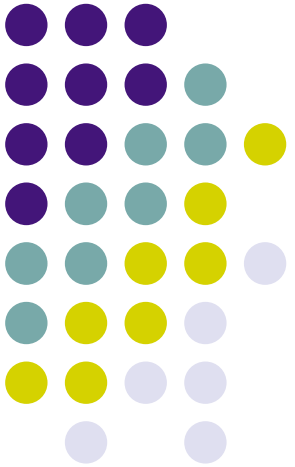
Tầng giao vận

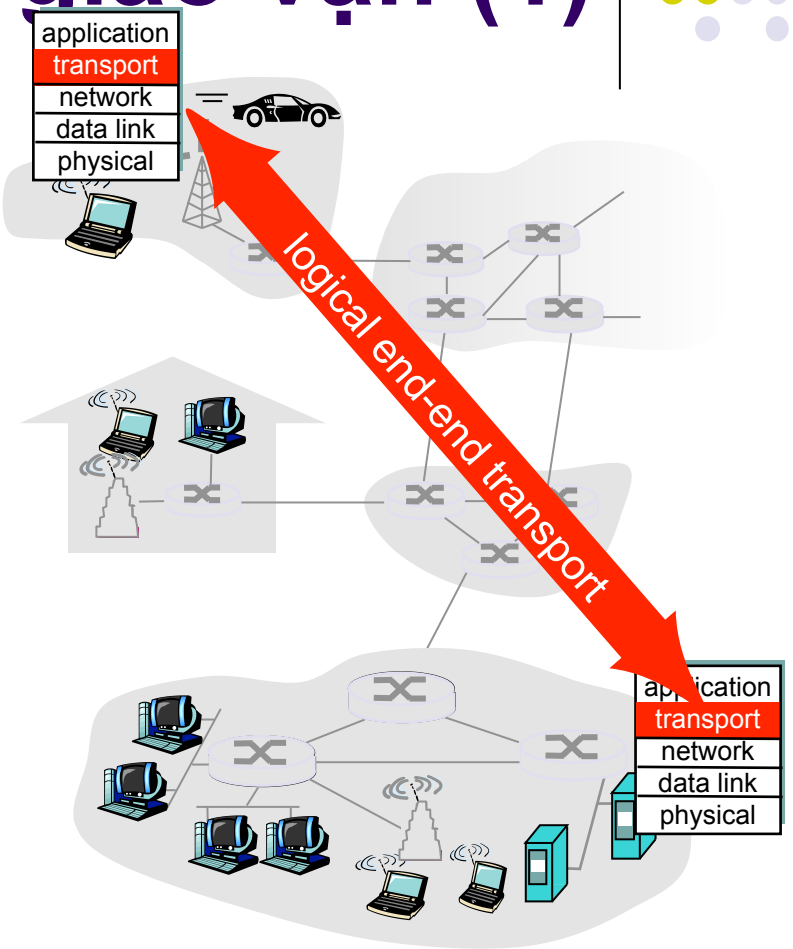


Nhắc lại về kiến trúc phân tầng

Application (HTTP, Mail, …)	Hỗ trợ các ứng dụng trên mạng
Transport (UDP, TCP …)	**Truyền dữ liệu giữa các 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

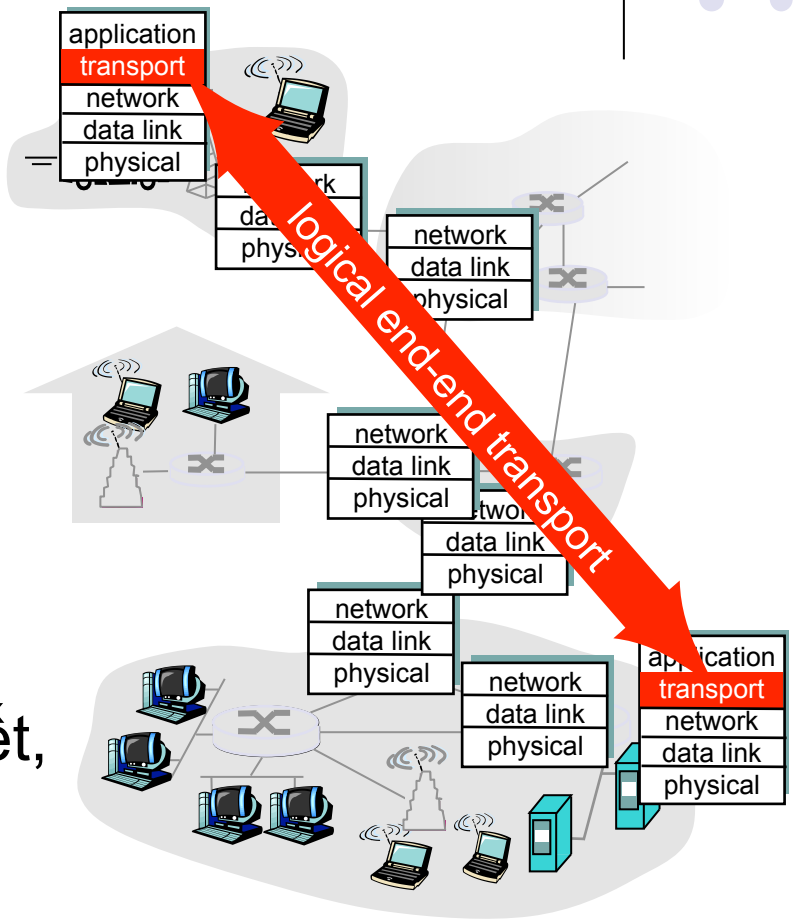
Tổng quan về tầng giao vận (1)

- Cung cấp phương tiện truyền giữa các ứng dụng cuối
- Bên gửi:
  - Nhận dữ liệu từ ứng dụng
  - Đặt dữ liệu vào các đoạn 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

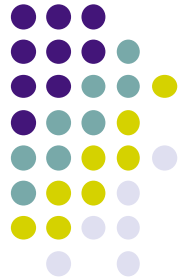


Tổng quan về tầng giao vận (2)

