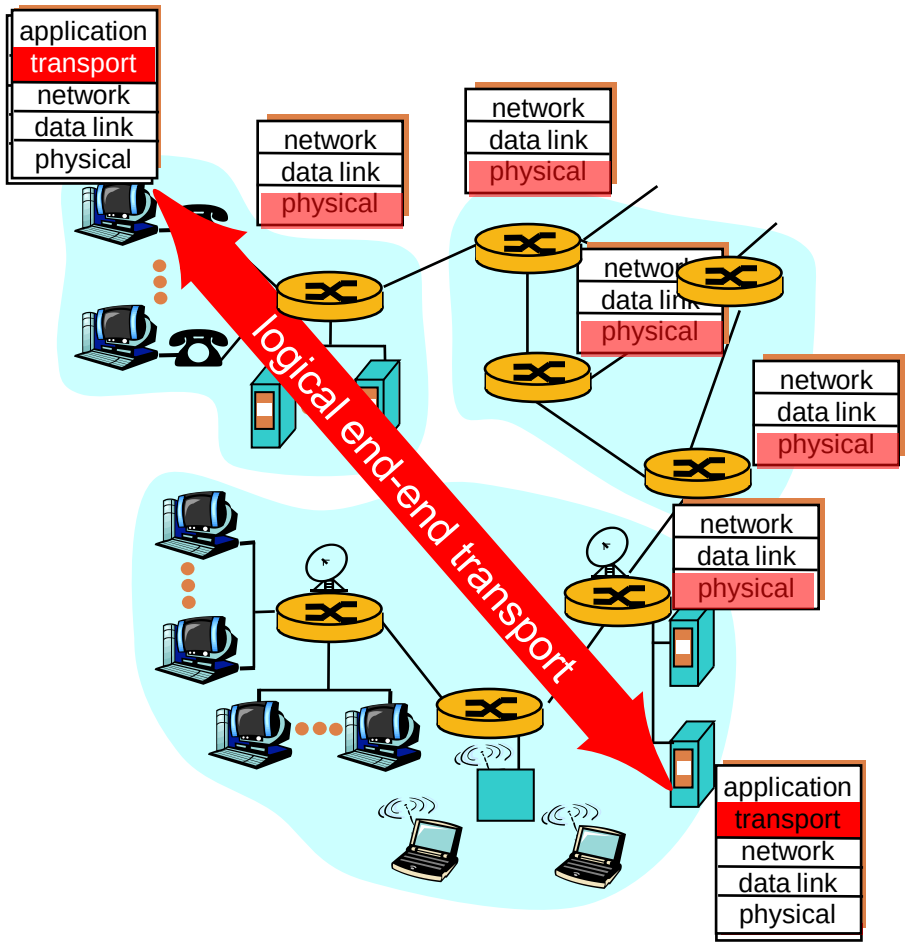
Tầng Vận chuyển

CHỨC NĂNG - 1

- Cung cấp kênh truyền dữ liệu ở mức logic giữa 2 tiến trình trên 2 máy

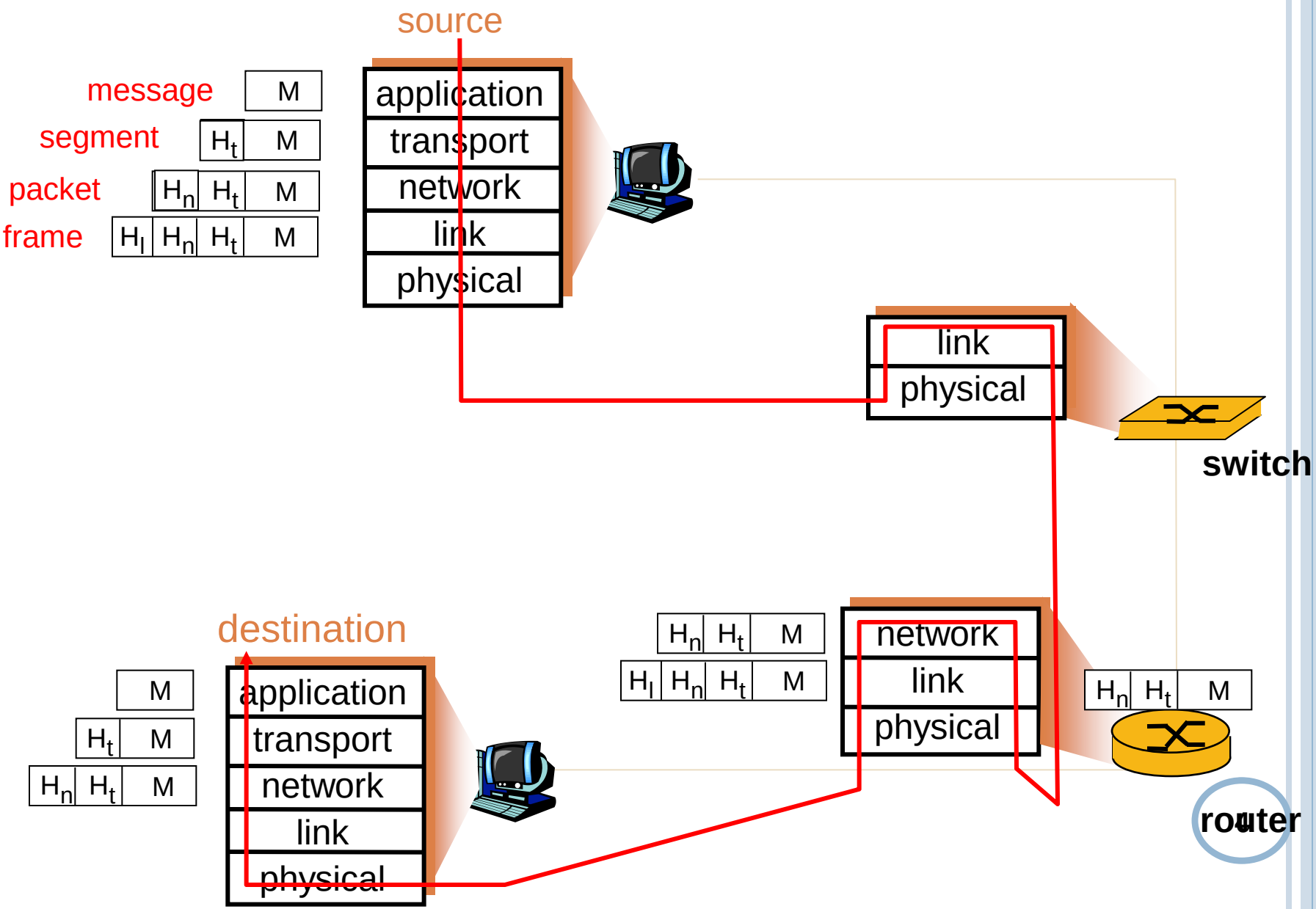


Application
Presentation
Session
Transport
Network
Data link
Physical

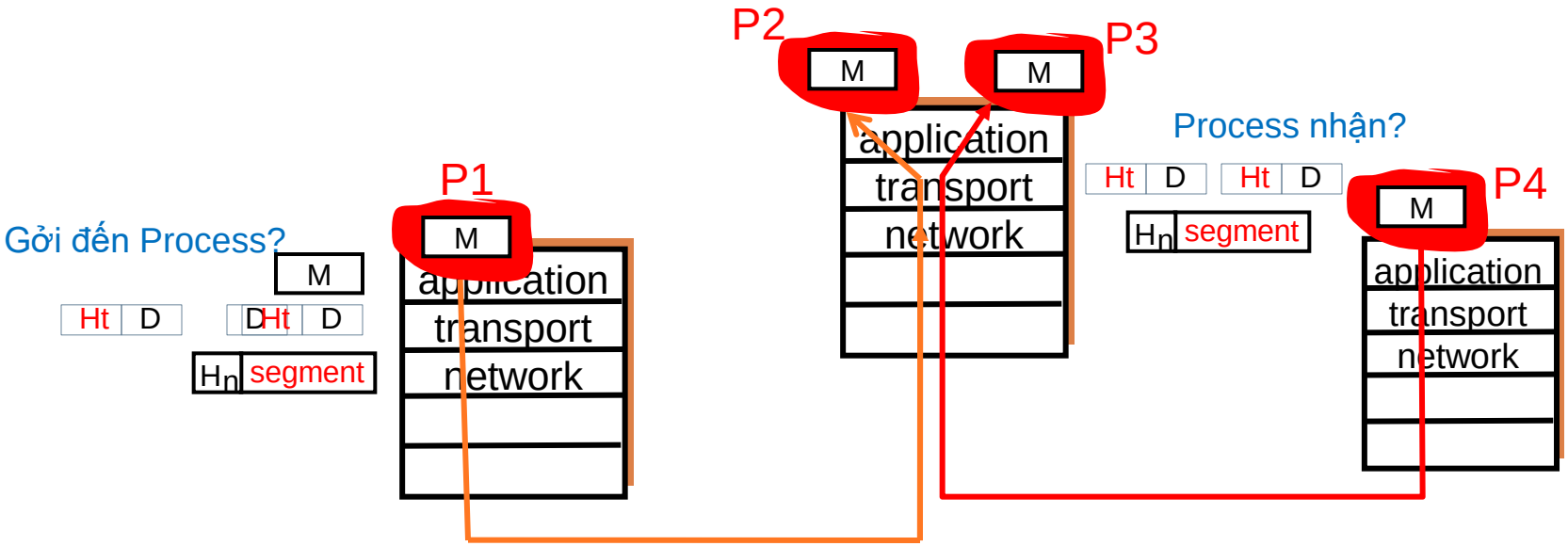
Nội dung

- Giới thiệu
- Nguyên tắc truyền dữ liệu đáng tin cậy
- Giao thức TCP
- Giao thức UDP

NHẮC LẠI


source
message
M
segment
H_t M
packet
H_n H_t M
frame
H_l H_n H_t M
application
transport
network
link
physical
link
physical
switch
destination
M
H_t M
H_n H_t M
application
transport
network
link
physical
H_n H_t M
H_l H_n H_t M
network
link
physical
H_n H_t M
router

TẦNG VẬN CHUYỂN - 1



TẦNG VẬN CHUYỂN - 2

- Thực thi ở end-system
- Bên gởi: thực hiện Dồn kênh
  - Nhận dữ liệu từ tầng ứng dụng (từ các socket)
  - Phân đoạn thông điệp ở tầng ứng dụng thành các **segment**
  - Dán nhãn dữ liệu: đóng gói theo giao thức tại tầng Transport
  - Chuyển các segment xuống tầng mạng (network layer)
- Bên nhận: thực hiện Phân kênh
  - Nhận các segment từ tầng mạng
  - Phân rã các segment thành thông điệp tầng ứng dụng
  - Chuyển thông điệp lên tầng ứng dụng (đến socket tương ứng)

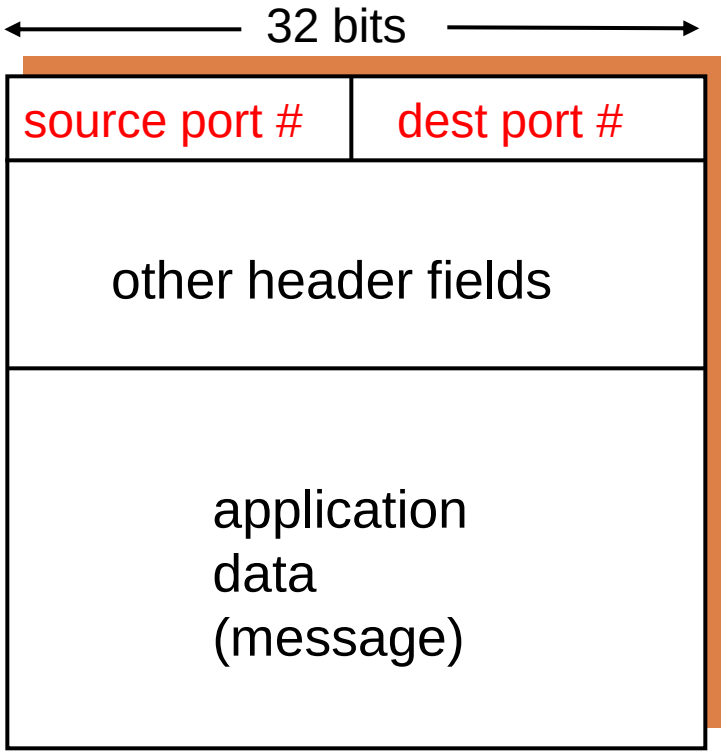
TẦNG VẬN CHUYỂN - 3

- Hỗ trợ
  - Truyền dữ liệu đáng tin cậy
    - Điều khiển luồng
    - Điều khiển tắt nghẽn
    - Thiết lập và duy trì kết nối
  - Truyền dữ liệu không đáng tin cậy
    - Nổ lực gởi dữ liệu hiệu quả nhất
- Không hỗ trợ
  - Đảm bảo thời gian trễ
  - Đảm bảo băng thông

Dồn kênh – Phân kênh - 1

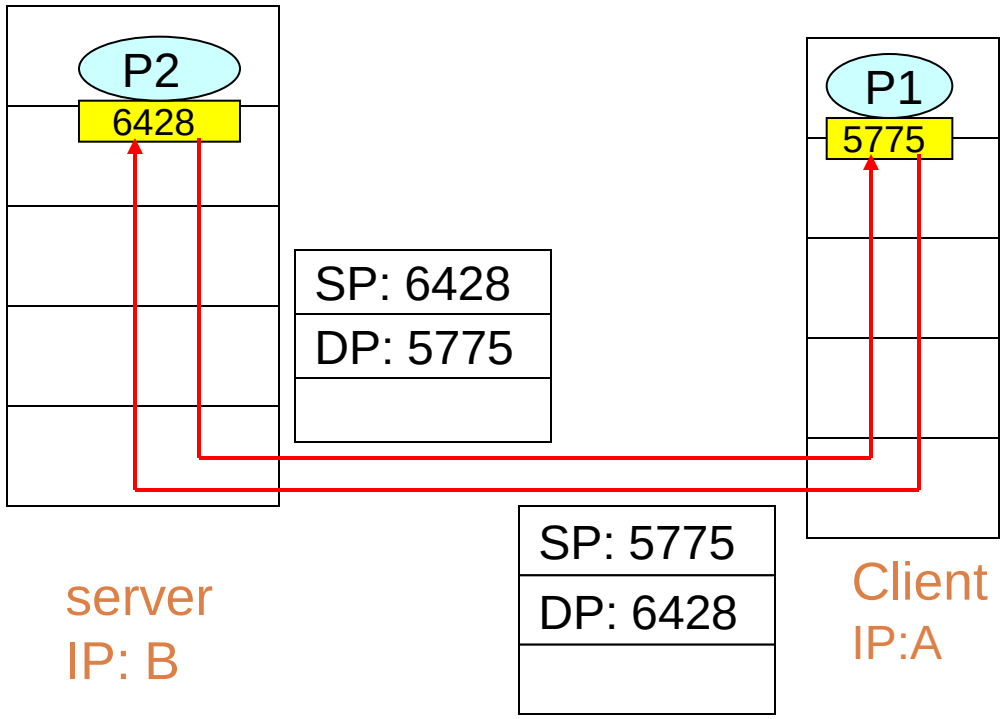
- Dồn kênh (Multiplexing):
  - Thực hiện tại bên gởi
  - Thu thập dữ liệu từ các socket
  - dán nhãn dữ liệu với 1 header
- Phân kênh (Demultiplexing):
  - Thực hiện tại bên nhận
  - phân phối các segment nhận được cho socket tương ứng
- Khi đóng gói dữ liệu ở tầng transport, header sẽ thêm vào:
  - Source port
  - Destination port

DỒN KÊNH – PHÂN KÊNH - 2



Cấu trúc của một segment

DỒN KÊNH – PHÂN KÊNH - 3



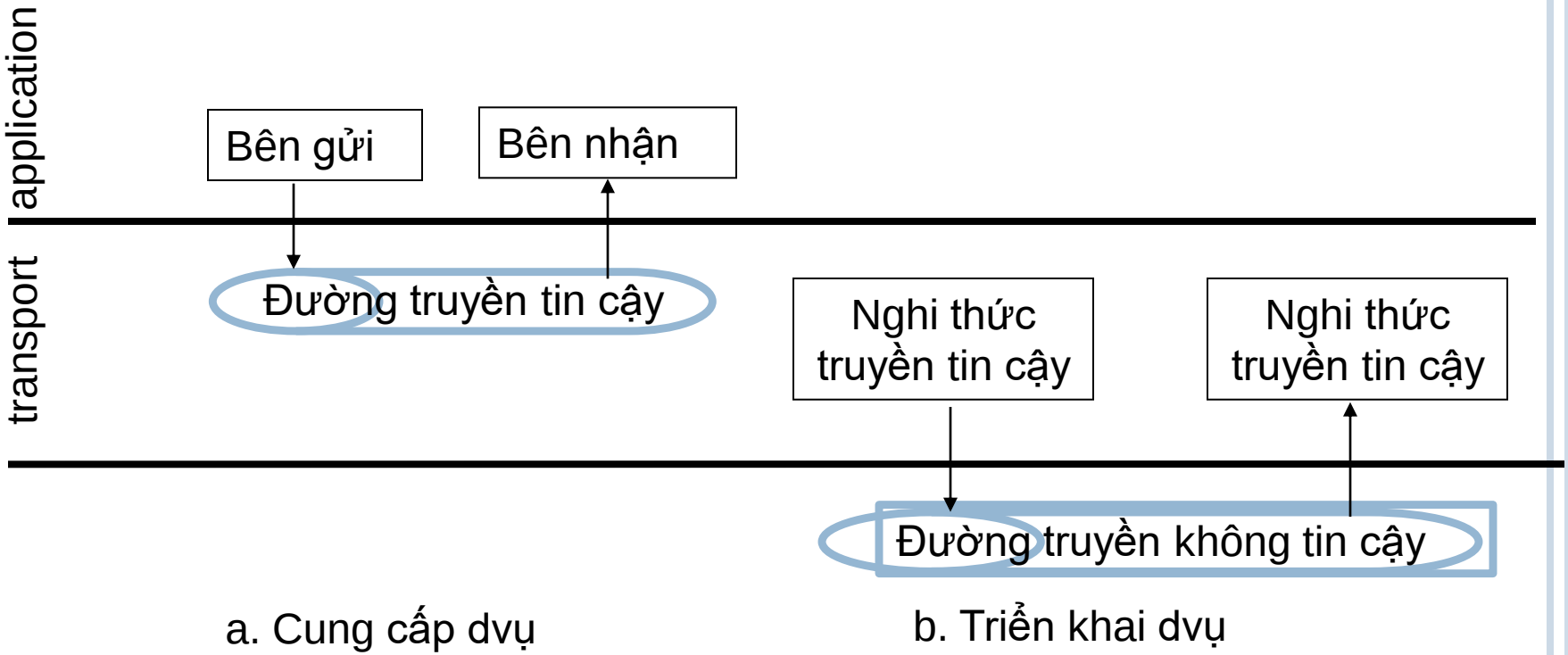
NỘI DUNG

- Giới thiệu
- Nguyên tắc truyền dữ liệu đáng tin cậy
- Giao thức TCP
- Giao thức UDP

Bài toán

Làm sao để truyền đáng tin cậy???

