TRƯỜNG ĐẠI HỌC THỦY LỢI
KHOA CÔNG NGHỆ THÔNG TIN

Mạng máy tính

TS. Đoàn Thị Quế

Chương 3: Tầng giao vận

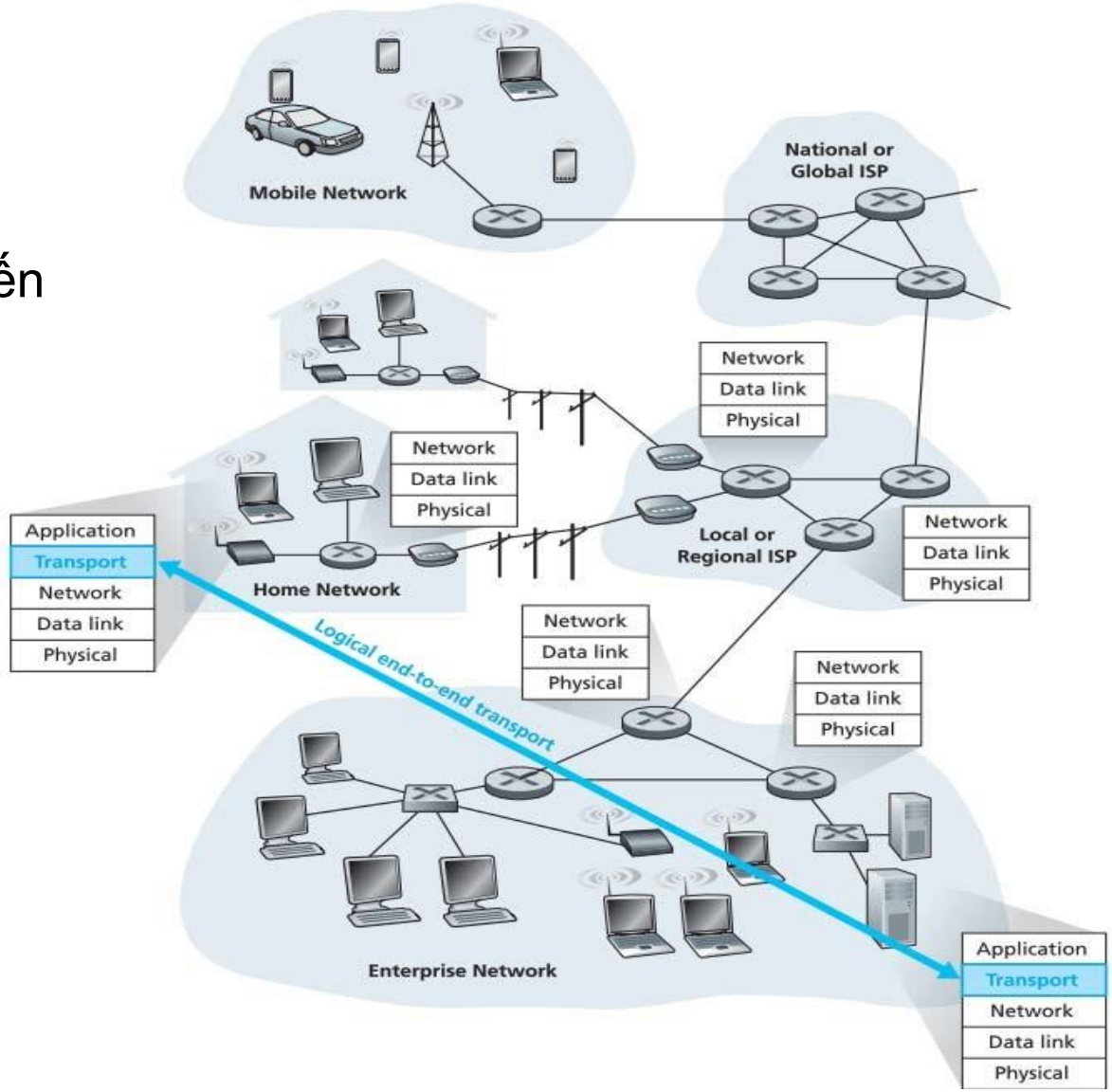
- Tổng quan về tầng giao vận
- Dồn kênh và phân kênh
- Giao thức UDP
- Truyền dữ liệu tin cậy
- Giao thức TCP

Tổng quan về tầng giao vận

- Dịch vụ và giao thức tầng giao vận
- Quan hệ giữa tầng giao vận và tầng mạng
- Tầng giao vận trong Internet

Dịch vụ và giao thức tầng giao vận

- Giao thức giao vận cung cấp *kênh truyền lôgic* giữa các tiến trình ứng dụng chạy trên các host khác nhau
- Giao thức giao vận chạy trong các end system
  - phía gửi: chia app. message thành các *segment*, chuyển tới tầng mạng
  - phía nhận: ghép các segment lại thành message, chuyển tới tầng ứng dụng
- Có nhiều hơn một giao thức giao vận cho các ứng dụng
  - Internet: TCP và UDP



Dịch vụ và giao thức tầng giao vận

- Các dạng dịch vụ giao vận:
  - Dồn kênh (multiplex) và phân kênh (demultiplex)
  - Đảm bảo thông lượng
  - Đảm bảo độ trễ
  - Truyền dữ liệu tin cậy
  - …

Quan hệ giữa tầng giao vận và tầng mạng

- *tầng giao vận:* cung cấp đường truyền lôgic giữa các tiến trình chạy trên các host
- *tầng mạng:* cung cấp đường truyền lôgic giữa các host

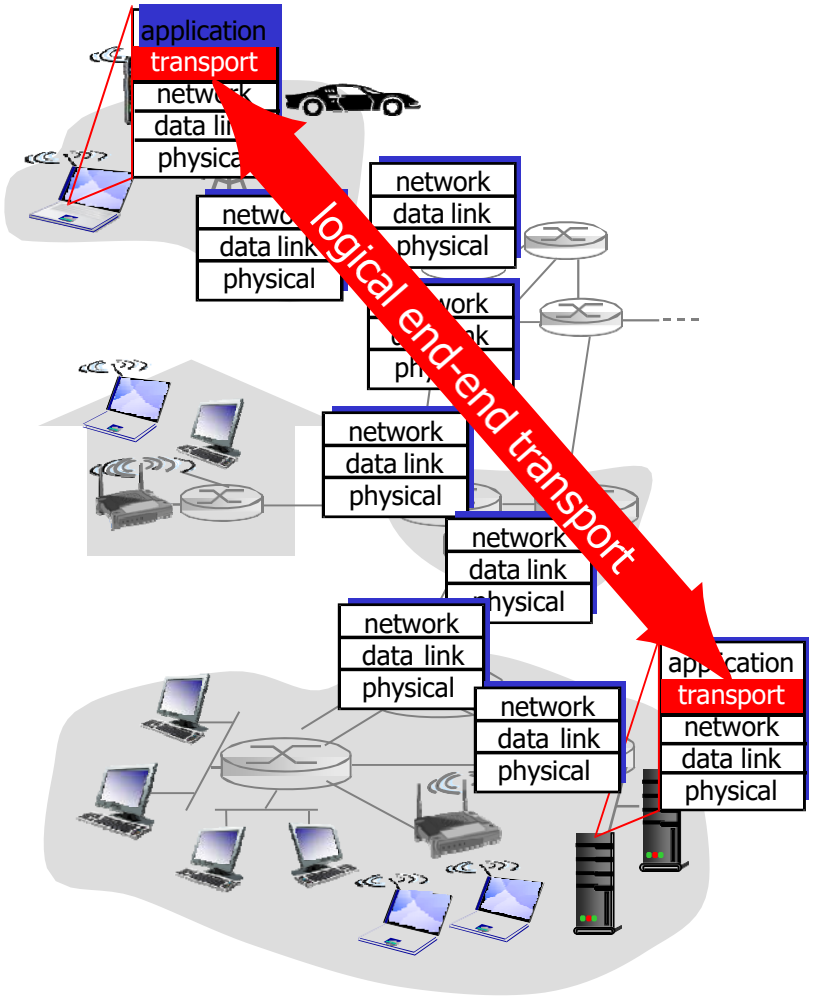
một so sánh:

12 trẻ trong nhà của An gửi thư cho 12 trẻ trong nhà của Bình:

- host = nhà
- tiến trình = trẻ
- Thông điệp = thư trong phong bì thư
- giao thức giao vận = An và Bình
- giao thức tầng mạng = dịch vụ thư bưu điện

Tầng giao vận của Internet

- Giao thức tầng giao vận của Internet
  - TCP (Transmission Control Protocol)
  - UDP(User Datagram Protocol)
- Hai dạng dịch vụ giao vận của Internet
  - Truyền dữ liệu tin cậy: TCP
  - Truyền dữ liệu không tin cậy: UDP
- Tầng giao vận Internet không cung cấp các dịch vụ:
  - đảm bảo độ trễ
  - đảm bảo thông lượng



Tầng giao vận của Internet

- Ứng dụng và dịch vụ giao vận

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

Chương 3: Tầng giao vận

- Tổng quan về tầng giao vận
- Dồn kênh và phân kênh
- Giao thức UDP
- Truyền dữ liệu tin cậy
- Giao thức TCP

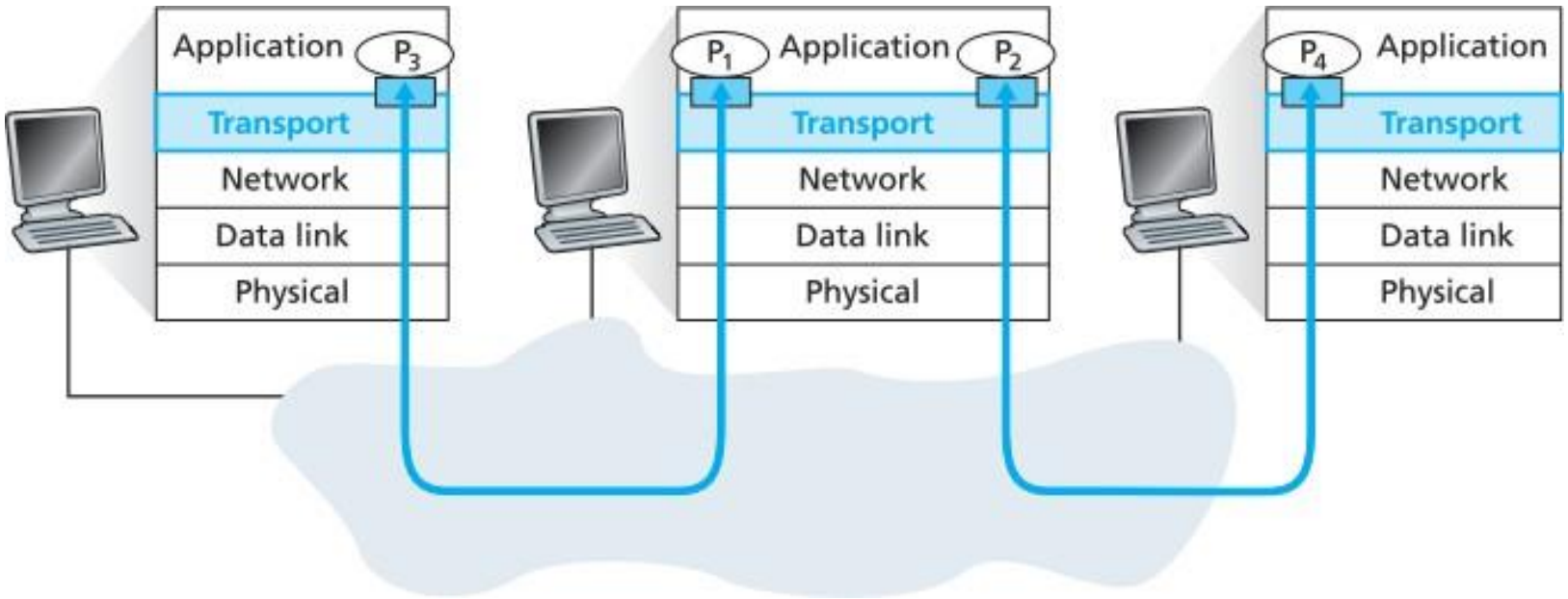
Dồn kênh và phân kênh

Dồn kênh (multiplexing) tại nút gửi

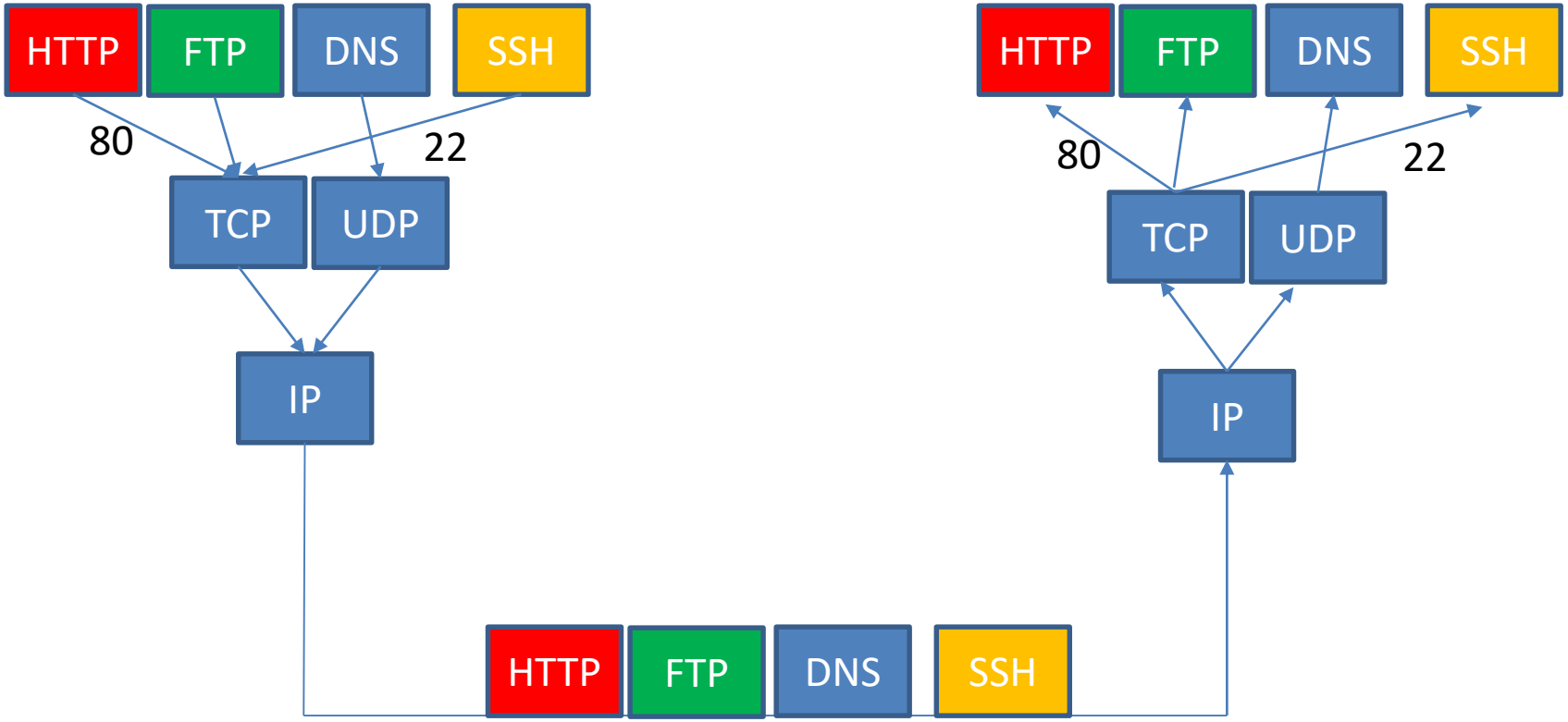
Tầng giao vận nhận dữ liệu từ nhiều tiến trình ứng dụng, bổ sung thêm tiêu đề của tầng giao vận để tạo thành các segment và gửi các segment này xuống tầng mạng

