Chương 3.
Tầng giao vận

1. Tổng quan về tầng giao vận

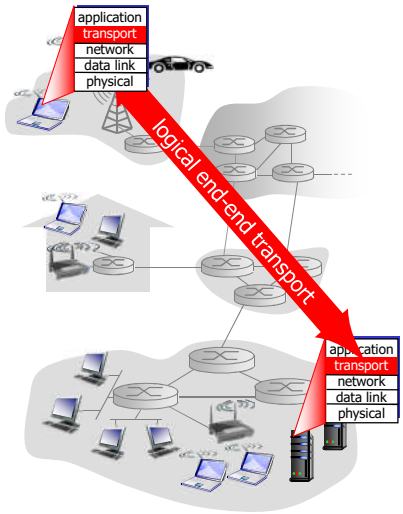
Nhắc lại kiến trúc phân tầng
Hướng liên kết vs. Không liên kết
UDP & TCP

Nhắc lại về kiến trúc phân tầng

Application (HTTP, Mail, …)	Hỗ trợ các ứng dụng trên mạng
Transport (UDP, TCP)	**Điều khiển truyền dữ liệu giữa các tiến trình của tầng ứng dụng**
Network (IP, ICMP…)	Chọn đường và chuyển tiếp gói tin giữa các máy, các mạng
Datalink (Ethernet, ADSL…)	Hỗ trợ việc truyền thông cho các thành phần kế tiếp trên cùng 1 mạng
Physical (bits…)	Truyền và nhận dòng bit trên đường truyền vật lý

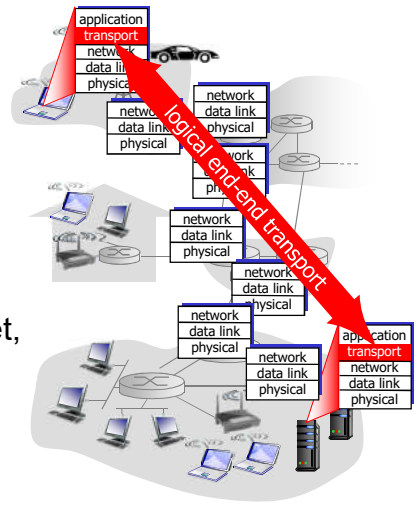
Tổng quan về tầng giao vận (1)

- Cung cấp phương tiện truyền giữa các ứng dụng cuối
- Bên gửi:
  - Nhận dữ liệu từ ứng dụng
  - Đặt dữ liệu vào các gói tin và chuyển cho tầng mạng
  - Nếu dữ liệu quá lớn, nó sẽ được chia làm nhiều phần và đặt vào nhiều đoạn tin khác nhau
- Bên nhận:
  - Nhận các đoạn tin từ tầng mạng
  - Tập hợp dữ liệu và chuyển lên cho ứng dụng



Tổng quan về tầng giao vận (2)

- Được cài đặt trên các hệ thống cuối
  - Không cài đặt trên các routers, switches…
- Hai dạng dịch vụ giao vận
  - Tin cậy, hướng liên kết, e.g TCP
  - Không tin cậy, không liên kết, e.g. UDP
- Đơn vị truyền: datagram (UDP), segment (TCP)



Tại sao lại cần 2 loại dịch vụ?

- Các yêu cầu đến từ tầng ứng dụng là đa dạng
- Các ứng dụng cần dịch vụ với 100% độ tin cậy như mail, web…
  - Sử dụng dịch vụ của TCP
- Các ứng dụng cần chuyển dữ liệu nhanh, có khả năng chịu lỗi, e.g. VoIP, Video Streaming
  - Sử dụng dịch vụ của UDP

Ứng dụng và dịch vụ giao vận

Ứng dụng	Giao thức ứng dụng	Giao thức giao vận
e-mail	SMTP	TCP
remote terminal access	Telnet	TCP
Web	HTTP	TCP
file transfer	FTP	TCP
streaming multimedia	giao thức riêng (e.g. RealNetworks)	TCP or UDP
Internet telephony	giao thức riêng (e.g., Vonage,Dialpad)	thường là UDP

2. Các chức năng chung

Dồn kênh/phân kênh

Mã kiểm soát lỗi

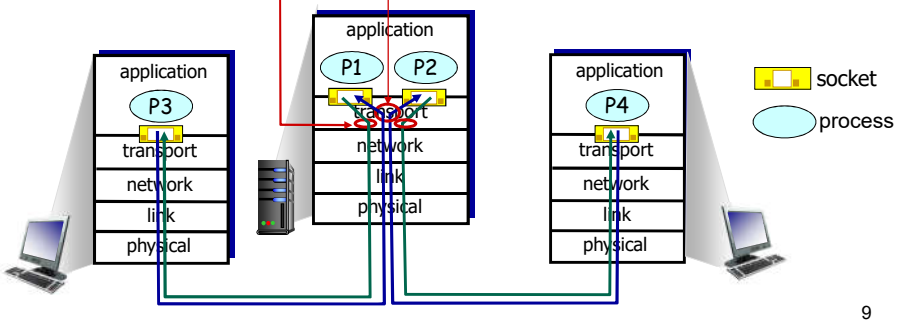
Dồn kênh/phân kênh - Mux/Demux

Gửi: Dồn kênh

Nhận dữ liệu từ các tiến trình tầng ứng dụng khác nhau (qua socket), đóng gói theo giao thức tầng giao vận và gửi trên liên kết mạng

Nhận: Phân kênh

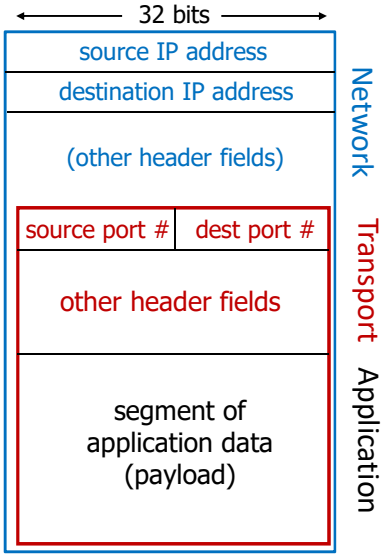
Sử dụng thông tin trên tiêu đề gói tin để gửi dữ liệu tới đúng socket



9

Mux/Demux hoạt động ntn?

- Nút mạng nhận gói tin với các địa chỉ:
  - Địa chỉ IP nguồn
  - Địa chỉ IP đích
  - Số hiệu cổng nguồn
  - Số hiệu cổng đích
- Địa chỉ IP và số hiệu cổng được sử dụng để xác định socket nhận dữ liệu



10

Checksum

- Phát hiện lỗi bit trong các đoạn tin/gói tin
- Gửi:*(nguyên lý chung)*
  - Chia dữ liệu thành các phần có kích thước n bit
  - Tính tổng các phần. Nếu kết quả tràn quá n bit, cộng các bit tràn vào phần kết quả
  - Đảo bit kết quả cuối cùng được checksum
  - Truyền checksum kèm theo dữ liệu
- Nhận:
  - Tách dữ liệu và checksum
  - Chia dữ liệu thành các phần có kích thước n bit
  - Tính tổng các phần và checksum. Nếu kết quả tràn quá n bit, cộng các bit tràn vào phần kết quả
  - Nếu kết quả cuối xuất hiện bit 0 → dữ liệu bị lỗi

3.UDP (User Datagram Protocol)

