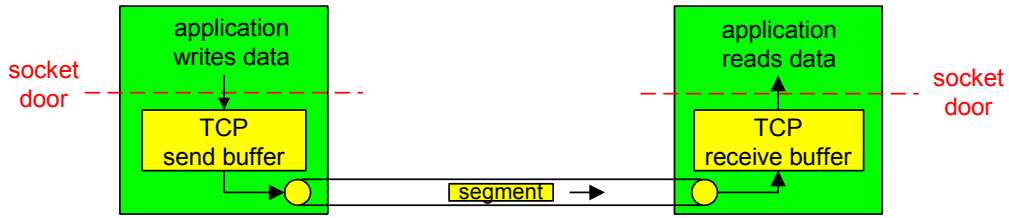
TCP: Tổng quan

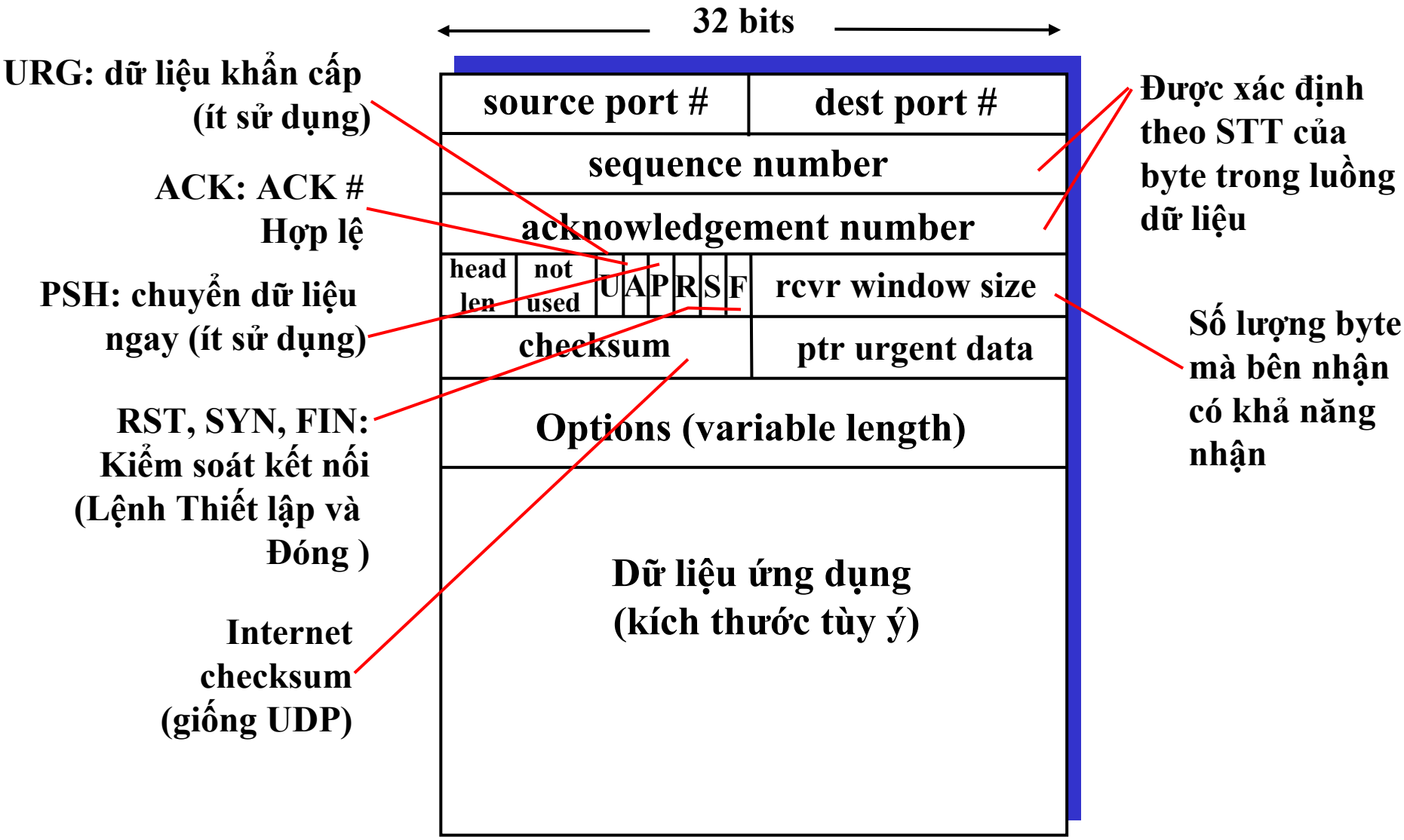
RFCs: 793, 1122, 1323, 2018, 2581

- Điểm nối điểm:
  - Một gửi, Một nhận
- Tin cậy, theo đúng thứ tự:
  - Không quan tâm đến khuôn dạng thông điệp.
- Đường ống:
  - Cửa sổ kiểm soát tắc nghẽn và điều khiển lưu lượng.
- *Bộ đệm ở phía Nhận và Gửi*



- Truyền song công:
  - Dữ liệu truyền theo cả hai hướng
  - MSS: Kích thước tối đa một segment
- Hướng nối:
  - Bắt tay, chào hỏi trước khi nói chuyện (trao đổi thông tin điều khiển). Thiết lập bộ đệm hai đầu.
- Kiểm soát lưu lượng:
  - Nói quá nhanh, nghe quá chậm

Cấu trúc TCP segment



TCP: Số thứ tự và Số biên nhận

Số thứ tự (STT):

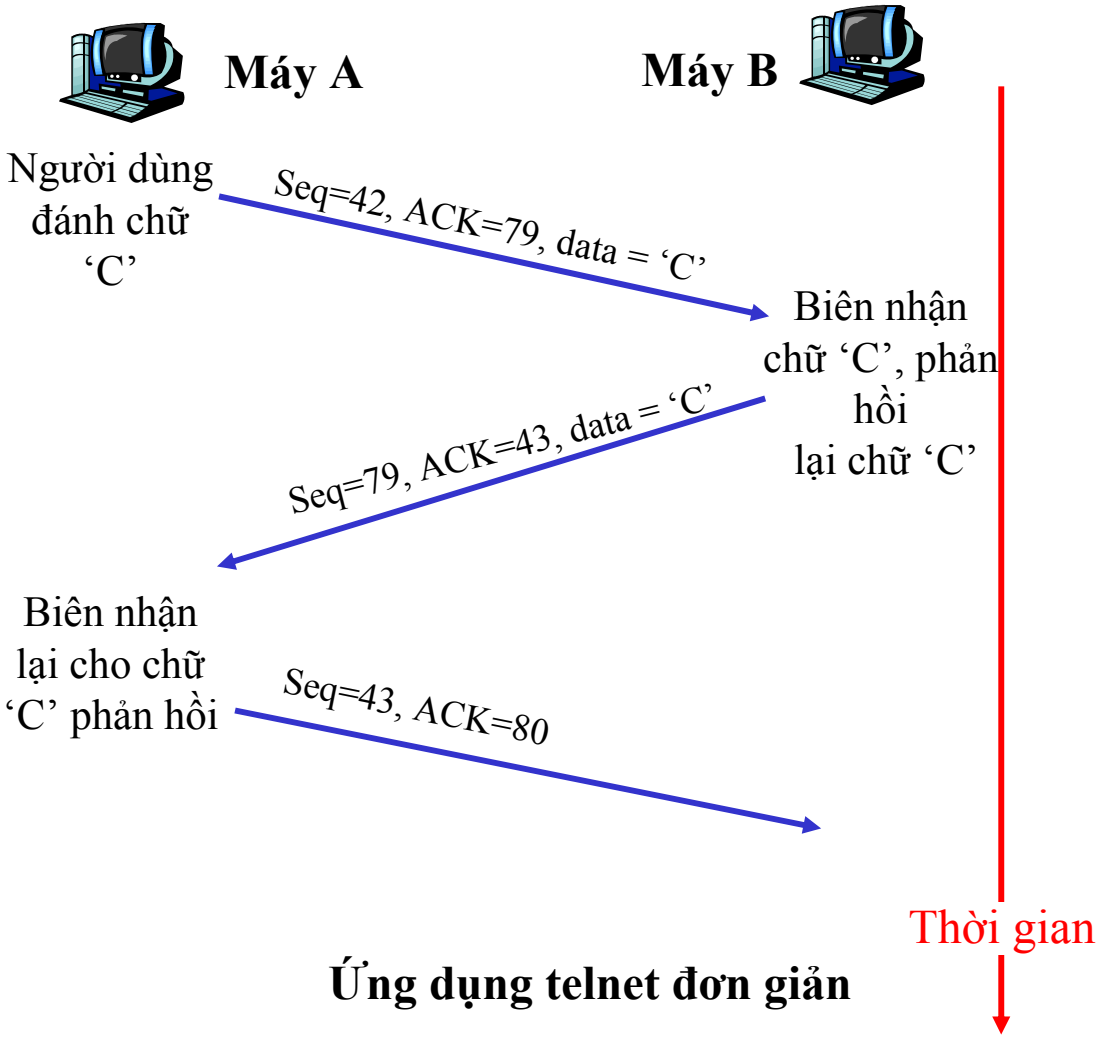
- Là Số thứ tự của byte đầu tiên trong luồng dữ liệu

Số biên nhận:

- Là Số thứ tự của byte kế tiếp mà bên nhận muốn nhận.
- Biên nhận tích lũy

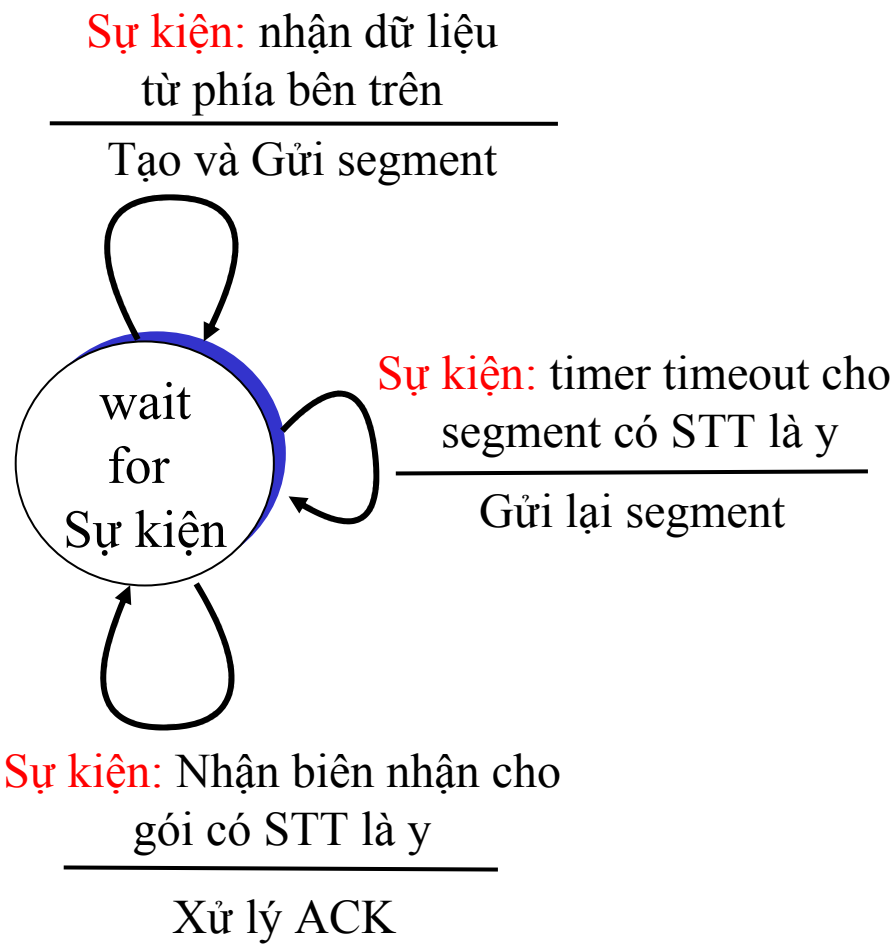
Q? Bên nhận xử lý gói tin không đúng thứ tự ntn ?