Phân kênh (demultiplexing) tại nút nhận

Tầng giao vận sử dụng thông tin header để chuyển dữ liệu trong segment đã nhận tới đúng tiến trình ứng dụng



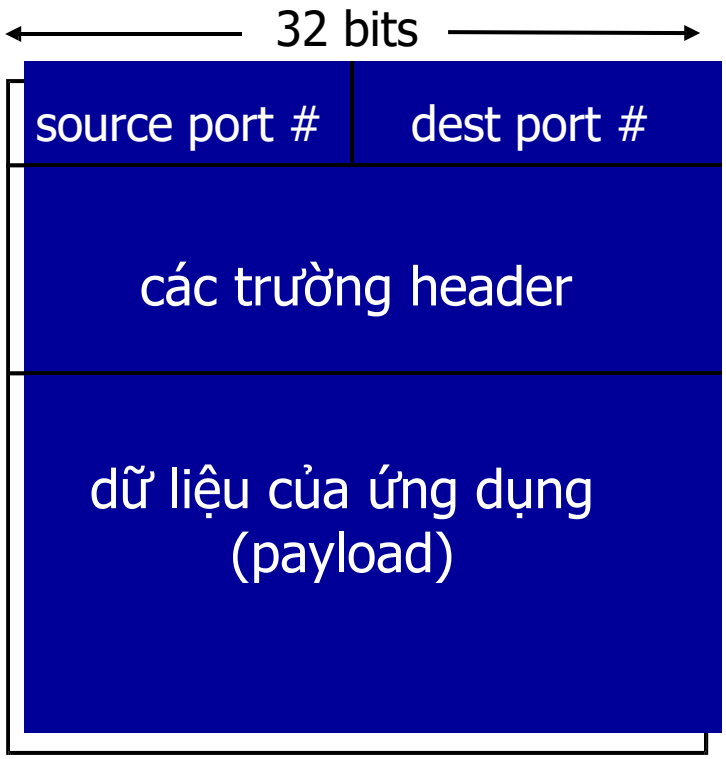
Dồn kênh và phân kênh



Dồn kênh và phân kênh hoạt động như thế nào?

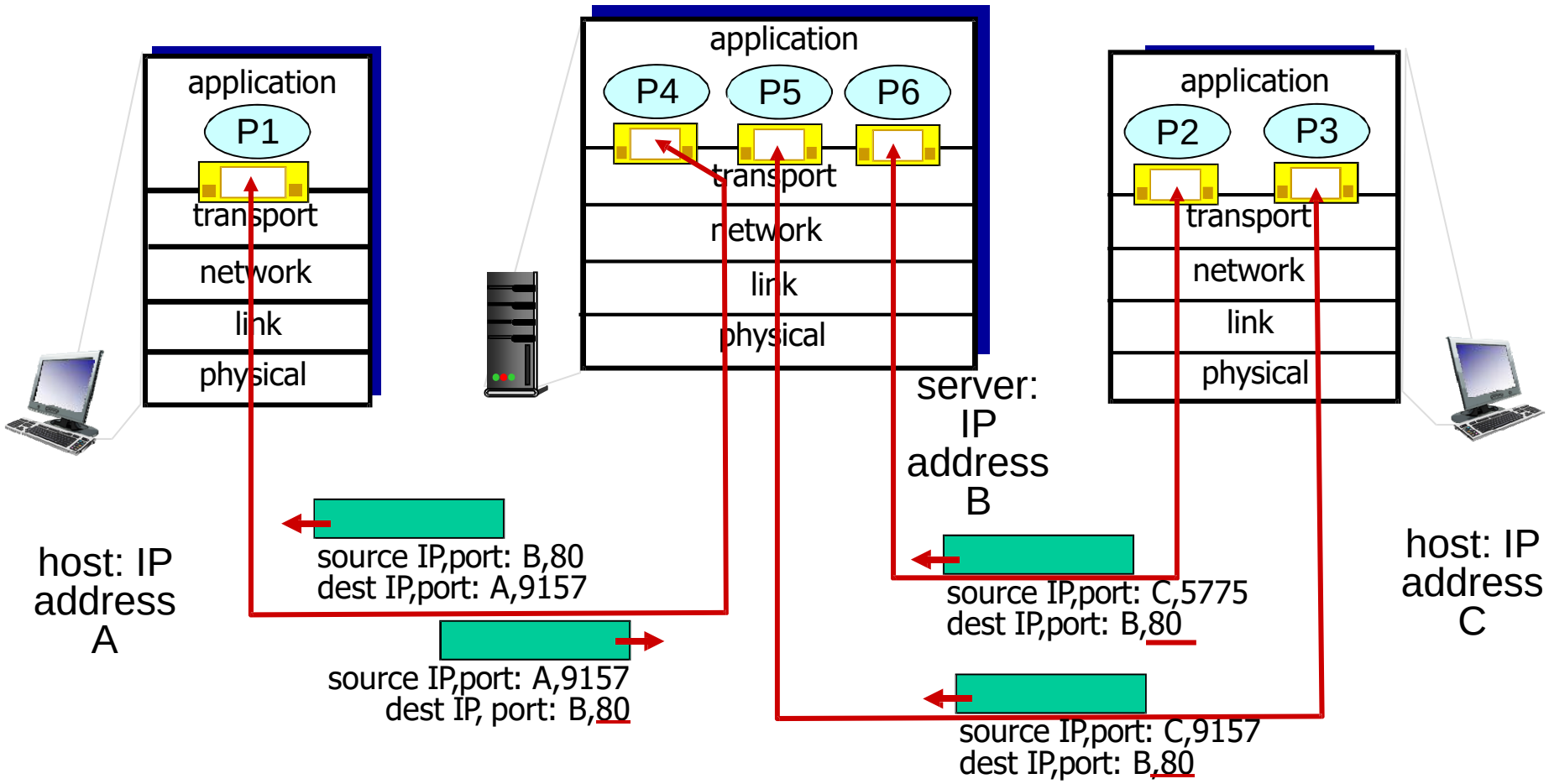
- Dồn kênh và phân kênh nhờ hai trường đặc biệt ở đầu segment
  - Số hiệu cổng nguồn (source port number)
  - Số hiệu cổng đích (dest. port number)

  Số hiệu cổng tham khảo trong RFC 1700
- Mỗi datagram ở tầng mạng có: Địa chỉ IP nguồn và IP đích để xác định host gửi và host nhận
- host sử dụng địa chỉ *IP và port number* để chuyển segment tới socket thích hợp



cấu trúc TCP/UDP segment

Minh họa về dồn kênh và phân kênh



3 segments, gửi tới IP address: B,
dest port: 80 được tách kênh tới các socket khác nhau

Chương 3: Tầng giao vận

- Tổng quan về tầng giao vận
- Dồn kênh và phân kênh
- Giao thức UDP
- Truyền dữ liệu tin cậy
- Giao thức TCP

Giao thức UDP

- RFC 768
- Giao thức không hướng kết nối (connectionless)
  - Không cần thiết lập kết nối
- Tiêu đề gói tin nhỏ: 8 bytes (so với 20 bytes của TCP)
- UDP có những chức năng cơ bản gì?
  - Dồn kênh/ phân kênh
  - Phát hiện lỗi bit đơn giản (dùng checksum)

UDP checksum

Mục đích: phát hiện lỗi bit trong segment đã gửi

Bên gửi:

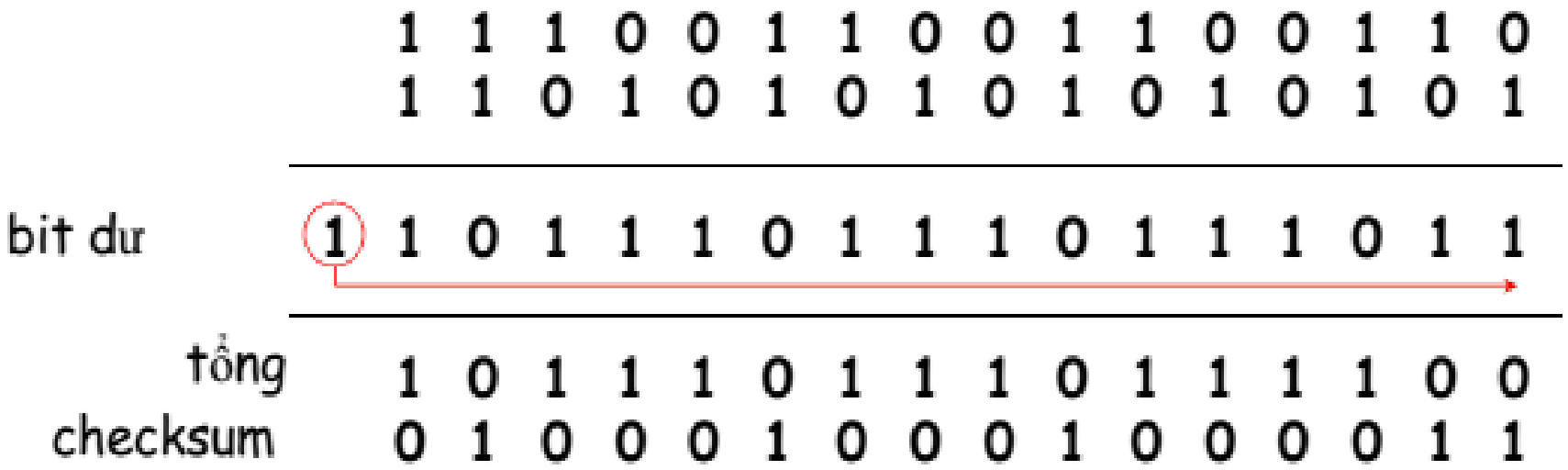
- coi dữ liệu của segment, bao gồm cả header gồm các từ 16 bít
  - Tính giá trị bù một của tổng các từ 16 bit trong segment (RFC 1071)
  - Giá trị checksum nhận được được đặt vào trường checksum trong gói dữ liệu UDP

Bên nhận:

- Tính tổng của các từ 16 bit (cả checksum)
  - Nếu tổng thu được *toàn các bit 1*, cho biết dữ liệu nhận được *không có lỗi*
  - Nếu tổng thu được có một bit nào đó bằng 0, cho biết dữ liệu nhận được *có lỗi*

Ví dụ: Internet Checksum

- Giả sử có 2 từ 16 bit



- Lưu ý: khi cộng các số, bit nhớ cao nhất cần được cộng vào kết quả

Ví dụ: Internet Checksum

- Dữ liệu truyền đi: 2 từ 16 bit + 16 bit checksum

1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	Từ 1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	Từ 2
0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	Checksum

- Dữ liệu nhận được không có lỗi: cộng tất cả các từ 16 bit kể cả checksum thì thu được tổng gồm các bit 1

Từ 1		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
Từ 2	+	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
bit dư thừa		(1)	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
checksum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
	+	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Ví dụ: Internet Checksum

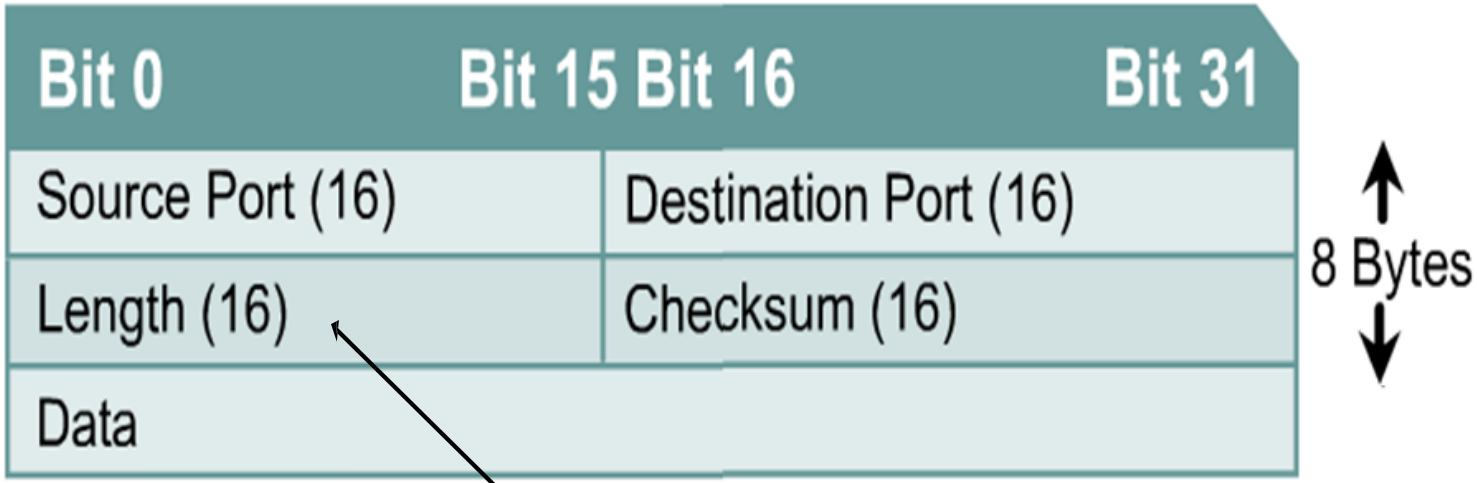
- Dữ liệu truyền đi: 2 từ 16 bit + 16 bit checksum

1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	Từ 1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	Từ 2
0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	Checksum

- Dữ liệu nhận được có lỗi: cộng tất cả các từ 16 bit kể cả checksum thì thu được tổng có một bit nào đó bằng 0

Từ 1		1	1	1	0	0	1	1	0	0	1	0	0	0	1	1	0
Từ 2	+	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
bit dư thừa	(1)	1	0	1	1	1	0	1	1	1	0	0	1	1	0	1	1
		1	0	1	1	1	0	1	1	1	0	0	1	1	1	0	0
checksum	+	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1
		1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1

Cấu trúc UDP segment



Length: độ dài toàn bộ gói UDP segment (bao gồm cả header), tính theo bytes

Các vấn đề của UDP

- Không có kiểm soát tắc nghẽn
  - UDP cứ gửi dữ liệu nhanh nhất, nhiều nhất có thể