Tổng quan

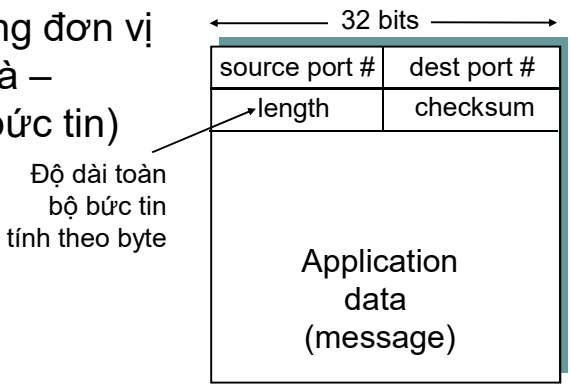
Khuôn dạng gói tin

Đặc điểm chung

- Giao thức hướng không kết nối (connectionless)
- Truyền tin "best-effort": chỉ gửi 1 lần, không phát lại
- Vì sao cần UDP?
  - Không cần thiết lập liên kết (giảm độ trễ)
  - Đơn giản: Không cần lưu lại trạng thái liên kết ở bên gửi và bên nhận
  - Phần đầu đoạn tin nhỏ
  - Không có quản lý tắc nghẽn: UDP cứ gửi dữ liệu nhanh nhất, nhiều nhất nếu có thể
- UDP có những chức năng cơ bản gì?
  - Dồn kênh/phân kênh
  - Phát hiện lỗi bit bằng checksum

Khuôn dạng bức tin (datagram)

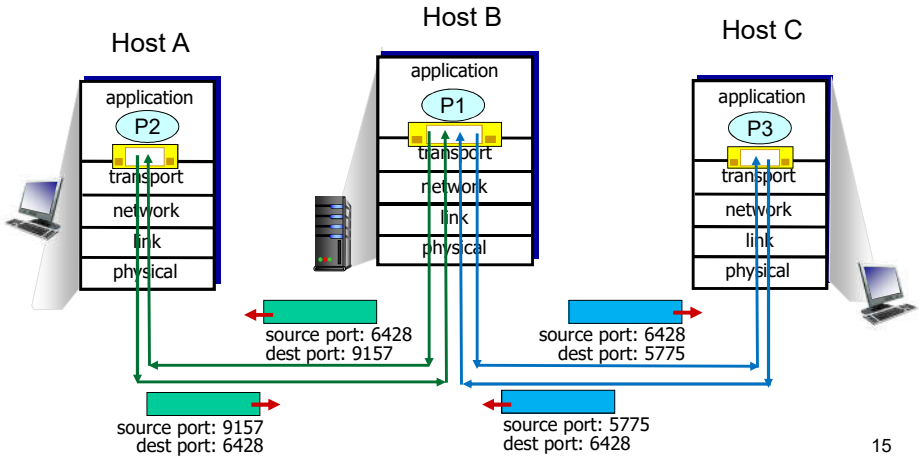
- UDP sử dụng đơn vị dữ liệu gọi là – datagram (bức tin)



Khuôn dạng đơn vị dữ liệu của UDP

mux/demux trên ứng dụng UDP

Mỗi tiến trình chỉ cần sử dụng một socket duy nhất để trao đổi dữ liệu với các tiến trình khác



Các vấn đề của UDP

- Không có kiểm soát tắc nghẽn
  - Làm Internet bị quá tải
- Không bảo đảm được 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

4. TCP (Transmission Control Protocol)

17

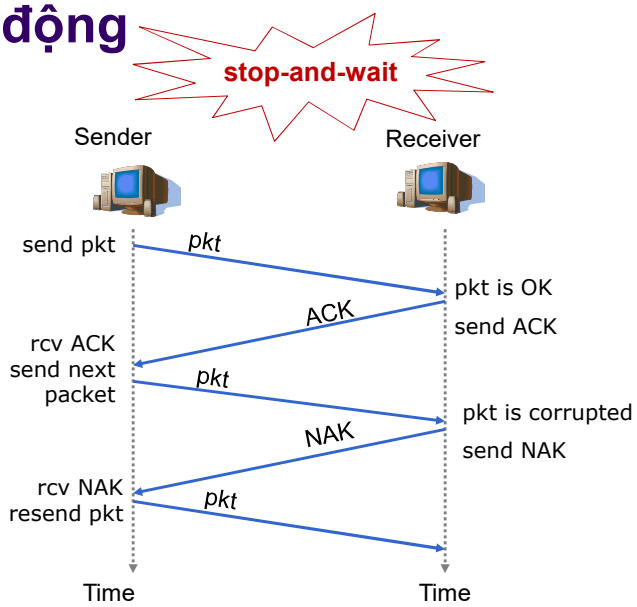
4.1.Khái niệm về truyền thông tin cậy

18

Kênh có lỗi bit, không bị mất tin

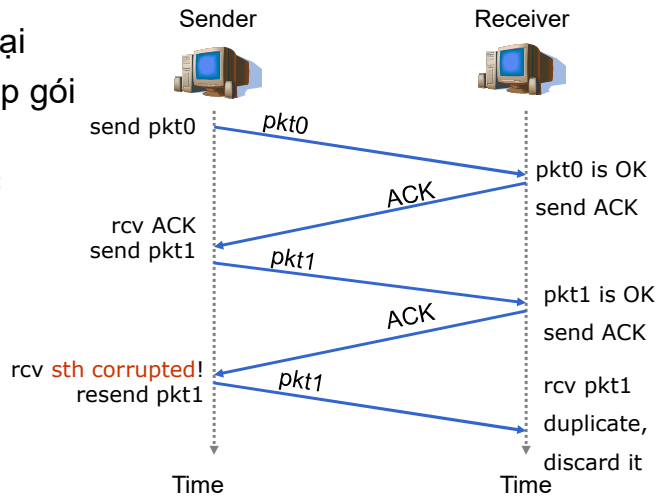
- Phát hiện lỗi?
  - Checksum
- Làm thế nào để báo cho bên gửi?
  - ACK (*acknowledgements*): gói tin được nhận thành công
  - NAK (*negative acknowledgements*): gói tin bị lỗi
- Phản ứng của bên gửi?
  - Truyền lại nếu là NAK