- **A**: TCP không quy định. Tùy thuộc vào người cài đặt.



Ứng dụng telnet đơn giản

TCP: Truyền Tin cậy



FSM bên Gửi **đơn giản**, giả định rằng:

- Dữ liệu truyền theo một hướng
- Không kiểm soát tắc nghẽn
- Không điều khiển lưu lượng

Nhanh chóng truyền lại

- Khoảng thời gian Timeout thường tương đối dài:
  - Chậm trễ trong việc gửi lại gói tin bị mất
- Phát hiện mất gói tin qua các ACK trùng lặp
  - Phía gửi thường gửi nhiều gói tin
  - Nếu gói tin bị mất, sẽ có ACK trùng lặp
- Nếu phía gửi nhận được 3 ACK trùng lặp, có thể giả thiết gói tin ngay sau gói tin được biên nhận 3 lần liên tiếp bị mất:
  - Gửi lại kể cả khi gói này chưa timeout

Fast Retransmit:

```
event: ACK received, with ACK field value of y
            if (y > SendBase) {
                …
                SendBase = y
                if (there are currently not-yet-acknowledged segments)
                    start timer
                …
              }
            else {
                increment count of dup ACKs received for y
                if (count of dup ACKs received for y = 3) {
                    resend segment with sequence number y
                 …
```

ACK trùng lặp cho gói tin Đã được biên nhận

Truyền lại nhanh chóng

TCP: Truyền tin cậy

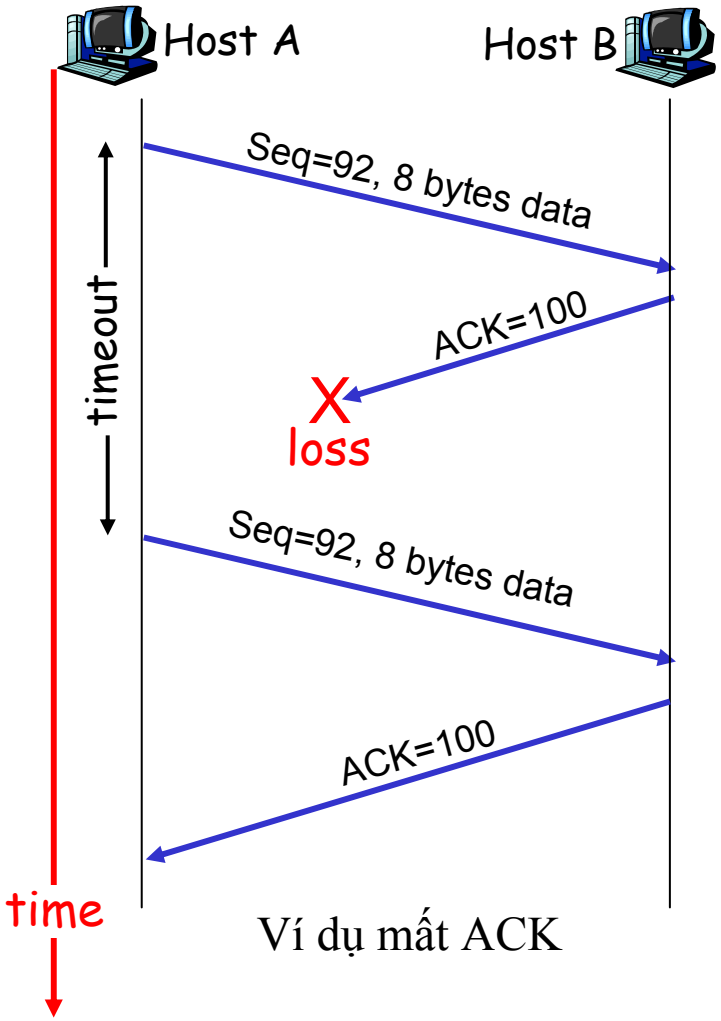
TCP phía Gửi đơn giản

```
00  sendbase = initial_sequence number agreed by TWH
01   nextseqnum = initial_sequence number by TWH
02    loop (forever) {
03      switch(event)
04      event: data received from application above
05                 if (window allow send)
06                    create TCP segment with sequence number nextseqnum
06                    if (no timer) start timer
07                    pass segment to IP
08                    nextseqnum = nextseqnum + length(data)
                   else put packet in buffer
09       event: timer timeout for sendbase
10            retransmit segment
11            compute new timeout interval
12            restart timer
13       event: ACK received, with ACK field value of y
14            if (y > sendbase) { /* cumulative ACK of all data up to y */
15                 cancel the timer for sendbase
16                 sendbase = y
17                 if (no timer and packet pending) start timer for new sendbase
17                 while (there are segments and window allow)
18                      sent a segment;
18            }
19            else { /* y==sendbase, duplicate ACK for already ACKed segment */
20                 increment number of duplicate ACKs received for y
21                 if (number of duplicate ACKS received for y == 3) {
22                      /* TCP fast retransmit */
23                    resend segment with sequence number y
24                    restart timer for segment y
25                 }
26       }  /* end of loop forever */
```

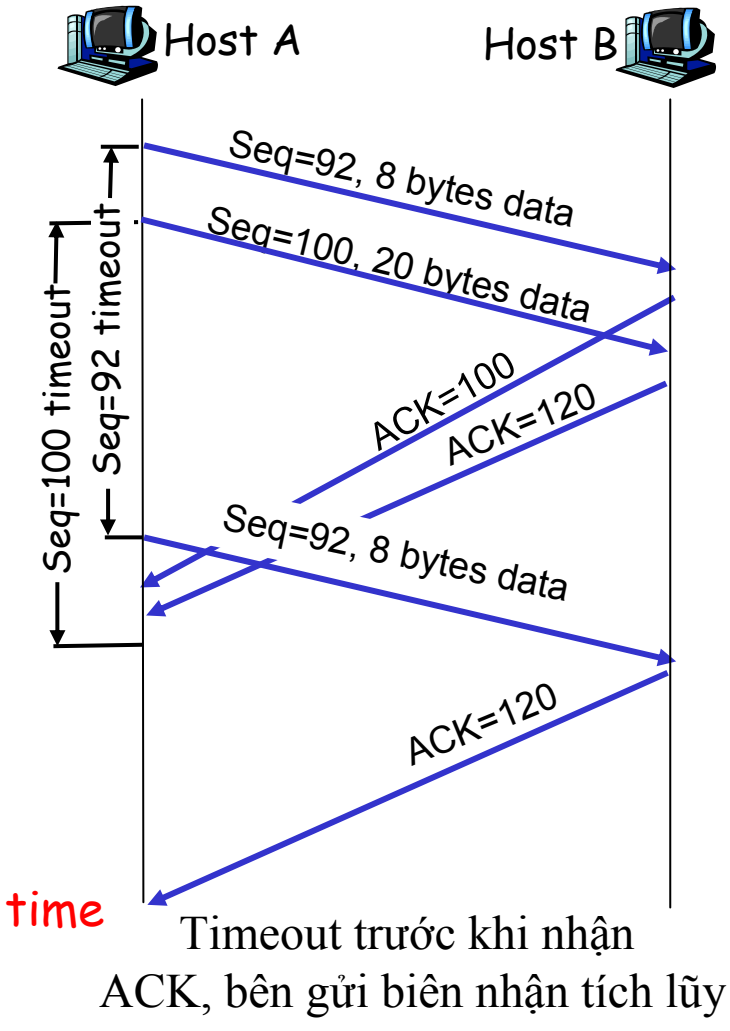
TCP: Chính sách ACK [RFC 1122, RFC 2581]

Sự kiện	Bên nhận (TCP)
Segment theo đúng STT đến, Không thiếu dữ liệu, Không có ACK treo	Trì hoãn ACK. Đợi segment kế tiếp trong 500ms. Nếu không có segment, gửi ACK
Segment theo đúng STT đến, Không thiếu dữ liệu, Có một ACK bị treo	Ngay lập tức gửi một ACK mang giá trị tích lũy
Segment theo đúng STT đến (STT đến lớn hơn số mong đợi). Thiếu dữ liệu	Gửi ACK trùng lặp, chỉ STT của byte dữ liệu mình muốn nhận
Một segment đến điền vào đoạn dữ liệu bị khuyết	Biên nhận STT bên nhận mong muốn nhận

TCP: Ví dụ về Truyền lại



Ví dụ mất ACK



Timeout trước khi nhận ACK, bên gửi biên nhận tích lũy

Điều khiển lưu lượng trong TCP

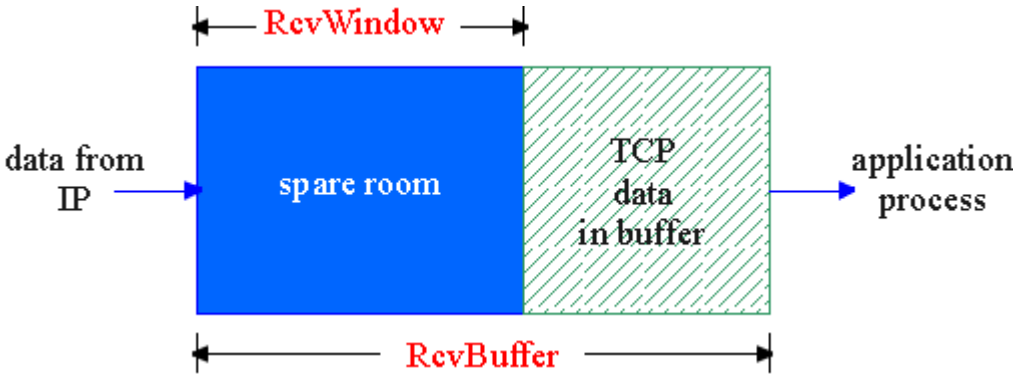
Điều khiển lưu lượng

Không cho bên Gửi gửi quá nhiều, quá nhanh

Phía Nhận: Thông báo rõ ràng cho phía Gửi khả năng nhận dữ liệu của mình (thay đổi thường xuyên)

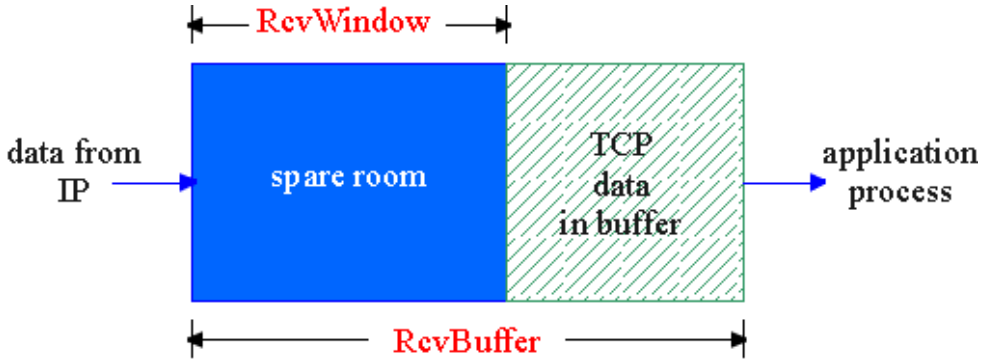
`RcvBuffer` = Kích thước Bộ đệm nhận
`RcvWindow` = Kích thước vùng còn trống trong Bộ đệm

Phía Gửi: Giữ khối lượng dữ liệu gửi đi nhưng chưa được biên nhận nhỏ hơn lượng bên kia chấp nhận được



Bộ đệm phía Nhận

Điều khiển lưu lượng trong TCP



- Chỗ trống trong Bộ đệm

= **RcvWindow**

source port #	dest port #
sequence number	
acknowledgement number	
head len / not used / U A P R S F	rcvr window size
checksum	ptr urgent data
Options (variable length)	
application data (variable length)	

TCP Round Trip Time and Timeout

Q: Thiết lập giá trị timeout ntn ?

