CHƯƠNG 4
TCP CONGESTION CONTROL

ET4230

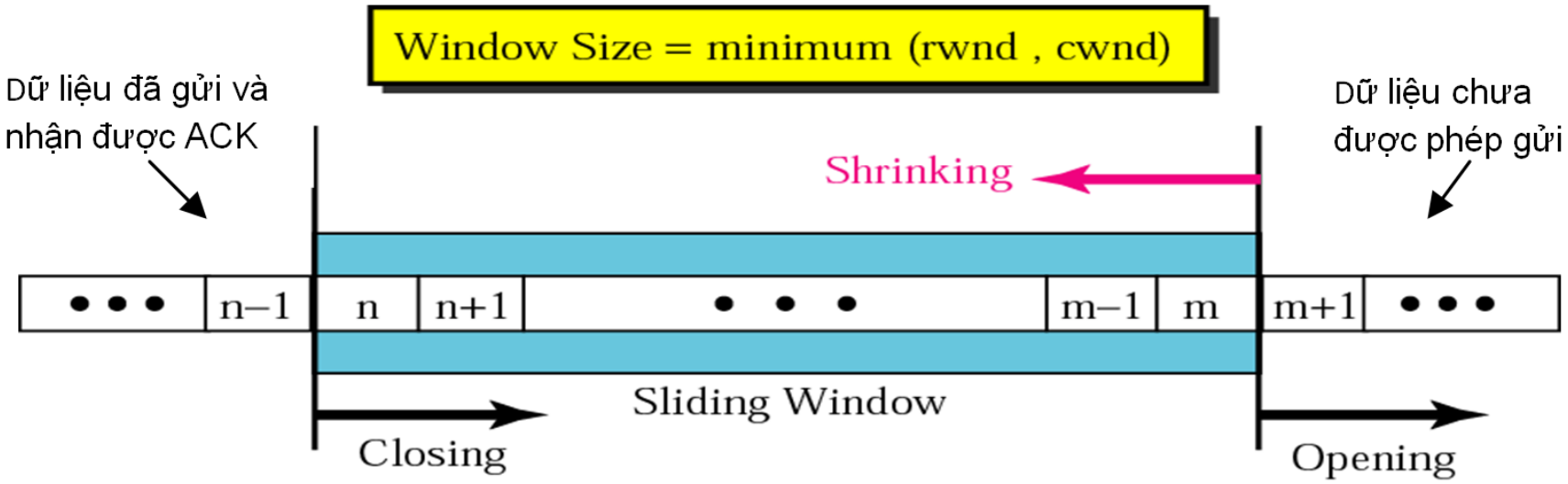
TS. Trần Quang Vinh
BM. Kỹ thuật Thông tin
Viện Điện tử - Viễn thông
Đại học Bách Khoa Hà Nội
vinhtq@hust.edu.vn

MỤC ĐÍCH

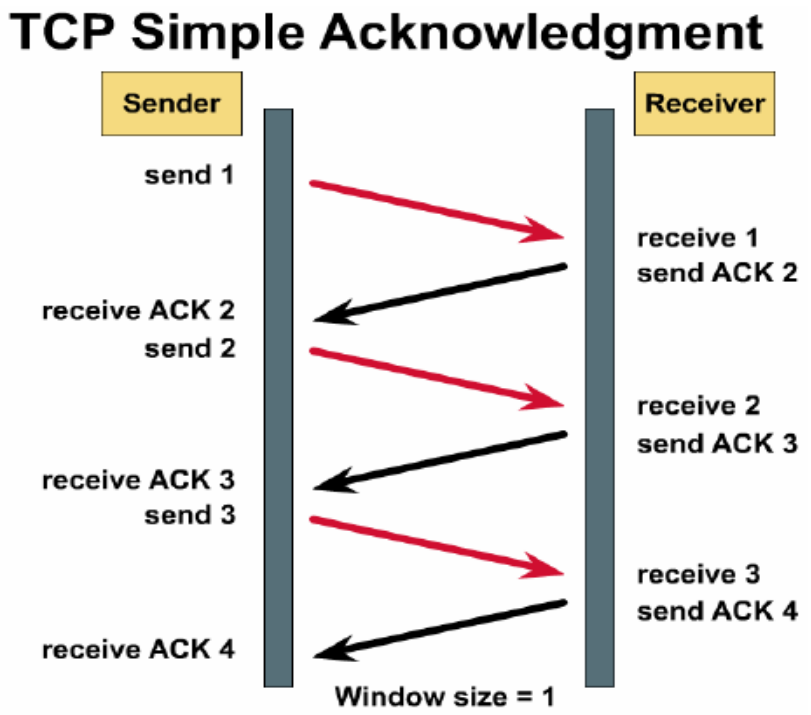
- Mục đích
  - Tối ưu hóa thông lượng sử dụng của mạng
  - Giảm trễ gói khi đi qua mạng
  - Đảm bảo tính công bằng cho việc trao đổi thông tin trên mạng
  - Đảm bảo tránh tắc nghẽn trong mạng
- Điều khiển luồng được thực hiện ở tầng nào?
  - Lớp giao vận:
    - Điều khiển luồng từ đầu cuối đến đầu cuối (end-to-end)
    - Tránh tràn bộ đệm của quá trình nhận tại đích
  - Lớp Liên kết dữ liệu và lớp Mạng:
    - Điều khiển luồng trên từng chặng (hop-by-hop):
    - Tránh cho từng đường truyền khỏi bị tắc nghẽn

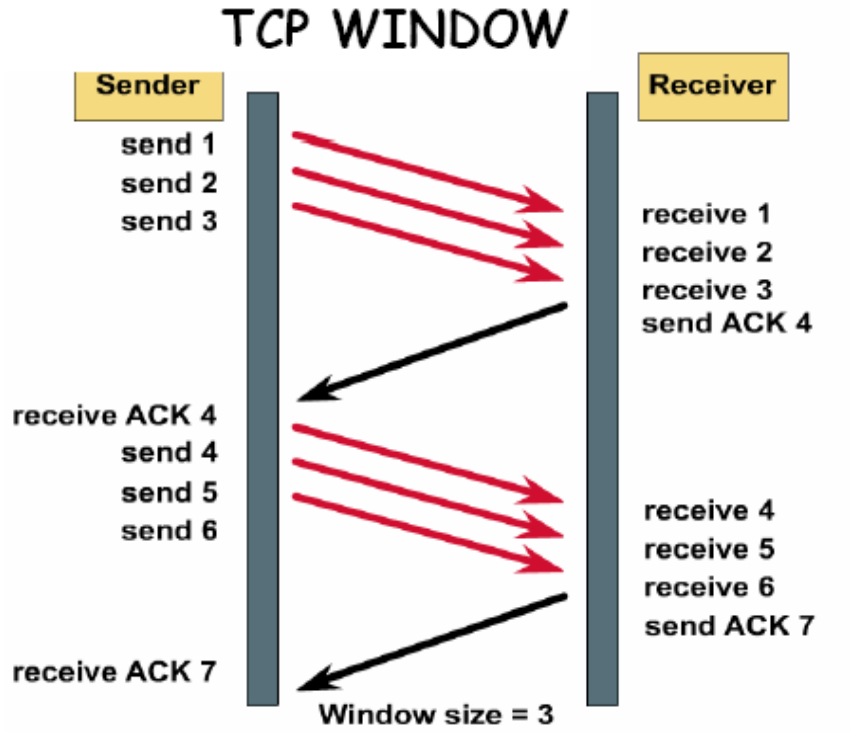
Kiểm soát luồng (Flow Control)

- Cơ chế cửa sổ trượt (sliding window)
  - Cửa sổ nhận (Rwnd): số lượng data tối đa bên thu có thể nhận
    - Trường Receive Window Size (4kB – 8kB)
  - Cửa sổ gửi (Cwnd: Congestion window size), tương ứng với các segment mà nó được phép gửi



TCP Window





RTT > Window Size: hiệu suất kênh truyền thấp
RTT= Window Size: hiệu suất kênh đạt 100%.

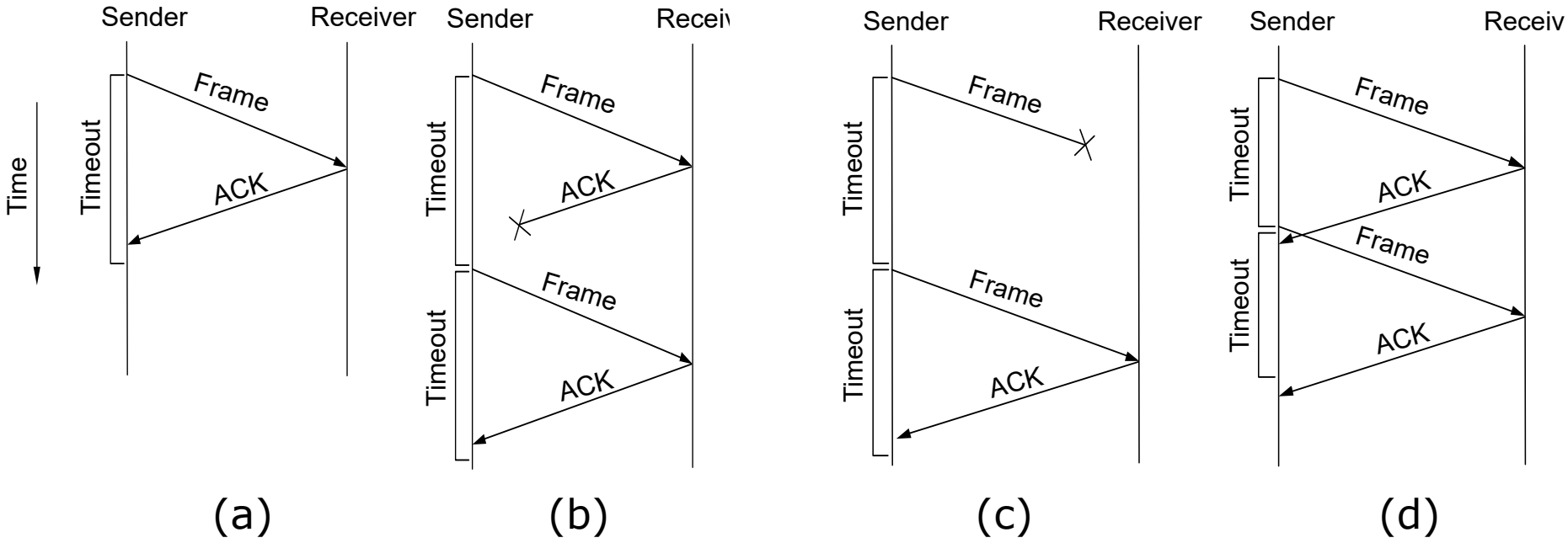
RTT phụ thuộc vào nhiều loại trễ lan truyền, hàng đợi, ...

