 chiều



Bắt tay 3 bước TCP

x and y is 2 con số ngẫu nhiên thuộc đoạn $[0;2^{32} - 1]$
 Data transfer through tcp ko giới hạn (tính theo hàm hash $(X + 1) \% 2^{32}$)

trạng thái máy chủ

```
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('',serverPort)) serverSocket.listen(1)
connectionSocket, addr = serverSocket.accept()
```

trạng thái khách hàng

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

NGHE

```
clientSocket.connect((serverName,serverPort))
```

TỔNG HỢP

chọn init seq num, x
gửi tin nhắn TCP SYN



THÀNH LẬP

nhận được SYNACK(x) cho
biết máy chủ đang hoạt
động; gửi ACK cho SYNACK;
phân khúc này có thể chứa dữ
liệu từ máy khách đến máy
chủ

chọn init seq num, y
gửi tin nhắn TCP SYNACK ,
acking SYN

NGHE

TỔNG HỢP RCVD

đã nhận được ACK(y)
cho biết máy khách đang hoạt động

THÀNH LẬP

Giao thức bắt tay 3 bước của con người



Đóng kết nối TCP

mỗi máy khách, máy chủ đóng phía kết nối của mình •

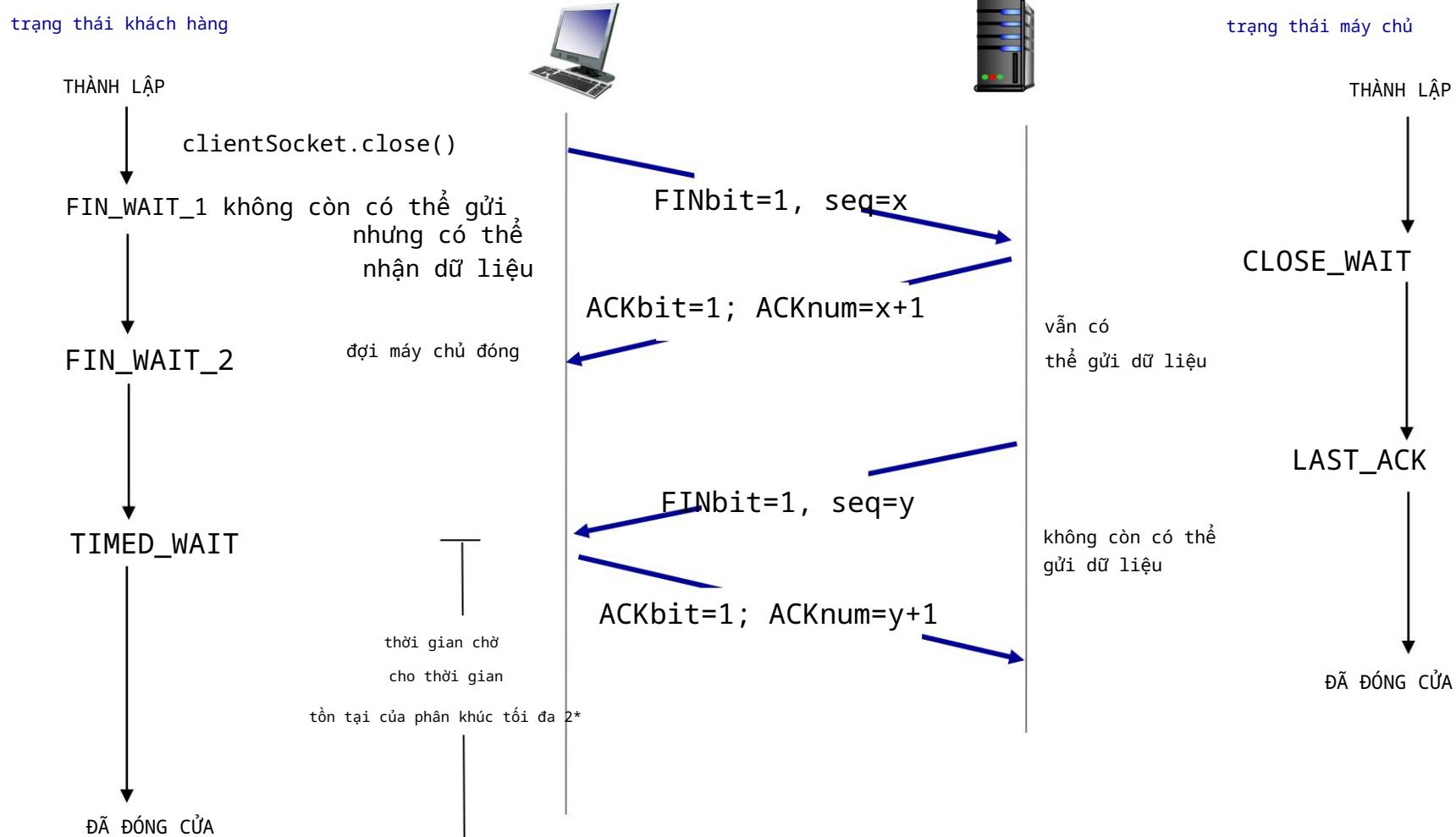
gửi phân đoạn TCP với bit FIN = 1 phản hồi FIN

nhận được bằng ACK • khi nhận được FIN, ACK có thể được
kết hợp với FIN riêng

trao đổi FIN đồng thời có thể được xử lý

Đóng kết nối TCP

always have XS loss data



Chương 3: lộ trình

Dịch vụ tầng vận chuyển
Ghép kênh và phân kênh Vận
chuyển không kết nối: UDP Nguyên
tắc truyền dữ liệu đáng tin cậy
Truyền tải hướng kết nối: TCP
Nguyên tắc kiểm soát tắc nghẽn Kiểm
soát tắc nghẽn TCP Sự phát triển
của chức năng tầng vận chuyển



Nguyên tắc kiểm soát tắc nghẽn

Tắc nghẽn:

một cách không chính thức: “quá nhiều nguồn gửi quá nhiều dữ liệu quá nhanh để mạng có thể xử lý”

biểu hiện:

- độ trễ dài (xếp hàng trong bộ đệm của bộ định tuyến)
- mất gói (tràn bộ đệm tại bộ định tuyến)

khác với kiểm soát dòng chảy!

một vấn đề top 10!



kiểm soát tắc nghẽn: quá nhiều người gửi, gửi quá nhanh

kiểm soát luồng: một người gửi quá nhanh cho một người nhận

Nguyên nhân/chi phí tắc nghẽn: kịch bản 1

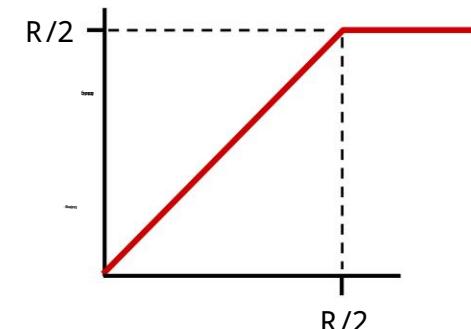
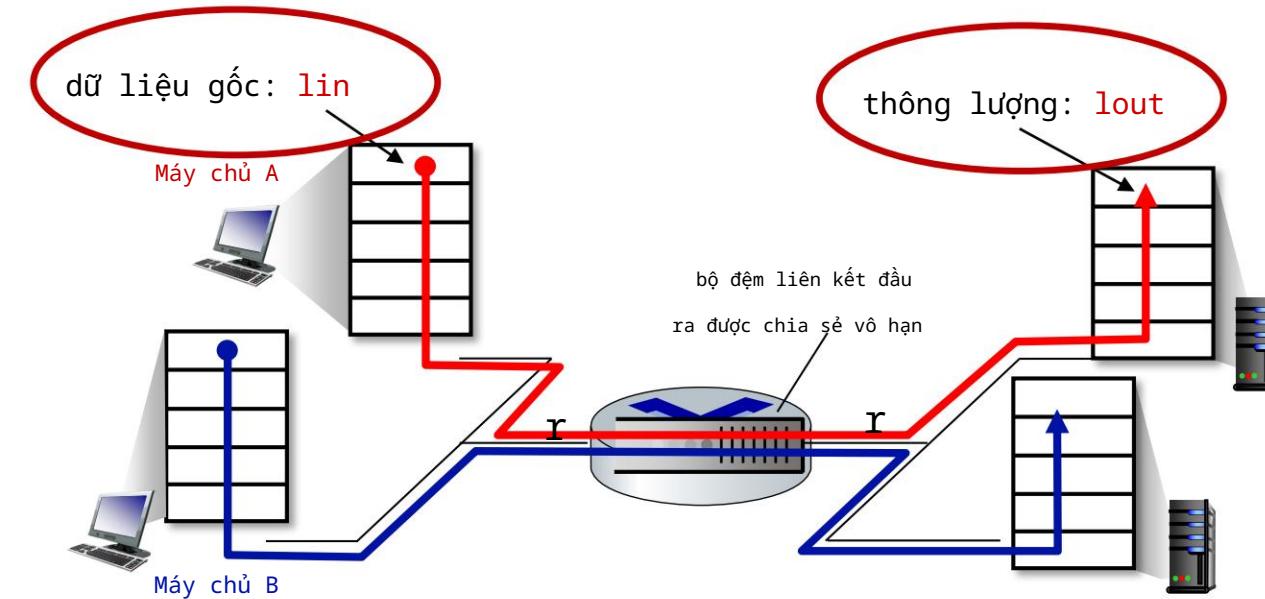
Kịch bản đơn giản nhất:

một bộ định tuyến, bộ đệm **vô hạn** không cần truyền lại

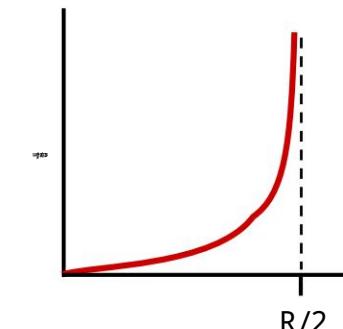
dung lượng liên kết đầu vào, đầu ra:

R hai luồng: khá, $R/2$

Q: Điều gì xảy ra
khi tỷ lệ đến
Trong gần $R/2$?



lin thông lượng tối đa
trên mỗi kết nối: $R/2$



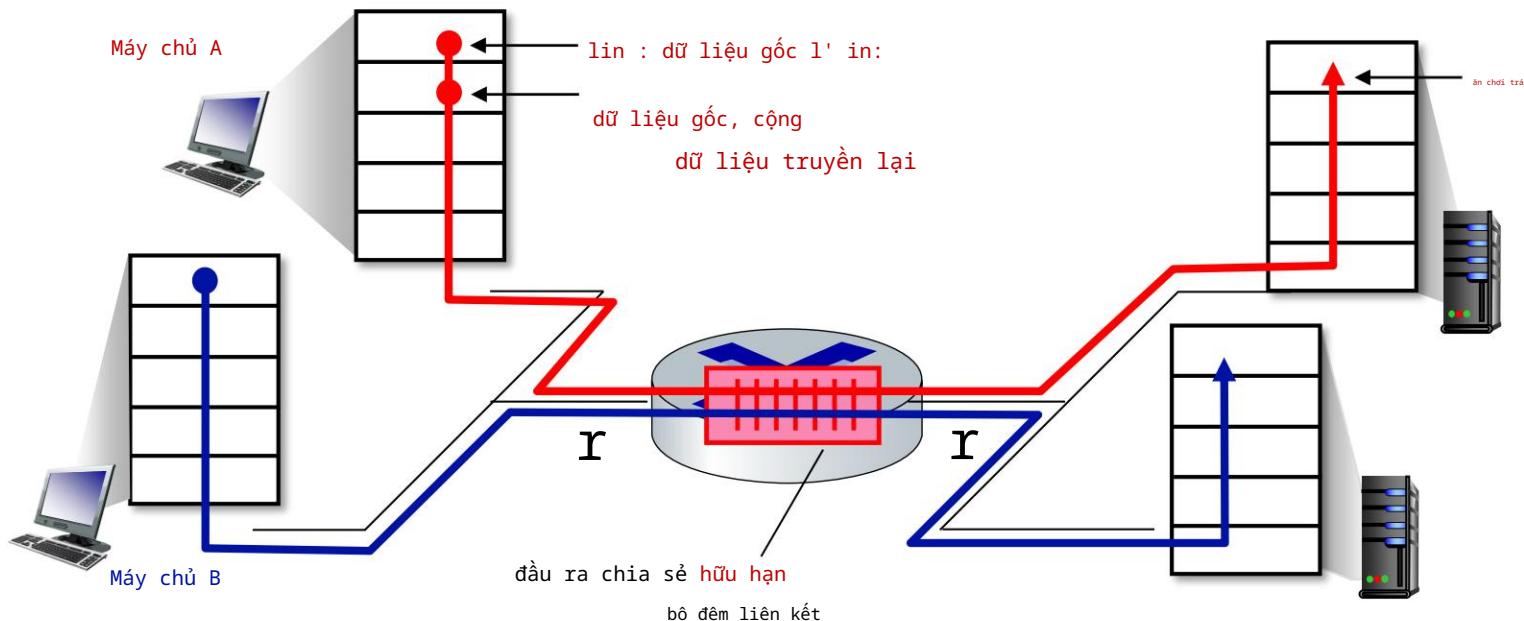
lin **chậm trễ** lớn khi tỷ
lệ đến lin đạt đến công suất