NGUYÊN LÝ TRUYỀN DỮ LIỆU ĐÁNG TIN CẬY



Đặc tính của đường truyền không tin cậy quyết định độ phức tạp của nghi thức truyền tin cậy

Nội dung

- Nghi thức truyền dữ liệu đáng tin cậy
  - RDT 1.0
  - RDT 2.0, RDT 2.1, RDT 2.2
  - RDT 3.0
- Pipeline
  - Go-back-N
  - Gởi lại có chọn

GIẢI QUYẾT LỖI BIT

- Bên gởi
  - Gởi kèm theo thông tin kiểm tra lỗi
  - Sử dụng các phương pháp kiểm tra lỗi
    - Checksum, parity checkbit, CRC,..
- Bên nhận
  - Kiểm tra có xảy ra lỗi bit?
  - Hành động khi xảy ra lỗi bit?
    - Báo về bên gởi

GIẢI QUYẾT MẤT GÓI

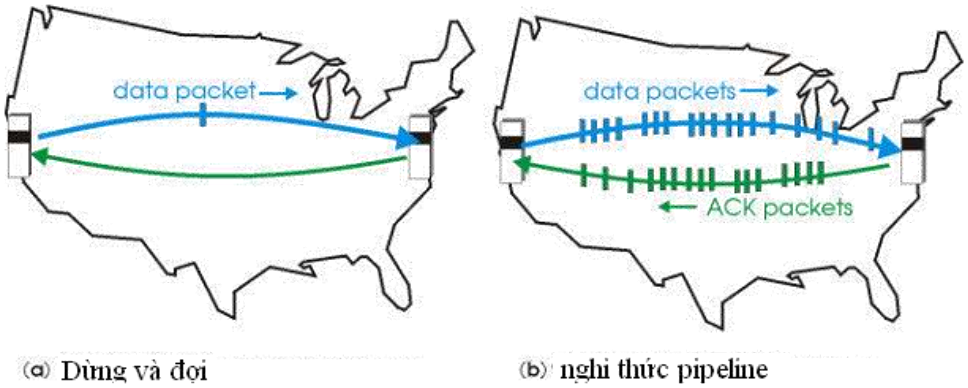
- Bên nhận
  - Gởi tín hiệu báo
    - Gởi gói tin báo hiệu ACK, NAK
- Bên gởi
  - Định nghĩa trường hợp mất gói
  - Chờ nhận tín hiệu báo
  - Hành động khi phát hiện mất gói

GIAO THỨC RDT

- RDT = Reliable Data Transfer
- Nguyên tắc: dừng và chờ
  - Bên gởi
    - Gởi gói tin kèm theo thông tin kiểm tra lỗi
    - **Dừng và chờ** đến khi nào gói tin vừa gởi đến được bên nhận ***an toàn:*** nhận được gói tin ACK
    - Gởi lại khi có lỗi xảy ra: lỗi bit, mất gói
  - Bên nhận:
    - Kiểm tra lỗi, trùng lắp dữ liệu
    - Gởi gói tin phản hồi
- Phiên bản:
  - RDT 1.0
  - RDT 2.0
  - RDT 2.1
  - RDT 2.2
  - RDT 3.0

NGUYÊN LÝ PIPE LINE

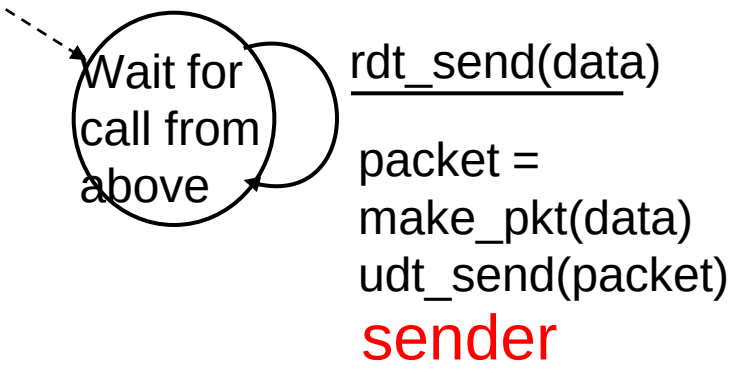
- Cho phép gởi nhiều gói tin khi chưa nhận ACK

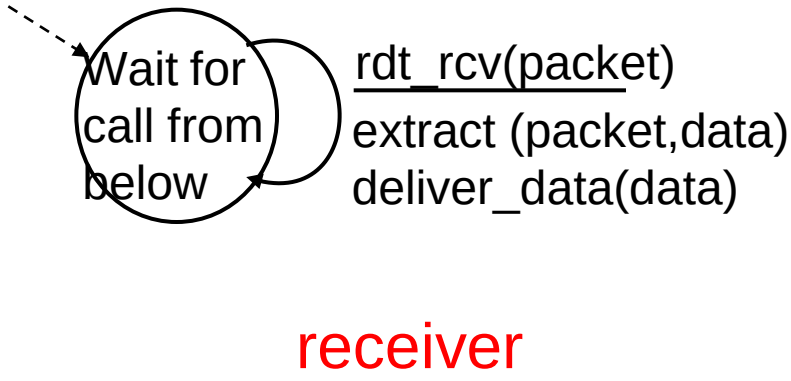


- Sử dụng buffer để lưu các gói tin
  - Bên gởi: lưu gói tin đã gởi nhưng chưa ack
  - Bên nhận: lưu gói tin đã nhận đúng nhưng chưa đúng thứ tự
- Giải quyết mất gói
  - Go back N
  - Selective Repeat (gởi lại có chọn)

RDT1.0 : ĐƯỜNG TRUYỀN LÝ TƯỞNG

- Giả thiết: kênh truyền bên dưới tuyệt đối
  - Không lỗi bit
  - Không mất gói tin
- FSM (finite state machine) cho bên gửi và nhận
  - Bên gửi chuyển dữ liệu xuống kênh bên dưới
  - Bên nhận đọc dữ liệu từ kênh truyền bên dưới

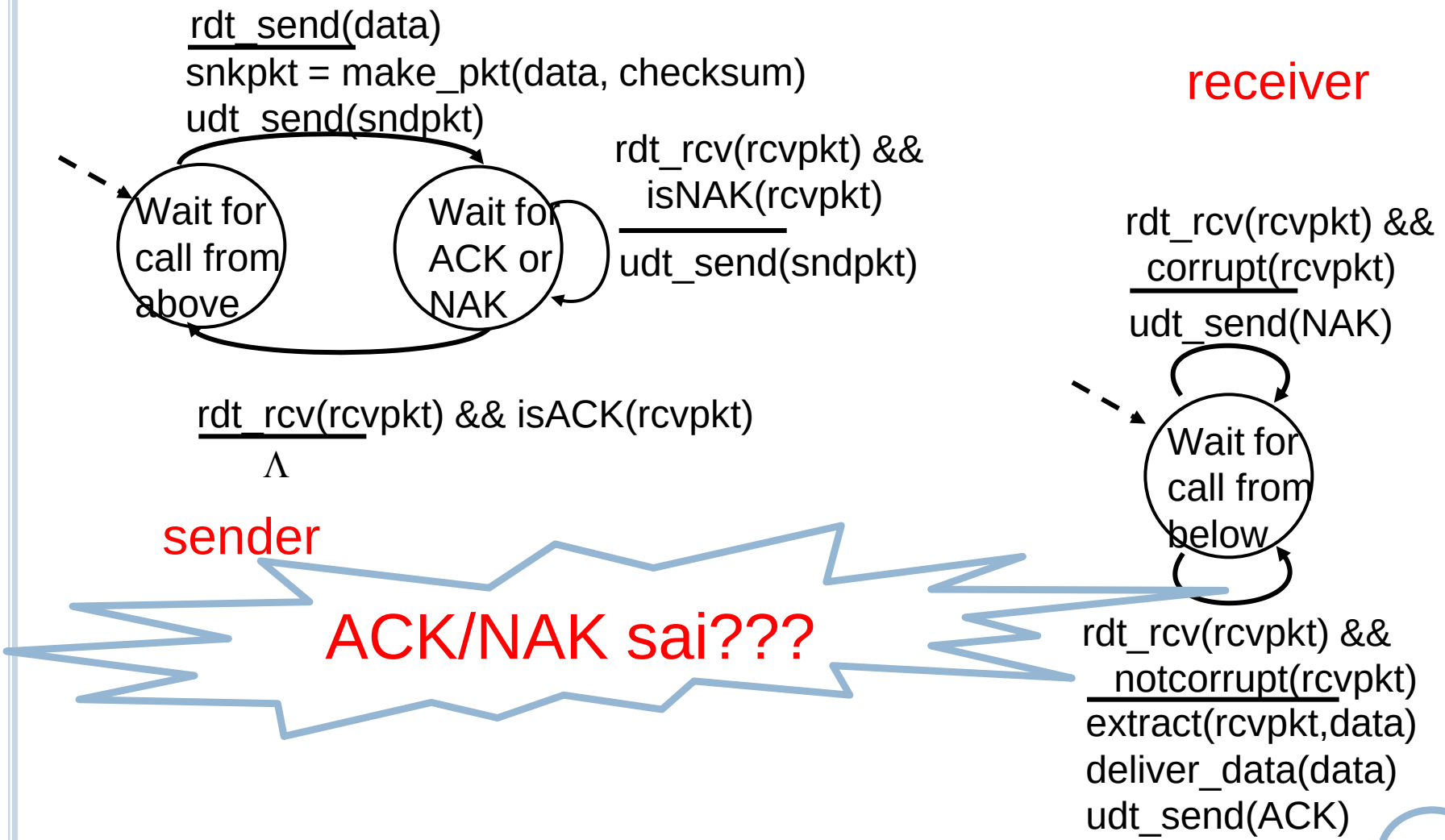




RDT2.0 KÊNH TRUYỀN CÓ LỖI BIT - 1

- Giả thiết: kênh truyền có thể xảy ra lỗi bit
  - Sử dụng các cơ chế kiểm tra lỗi
    - checksum
- Làm sao để khắc phục khi nhận ra lỗi?
  - Acknowledgement(ACKs): bên nhận báo cho bên gửi đã nhận được dữ liệu
  - Nagetive acknowledgement(NAKs): bên nhận báo gói tin bị lỗi
  - Bên gửi sẽ gửi lại gói tin khi nhận NAK
- So với rdt1.0, rdt2.0:
  - Nhận dạng lỗi
  - Cơ chế phản hồi: ACK, NAK

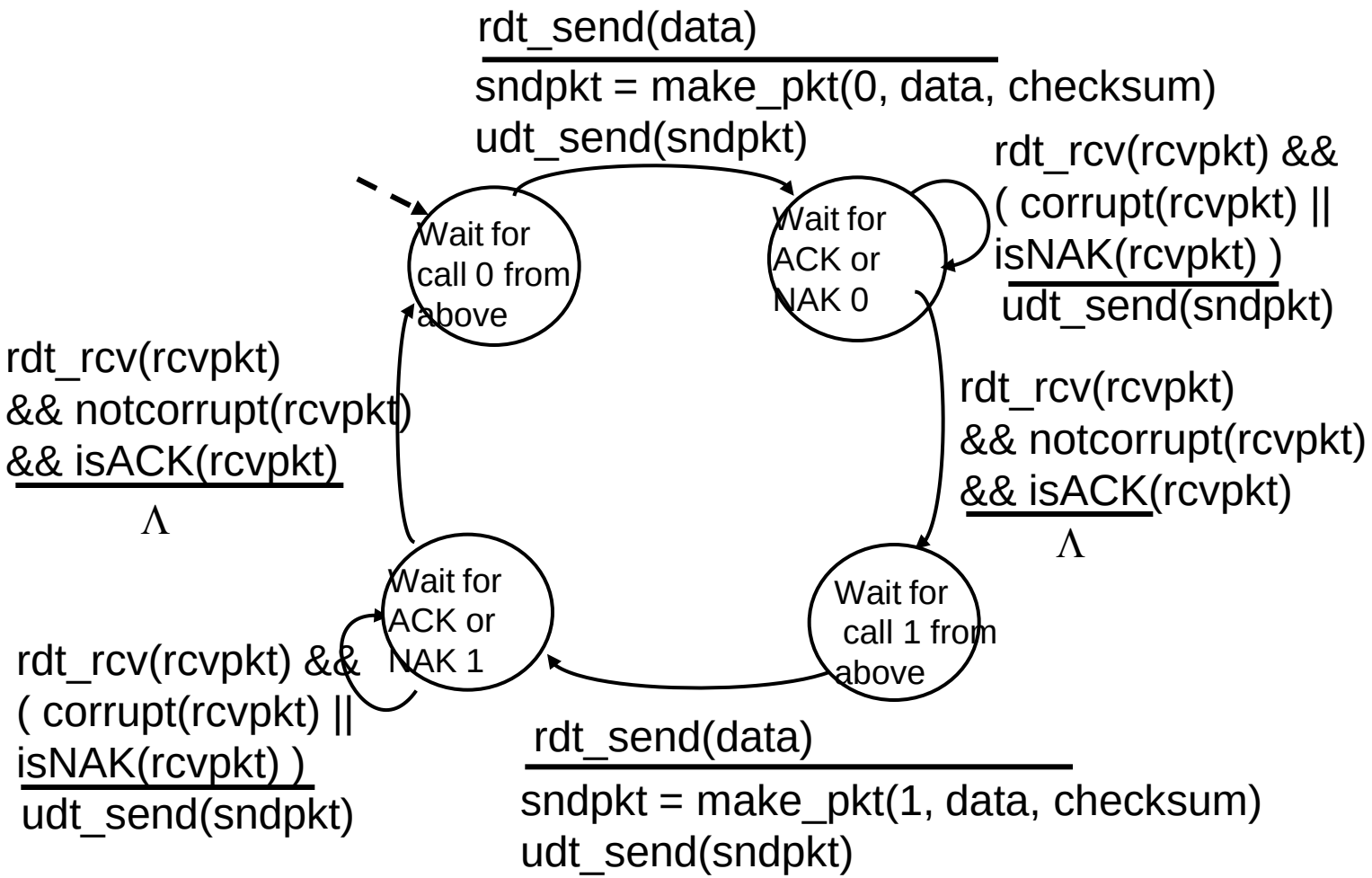
RDT2.0 FSM - 2


rdt_send(data)
snkpkt = make_pkt(data, checksum)
udt_send(sndpkt)
Wait for call from above
Wait for ACK or NAK
rdt_rcv(rcvpkt) && isNAK(rcvpkt)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && isACK(rcvpkt)
Λ
sender
receiver
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
udt_send(NAK)
Wait for call from below
ACK/NAK sai???
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
udt_send(ACK)