Kiểm soát lỗi (Error Control)

- Checksum
- Acknowledgement (ACK)
  - Rule 1: Gói ACK không được gán sequence number và không có báo nhận cho bản thân gói ACK
- Timeout (Retransmission TimeOut -RTO)
  - Sau khoảng thời gian RTO>RTT mà không nhận được ACK → truyền lại
- Retransmission:
  - Rule 2: việc truyền lại sảy ra khi timeout hoặc khi nhận được 3 ACK liên tiếp trùng nhau
  - Rule 3: Không thiết lập timer cho gói ACK
- Out-of-order
  - Rule 4: Dữ liệu đến bên nhận có thể không đúng thứ tự, chúng được lưu trong bộ đệm bên thu và chỉ được chuyển lên tầng trên khi đã sắp xếp lại

mất 1 gói nhưng ACK vẫn gửi về

Kiểm soát lỗi (Error Control)



Cơ chế truyền lại sử dụng timeout:

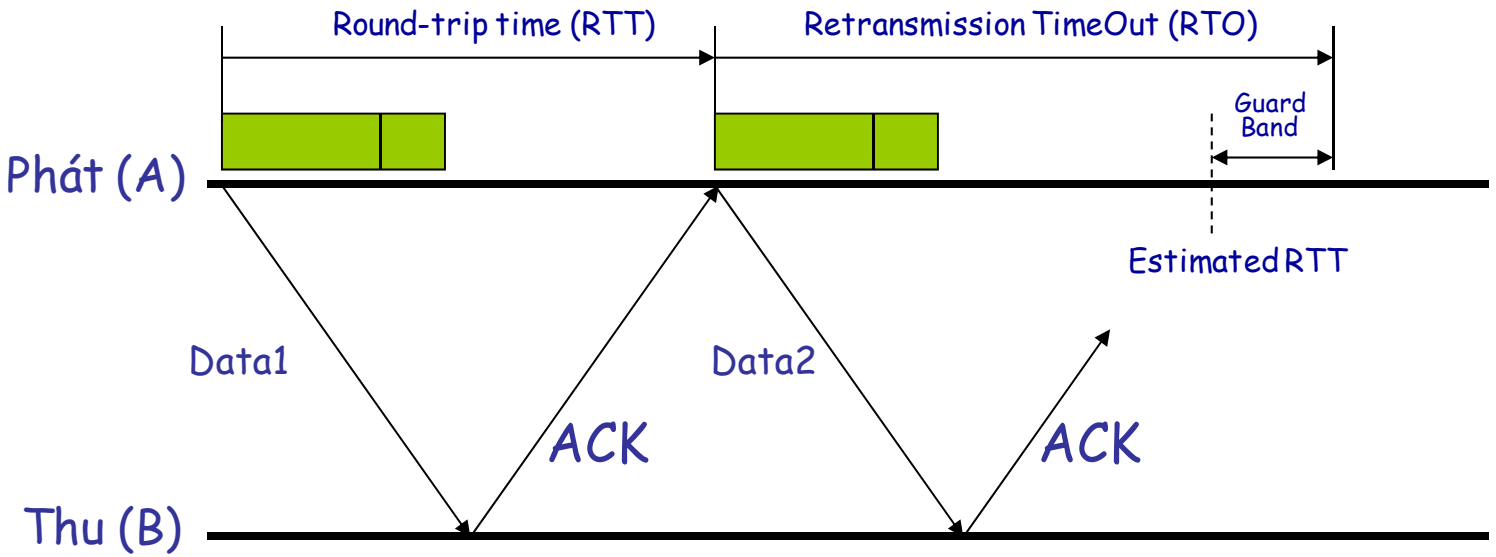
(a) trường hợp bình thường

(b,c) truyền lại khi mất ACK hoặc mất frame

(d) timeout < RTT gay ra phát trùng frame

RTO

- RTO thay đổi, thích ứng với sự thay đổi RTT



Round trip time + một khoảng bảo vệ để thích ứng với sự thay đổi của mạng

Thuật toán Jacobson

- EWMA với thuật toán **Jacobson**

+ Tính giá trị ước lượng eRTT:

$$eRTT_k = \alpha\, eRTT_{k-1} + (1 - \alpha)\, SampleRTT \quad (1)$$

+ Thiết lập timeout dựa trên eRTT

$$RTO = 2 * eRTT_k \quad (2)$$

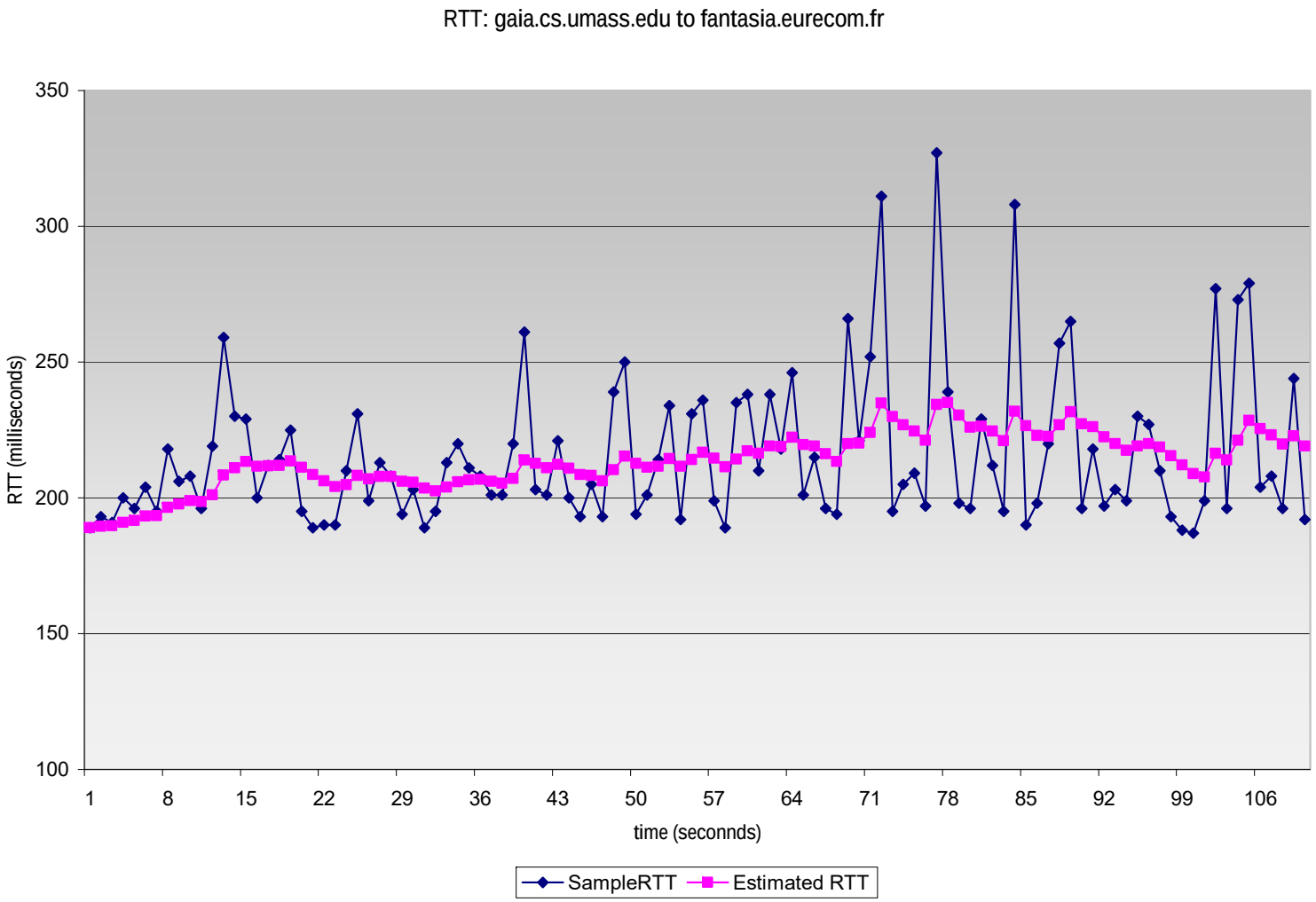
Trong đó:

+ α là trọng số thuật toán EWMA ($0 \leq \alpha \leq 1$, thông thường $\alpha = 7/8$),

+ k là số bước lặp

+ SampleRTT: giá trị RTT đo được (thực tế) cho mỗi cặp packet/ACK tại bước k.

Thuật toán Jacobson (tiếp)



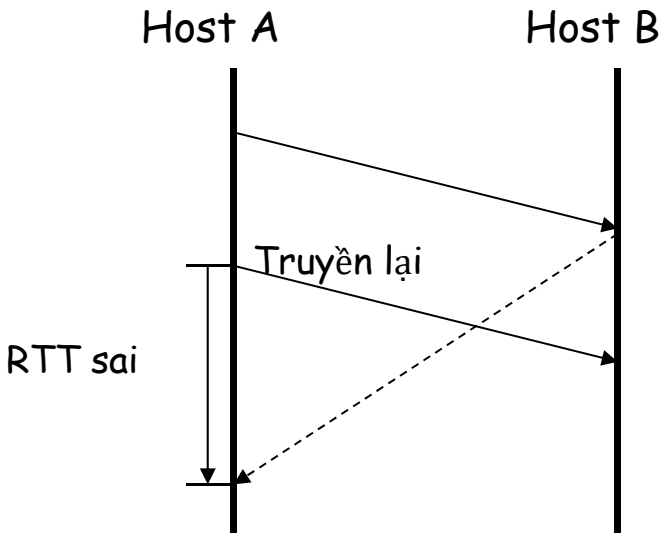
Không ổn định

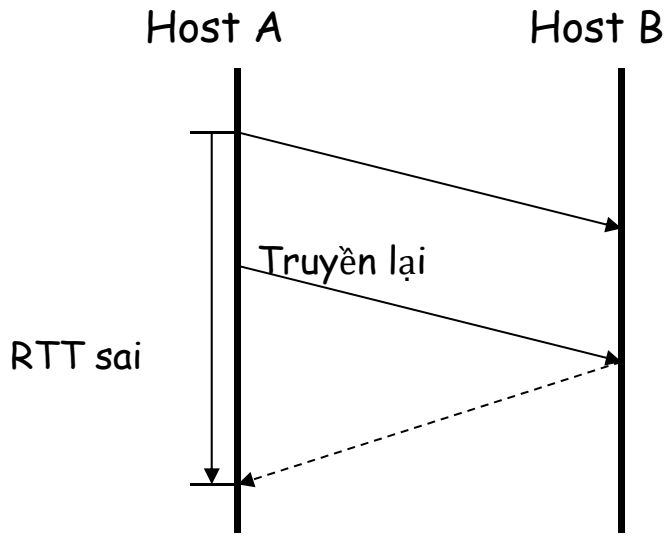
VÍ DỤ

- Một kết nối TCP có RTT hiện tại là 30 ms, nhận được các bản tin ACK đến sau 26, 32, và 24 ms. Tính ước lượng giá trị RTT theo thuật toán Jacobson với $\alpha = 0.9$.

Cơ chế ước lượng RTO

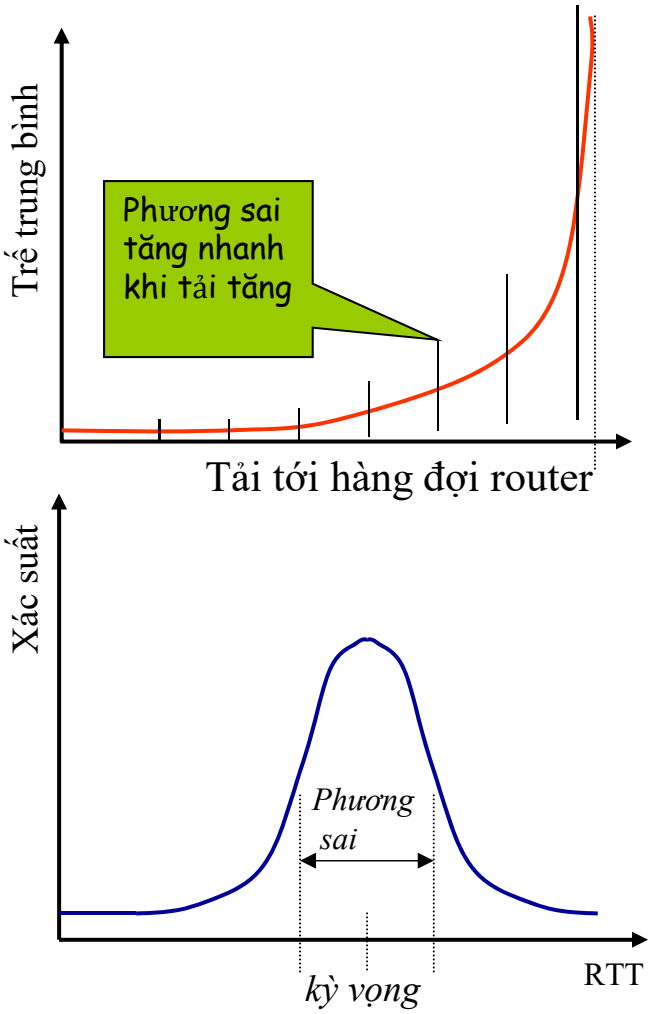
- Thuật toán Karn:
  - Vấn đề: Khi truyền lại thì RTT được ước lượng thế nào cho chính xác?
  - Giải pháp:
    - Không cập nhật giá trị EstimatedRTT
    - $RTO_k = 2 * RTO_{k-1}$





Cơ chế ước lượng RTO

- Trong thuật toán trên:
  - Giả thiết: Phương sai của giá trị RTT là hằng số
  - Trong thực tế:
    - Chiều dài hàng đợi của router tăng dần khi tải tăng
    - Phương sai của RTT tăng nhanh khi tải tăng
    - Phân bố xác suất của RTT không xác định
- Yêu cầu:
  - Ước lượng RTT chính xác để tránh ước lượng sai RTO



