

Chương 3: Nội dung

3.1 Các dịch vụ tầng giao vận

3.2 Ghép kênh và phân kênh

3.3 Vận chuyển không kết nối: UDP

3.4 Các nguyên lý truyền dữ liệu tin cậy

3.5 Vận chuyển hướng kết nối: TCP

- Cấu trúc đoạn dữ liệu (segment)
- Truyền dữ liệu tin cậy
- Điều khiển luồng
- Quản lý kết nối

3.6 Các nguyên lý điều khiển tắc nghẽn

3.7 Điều khiển tắc nghẽn TCP

Tầng giao vận 3-56

Khái quát TCP RFCs: 793,1122,1323, 2018, 2581

❖ **Điểm-tới-điểm:**

- Một bên gửi, một bên nhận

❖ **Truyền dòng byte theo đúng thứ tự và truyền tin cậy:**

- Không có “ranh giới thông điệp”

❖ **pipeline:**

- Điều khiển tắc nghẽn và điều khiển luồng TCP thiết lập kích thước cửa sổ

❖ **Truyền dữ liệu song công (full duplex):**

- Luồng dữ liệu đi theo 2 hướng trên cùng một kết nối
- MSS: maximum segment size (kích thước đoạn lớn nhất)

❖ **Hướng kết nối:**

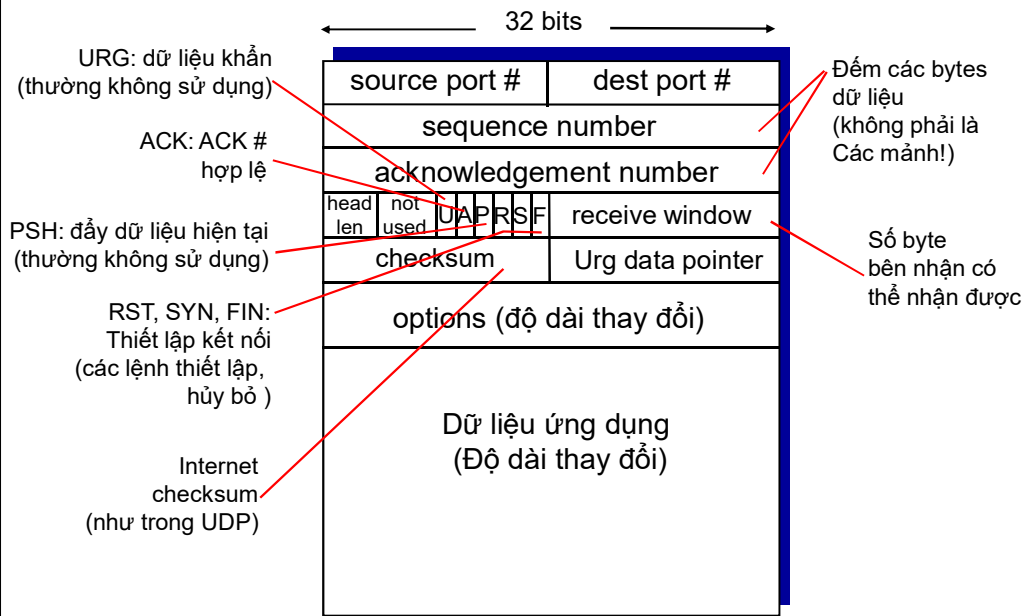
- Bắt tay (trao đổi các thông điệp điều khiển) khởi tạo trạng thái cho bên gửi và bên nhận trước khi trao đổi dữ liệu

❖ **Điều khiển luồng:**

- Bên gửi không lấn át bên nhận

Tầng giao vận 3-57

Cấu trúc TCP segment



Tầng giao vận 3-58

Số thứ tự và báo nhận ACK trong TCP

Số thứ tự:

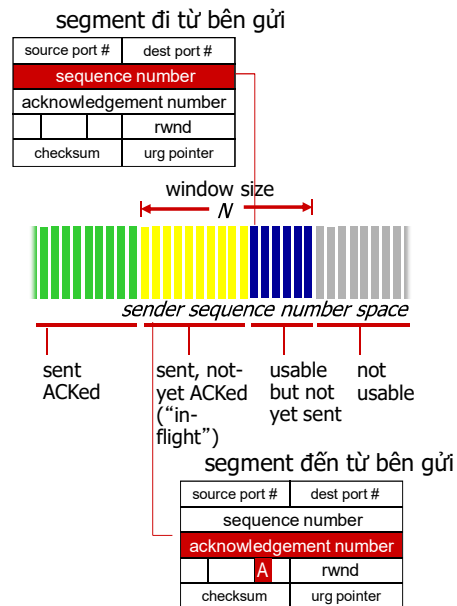
- “Số” dòng byte của byte đầu tiên trong đoạn (segment) dữ liệu

Báo nhận:

- Số thứ tự của byte tiếp theo được mong đợi từ phía bên kia
- ACK tích lũy

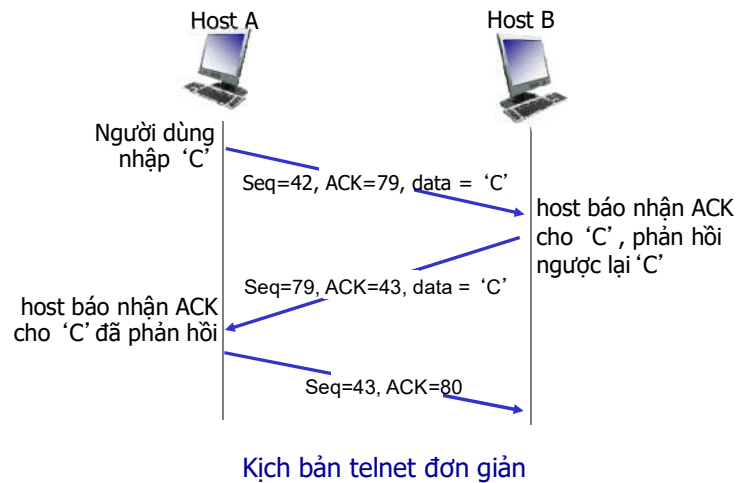
Hỏi: Làm thế nào bên nhận xử lý được các segment không đúng thứ tự?

- Trả lời: TCP không đề cập, tùy thuộc vào người thực hiện



Tầng giao vận 3-59

Số thứ tự và báo nhận ACK trong TCP



Tầng giao vận 3-60

TCP round trip time và timeout

Hỏi: Làm thế nào để thiết lập giá trị TCP timeout?

- ❖ Dài hơn RTT
 - nhưng RTT thay đổi
- ❖ *Quá ngắn*: timeout sớm, không cần truyền lại
- ❖ *Quá dài*: phản ứng chậm với các segment bị mất

Hỏi: Ước lượng RTT như nào?

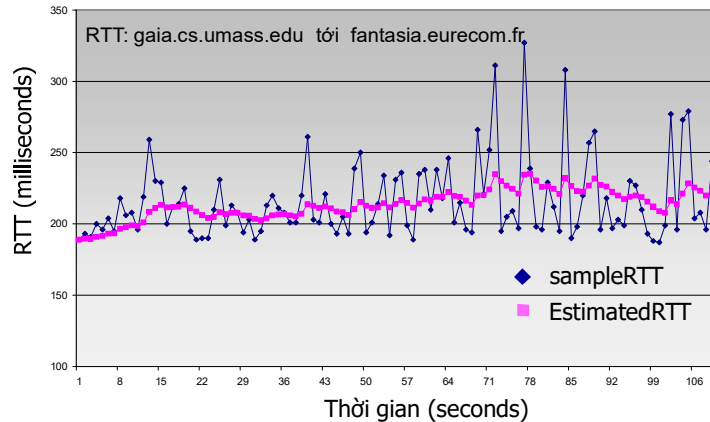
- ❖ **SampleRTT**: thời gian đo được từ khi truyền segment đến khi nhận được ACK
 - Bỏ qua việc truyền lại
- ❖ **SampleRTT** có thể thay đổi, cần giá trị RTT ước lượng “mượt hơn”
 - Tính trung bình một vài độ đo gần đây, không chỉ **SampleRTT** hiện tại