Hoạt động

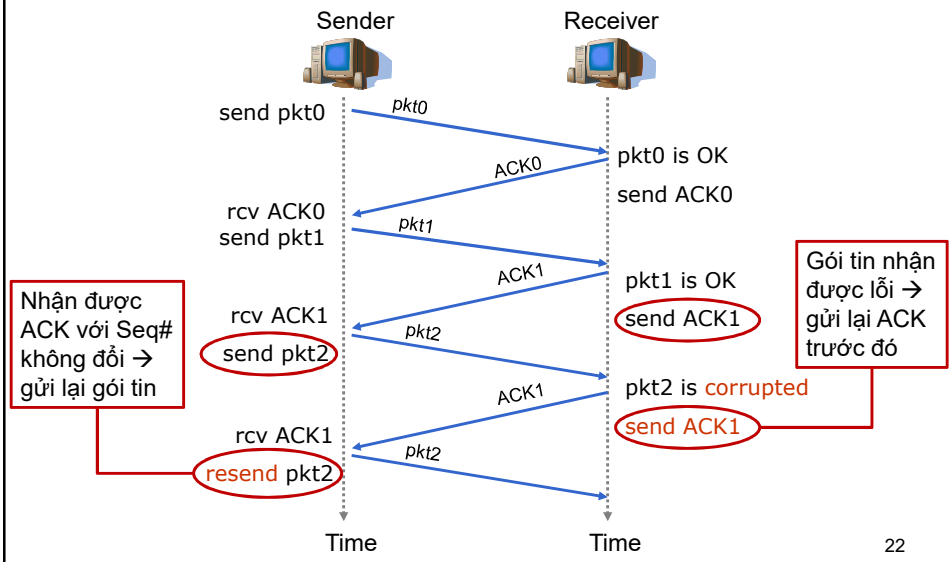


Lỗi ACK/NAK

- Cần truyền lại
- Xử lý việc lặp gói tin ntn?
- Thêm Seq.#



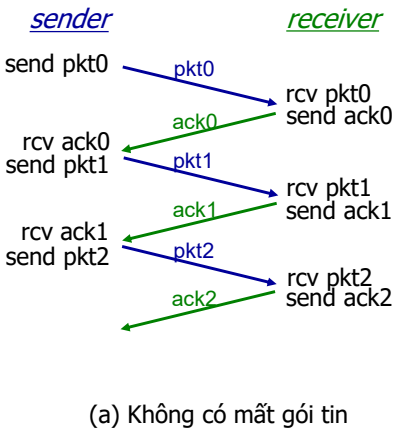
Giải pháp không dùng NAK



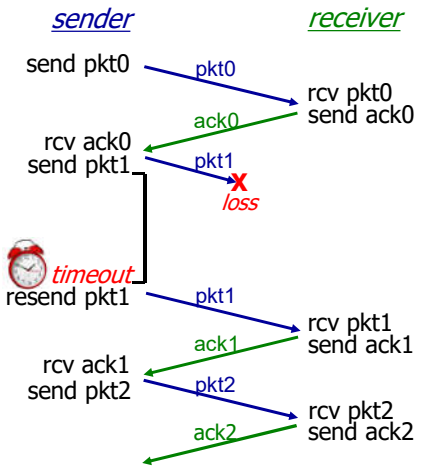
Kênh có lỗi bit và mất gói tin

- Dữ liệu và ACK có thể bị mất
  - Nếu không nhận được ACK?
  - Truyền lại như thế nào?
  - Timeout!
- 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?
  - Dùng số hiệu gói tin

Minh họa

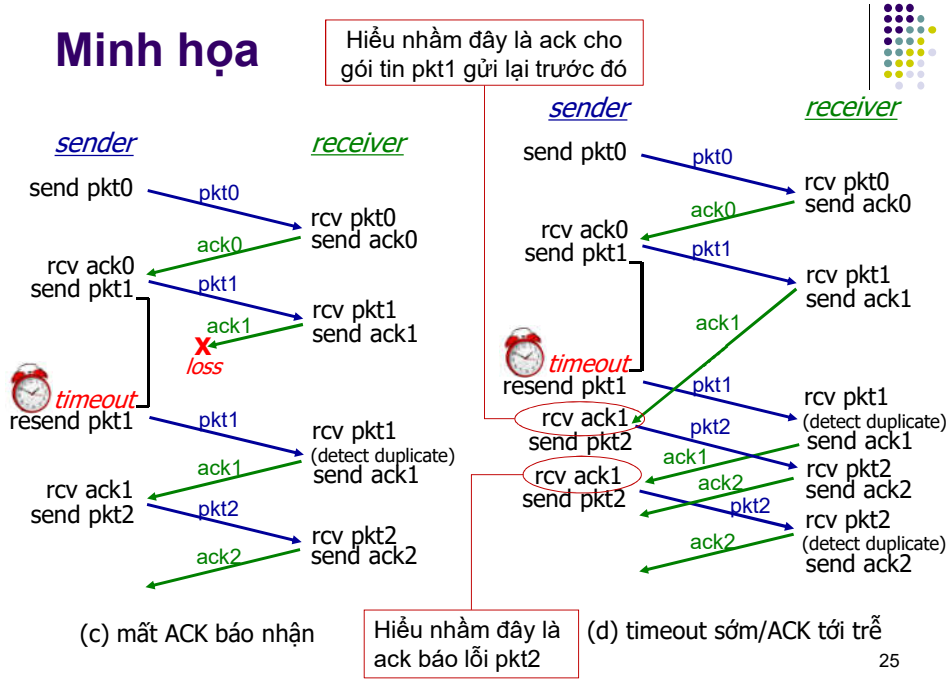


(a) Không có mất gói tin

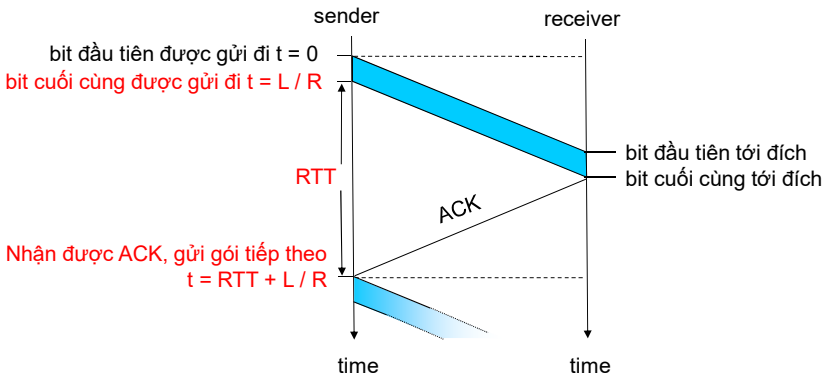


(b) mất gói tin gửi đi

Minh họa



Hiệu năng của stop-and-wait

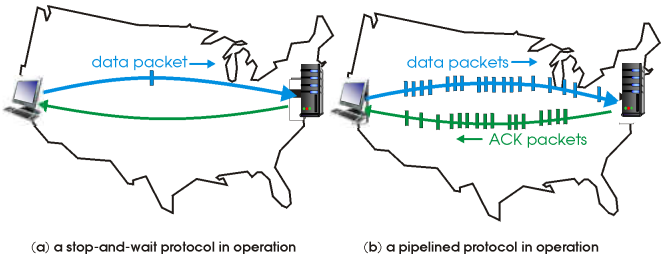


L: Kích thước gói tin
R: Băng thông
RTT: Round trip time

$$performance = \frac{L/R}{RTT + L/R}$$

Pipeline

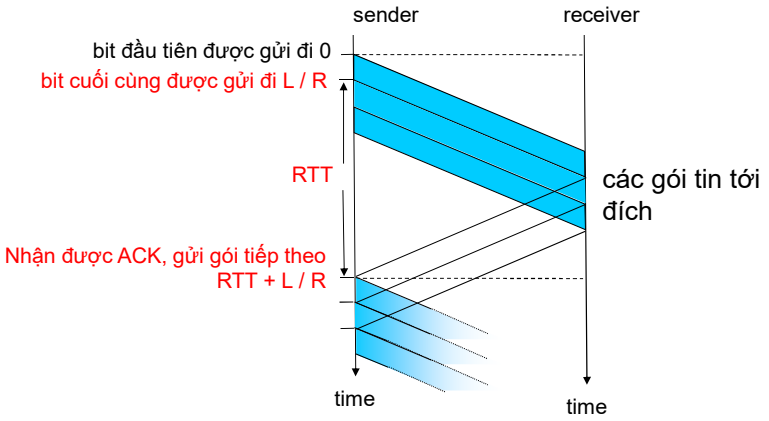
- Gửi liên tục một lượng hữu hạn các gói tin mà không cần chờ ACK
  - Số thứ tự các gói tin phải tăng dần
  - Dữ liệu gửi đi chờ sẵn ở bộ đệm gửi
  - Dữ liệu tới đích chờ ở bộ đệm nhận