Cơ chế ước lượng RTO

- Thuật toán ước lượng RTO mới:
  - Cho phép đánh giá cả phương sai của RTT

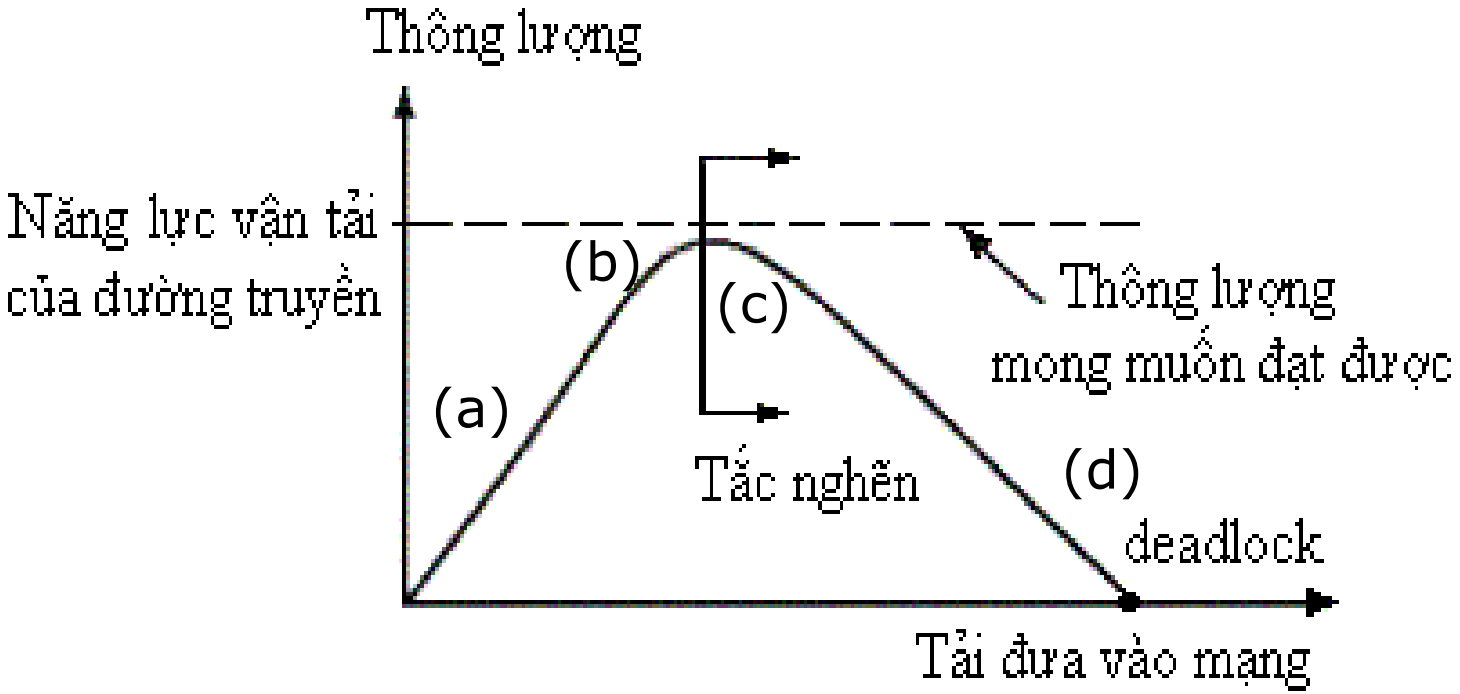
1. $EstimatedRTT_k = \alpha\, EstimatedRTT_{k-1} + (1 - \alpha)\, SampleRTT$
2. $Difference_k = (1 - \delta)*Difference_{k-1} + \delta*|SampleRTT - EstimatedRTT_k|$
3. $RTO = \mu * EstimatedRTT_k + \phi*Difference_k$

   $\alpha \approx 0{,}125;\ \mu \approx 1;\ \phi \approx 4;\ \delta \approx 0{,}25$

phức tạp, k đề cập

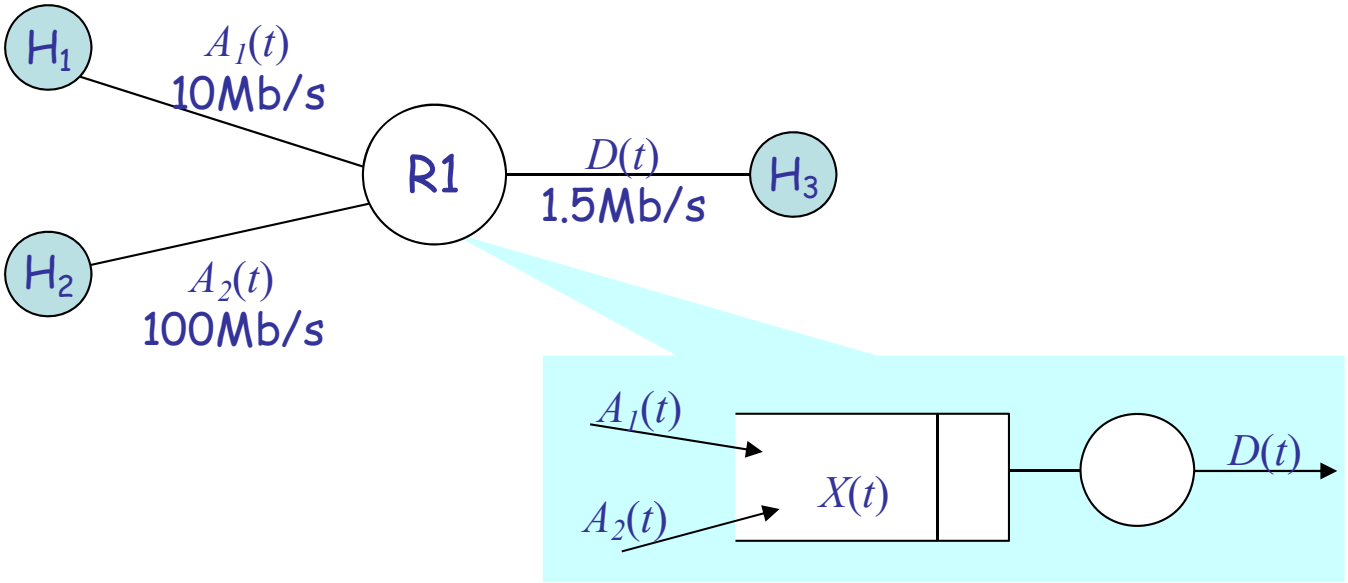
Chống tắc nghẽn (Congestion Control)

- Hiện tượng tắc nghẽn



Nguyên nhân gây tắc nghẽn mạng

- Tràn bộ đệm
- Tốc độ xử lý chậm (hay nghẽn cổ chai)



Nhận xét:

Nếu bộ đệm rỗng → trễ nhỏ, mạng không bị tắc nghẽn, nhưng hiệu suất sử dụng kênh thấp

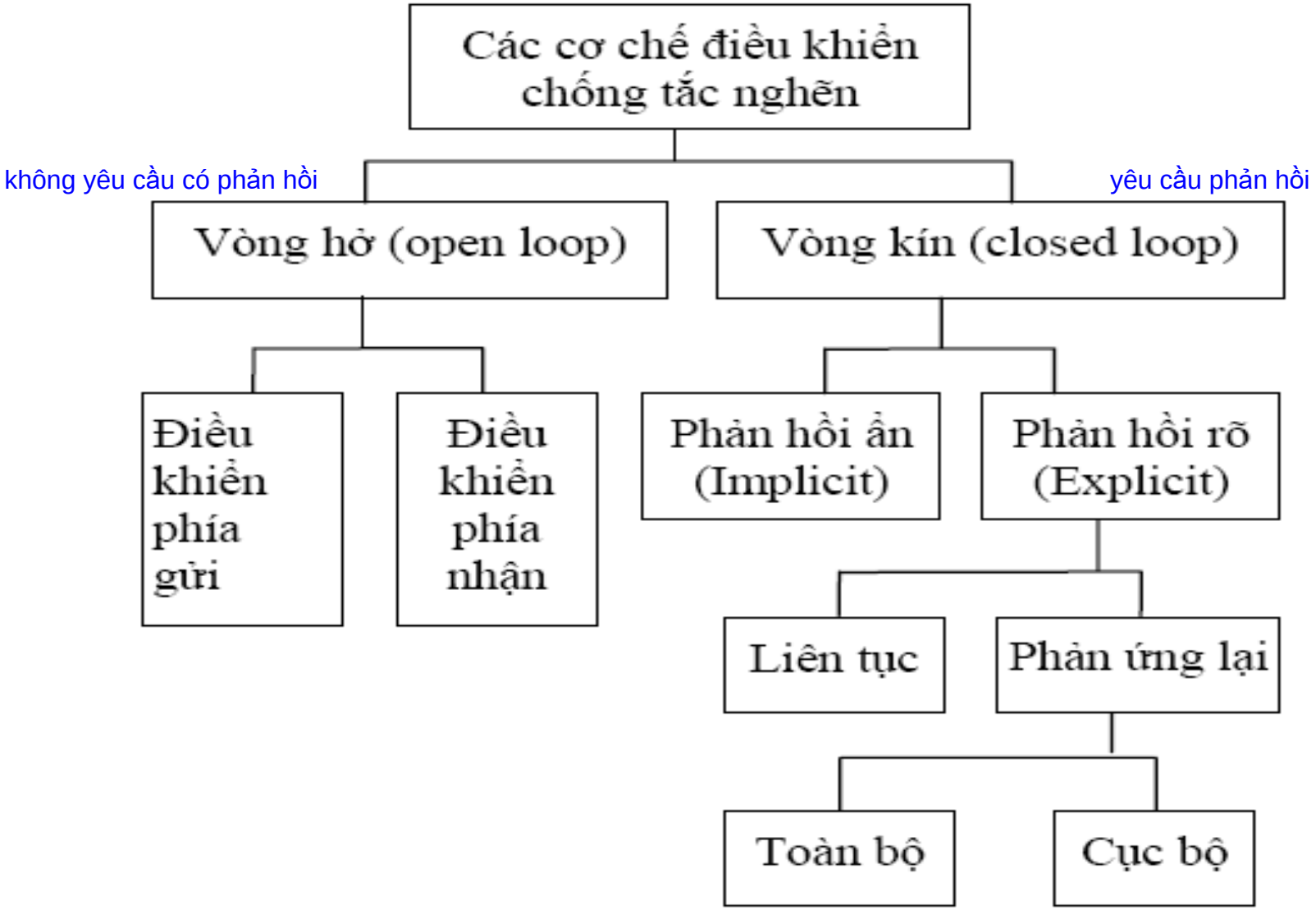
Nếu bộ đệm luôn đầy → trễ lớn, có thể tắc nghẽn, nhưng hiệu suất sử dụng kênh cao