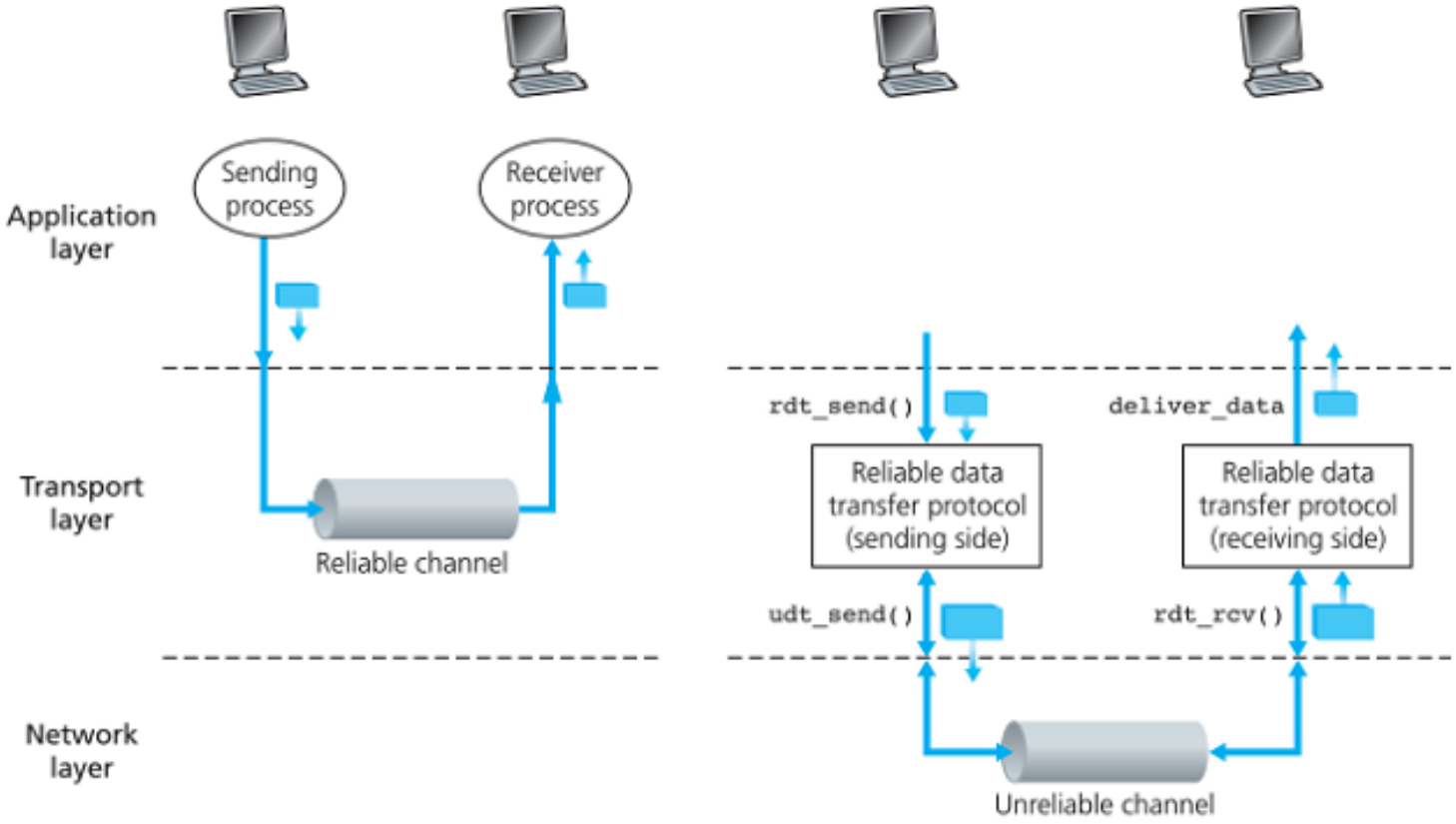
  $\Rightarrow$làm Internet quá tải
- Không đảm bảo độ tin cậy
  - Các ứng dụng phải cài đặt cơ chế tự kiểm soát độ tin cậy
  - Việc phát triển ứng dụng sẽ phức tạp hơn

Chương 3: Tầng giao vận

- Tổng quan về tầng giao vận
- Dồn kênh và phân kênh
- Giao thức UDP
- Truyền dữ liệu tin cậy
- Giao thức TCP

Sơ đồ cấu trúc của quá trình truyền dữ liệu tin cậy:

- Đường truyền tin cậy:
  1) Dữ liệu không bị lỗi
  2) Dữ liệu không bị mất
  3) Dữ liệu nhận được theo đúng trình tự gửi



a) Dịch vụ cung cấp

b) Cài đặt dịch vụ

rdt - reliable data transfer protocol
udt - unreliable data transfer protocol

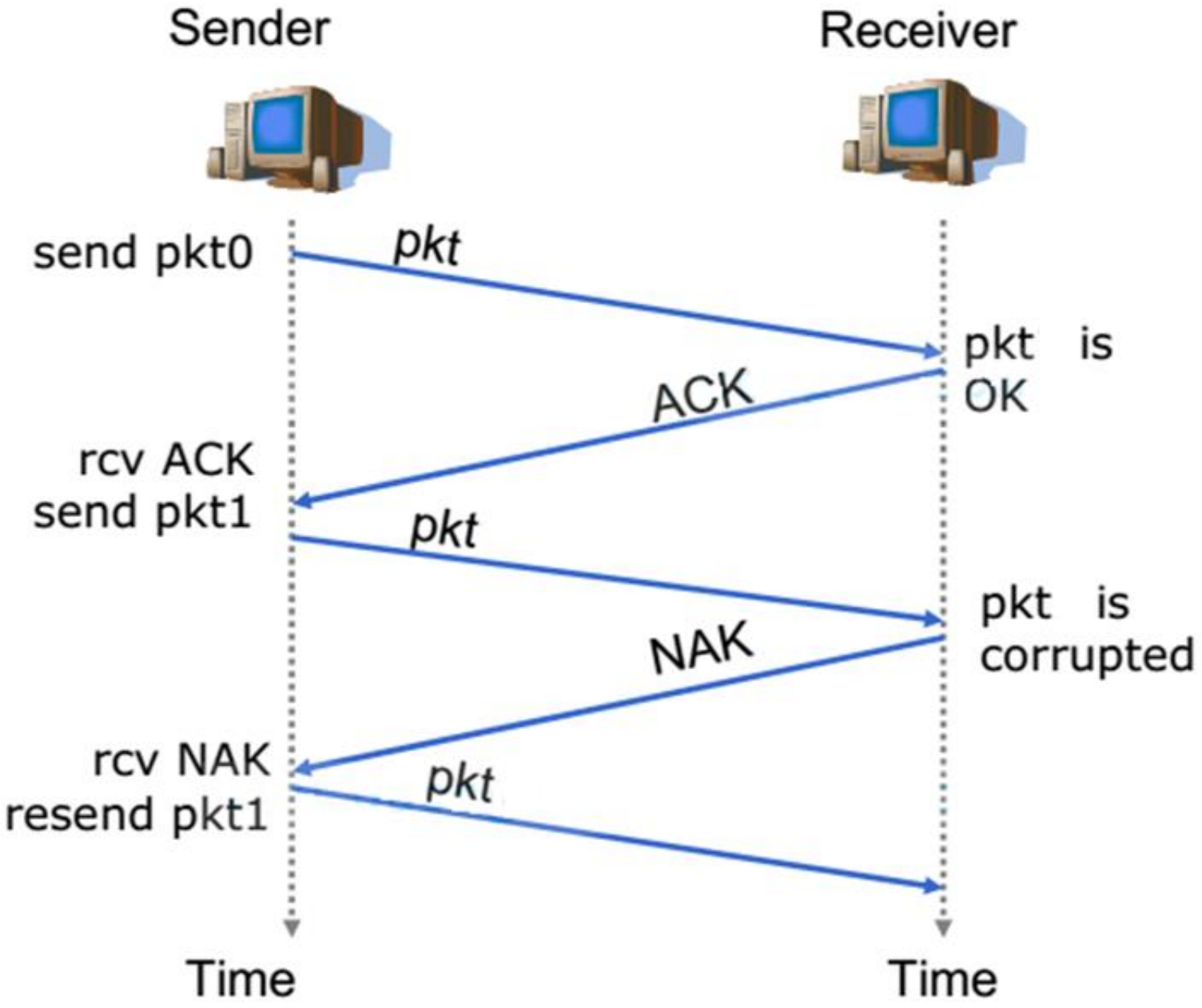
Xây dựng giao thức truyền dữ liệu tin cậy

- Cách tiếp cận:
  - từng bước phát triển giao thức truyền dữ liệu tin cậy giữa bên gửi và bên nhận

Kênh có lỗi bit, không bị mất tin

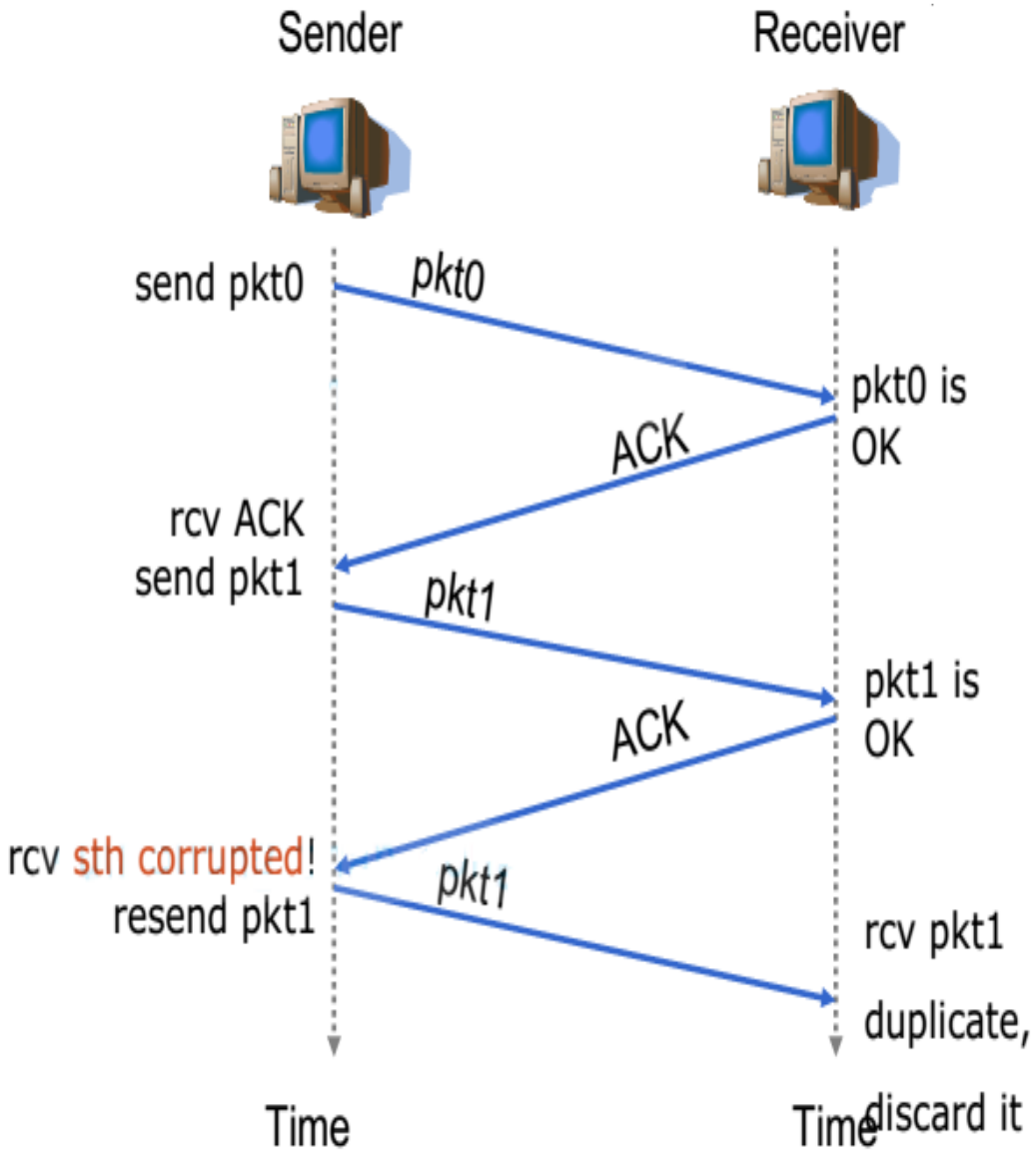
- Phát hiện lỗi?
  - Ví dụ, dùng checksum để phát hiện bit bị lỗi
- Làm thế nào để báo cho bên gửi?
  - *acknowledgements (ACKs):* bên nhận thông báo cho bên gửi là gói tin đã nhận không có lỗi
  - *negative acknowledgements (NAKs):* bên nhận thông báo cho bên gửi là gói tin có lỗi
- Phản ứng của bên gửi?
  - bên gửi truyền lại gói tin khi nhận được NAK

Hoạt động


Sender
Receiver
send pkt0
pkt
pkt is OK
ACK
rcv ACK
send pkt1
pkt
pkt is corrupted
NAK
rcv NAK
resend pkt1
pkt
Time
Time

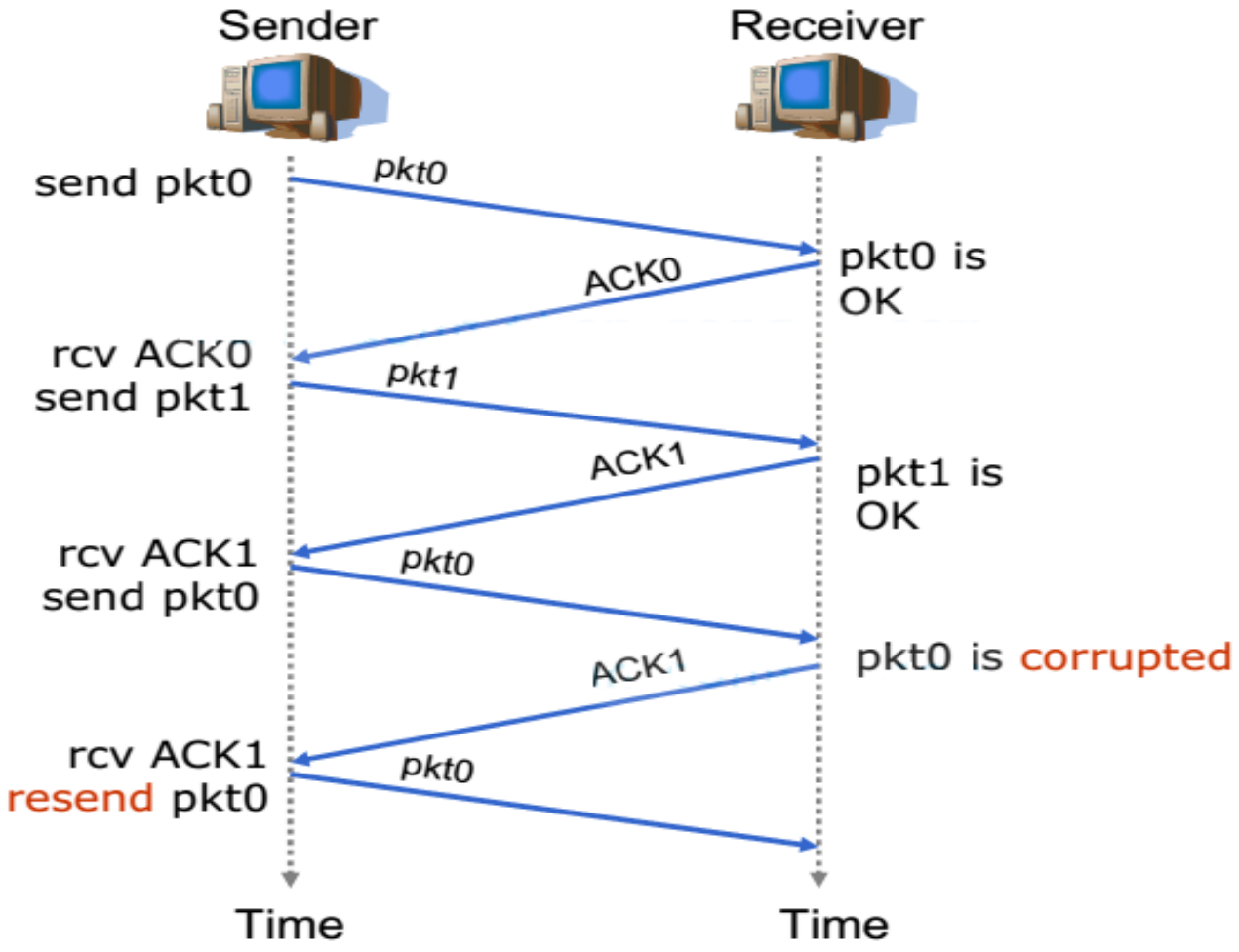
Lỗi ACK/NAK

- Cần truyền lại
- Xử lý việc lặp gói như thế nào?
  - Bên gửi: thêm seq #
  - Bên nhận: hủy packet trùng lặp