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

27

Hiệu năng của pipeline

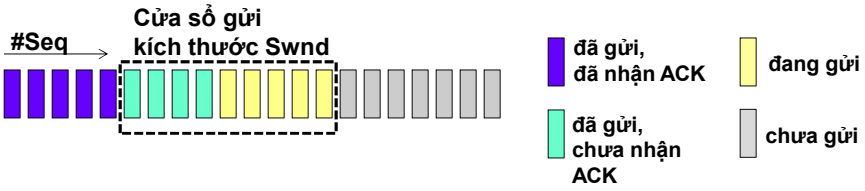


L: Kích thước gói tin
R: Băng thông
RTT: Round trip time
n: Số gói tin gửi liên tục

$$performance = \frac{n * L/R}{RTT + L/R}$$

28

Go-back-N



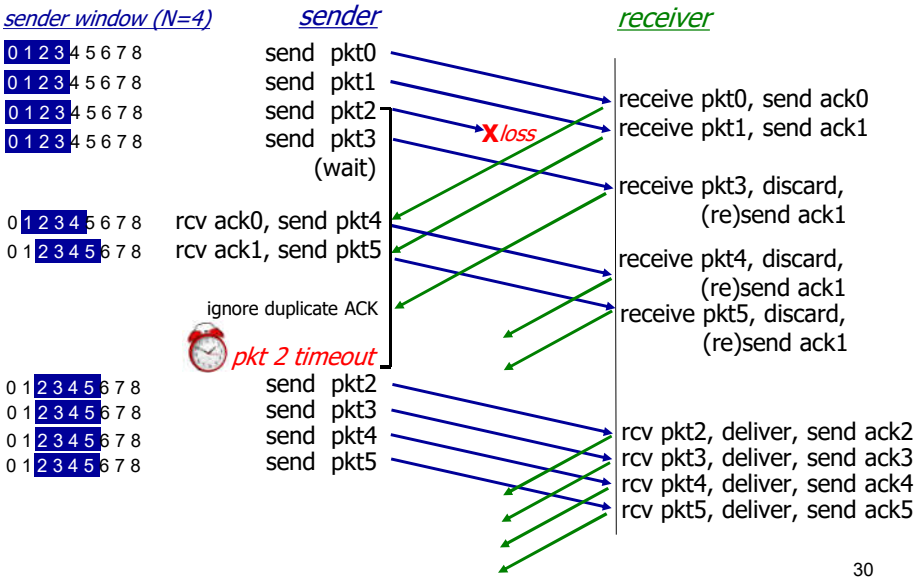
Bên gửi

- Chỉ gửi gói tin trong cửa sổ. Chỉ dùng 1 bộ đếm (timer) cho gói tin đầu tiên trong cửa sổ
- Nếu nhận được ACK_i, dịch cửa sổ sang vị trí (i+1). Đặt lại timer
- Nếu timeout cho gói tin pkt_i gửi lại tất cả gói tin trong cửa sổ

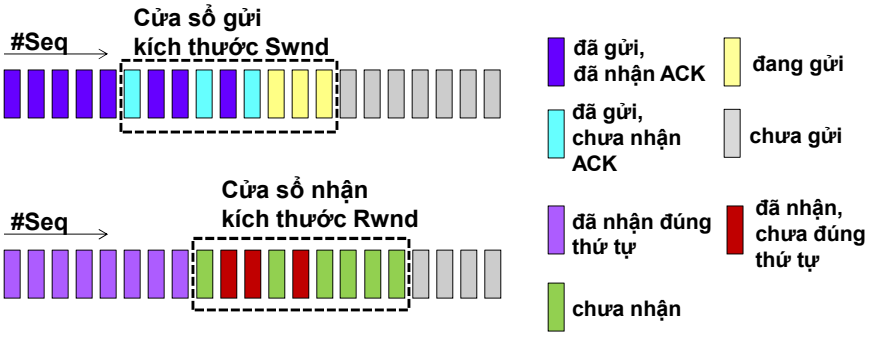
Bên nhận

- Gửi ACK_i cho gói tin pkt_i đã nhận được theo thứ tự
- Gói tin đến không theo thứ tự: hủy gói tin và gửi lại ACK của gói tin gần nhất còn đúng thứ tự

Go-back-N



Selective Repeat



- Gửi: chỉ gửi gói tin trong cửa sổ gửi
- Nhận: chỉ nhận gói tin trong cửa sổ nhận
  - Sử dụng bộ đệm để lưu tạm thời các gói tin tới chưa đúng thứ tự

Selective Repeat

Bên gửi

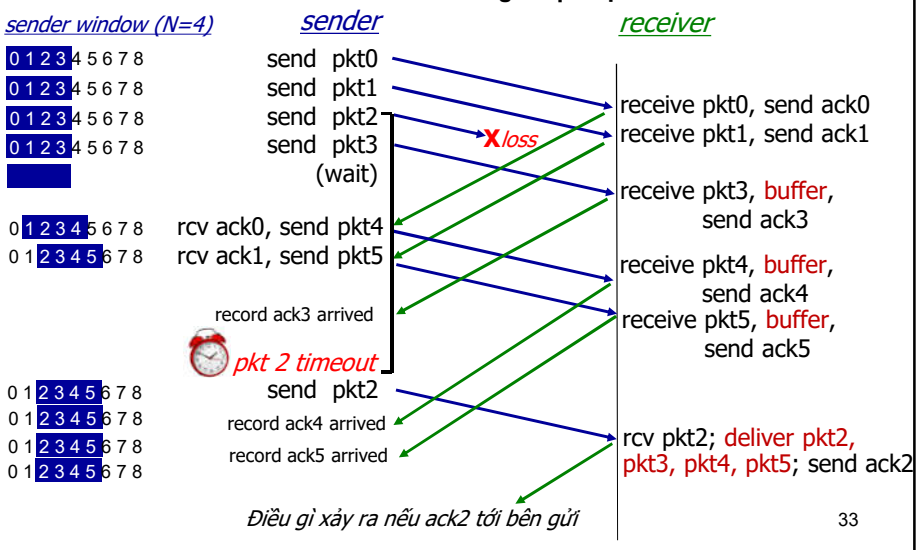
- Chỉ gửi gói tin trong cửa sổ gửi
- Dùng 1 timer cho mỗi gói tin trong cửa sổ
- Nếu timeout cho gói tin pkt_i chỉ gửi lại pkt_i
- Nhận được ACK_i:
  - Đánh dấu pkt_i đã có ACK
  - Nếu i là giá trị nhỏ nhất trong các gói tin chưa nhận ACK, dịch cửa sổ sang vị trí gói tin tiếp theo chưa nhận ACK

Bên nhận

- Chỉ nhận gói tin trong cửa sổ nhận
- Nhận pkt_i:
  - Gửi lại ACK_i
  - Không đúng thứ tự: đưa vào bộ đệm
  - Đúng thứ tự: chuyển cho tầng ứng dụng cùng với các gói tin trong bộ đệm đã trở thành đúng thứ tự sau khi nhận pkt_i

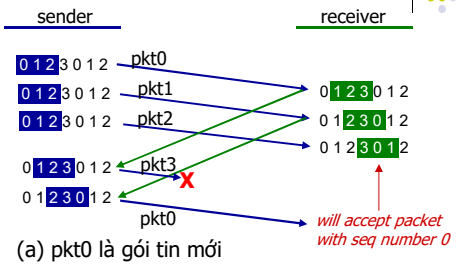
Selective Repeat

Điều gì xảy ra nếu kích thước cửa sổ lớn hơn ½ giá trị Seq# lớn nhất?