giải pháp tăng bộ đệm làm tăng thời gian gói trong bộ đệm

Các giải pháp chống tắc nghẽn

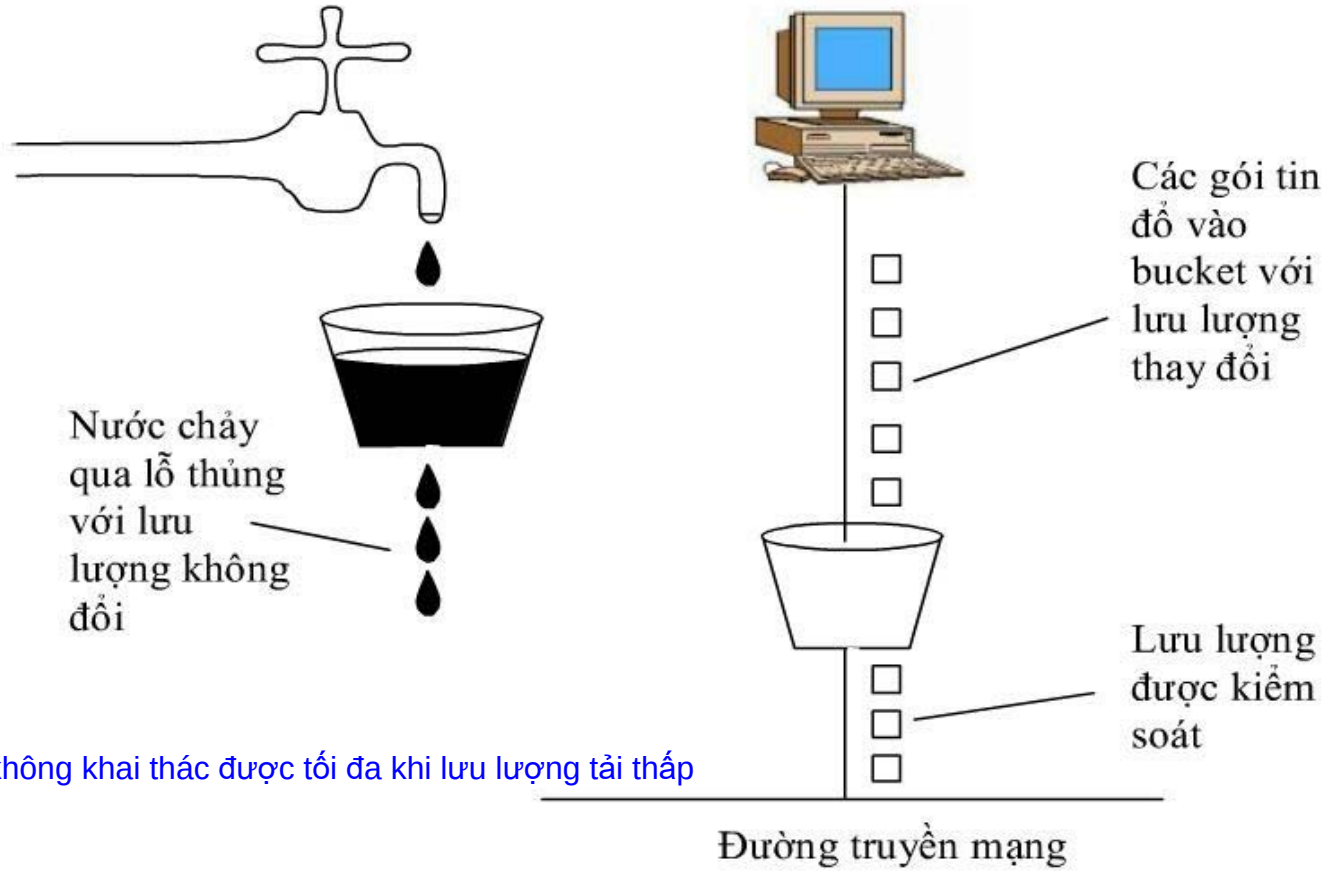
- Chính sách chung
  - Điều khiển tiếp nhận (Admission control)
    - Cho phép một kết nối mới chỉ khi mạng có thể đáp ứng một cách thích hợp mở kết nối chỉ khi mạng còn tài nguyên
  - Kiểm soát (Policing)
  - Điều khiển luồng lưu lượng (Flow control)
    - Yêu cầu giảm tốc độ luồng phát

Phân loại


Các cơ chế điều khiển chống tắc nghẽn
không yêu cầu có phản hồi
yêu cầu phản hồi
Vòng hở (open loop)
Vòng kín (closed loop)
Điều khiển phía gửi
Điều khiển phía nhận
Phản hồi ẩn (Implicit)
Phản hồi rõ (Explicit)
Liên tục
Phản ứng lại
Toàn bộ
Cục bộ

Open-loop congestion control

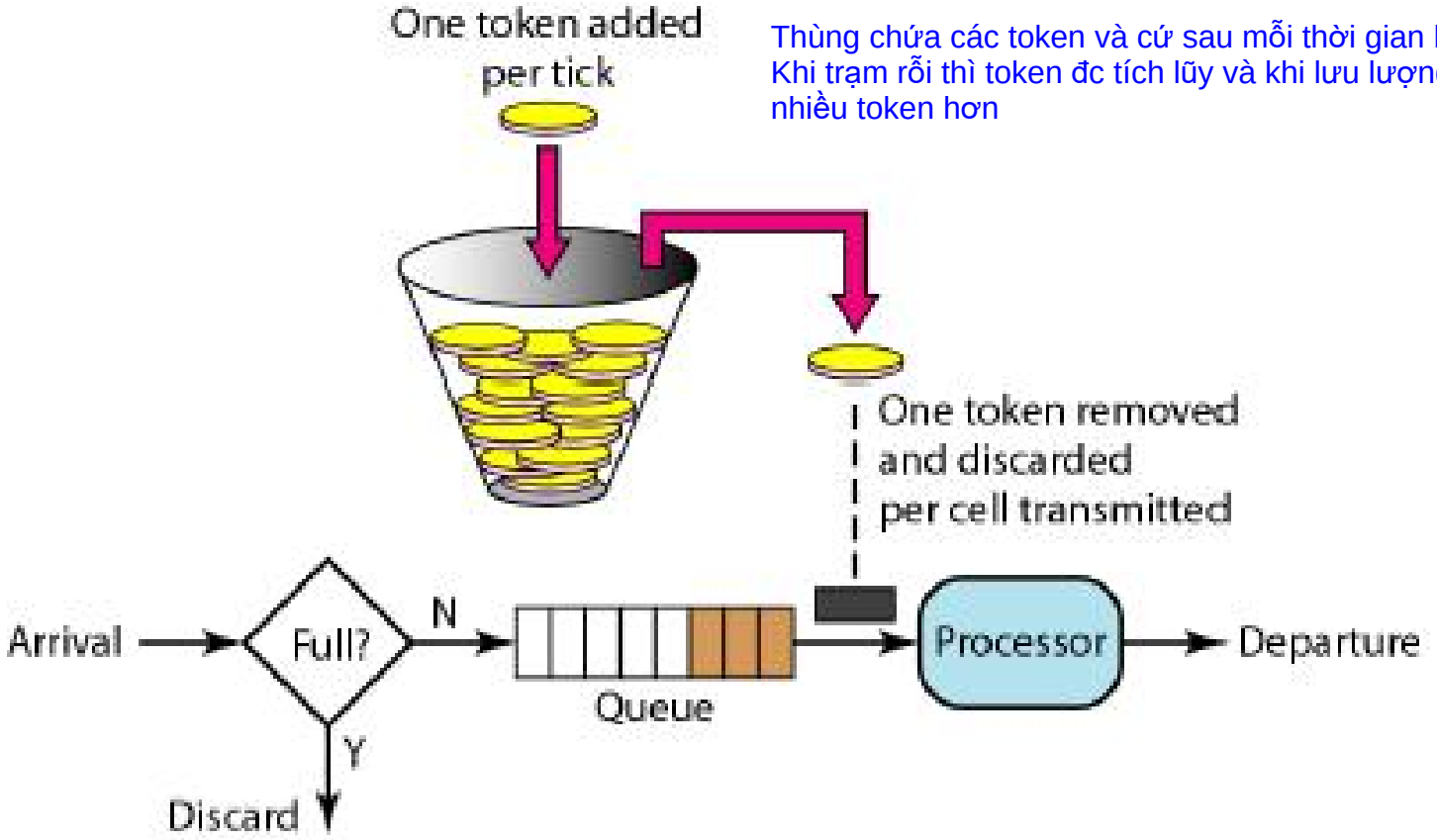
- Thuật toán thùng rò (Leaky Bucket)



Nhược: không khai thác được tối đa khi lưu lượng tải thấp

Open-loop congestion control

- Thuật toán Token Bucket



Thùng chứa các token và cứ sau mỗi thời gian lại sinh ra token. Khi trạm rỗi thì token đc tích lũy và khi lưu lượng lớn có thể sử dụng nhiều token hơn

Close-loop congestion control

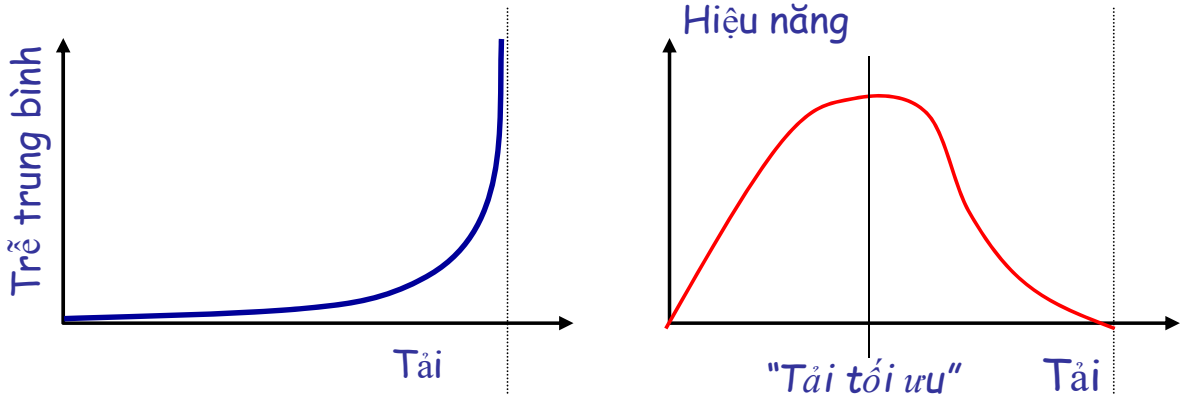
- Phản hồi ẩn (implicit feedback):
  - bên phát sử dụng time-out để xác định liệu có xảy ra tắc nghẽn hay không, ví dụ TCP
- Phản hồi hiện (explixit feedback):
  - một số bản tin tường minh được gửi đến nguồn phát
- Điều khiển theo tốc độ
  - điều khiển một cách trực tiếp tốc độ truyền tại phía gửi (nguồn gửi tin)
- Điều khiển theo kích thước cửa sổ
  - điều khiển gián tiếp tốc độ truyền thông qua việc thay đổi kích thước cửa sổ

Close-loop congestion control

- Các bước điều khiển vòng đóng
  - Bước một: theo dõi hệ thống để phát hiện tắc nghẽn xảy ra khi nào và ở đâu
  - Bước hai: nơi phát hiện ra tắc nghẽn cần phải chuyển thông tin về sự tắc nghẽn đến những nơi có thể phản ứng lại lại tăng lưu lượng trong mạng
  - Bước ba: điều chỉnh lại hệ thống để sửa chữa sự cố

Chống tắc nghẽn trong TCP

- Nguyên tắc phát hiện nghẽn
  - Timeout
  - 3dupack 3 ack giống nhau
- Hoạt động