- 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



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

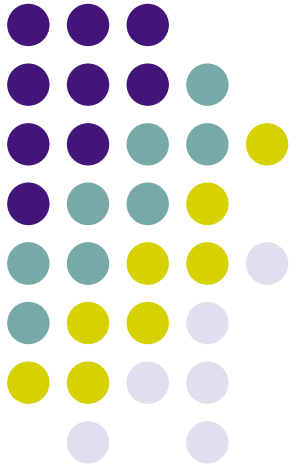
Ứng dụng và dịch vụ giao vận

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

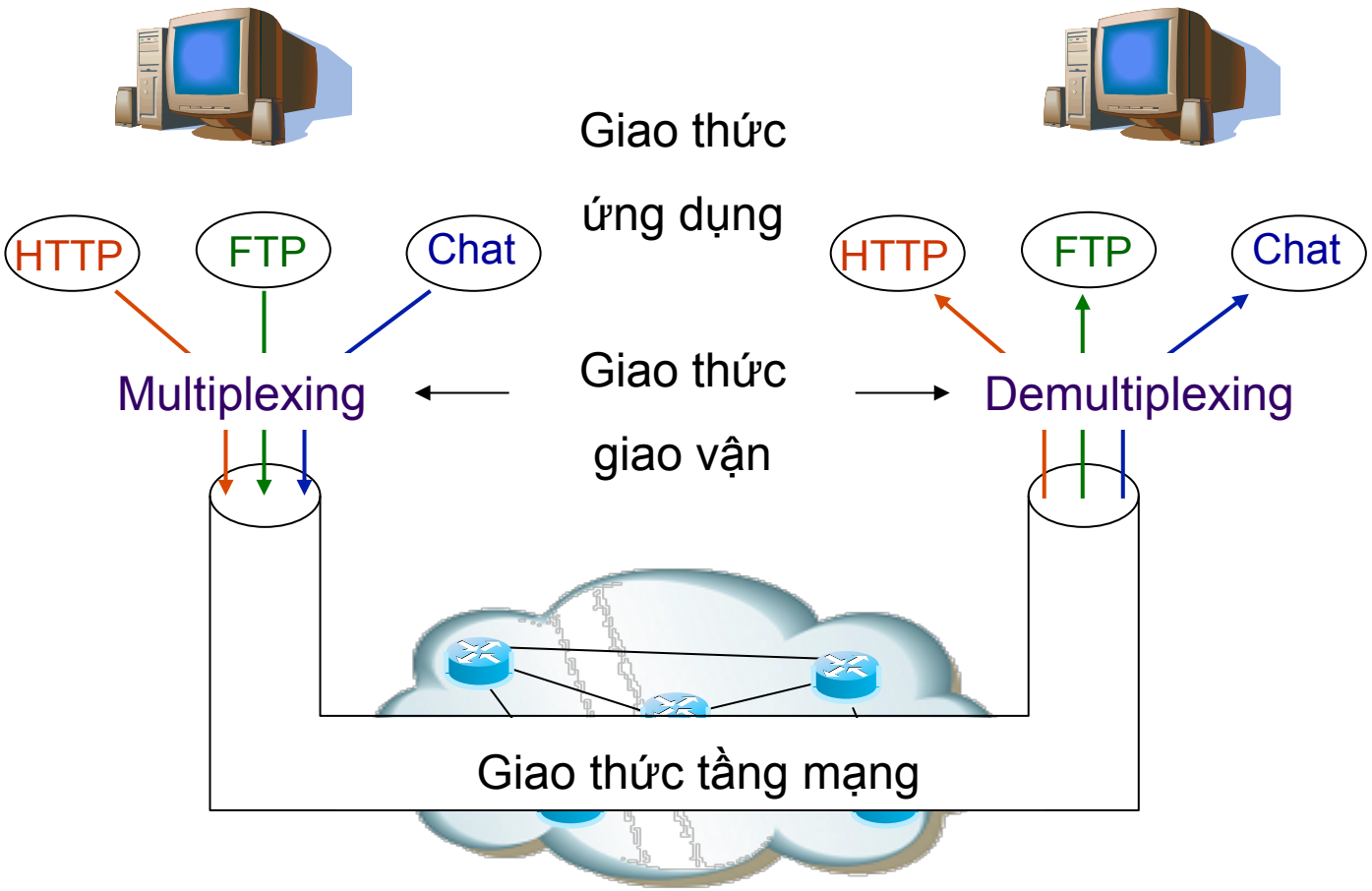
Các chức năng chung

Dồn kênh/phân kênh

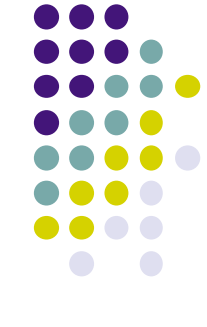
Mã kiểm soát lỗi



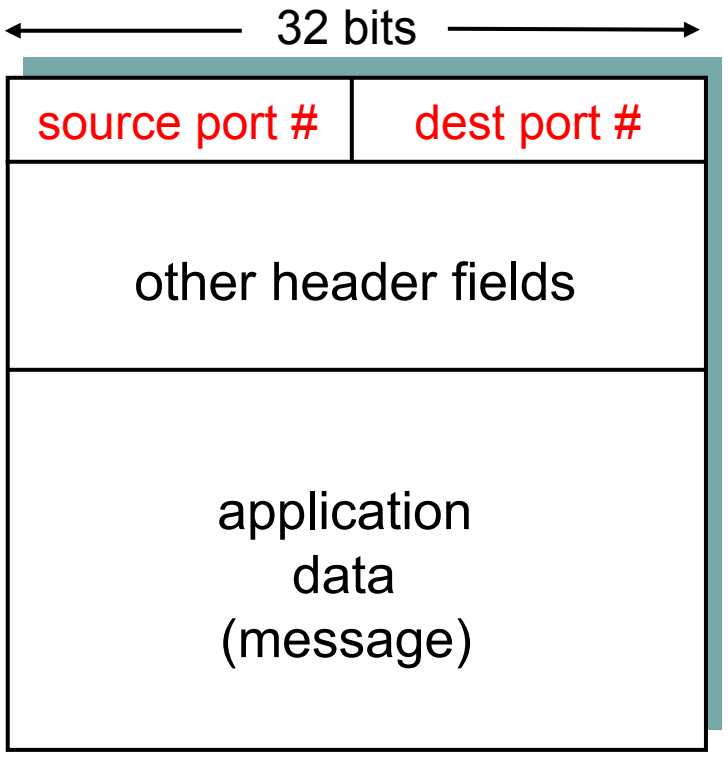
Dồn kênh/phân kênh - Mux/Demux



Mux/Demux hoạt động ntn?

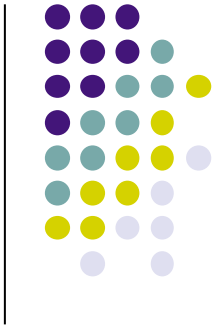


- Tại tầng mạng, gói tin IP được định danh bởi địa chỉ IP
  - Để xác định máy trạm
- Làm thế nào để phân biệt các ứng dụng trên cùng một máy?
  - Sử dụng số hiệu cổng (16 bits)
  - Mỗi tiến trình ứng dụng được gán 1 cổng
- Socket: Một cặp địa chỉ IP và số hiệu cổng



TCP/UDP segment format

Checksum



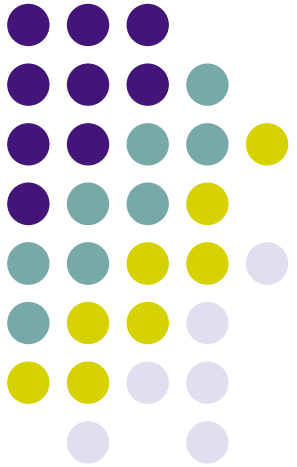
- Phát hiện lỗi bit trong các đoạn tin/gói tin
- Nguyên lý giống như checksum (16 bits) của giao thức IP
- Ví dụ:

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
Tổng	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
Checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

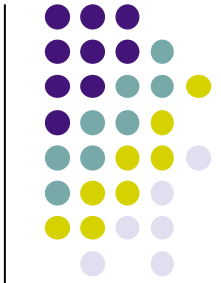
UDP
User Datagram Protocol

Tổng quan

Khuôn dạng gói tin

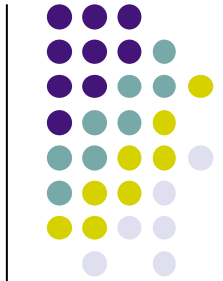


Giao thức dạng "Best effort"