Nguyên nhân/chi phí tắc nghẽn: kịch bản 2

một bộ định tuyến, bộ đệm hữu

hạn bên gửi **truyền lại** gói bị mất, hết thời gian chờ

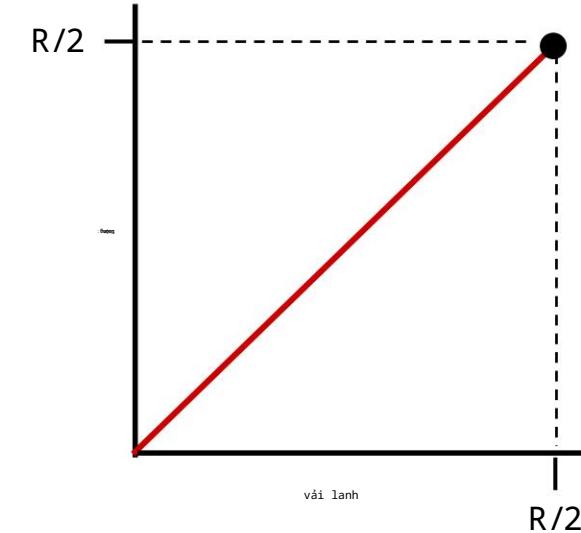
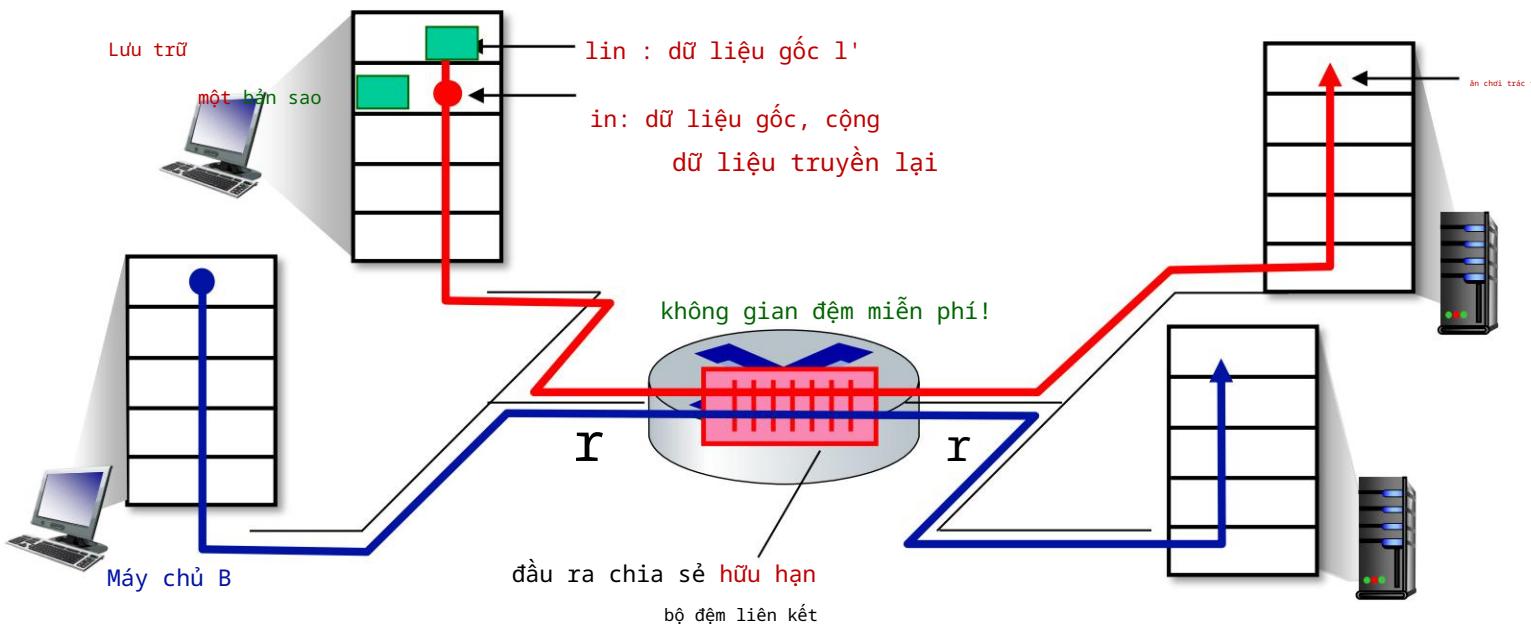
- đầu vào của tầng ứng dụng = đầu ra của tầng ứng dụng: • , $t_{\text{trong}} = t_{\text{ngoài}}$
- đầu vào của tầng vận chuyển bao gồm các lần **truyền lại** : $t_{\text{trong}} \geq t_{\text{ngoài}}$



Nguyên nhân/chi phí tắc nghẽn: kịch bản 2

Lý tưởng hóa: kiến thức hoàn hảo

người gửi chỉ gửi khi bộ định tuyến có sẵn bộ đệm

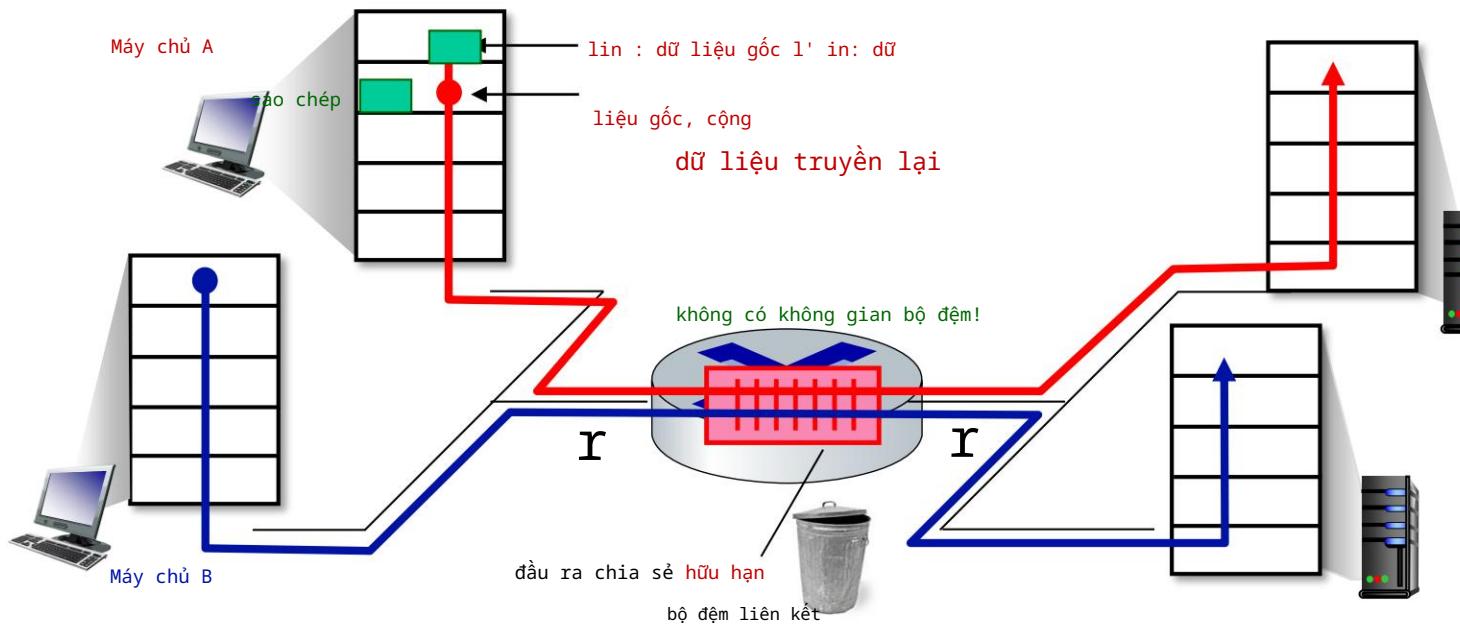


Nguyên nhân/chi phí tắc nghẽn: kịch bản 2

Lý tưởng hóa: một số kiến thức hoàn hảo

các gói có thể bị mất (rơi tại bộ định tuyến)
do đầy bộ đệm

bên gửi biết khi nào gói bị hủy: chỉ gửi
lại nếu gói đã biết là bị mất

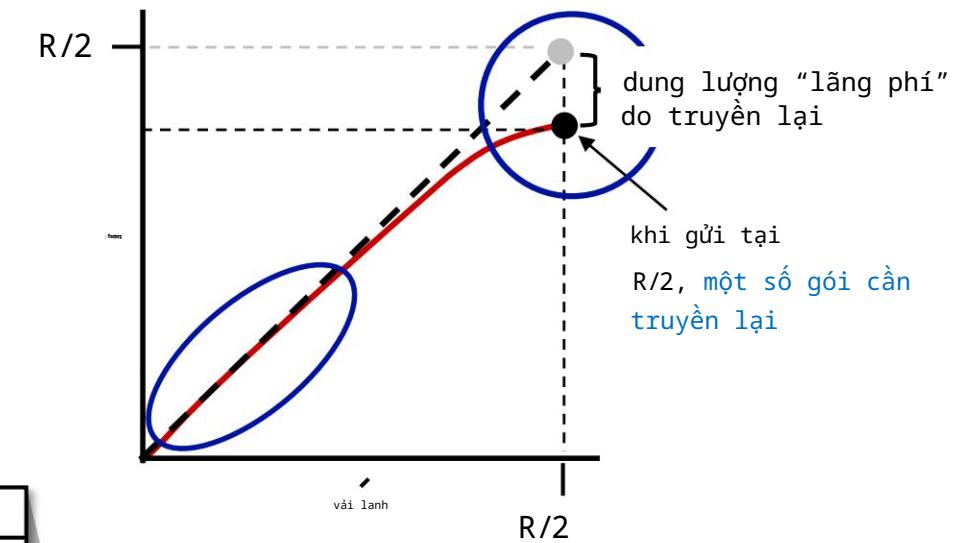
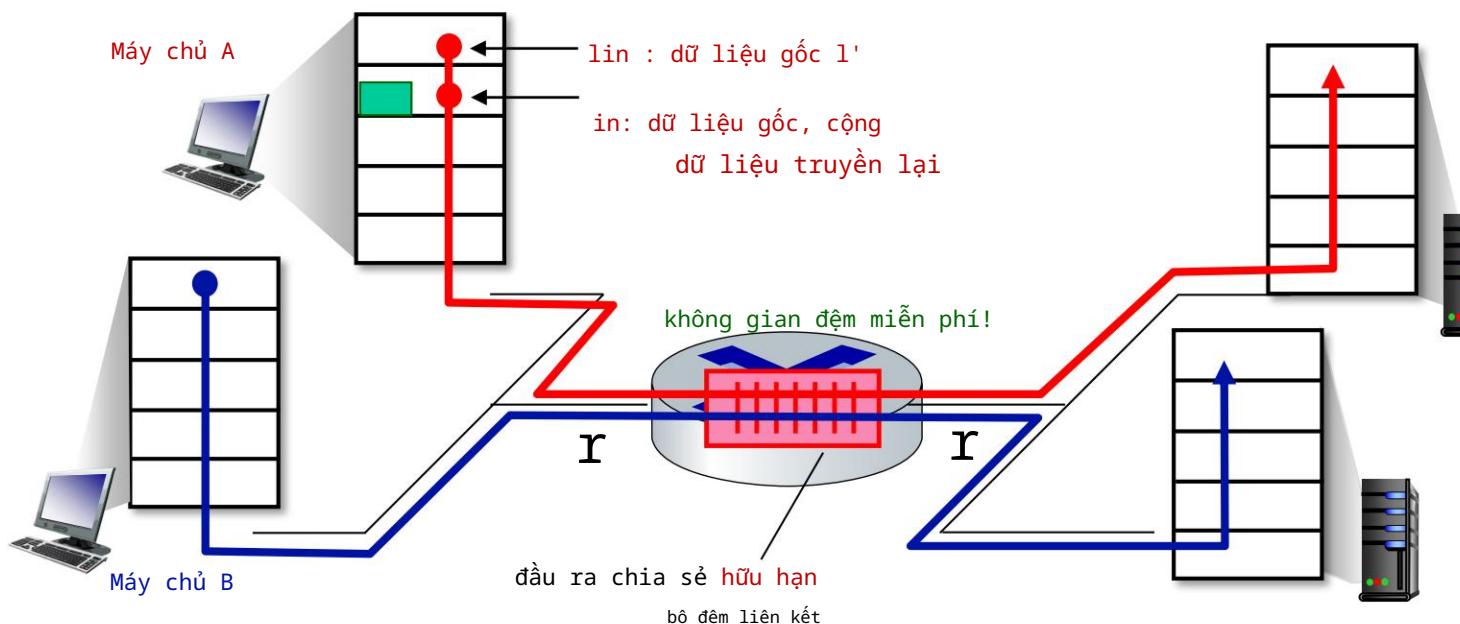