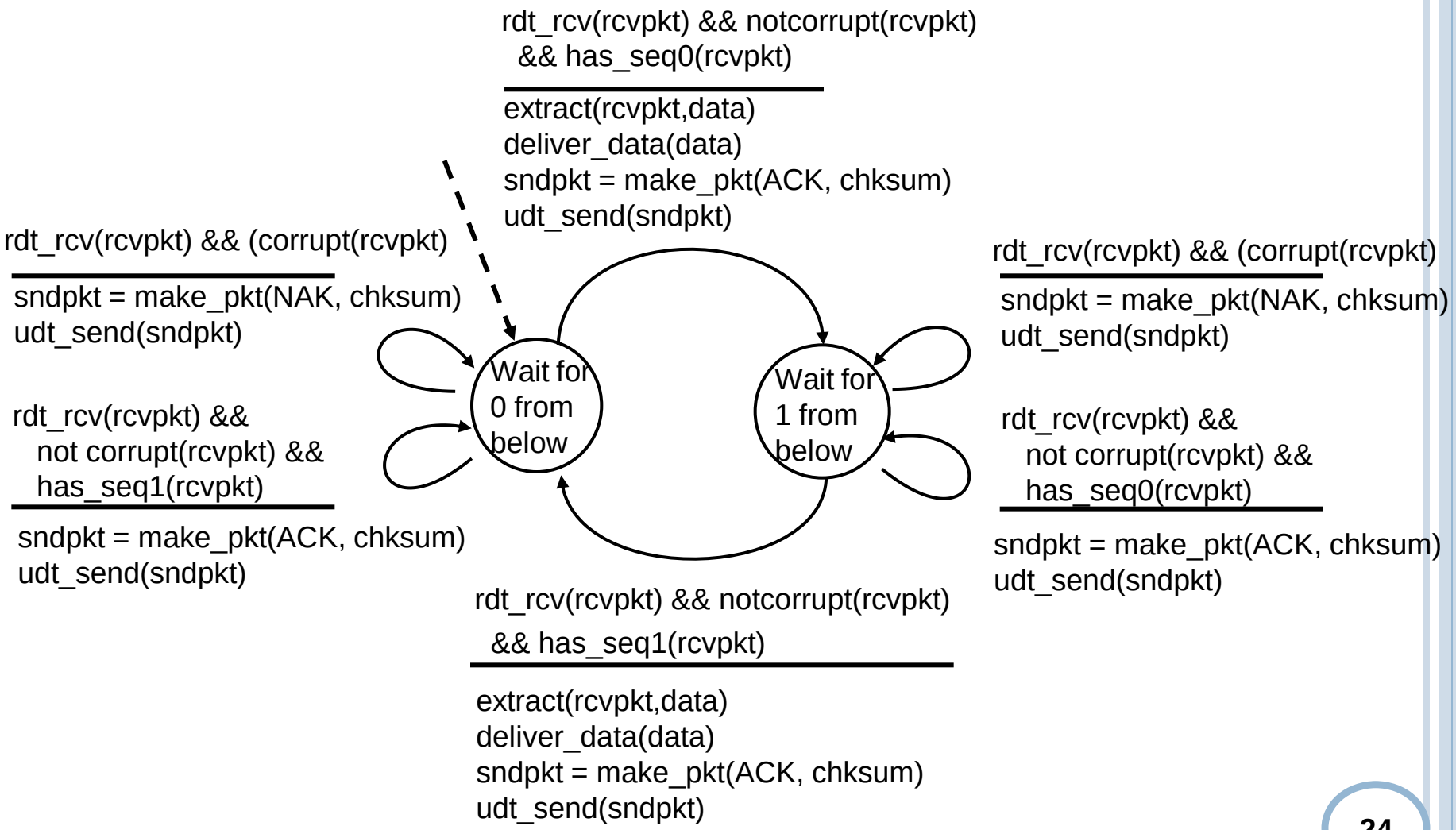
RDT2.0 - 3

- Giải quyết:
  - Bên gửi gửi lại gói tin khi nhận ACK/NAK sai
  - Bên gửi đánh số thứ tự cho mỗi gói tin
  - Bên nhận sẽ loại bỏ gói tin trùng.
- Dừng và đợi
  - Bên gửi gửi một gói tin và chờ phản hồi từ bên nhận

RDT2.1 BÊN GỬI XỬ LÍ LỖI ACK/NAK



RDT2.1 BÊN NHẬN XỬ LÍ LỖI ACK/NAK



RDT2.1 THẢO LUẬN

Bên gửi

- Thêm số thứ tự vào gói tin
  - 0 và 1???
- Phải kiểm tra: ACK/NAK sai không
- Phải nhớ gói tin hiện thời có thứ tự 0 hay 1

Bên nhận

- Phải kiểm tra nếu nhận trùng
  - So sánh trạng thái đang chờ (0 hay 1) với trạng thái gói tin nhận được
- Bên nhận không biết ACK/NAK cuối cùng có chuyển tới bên gửi an toàn không?

Cơ chế truyền đáng tin cậy - RDT

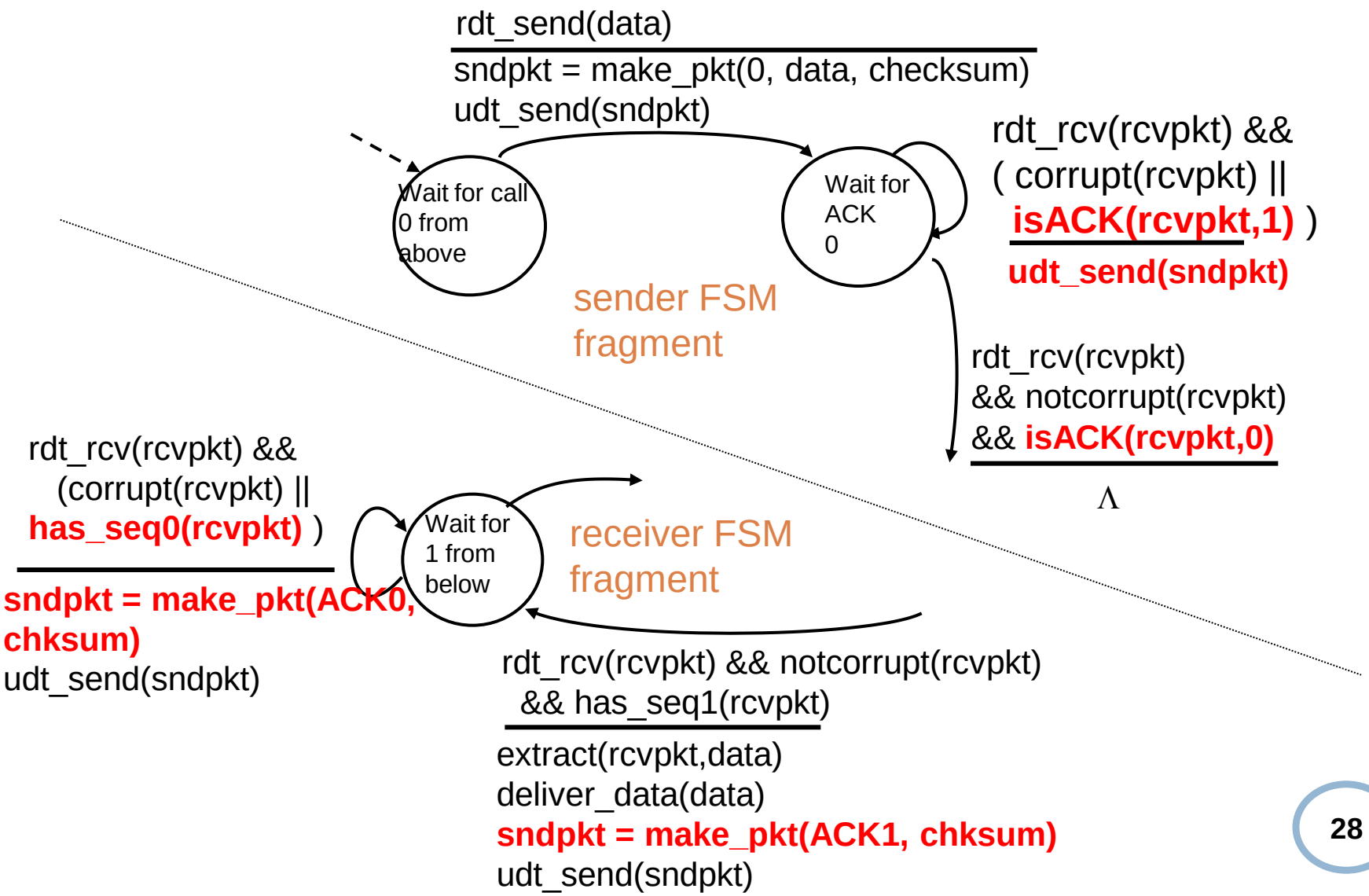
- Cơ chế:
  - Checksum: kiểm tra có lỗi xảy ra không?
  - ACK: bên nhận nhận đúng gói tin
  - NAK: bên nhận nhận sai gói tin
  - Sequence Number (1 bit = 0 hoặc 1)

RDT2.2 KHÔNG SỬ DỤNG NAK

- Hoạt động giống rdt2.1, nhưng không dùng NAK
- Bên nhận gửi ACK cho gói tin không lỗi nhận được cuối cùng.
  - Bên nhận phải **thêm số thứ tự vào gói tin ACK**
- Bên gửi nhận trùng gói tin ACK xem như gói tin NAK

➔ gửi lại gói vừa gởi vì gói này chưa nhận được ACK

RDT2.2: BÊN GỬI VÀ BÊN NHẬN



RDT3.0 KÊNH TRUYỀN CÓ LỖI VÀ MẤT - 1

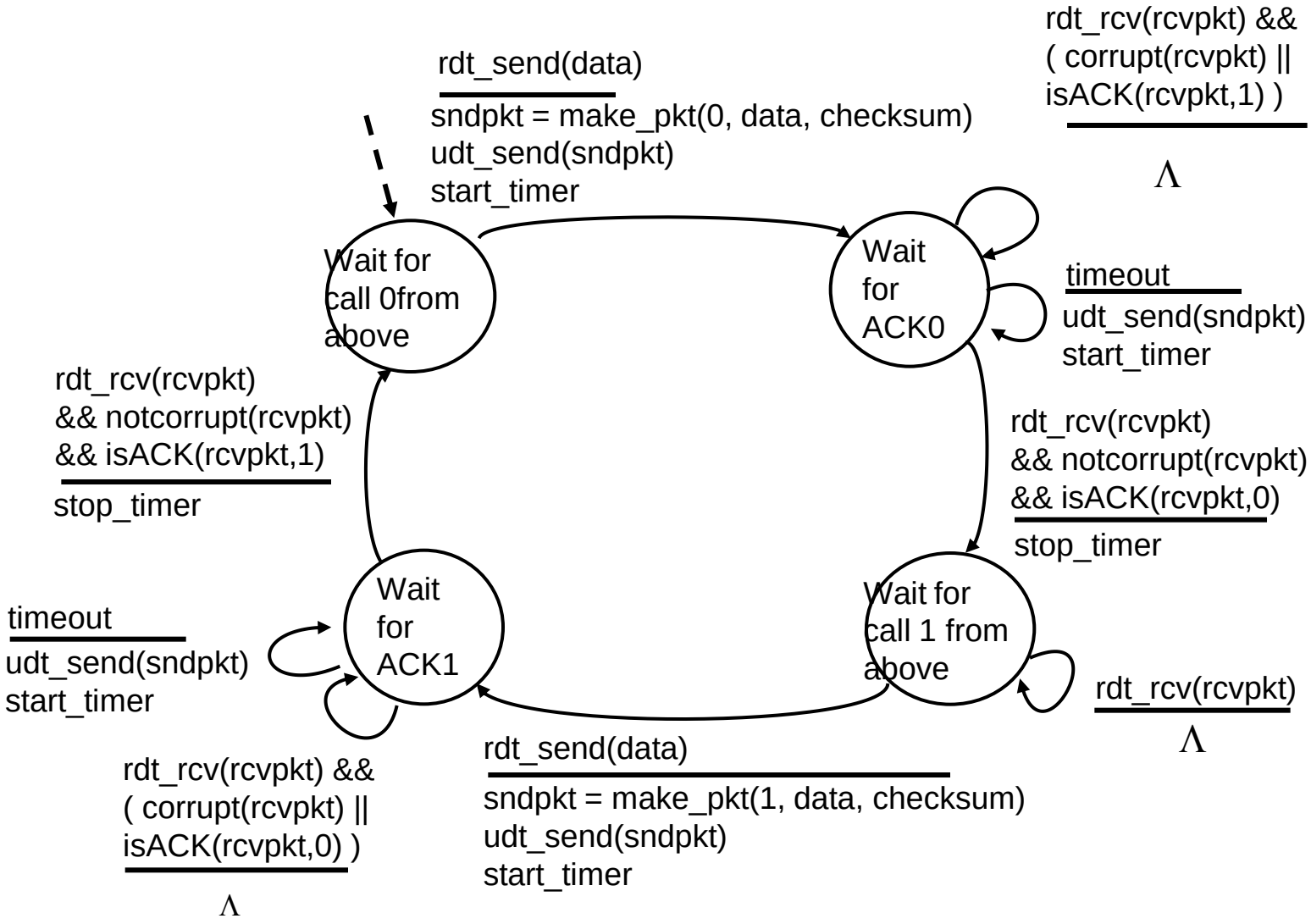
- Giả thiết:
  - Lỗi bit
  - mất gói

  ➔ Checksum, số thứ tự, ACKs, truyền lại vẫn chưa đủ
- Xử lý?

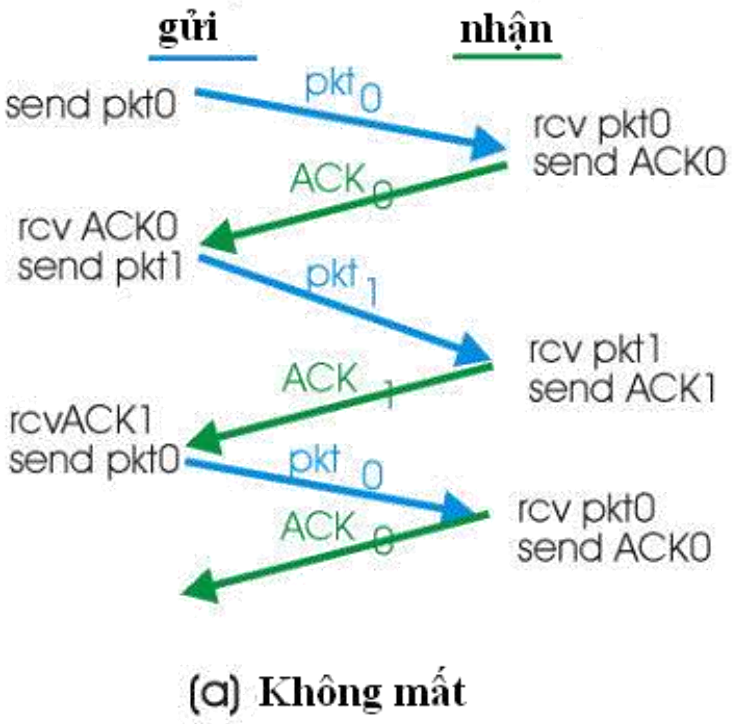
Giải pháp:

- bên gửi đợi một khoảng thời gian hợp lí cho ACK
- Gửi lại nếu không nhận đc ACK trong khoảng thời gian này
- Nếu gói tin (hay ACK) bị trễ (không mất)
  - Gửi lại có thể trùng, phải đánh số thứ tự
  - Bên nhận phải xác định thứ tự của gói tin đã ACK
- Yêu cầu đếm thời gian