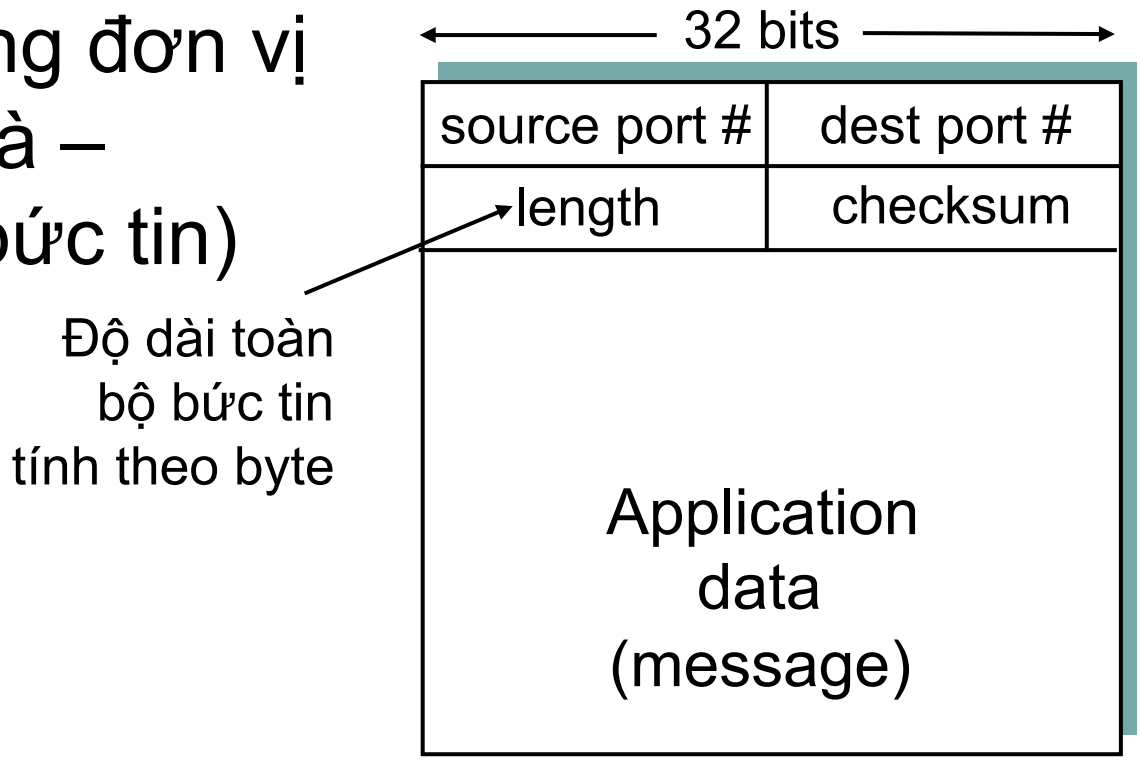


- Vì sao cần UDP?
  - Không cần thiết lập liên kết (tăng độ trễ)
  - Đơn giản: Không cần lưu lại trạng thái liên kết ở bên gửi và bên nhận
  - Phần đầu đoạn tin nhỏ
  - Không có quản lý tắc nghẽn: UDP cứ gửi dữ liệu nhanh nhất, nhiều nhất nếu có thể
- UDP có những chức năng cơ bản gì?
  - Dồn kênh/phân kênh
  - Phát hiện lỗi bit bằng checksum

Khuôn dạng bức tin (datagram)

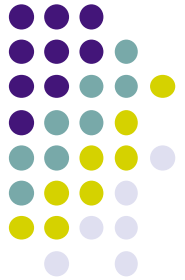


- UDP sử dụng đơn vị dữ liệu gọi là – datagram (bức tin)



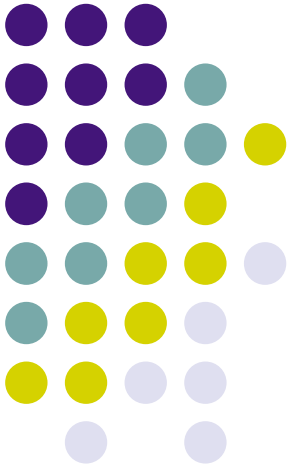
Khuôn dạng đơn vị dữ liệu của UDP

Các vấn đề của UDP

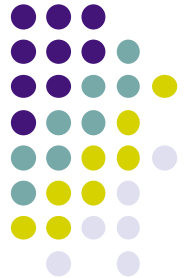


- Không có kiểm soát tắc nghẽn
  - Làm Internet bị quá tải
- Không bảo đảm được độ tin cậy
  - Các ứng dụng phải cài đặt cơ chế tự kiểm soát độ tin cậy
  - Việc phát triển ứng dụng sẽ phức tạp hơn