Nguyên nhân/chi phí tắc nghẽn: kịch bản 2

Lý tưởng hóa: một số kiến thức hoàn hảo

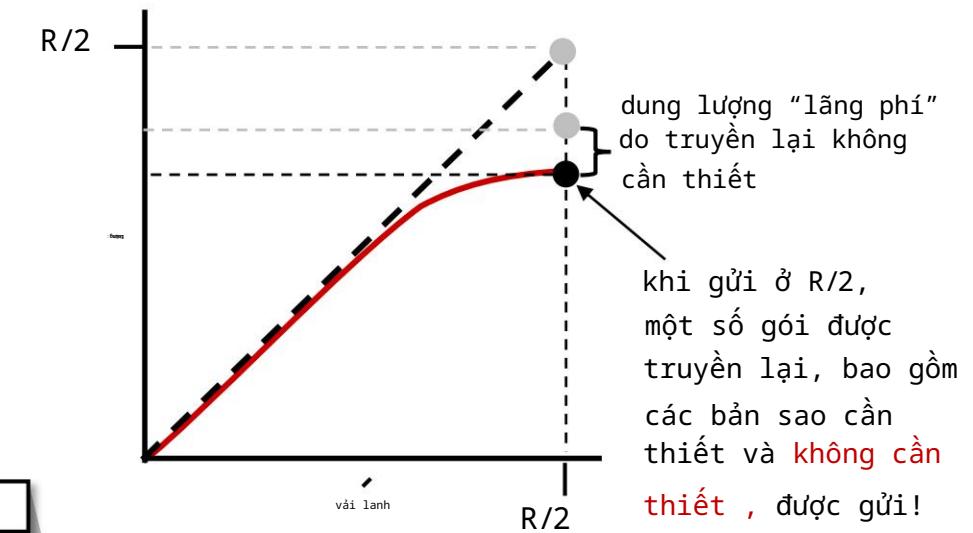
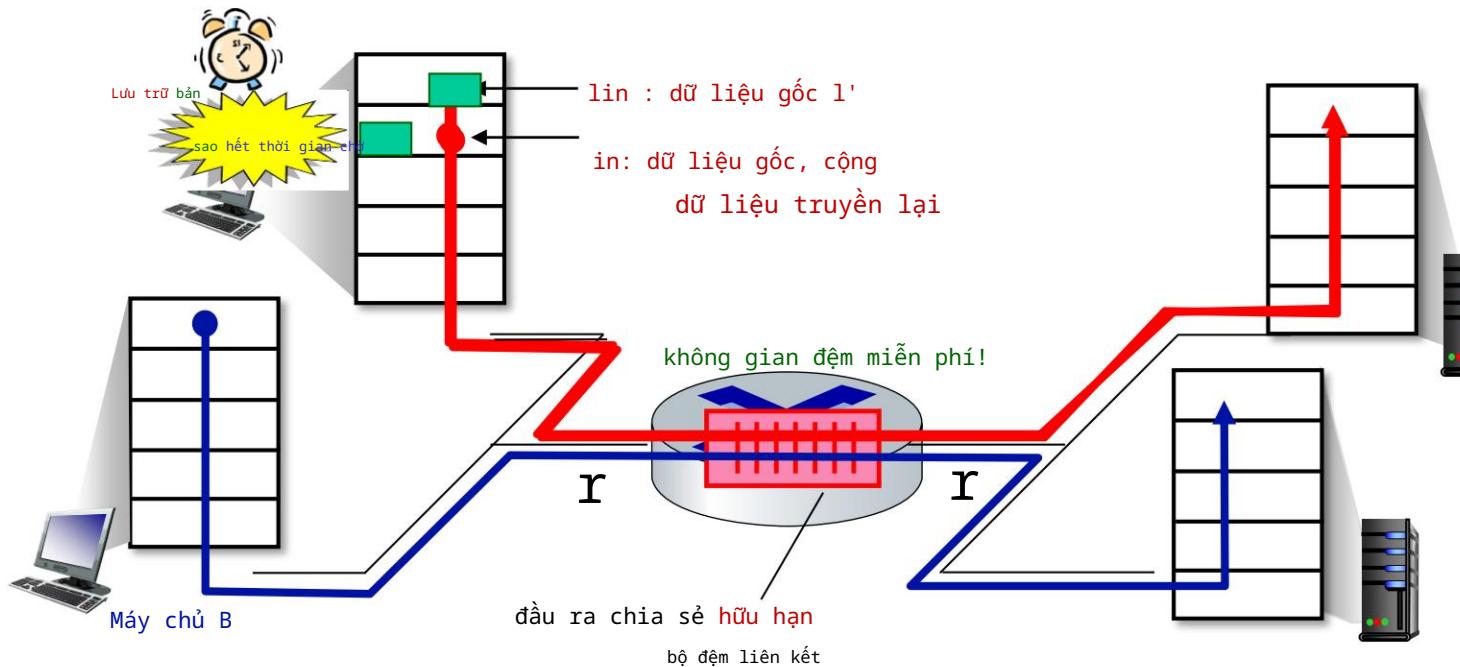
các gói có thể bị mất (rơi tại bộ định tuyến) do đầy bộ đệm

bên gửi biết khi nào gói bị hủy: chỉ gửi lại nếu gói đã biết là bị mất



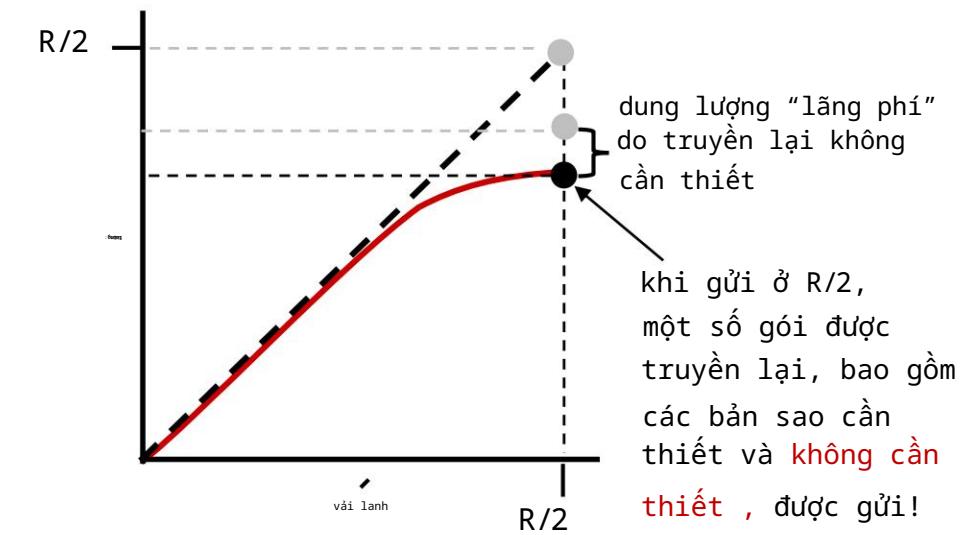
Nguyên nhân/chi phí tắc nghẽn: kịch bản 2

Kịch bản thực tế: các bản sao không cần thiết các gói có thể bị mất, bị hủy tại bộ định tuyến do đầy bộ đệm - yêu cầu **truyền lại** nhưng bộ hẹn giờ của người gửi có thể **hết thời gian sớm**, gửi **hai** bản sao, **cả hai** đều được **giao**



Nguyên nhân/chi phí tắc nghẽn: kịch bản 2

Kịch bản thực tế: các **bản sao không cần thiết** các gói có thể **bị mất**, bị hủy tại bộ định tuyến do đầy bộ đệm - yêu cầu truyền lại nhưng bộ hẹn giờ của người gửi có thể **hết thời gian sớm**, gửi **hai** bản sao, **cả hai** đều được **giao**



"chi phí" của tắc nghẽn:

nhiều công việc hơn (**truyền lại**) đối với thông lượng máy thu nhất định **truyền lại không cần thiết**: liên kết mang nhiều bản sao của một gói • **giảm** thông lượng tối đa có thể đạt được

Nguyên nhân/chi phí tắc nghẽn: kịch bản 3

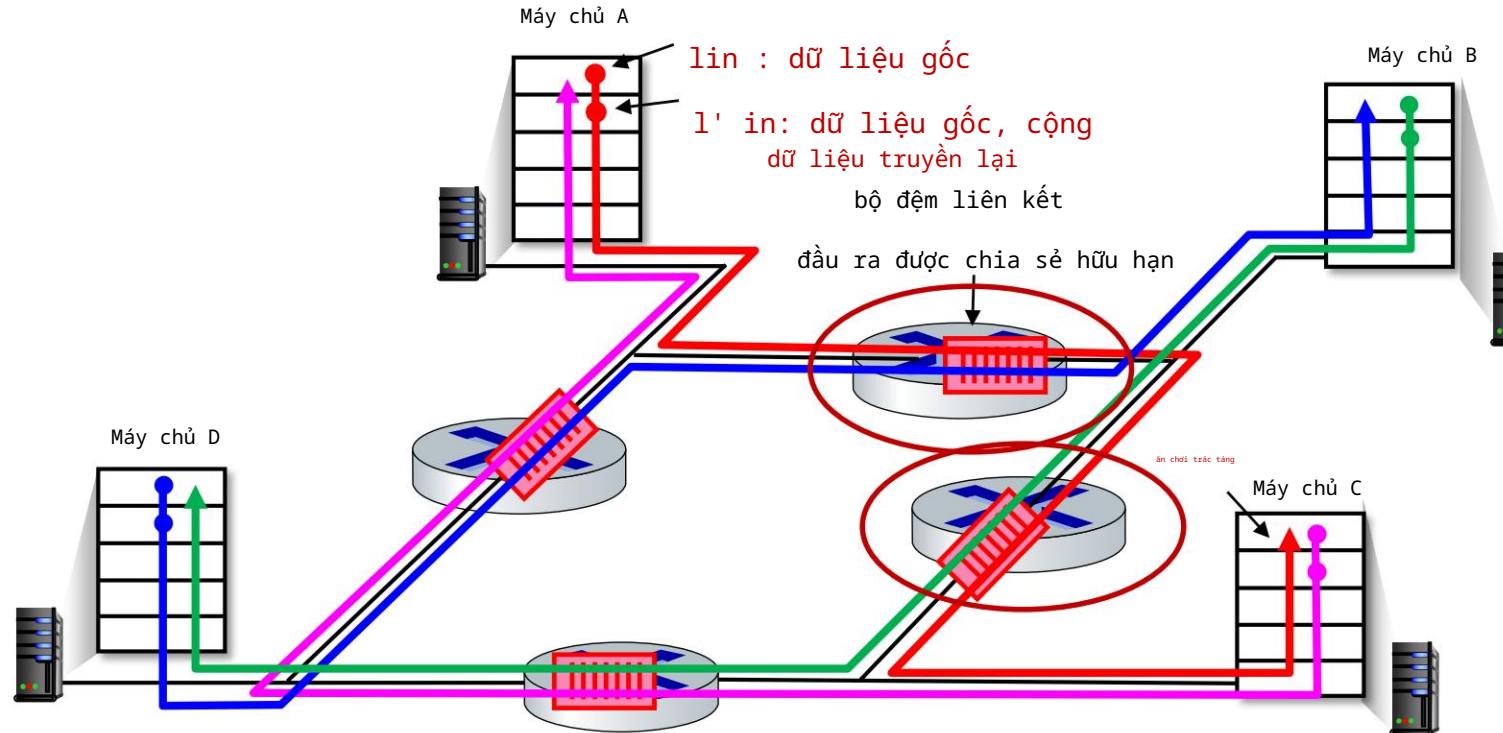
bốn người gửi

đường **nhiều** chặng

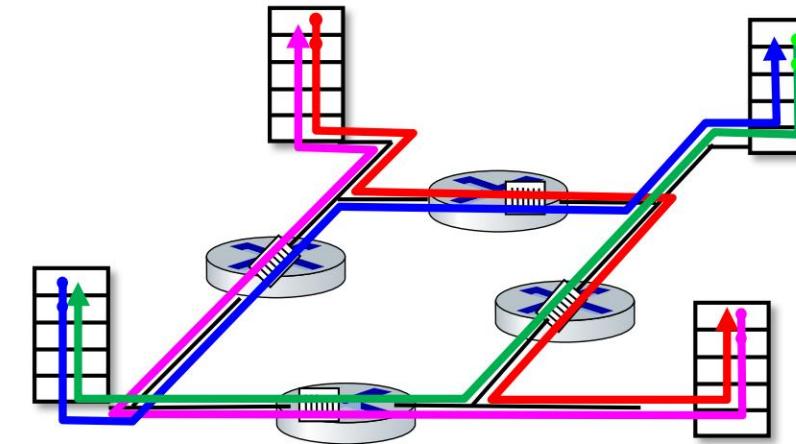
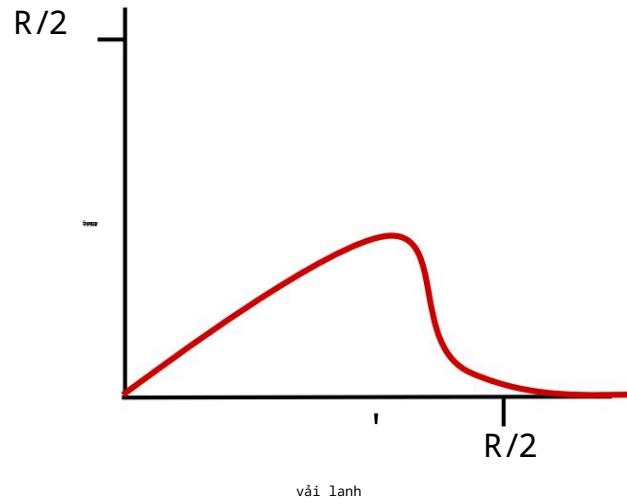
hết thời gian chờ/truyền lại

Q: điều gì xảy ra như lin và lin' tăng ?

A: khi hàng đợi tăng lên, tất cả các gói màu xanh đến ở trên màu đỏ bị loại bỏ, **thông lượng** màu xanh g 0



Nguyên nhân/chi phí tắc nghẽn: kịch bản 3



một “chi phí” tắc nghẽn khác:

khi gói tin bị rớt, bất kỳ dung lượng **truyền ngược dòng** nào và **bộ đếm** được sử dụng cho gói tin đó đều **bị lãng phí**!

Nguyên nhân/chi phí tắc nghẽn: hiểu biết sâu sắc

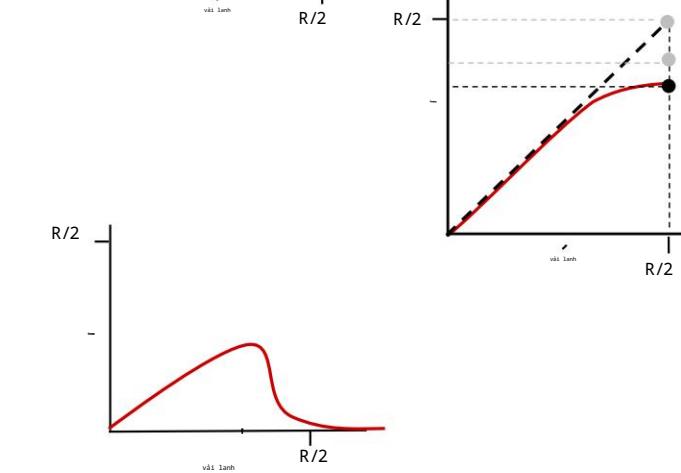
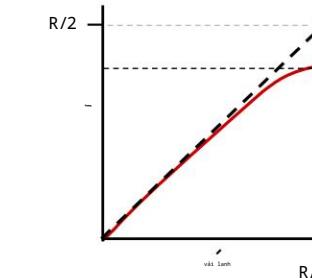
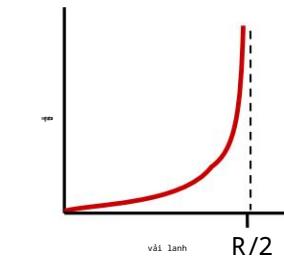
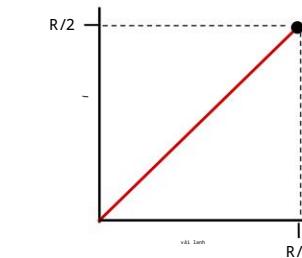
thông lượng không bao giờ được vượt quá công suất