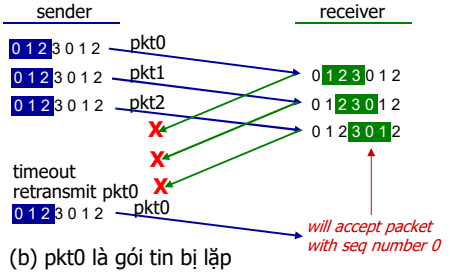


Kích thước cửa sổ quá lớn

- Giả sử Seq# = {0, 1 , 2 ,3}
- Kích thước cửa sổ: 3
- Phía nhận không phân biệt được 2 trường hợp
- Trong trường hợp b, gói tin pkt0 gửi lại được bên nhận coi như gói tin mới, đưa vào bộ đệm chờ xử lý



(a) pkt0 là gói tin mới



(b) pkt0 là gói tin bị lặp

4.2. Hoạt động của TCP

Cấu trúc đoạn tin TCP
Quản lý liên kết
Kiểm soát luồng
Kiểm soát tắc nghẽn

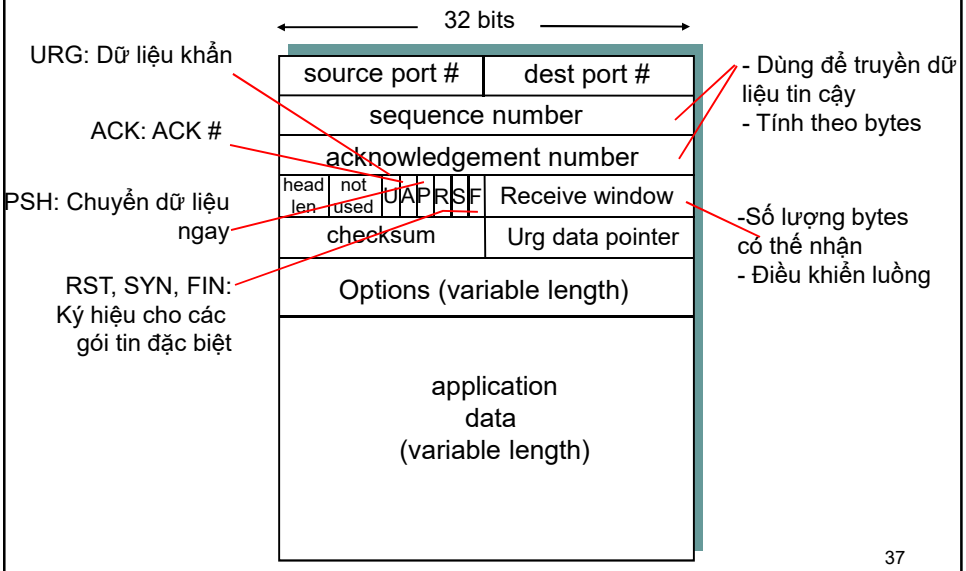
35

Tổng quan về TCP

- Giao thức hướng liên kết
  - Bắt tay ba bước
- Giao thức truyền dữ liệu theo dòng byte, tin cậy
  - Sử dụng vùng đệm
- Truyền theo kiểu pipeline
  - Tăng hiệu quả
- Kiểm soát luồng
  - Bên gửi không làm quá tải bên nhận
- Kiểm soát tắc nghẽn
  - Việc truyền dữ liệu không nên làm tắc nghẽn mạng

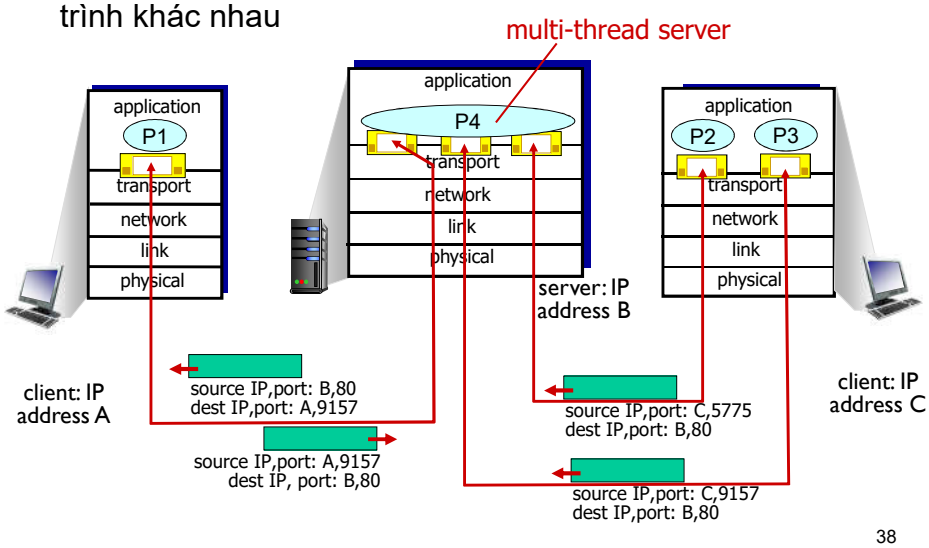
36

Khuôn dạng đoạn tin - TCP segment



mux/demux trên ứng dụng TCP

Sử dụng socket khác nhau để trao đổi với các tiến trình khác nhau



Thông số của liên kết TCP

- Mỗi một liên kết TCP giữa hai tiến trình được xác định bởi bộ 4 thông số (4-tuple):
  - Địa chỉ IP nguồn
  - Địa chỉ IP đích

  } Tầng mạng
  - Số hiệu cổng nguồn
  - Số hiệu cổng đích

  } Tầng giao vận

39

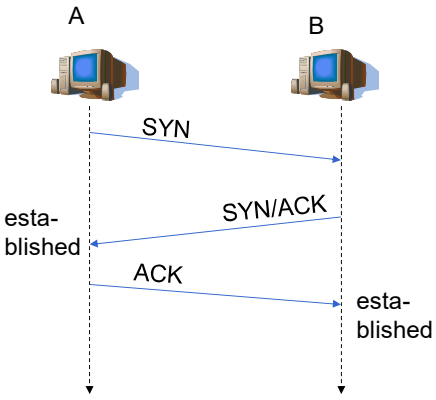
TCP cung cấp dịch vụ tin cậy ntn?

- Kiểm soát lỗi dữ liệu: checksum
- Kiểm soát mất gói tin: phát lại khi có time-out
- Kiểm soát dữ liệu đã được nhận chưa:
  - Seq. #
  - Ack

  } Cơ chế báo nhận
- Chu trình làm việc của TCP:
  - Thiết lập liên kết
    - Bắt tay ba bước
  - Truyền/nhận dữ liệu: có thể thực hiện đồng thời(duplex) trên liên kết
  - Đóng liên kết

40

Thiết lập liên kết TCP : Giao thức bắt tay 3 bước



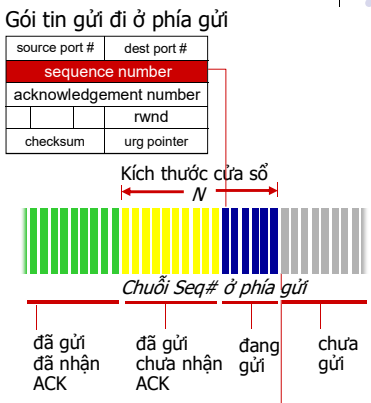
- Bước 1: A gửi SYN cho B
  - chỉ ra giá trị khởi tạo seq # của A
  - không có dữ liệu
- Bước 2: B nhận SYN, trả lời bằng SYN/ACK
  - B khởi tạo vùng đệm
  - chỉ ra giá trị khởi tạo seq. # của B
- Bước 3: A nhận SYNACK, trả lời ACK, có thể kèm theo dữ liệu