Window=min{ReceiveWindow, CongestionWindow}

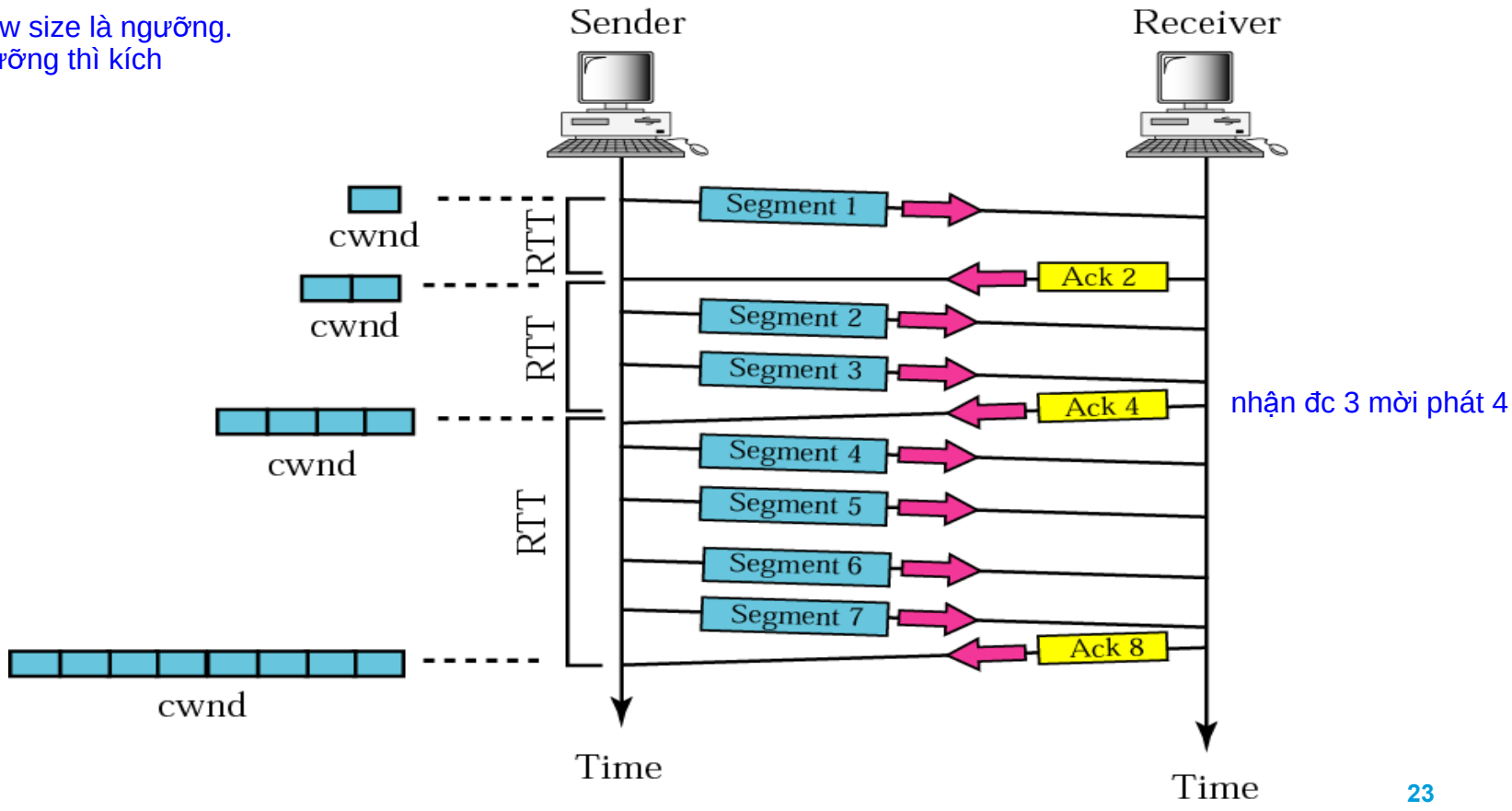
Rwnd bên thu

Cwnd bên phát

Slow Start

- Giai đoạn khởi đầu chậm SS (Slow Start) tăng nhanh tốc độ gửi theo hàm mũ
  - Cwnd := 1 MSS (Maximum Segment Size) cửa sổ chống tắc nghẽn
  - Nhận được ACK→ Cwnd:=Cwnd * 2 mỗi khi nhận đc ACK thì kích thước cửa sổ nhân 2

reserved window size là ngưỡng.
Sau khi đạt ngưỡng thì kích

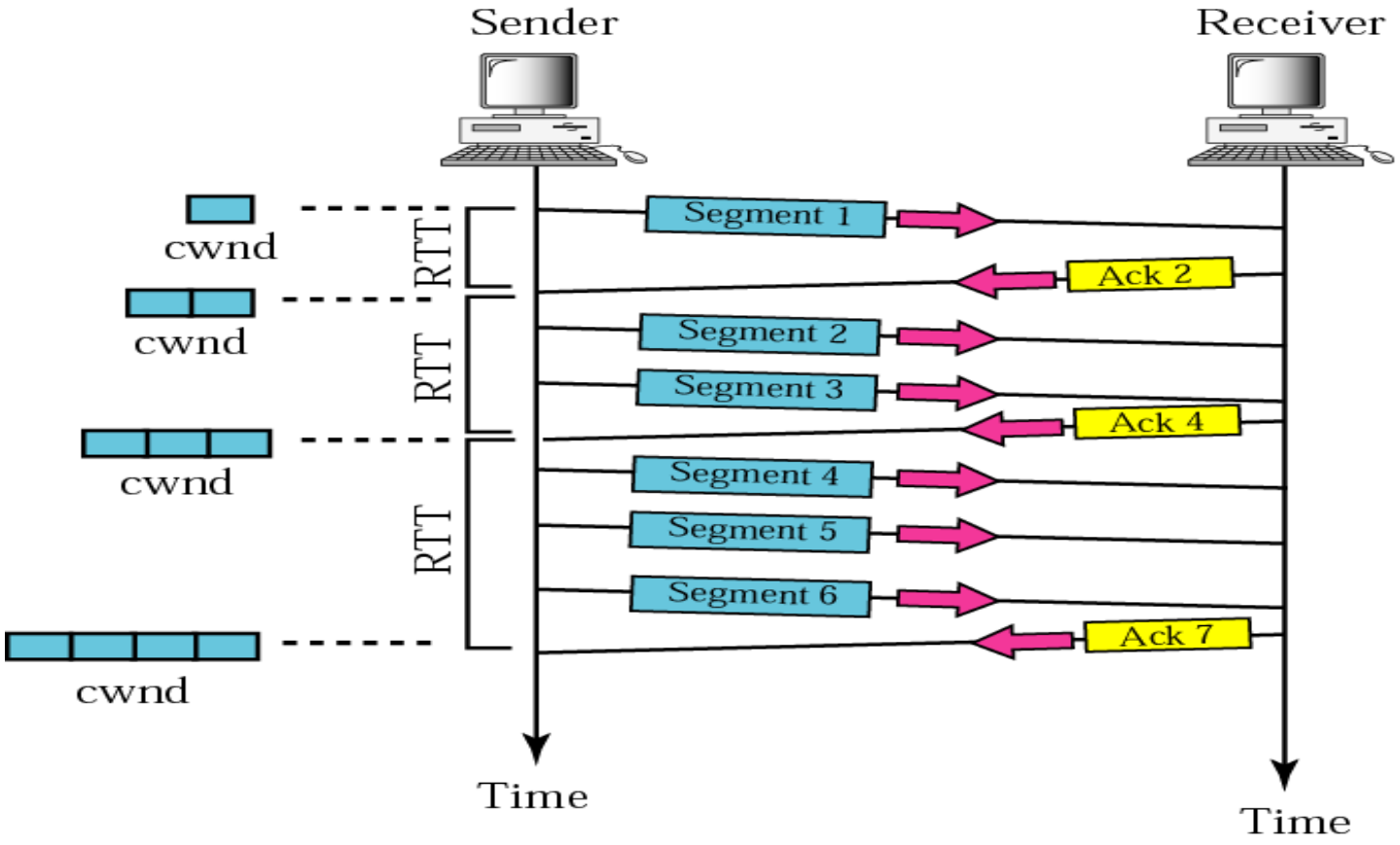


nhận đc 3 mời phát 4

Congestion Avoidance

- Giai đoạn chống tắc nghẽn CA (Congestion Avoidance)
  - Cwnd := Cwnd +1 sau mỗi RTT không có lỗi cho đến khi có nghẽn

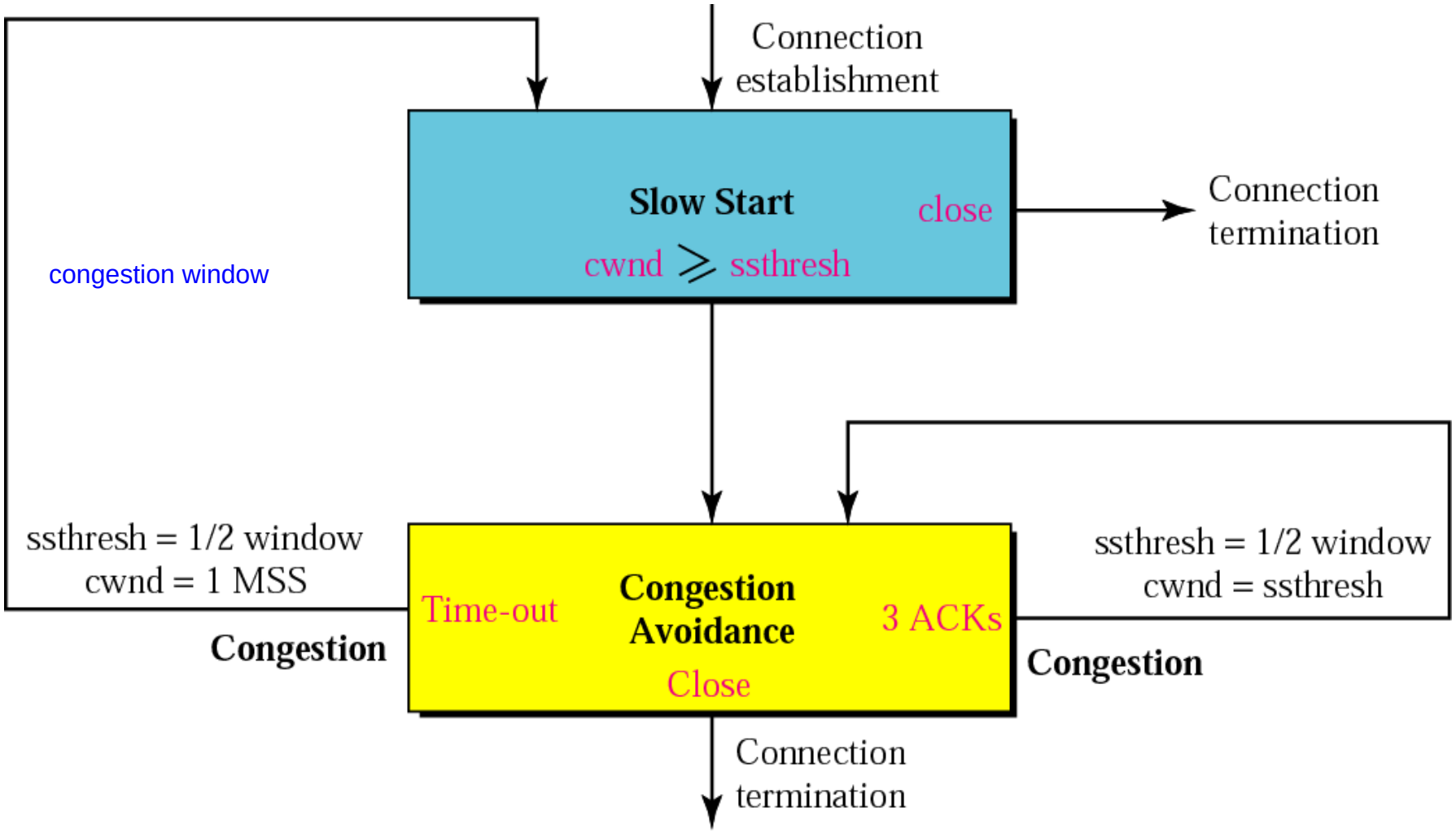
    Sau khi đạt ngưỡng thì kích thước tăng theo cấp số cộng