- Timeout > RTT
  - Chú ý: RTT thay đổi thường xuyên
- **Quá bé**: timeout ngay
  - Truyền lại không cần thiết
- **Quá lớn**: xử lý việc mất gói tin bị chậm trễ

Q: Làm thế nào để ước lược RTT?

- **`SampleRTT`**: khoảng thời gian từ khi gửi gói tin cho đến khi nhận được biên nhận
  - Bỏ qua truyền lại
- **`SampleRTT`** thay đổi thường xuyên. Chúng ta muốn ước lượng RTT "mịn hơn"
  - Sử dụng nhiều giá trị đo được trong quá khứ, không phải chỉ có một **`SampleRTT`** gần nhất

TCP Round Trip Time và Timeout

EstimatedRTT = (1-x)*EstimatedRTT + x*SampleRTT

- Trọng số sẽ thay đổi giá trị trung bình
- Ảnh hưởng của SampleRTT
- x thường chọn giá trị 0.1

Thiết đặt giá trị timeout

- **EstimtedRTT** cộng thêm một "giá trị an toàn"
- Biến thiên **EstimatedRTT** càng lớn **->** tăng "giá trị an toàn"

Timeout = EstimatedRTT + 4*Deviation

Deviation = (1-x)*Deviation +
x*|SampleRTT-EstimatedRTT|

TCP : Quản lý Kết nối

Chú ý: Trong TCP, phía Gửi và Nhận thiết lập "kết nối" trước khi trao đổi các segment dữ liệu.

- Khởi tạo các biến TCP:
  - Số thứ tự
  - Bộ đệm, Thông tin về lưu lượng (**`RcvWindow`**)
- *client*: Khởi tạo kết nối

```
Socket clientSocket = new
 Socket("hostname","port
 number");
```

- *server*: Đợi kết nối từ client

```
Socket connectionSocket =
 welcomeSocket.accept();
```

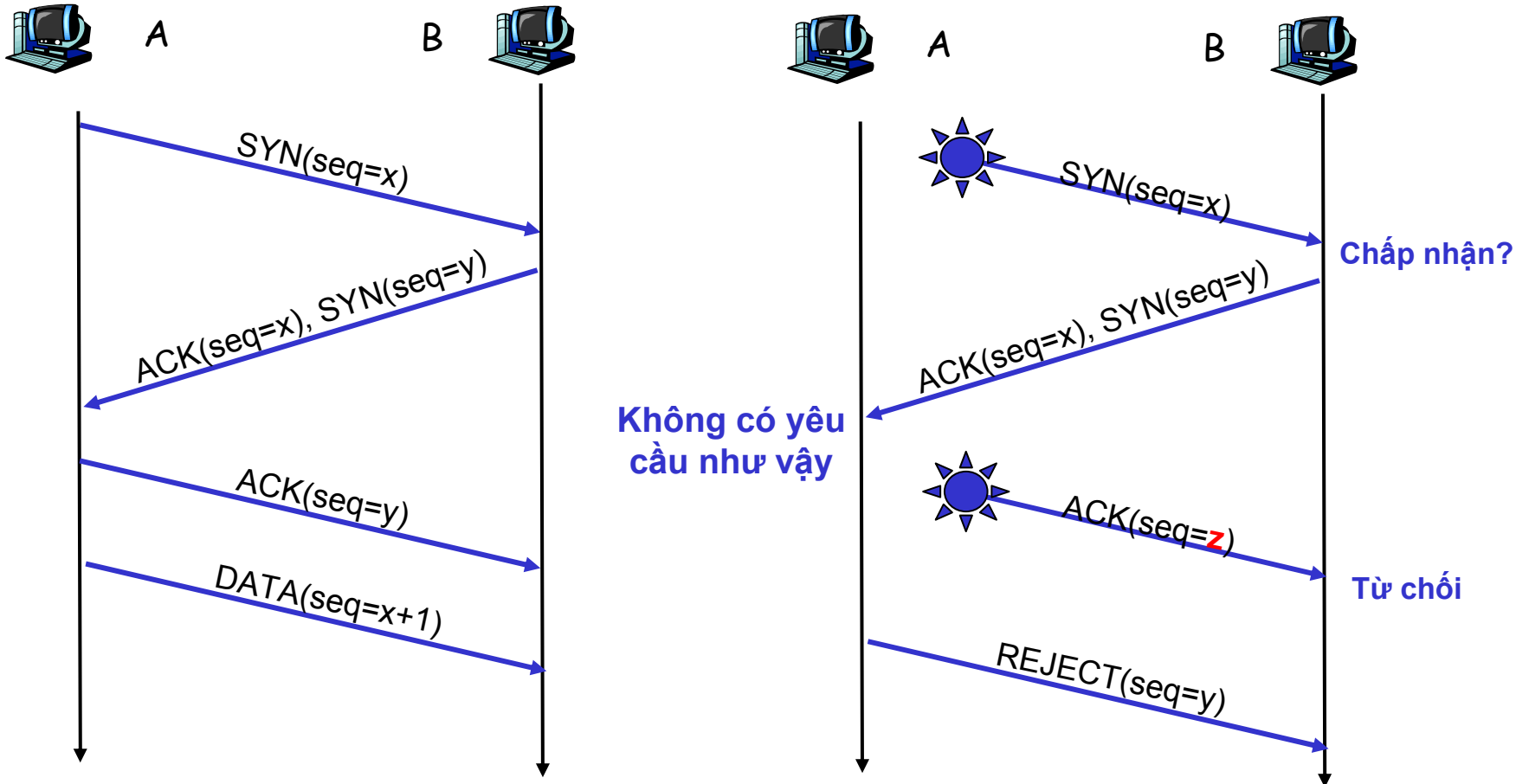
Bắt tay ba bước:

Bước 1: Phía client gửi gói tin điều khiển TCP SYN tới server
- Chứa Số thứ tự khởi đầu

Bước 2: Nhận được gói SYN, nếu chấp nhận kết nối, server gửi trả lời gói tin điều khiển SYNACK
- Biên nhận cho gói SYN vừa nhận
- Cấp phát bộ đệm
- Thông báo về STT khởi đầu của server

Bắt tay ba bước

- Để đảm bảo rằng bên kia thực sự mong muốn thiết lập kết nối



TCP: Quản lý Kết nối (tiếp)

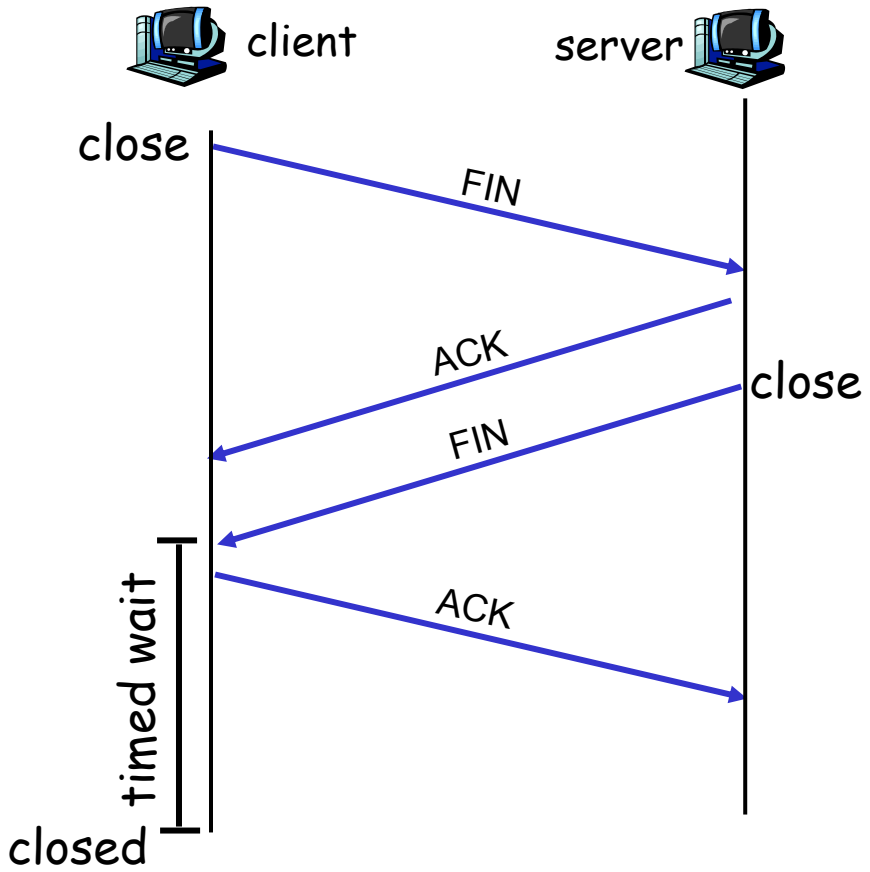
Đóng một kết nối:

client đóng socket:

```
clientSocket.close();
```

Bước 1: client gửi gói điều khiển FIN tới server

Bước 2: server nhận được gói FIN, biên nhận cho gói tin này. Đóng kết nối, gửi gói FIN.



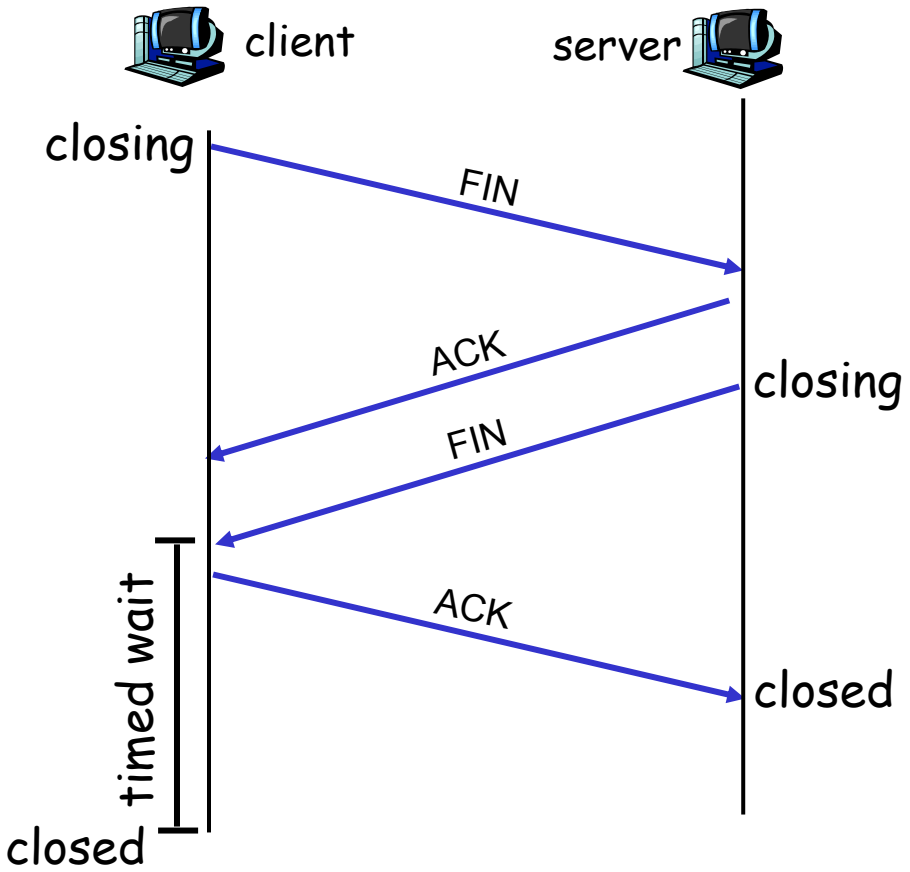
TCP: Quản lý Kết nối (tiếp)

Bước 3: client nhận gói FIN, biên nhận lại ACK.