Tại sao không dùng giao thức bắt tay 2 bước

Cơ chế báo nhận trong TCP

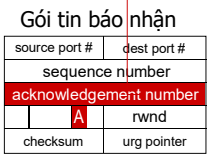
sequence numbers:

- Vị trí của byte đầu tiên trên payload của gói tin (segment) trong luồng dữ liệu



acknowledgements:

- Vị trí của byte tiếp theo muốn nhận trong luồng dữ liệu



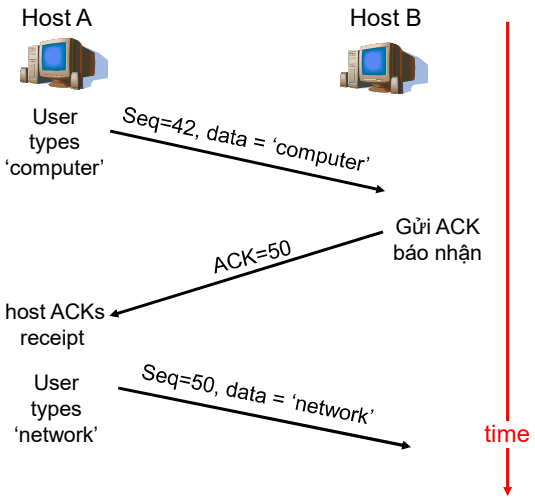
Cơ chế báo nhận trong TCP

Seq. #:

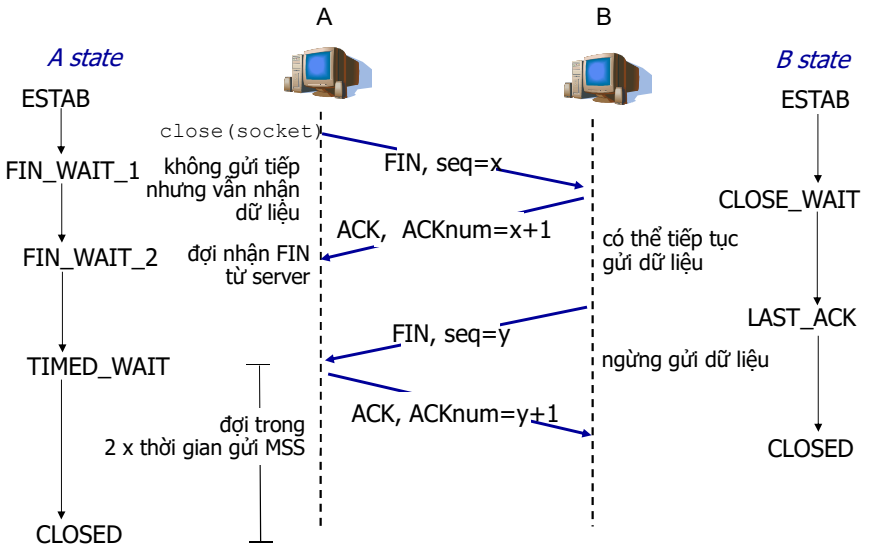
- Số hiệu của byte đầu tiên của đoạn tin trong dòng dữ liệu

ACK:

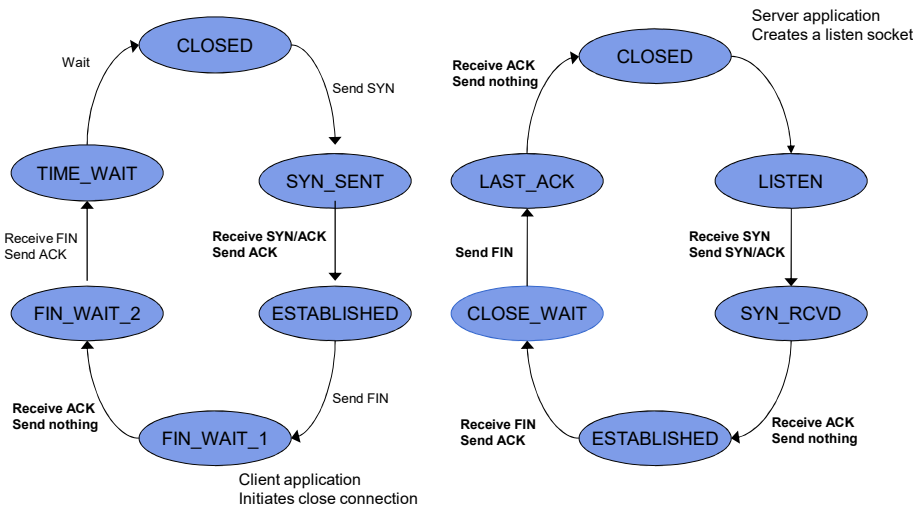
- Số hiệu byte đầu tiên mong muốn nhận từ đối tác



Đóng liên kết



Chu trình sống của TCP (đơn giản hóa)



Pipeline trong TCP

Go-back-N hay ***Selective Repeat***?

- Bên gửi:
  - *Nếu nhận được ACK# = i thì coi tất cả gói tin trước đó đã tới đích (ngay cả khi chưa nhận được các ACK# < i). Dịch cửa sổ sang vị trí i*
  - ***Nếu có timeout của gói tin Seq# = i chỉ gửi lại gói tin đó***
- Bên nhận:
  - ***Đưa vào bộ đệm các gói tin không đúng thứ tự và gửi ACK***

→ thuật toán lai

TCP: Hoạt động của bên gửi

Nhận dữ liệu từ tầng ứng dụng

- Đóng gói dữ liệu vào gói tin TCP với giá trị Seq# tương ứng
- Tính toán và thiết lập giá trị ***TimeOutInterval*** cho bộ đếm thời gian (timer)
- Gửi gói tin TCP xuống tầng mạng và khởi động bộ đếm cho gói đầu tiên trong cửa sổ

timeout:

- Gửi lại gói tin bị timeout
- Khởi động lại bộ đếm

Nhận ACK# = i

- Nếu là ACK cho gói tin nằm bên trái cửa sổ → bỏ qua
- Ngược lại, trượt cửa sổ sang vị trí i
- Khởi động timer cho gói tin kế tiếp đang chờ ACK

Tính toán timeout

- Dựa trên giá trị RTT (> 1 RTT)
  - Nhưng RTT thay đổi theo từng lượt gửi
  - Timeout quá dài: hiệu năng giảm
  - Timeout quá ngắn: không đủ thời gian để ACK báo về
- Ước lượng RTT

$$\mathbf{EstimatedRTT_i} = \alpha\mathbf{*EstimatedRTT_{i-1}} + \mathbf{(1-}\alpha\mathbf{)*SampleRTT_{i-1}}$$

  - **EstimatedRTT:** RTT ước lượng
  - **SampleRTT:** RTT đo được
  - $0 < \alpha < 1$: Jacobson đề nghị $\alpha = 0.875$

Tính toán timeout

- Độ lệch:

$$DevRTT_i = (1-\beta)*DevRTT_{i-1} + \beta*|SampleRTT_{i-1} - EstimatedRTT_{i-1}|$$