Tầng giao vận 3-61

TCP round trip time và timeout

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

❖ Giá trị đặc trưng: $\alpha = 0.125$



Tăng giao vận 3-62

TCP round trip time và timeout

- ❖ Khoảng thời gian timeout: **EstimatedRTT** cộng với “hệ số dự trữ an toàn”
 - Nếu có biến thiên lớn trong **EstimatedRTT**, thì hệ số dự trữ an toàn phải lớn hơn
- ❖ Ước lượng sự biến thiên của SampleRTT từ EstimatedRTT:

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

(Giá trị đặc trưng: $\beta = 0.25$)

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



RTT ước lượng

“hệ số dự trữ an toàn”

Tăng giao vận 3-63

Chương 3: Nội dung

3.1 Các dịch vụ tầng giao vận

3.2 Ghép kênh và phân kênh

3.3 Vận chuyển không kết nối: UDP

3.4 Các nguyên lý truyền dữ liệu tin cậy

3.5 Vận chuyển hướng kết nối: TCP

- Cấu trúc đoạn dữ liệu (segment)
- Truyền dữ liệu tin cậy
- Điều khiển luồng
- Quản lý kết nối

3.6 Các nguyên lý điều khiển tắc nghẽn

3.7 Điều khiển tắc nghẽn TCP

Tầng giao vận 3-64

Truyền dữ liệu tin cậy trong TCP

❖ TCP tạo dịch vụ rdt trên dịch vụ không tin cậy của IP

- Truyền segment theo kiểu pipelining
- ACK tích lũy
- Dùng bộ định thời cho việc truyền lại

❖ Việc truyền lại được kích hoạt bởi:

- Các sự kiện timeout
- ACK bị trùng lặp

Hãy bắt đầu xem xét bên gửi TCP theo cách đơn giản:

- Bỏ qua trùng lặp ACK
- Bỏ qua điều khiển luồng, điều khiển tắc nghẽn

Tầng giao vận 3-65

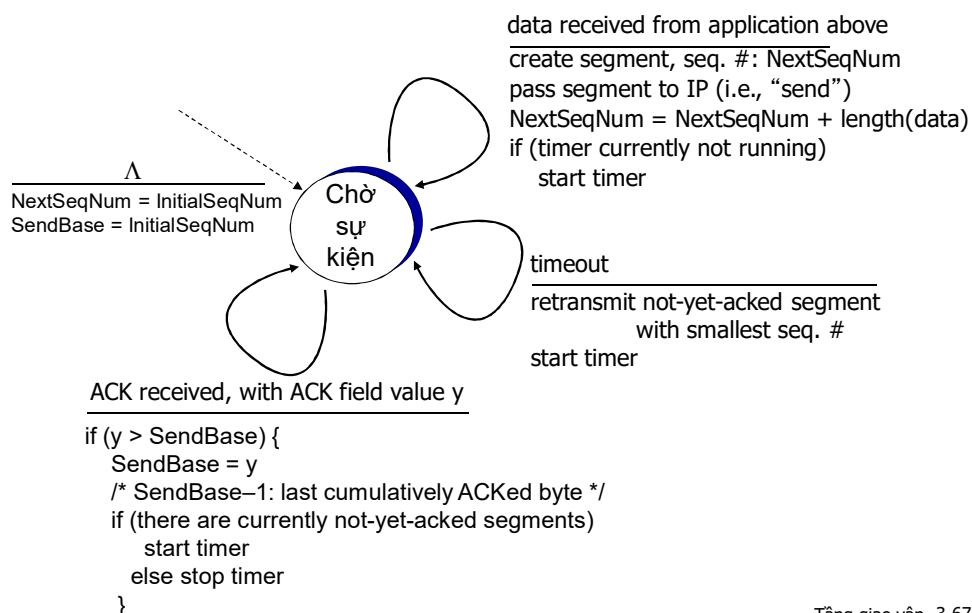
Các sự kiện của TCP bên gửi:

Dữ liệu nhận từ ứng dụng: Timeout:

- ❖ Tạo segment với số thứ tự
 - ❖ Số thứ tự là số dòng byte của byte dữ liệu đầu tiên trong segment
 - ❖ Khởi tạo bộ định thời nếu chưa chạy:
 - Chú ý bộ định thời của segment chưa được báo nhận muộn nhất
 - Hết thời hạn: **TimeOutInterval**
 - ❖ Truyền lại segment bị timeout
 - ❖ Khởi tạo lại bộ định thời
- ### *ACK đã nhận:*
- ❖ Nếu ACK báo nhận cho các segment chưa được báo nhận trước đó, thì:
 - Cập nhật lại các segment đã được báo nhận
 - Khởi tạo bộ định thời nếu vẫn còn các segment chưa được báo nhận

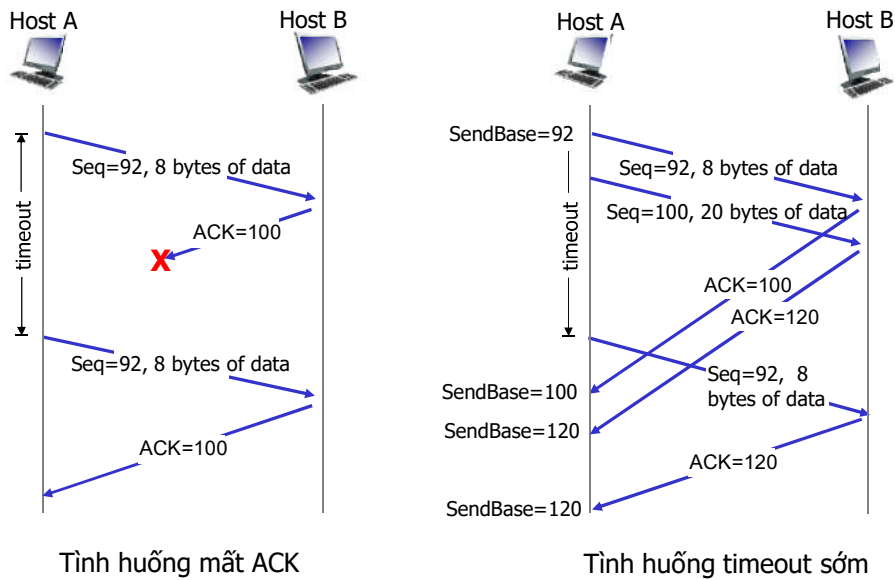
Tầng giao vận 3-66

TCP bên gửi (đơn giản hóa)



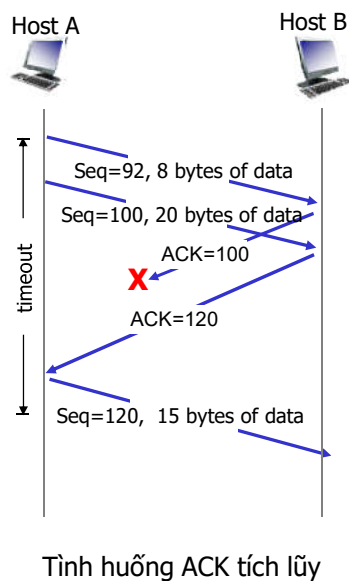
Tầng giao vận 3-67

TCP: Các tình huống phải truyền lại



Tầng giao vận 3-68

TCP: Các tình huống phải truyền lại



Tầng giao vận 3-69

Tạo ACK trong TCP [RFC 1122, RFC 2581]

Sự kiện tại bên nhận

Hành động của TCP tại bên nhận

Segment đến đúng thứ tự với số thứ tự mong muốn. Tất cả dữ liệu đến đã được báo nhận