- Bước vào trạng thái "timed wait" – sẽ biên nhận ACK cho các gói FIN nhận được

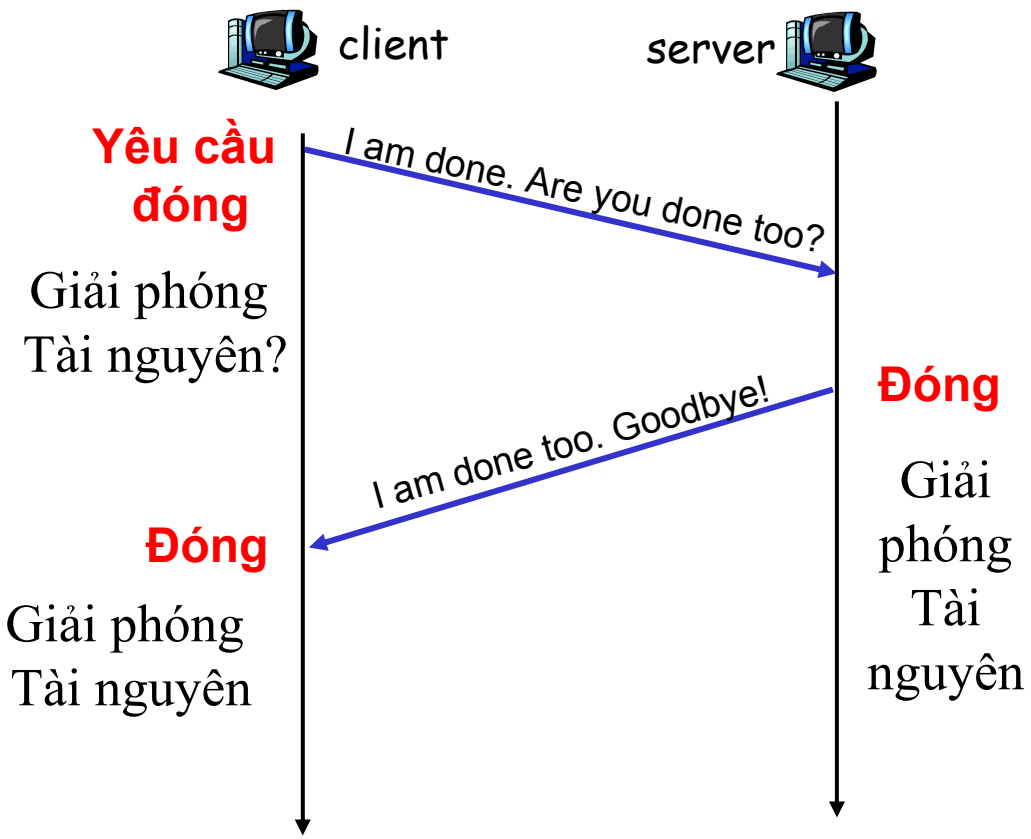
Bước 4: server nhận được ACK, đóng kết nối.

Chú ý: Với vài cải tiến nhỏ, ta có thể xử lý đồng thời nhiều gói FIN.

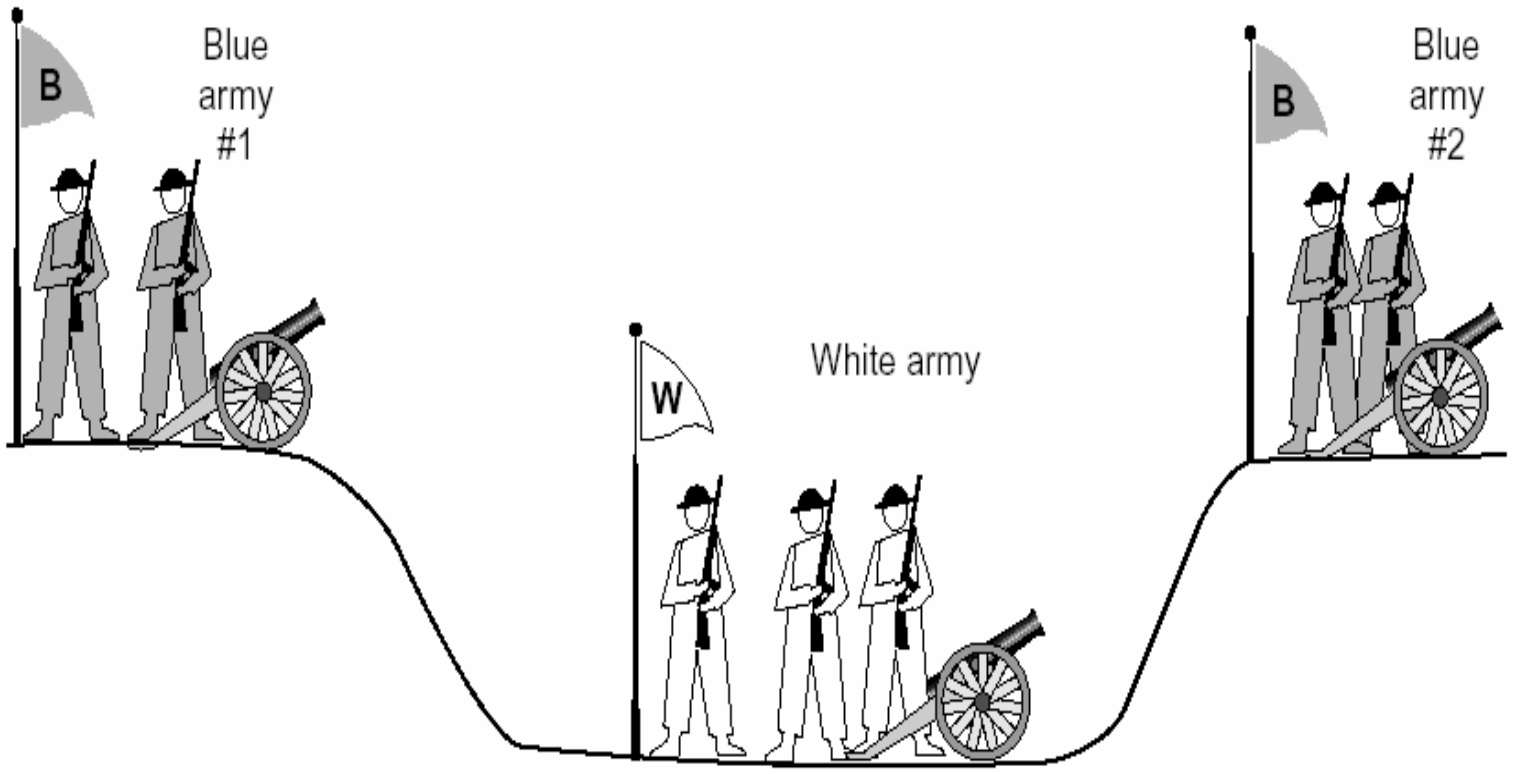


Đóng kết nối

- Mục tiêu:
  - Mỗi phía giải phóng tài nguyên và xóa bỏ trạng thái về kênh truyền