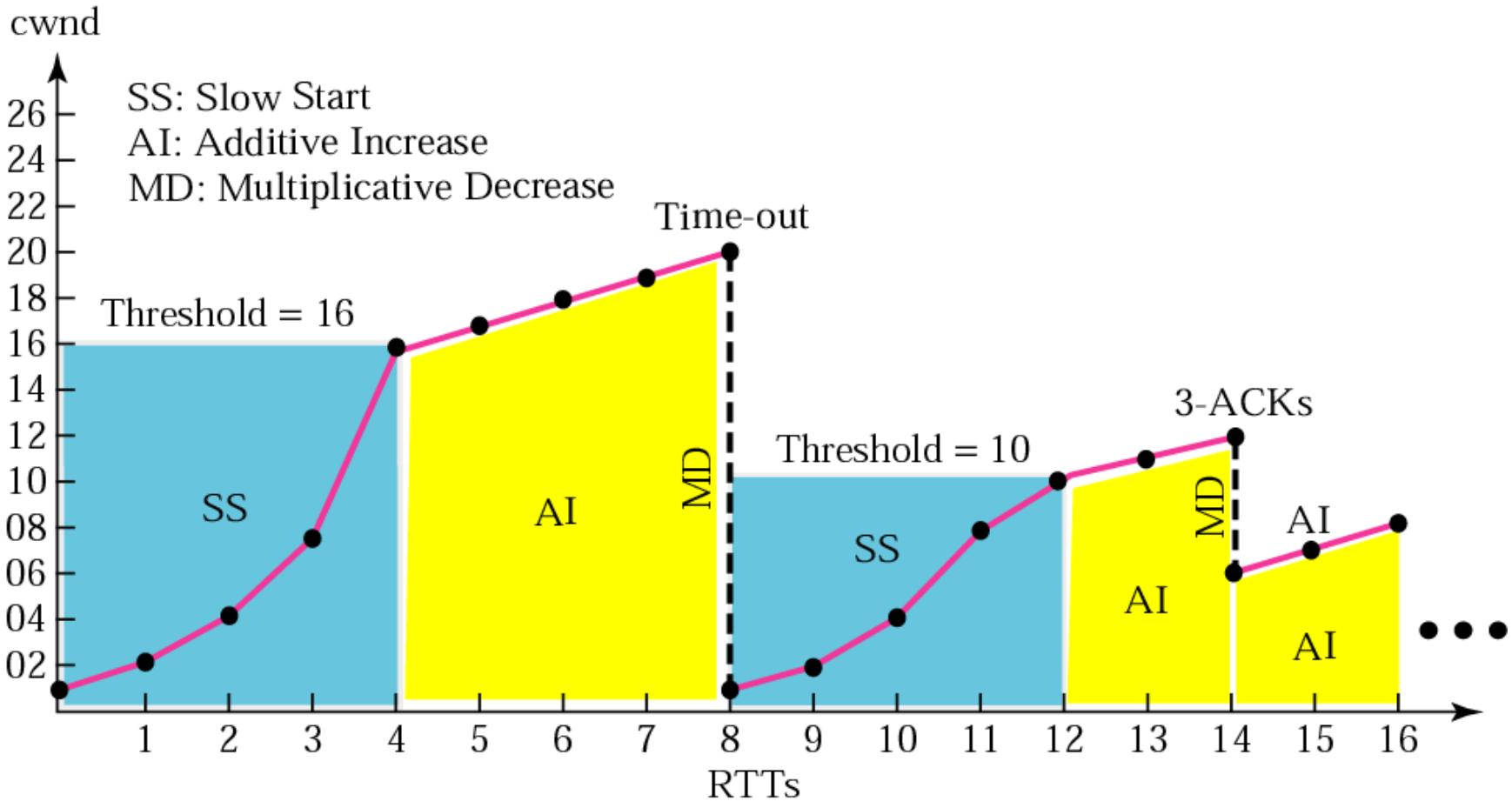
Congestion Avoidance

- Dấu hiệu tắc nghẽn:
  - RTT > Timeout mà không nhận được ACK Tắc nghẽn ở mức nghiêm trọng
    - ssthresh := Cwnd/2 (giảm theo cấp số nhân) ngưỡng chia đôi
    - Cwnd := 1 cửa số =1
    - TCP chuyển về trạng thái slow start (SS) quay về ban đầu
  - Nhận được 3 Ack (báo nhận lặp) Tắc nghẽn chưa nghiêm trọng
    - Đặt ngưỡng ssthresh xuống còn một nửa giá trị hiện tại của Cwnd: ssthresh := cwnd/2
    - Đặt Cwnd bằng ½ giá trị hiện tại: Cwnd:= Cwnd/2
    - TCP quay lại trạng thái chống tắc nghẽn (CA)

TCP Congesttion control

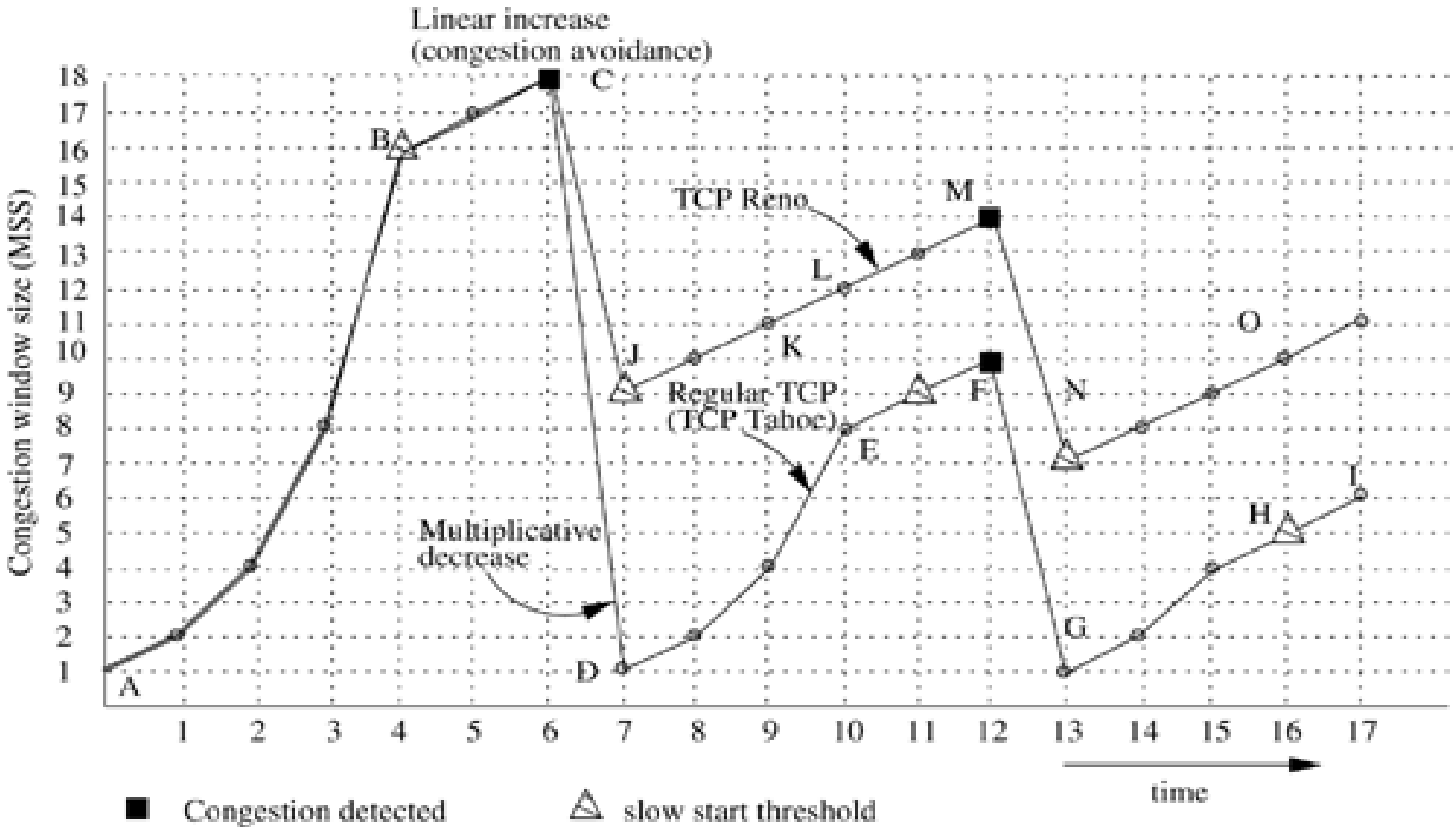


Ví dụ


cwnd
SS: Slow Start
AI: Additive Increase
MD: Multiplicative Decrease
Time-out
Threshold = 16
Threshold = 10
3-ACKs
SS
AI
MD
SS
AI
MD
AI
AI
26
24
22
20
18
16
14
12
10
08
06
04
02
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
RTTs

TCP Tahoe and Reno

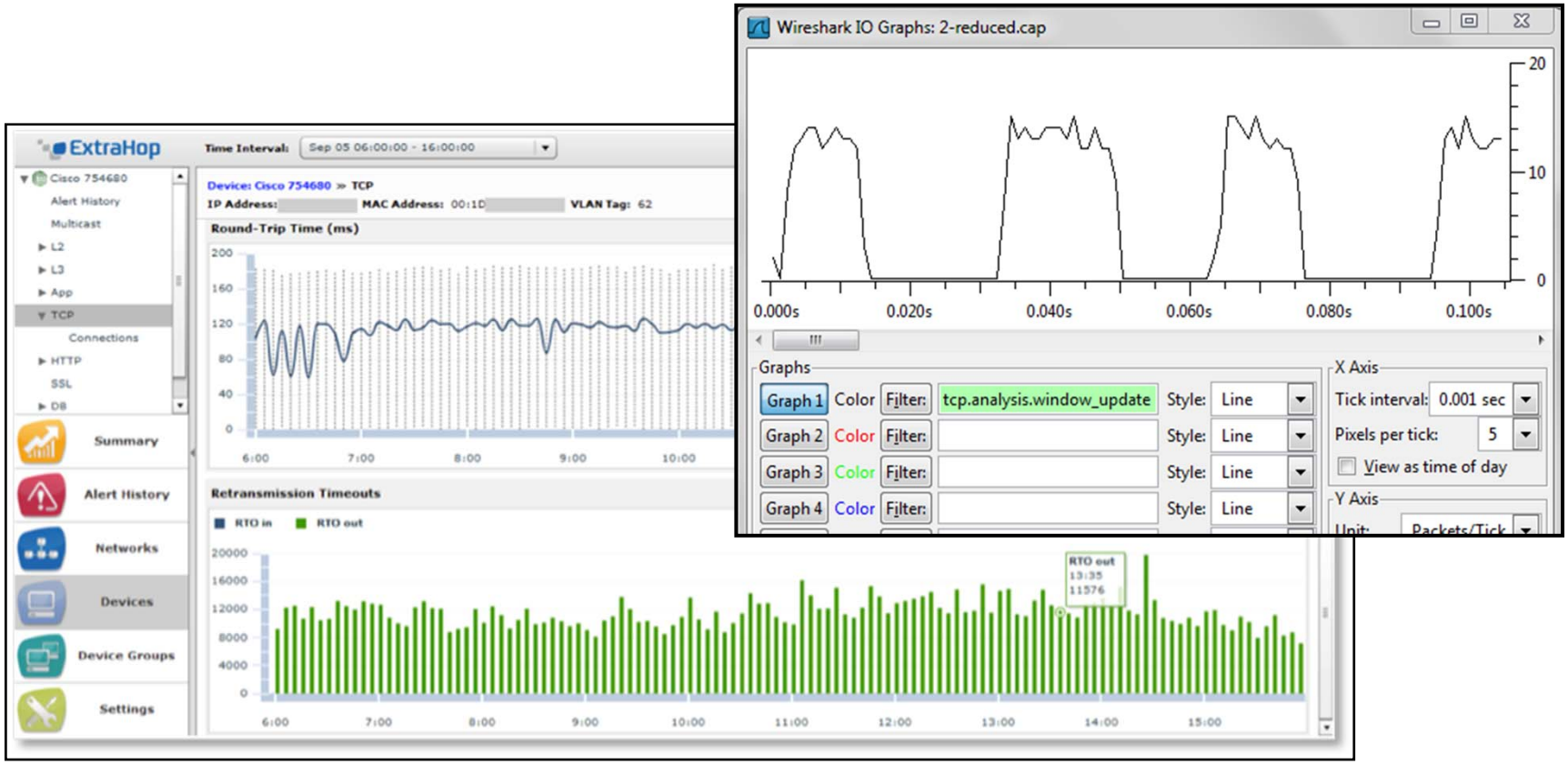
cải tiến để đường đặc tuyến bớt răng cưa



REno: 3ack trùng nhau thì giảm kích thước không nhỏ hơn 3 gói tin và thay vì tăng 1 thì tăng 3