Giải pháp không dùng NAK

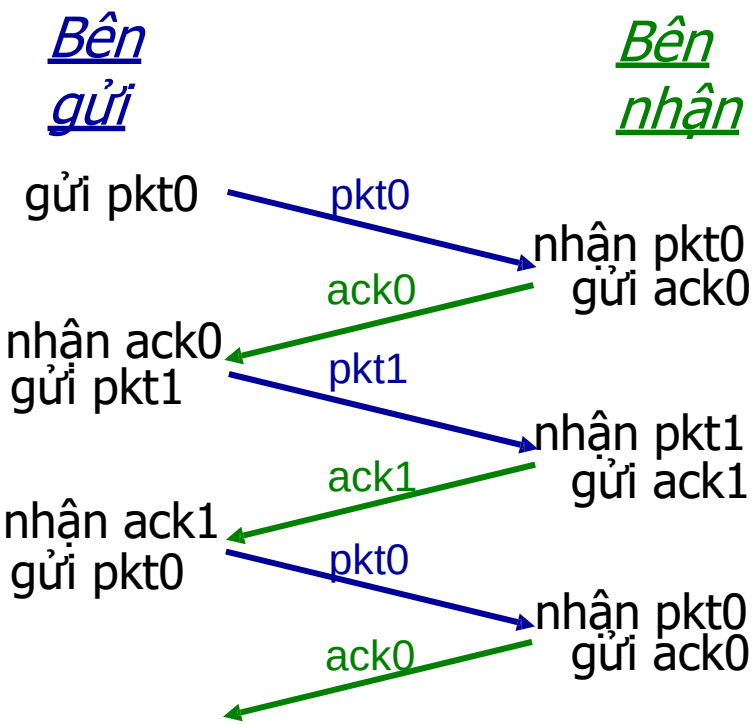
- Dùng ACK cho gói tin cuối cùng đã nhận được đúng
- Nếu nhận 2 ACK cho cùng một gói dữ liệu=> bên gửi gửi lại gói dữ liệu



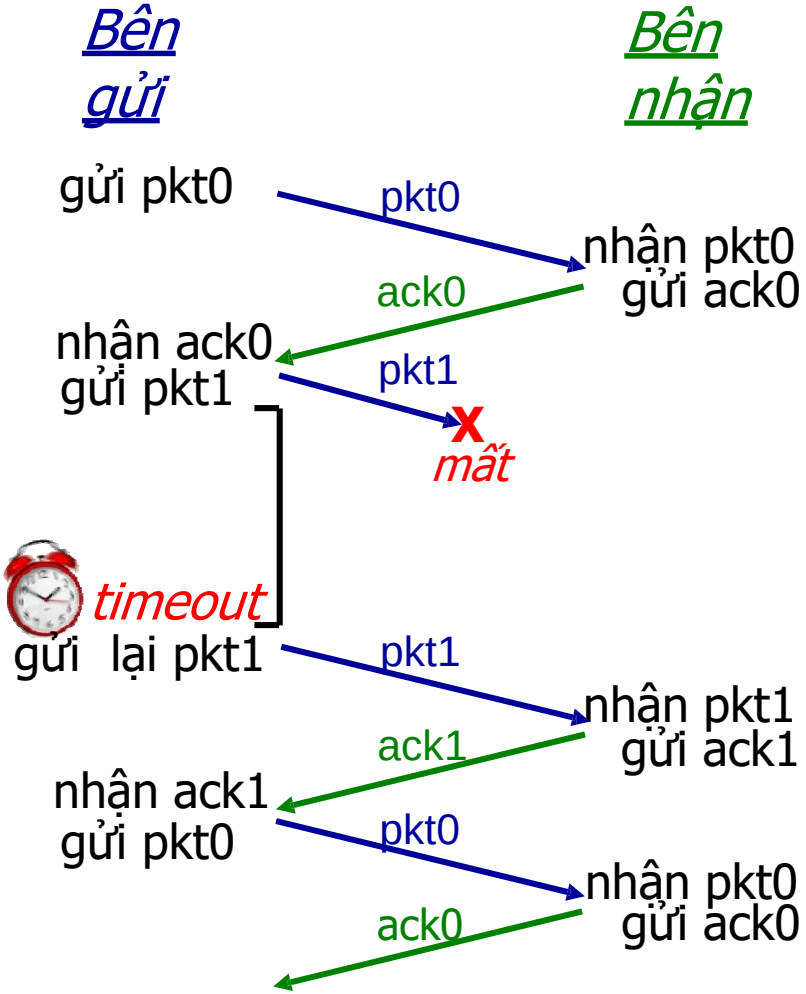
Kênh có lỗi và mất gói tin

- Dữ liệu và ACK có thể bị mất
  - bên nhận đợi một khoảng thời gian hợp lý (thời gian chờ - timeout) cho ACK
  - truyền lại nếu không có ACK tới trong thời gian này
- Thời gian chờ là bao lâu?
  - Ít nhất là 1 RTT (Round Trip Time)
  - Mỗi gói tin gửi đi cần 1 timer
- Nếu gói tin vẫn đến đích và ACK bị mất?
  - Truyền lại gói tin và xảy ra trùng lặp gói tin => Dùng số hiệu gói tin để phát hiện trùng lặp