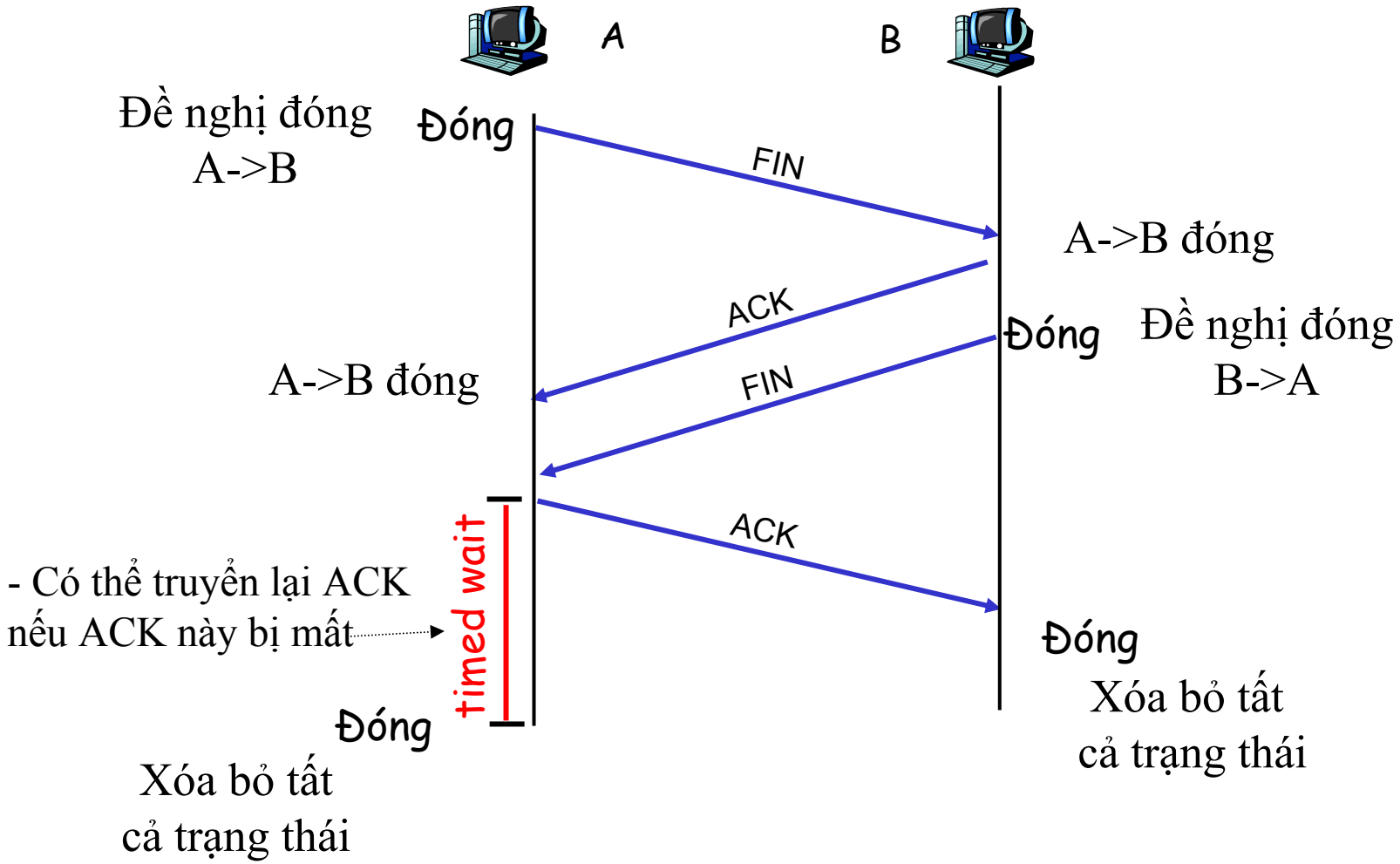


Vấn đề tổng quát: Quân Xanh-Trắng

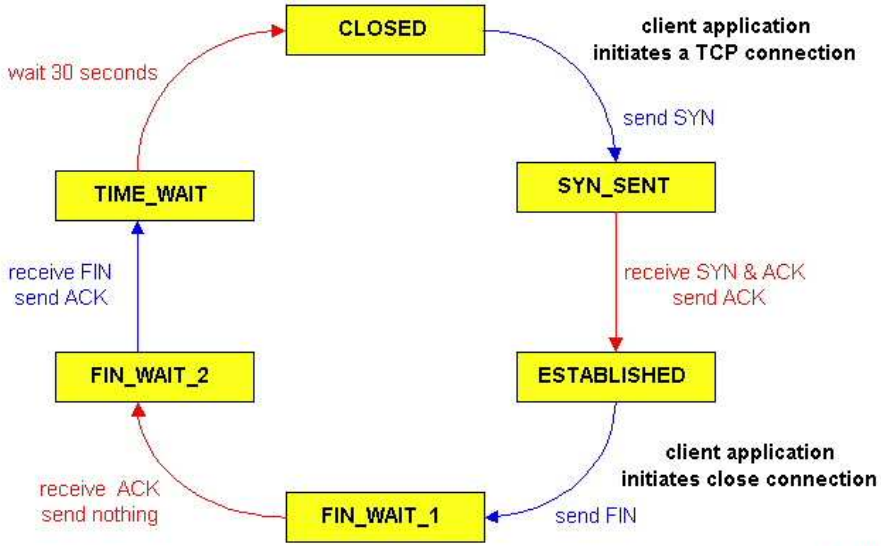


Hai phía quân xanh cần thống nhất thời điểm để cùng tấn công quân trắng. Họ thỏa thuận bằng cách gửi thông điệp cho nhau. Nếu cùng đồng ý : tấn công, còn không sẽ không tấn công. Chú ý rằng người truyền tin có thể bị bắt ! Nếu cùng tấn công, bên xanh thắng, còn nếu tấn công riêng lẻ, bên trắng thắng

Đóng kết nối trong bốn bước

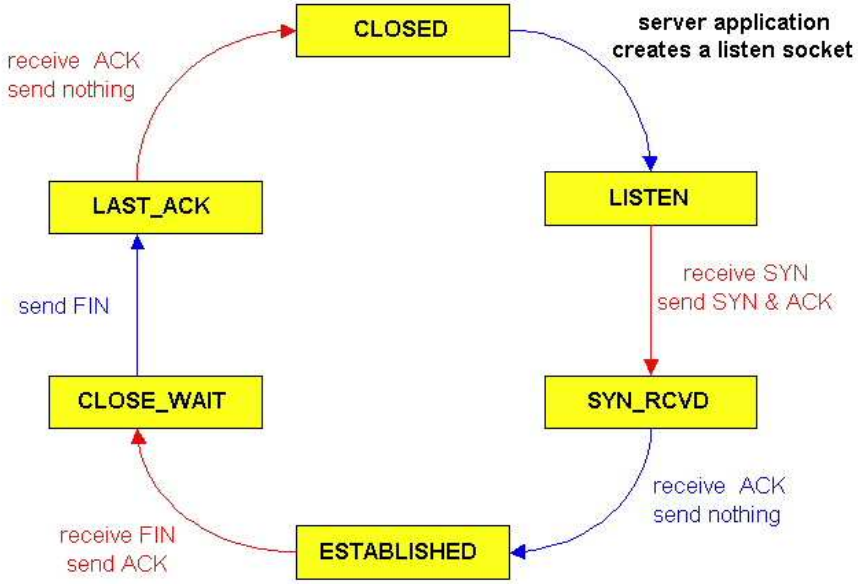


TCP: Quản lý Kết nối (tiếp)

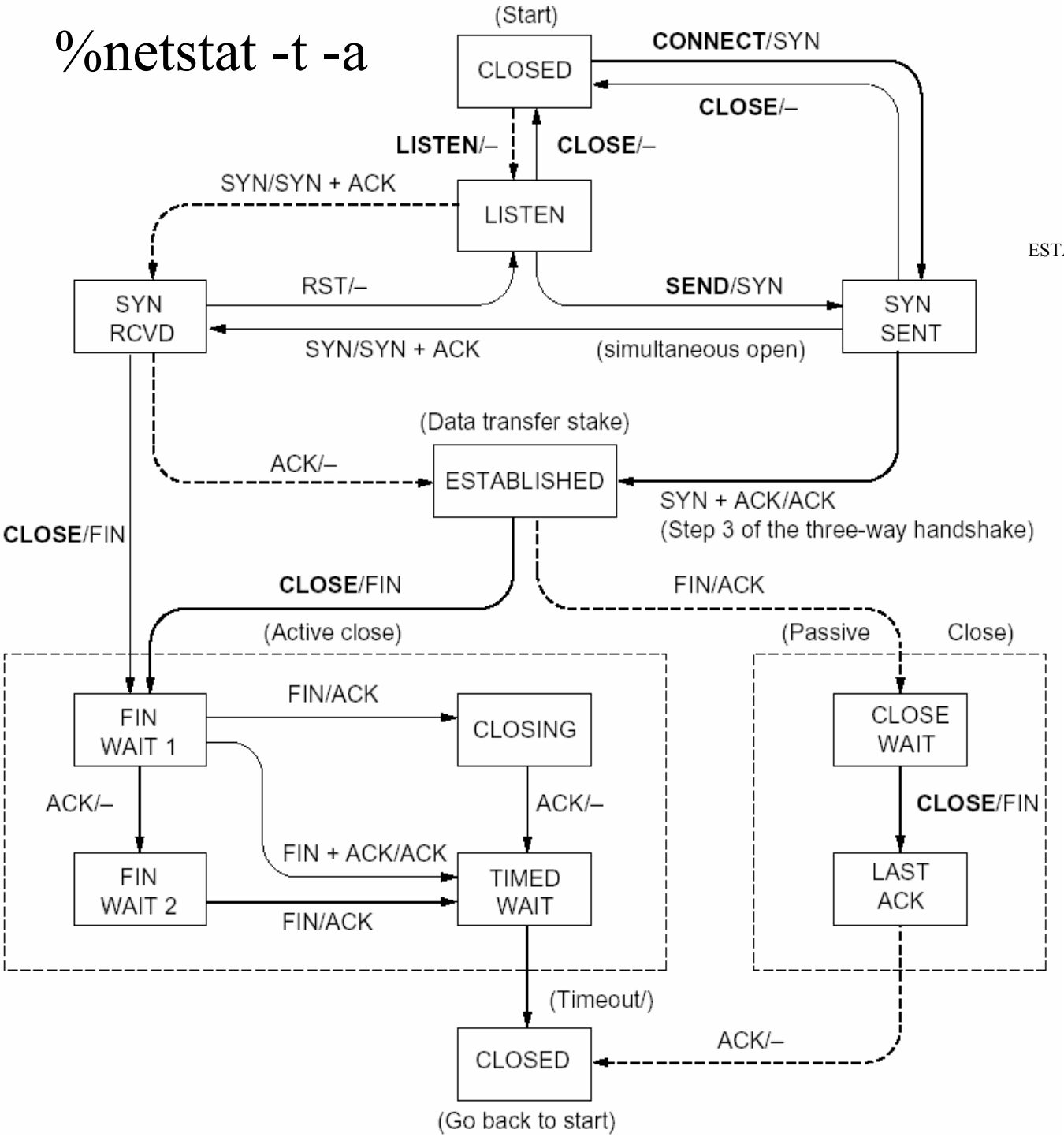


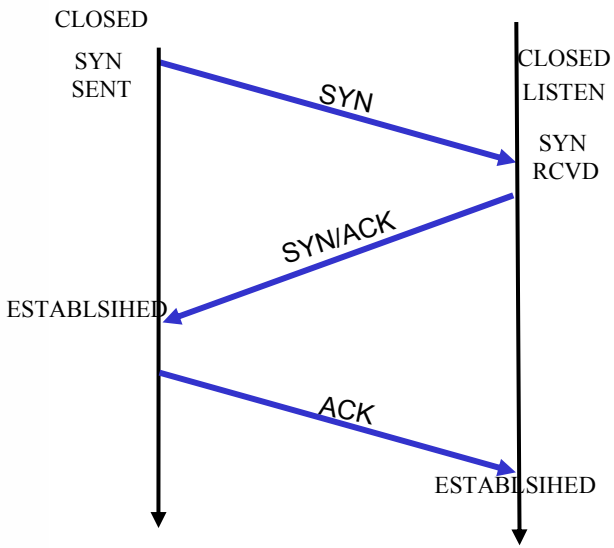
Vòng đời TCP phía Client

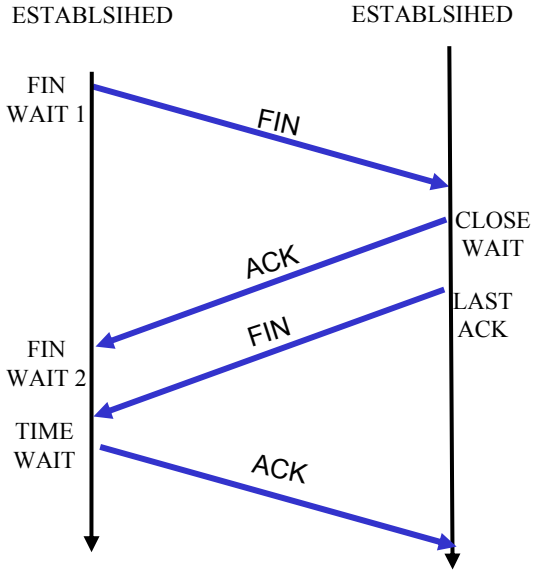
Vòng đời TCP phía Server



%netstat -t -a


(Start)
CLOSED
CONNECT/SYN
CLOSE/–
LISTEN/–
CLOSE/–
SYN/SYN + ACK
LISTEN
SYN RCVD
RST/–
SEND/SYN
SYN SENT
SYN/SYN + ACK
(simultaneous open)
(Data transfer stake)
ACK/–
ESTABLISHED
SYN + ACK/ACK
(Step 3 of the three-way handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
(Active close)
(Passive Close)
FIN WAIT 1
FIN/ACK
CLOSING
ACK/–
ACK/–
FIN + ACK/ACK
FIN WAIT 2
TIMED WAIT
FIN/ACK
CLOSE WAIT
CLOSE/FIN
LAST ACK
(Timeout/)
CLOSED
ACK/–
(Go back to start)