độ trễ tăng lên khi công suất tiếp cận

mất mát/truyền lại làm giảm thông lượng hiệu dụng

bản sao không cần thiết tiếp tục giảm
thông lượng hiệu quả

Dung lượng truyền ngược dòng/bộ nhớ đệm bị lãng phí
do các gói bị mất ở lưu

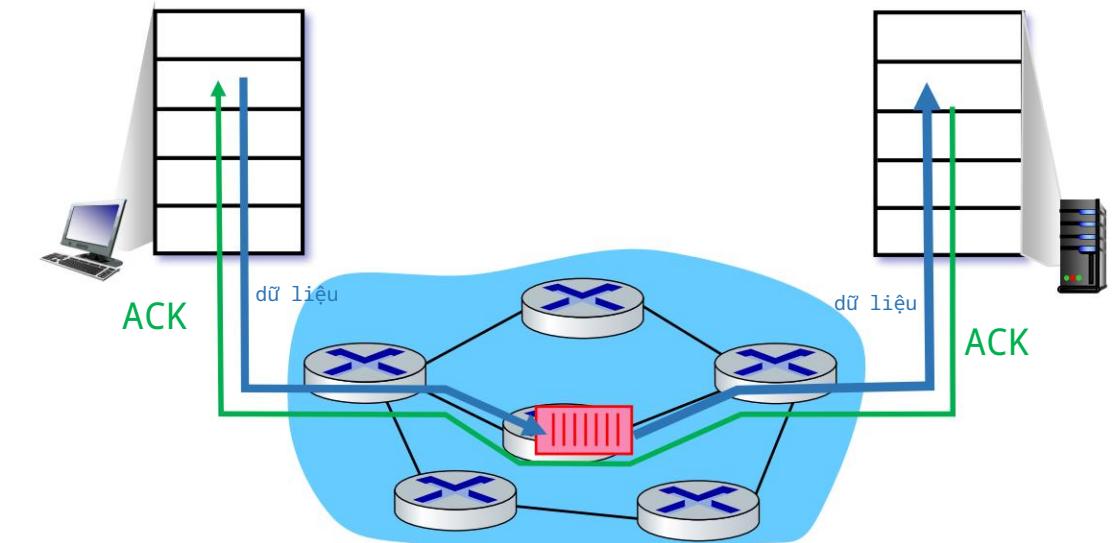


Các phương pháp điều khiển tắc nghẽn

Kiểm soát tắc nghẽn đầu cuối:

không có phản hồi rõ ràng từ mạng

tắc nghẽn **suy ra** từ mất mát, chậm trễ quan sát được cách tiếp cận của TCP



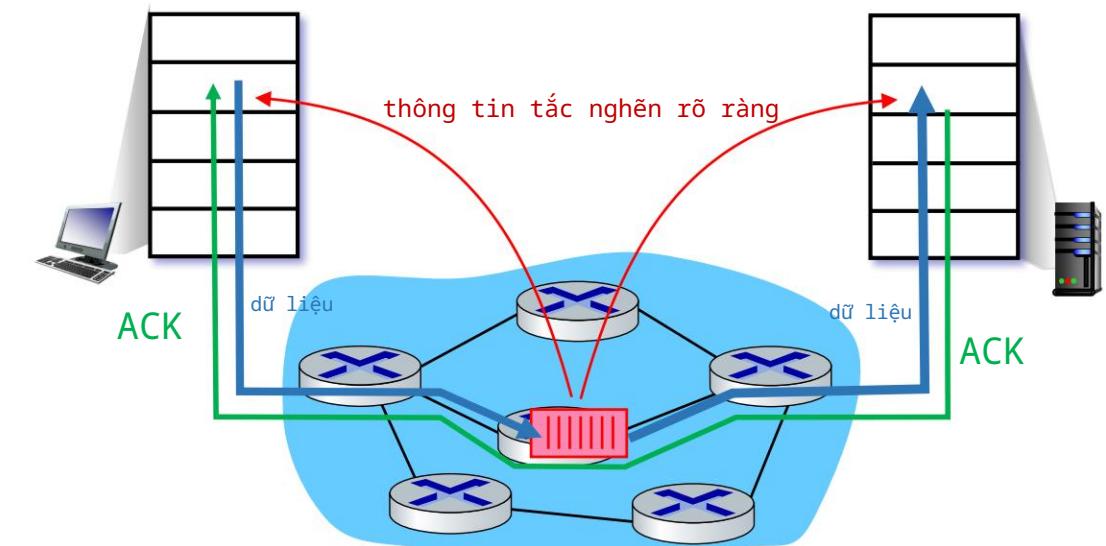
Các phương pháp điều khiển tắc nghẽn

Kiểm soát tắc nghẽn có sự hỗ trợ của mạng:

các bộ định tuyến cung cấp phản hồi trực tiếp cho các máy chủ gửi/nhận với các luồng đi qua bộ định tuyến bị tắc nghẽn

có thể chỉ ra mức độ tắc nghẽn hoặc thiết lập rõ ràng tỷ lệ gửi

- ví dụ: giao thức TCP ECN, ATM, DECbit



Chương 3: lộ trình

Dịch vụ tầng vận chuyển
Ghép kênh và phân kênh Vận
chuyển không kết nối: UDP Nguyên
tắc truyền dữ liệu đáng tin cậy
Truyền tải hướng kết nối: TCP
Nguyên tắc kiểm soát tắc nghẽn Kiểm
soát tắc nghẽn TCP Sự phát triển
của chức năng tầng vận chuyển



Kiểm soát tắc nghẽn TCP: AIMD

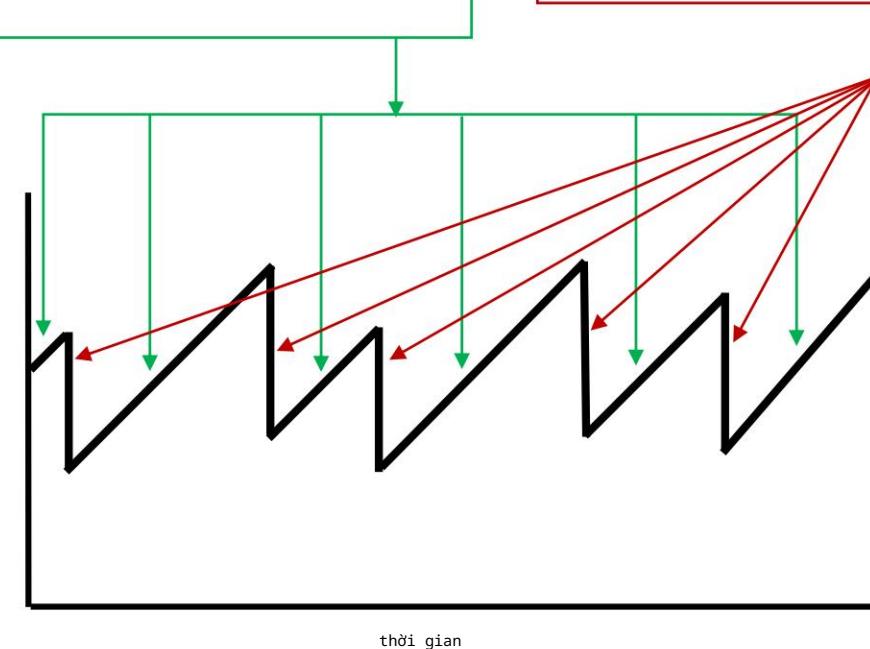
cách tiếp cận: người gửi có thể tăng tốc độ gửi cho đến khi xảy ra mất gói (tắc nghẽn), sau đó giảm tốc độ gửi trong trường hợp mất gói

Phụ Gia Tăng

tăng tốc độ gửi lên 1 kích thước phân đoạn tối đa (MSS) mỗi RTT cho đến khi phát hiện mất mát

Phép nhân Giảm_giảm

một nửa tốc độ gửi ở mỗi sự kiện mất mát



Hành vi răng cưa
AIMD : thăm dò băng thông

TCP AIMD: thêm

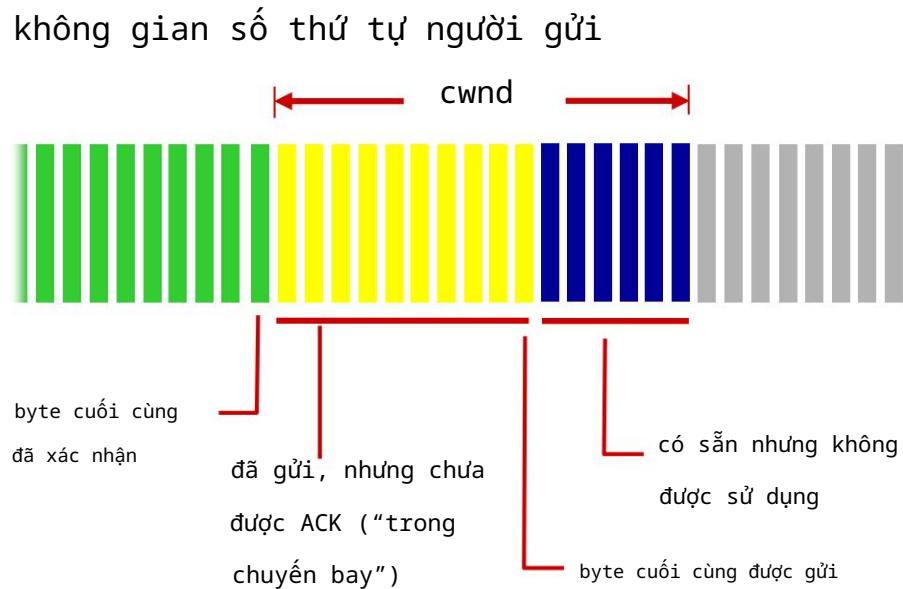
Chi tiết **giảm nhân** : 2 sự kiện, tốc độ gửi là Giảm **một nửa** khi mất được phát hiện bởi **ACK trùng lặp ba lần** (TCP Reno) **Cắt thành 1 MSS** (kích thước phân đoạn tối đa) khi mất được phát hiện do **hết thời gian chờ** (TCP Tahoe)

Tại sao AIMD?

AIMD - thuật toán phân tán, không đồng bộ - đã được hiển

thị để: • **tối ưu hóa tỷ lệ lưu lượng bị tắc nghẽn** trên toàn mạng! • có **đặc tính ổn định mong muốn**

Kiểm soát tắc nghẽn TCP: chi tiết



Hành vi gửi TCP:

đại khái: gửi **cwnd** byte, đợi RTT cho ACK, sau đó gửi thêm byte

$$\text{Tốc độ TCP} \sim \frac{\text{cwnd}}{\text{RTT}} \text{ byte/giây}$$

Người gửi TCP giới hạn truyền:

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

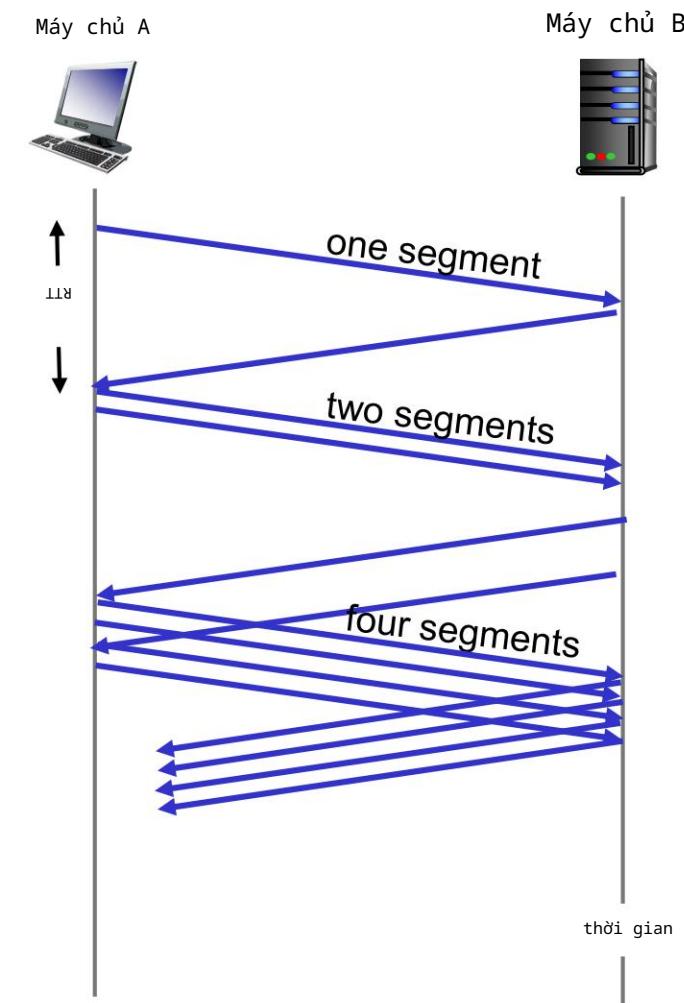
cwnd được tự động **điều chỉnh** để đáp ứng với quan sát tắc nghẽn mạng (thực hiện kiểm soát tắc nghẽn TCP)

TCP bắt đầu chậm

khi kết nối bắt đầu, **tăng tốc độ theo cấp số nhân** cho đến khi xảy ra sự kiện mất mát đầu tiên:

- **cwnd** ban đầu = 1 MSS
- **cwnd nhân đôi** mỗi RTT
- **được thực hiện bằng cách tăng cwnd cho mỗi ACK nhận được**

tóm tắt: tốc độ ban đầu chậm, nhưng **tăng nhanh theo cấp số nhân**



TCP: từ bắt đầu chậm đến tránh tắc nghẽn

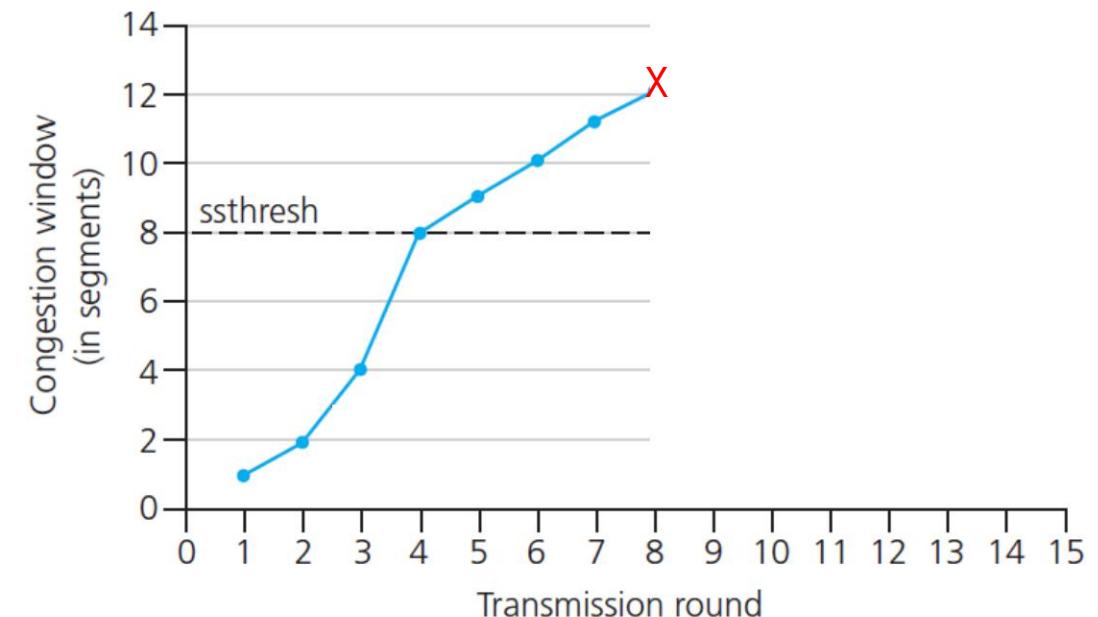
Hỏi: khi nào mức tăng theo cấp số nhân nên chuyển sang tuyến tính?