Khái niệm về truyền thông tin cậy

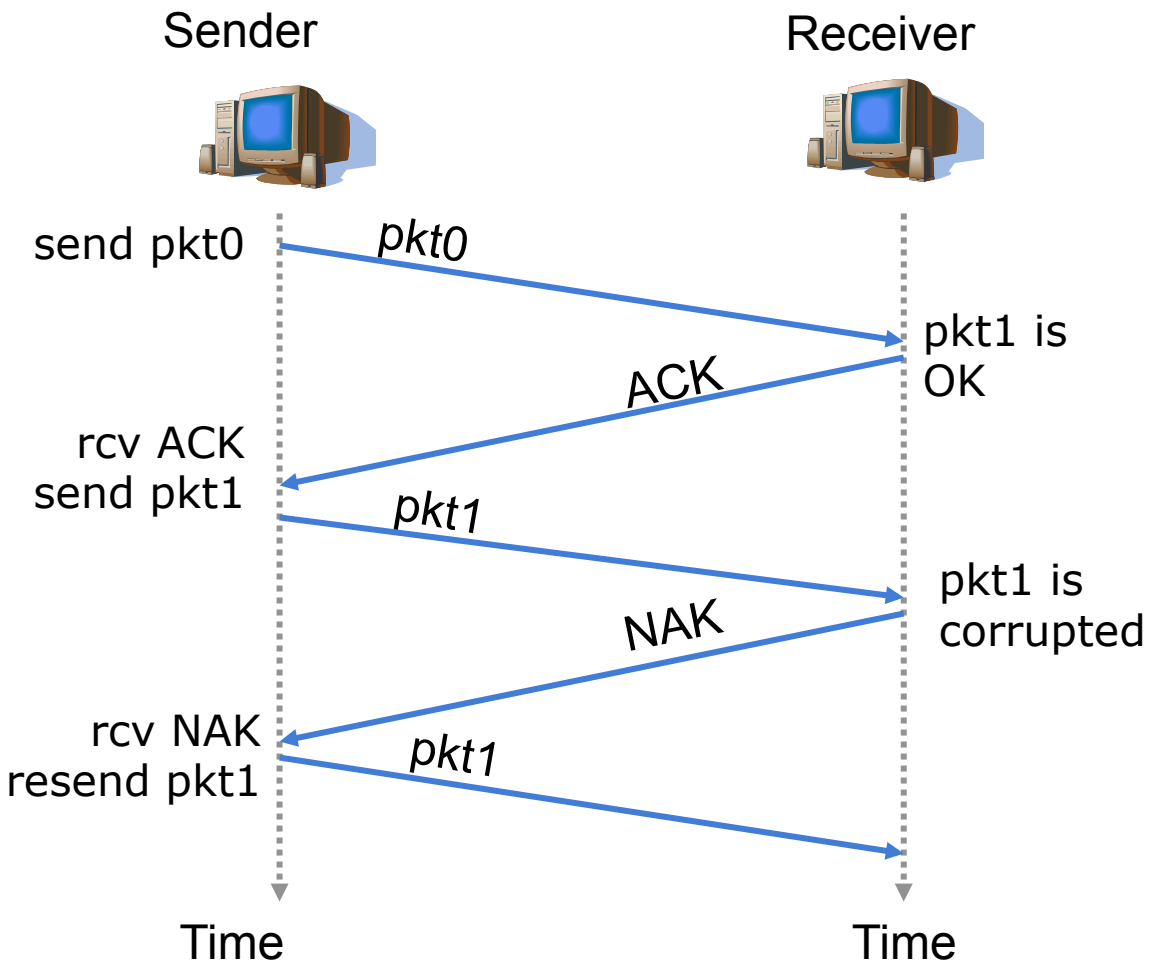


Kênh có lỗi bit, không bị mất tin



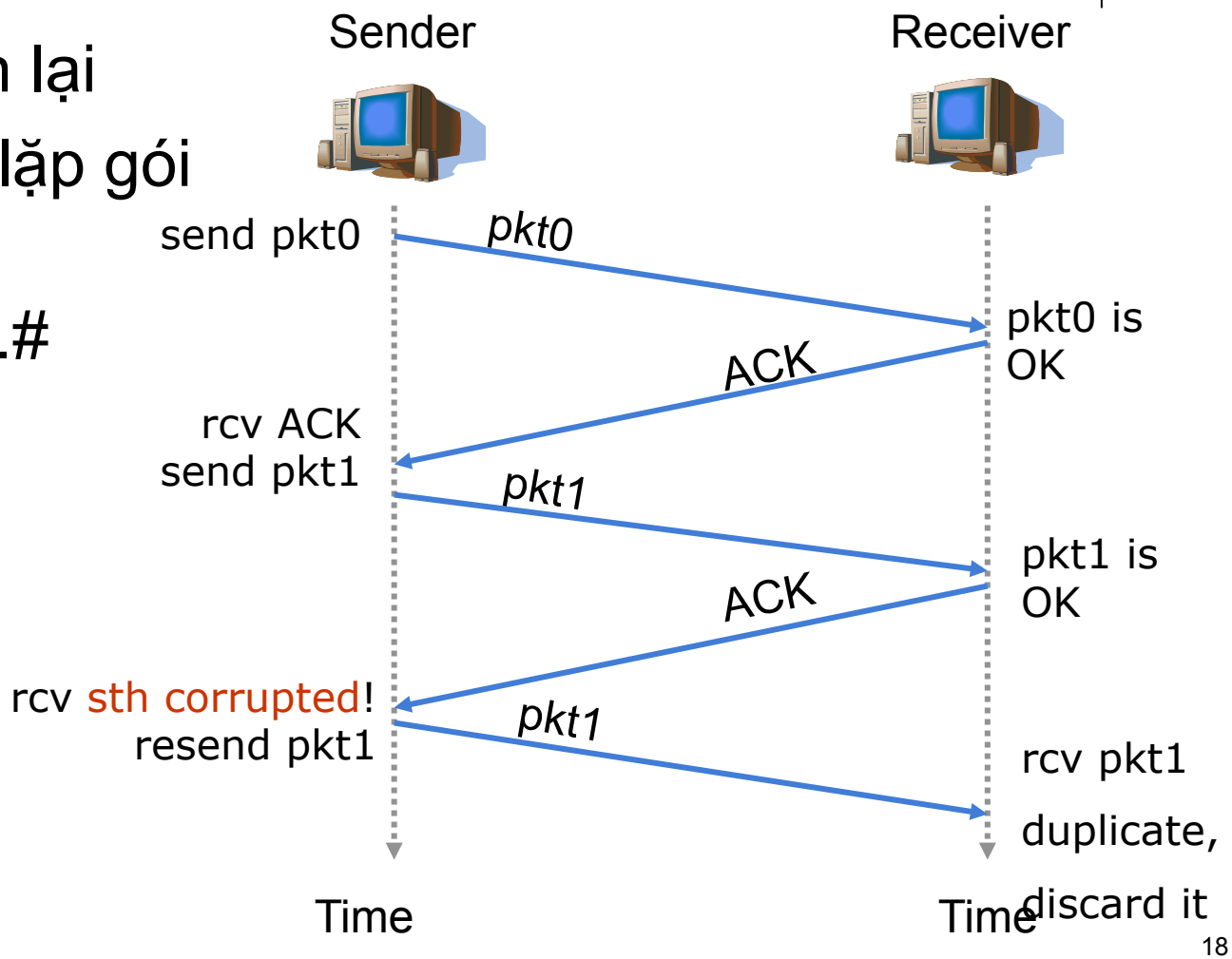
- Phát hiện lỗi?
  - Checksum
- Làm thế nào để báo cho bên gửi?
  - ACK (*acknowledgements*):
  - NAK (*negative acknowledgements*): tell sender that pkt has error
- Phản ứng của bên gửi?
  - Truyền lại nếu là NAK

Hoạt động

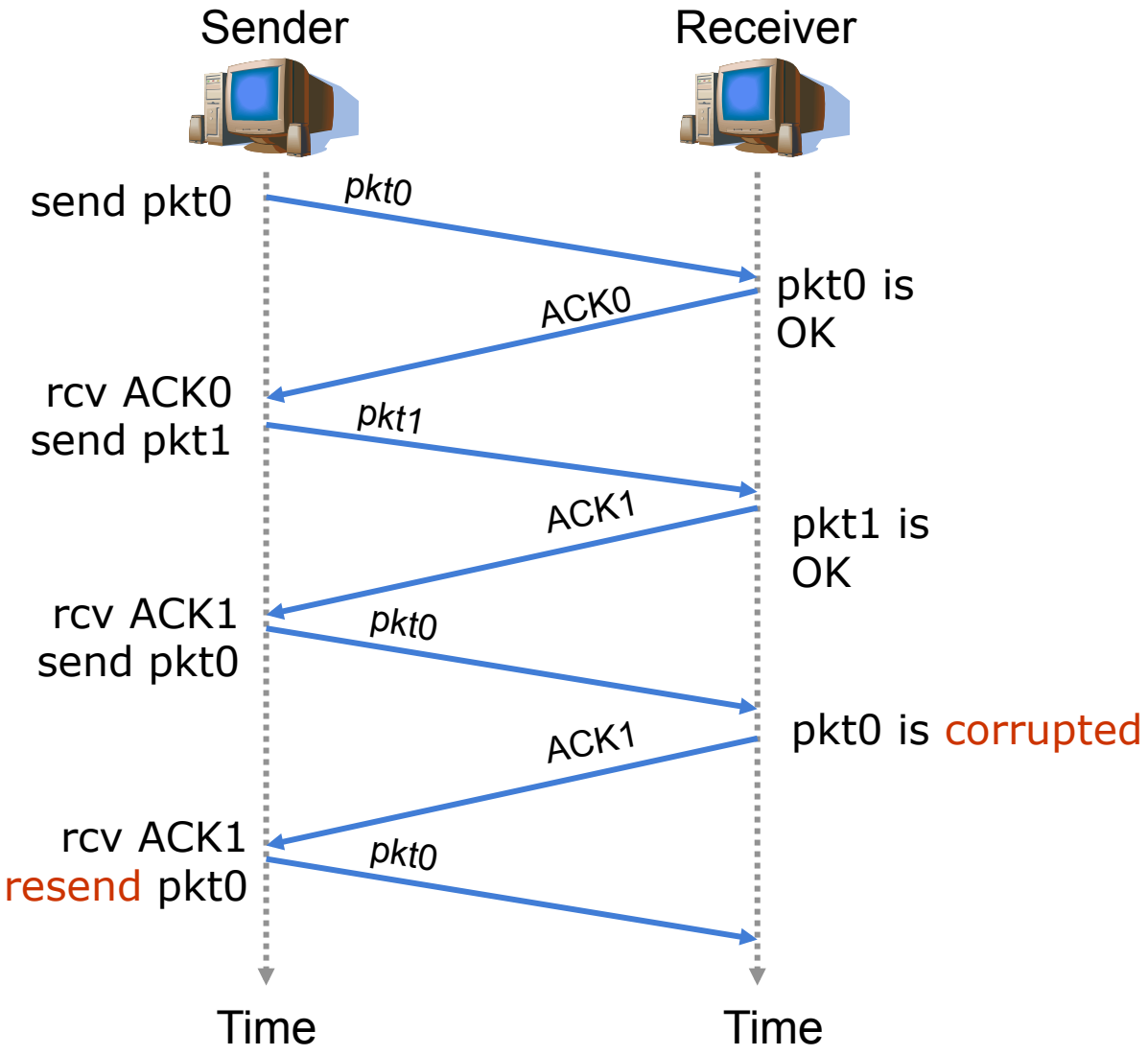

Sender
Receiver
send pkt0
pkt0
pkt1 is
OK
ACK
rcv ACK
send pkt1
pkt1
pkt1 is
corrupted
NAK
rcv NAK
resend pkt1
pkt1
Time
Time