  - Jacobson đề nghị $\beta = 0.25$

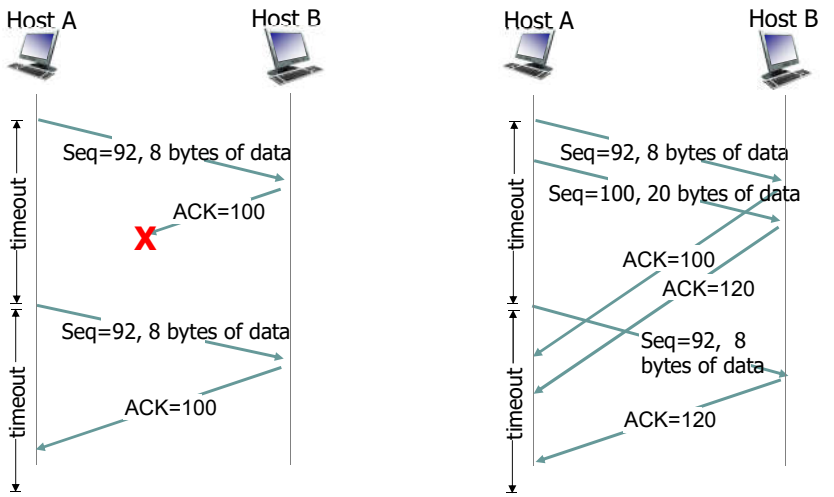
- Timeout:

$$TimeOutInterval_i = EstimatedRTT_i + 4*DevRTT_i$$

Ước lượng RTT – Ví dụ

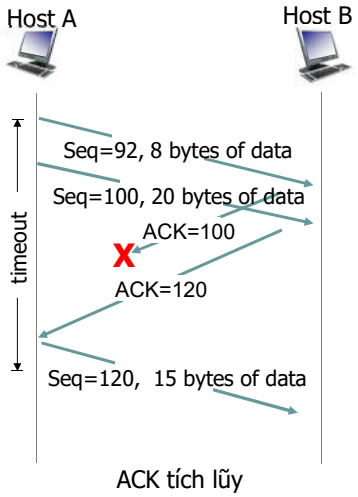
Packet#	EstimatedRTT(ms)	DevRTT(ms)	TimeoutInterval(ms)	SampleRTT(ms)
1	40	20	120	80
2	45	25	145	15
3				
4				
5			?	

Phát lại như thế nào?



51

Phát lại như thế nào? (tiếp)



52

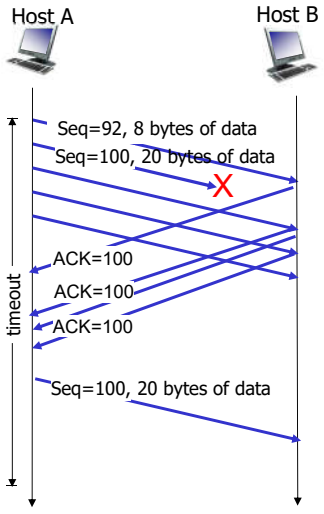
Hoạt động của bên nhận

Sự kiện	*Hành động*
Nhận 1 gói tin với Seq# = i đúng thứ tự	Đợi 500ms, nếu không có gói kế tiếp, gửi ACK = i+Kích thước dữ liệu nhận được
Nhận 1 gói tin với Seq# = i đúng thứ tự, trong khi chưa gửi ACK gói tin đã nhận thành công trước đó	Gửi ACK tích lũy ACK# = i + Kích thước dữ liệu nhận được
Nhận 1 gói tin với Seq# = i đúng thứ tự, trong bộ đệm còn gói tin với Seq# = i+N	Gửi ACK# = i + N + Kích thước dữ liệu nhận được trên gói tin N
Nhận gói tin không đúng thứ tự	Hồi phục nhanh

53

Hồi phục nhanh

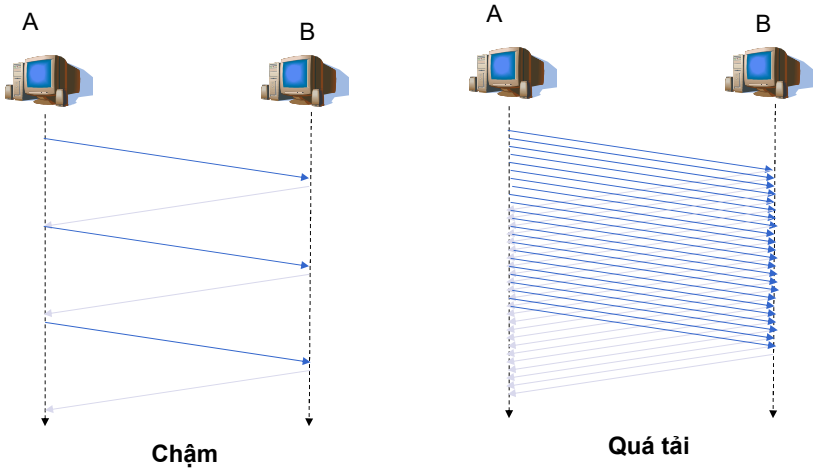
- Thời gian timeout khá dài có thể làm giảm hiệu năng
- Cơ chế hồi phục nhanh:
  - *Bên nhận:* Khi nhận gói tin không đúng thứ tự, gửi liên tiếp 2 gói tin lặp lại ACK# của gói tin còn đúng thứ tự trước đó
  - *Bên gửi:* Nhận được 3 ACK# liên tiếp giống nhau, gửi lại ngay gói tin mà không chờ time-out



54

3.3. Kiểm soát luồng

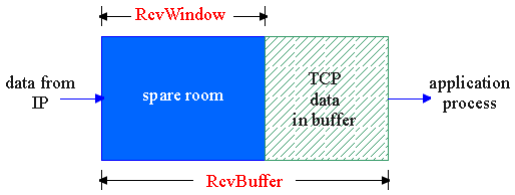
Kiểm soát luồng (1)



Kiểm soát luồng (2)

- Điều khiển lượng dữ liệu được gửi đi
  - Bảo đảm rằng hiệu quả là tốt
  - Không làm quá tải các bên
- Các bên sẽ có cửa sổ kiểm soát
  - Rwnd: Cửa sổ nhận
  - Cwnd: Cửa sổ kiểm soát tắc nghẽn
- Lượng dữ liệu gửi đi phải nhỏ hơn min(Rwnd, Cwnd)

Kiểm soát luồng trong TCP

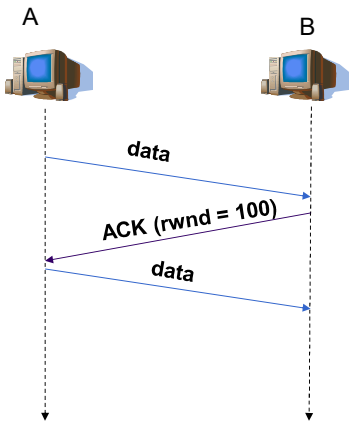


- Kích thước vùng đệm trống

= Rwnd

= **RcvBuffer-[LastByteRcvd - LastByteRead]**

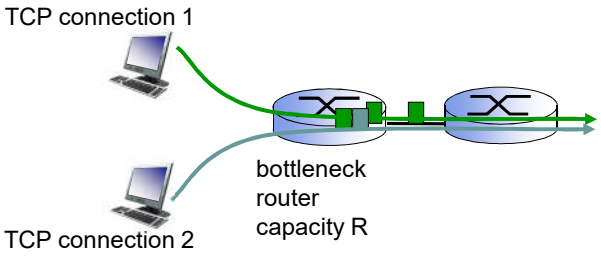
Trao đổi thông tin về Rwnd



- Bên nhận sẽ báo cho bên gửi biết Rwnd trong các đoạn tin
- Bên gửi đặt kích thước cửa sổ gửi theo Rwnd