A: khi `cwnd` đạt $1/2$ giá trị trước khi hết thời gian chờ.

Thực hiện:

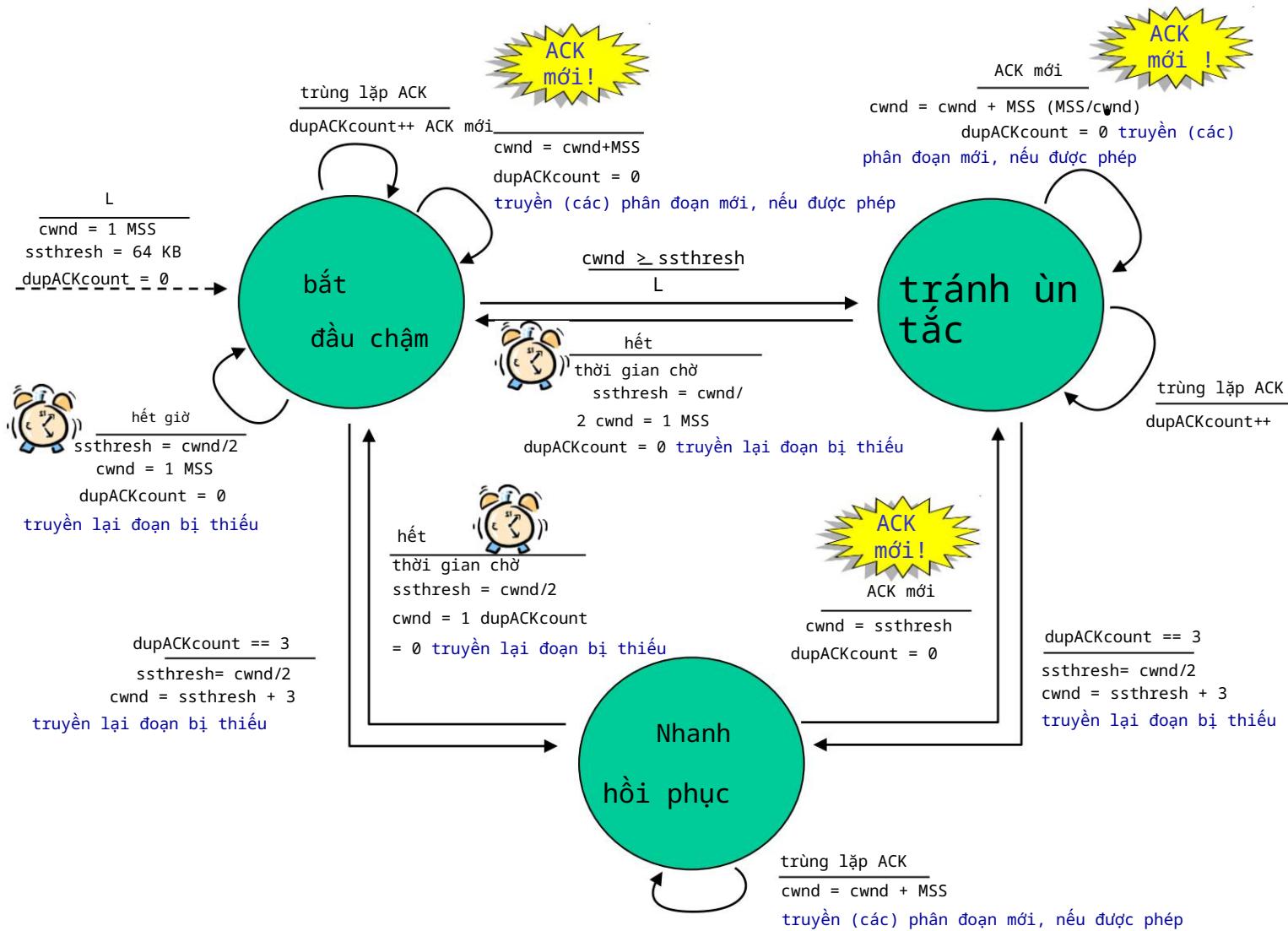
biến `ssthresh`

trong trường hợp thua lỗ, `ssthresh` được đặt thành $1/2$ của `cwnd` ngay trước sự kiện thua lỗ



* Xem các bài tập tương tác trực tuyến để biết thêm ví dụ: http://gaia.cs.umass.edu/kurose_ross/interactive/

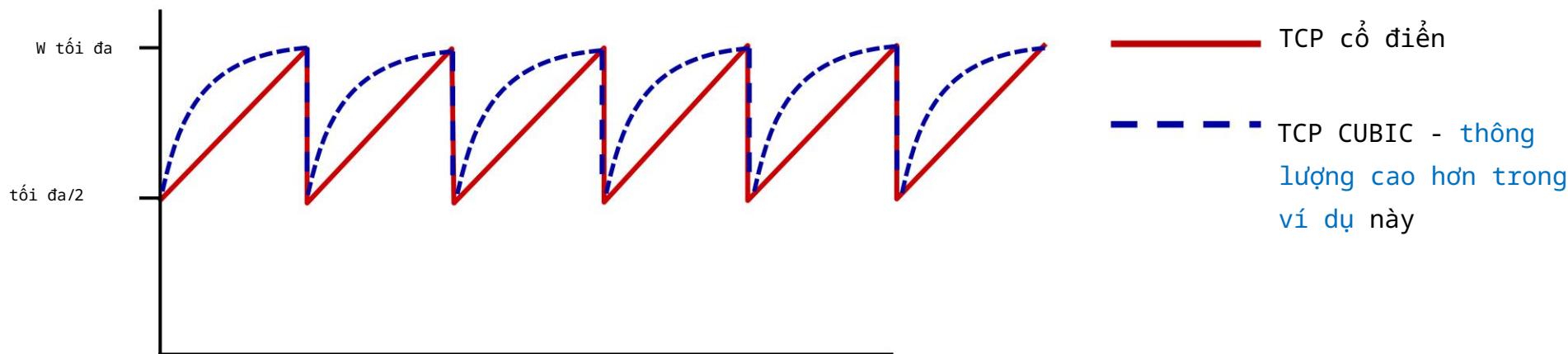
Tóm tắt: Kiểm soát tắc nghẽn TCP



Khối TCP

Có cách nào tốt hơn AIMD để “thăm dò” băng thông khả dụng không? Thông tin chi tiết/trực giác:

- W_{max} : tốc độ gửi mà tại đó phát hiện mất tắc nghẽn
- trạng thái tắc nghẽn của liên kết cổ chai có lẽ (?) không thay đổi nhiều
- sau khi giảm một nửa tốc độ/cửa sổ khi mất, ban đầu tăng dần đến W_{max} nhanh hơn, nhưng sau đó tiếp cận W_{max} chậm hơn



Khối TCP

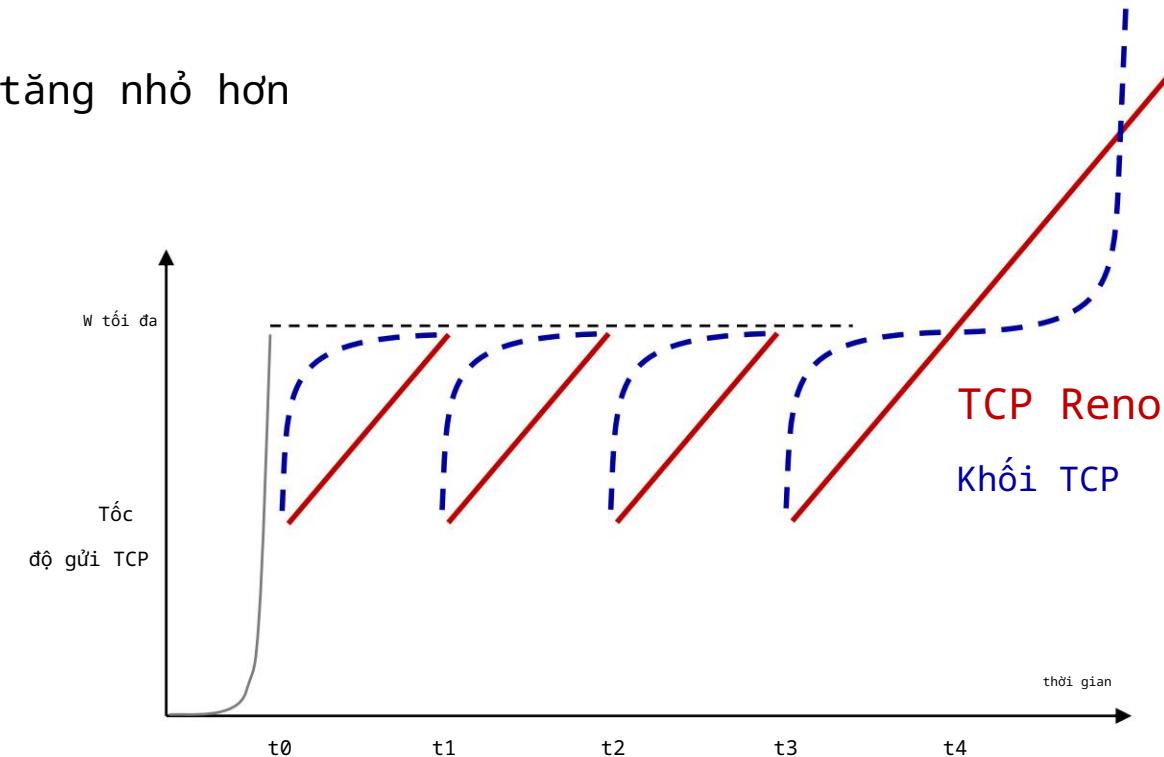
K: thời điểm khi kích thước cửa sổ TCP đạt W_{max}

- Bản thân K điều chỉnh được

tăng W như một hàm lập phương của khoảng cách giữa dòng điện thời gian và k

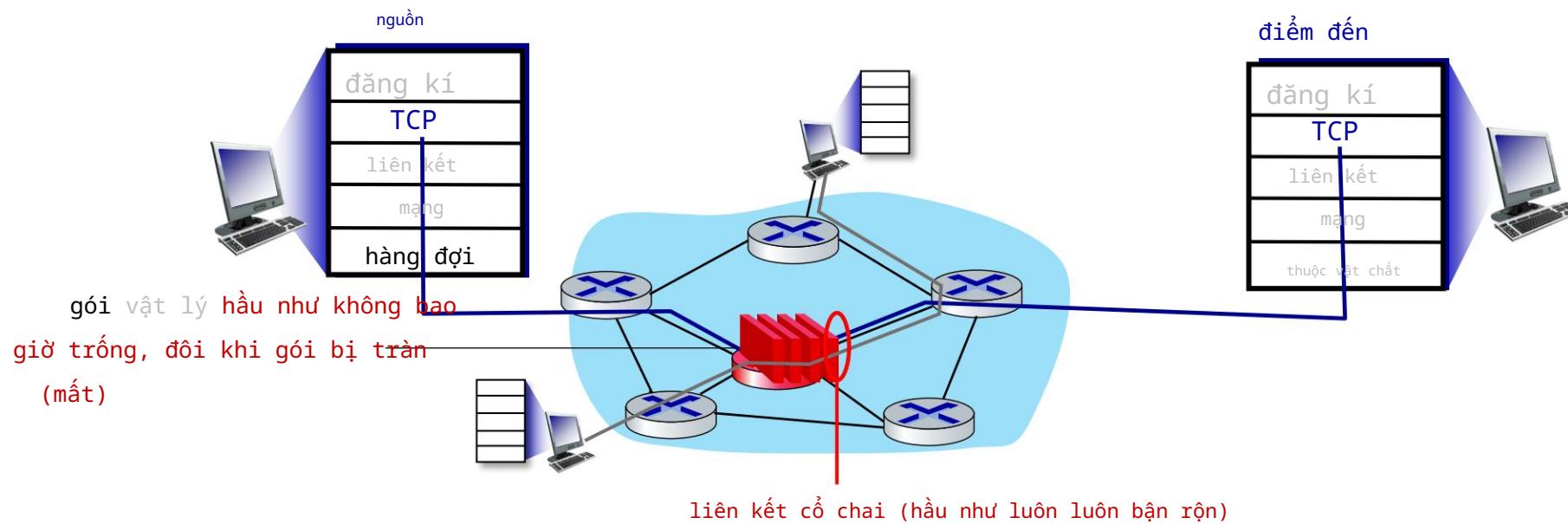
- tăng nhiều hơn khi ở xa K
- tăng nhỏ hơn
(thận trọng) khi ở gần K hơn