Minh họa

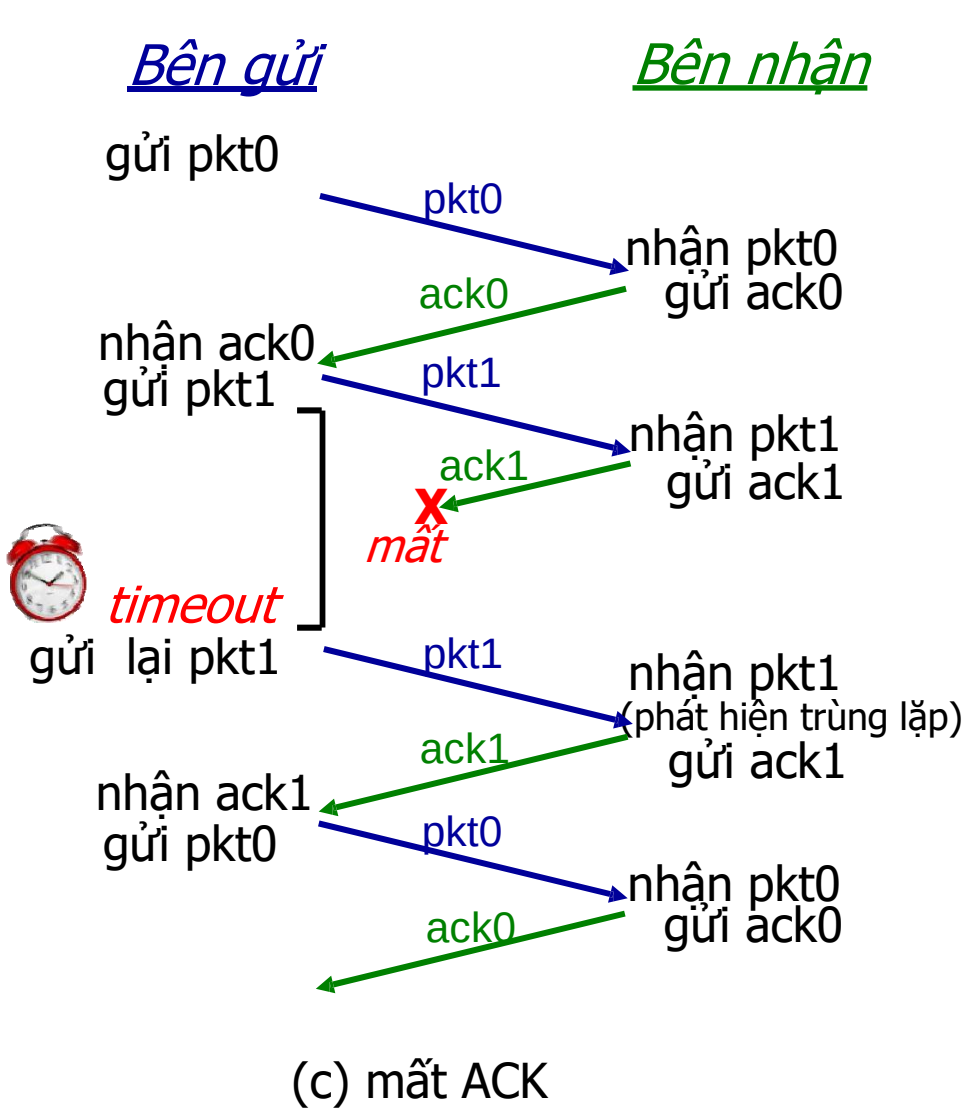


(a) không mất gói

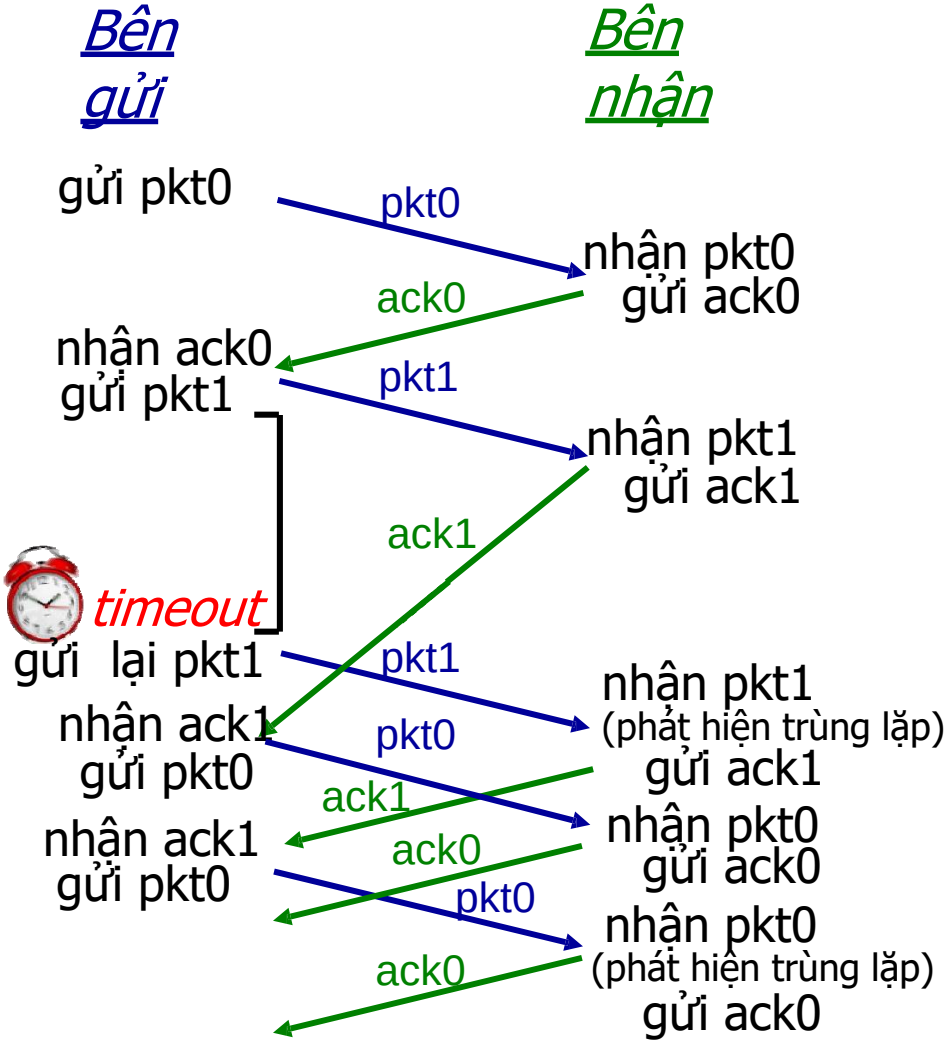


(b) có mất gói

Minh họa

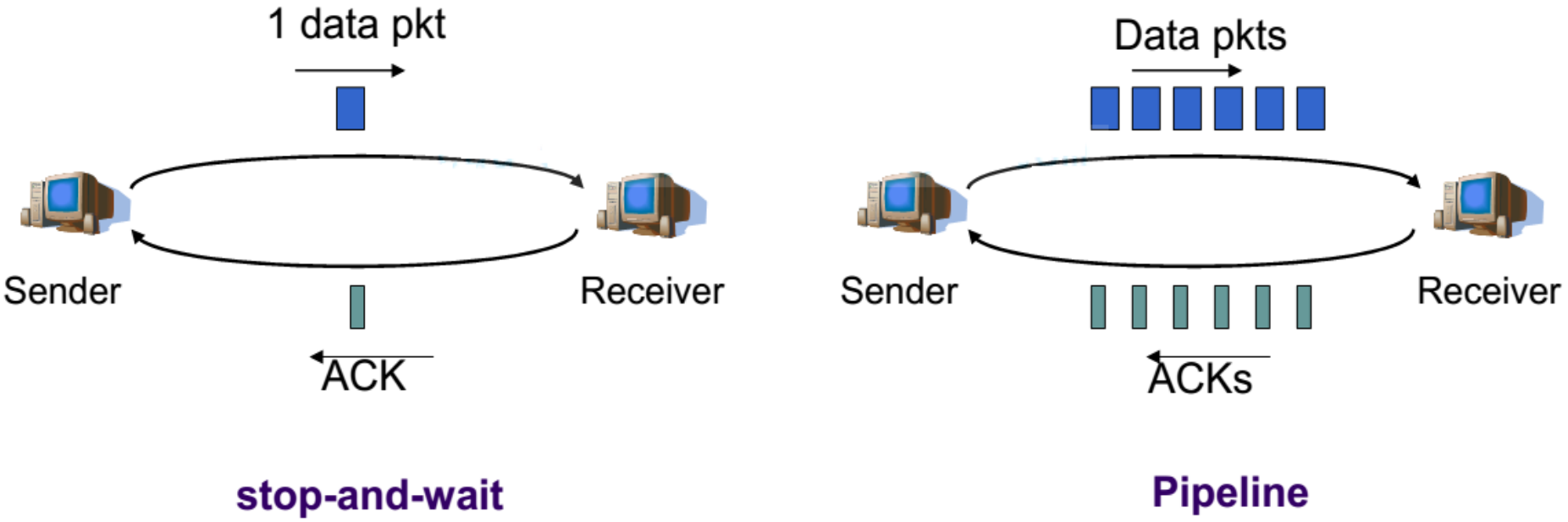


(c) mất ACK



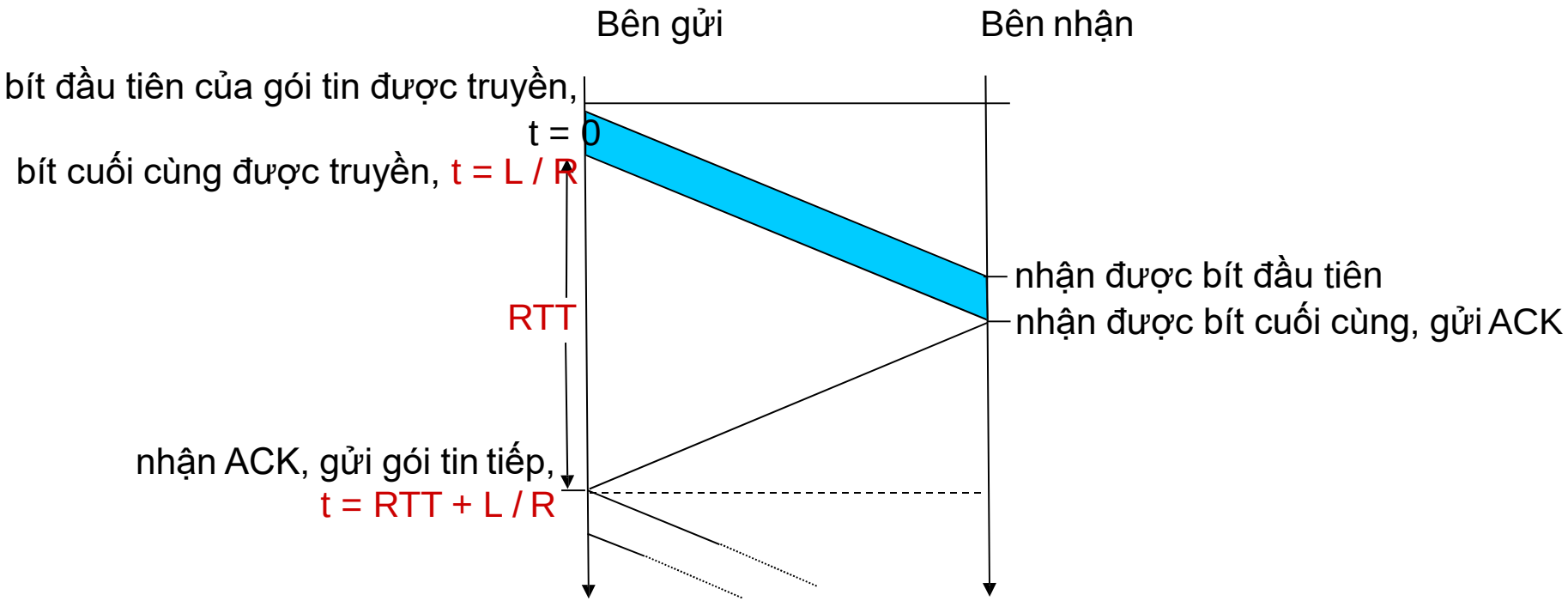
(d) timeout quá ngắn/ ACK tới trễ

Truyền theo kiểu pipeline



- Giao thức được xử lý liên tục (Pipelined protocols): bên gửi cho phép nhiều gói tin chưa được ACK
  - cần tăng dải sequence number
  - vùng đệm tại bên gửi và bên nhận
- Hai dạng của pipelined protocols: *go-Back-N, selective repeat*

Hiệu năng của kiểu xử lý stop-and-wait



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

U_{sender}: *utilization* – phần thời gian bên gửi thực hiện gửi

$$U_{sender} = \frac{L / R}{RTT + L / R}$$

Hiệu năng của kiểu xử lý stop-and-wait

- Liên kết 1 Gbps, độ trễ lan truyền 15 ms, gói tin 8000 bit

$$d_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\ microsecs$$
$$0{,}008\ ms$$

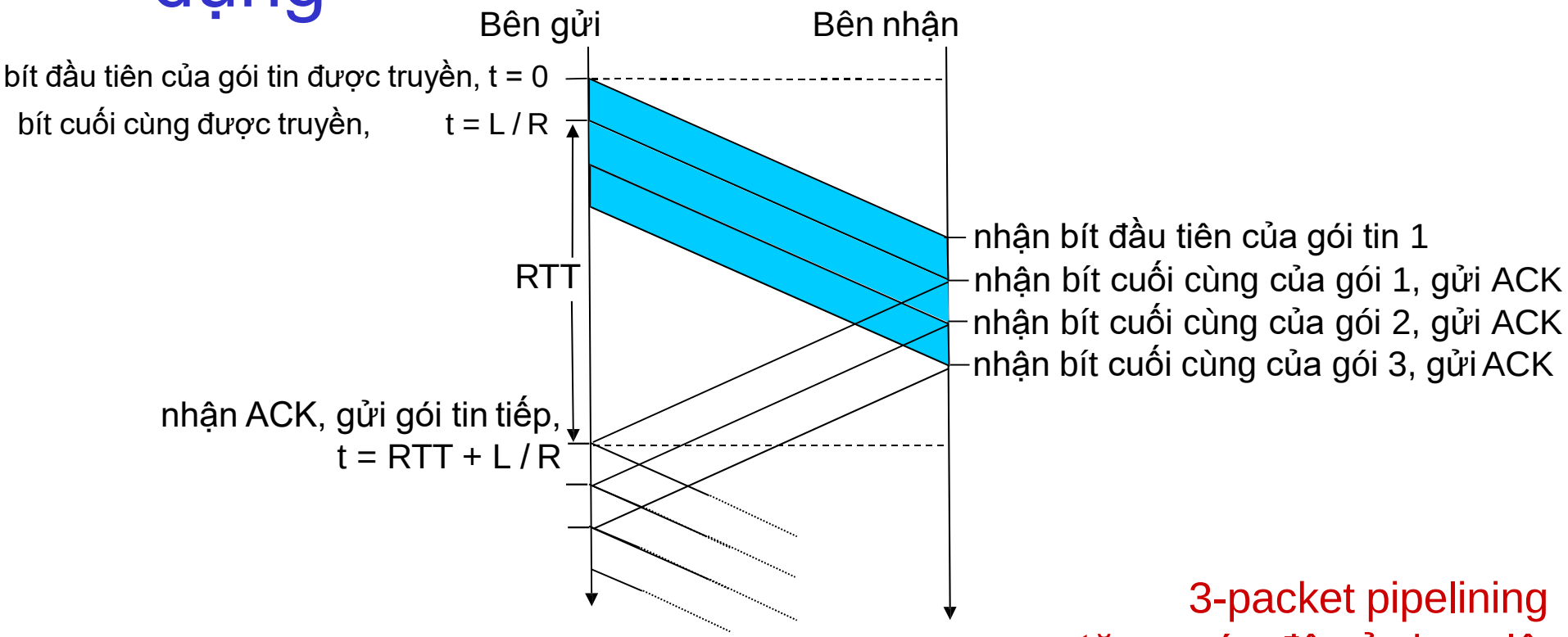
- *RTT =15 +15 = 30 ms*

$$U_{sender} = \frac{L/R}{RTT + L/R} \approx \frac{0{,}008}{30{,}008} \approx 0{,}000267$$

- nếu *RTT=30 ms*, gói tin 8000 bit mất *30 ms*: 1s truyền được $8000/30.10^{-3} \approx 267$ Kbit trên liên kết 1Gbps = 10^9 bit/s

➔ kiểu stop-and-wait hạn chế việc sử dụng tài nguyên băng thông của đường truyền

Hiệu năng của Pipeline: Tăng mức độ sử dụng

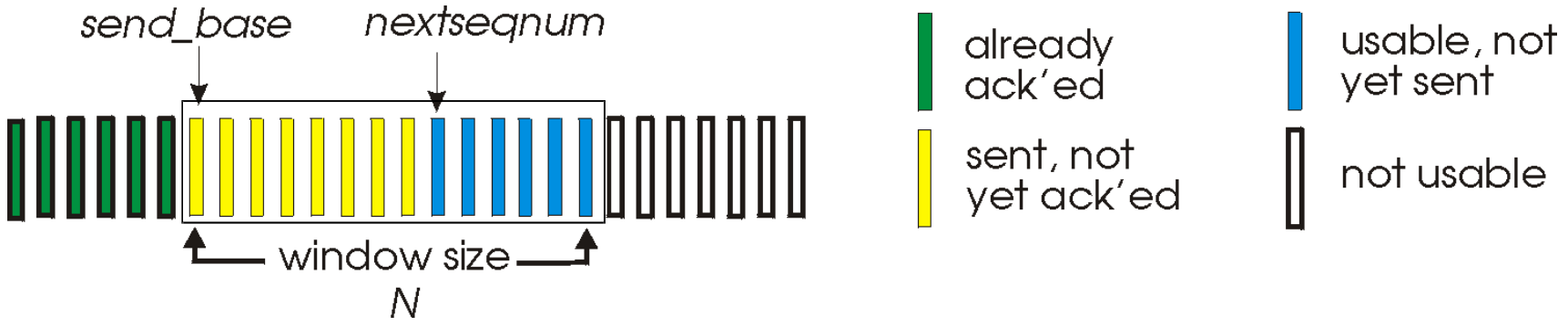


3-packet pipelining
tăng mức độ sử dụng lên 3 lần

$$U_{sender} = \frac{3L / R}{RTT + L / R} \approx \frac{0{,}024}{30{,}008} \approx 0{,}000801$$

Go-Back-N: Bên gửi

- "window" kích thước N: có thể gửi N gói tin liên tục chưa được ack

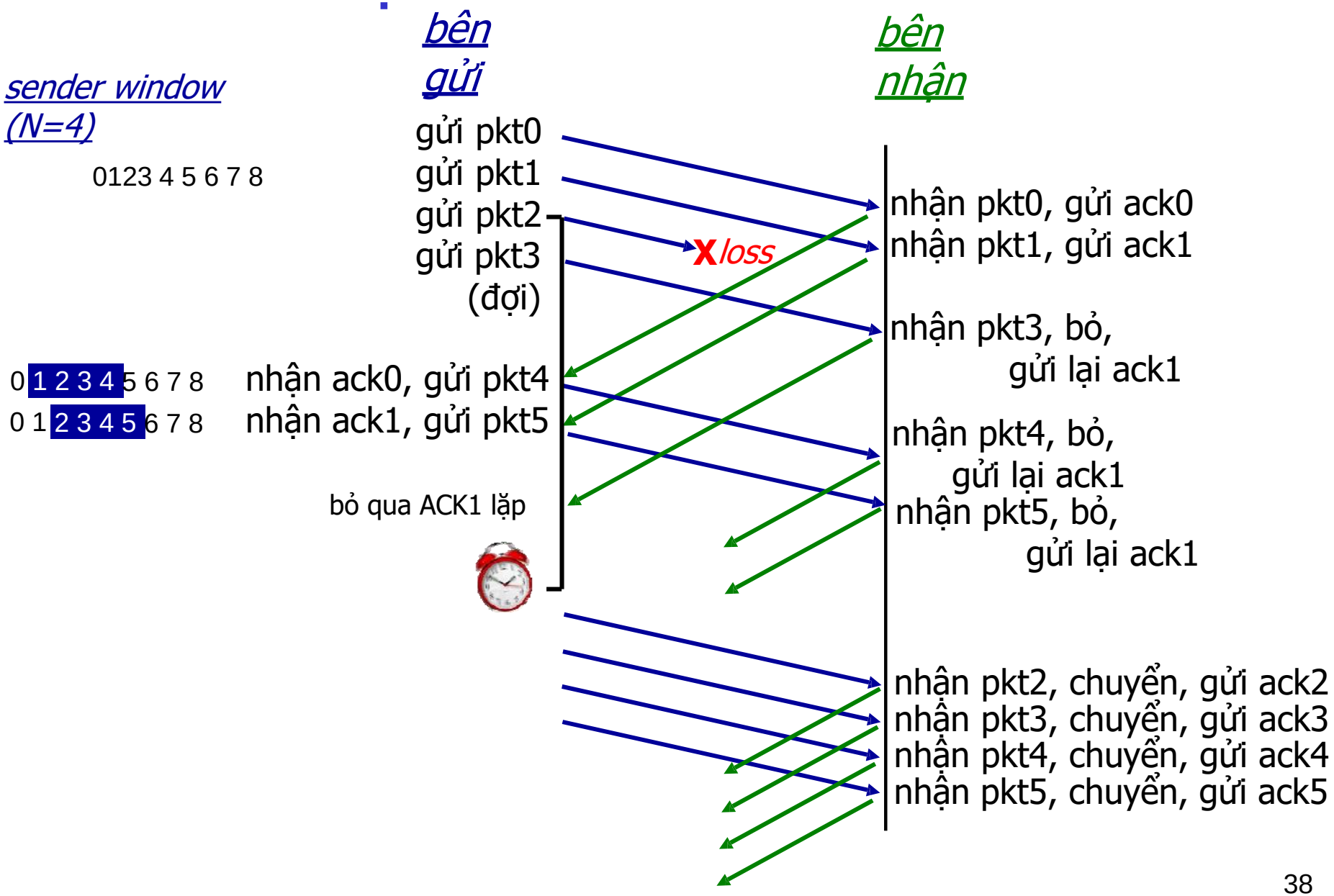


- Chỉ gửi các gói có số hiệu trong cửa sổ (gói 0 đến gói N-1), sau đó đợi xác nhận cho các gói tin này trước khi tiếp tục gửi gói, cửa sổ "dịch" sang phải mỗi khi nhận được ACK
- ACK(*n*): xác nhận tất cả các gói tin tới số trình tự *n* đã được nhận - *"cumulative ACK"*
- Đặt đồng hồ cho gói tin cũ nhất chờ ack
  - Khi hết *timeout(n):* gửi lại gói tin *n* và tất cả gói tin có seq # lớn hơn trong window

Go-Back-N: Bên nhận

- Chỉ gửi 1 xác nhận ACK cho gói tin có số hiệu lớn nhất đã nhận được theo đúng thứ tự.
- Với các gói tin không theo thứ tự:
  - Hủy bỏ -> không lưu vào vùng đệm
  - Xác nhận lại gói tin với số hiệu lớn nhất còn đúng thứ tự