ACK bị trễ. Chờ 500ms cho segment tiếp theo. Nếu không có segment tiếp theo thì gửi ACK

Segment đến đúng thứ tự với số thứ tự mong muốn. Một segment khác đang chờ ACK

Gửi ngay một ACK tích lũy, báo nhận ACK cho cả hai segment đến đúng thứ tự

Segment đến không đúng số thứ tự, số thứ tự lớn hơn mong đợi. Phát hiện có khoảng trống

Gửi ngay **ACK trùng lặp**, chỉ ra số thứ tự của byte mong đợi tiếp theo

Segment đến lấp đầy hoặc một phần khoảng trống

Gửi ngay ACK, với điều kiện là segment bắt đầu ngay tại điểm có khoảng trống

Tầng giao vận 3-70

Truyền lại nhanh trong TCP

❖ Chu kỳ time-out thường tương đối dài:

- Trễ dài trước khi gửi lại gói tin đã bị mất

❖ Phát hiện các segment bị mất qua các ACK bị trùng lặp.

- Bên gửi thường gửi nhiều segment song song
- Nếu segment, có thể sẽ có nhiều ACK bị trùng lặp.

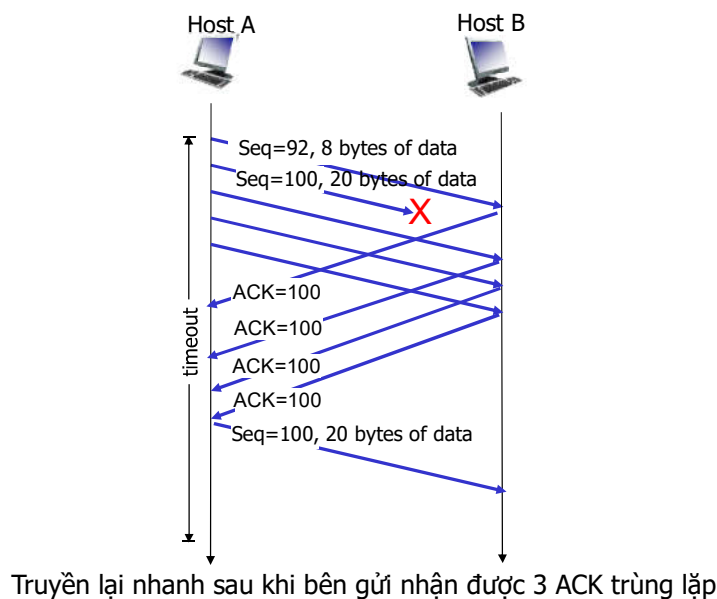
Truyền lại nhanh trong TCP

Nếu bên gửi nhận được 3 ACK trùng lặp cho cùng một dữ liệu ("Ba ACK trùng lặp"), thì sẽ gửi lại segment chưa được báo nhận có số thứ tự nhỏ nhất

- Có thể là đã bị mất segment chưa được báo nhận, nên không cần phải đợi đến timeout

Tầng giao vận 3-71

Truyền lại nhanh trong TCP



Tầng giao vận 3-72

Chương 3: Nội dung

3.1 Các dịch vụ tầng giao vận

3.2 Ghép kênh và phân kênh

3.3 Vận chuyển không kết nối: UDP

3.4 Các nguyên lý truyền dữ liệu tin cậy

3.5 Vận chuyển hướng kết nối: TCP

- Cấu trúc đoạn dữ liệu (segment)
- Truyền dữ liệu tin cậy
- Điều khiển luồng
- Quản lý kết nối