TCP CUBIC mặc định
trong Linux, TCP
phổ biến nhất cho
Web phổ biến
may chu



TCP và “liên kết thắt cổ chai” tắc nghẽn

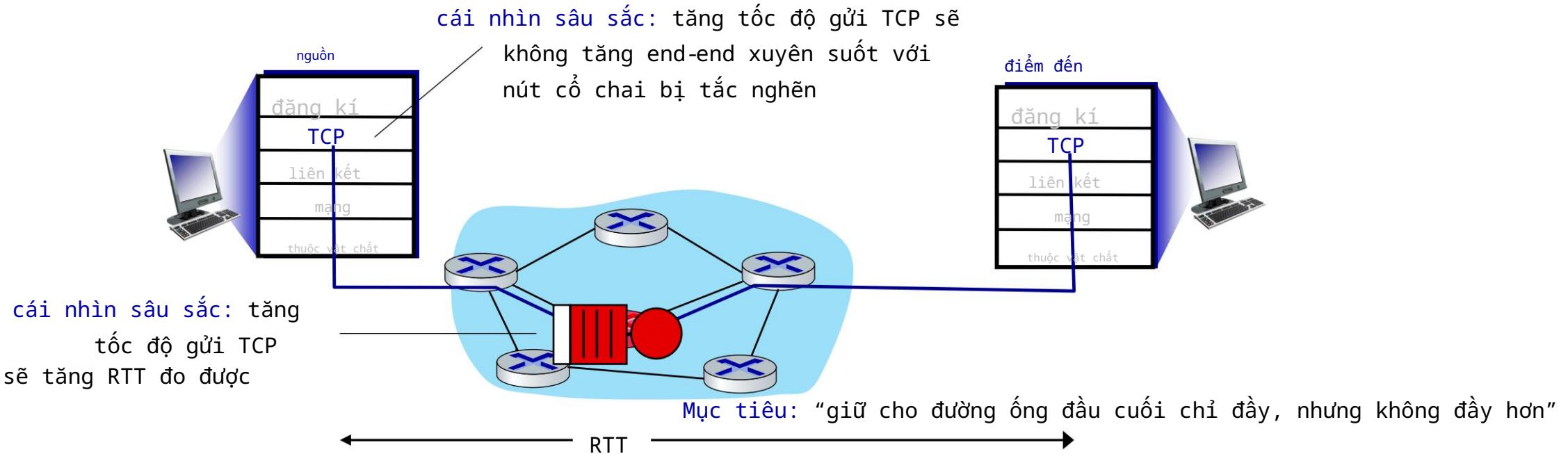
TCP (cổ điển, CUBIC) tăng tốc độ gửi của TCP cho đến khi mất gói xảy ra ở đầu ra của một số bộ định tuyến: **liên kết cổ chai**



TCP và “liên kết thắt cổ chai” tắc nghẽn

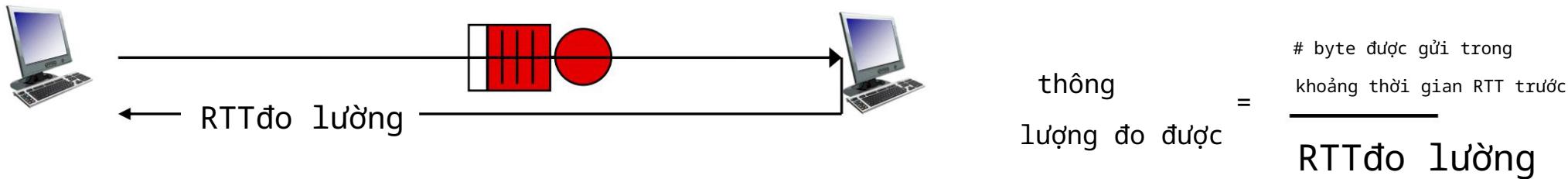
TCP (cổ điển, CUBIC) tăng tốc độ gửi của TCP cho đến khi mất gói xảy ra ở đầu ra của một số bộ định tuyến: **liên kết cổ chai**

hiểu tắc nghẽn: hữu ích khi tập trung vào liên kết cổ chai tắc nghẽn



Kiểm soát tắc nghẽn TCP dựa trên độ trễ

Giữ đường ống giữa người gửi và người nhận “vừa đủ, nhưng không đầy hơn”: giữ cho liên kết cổ chai **bận rộn truyền**, nhưng **tránh độ trễ/đệm cao**



Phương pháp tiếp cận dựa trên độ trễ:

RTT_{min} - RTT quan sát tối thiểu (đường dẫn không bị tắc nghẽn)
 Thông lượng không bị tắc nghẽn với **cwnd** cửa sổ tắc nghẽn là cwnd/RTT_{min}
 nếu thông lượng đo được “rất gần” với thông lượng không bị tắc nghẽn
 tăng **cwnd** tuyến tính /* do đường dẫn không bị tắc nghẽn */
 thấp” không bị tắc nghẽn trong suốt **cwnd giảm** tuyến tính /* do đường
 dẫn bị tắc nghẽn */

Kiểm soát tắc nghẽn TCP dựa trên độ trễ

kiểm soát tắc nghẽn mà không gây ra/bắt buộc mất mát

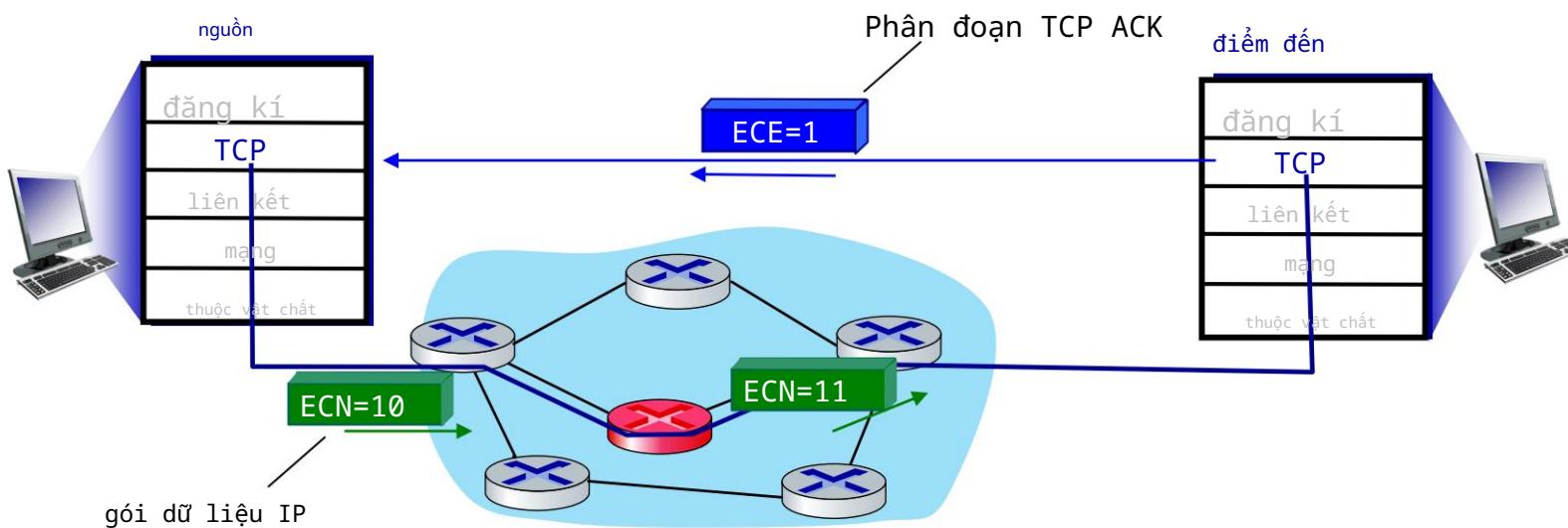
tối đa hóa thông lượng ("giữ đường ống vừa đủ.") trong khi vẫn giữ **độ trễ thấp** ("nhưng không đầy hơn")

một số TCP đã triển khai sử dụng cách tiếp cận dựa trên độ trễ

ví dụ: **Băng thông cổ chai và thời gian lan truyền khứ hồi** (BBR) được triển khai trên mạng đường trực (nội bộ) của Google

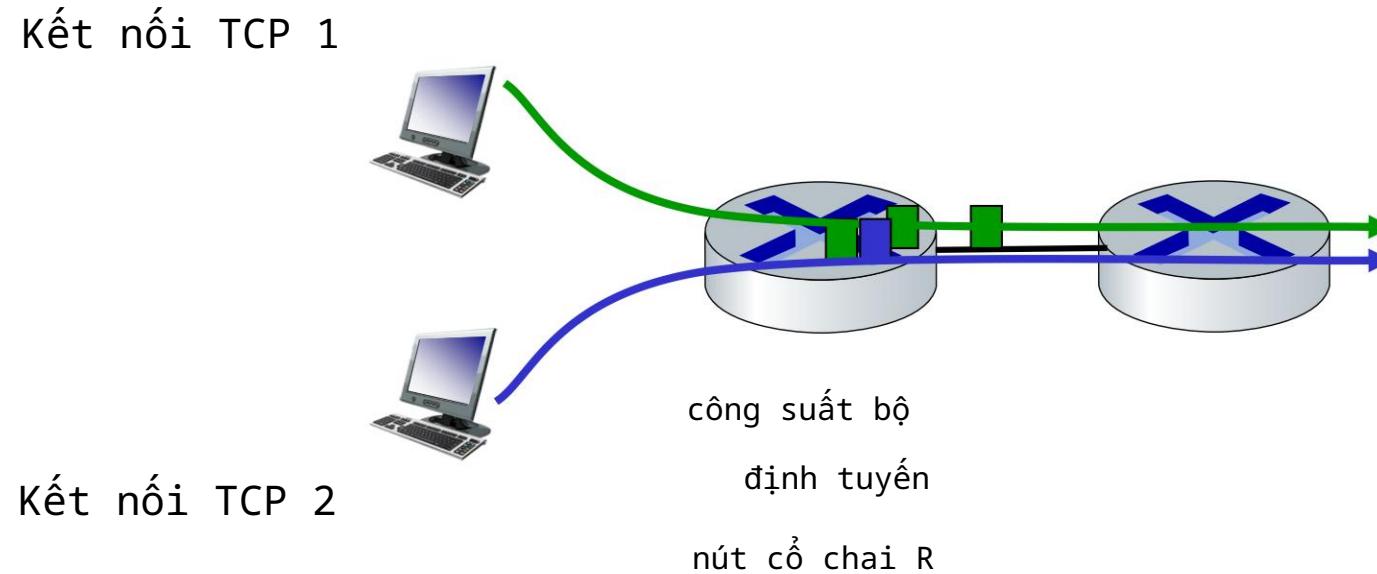
Thông báo tắc nghẽn rõ ràng (ECN)

Việc triển khai TCP thường thực hiện kiểm soát tắc nghẽn do mạng hỗ trợ : hai bit trong tiêu đề IP (trường ToS) được đánh dấu bởi bộ định tuyến mạng để biểu thị tắc nghẽn • chính sách xác định việc đánh dấu do nhà điều hành mạng chọn chỉ báo tắc nghẽn được mang đến đích đích đặt bit ECE trên phân đoạn ACK để thông báo cho người gửi về tắc nghẽn liên quan đến cả IP (đánh dấu bit ECN tiêu đề IP) và TCP (đánh dấu bit C,E tiêu đề TCP)



công bằng TCP

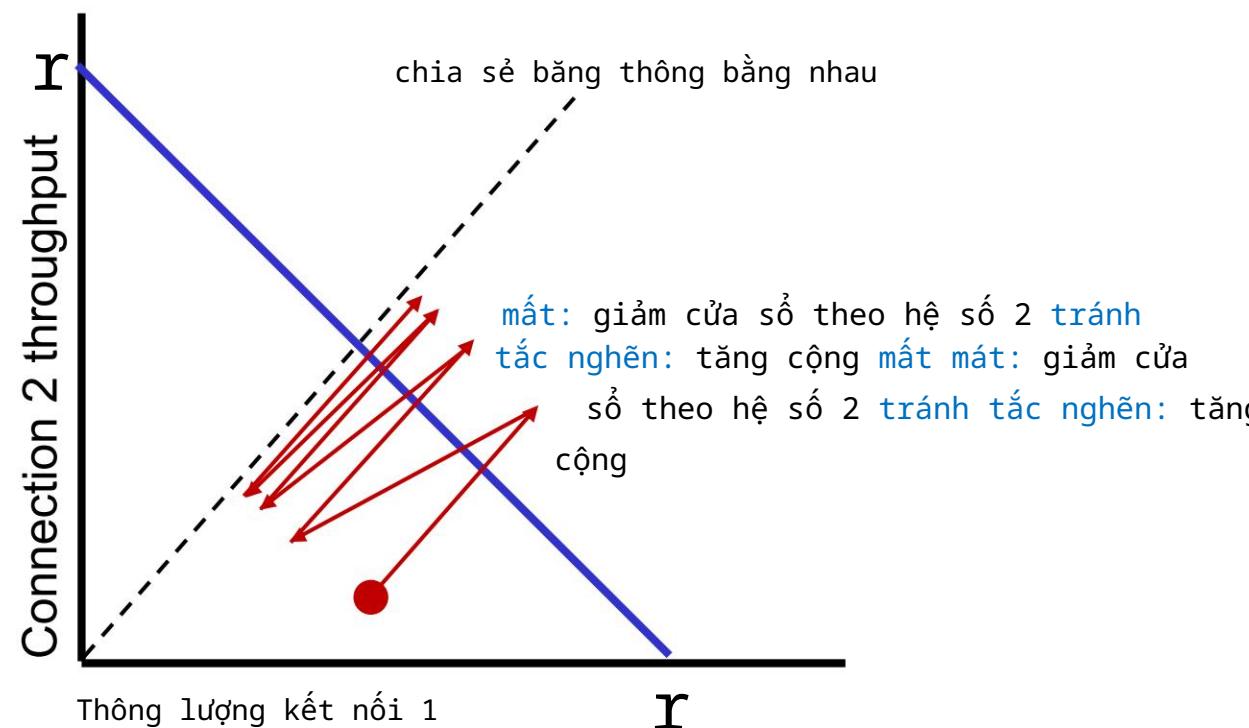
Mục tiêu công bằng: nếu K phiên TCP chia sẻ cùng một liên kết cổ chai bằng thông R, thì mỗi phiên sẽ có tốc độ R/K trung bình



Hỏi: TCP có công bằng không?