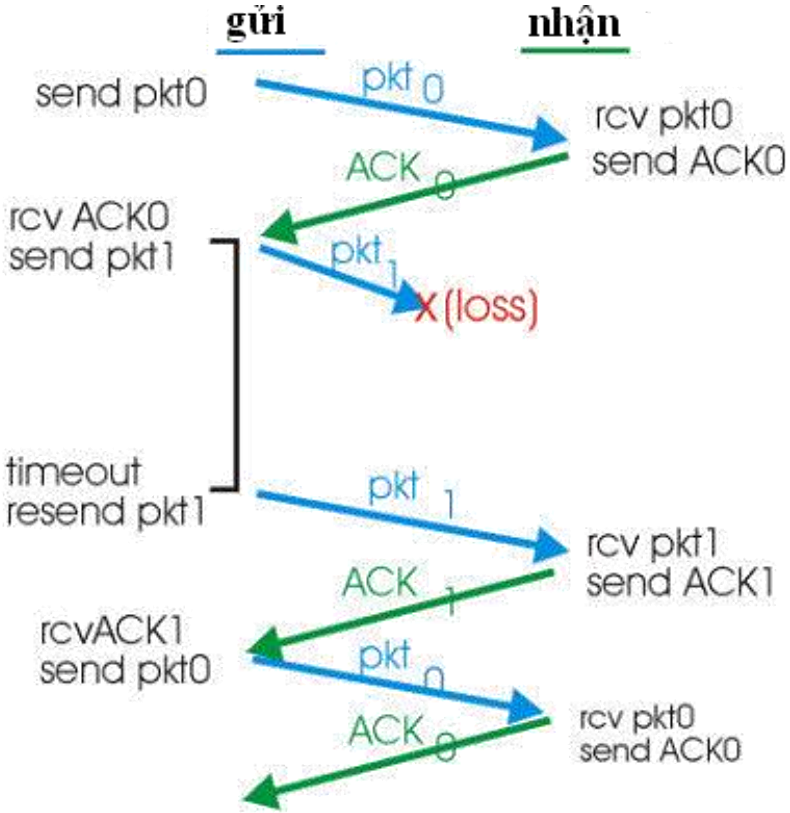
RDT3.0 BÊN GỬI - 2



RDT3.0 - 3

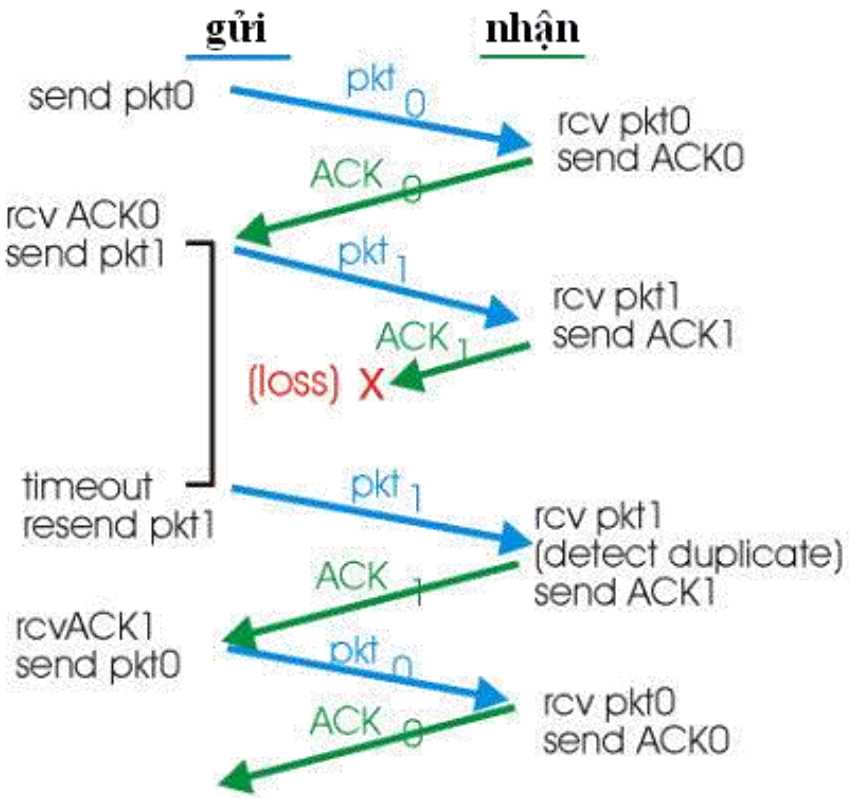


(a) Không mất

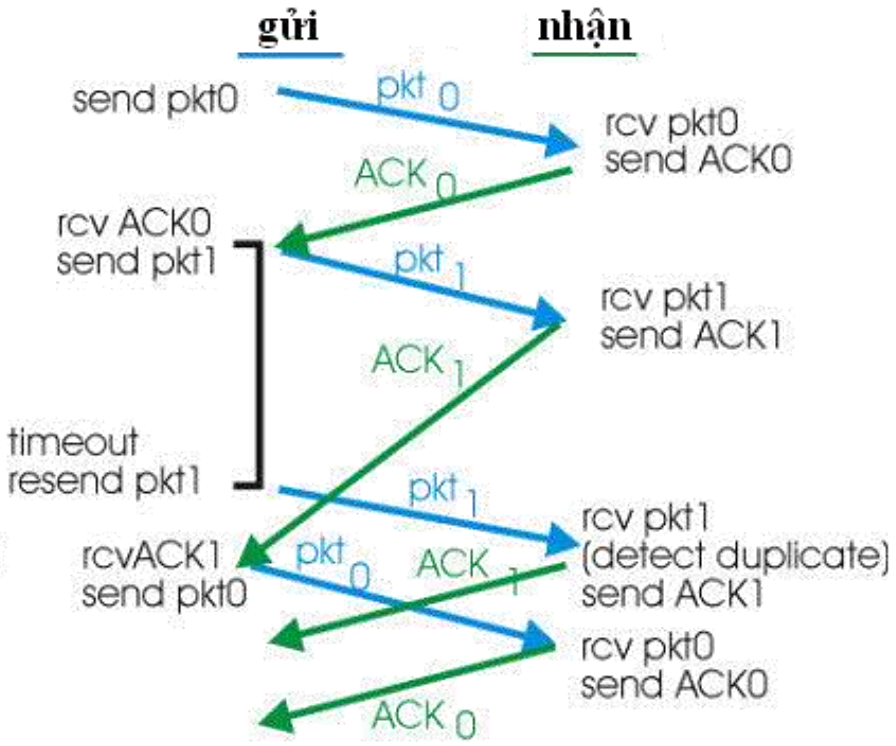


(b) mất gói tin

RDT3.0 - 4

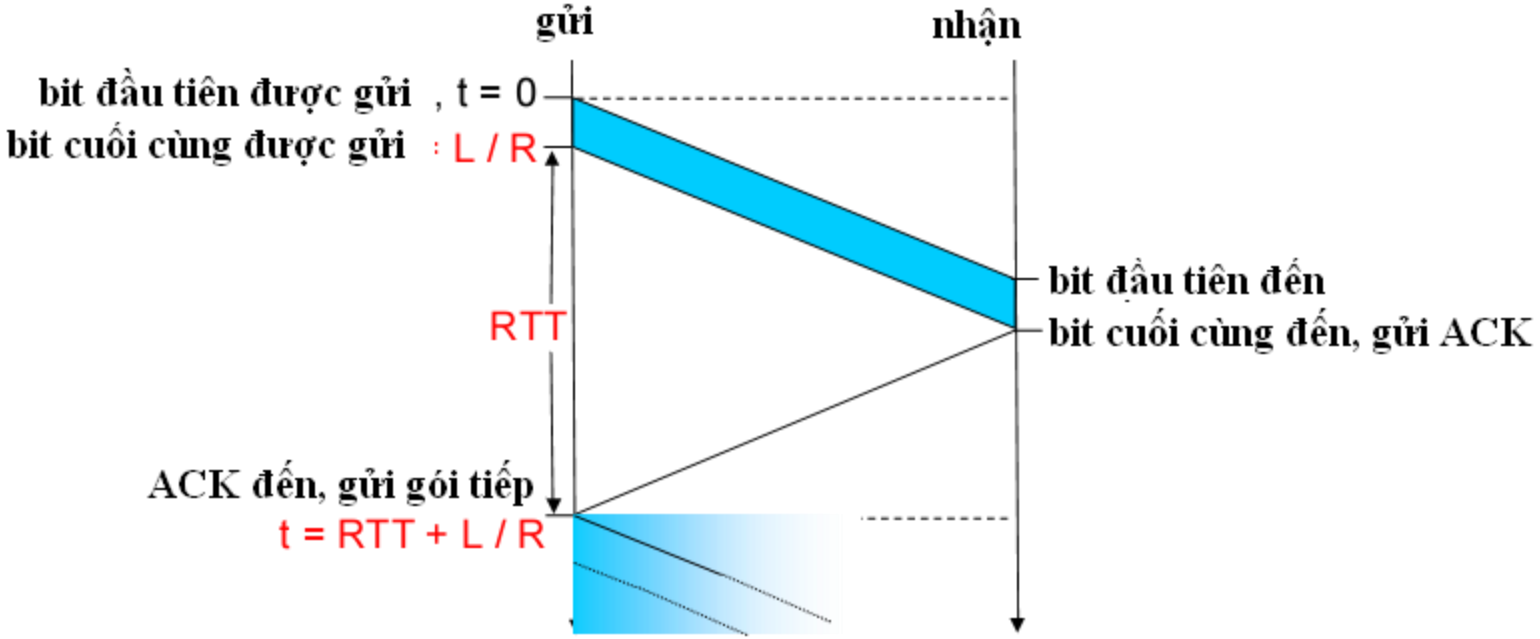


(c) mất ACK



(d) timeout

RDT3.0 DỪNG VÀ ĐỢI - 5



RDT3.0 – HIỆU QUẢ - 6

- Rdt3.0 làm việc, nhưng không hiệu quả
- Vd:băng thông 1Gbps, 15ms end2end delay, gói tin 8Kb

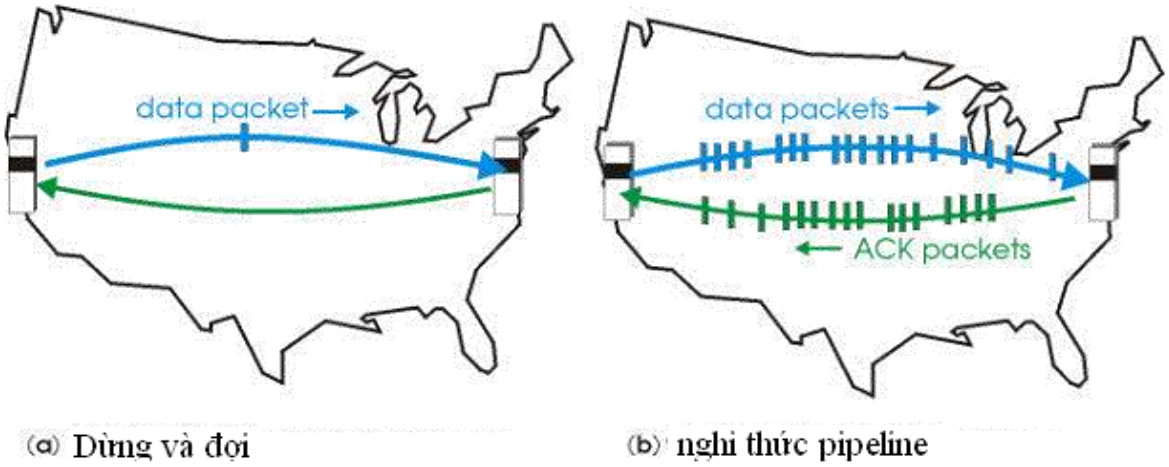
$$T_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8kb/pkt}{10**9 \text{ b/sec}} = 8 \text{ microsec}$$

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- U_{sender} : tỉ lệ thời gian bên gửi gửi gói tin
- Nghi thức đã hạn chế việc sử dụng tài nguyên mạng

NGHI THỨC PIPELINE - 1

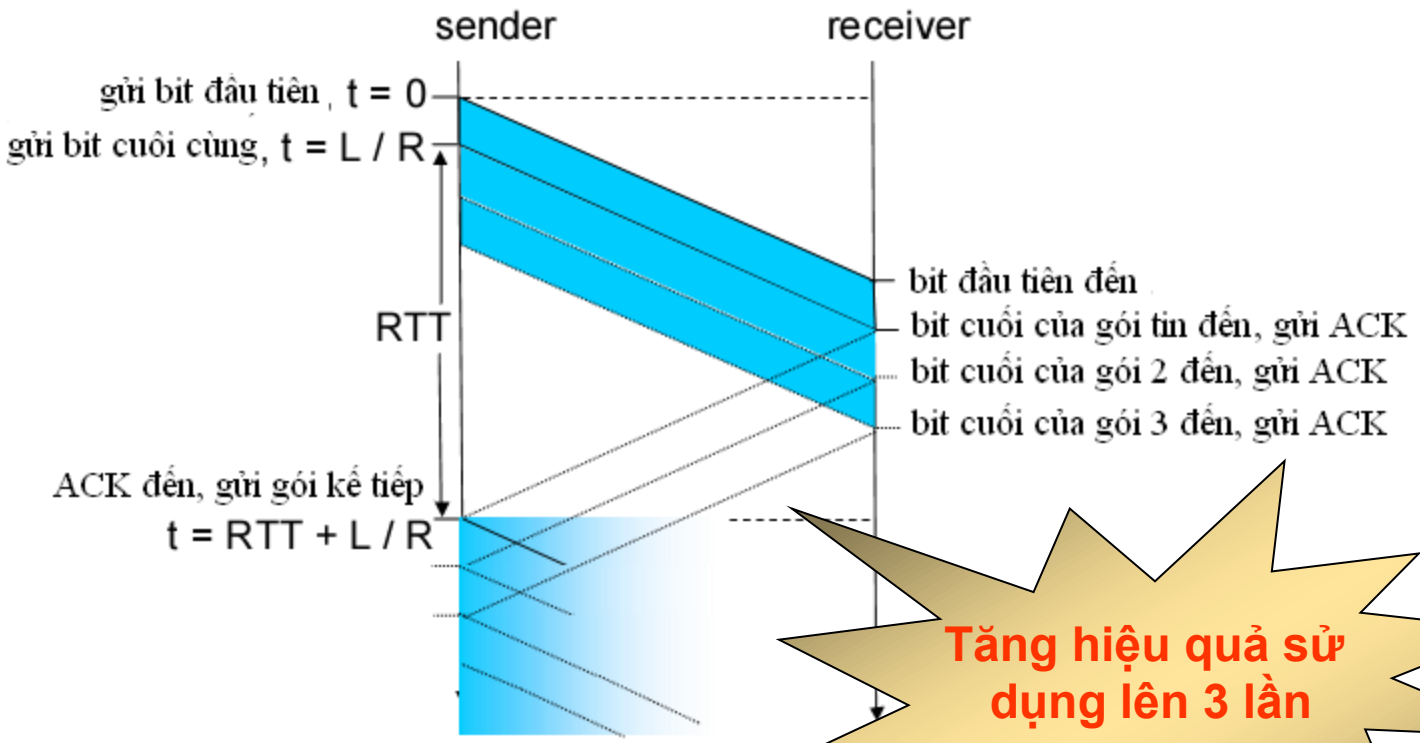
- Pipelining: bên gửi cho phép gửi nhiều gói tin khi chưa được báo nhận (ACK)
  - Gói tin: sắp theo thứ tự tăng dần
  - Dùng bộ đệm ở bên gửi hoặc/và bên nhận: “Sliding window”



(a) Dừng và đợi (b) nghi thức pipeline

- Có hai giải pháp chính của nghi thức pipeline:
  - go-Back-N
  - gửi lại có chọn.