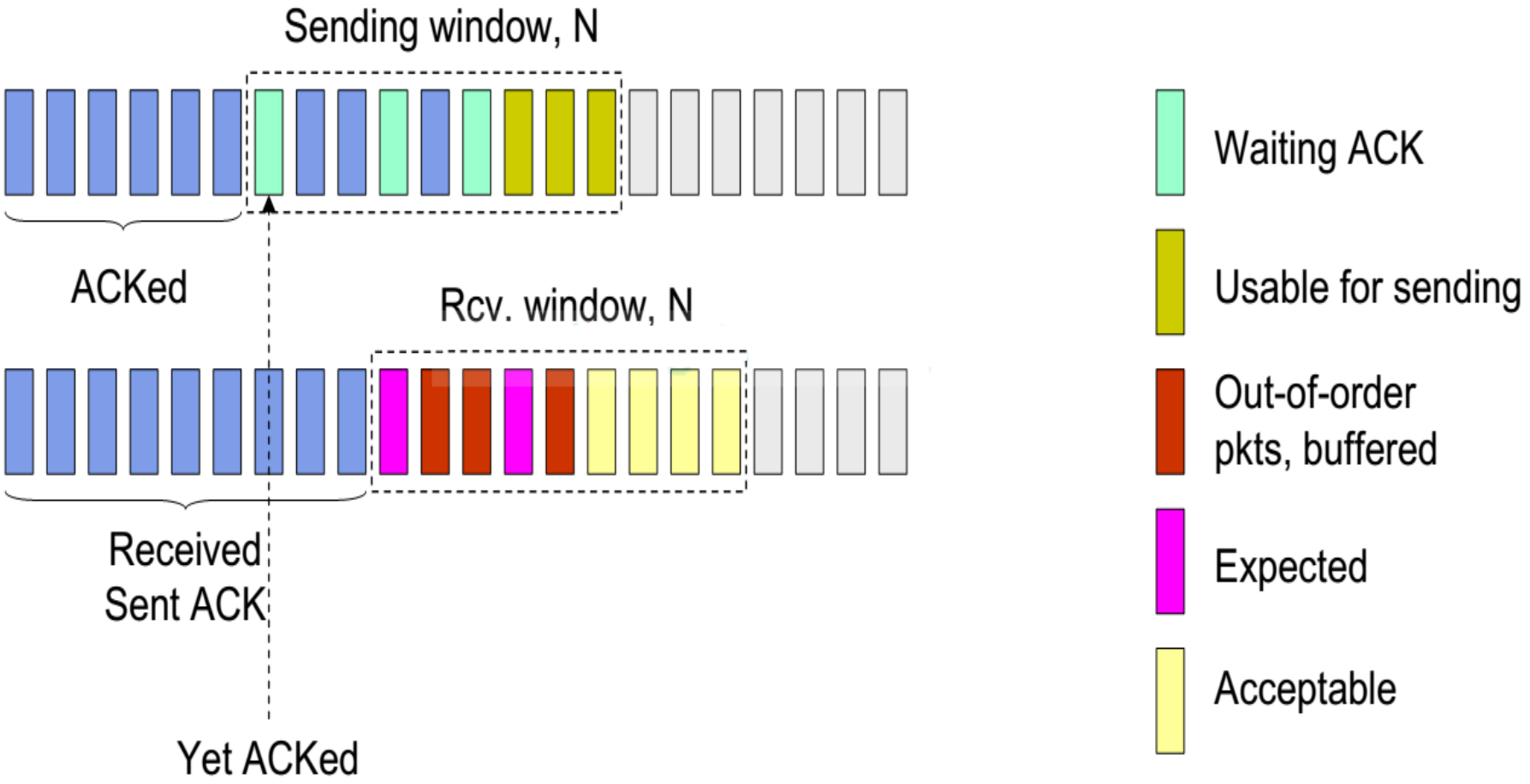
GBN: Ví dụ


bên gửi
bên nhận
sender window (N=4)
0123 4 5 6 7 8
gửi pkt0
gửi pkt1
gửi pkt2
gửi pkt3
(đợi)
X loss
nhận pkt0, gửi ack0
nhận pkt1, gửi ack1
nhận pkt3, bỏ,
gửi lại ack1
0 1 2 3 4 5 6 7 8
nhận ack0, gửi pkt4
0 1 2 3 4 5 6 7 8
nhận ack1, gửi pkt5
nhận pkt4, bỏ,
gửi lại ack1
nhận pkt5, bỏ,
gửi lại ack1
bỏ qua ACK1 lặp
nhận pkt2, chuyển, gửi ack2
nhận pkt3, chuyển, gửi ack3
nhận pkt4, chuyển, gửi ack4
nhận pkt5, chuyển, gửi ack5

Selective repeat

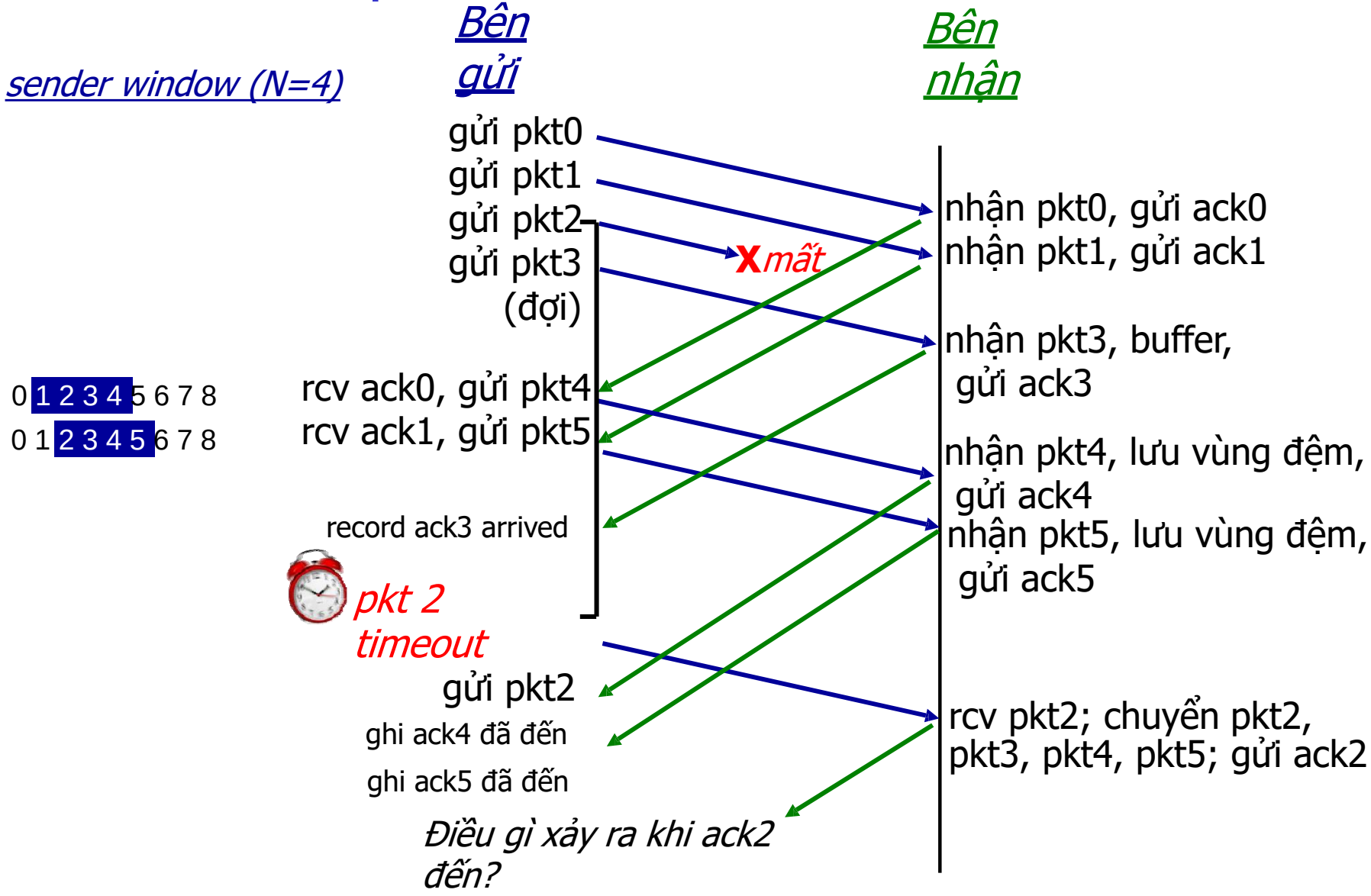
- Bên gửi: có thể gửi đến N gói tin chưa ack, sau đó chờ xác nhận
- Bên nhận: gửi xác nhận ack riêng lẻ cho từng gói tin nhận đúng
  - chứa gói tin vào vùng đệm, khi cần, để chuyển đảm bảo thứ tự cho tầng trên
- Bên gửi: khi nhận được ACK cho từng gói sẽ dịch cửa sổ sang phải, sau đó truyền gói tiếp theo
- Bên gửi: không nhận được ACK
  - Đặt đồng hồ cho từng gói tin chưa được ack
  - Sau timeout của pkt n thì truyền lại pkt n

Selective repeat



- Tổ chức vùng đệm để sắp xếp các gói tin theo đúng thứ tự để truyền cho tầng trên

Selective repeat: Ví dụ



Chương 3: Tầng giao vận

- Tổng quan về tầng giao vận
- Dồn kênh và phân kênh
- Giao thức UDP
- Truyền dữ liệu tin cậy
- Giao thức TCP

Giao thức TCP

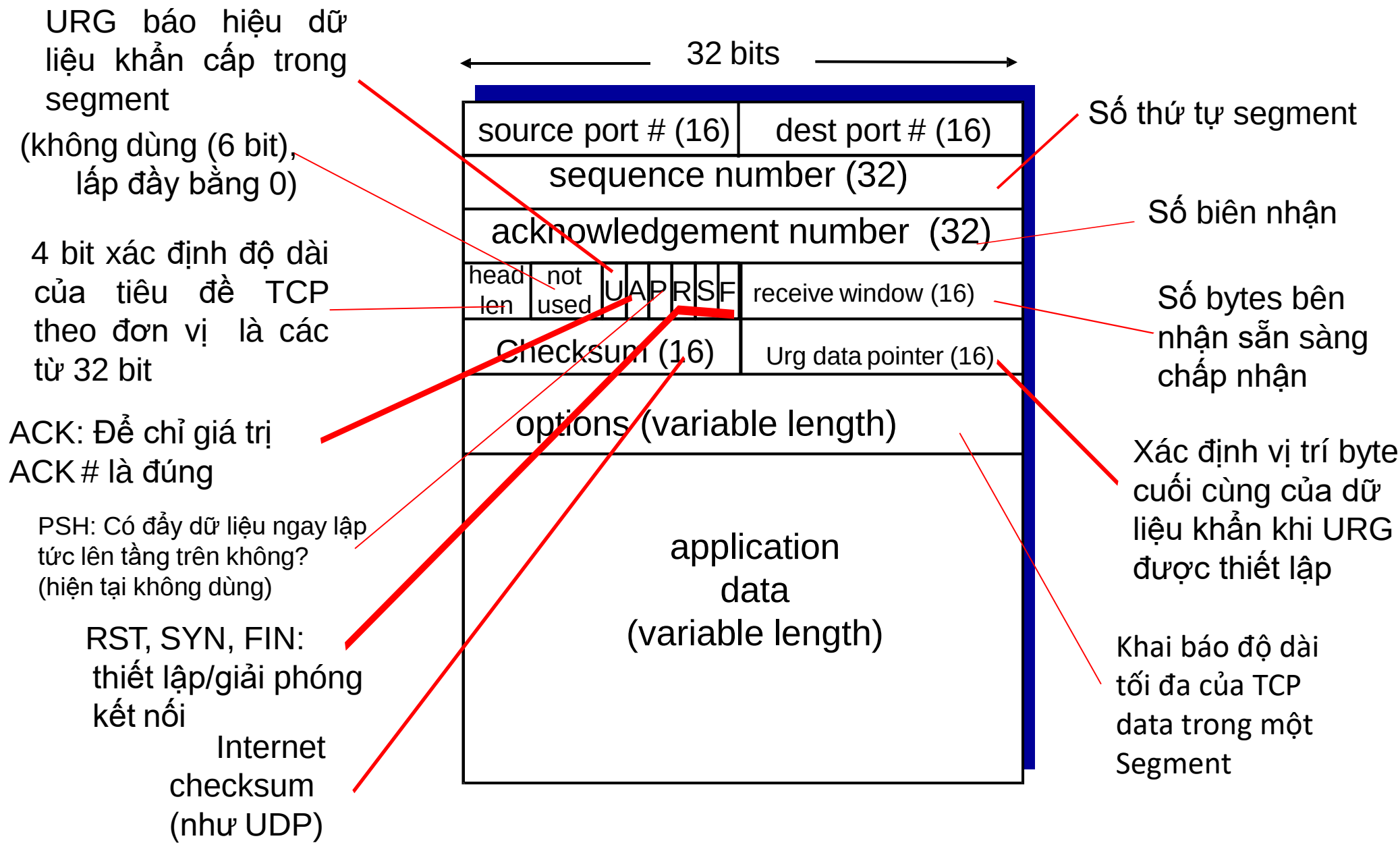
- Tổng quan về TCP
- Cấu trúc của TCP segment
- Truyền dữ liệu tin cậy của TCP
- Điều khiển luồng
- Điều khiển tắc nghẽn

Tổng quan về TCP

RFCs: 793,1122,1323, 2018, 2581

- hướng kết nối (connection-oriented):
  - Bắt tay ba bước (handshaking)
- Giao thức truyền dữ tin cậy, truyền theo chuỗi byte đảm bảo thứ tự:
  - Sử dụng vùng đệm
- Truyền theo kiểu liên tục (pipelined):
  - Tăng hiệu quả
- Điều khiển luồng:
  - bên gửi không làm quá tải bên nhận
- Điều tắc nghẽn:
  - Việc truyền dữ liệu không nên làm tắc nghẽn mạng

Cấu trúc của TCP segment



Sequence number, acknowledgement number

- Giả sử rằng một tiến trình trong máy A muốn gửi một luồng dữ liệu đến một tiến trình trong máy B qua kết nối TCP. TCP trong máy A sẽ đánh số thứ tự cho từng byte của luồng dữ liệu. Mỗi luồng chia thành nhiều segment, mỗi segment có số thứ tự là số thứ tự của byte đầu tiên của segment
- Giả sử luồng dữ liệu gồm 500.000 byte, mỗi segment là 1.000 byte. TCP xây dựng 500 segment dữ liệu.
  - Segment đầu tiên được gán số thứ tự 0
  - Segment thứ hai được gán số thứ tự 1.000
  - Segment thứ ba được gán số thứ tự 2.000, v.v.
  - Mỗi số thứ tự như vậy được chèn vào trường Sequence number trong tiêu đề của phân đoạn TCP thích hợp.