Ví dụ: hai phiên TCP cạnh tranh:

tăng cộng dồn cho độ dốc bằng 1, khi tăng xuyên suốt giảm cấp số nhân làm giảm thông lượng theo tỷ lệ



TCP có công bằng không ?

Đ: Có, theo các giả định lý tưởng: RTT giống nhau số phiên cố định chỉ để tránh tắc nghẽn

Công bằng: tất cả các ứng dụng mạng phải “công bằng”?

Công bằng và UDP

các ứng dụng đa phương tiện thường không sử dụng TCP • không muốn điều chỉnh tốc độ bằng kiểm soát tắc nghẽn

thay vào đó hãy sử dụng

UDP: • gửi âm thanh/video với tốc độ không đổi, chấp nhận mất gói tin
không có “cảnh sát Internet” kiểm soát việc sử dụng kiểm soát tắc nghẽn

Công bằng, các kết nối TCP song song

ứng dụng có thể mở nhiều kết nối song song giữa hai máy chủ

trình duyệt web thực hiện việc này, ví dụ: liên kết tốc độ R với 9 kết nối hiện có: • ứng dụng mới yêu cầu 1 TCP, nhận tốc độ R/10 • ứng dụng mới yêu cầu 11 TCP, nhận R/20

Lớp vận chuyển: lộ trình

Dịch vụ tầng vận chuyển

Ghép kênh và phân kênh Vận
chuyển không kết nối: UDP Nguyên

tắc truyền dữ liệu đáng tin cậy

Truyền tải hướng kết nối: TCP

Nguyên tắc kiểm soát tắc nghẽn Kiểm

soát tắc nghẽn TCP Sự phát triển

của chức năng tầng vận chuyển



Phát triển chức năng tầng vận chuyển

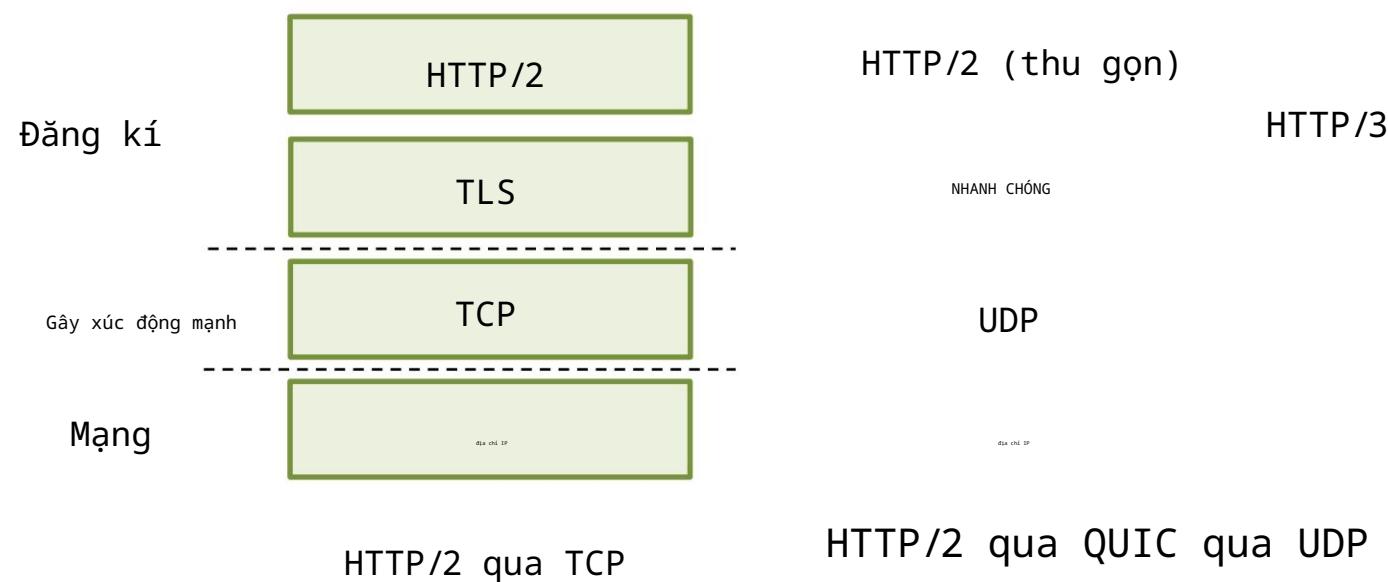
TCP, UDP: các giao thức truyền tải chính trong 40 năm
 các “**hương vị**” khác nhau của TCP được phát triển, cho các kịch bản cụ thể:

| Kịch bản | thử thách |
|--|---|
| Đường ống dài, mập (truyền dữ liệu lớn) | Nhiều gói “đang bay”; mất mát đóng cửa đường ống |
| Mạng không dây | Suy hao do liên kết không dây ồn ào, tính di động; TCP coi đây là mất mát tắc nghẽn |
| Liên kết có độ trễ dài | RTT cực dài |
| Mạng trung tâm dữ liệu | nhạy cảm với độ trễ |
| Luồng giao thông nền | Luồng TCP ưu tiên thấp, “nền” |

chuyển các chức năng của tầng vận chuyển sang tầng ứng dụng, trên UDP • HTTP/3,
 QUIC

QUIC: Kết nối Internet UDP nhanh

giao thức tầng ứng dụng, nằm trên UDP • tăng hiệu suất của HTTP • được triển khai trên nhiều máy chủ, ứng dụng của Google ([Chrome](#), ứng dụng [YouTube](#) dành cho thiết bị di động)



QUIC: Kết nối Internet UDP nhanh

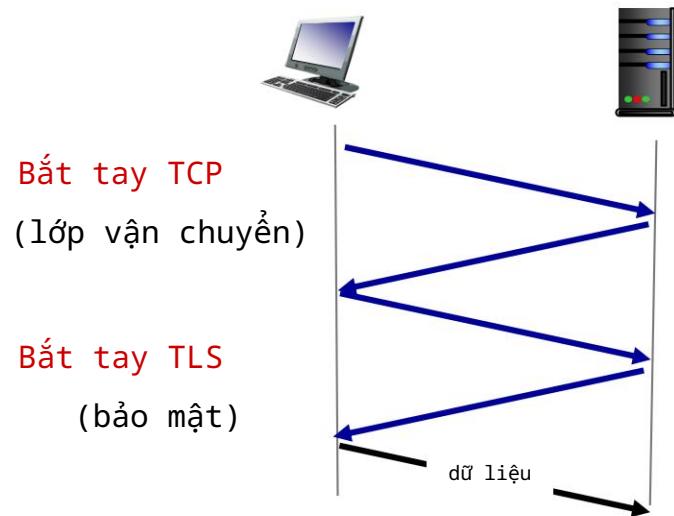
áp dụng các phương pháp mà chúng ta đã nghiên cứu trong chương này để **thiết lập kết nối, kiểm soát lỗi, kiểm soát tắc nghẽn**

- **Kiểm soát lỗi và tắc nghẽn:** “Bạn đọc đã quen với sự mất mát của TCP phát hiện và kiểm soát tắc nghẽn sẽ tìm thấy các thuật toán ở đây *song song* với các thuật toán TCP nổi tiếng.” [từ đặc tả QUIC]
- **thiết lập kết nối:** độ tin cậy, kiểm soát tắc nghẽn, xác thực, mã hóa, trạng thái được thiết lập **trong một RTT**

nhiều “luồng” cấp ứng dụng được ghép kênh trên QUIC đơn sự liên quan

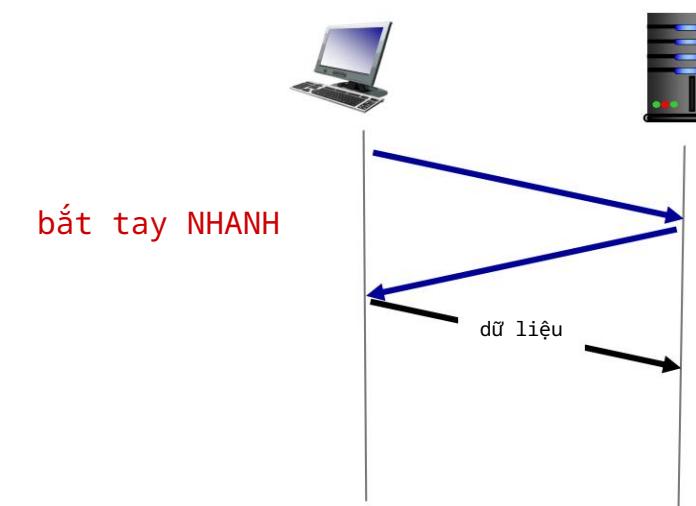
- truyền dữ liệu đáng tin cậy riêng biệt,
- bảo mật • kiểm soát tắc nghẽn chung

QUIC: Thiết lập kết nối



TCP (độ tin cậy, trạng thái kiểm soát tắc nghẽn)
+ TLS (xác thực, trạng thái tiền điện tử)