Lỗi ACK/NAK

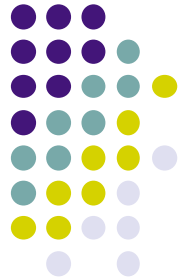
- Cần truyền lại
- Xử lý việc lặp gói tin ntn?
- Thêm Seq.#



Giải pháp không dùng NAK

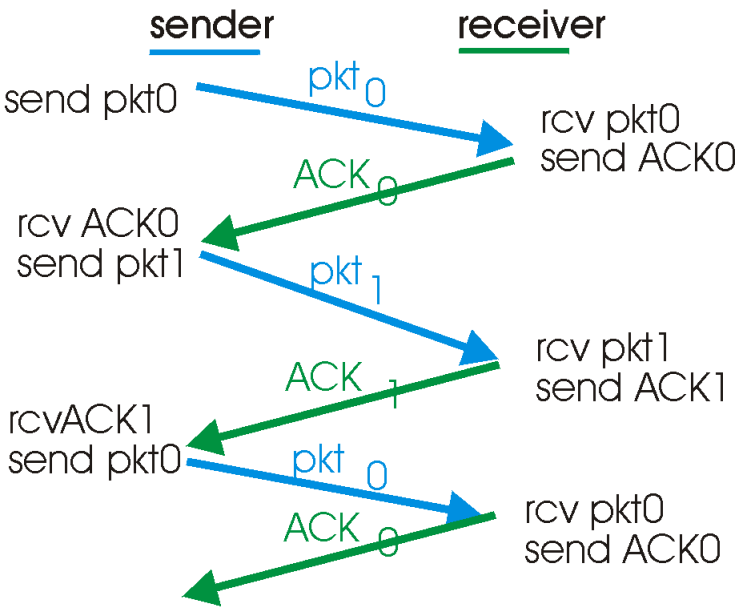


Kênh có lỗi bit và mất gói tin

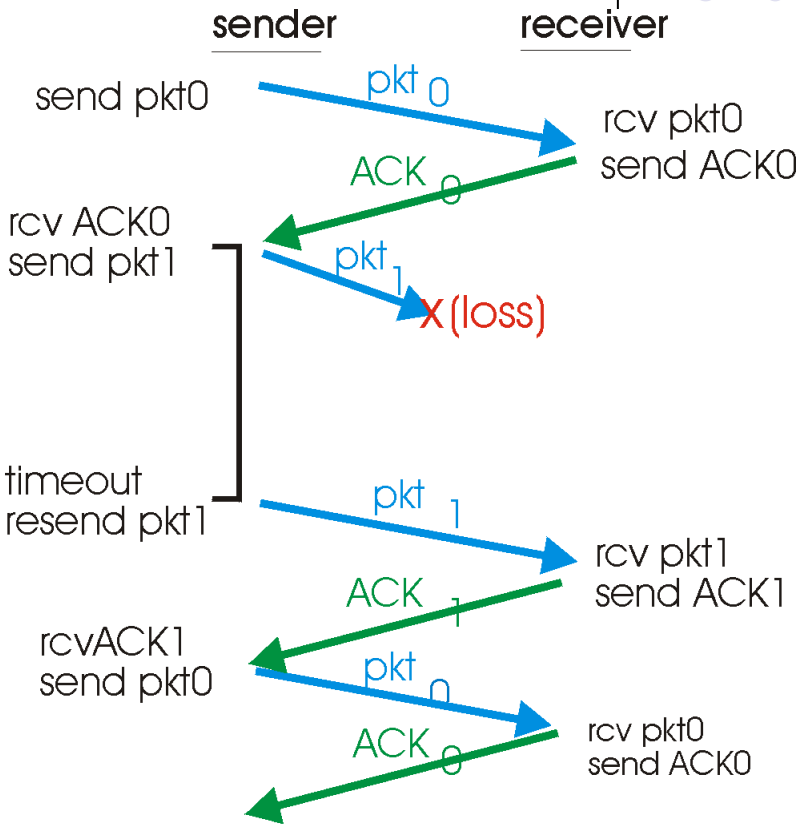


- Dữ liệu và ACK có thể bị mất
  - Nếu không nhận được ACK?
  - Truyền lại như thế nào?
  - Timeout!
- Thời gian chờ là bao lâu?
  - Ít nhất là 1 RTT (Round Trip Time)
  - Mỗi gói tin gửi đi cần 1 timer
- Nếu gói tin vẫn đến đích và ACK bị mất?
  - Dùng số hiệu gói tin