3.6 Các nguyên lý điều khiển tắc nghẽn

3.7 Điều khiển tắc nghẽn TCP

Tầng giao vận 3-73

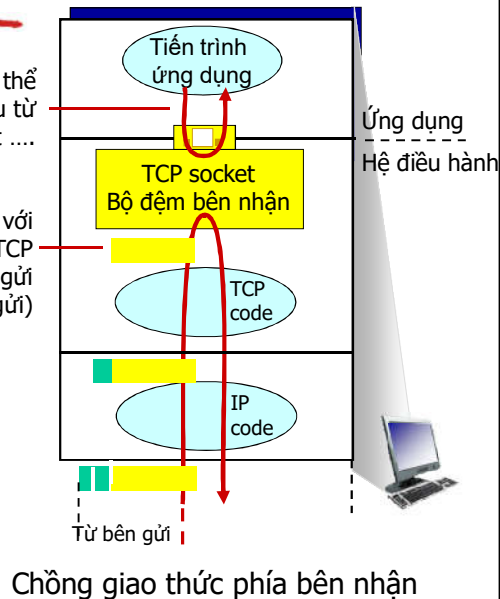
Điều khiển luồng trong TCP

Ứng dụng có thể di chuyển dữ liệu từ bộ đệm TCP socket

... chậm hơn so với bên nhận TCP (trong khi bên gửi vẫn đang gửi)

Điều khiển luồng

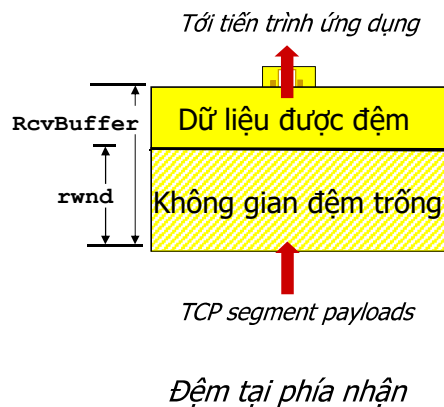
Bên nhận điều khiển bên gửi, để bên gửi sẽ không làm tràn bộ đệm của bên nhận do việc gửi quá nhanh và quá nhiều



Tầng giao vận 3-74

Điều khiển luồng trong TCP

- ❖ Bên nhận “thông báo” không gian đệm còn trống bởi giá trị **rwnd** trong TCP header của các segment gửi-nhận
 - Kích thước **RcvBuffer** được thiết lập qua tùy chọn (option) của socket (thường mặc định là 4096 byte)
 - Nhiều hệ điều hành tự động điều chỉnh **RcvBuffer**
- ❖ Bên gửi giới hạn tổng số dữ liệu chưa được báo nhận đến bên nhận theo giá trị **rwnd**
- ❖ Đảm bảo vùng đệm nhận không bị tràn



Tầng giao vận 3-75

Chương 3: Nội dung

3.1 Các dịch vụ tầng giao vận

3.2 Ghép kênh và phân kênh

3.3 Vận chuyển không kết nối: UDP

3.4 Các nguyên lý truyền dữ liệu tin cậy

3.5 Vận chuyển hướng kết nối: TCP

- Cấu trúc đoạn dữ liệu (segment)
- Truyền dữ liệu tin cậy
- Điều khiển luồng
- Quản lý kết nối

3.6 Các nguyên lý điều khiển tắc nghẽn

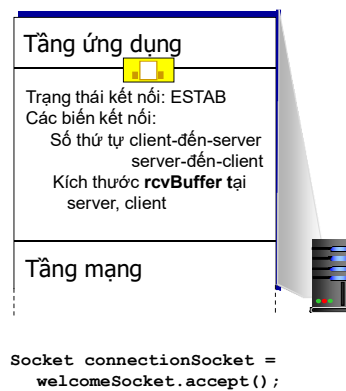
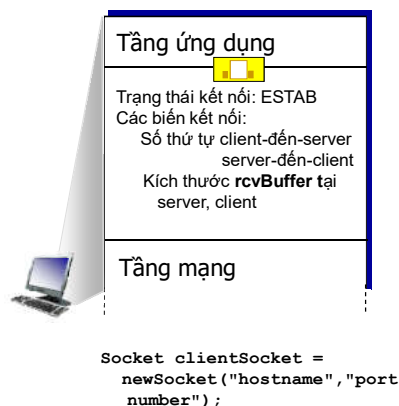
3.7 Điều khiển tắc nghẽn TCP