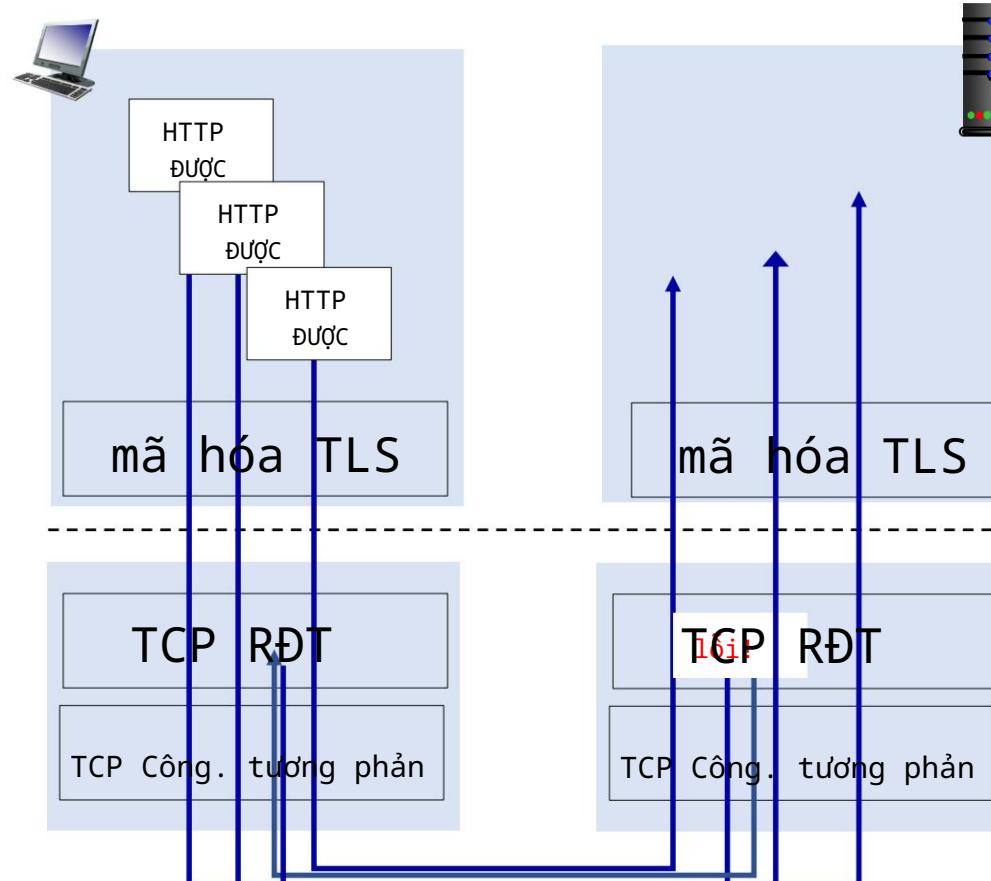
2 lần bắt tay nối tiếp



QUIC: độ tin cậy, kiểm soát tắc nghẽn,
xác thực, trạng thái tiền điện tử

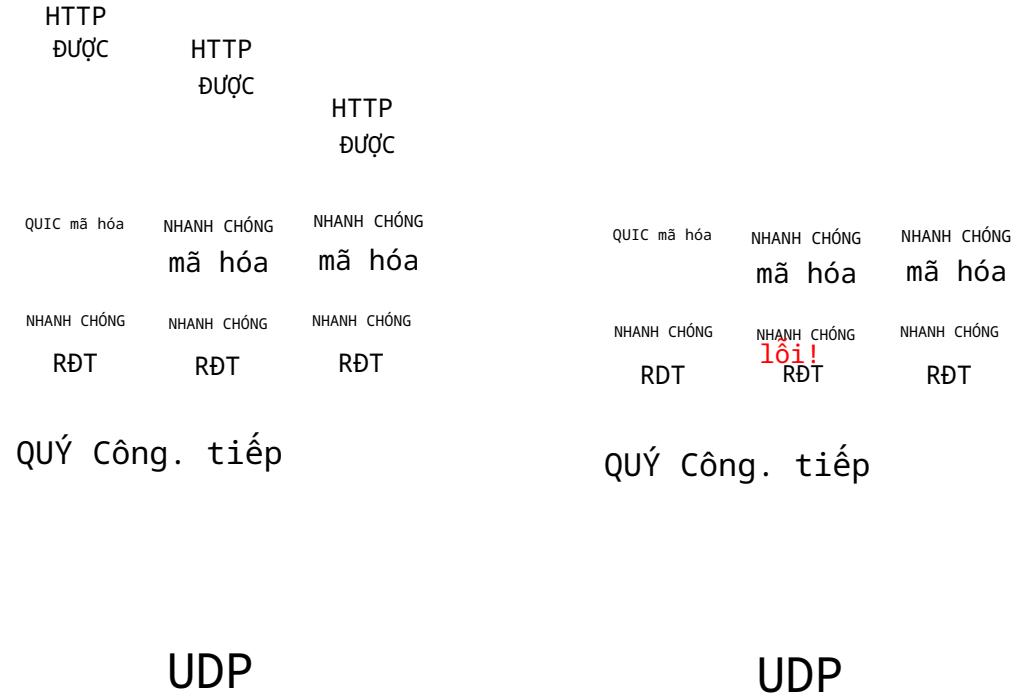
1 cái bắt tay

QUIC: luồng: song song, không chặn HOL



(a) HTTP 1.1

(b) HTTP/2 với QUIC: không chặn HOL



chương 3: tóm tắt

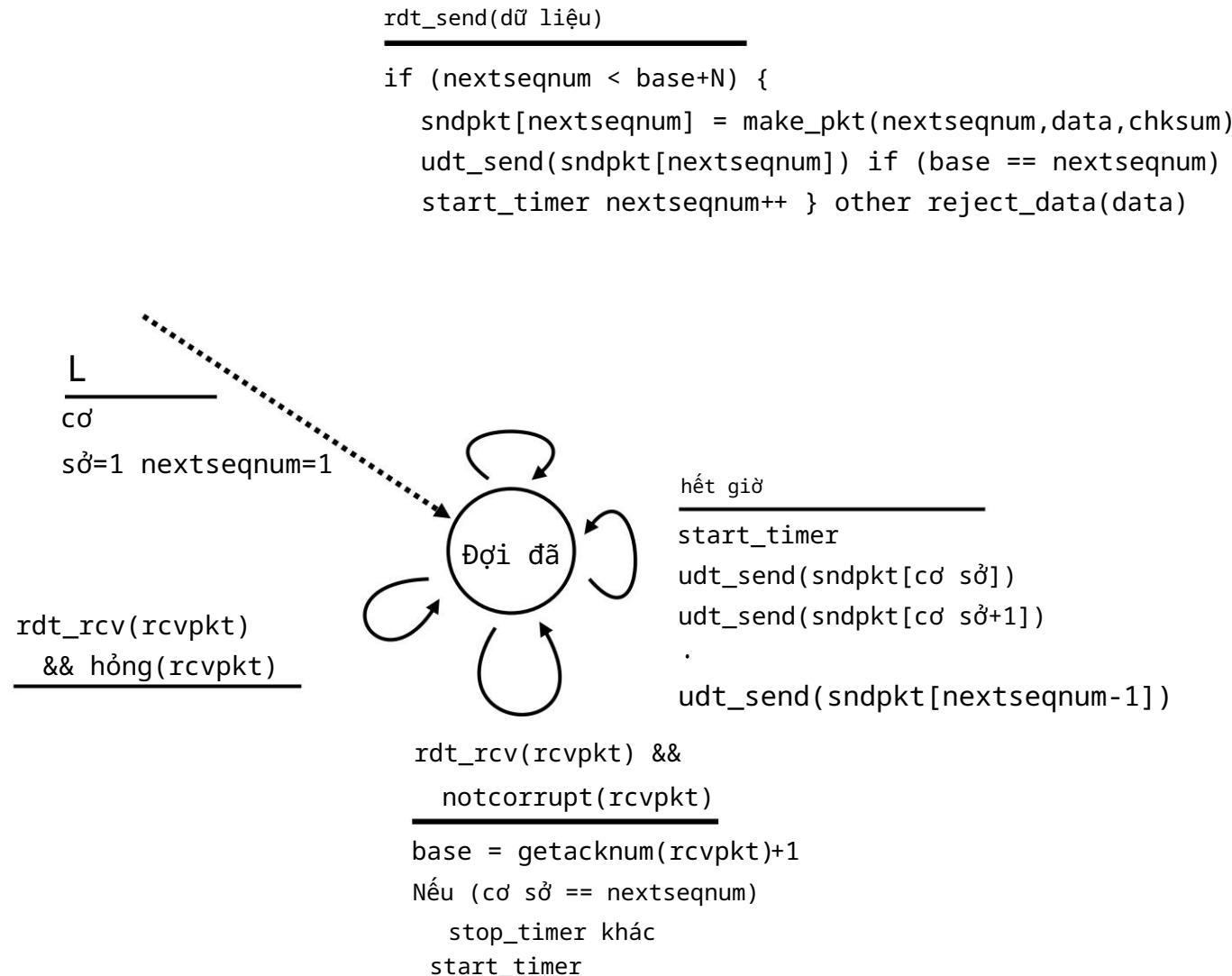
- nguyên tắc đằng sau các dịch vụ
tầng vận chuyển:
- ghép kênh, tách kênh
 - truyền dữ liệu đáng tin cậy
 - kiểm soát luồng
 - kiểm soát tắc nghẽn
 - khởi tạo, triển khai trên Internet
-
- UDP
 - TCP

Phần tiếp theo: rời mạng “cạnh” (các lớp ứng dụng, vận chuyển)
vào mạng “lõi”

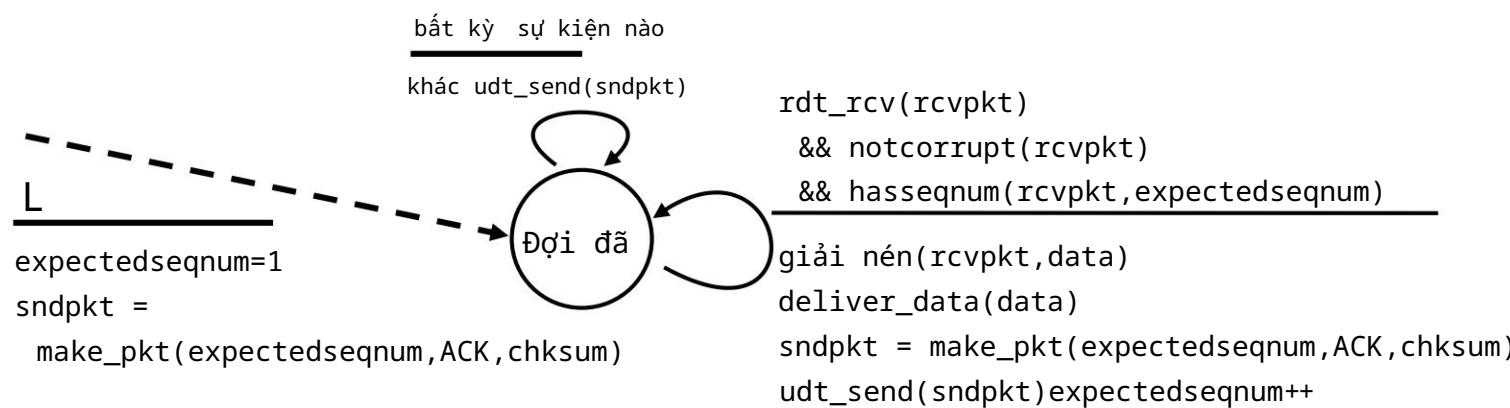
hai chương về tầng mạng:
• mặt phẳng dữ liệu •
mặt phẳng điều khiển

Các slide Chương 3 bổ sung

Go-Back-N: FSM mở rộng của người gửi



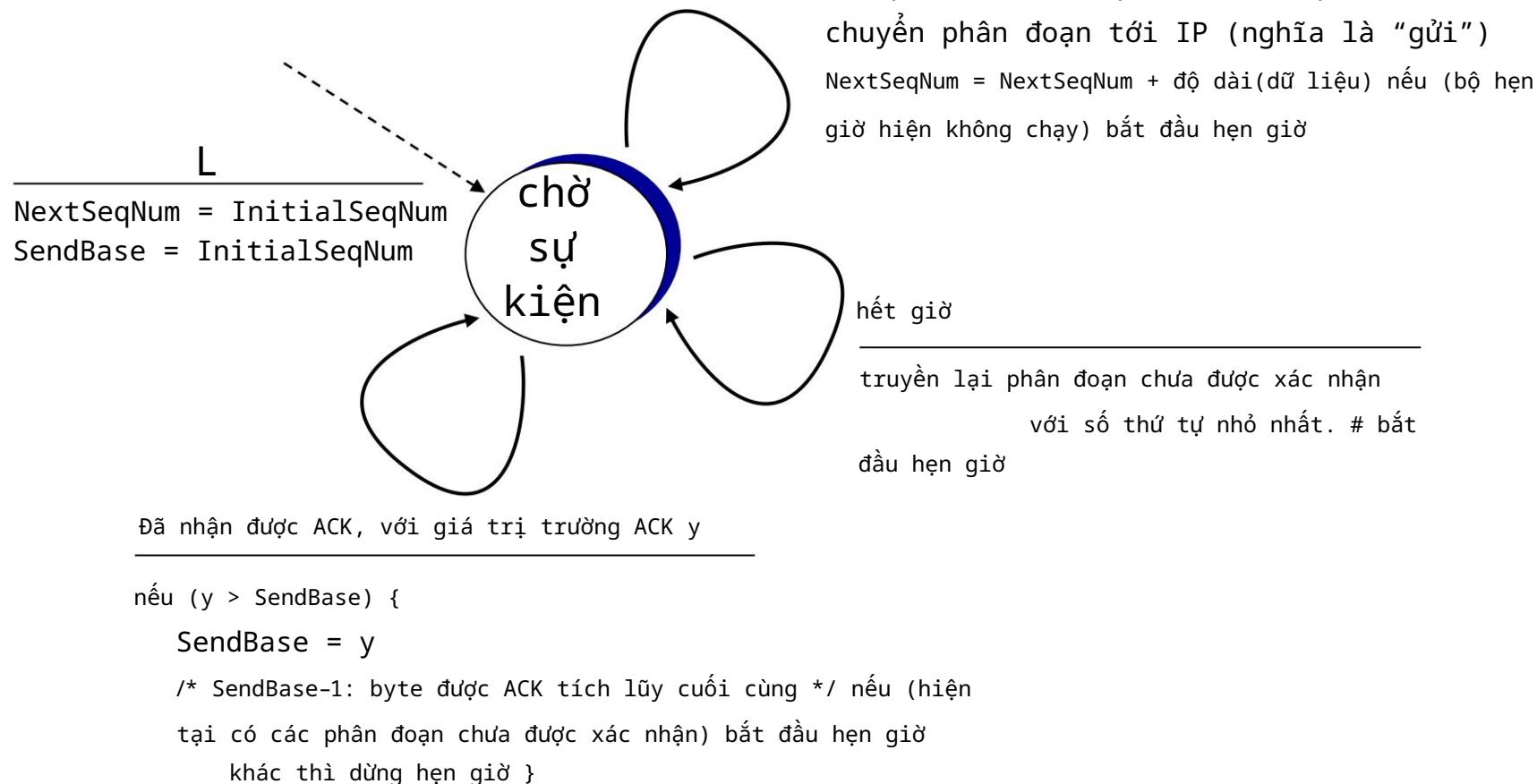
Go-Back-N: FSM mở rộng của máy thu



Chỉ ACK: luôn gửi ACK cho gói đã nhận đúng với **thứ tự** cao nhất seq # có thể tạo ra các ACK trùng lặp • chỉ cần nhớ số thứ tự dự kiến gói không

- theo thứ tự: • loại bỏ (không lưu vào bộ đệm): **không có bộ đệm của người nhận!** • ACK lại gói có số thứ tự cao nhất #

Người gửi TCP (đơn giản hóa)



Bắt tay 3 bước TCP FSM

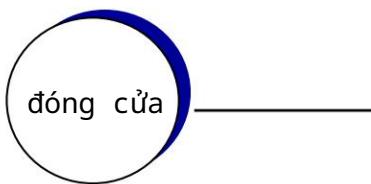
Kết nối ở cămSocket =

```
welcomeSocket.accept();
```

L

TỔNG HỢP(x)

SYNACK(seq=y,ACKnum=x+1) tạo ở
cắm mới để liên lạc lại với máy
khách



ở căm clientSocket =

```
newSocket("tên máy chủ","số cổng");
```

SYN(seq=x)

SYN
rcvd

ACK(ACKnum=y+1)

L

TỔNG HỢP (seq=y,ACKnum=x+1)

ACK(ACKnum=y+1)

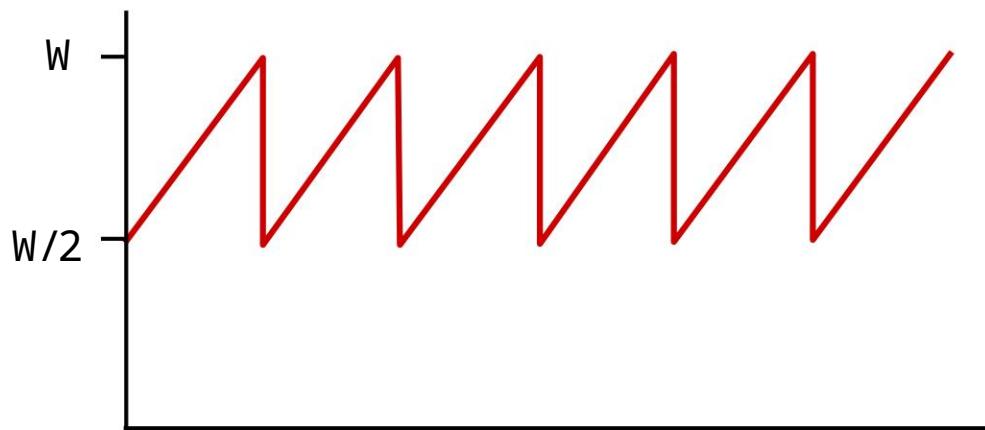
Thông lượng TCP

trung bình Thông lượng TCP là chức năng của kích thước cửa sổ, RTT? • bỏ qua khởi động chậm, giả sử luôn có dữ liệu để gửi

W : kích thước cửa sổ (được đo bằng byte) nơi xảy ra mất mát

- trung bình kích thước cửa sổ (# byte trong chuyến bay)
- là $\frac{3}{4}W$ trung bình. thông lượng là $\frac{3}{4}W$ mỗi RTT

$$\text{3 thông lượng TCP trung bình} = \frac{3}{4} \frac{W}{RTT} \text{ byte/giây}$$



TCP qua “đường ống dài, béo”

ví dụ: phân đoạn 1500 byte, RTT 100ms, muốn có thông lượng 10 Gbps yêu cầu
 $W = 83.333$ phân đoạn trong chuyến bay thông lượng xét về xác suất mất phân
 đoạn, L [Mathis 1997]:

$$\text{Thông lượng MSS TCP} = \frac{1.22}{RTT}$$


để đạt được thông lượng 10 Gbps, cần tỷ lệ hao hụt là $L = 2 \cdot 10^{-10}$ – một
tỷ lệ hao hụt rất nhỏ! các phiên bản TCP cho các tình huống dài, tốc độ
 cao