CLOSED
SYN SENT
CLOSED LISTEN
SYN
SYN RCVD
SYN/ACK
ESTABLSIHED
ACK
ESTABLSIHED


ESTABLSIHED
ESTABLSIHED
FIN WAIT 1
FIN
CLOSE WAIT
ACK
LAST ACK
FIN
FIN WAIT 2
TIME WAIT
ACK

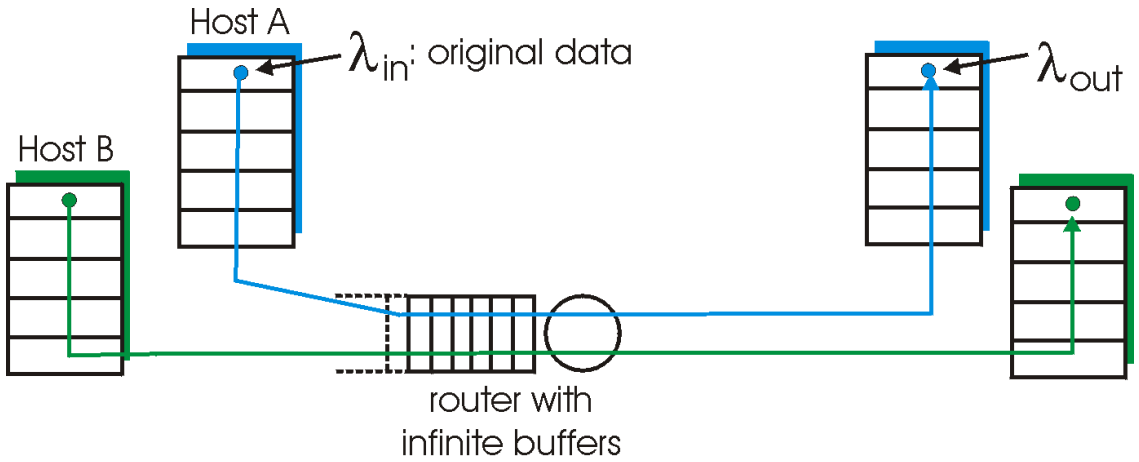
Nguyên tắc Kiểm soát Tắc nghẽn

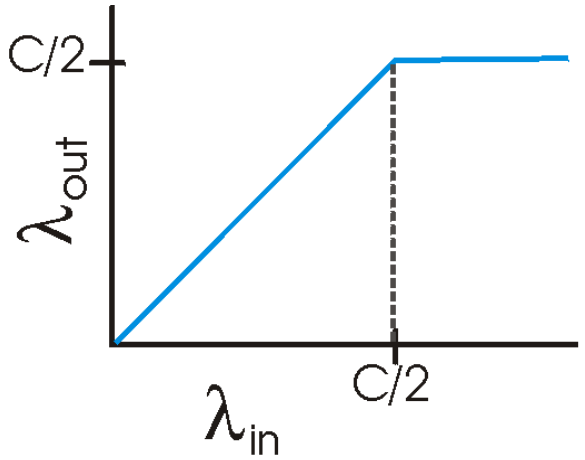
Tắc nghẽn:

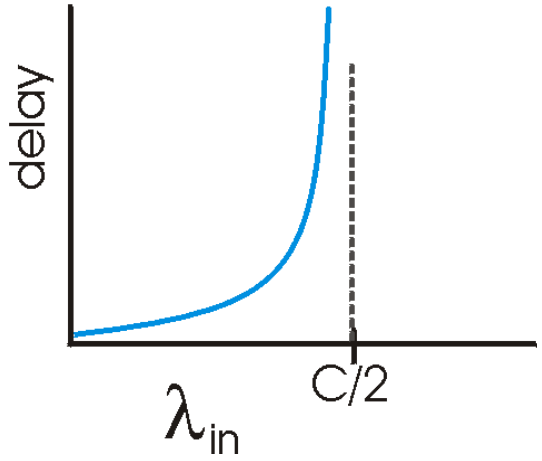
- Mường tượng: "có ***quá nhiều*** nút gửi ***quá nhiều*** dữ liệu với tốc độ ***quá nhanh*** mà mạng không chuyển kịp"
- Khác với ***Điều khiển lưu lượng***!
- Biểu hiện :
  - ***Mất gói tin*** (Tràn bộ đệm tại router)
  - ***Độ trễ lớn*** (Các gói tin phải "xếp hàng" tại router)
- Là một trong **10** vấn đề quan trọng nhất !

Nguyên nhân và Giá tắc nghẽn: Ví dụ 1

- 2 gửi, 2 nhận
- Router với bộ đệm vô hạn
- Không có cơ chế truyền lại



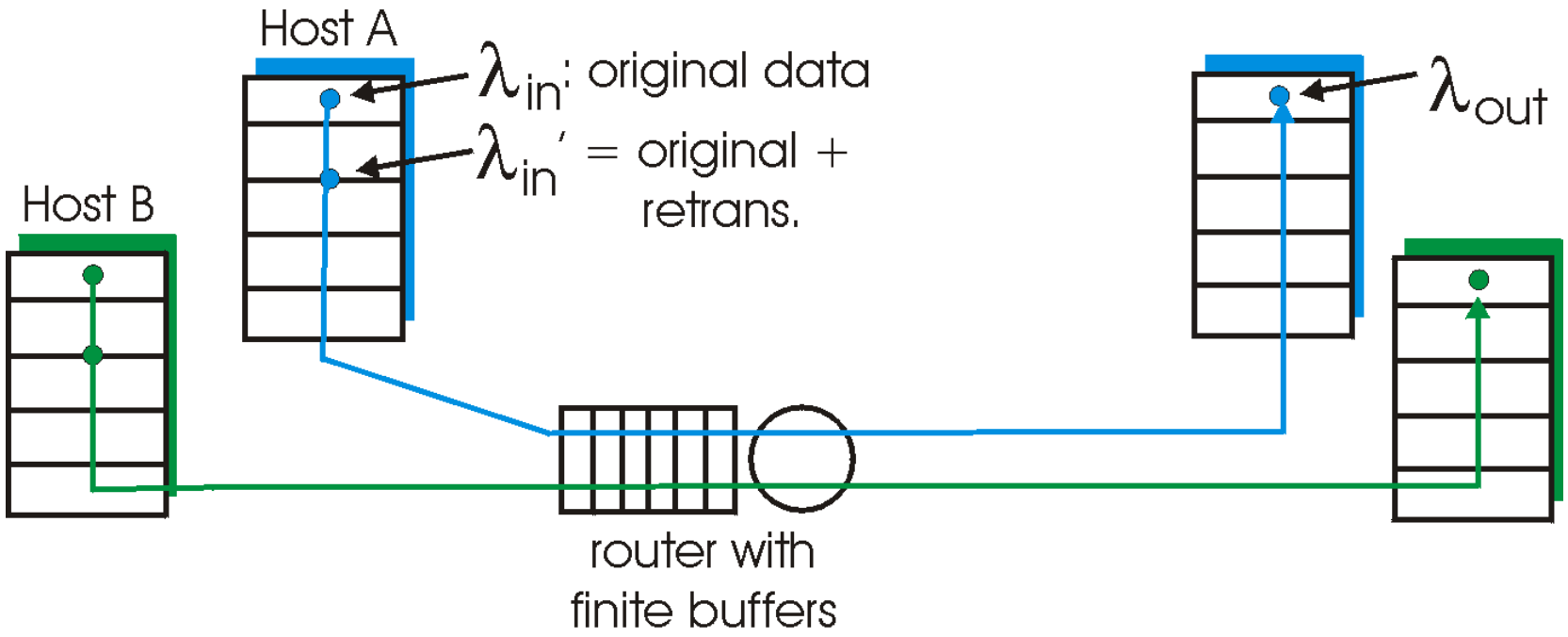




- Độ trễ lớn khi tắc nghẽn
- Thông lượng có thể đạt cực đại

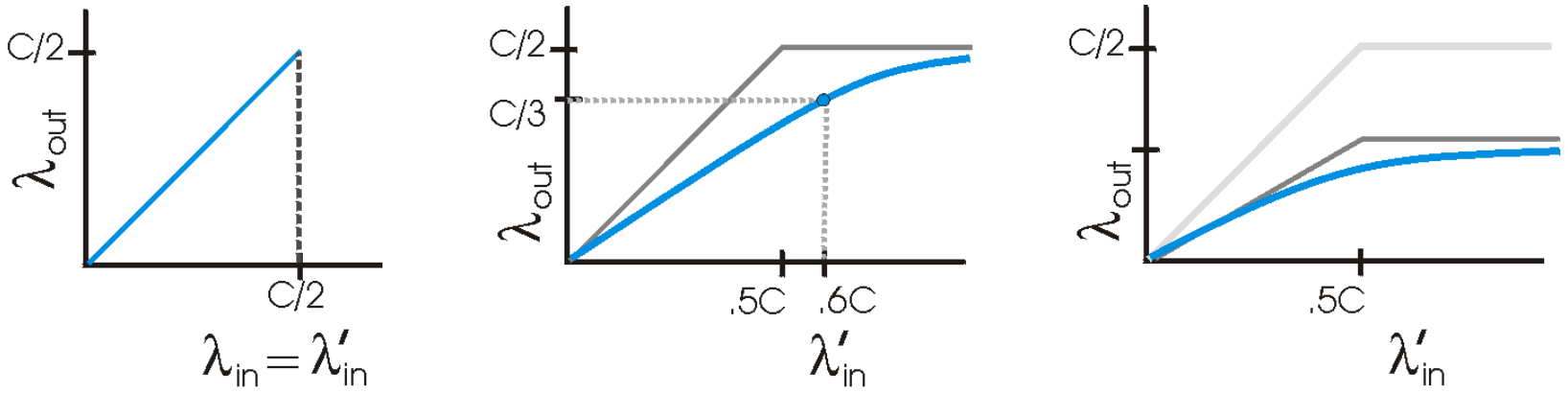
Nguyên nhân và Giá tắc nghẽn: Ví dụ 2

- Một router, bộ đệm *hữu hạn*
- Gửi lại các packet bị mất



Nguyên nhân và Giá tắc nghẽn: Ví dụ 2

- Thông thường: $\lambda_{in} = \lambda_{out}$ (tốt)
- Truyền lại khi mất (lý tưởng): $\lambda'_{in} > \lambda_{out}$
- Truyền lại của các gói tin đến trễ (không bị mất) khiến λ'_{in} lớn hơn (so với trường hợp lý tưởng) λ_{out}