Tầng giao vận 3-76

Quản lý kết nối

Trước khi trao đổi dữ liệu, bên gửi/bên nhận “bắt tay”:

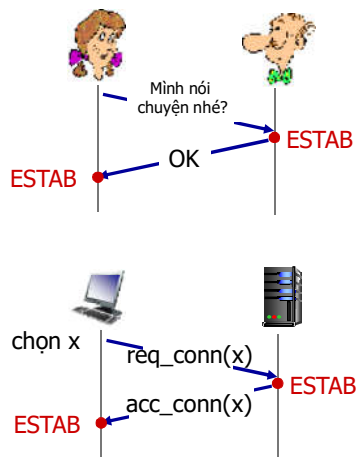
- ❖ Đồng ý thiết lập kết nối
- ❖ Đồng ý các tham số kết nối



Tầng giao vận 3-77

Đồng ý thiết lập kết nối

Bắt tay hai bước

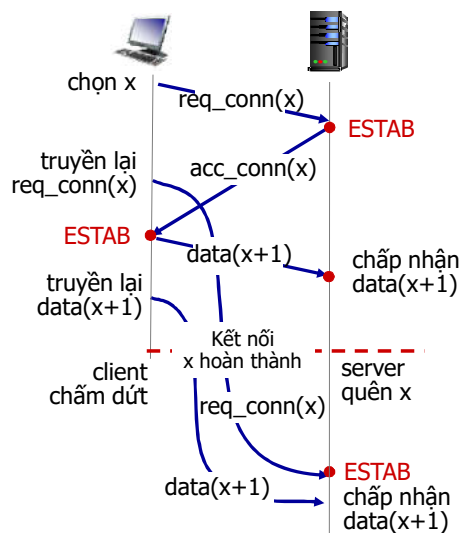
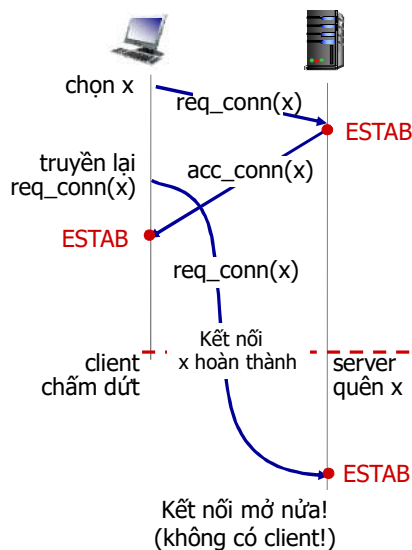


Hỏi: Bắt tay hai bước có luôn được thực hiện trên mạng?

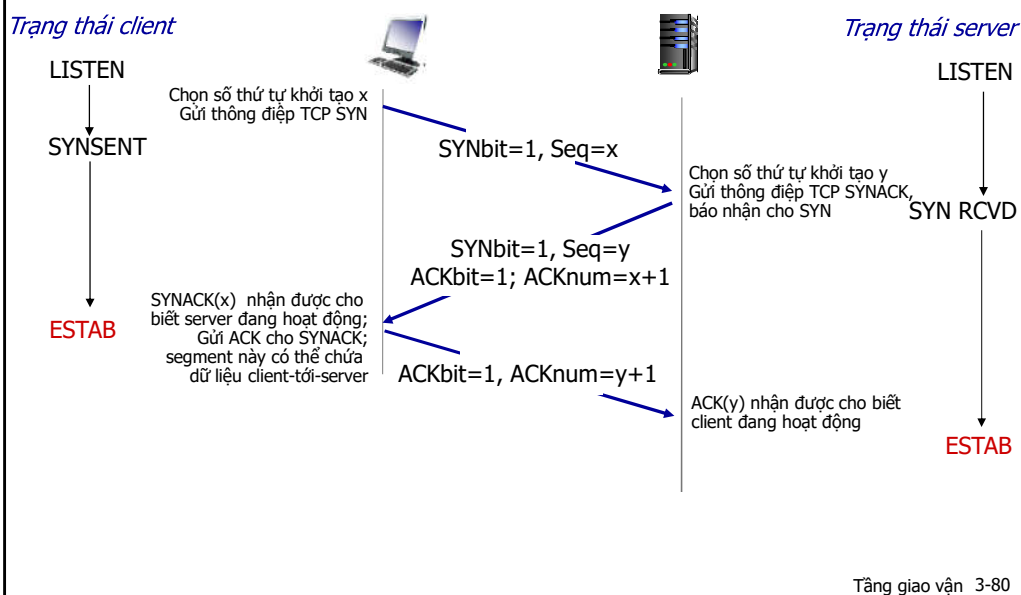
- ❖ Chậm trễ liên quan tới các biến
- ❖ Các thông điệp được truyền lại (ví dụ, `req_conn(x)`) là do mất thông điệp
- ❖ Sắp xếp lại thông điệp
- ❖ Không thể “nhìn thấy” phía bên kia