NGHI THỨC PIPELINE - 2

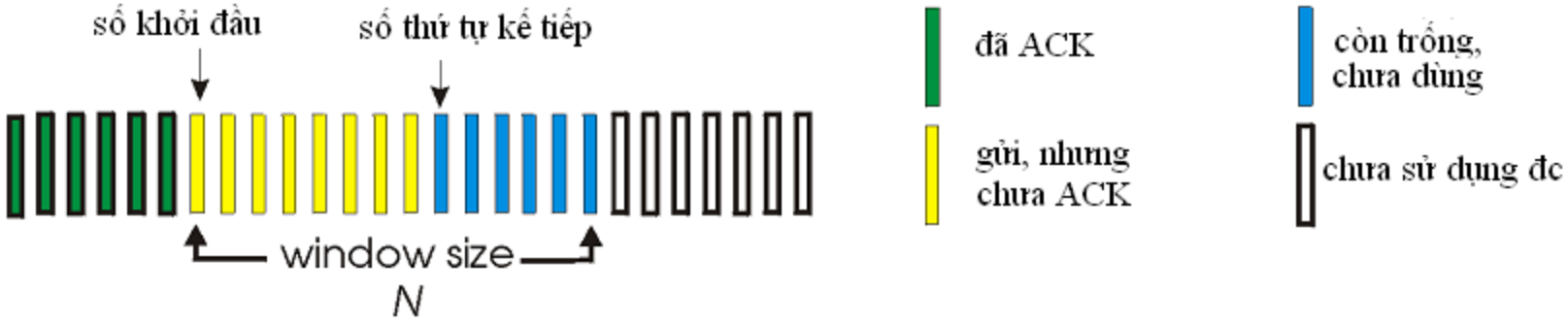


Tăng hiệu quả sử dụng lên 3 lần

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

GO-BACK-N – 1

- Số thứ tự: k-bit
- "window" = N ➔ số gói tin được gởi liên tục không ACK



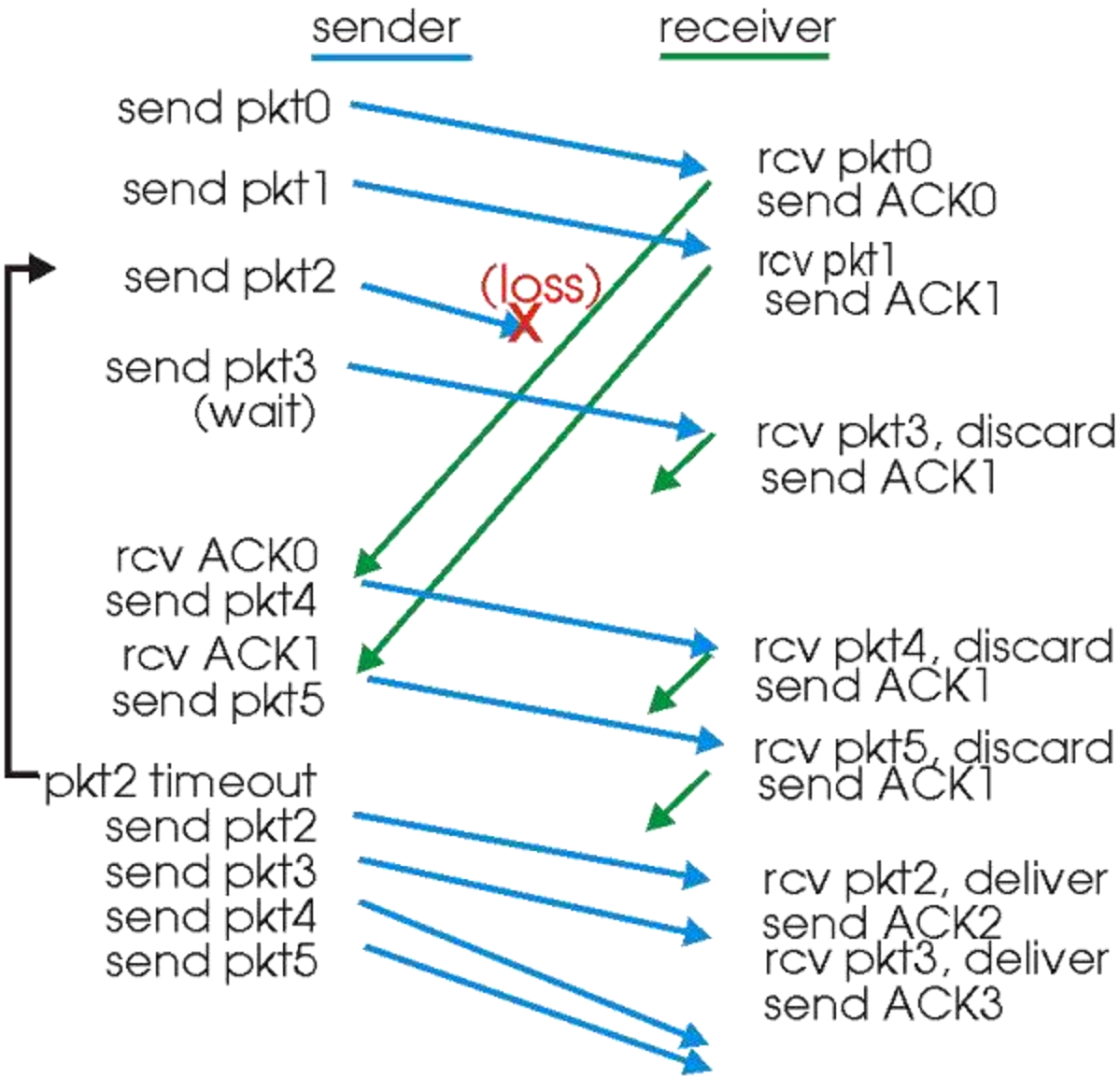
- ACK(seq#): nhận đúng đến seq#

Youtube.com: Go back N sliding window Protocol by Khurram Tanvir

Go-Back-N: bên nhận - 2

- Bên gởi:
  - Sử dụng buffer (“window”) để lưu các gói tin đã gởi nhưng chưa nhận được ACK
  - Gởi nếu gói tin có thể đưa vào “window”
  - Thiết lập đồng hồ cho gói tin cũ nhất (gói tin ở đầu “window”)
  - Timeout: gửi lại tất cả các gói tin chưa ACK trong window
- Bên nhận:
  - Chỉ gửi ACK cho gói tin đã nhận đúng với số thứ tự cao nhất
    - Có thể phát sinh trùng ACK
  - Chỉ cần nhớ số thứ tự đang đợi
  - Gói tin không theo thứ tự:
    - Loại bỏ: không có bộ đệm
    - Gửi lại ACK với số thứ tự lớn nhất

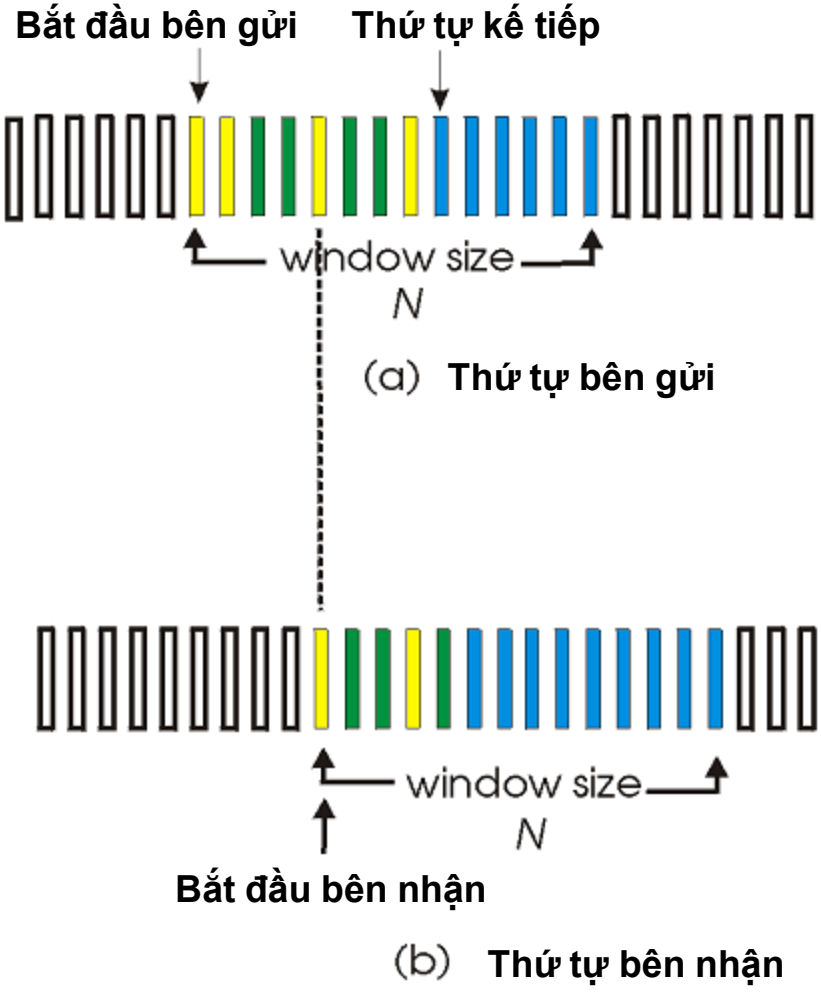
GO-BACK-N – VÍ DỤ - 3



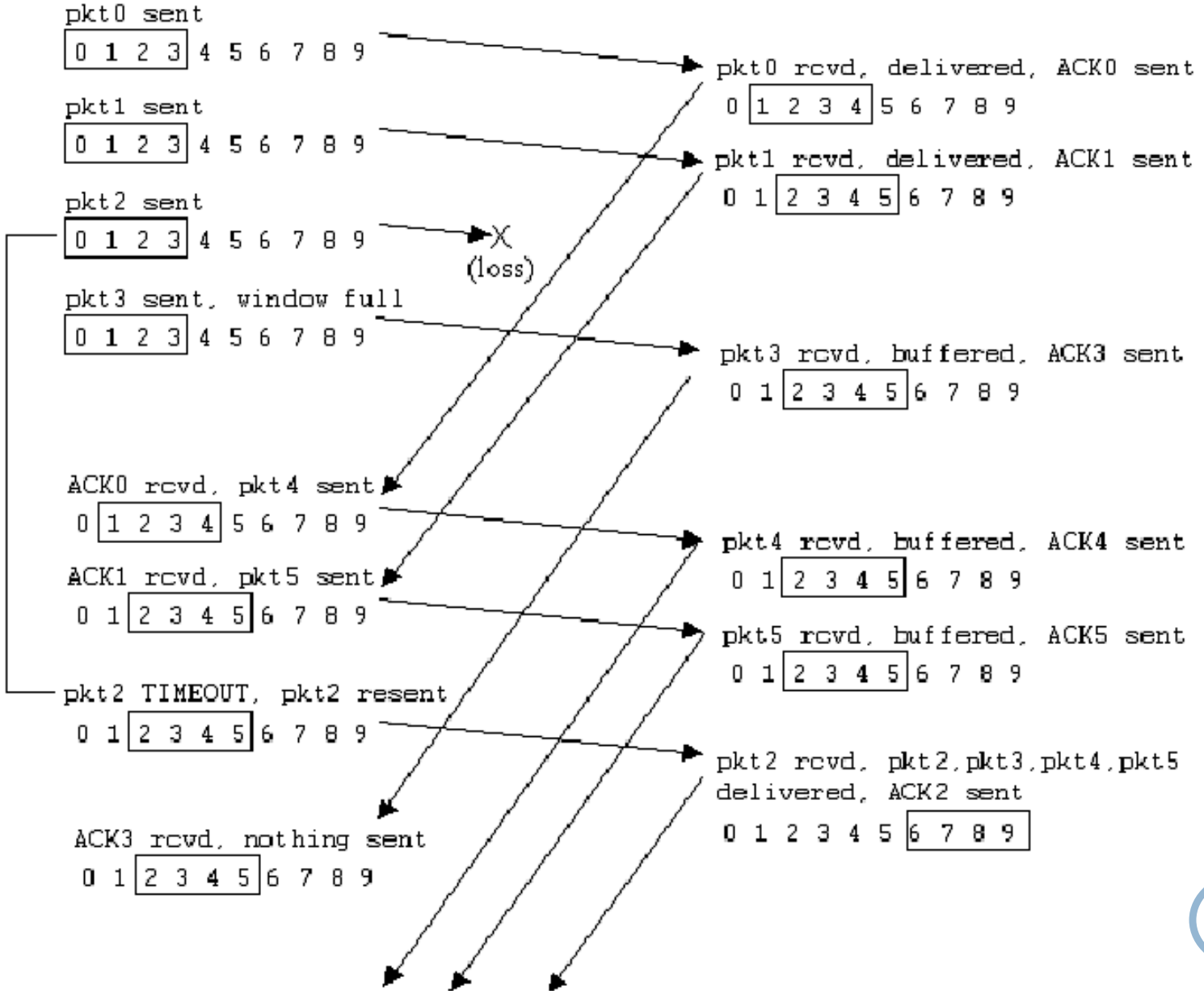
GỬI LẠI CÓ CHỌN - 1

- Bên nhận:
  - Báo nhận riêng lẻ từng gói tin nhận đúng
    - ACK(seq#): đã nhận đúng gói tin seq#
  - dùng bộ đệm để lưu các gói tin không đúng thứ tự
  - Nhận 1 gói tin không đúng thứ tự
    - Đưa vào bộ đệm nếu còn chỗ
    - Hủy gói tin
- Bên gởi:
  - Có đồng hồ cho mỗi gói tin chưa nhận đc ACK
  - Time out: chỉ gửi những gói tin không nhận được ACK

GỬI LẠI CÓ CHỌN - 2



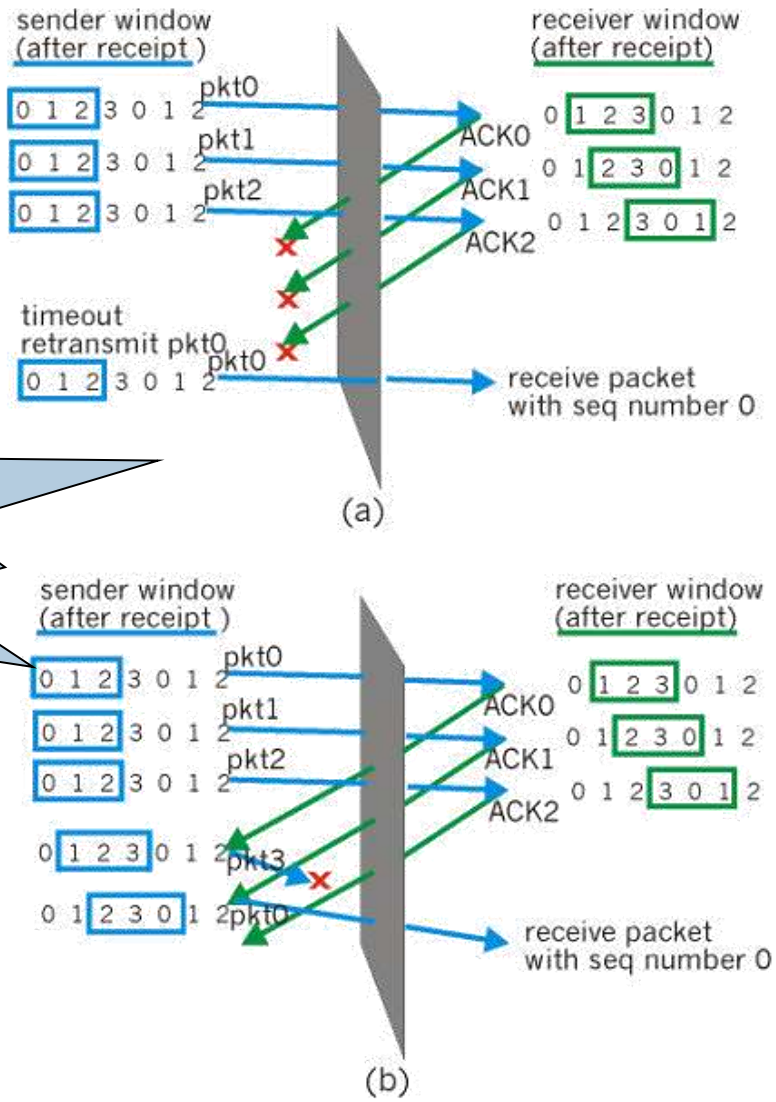
GỬI LẠI CÓ CHỌN - 4



GỬI LẠI CÓ CHỌN - 5

- Vd:
  - Số thứ tự:0,1,2,3
  - Window size: 3

Mối quan hệ giữa số thứ tự và window size???