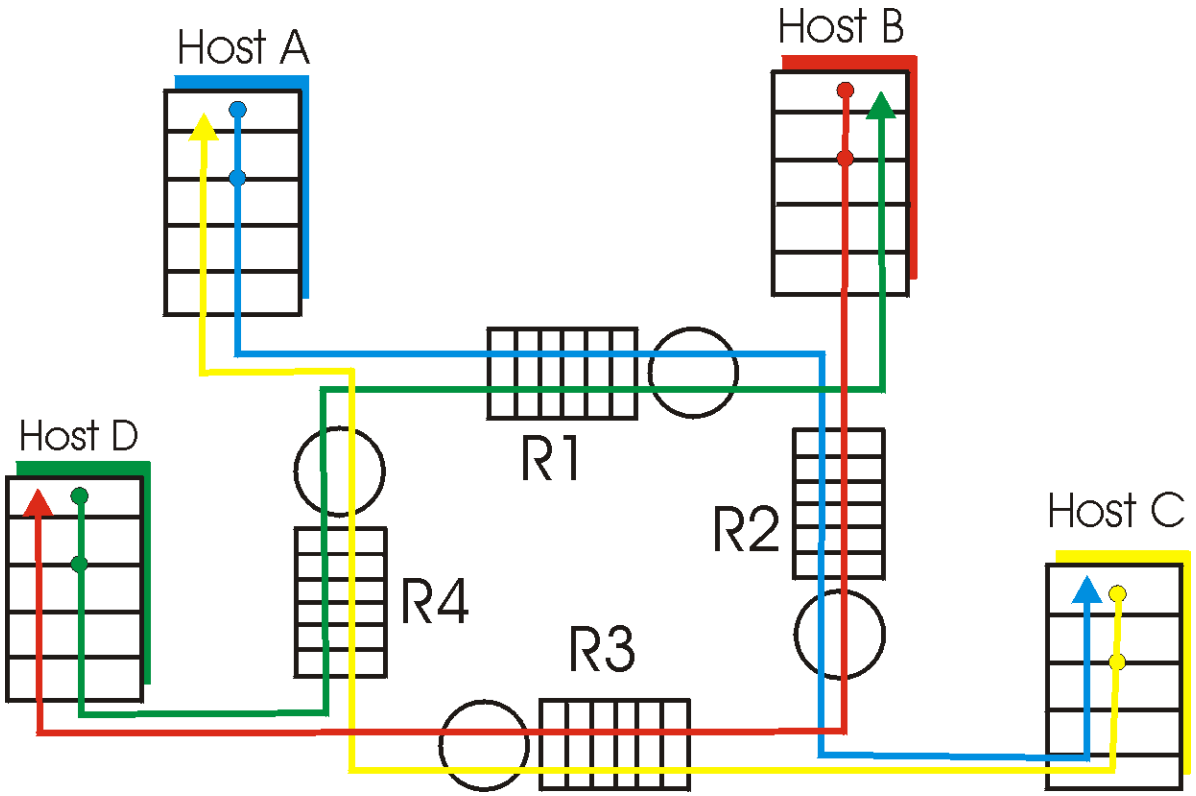
"Giá" của Tắc nghẽn:

- Phải truyền lại nhiều
- Truyền lại không cần thiết: Nhiều bản sao của cùng 1 gói tin có thể nằm trên mạng

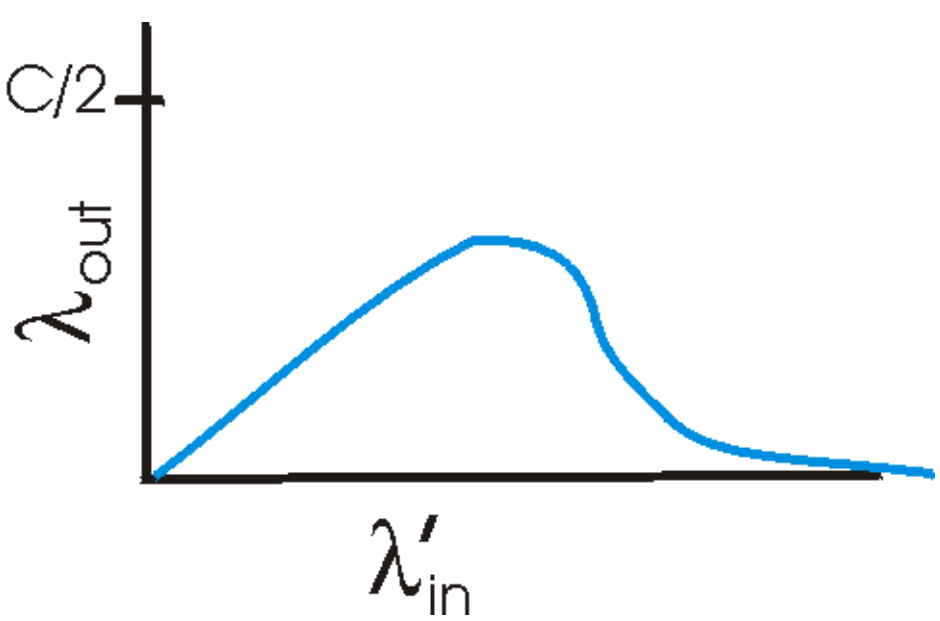
Nguyên nhân và Giá tắc nghẽn: Ví dụ 3

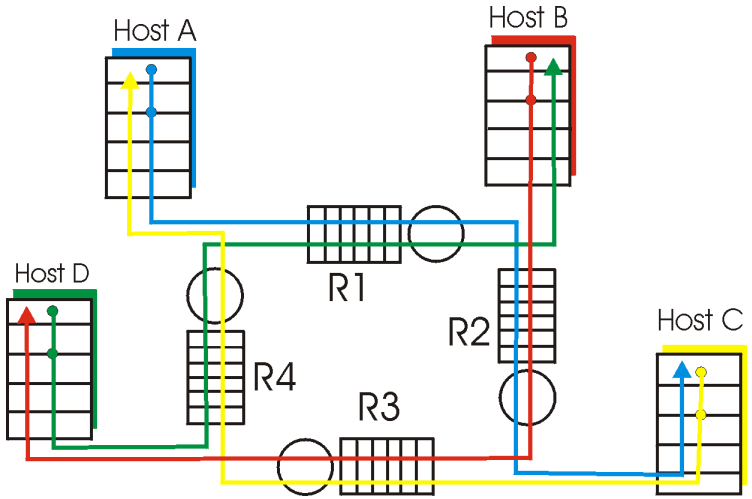
- 4 người gửi
- Nhiều tuyến
- Timeout => Gửi lại

Q: Chuyện gì xảy ra khi λ_{in} và tăng λ'_{in} ?



Nguyên nhân và Giá tắc nghẽn: Ví dụ 3





Một vấn đề khác của tắc nghẽn:

- Khi một packet bị mất, tất cả "công sức" tạo và chuyển gói tin này của các tầng bên trên đều bị mất !

Giải pháp chống Tắc nghẽn

Có hai lớp giải pháp chính:

Giải pháp đầu cuối:

- Tầng mạng (router) không thông báo cho các nút về Tắc nghẽn (nếu có)
- Mất gói tin, Độ trễ lớn: dấu hiệu của Tắc nghẽn
- Là giải pháp được TCP áp dụng

Có sự hỗ trợ từ mạng:

- routers thông báo cho thiết bị đầu cuối
  - Sử dụng một bit thông báo tình trạng tắc nghẽn (SNA, DECbit, TCP/IP ECN, ATM)
  - Thông báo tốc độ gửi tối đa

Ví dụ : Chống tắc nghẽn trong ATM

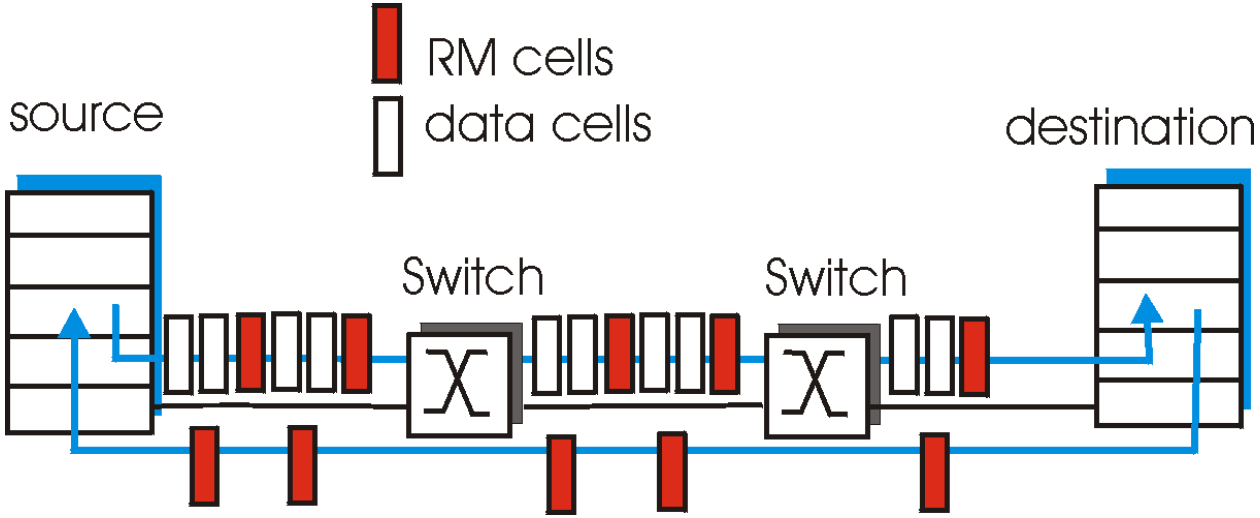
ABR: available bit rate:

- Dịch vụ “co giãn”
- Nếu đường truyền ở phía gửi chưa dùng hết:
  - Phía gửi có thể gửi thêm
- Nếu đường truyền ở phía gửi tắc nghẽn :
  - Phía gửi có thể được đảm bảo một băng thông tối thiểu

Tế bào RM (resource management) :

- Phía Gửi gửi kèm cùng các tế bào dữ liệu
- ATM switch có thể thiết lập một số bit trong tế bào RM (“có sự trợ giúp từ mạng”)
  - NI bit: Không được tăng tốc độ gửi (Tắc nghẽn ít)
  - CI bit: Có tắc nghẽn
- Tế bào RM được phía Nhận gửi trả cho phía Gửi

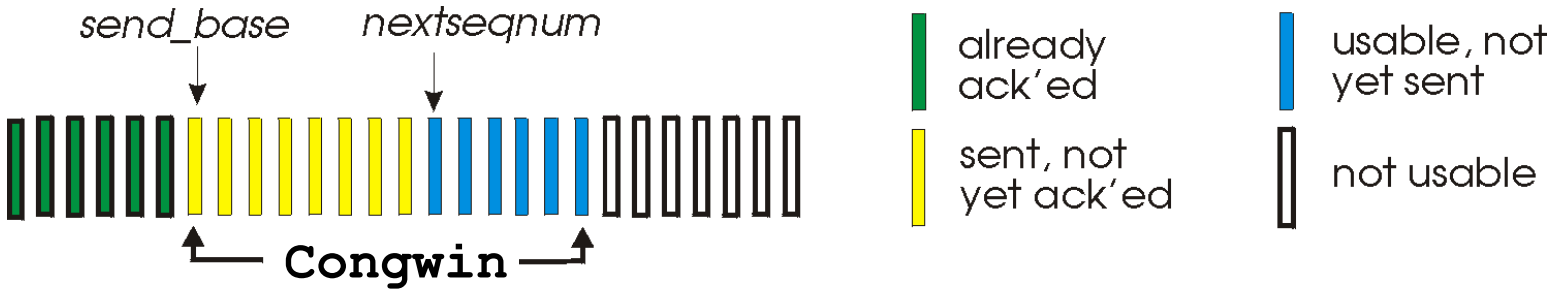
Ví dụ : Chống tắc nghẽn trong ATM



- Trường ER (explicit rate) 2 byte trong tế bào RM
  - switch bị tắc nghẽn có thể giảm ER trong tế bào
  - sender' send rate thus minimum supportable rate on path
- Trường EFCI 1 bit được switch tắc nghẽn thiết lập giá trị 1
  - Nếu tế bào dữ liệu đằng trước tế bào RM có giá trị EFCI =1, phía gửi thiết lập bit CI trong tế bào RM phản hồi

Kiểm soát tắc nghẽn trong TCP

- Kiểm soát Đầu cuối (Mạng không hỗ trợ)
- Tốc độ truyền bị giới hạn bởi cửa sổ kiểm soát tắc nghẽn, **Congwin**, (số lượng segment) :