Tính công bằng trong TCP

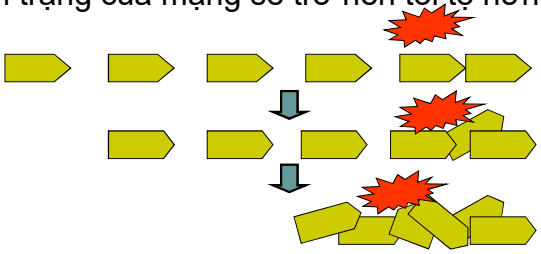
- Nếu có K kết nối TCP chia sẻ đường truyền có băng thông R thì mỗi kết nối có tốc độ truyền trung bình là R/K



4.4. Điều khiển tắc nghẽn trong TCP

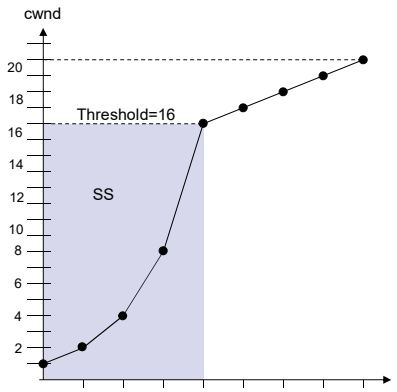
Tổng quan về tắc nghẽn

- Khi nào tắc nghẽn xảy ra ?
  - Quá nhiều cặp gửi-nhận trên mạng
  - Truyền quá nhiều làm cho mạng quá tải
- Hậu quả của việc nghẽn mạng
  - Mất gói tin
  - Thông lượng giảm, độ trễ tăng
  - Tình trạng của mạng sẽ trở nên tồi tệ hơn.



Nguyên lý kiểm soát tắc nghẽn

- Slow-start
  - Tăng tốc độ theo hàm số mũ
  - Tiếp tục tăng đến một ngưỡng nào đó
- Tránh tắc nghẽn
  - Tăng dẫn tốc độ theo hàm tuyến tính cho đến khi phát hiện tắc nghẽn
- Phát hiện tắc nghẽn
  - Gói tin bị mất



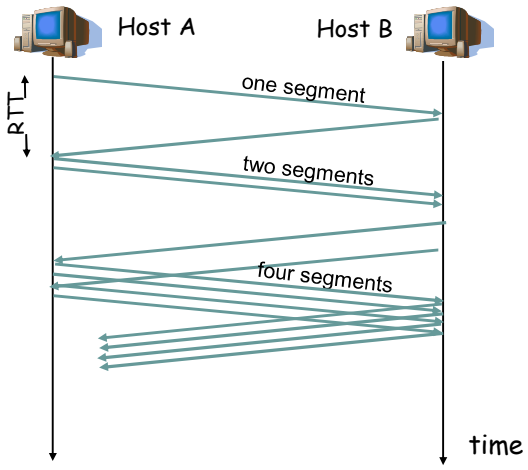
63

TCP Slow Start (1)

- Ý tưởng cơ bản
  - Đặt cwnd bằng 1 MSS (Maximum segment size)
    - 1460 bytes (giá trị này có thể được thỏa thuận trong quá trình thiết lập liên kết)
  - Tăng cwnd lên gấp đôi
    - Khi nhận được ACK
  - Bắt đầu chậm, nhưng tăng theo hàm mũ
- Tăng cho đến một ngưỡng: ssthresh
  - Sau đó, TCP chuyển sang trạng thái tránh tắc nghẽn

64

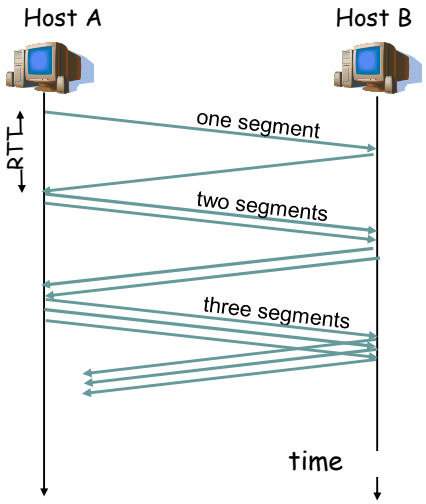
TCP Slow Start (2)



65

Tránh tắc nghẽn - Congestion avoidance

- ý tưởng cơ bản
  - Tăng cwnd theo cấp số cộng sau khi nó đạt tới ssthresh
  - Khi bên gửi nhận được ACK
    - Tăng cwnd thêm 1 MSS



66

Phản ứng của TCP (1)

- Giảm tốc độ gửi
- Phát hiện tắc nghẽn?
  - Nếu như phải truyền lại
    - Có thể đoán mạng đang có "tắc nghẽn"
- Khi nào thì phải truyền lại?
  - Timeout!
  - Nhận được nhiều ACK cùng ACK#

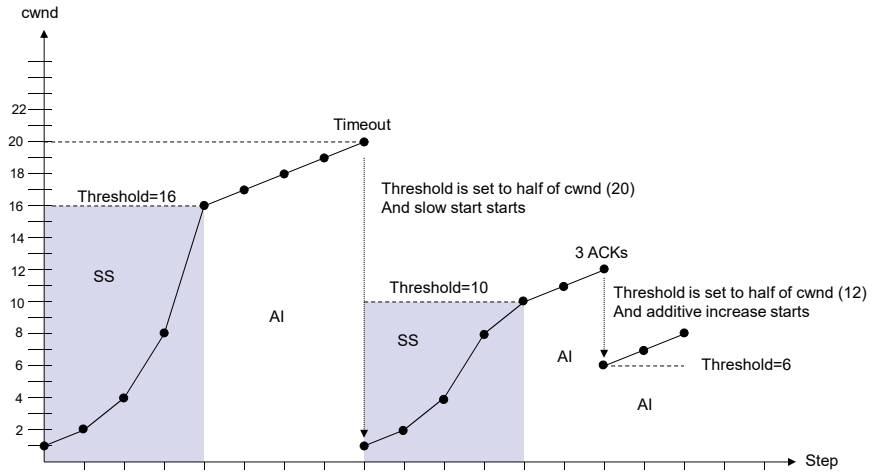
67

Phản ứng của TCP (2)

- Khi có timeout của bên gửi
  - TCP đặt ngưỡng ssthresh xuống còn một nửa giá trị hiện tại của cwnd
  - TCP đặt cwnd về 1 MSS
  - TCP chuyển về slow start
- Hồi phục nhanh:
  - Nút nhận: nhận được 1 gói tin không đúng thứ tự thì gửi liên tiếp 3 ACK giống nhau.
  - Nút gửi: nhận được 3 ACK giống nhau
    - TCP đặt ngưỡng ssthresh xuống còn một nửa giá trị hiện tại của cwnd
    - TCP đặt cwnd về giá trị hiện tại của ngưỡng mới
    - TCP chuyển trạng thái "congestion avoidance"

68

Kiểm soát tắc nghẽn – minh họa



Tổng kết

- Có hai dạng giao thức giao vận
  - UDP và TCP
  - Best effort vs. reliable transport protocol
- Các cơ chế bảo đảm độ tin cậy
  - Báo nhận
  - Truyền lại
  - Kiểm soát luồng và kiểm soát tắc nghẽn

Tài liệu tham khảo

- Keio University
- “Computer Networking: A Top Down Approach”, J.Kurose
- “Computer Network”, Berkeley University

71