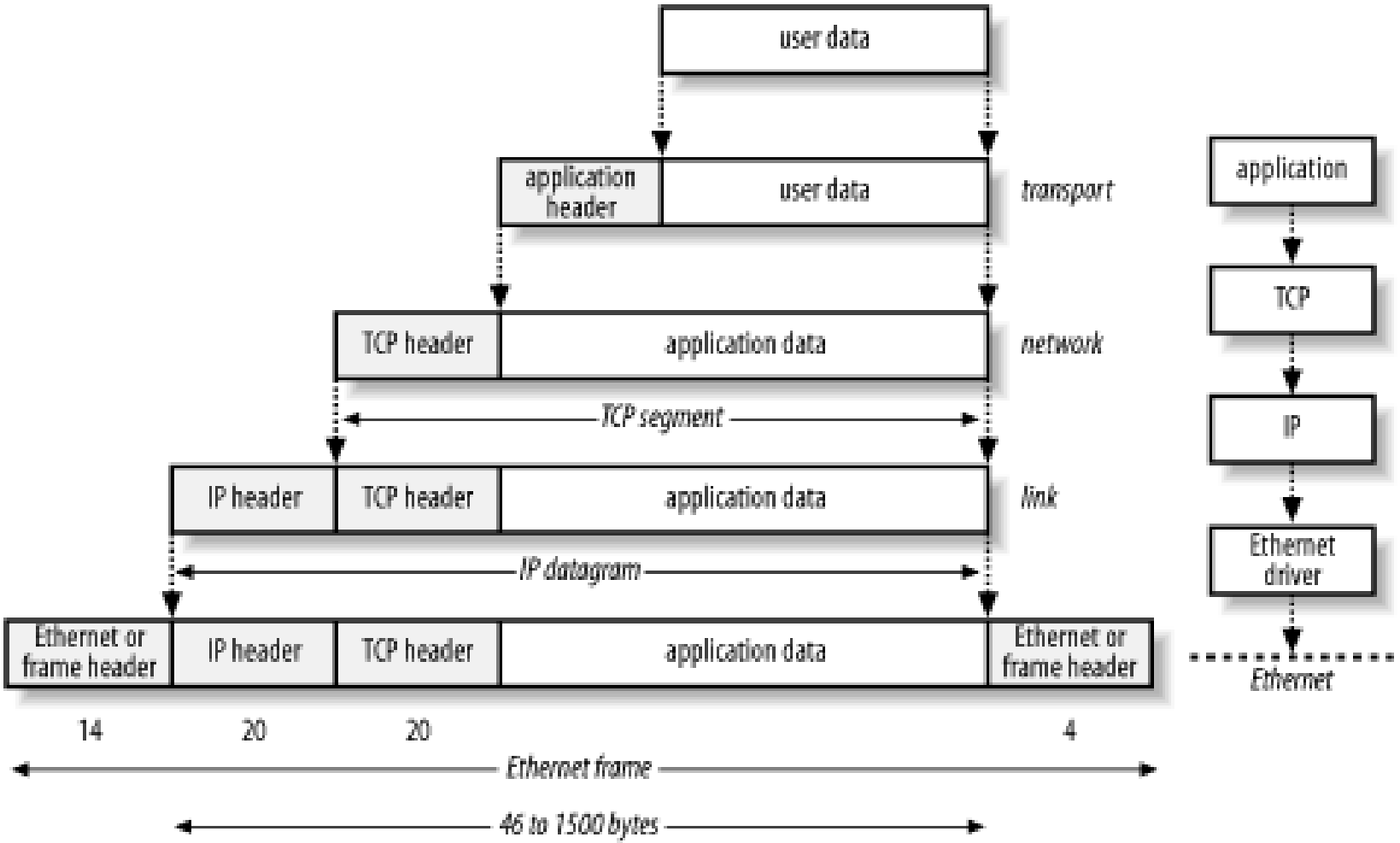
TCP Performance problems

- So sánh TCP trên Windows với MacOSX
  - Hệ thống WIFI
  - Data message: 100KB



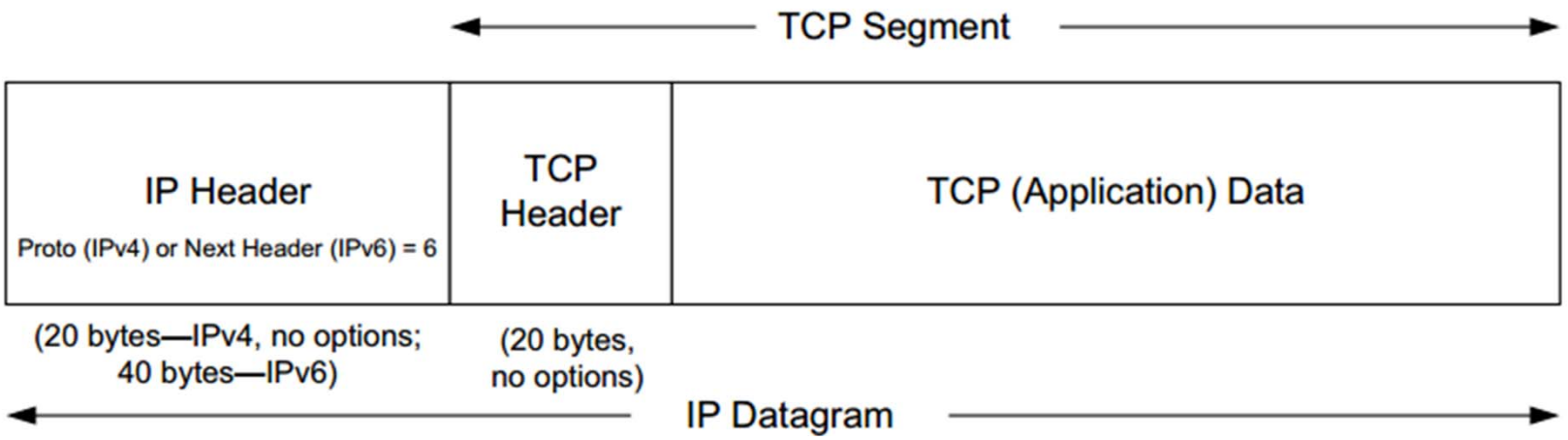
TCP Performance problems

- TCP/IP Data Encapsulation



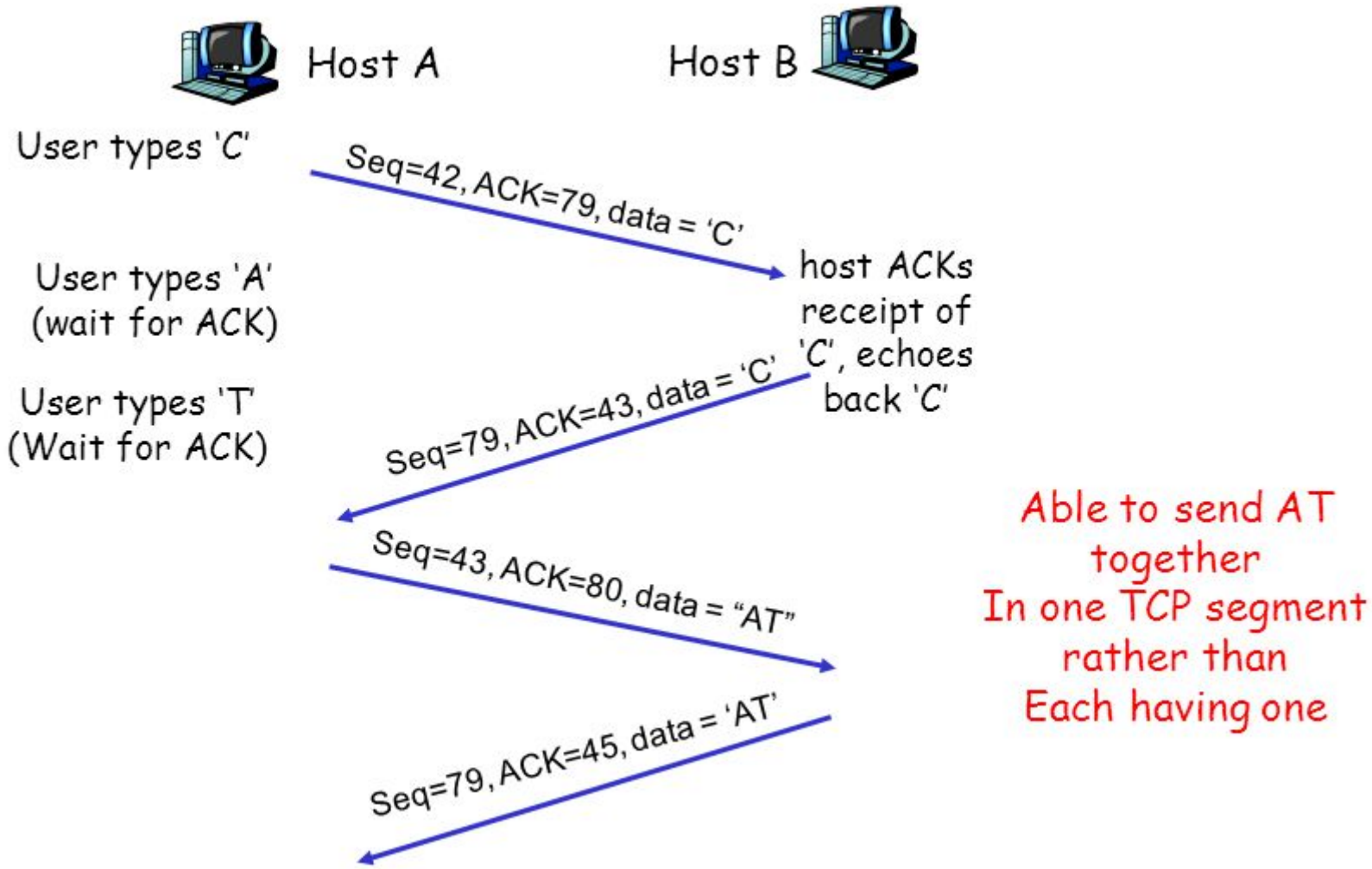
TCP Performance problems

- TCP Segment
  - MSS: 576 – 40 = 536 (mặc định, X25)
  - MSS: 1500-40 = 1460 (Ethernet)