- w segments, kích thước là MSS byte được gửi đi trong 1 RTT:

$$\text{Thông lượng} = \frac{w * MSS}{RTT} \text{ Bytes/sec}$$

TCP : Kiểm soát tắc nghẽn

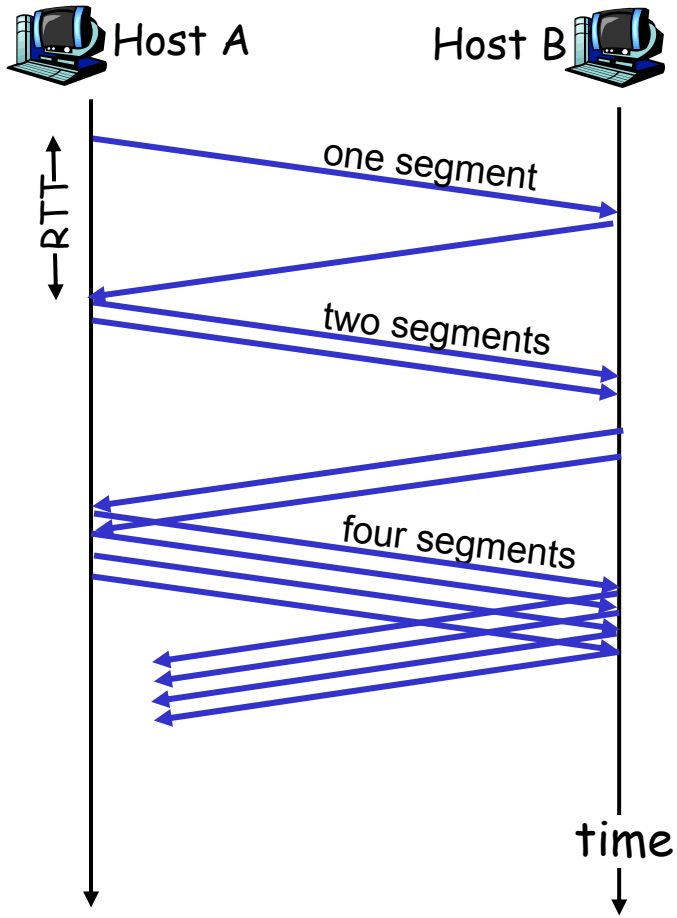
- "**thăm dò**" băng thông của đường truyền:
  - **Lý tưởng**: khi không có tắc nghẽn, truyền nhanh nhất có thể (**`Congwin`** càng lớn càng tốt)
  - ***Tăng*** **`Congwin`** cho đến khi có mất dữ liệu (tắc nghẽn)
  - Mất mát: ***Giảm*** **`Congwin`**, và bắt đầu quá trình thăm dò
- hai "giai đoạn"
  - Khởi đầu chậm
  - Tránh tắc nghẽn
- Một số biến quan trọng:
  - **`Congwin`**
  - **`threshold:`** xác định giá trị ***Ngưỡng*** giữa hai pha

TCP : Khởi đầu chậm

Thuật toán khởi đầu chậm

initialize: Congwin = 1
for (each segment ACKed)
    Congwin++
until (loss Sự kiện OR
    CongWin > threshold)

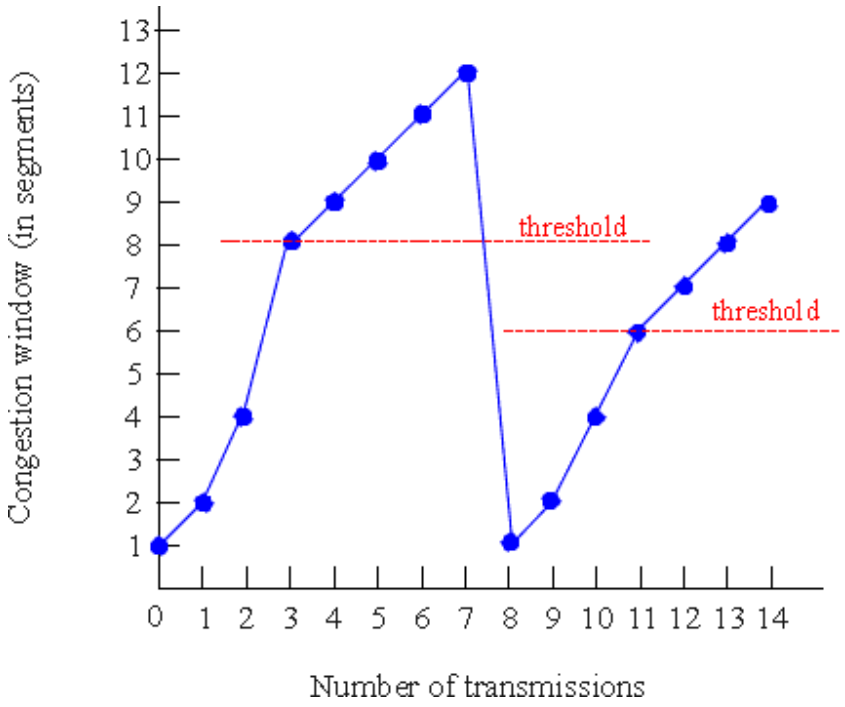
- Kích thước cửa sổ tăng theo hàm số mũ (không quá chậm !)
- Sự kiện loss : timeout (Tahoe TCP) hoặc/và ba lần nhận ACK trùng lặp (Reno TCP)



TCP : Tránh tắc nghẽn

Tránh tắc nghẽn

```
/* slowstart is over        */
/* Congwin > threshold */
Until (loss Sự kiện) {
  every w segments ACKed:
      Congwin++
  }
threshold = Congwin/2
Congwin = 1
perform slowstart[1]
```



1: TCP Reno bỏ qua giai đoạn khởi đầu chậm (khôi phục nhanh) sau khi nhận 3 ACK trùng lặp

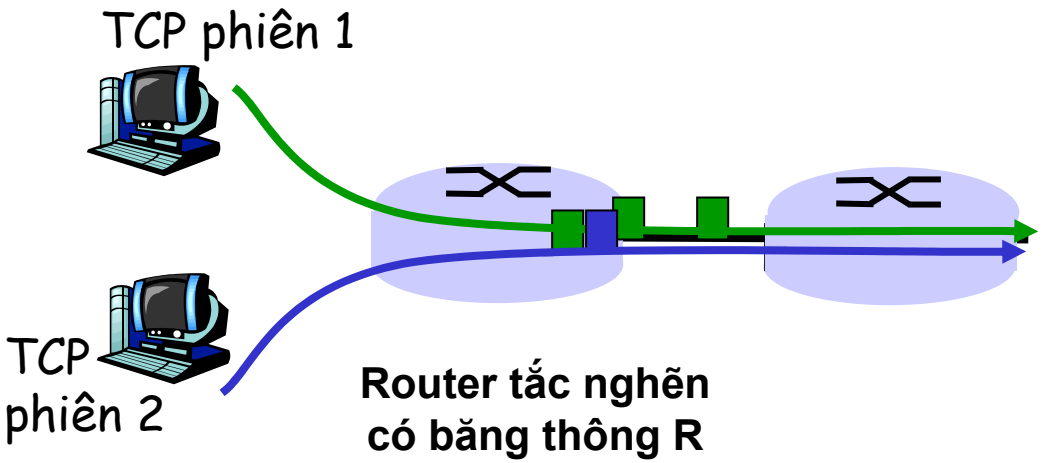
AIMD

Tránh tắc nghẽn trong TCP:

- AIMD: *additive increase, multiplicative decrease*
  - Tăng cửa sổ lên 1 khi nhận được một gói phản hồi
  - Giảm cửa sổ theo số mũ của 2 khi có sự kiện mất gói dữ liệu

Tính công bằng trong TCP

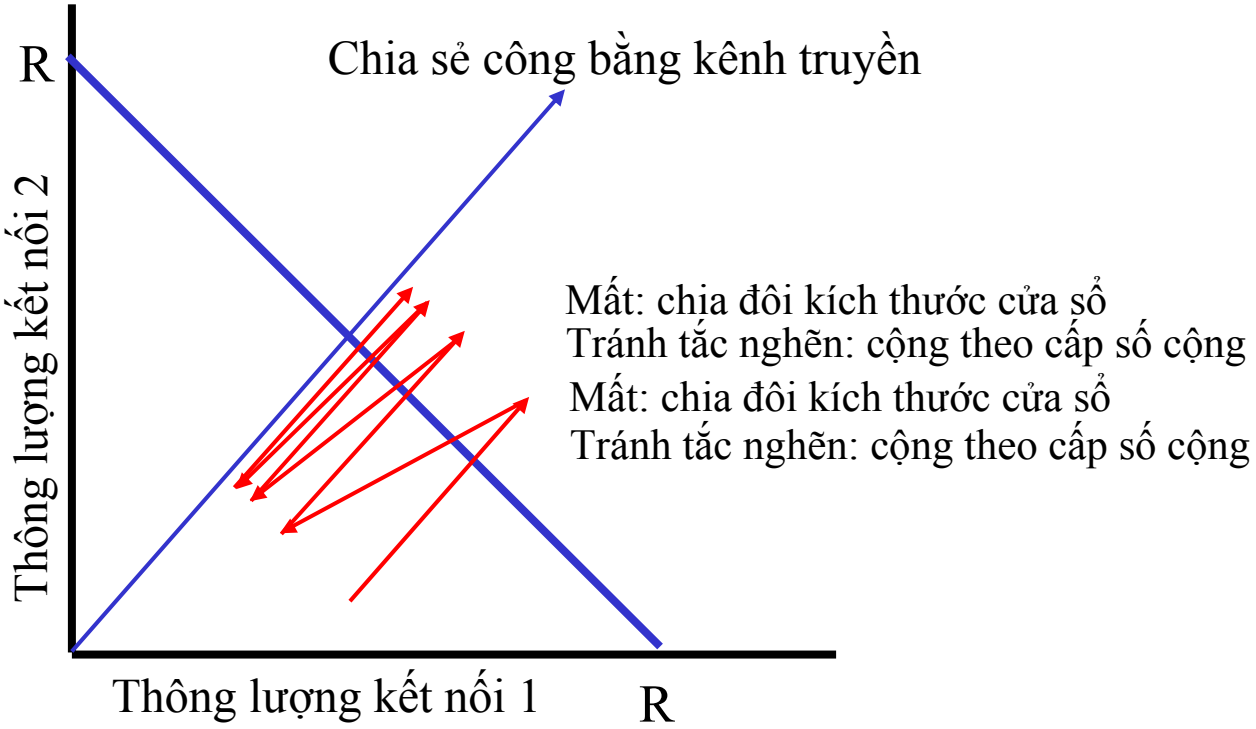
Công bằng: Nếu N phiên TCP cùng nhau chia sẻ một kênh truyền tắc nghẽn, mỗi phiên nhận được 1/N băng thông



Tại sao TCP công bằng?

Hai phiên cạnh tranh nhau sử dụng đường truyền:

- Tăng theo cấp số cộng : băng thông tăng dần dần
- Giảm theo cấp số nhân : giảm đột ngột băng thông



Chapter 3: Tổng kết

- Các dịch vụ của tầng giao vận:
  - Phân kênh/ Dồn kênh
  - Truyền tin cậy
  - Điều khiển lưu lượng
  - Kiểm soát tắc nghẽn
- Cài đặt trên Internet
  - UDP
  - TCP

Tiếp theo:

- Rời khỏi lớp "Rìa" của Mạng
- Tiến vào lớp "Lõi"