Minh họa

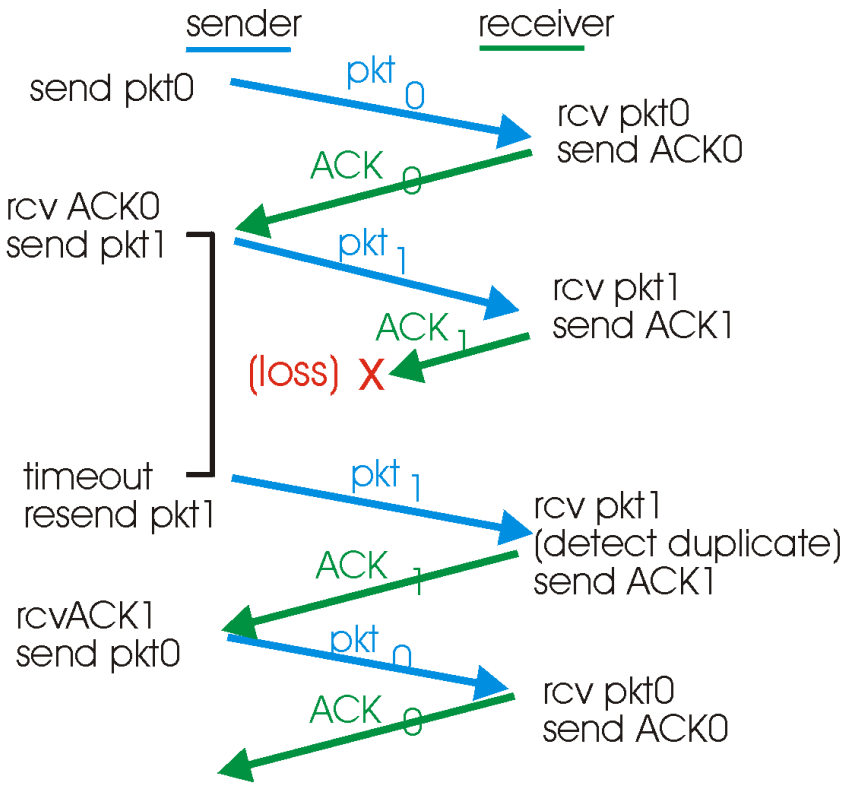


(a) operation with no loss

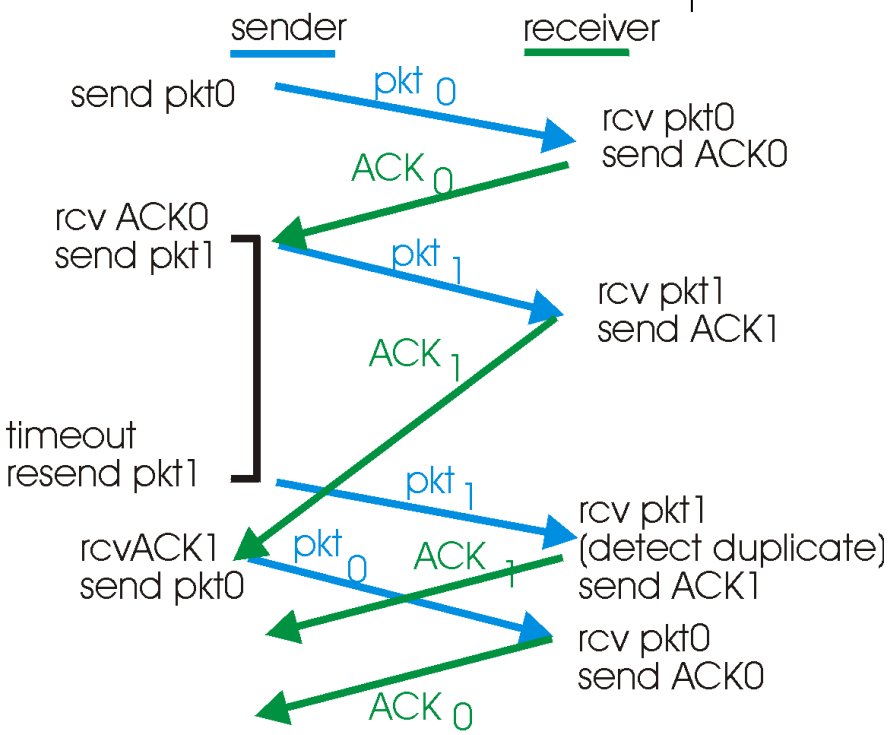


(b) lost packet

Minh họa

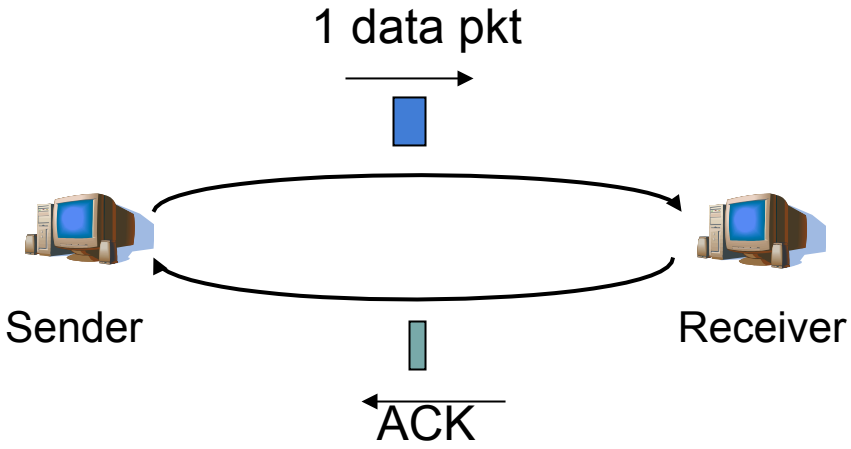


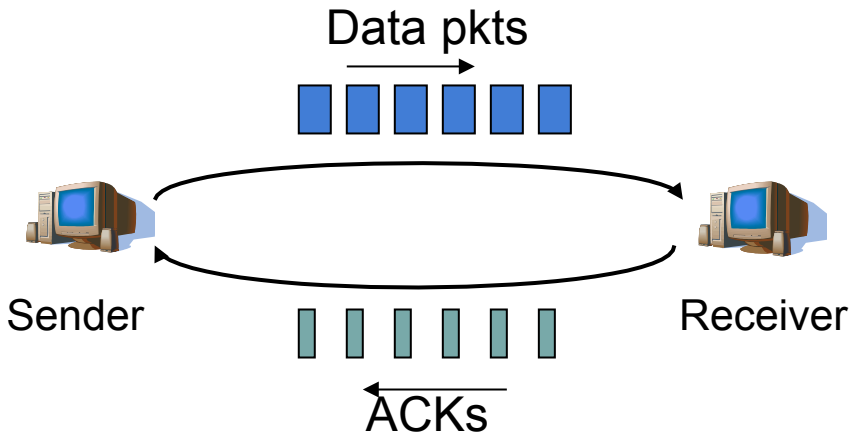
(c) lost ACK



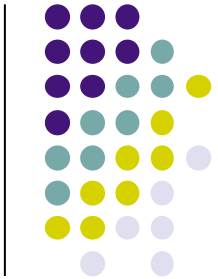
(d) premature timeout

Truyền theo kiểu pipeline

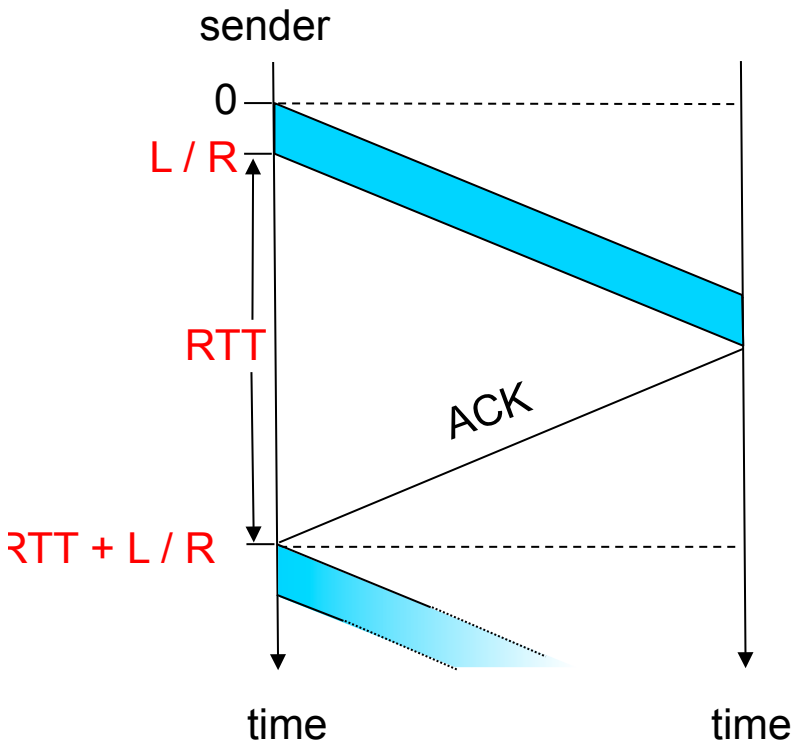




So sánh hiệu quả



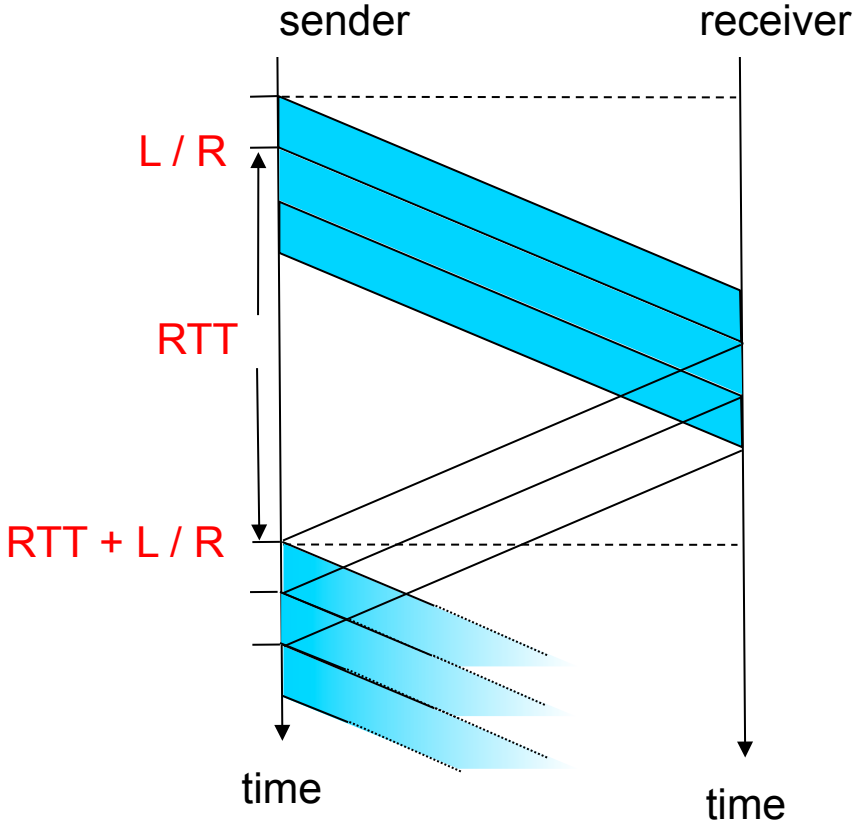
stop-and-wait



L: Size of data pkt
R: Link bandwidth
RTT: Round trip time

$$\text{Performance} = \frac{L / R}{RTT + L / R}$$

Pipeline



$$\text{Performance} = \frac{3 * L / R}{RTT + L / R}$$

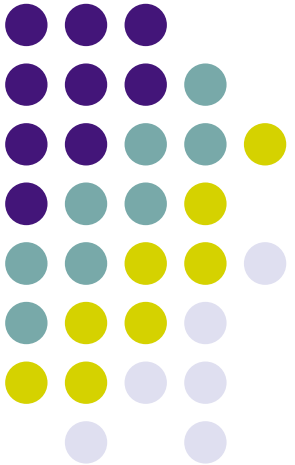
TCP
Transmission Control Protocol

Cấu trúc đoạn tin TCP

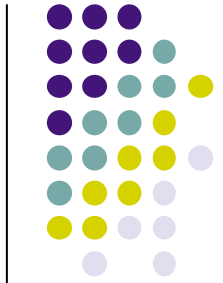