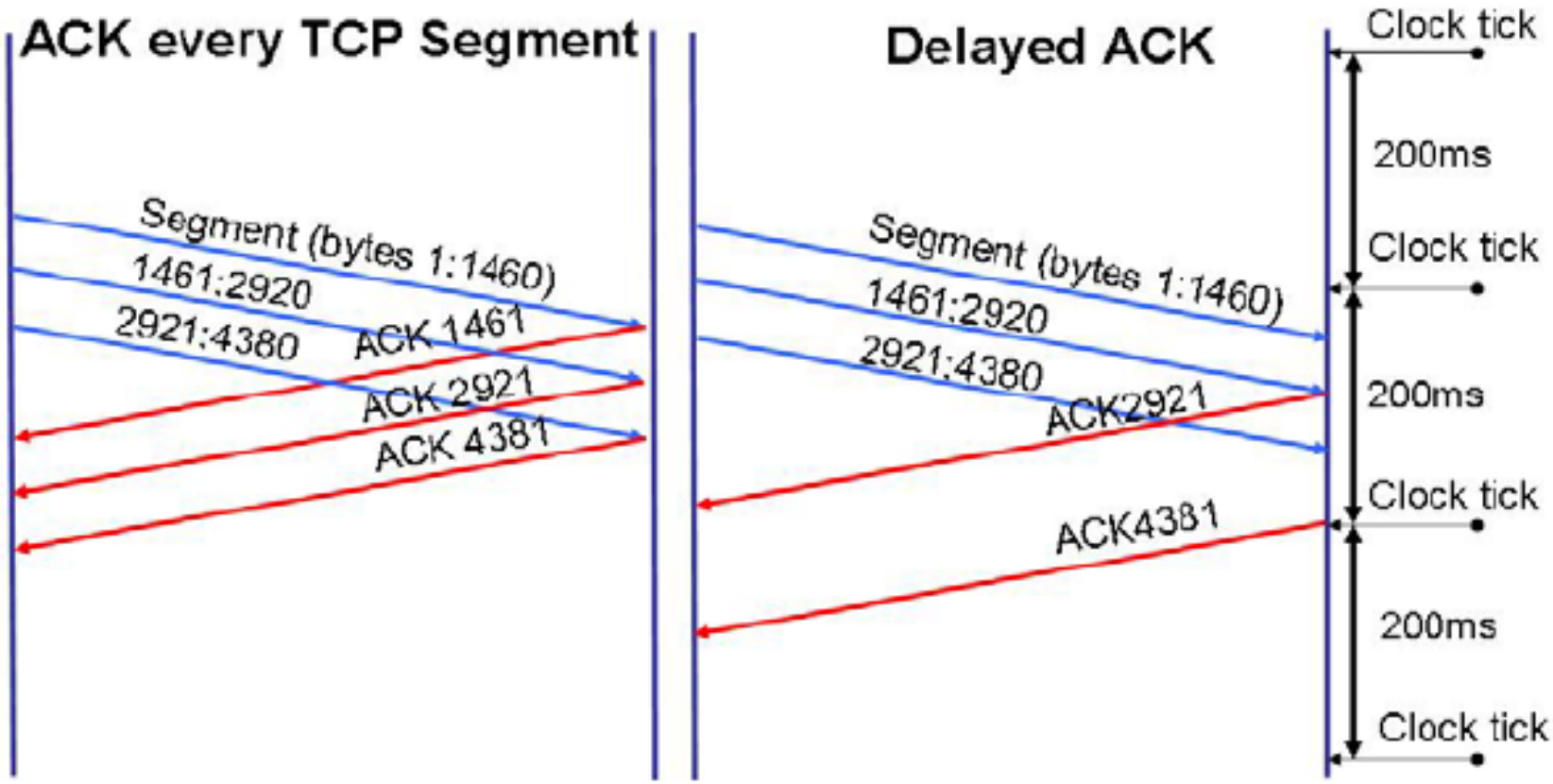
TCP Performance problems

- Nagle's Algorithm



TCP Performance problems

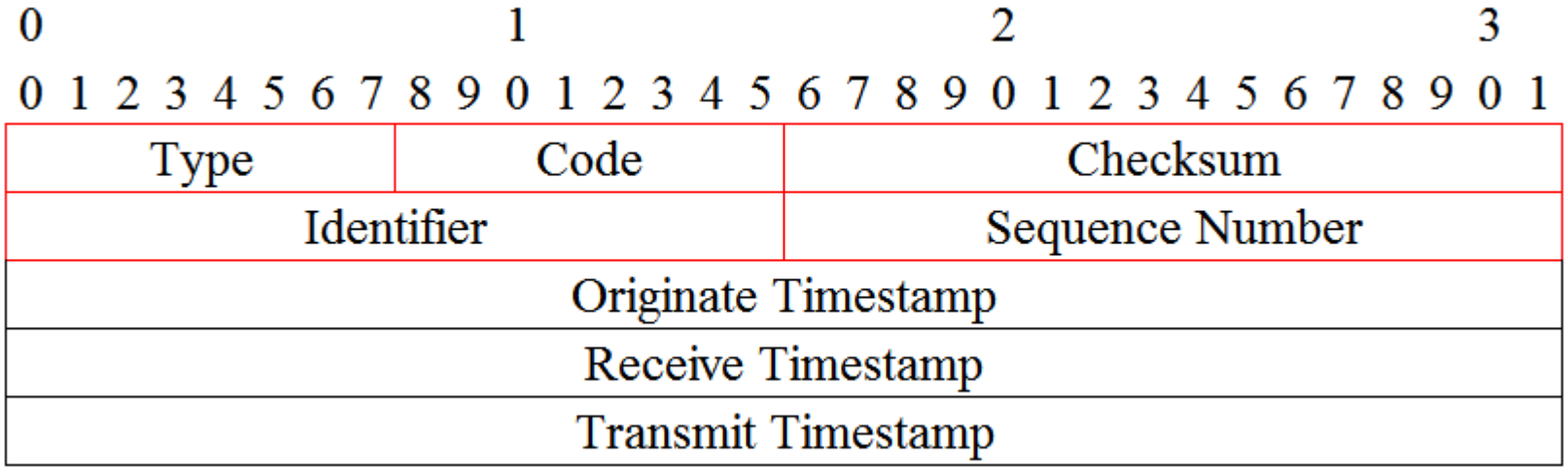
- Delayed ACK (hồi âm muộn)



Thêm thời gian chờ để hồi âm cho nhiều gói tin

TCP Performance problems

- Timestamp (option)
  - Timestamp = 12 byte
  - MSS = 1500 – 52 = 1448 byte

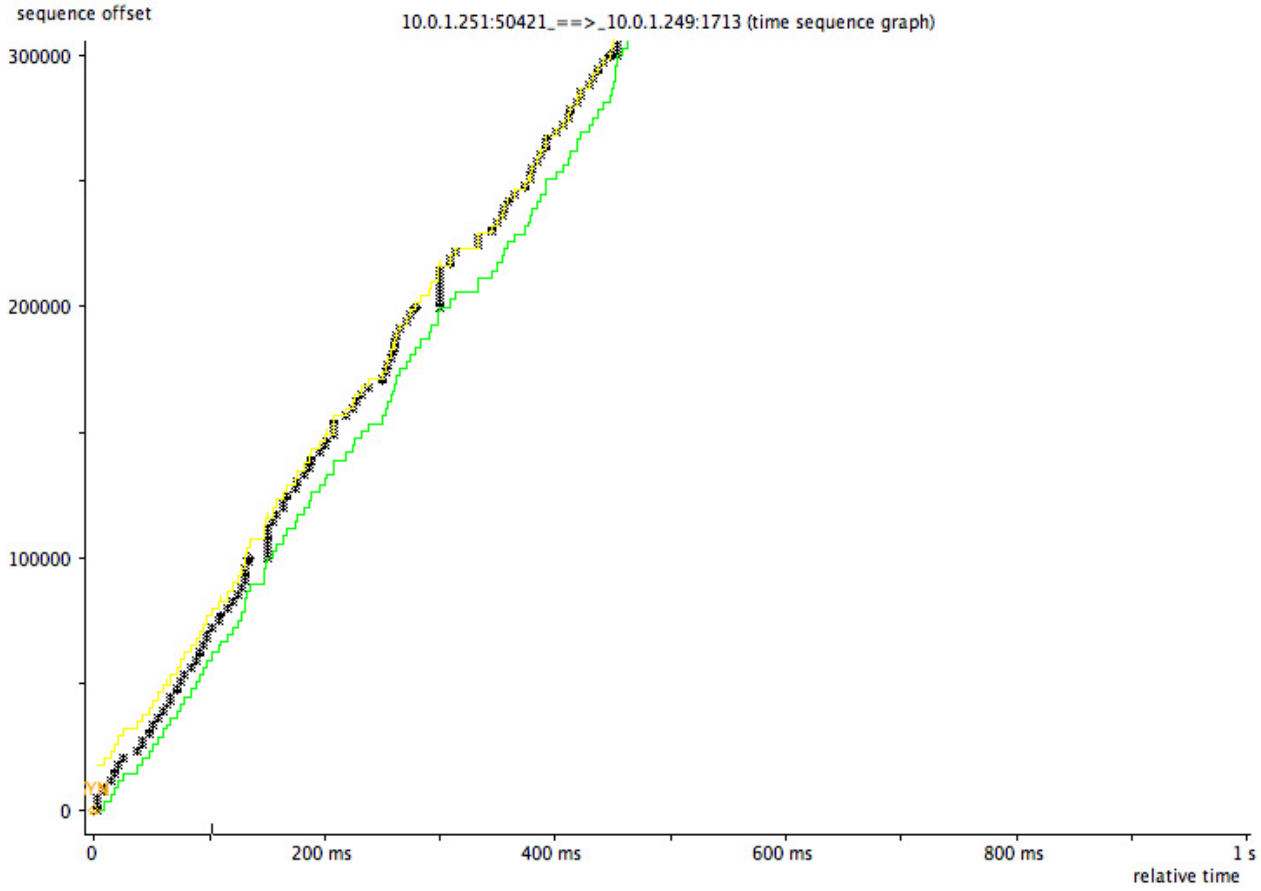


TCP Performance problems

- Windows
  - 68 mẩu x 1460 bytes + mẩu 720 bytes (no timestamp)
  - Bên gửi gửi đi 68 mẩu (đầy MSS nên gửi ngay), và giữ lại mẩu cuối (720 bytes) chờ hồi âm của mẩu 68
  - Bên nhận dùng luật hồi âm kép nên sẽ hồi âm cho các mẩu 2, 4, …, 68 ngay lập tức, và cuối cùng khi bên gửi nhận được hồi âm của mẩu 68 sẽ gửi ngay mẩu lẻ cuối cùng
- MacOSX
  - 69 mẩu 1448 bytes + mẩu 88 bytes (có timestamp)
  - Bên gửi gửi đi 69 mẩu (đầy MSS nên gửi ngay), và giữ lại mẩu cuối (88 bytes) chờ hồi âm mẩu 69
  - Bên nhận theo luật hồi âm kép, nên sẽ hồi âm cho các mẩu 2, 4, …, 68 ngay lập tức, nhưng giữ lại không hồi âm mẩu 69 ngay
- Kết quả:
  - Cứ mỗi 100KB thì MacOSX bị chậm lại 200ms!

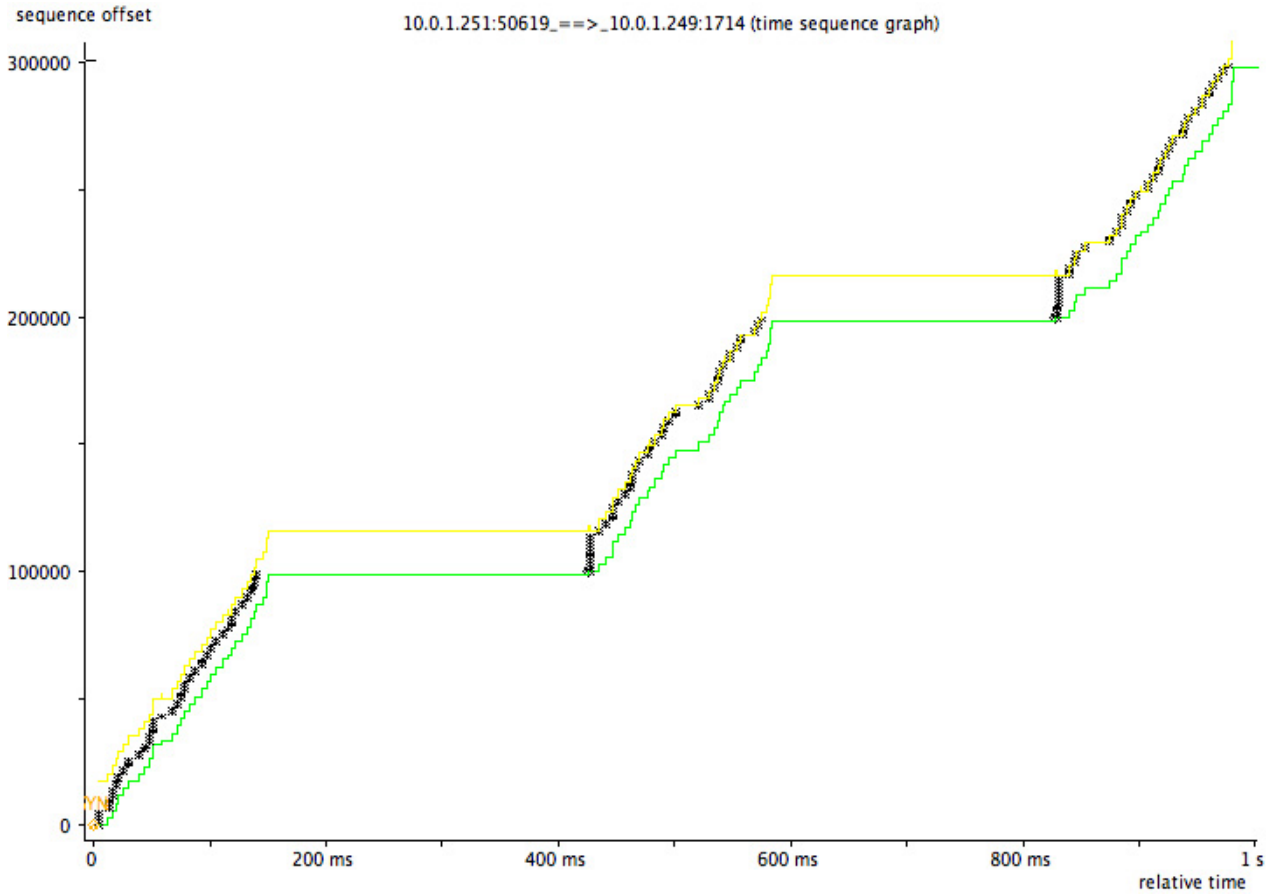
TCP Performance problems

- Windows



TCP Performance problems

- MacOSX



đạt được tối ưu chậm hơn windows

BÀI TẬP

- Bài 1:

  Xét tính hiệu quả của việc xử dụng cơ chế SS với một liên kết có RTT=10ms và không có tắc nghẽn. Cho biết Rwnd = 24 KB và MSS = 2 KB. Cần bao nhiêu thời gian trước khi cửa sổ thu đầy đủ có thể được gửi đi?

- Bài 2:

  Giả sử một kết nối TCP sử dụng cửa sổ tắc nghẽn Cwnd = 18 KB thì sảy ra timeout. Tính kích thước cửa sổ nếu 4 lần truyền sau đó đều thành công. Giải thiết kích thước segment tối đa là 1 KB

- Bài 3:

  A TCP machine is sending full windows of 65,535 bytes over a 1-Gbps channel that has a 10-msec one-way delay. What is the maximum throughput achievable? What is the line efficiency?