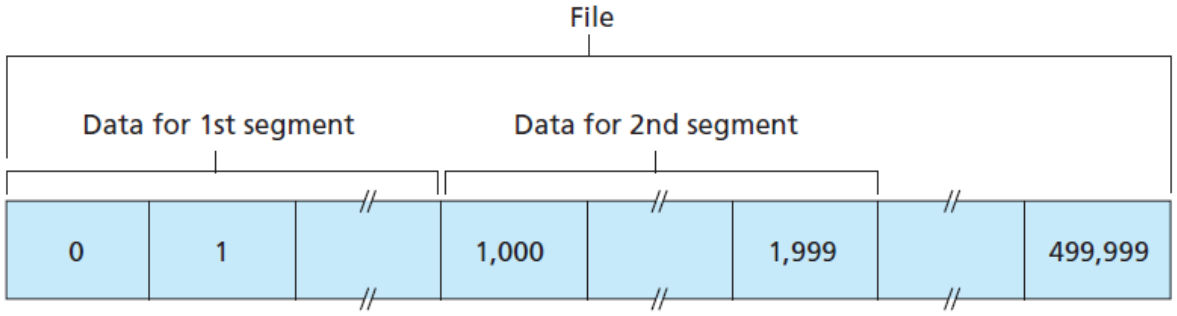


Figure 3.30 ♦ Dividing file data into TCP segments

Sequence number, acknowledgement number

- TCP là kênh truyền song công, nên bên A có thể nhận được dữ liệu từ B, trong khi A gửi dữ liệu cho B (trên cùng kết nối TCP)
- Acknowledgement number: là số thứ tự của byte tiếp theo mà máy A đang chờ máy B gửi tới
- Ví dụ 1:
  - A đã nhận được tất cả các byte được đánh số từ 0 đến 535 từ B và giả sử rằng A sắp gửi một đoạn đến B.
  - A đang chờ byte từ 536 và các byte tiếp theo trong luồng dữ liệu của B. Vì vậy, A đặt 536 vào trường Acknowledgement number của đoạn mà nó gửi đến B.
- Ví dụ 2:
  - A đã nhận được đoạn chứa byte 0 đến 535 và đoạn khác chứa byte 900 đến 1.000 từ B (A chưa nhận được byte 536 đến 899) Khi đó, A vẫn đang chờ đoạn từ byte 536 để tạo lại luồng dữ liệu. Do đó, phân đoạn tiếp theo của A tới B sẽ chứa 536 trong trường Acknowledgement number (TCP được cho là cung cấp các xác nhận tích lũy)
  - A đã nhận được đoạn thứ ba (byte 900 đến 1.000) trước khi nhận được đoạn thứ hai (byte 536 đến 899). Do đó, phân khúc thứ ba đã ra khỏi trật tự. Vấn đề là: Máy nhận sẽ làm gì khi nhận được các phân đoạn không theo thứ tự trong kết nối TCP?
- Bên gửi xử lý segment không đúng thứ tự như thế nào?
  - TCP không mô tả

Giao thức TCP

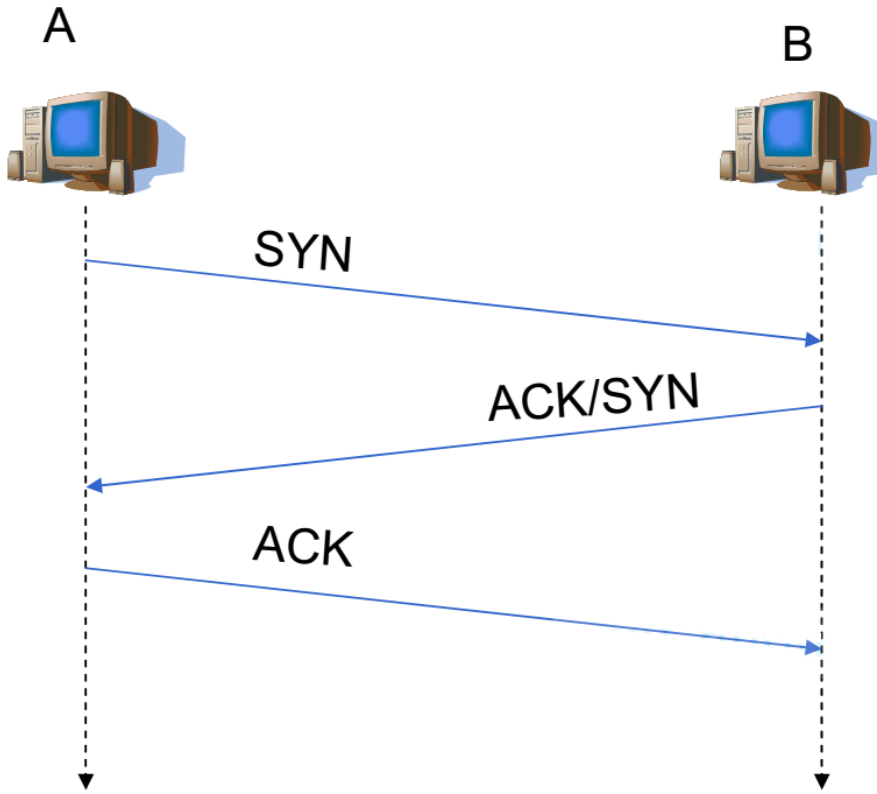
- Tổng quan về TCP
- Cấu trúc của TCP segment
- Truyền dữ liệu tin cậy của TCP
- Điều khiển luồng
- Điều khiển tắc nghẽn

Truyền dữ liệu tin cậy của TCP

- TCP kiểm soát dữ liệu đã được nhận chưa
  - Seq. #
  - ACK (Acknowledgement number)
- Chu trình làm việc của TCP: Bắt tay ba bước
  - Thiết lập kết nối
  - Truyền/nhận dữ liệu
  - Đóng kết nối

Thiết lập kết nối TCP: Giao thức bắt tay ba bước

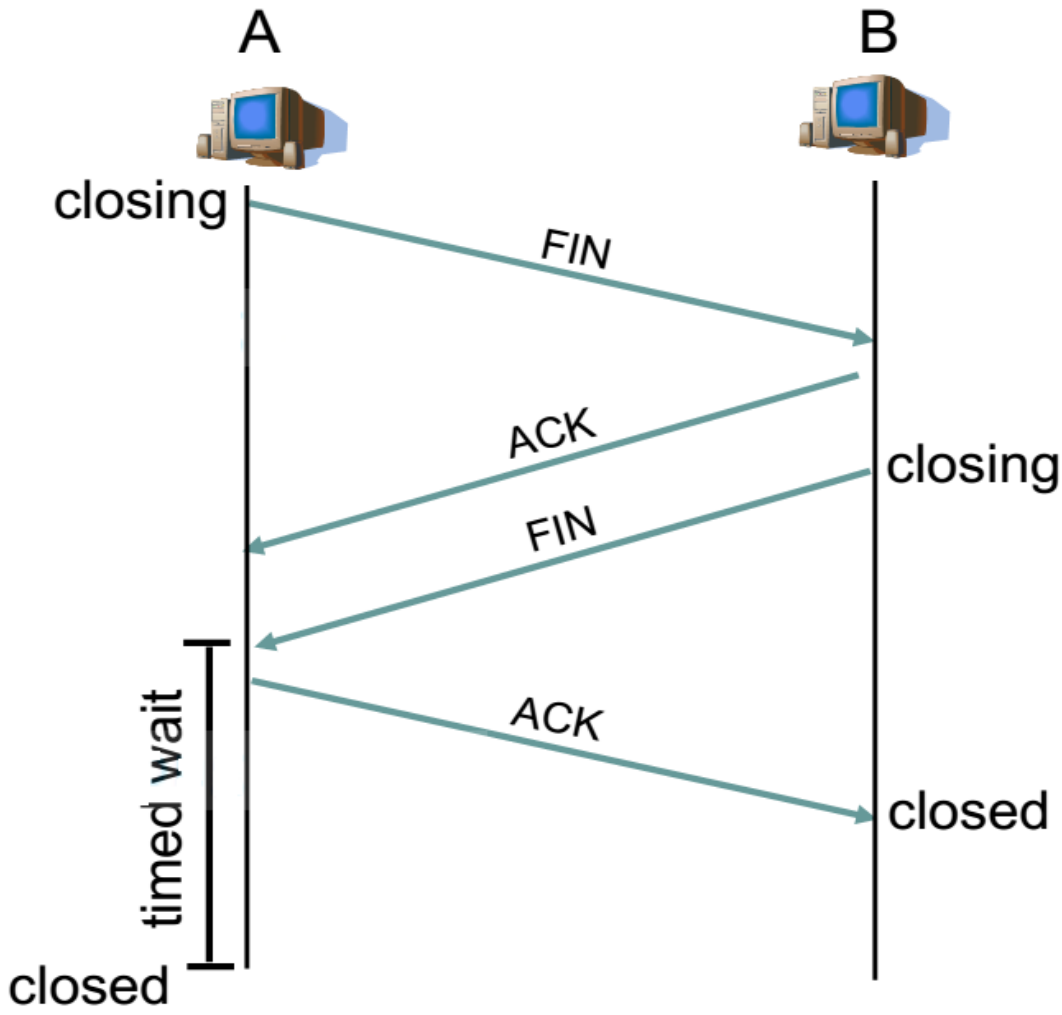
- Bước 1: A gửi SYN cho B
  - chỉ ra giá trị khởi tạo seq # của A
  - không có dữ liệu
- Bước 2: B nhận SYN, trả lời bằng SYNACK
  - B khởi tạo vùng đệm
  - chỉ ra giá trị khởi tạo seq. # của B
- Bước 3: A nhận SYNACK, trả lời ACK, có thể kèm theo dữ liệu



Ví dụ về đóng kết nối TCP

- Bước 1: A Gửi FIN cho B
- Bước 2: B nhận được FIN, trả lời ACK, đồng thời đóng liên kết 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



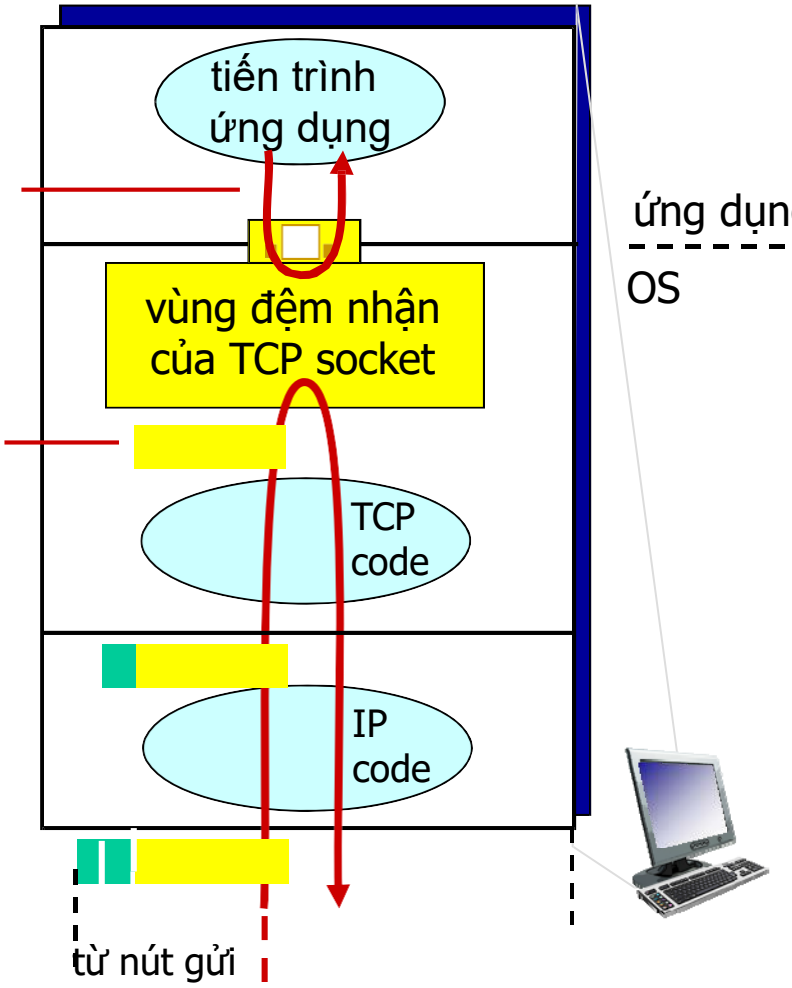
Giao thức TCP

- Tổng quan về TCP
- Cấu trúc của TCP segment
- Truyền dữ liệu tin cậy của TCP
- Điều khiển luồng
- Điều khiển tắc nghẽn

Điều khiển luồng của TCP

ứng dụng có thể lấy dữ liệu khỏi TCP socket buffers ....

... chậm hơn TCP bên nhận đang chuyển (sender đang gửi)



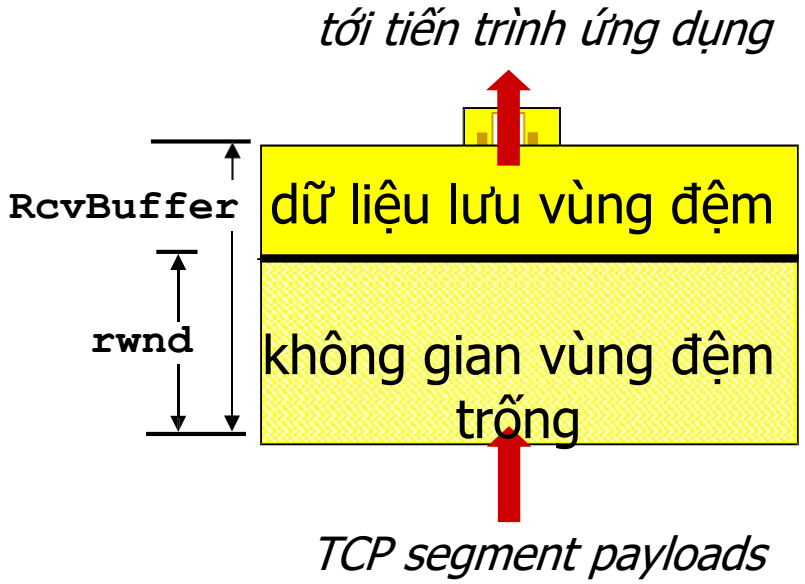
điều khiển luồng

nút nhận điều khiển nút gửi, để nút gửi không làm tràn vùng đệm của nút nhận bởi truyền quá nhiều và quá nhanh

ngăn xếp giao thức tại nút nhận

Điều khiển luồng của TCP

- Nút nhận thông báo không gian vùng đệm còn trống bằng cách đưa giá trị **`rwnd(receive window)`** trong TCP header của segment gửi từ nút nhận tới nút gửi
  - **`Kích thước của RcvBuffer`** được đặt thông qua tùy chọn của socket (thường mặc định 4096 byte)
  - nhiều hệ điều hành tự động điều chỉnh **`RcvBuffer`**
- Nút gửi giới hạn dữ liệu chưa được ack bằng giá trị **`rwnd`** của nút nhận
- Đảm bảo vùng đệm nhận không bị tràn