Đồng ý thiết lập kết nối

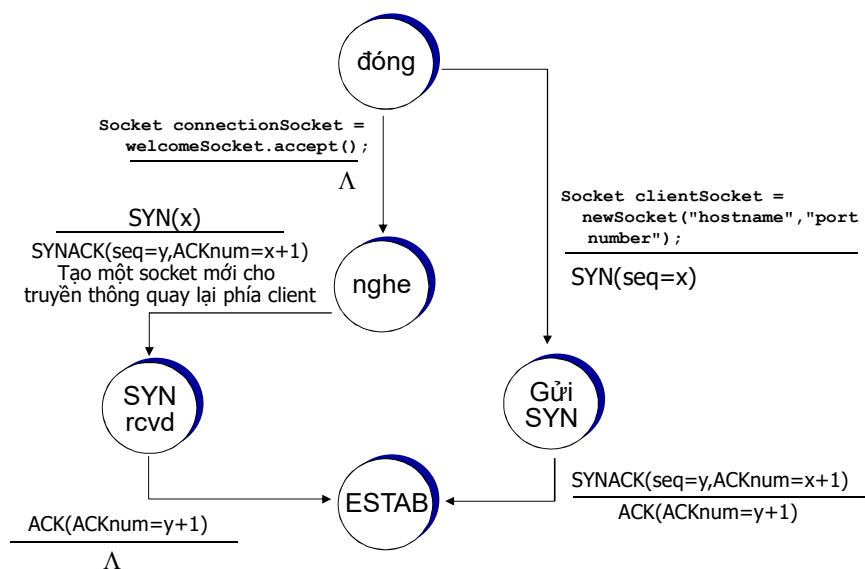
Các kịch bản có lỗi khi thực hiện bắt tay 2 bước:



Bắt tay 3 bước trong TCP



Bắt tay 3 bước trong TCP: FSM



TCP: đóng kết nối

- ❖ Mỗi bên client và server thực hiện đóng kết nối
 - Gửi TCP segment với bit FIN = 1
- ❖ Đáp ứng lại FIN nhận được bằng ACK
 - FIN, ACK đang nhận có thể được kết nối với FIN của nó
- ❖ Có thể thực hiện đồng bộ trao đổi FIN

Tầng giao vận 3-82

TCP: đóng kết nối

Trạng thái client

ESTAB

`clientSocket.close()`

FIN_WAIT_1

Không thể gửi nữa, nhưng vẫn có thể nhận dữ liệu

FIN_WAIT_2

Chờ server đóng

TIMED_WAIT

chờ một khoảng thời gian bằng $2 * \text{max thời gian sống của segment}$

CLOSED



Trạng thái server

ESTAB

CLOSE_WAIT

Vẫn có thể gửi dữ liệu

LAST_ACK

Không thể gửi dữ liệu thêm nữa

CLOSED

FINbit=1, seq=x

ACKbit=1; ACKnum=x+1

FINbit=1, seq=y

ACKbit=1; ACKnum=y+1

Tầng giao vận 3-83