Quản lý liên kết

Kiểm soát luồng

Kiểm soát tắc nghẽn

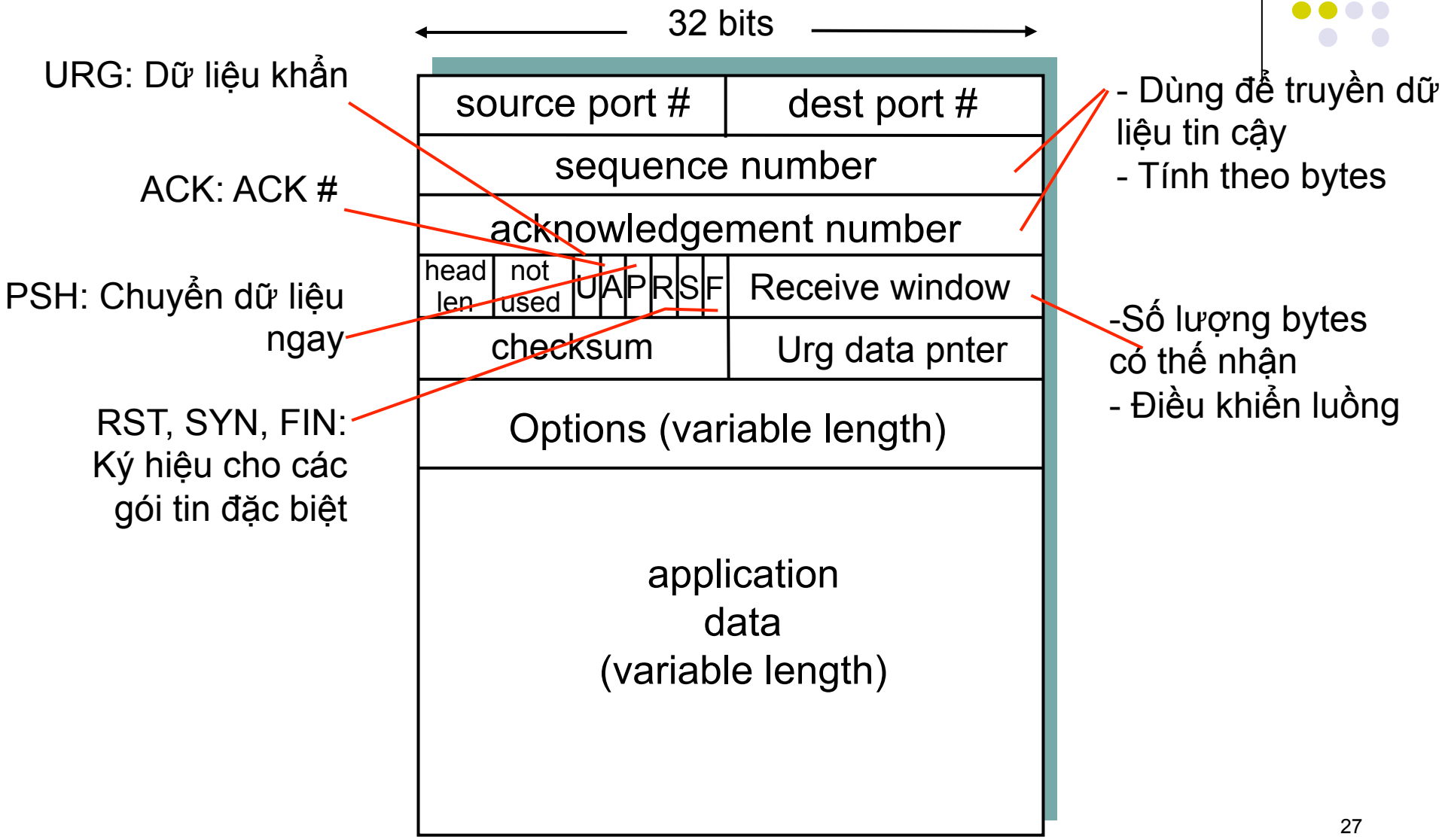


Tổng quan về TCP

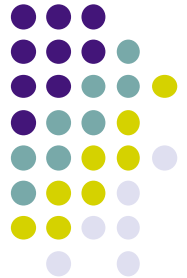


- Giao thức hướng liên kết
  - Bắt tay ba bước
- Giao thức truyền dữ liệu theo dòng byte, tin cậy
  - Sử dụng vùng đệm
- Truyền theo kiểu pipeline
  - Tăng hiệu quả
- Kiểm soát luồng
  - Bên gửi không làm quá tải bên nhận (thực tế: quá tải)
- Kiểm soát tắc nghẽn
  - Việc truyền dữ liệu không nên làm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

Khuôn dạng đoạn tin - TCP segment



TCP cung cấp dịch vụ tin cậy ntn?



- Kiểm soát dữ liệu đã được nhận chưa:
  - Seq. #
  - Ack
- Chu trình làm việc của TCP:
  - Thiết lập liên kết
    - Bắt tay ba bước
  - Truyền/nhận dữ liệu
  - Đóng liên kết