Nội dung

- Giới thiệu
- Nguyên tắc truyền dữ liệu đáng tin cậy
- Giao thức TCP
- Giao thức UDP

TCP

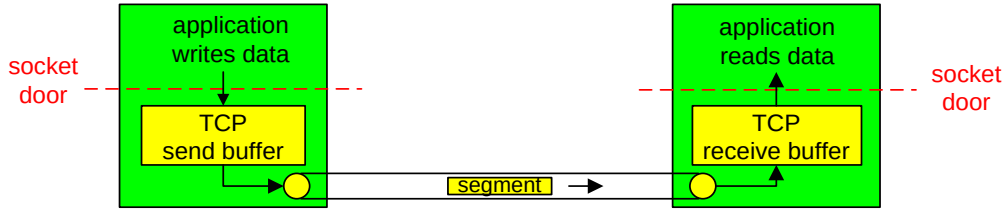
- Giới thiệu
- Nguyên tắc hoạt động
- Quản lý kết nối
- Điều khiển luồng
- Điều khiển tắt nghẽn

TCP – GIỚI THIỆU - 1

- TCP = Transport Control Protocol
  - rfc: 793,1122,1323,2018,2581
  - Point – to – point
    - 1 người gởi và 1 người nhận
  - Full-duplex
    - Dữ liệu truyền 2 chiều trên cùng kết nối
    - MSS: maximum segment size
  - Hướng kết nối
    - Handshaking trước khi gửi dữ liệu

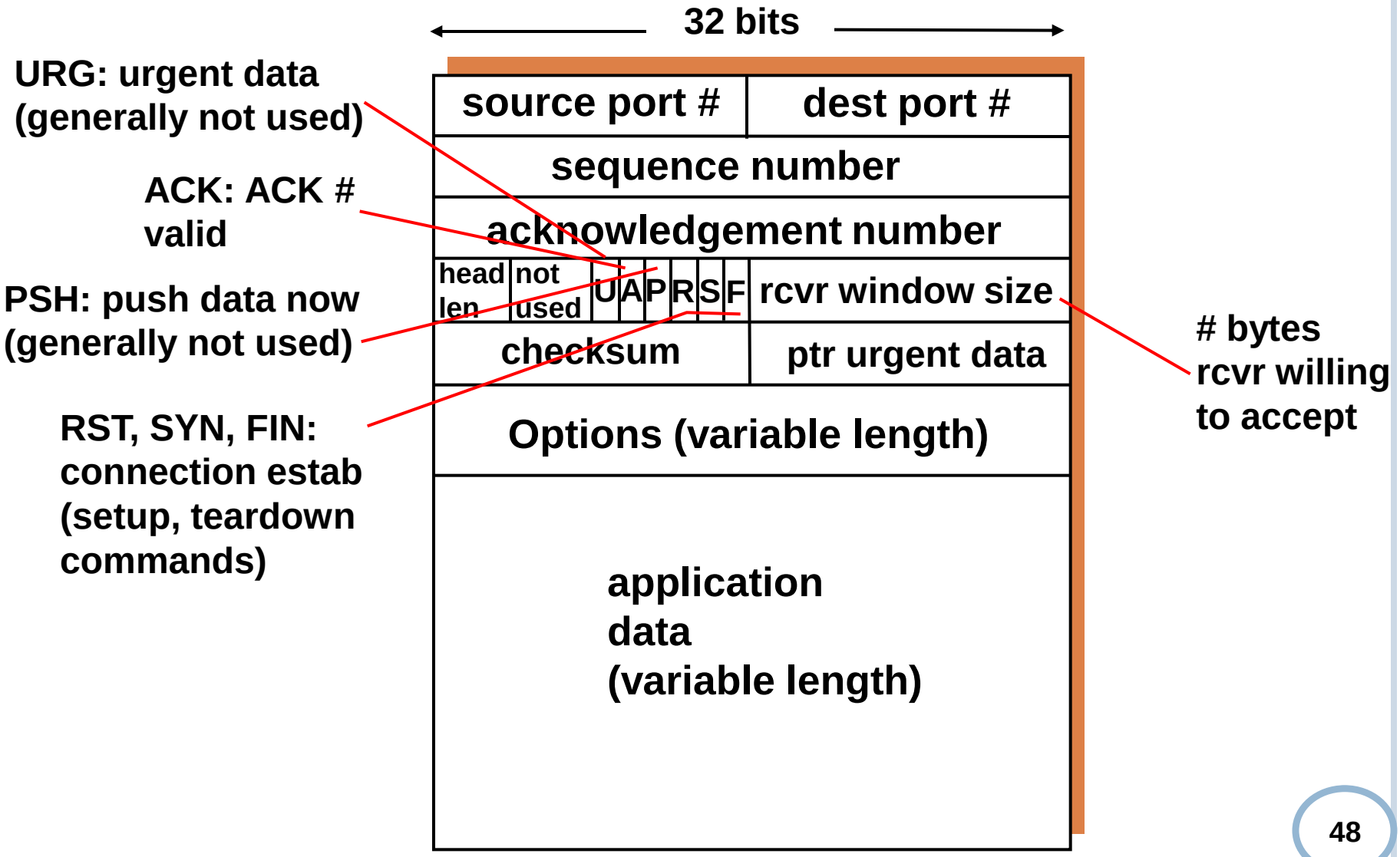
TCP - GIỚI THIỆU - 2

- TCP = Transport Control Protocol
  - TCP cung cấp kết nối theo kiểu dòng (stream-of-bytes)
    - Không có ranh giới giữa các gói tin
    - Sử dụng buffer gởi và nhận



  - Tin cậy, theo thứ tự
  - Pipeline
  - Kiểm soát luồng
  - Kiểm soát tắt nghẽn

TCP – CẤU TRÚC GÓI TIN



TCP – ĐỊNH NGHĨA CÁC TRƯỜNG - 1

- Source & destination port
  - Port của nơi gởi và nơi nhận
- Sequence number
  - Số thứ tự của byte đầu tiên trong phần data của gói tin
- Acknowledgment number
  - Số thứ tự của byte đang mong chờ nhận tiếp theo
- Window size
  - Thông báo có thể nhận bao nhiêu byte sau byte cuối cùng được xác nhận đã nhận

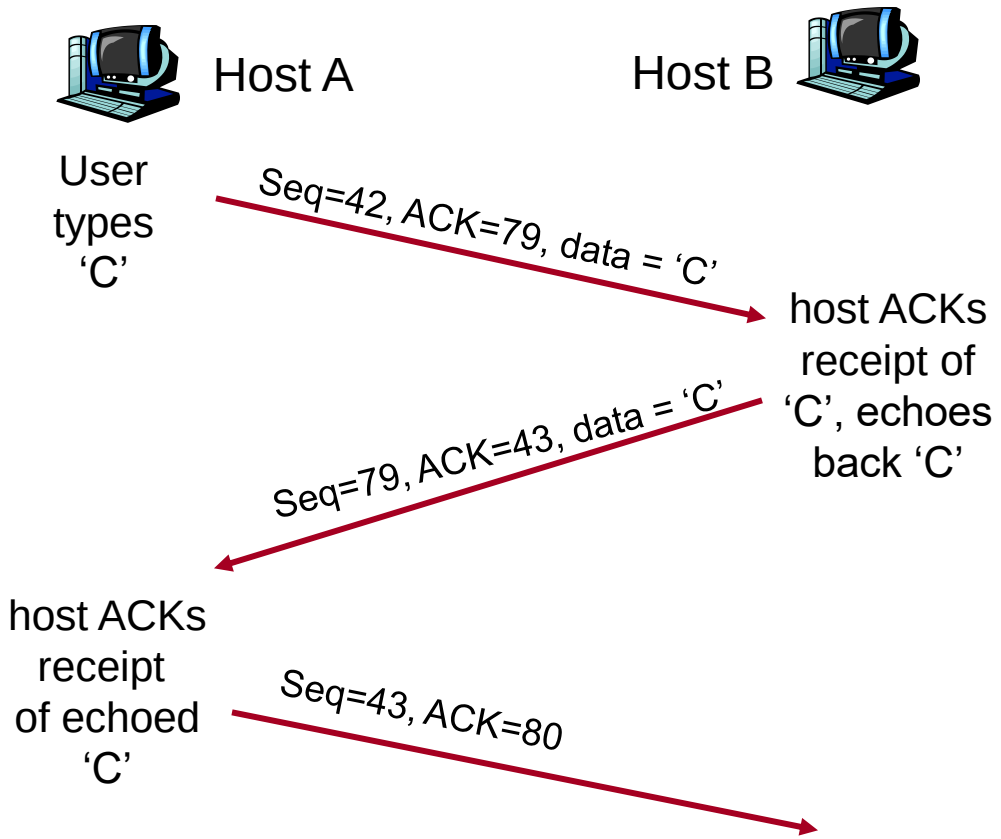
TCP – ĐỊNH NGHĨA CÁC TRƯỜNG - 2

- Checksum
  - Checksum TCP header
- Urgent pointer
  - Chỉ đến dữ liệu khẩn trong trường dữ liệu
- Cờ:
  - URG = trường urgent pointer valid
  - ACK = trường Acknowledge number valid
  - PSH = dữ liệu cần phân phối ngay
  - RST = chỉ định nối kết cần thiết lập lại (reset)
  - SYN = sử dụng để thiết lập kết nối
  - FIN = sử dụng để đóng kết nối

TCP – VÍ DỤ

Seq: số thứ tự của byte đầu tiên trong vùng data

ACK: số thứ tự của byte chờ nhận tiếp theo



simple telnet scenario

TCP – TRUYỀN DỮ LIỆU ĐÁNG TIN CẬY

- Nguyên tắc: dùng pipeline
  - Bên gởi đính kèm thông tin kiểm tra lỗi trong mỗi gói tin
  - Sử dụng ACK để báo nhận
  - Thiết lập thời gian timeout khi cho gói tin ở đầu buffer
  - Gởi lại toàn bộ dữ liệu trong buffer khi hết time out

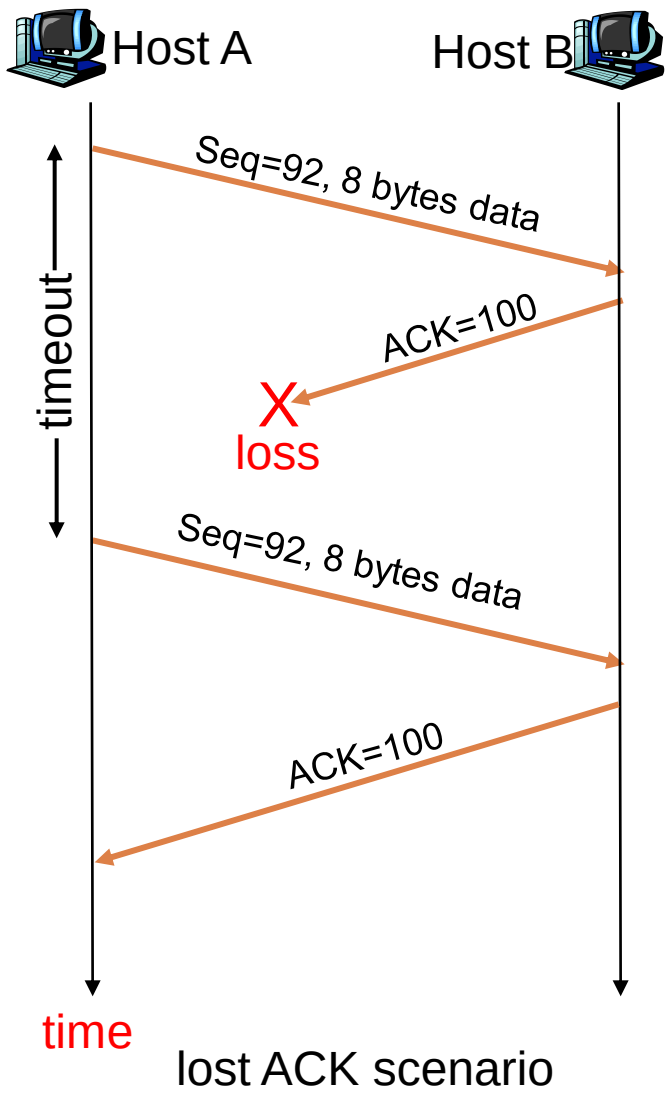
TCP – BÊN GỞI

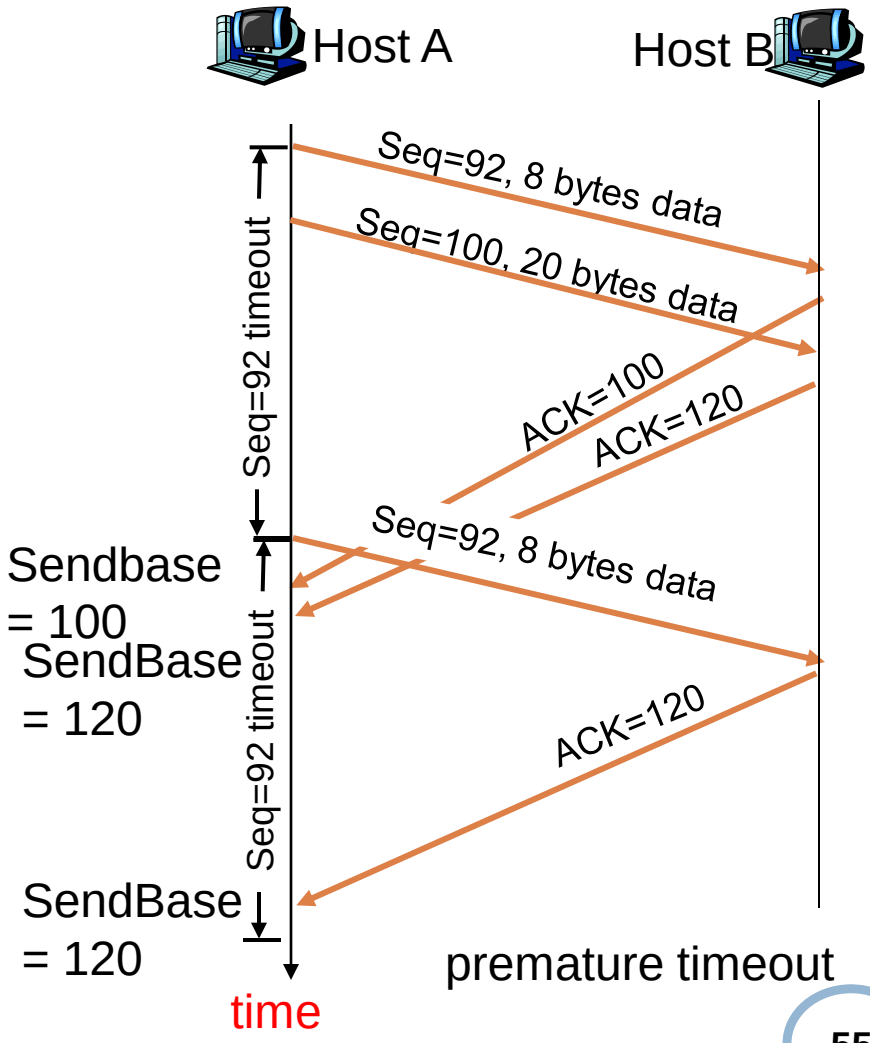
- Nhận dữ liệu từ tầng ứng dụng
  - Tạo các segment
  - Bật đồng hồ (nếu chưa bật)
  - Thiết lập thời gian chờ, timeout
- Nhận gói tin ACK
  - Nếu trước đó chưa nhận: trượt “cửa sổ”
  - Thiết lập lại thời gian của đồng hồ
- Hết time out
  - Gởi lại dữ liệu còn trong buffer
  - Reset đồng hồ

TCP – BÊN NHẬN

- Nhận gói tin đúng thứ tự
  - Chấp nhận
  - Gởi ACK về cho bên gởi
- Nhận gói tin không đúng thứ tự
  - Phát hiện “khoảng trống dữ liệu (GAP)”
  - Gởi ACK trùng

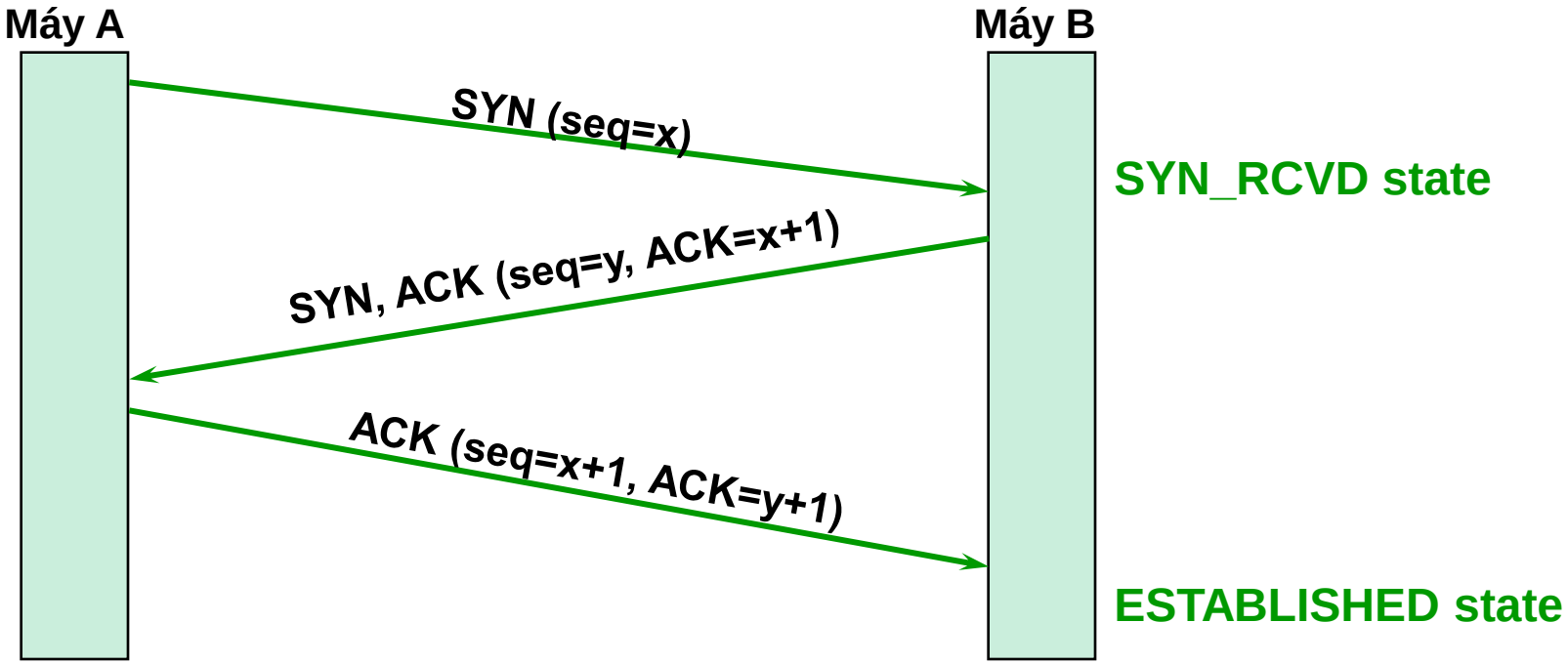
TCP – VÍ DỤ


Host A
Host B
timeout
Seq=92, 8 bytes data
ACK=100
X
loss
Seq=92, 8 bytes data
ACK=100
time
lost ACK scenario