Bộ đệm phía nút nhận

Giao thức TCP

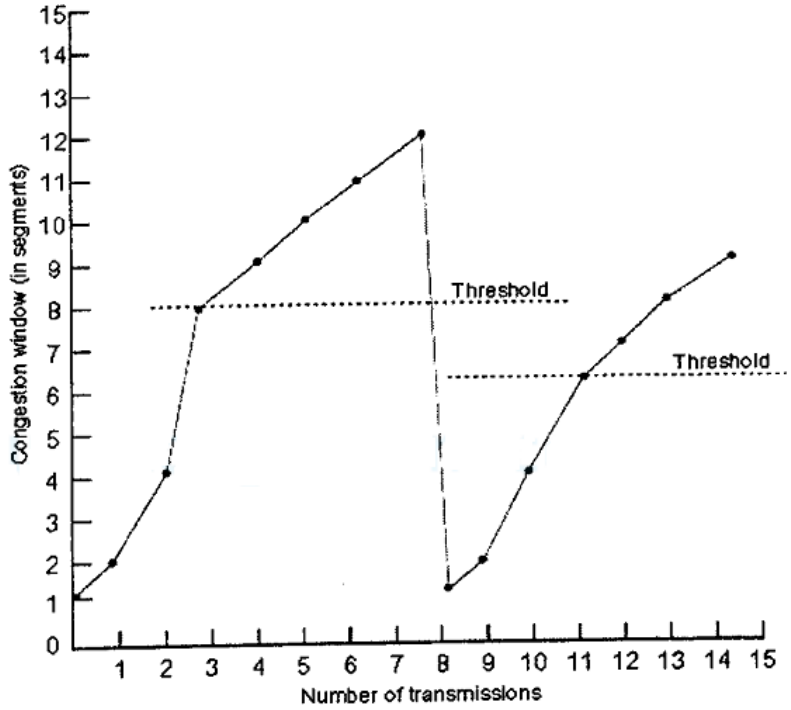
- Tổng quan về TCP
- Cấu trúc của TCP segment
- Truyền dữ liệu tin cậy của TCP
- Điều khiển luồng
- Điều khiển tắc nghẽn

Điều khiển tắc nghẽn

- *Khi nào tắc nghẽn* xảy ra?
  - Truyền quá nhiều làm cho mạng quá tải
- Hậu quả của việc nghẽn mạng:
  - mất gói tin (tràn vùng đệm tại router)
  - độ trễ lớn (đợi trong vùng đệm của router)
- Bổ sung thêm thông tin cho điều khiển tắc nghẽn
  - Congestion window (cwnd): số lượng dữ liệu tối đa mà bên gửi có thể gửi qua kết nối
  - Threshold (ngưỡng)

Nguyên tắc điều khiển tắc nghẽn

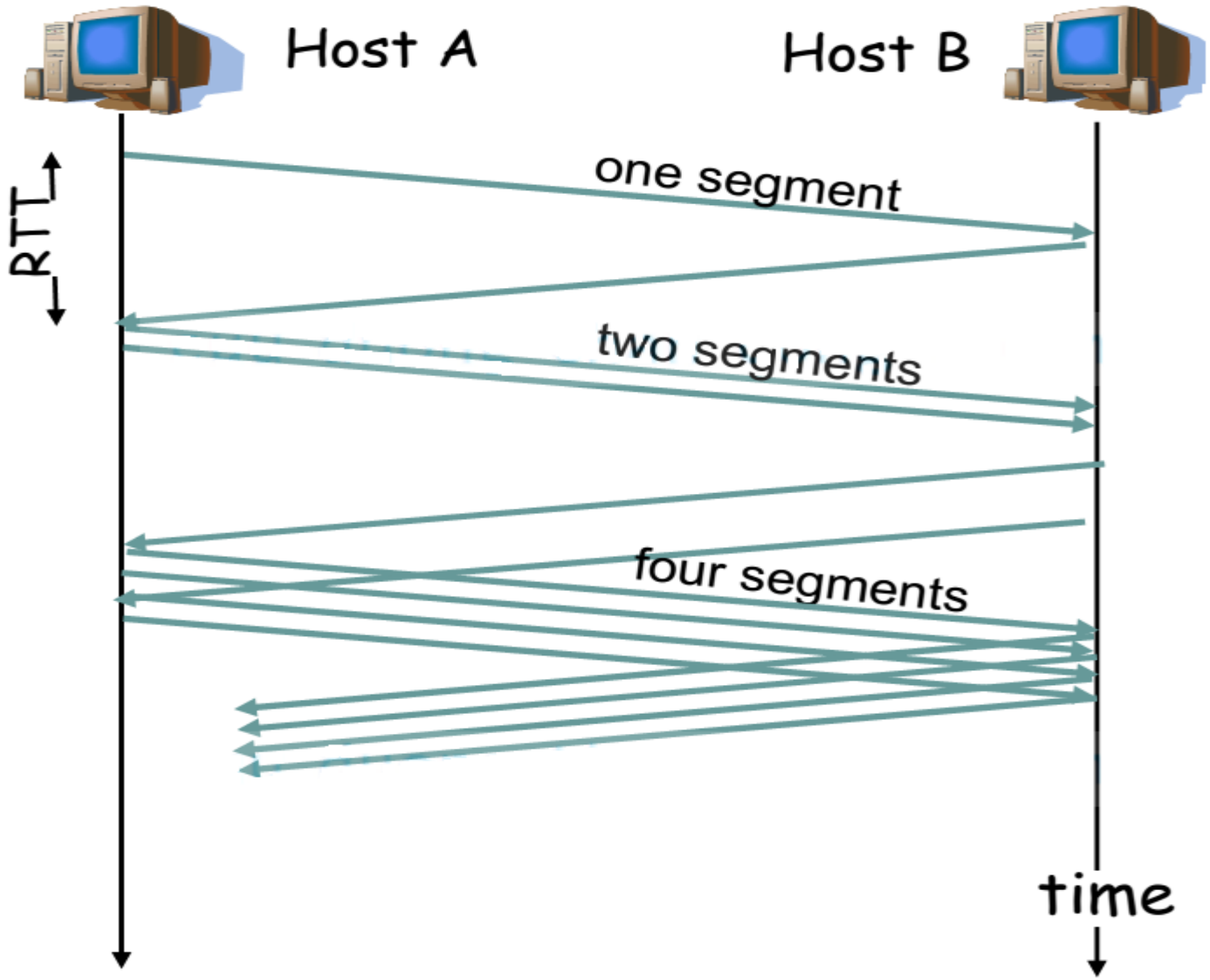
- Tăng dần tốc độ (slow-start)
  - Tăng tốc độ theo hàm số mũ
  - Tiếp tục tăng đến một ngưỡng nào đó
- Tránh tắc nghẽn
  - Tăng dần tốc độ theo hàm tuyến tính cho đến khi phát hiện tắc nghẽn
- Phát hiện tắc nghẽn
  - Nếu gói tin bị mất



TCP slow-start

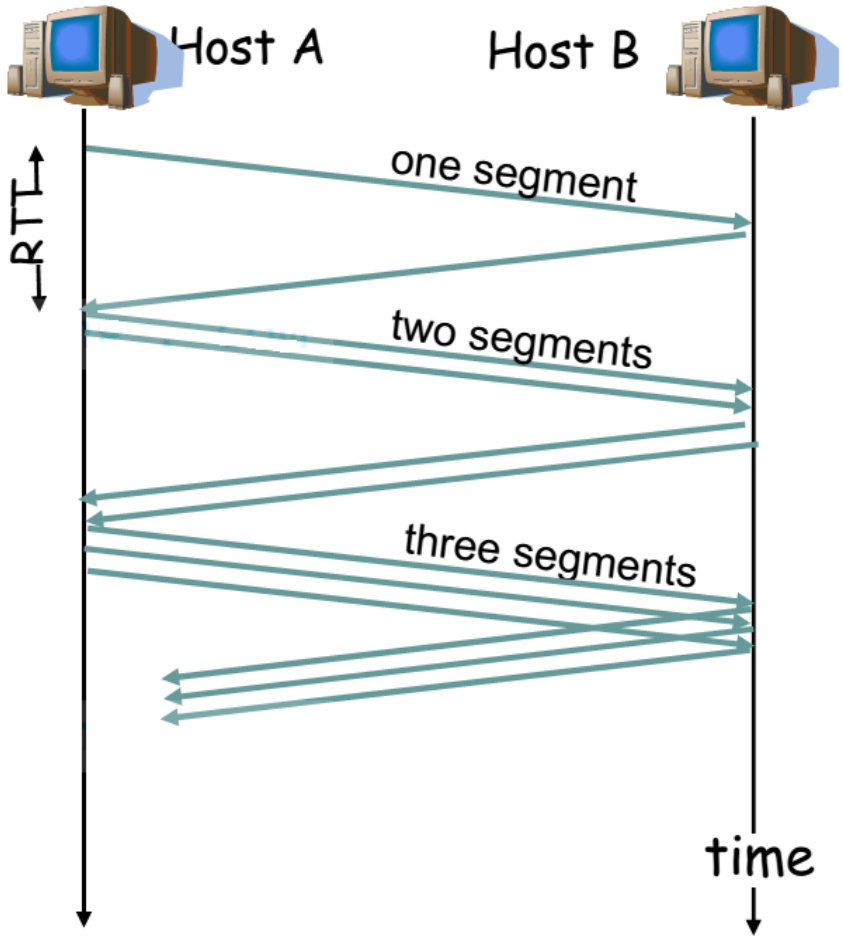
- Ý tưởng cơ bản
  - đặt cwnd bằng 1 MSS (Maximum segment size)
  - Tăng cwnd lên gấp đôi
    - ✓ Khi nhận được ACK
  - Bắt đầu chậm, nhưng tăng theo hàm mũ
- Tăng cho đến một ngưỡng: threshold
  - Sau đó, TCP chuyển sang trạng thái tránh tắc nghẽn

TCP slow-start: minh họa



TCP tránh tắc nghẽn

- Ý tưởng cơ bản
  - Giảm tốc độ gửi
  - đặt cwnd theo cấp số cộng sau khi nó đạt tới threshold
  - Khi bên gửi nhận được ACK
    - ✓ Tăng cwnd thêm 1 MSS (Maximum segment size)



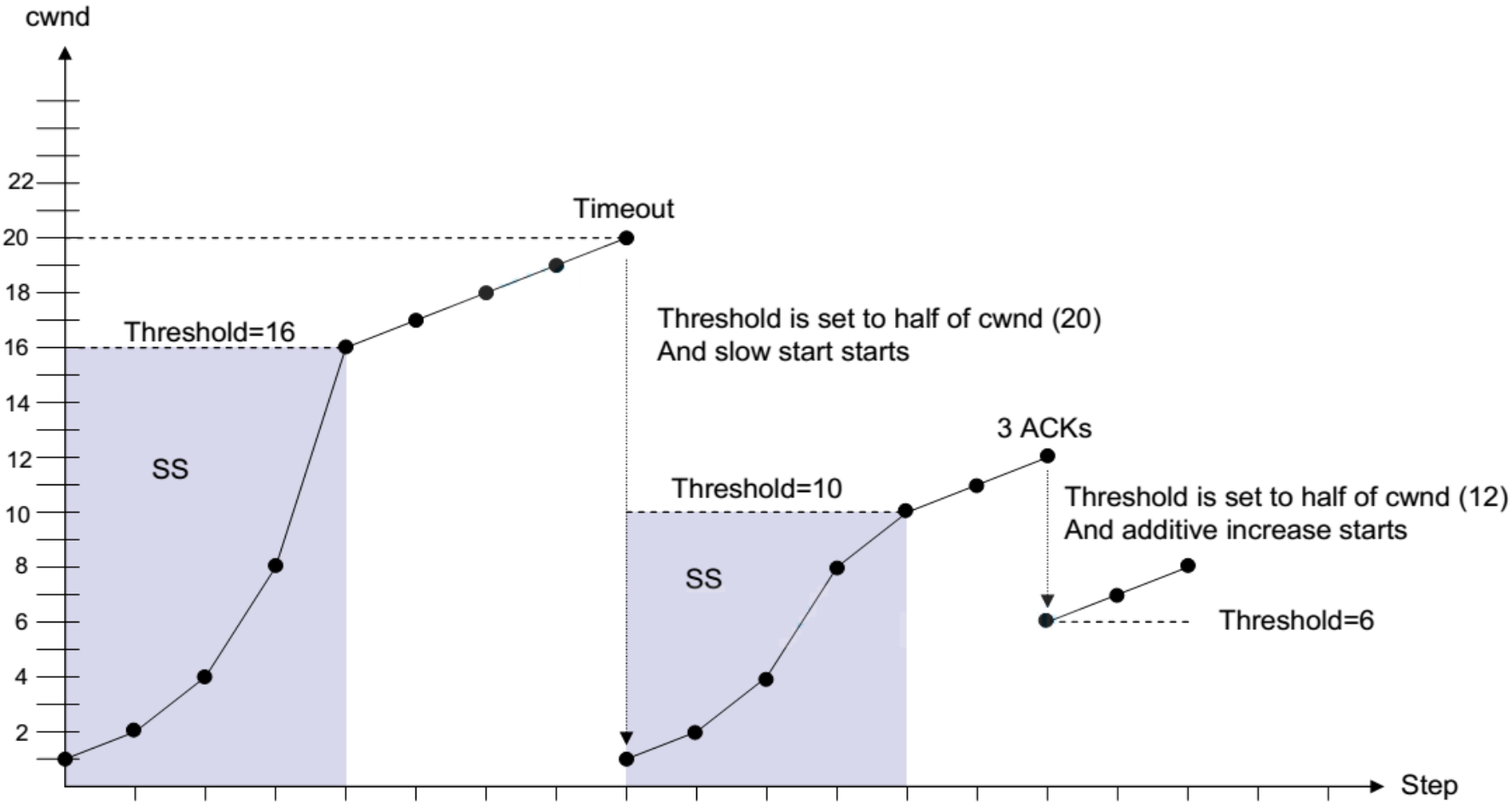
TCP phát hiện tắc nghẽn

- 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 số hiệu gói tin trong ACK

TCP phát hiện tắc nghẽn (tiếp)

- Khi có timeout của bên gửi
  - TCP đặt ngưỡng 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 đặt cwnd về giá trị của ngưỡng hiện tại
  - TCP chuyển trạng thái “tránh tắc nghẽn”

Kiểm soát tắc nghẽn: minh họa



Chương 3: Tóm tắt

- Có hai dạng giao thức giao vận trong Internet
  - UDP: truyền dữ liệu không tin cậy
  - TCP: truyền dữ liệu tin cậy
- Các cơ chế bên trong các dịch vụ của tầng giao vận:
  - Dồn kênh, phân kênh
  - Truyền dữ liệu tin cậy
    - Báo nhận
    - Truyền lại
    - điều khiển luồng
    - điều khiển tắc nghẽn

Tiếp:

- chuyển từ phần biên mạng (network edge) (tầng giao vận, tầng ứng dụng)
- vào phần lõi mạng (network core)

Mạng máy tính

- Hình ảnh và nội dung trong bài giảng này có tham khảo từ sách và bài giảng của TS. J.F. Kurose and GS. K.W. Ross