Host A
Host B
Seq=92 timeout
Seq=92, 8 bytes data
Seq=100, 20 bytes data
ACK=100
ACK=120
Seq=92, 8 bytes data
Sendbase
= 100
SendBase
= 120
Seq=92 timeout
ACK=120
SendBase
= 120
premature timeout
time

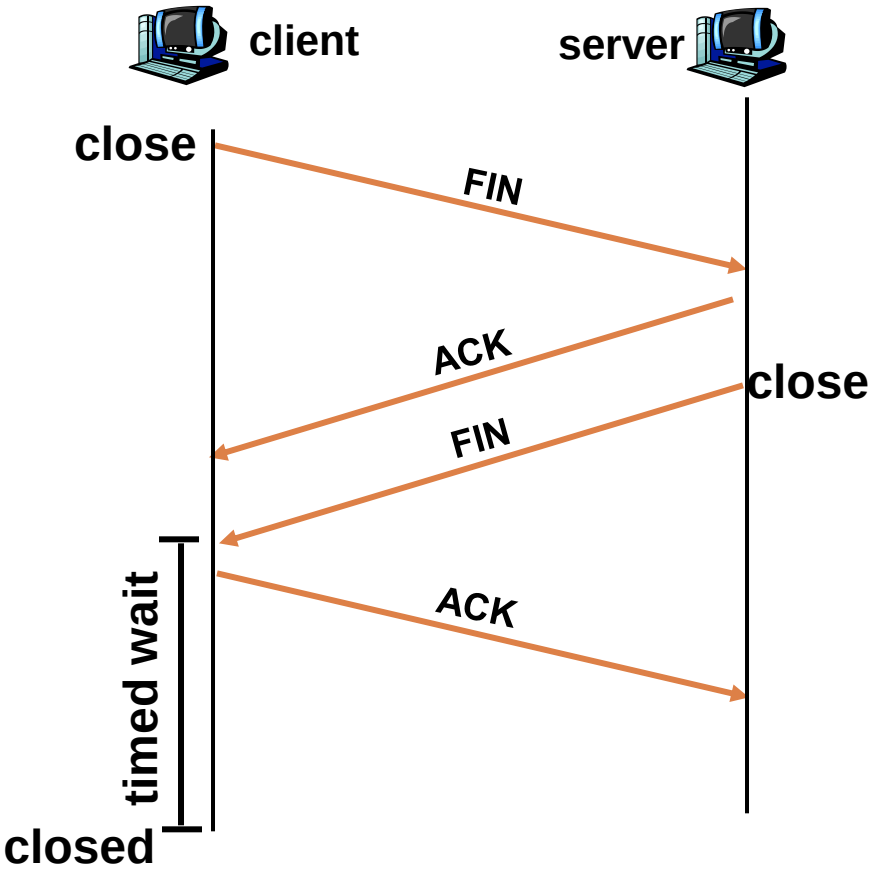
TCP – THIẾT LẬP KẾT NỐI

- Thực hiện thao tác bắt tay 3 lần (Three way handshake)

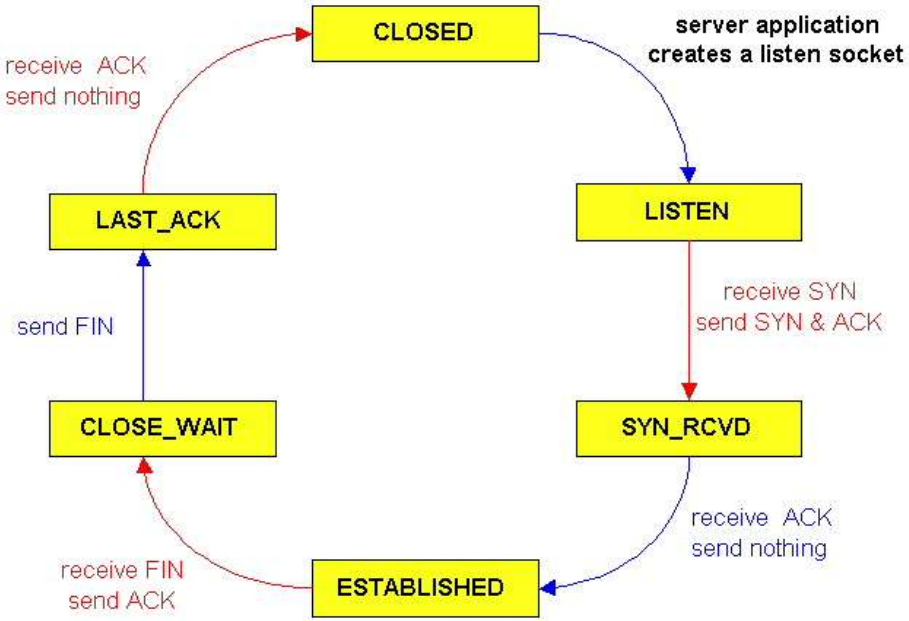


TCP – ĐÓNG KẾT NỐI

- Thực hiện thao tác bắt tay 2 lần

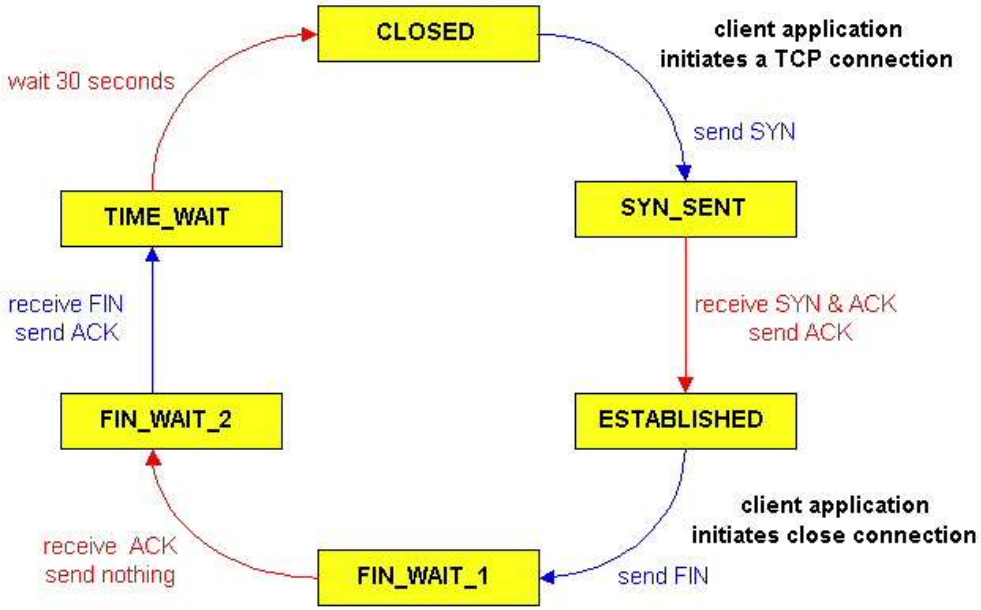


TCP – QUẢN LÝ KẾT NỐI

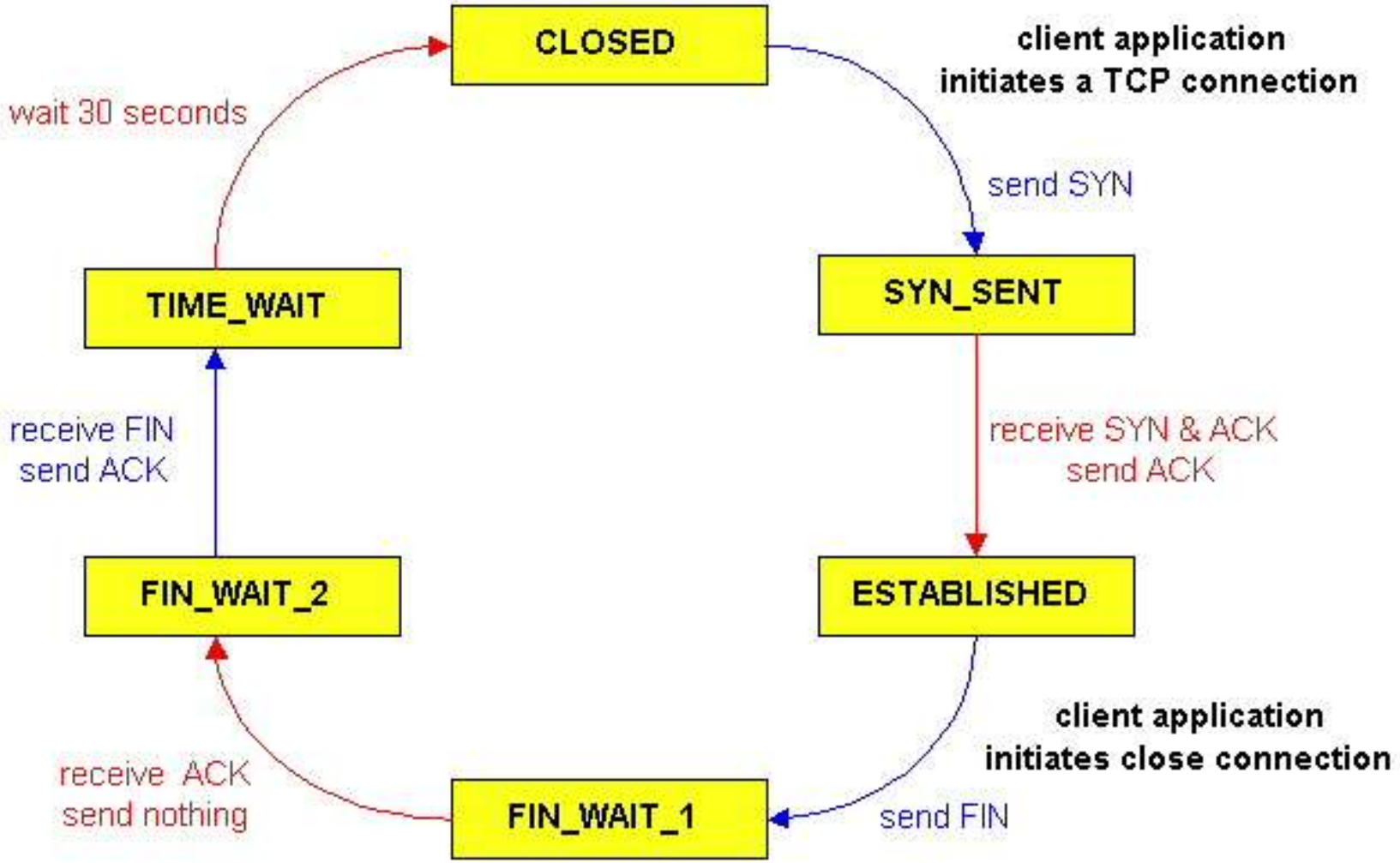


TCP server lifecycle

TCP client lifecycle

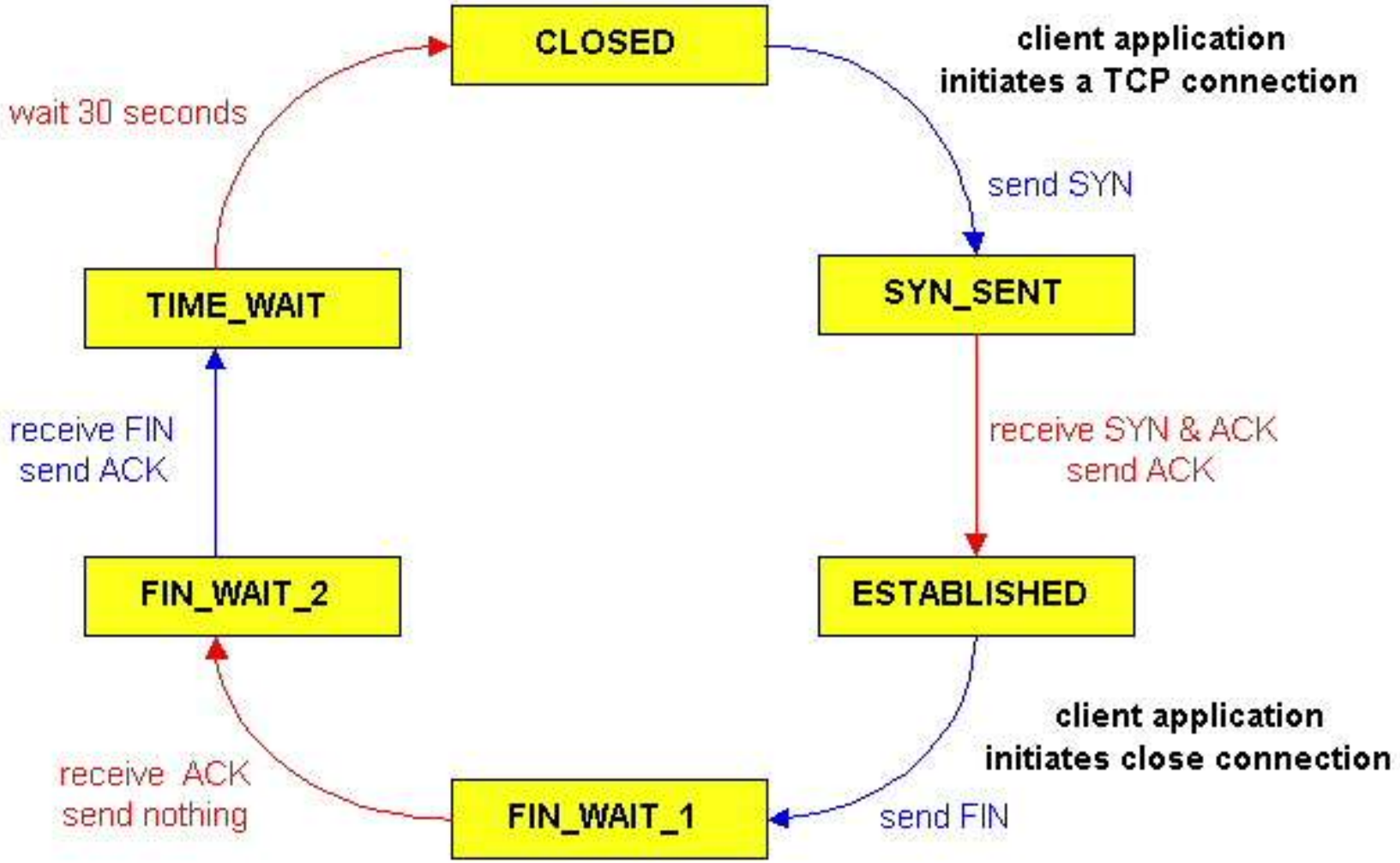


TCP – QUẢN LÝ KẾT NỐI



TCP client lifecycle

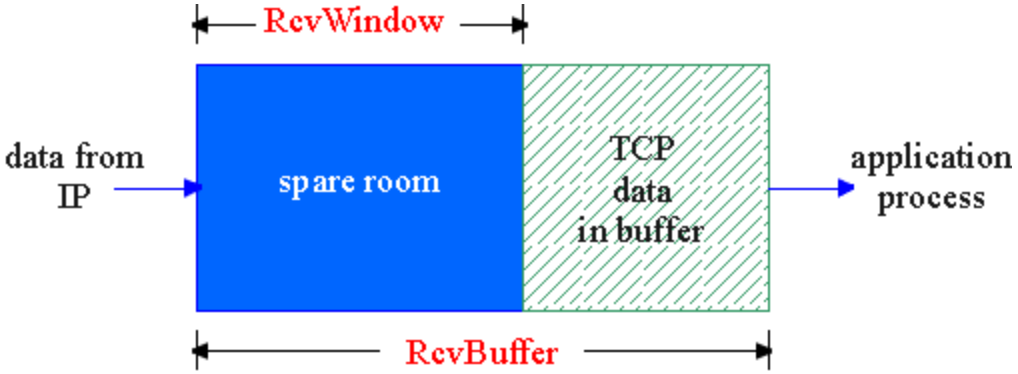
TCP – QUẢN LÝ KẾT NỐI



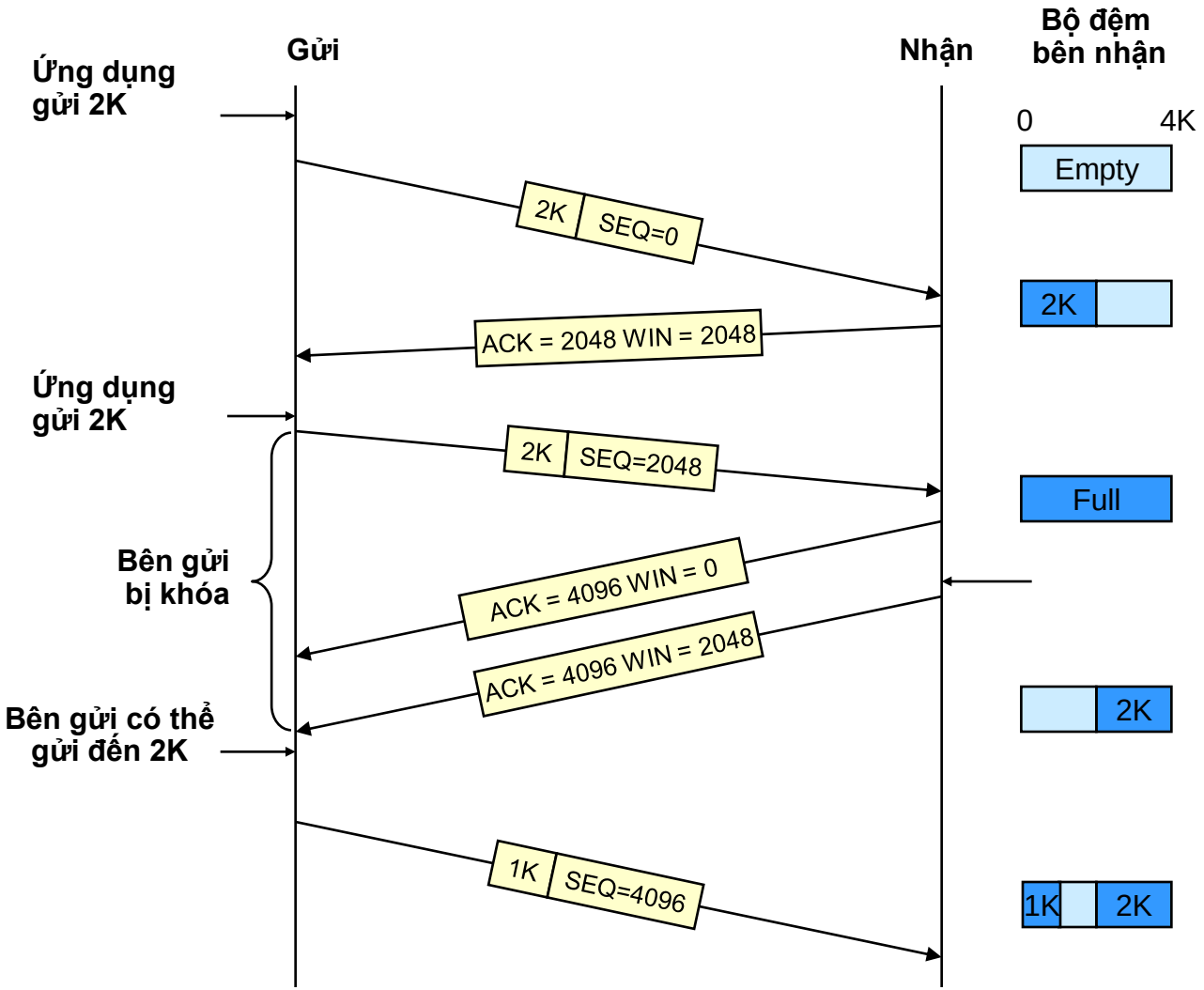
TCP client lifecycle

TCP - Điều khiển luồng - 1

- Nguyên nhân:
  - Bên gởi làm tràn bộ đệm của bên nhận khi gởi quá nhiều dữ liệu hoặc gởi quá nhanh
- Sử dụng trường "window size"
  - Window size: lượng DL có thể đưa vào buffer



TCP - ĐIỀU KHIỂN LUỒNG - 2



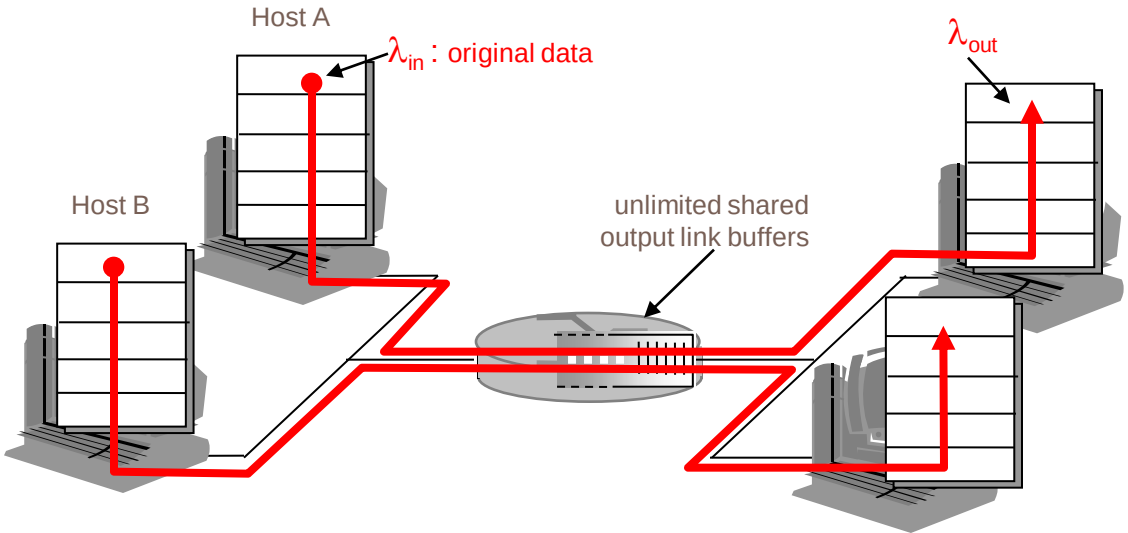
KIỂM SOÁT TẮT NGHẼN - 1

- Vấn đề: 1 node có thể nhận dữ liệu từ nhiều nguồn
  - Buffer: giới hạn
  - gói tin: đến ồ ạt

➔ xử lý không kịp ➔ tắt nghẽn

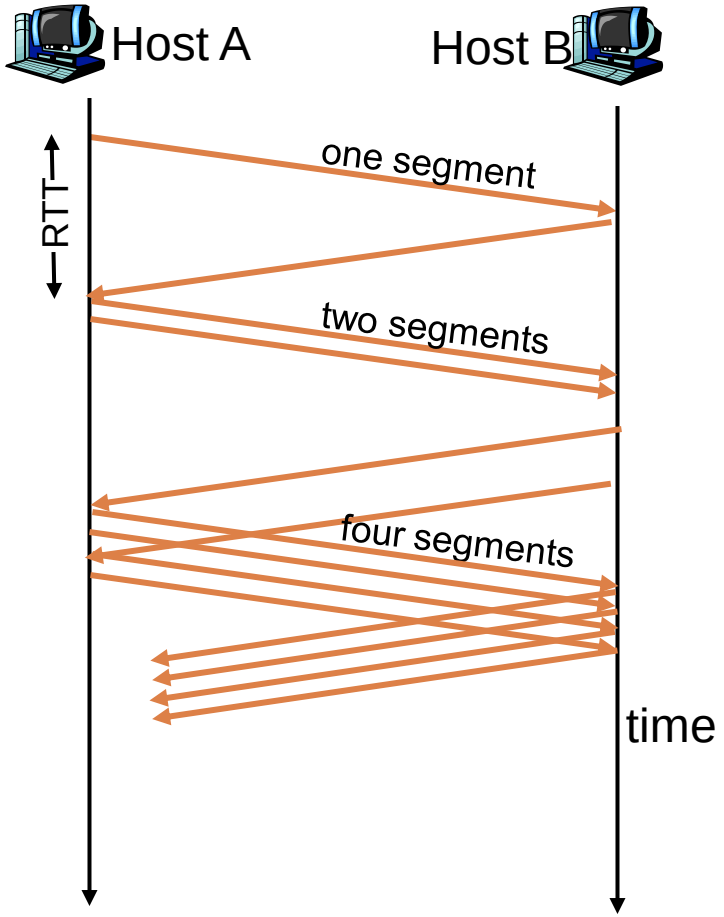
- Hiện tượng:
  - Mất gói
  - Delay cao

➔ Sử dụng đường truyền không hiệu quả



KIỂM SOÁT TẮT NGHẼN - 2

- Giải quyết trong TCP:
  - Bên gởi:
    - Thiết lập tốc độ gởi dựa trên phản hồi từ bên nhận
      - Nhận ACK
      - Mất gói
      - Độ trễ gói tin
  - Tốc độ gởi: có 2 pha
    - Slow-Start
    - Congestion Avoidance



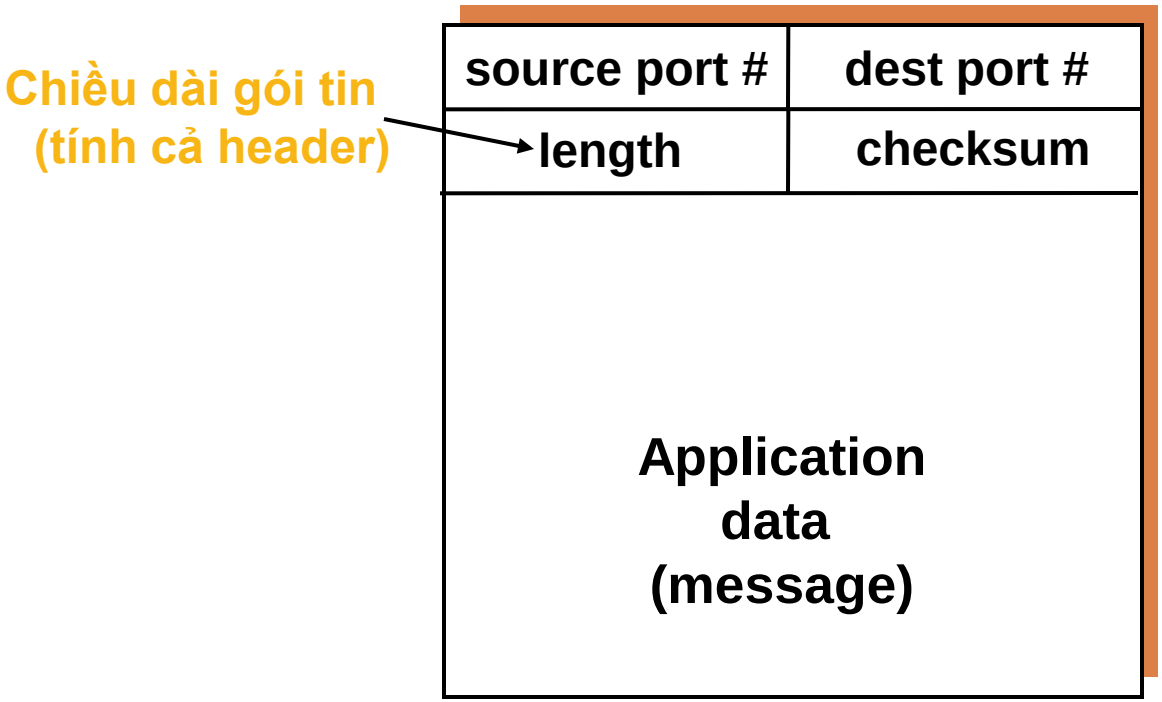
Nội dung

- Giới thiệu
- Nguyên tắc truyền dữ liệu đáng tin cậy
- Giao thức TCP
- Giao thức UDP

UDP - 1

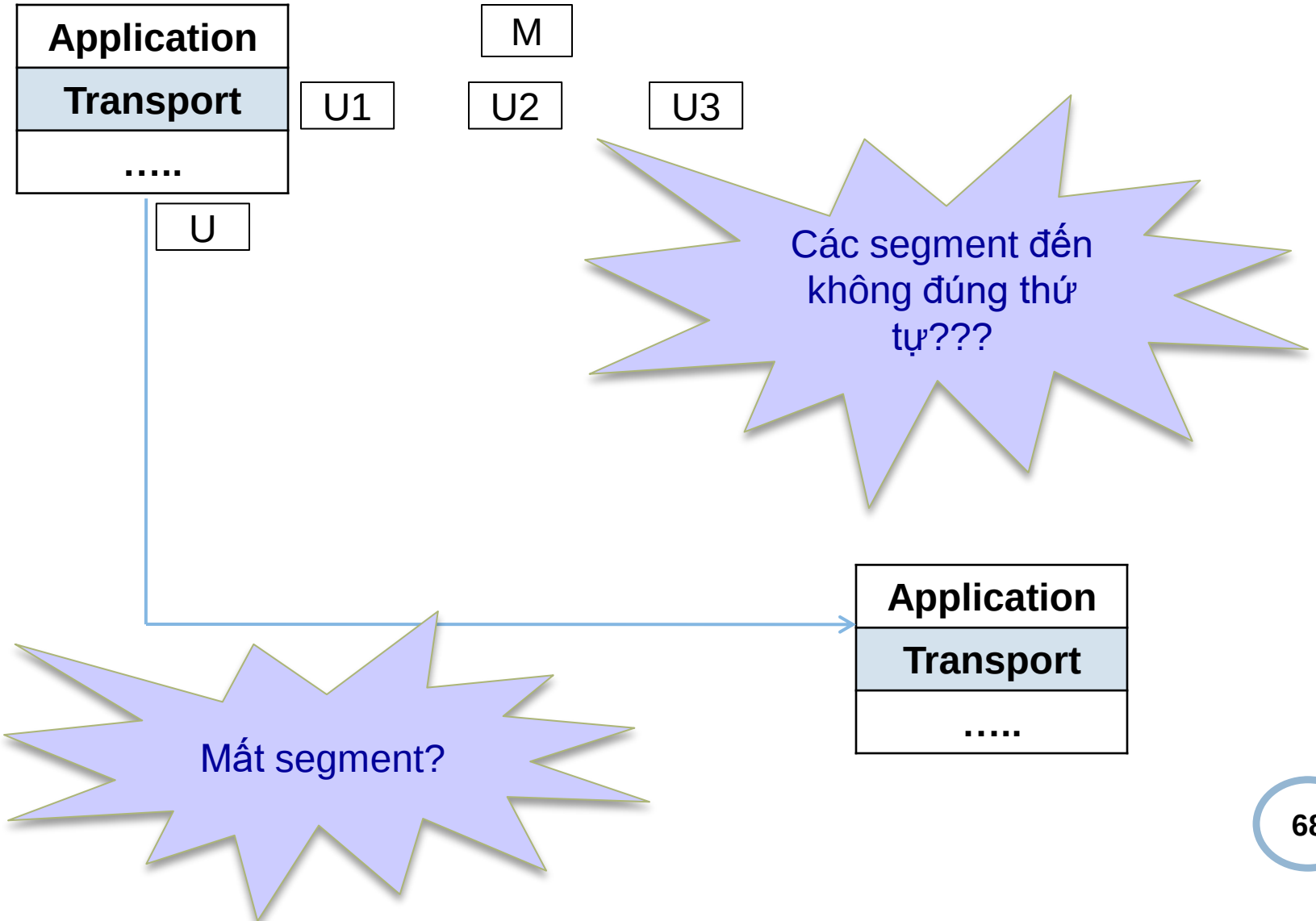
- UDP: User Datagram Protocol [rfc768]
  - Dịch vụ “nỗ lực” để truyền nhanh
  - Gói tin UDP có thể:
    - Mất
    - Không đúng thứ tự
  - Không kết nối:
    - Không có handshaking giữa bên gửi và nhận
    - Mỗi gói tin UDP được xử lý độc lập
    - Không có trạng thái kết nối

UDP - 2



UDP segment format

UDP - 3


Application
Transport
.....
M
U1
U2
U3
U
Các segment đến không đúng thứ tự???
Application
Transport
.....
Mất segment?

UDP - 4

- Tại sao lại sử dụng UDP?
  - Không thiết lập kết nối
  - Đơn giản:
    - không quản lý trạng thái nối kết
    - Không kiểm soát luồng
  - Header nhỏ
  - Nhanh
- Truyền thông tin cậy qua UDP
  - Tầng application phát hiện và phục hồi lỗi

UDP - 5

- Thường sử dụng cho các ứng dụng multimedia
  - Chịu lỗi
  - Yêu cầu tốc độ
- Một số ứng dụng sử dụng UDP
  - DNS
  - SNMP
  - TFTP
  - …

TÀI LIỆU THAM KHẢO

- Bài giảng của J.F Kurose and K.W. Ross về Computer Networking: A Top Down Approach