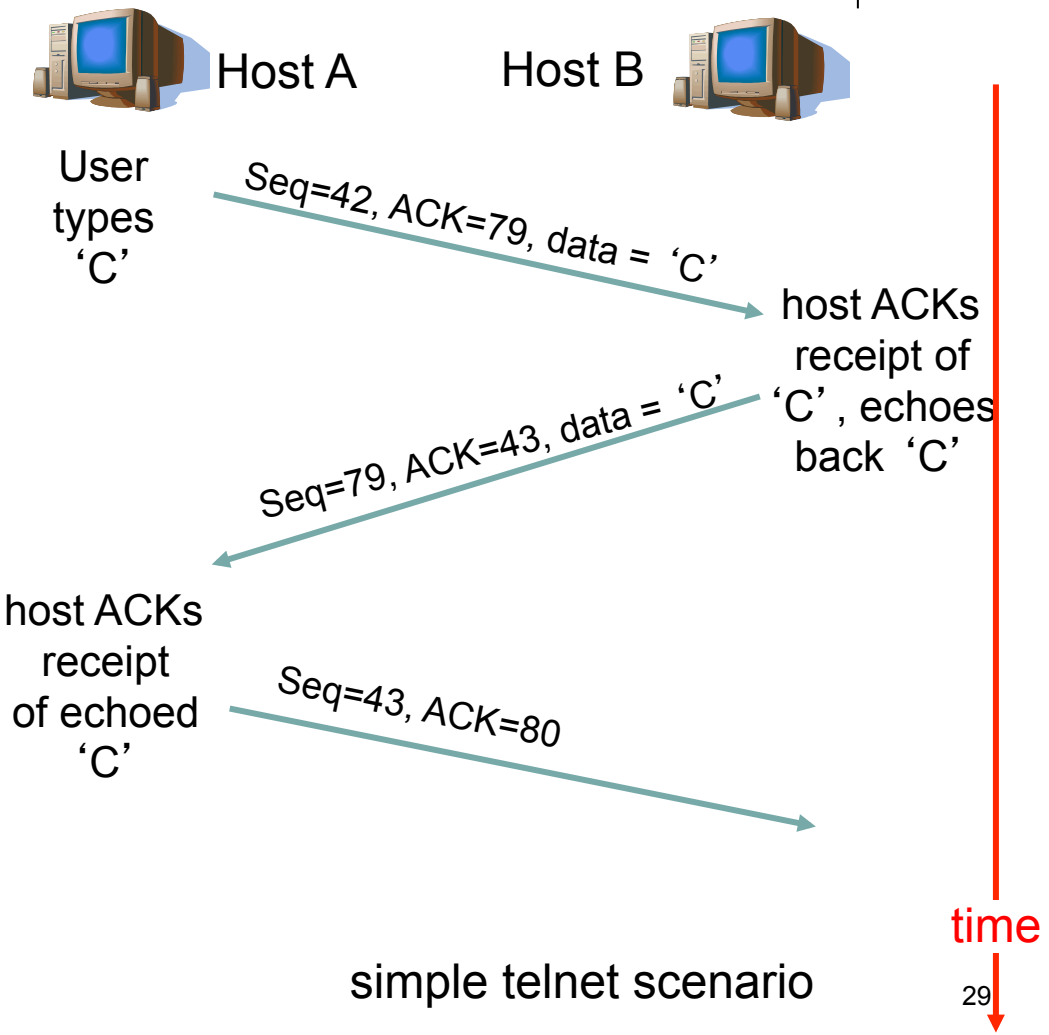
Cơ chế báo nhận trong TCP

Seq. #:

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

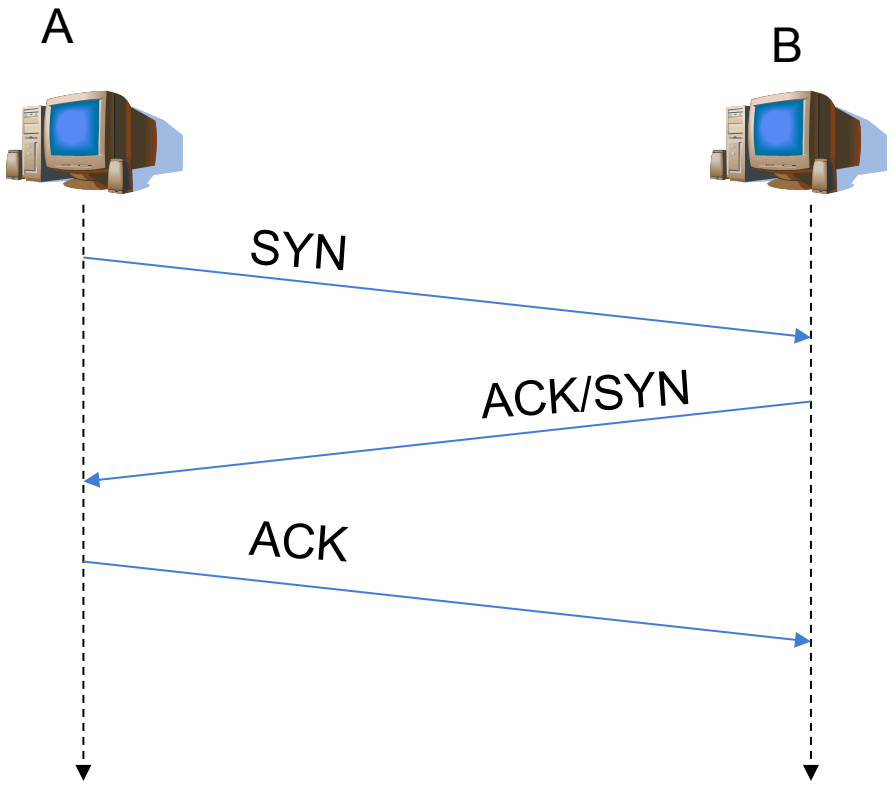
ACK:

- Số hiệu byte đầu tiên mong muốn nhận từ đối tác
- Ngầm xác nhận đã nhận tốt các byte trước đó.



simple telnet scenario

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

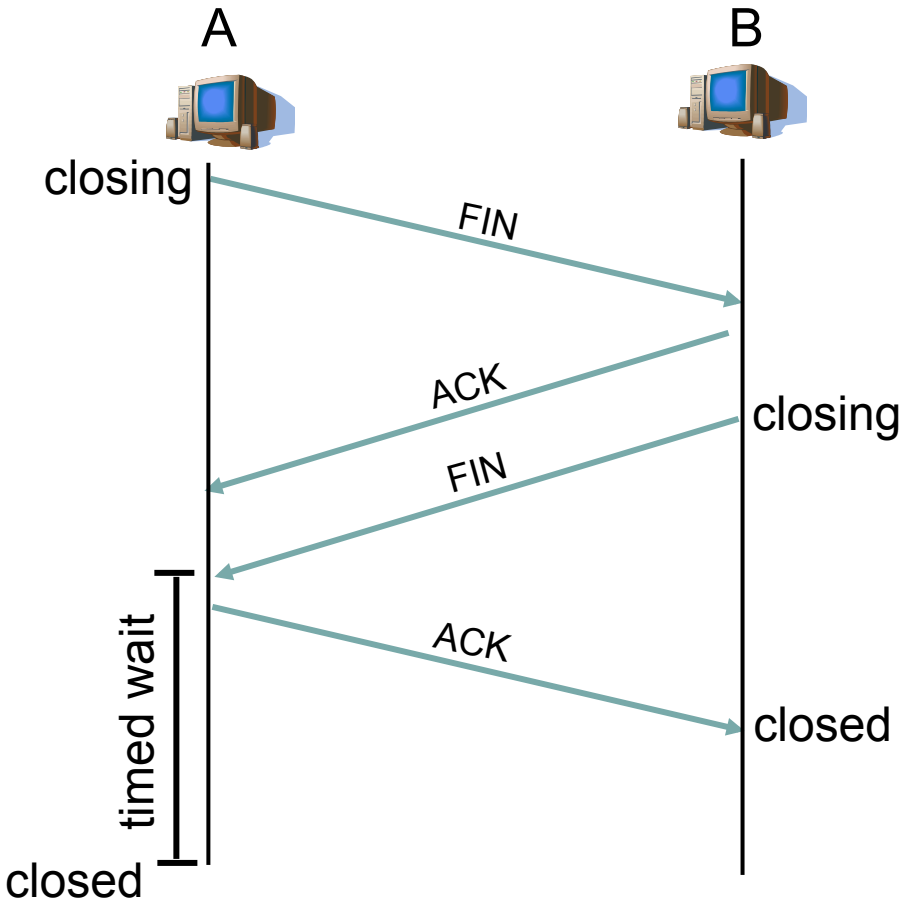


- Bước 1: A gửi SYN cho B
  - chỉ ra giá trị khởi tạo seq # của A
  - không có dữ liệu
- Bước 2: B nhận SYN, trả lời bằng SYNACK
  - B khởi tạo vùng đệm
  - chỉ ra giá trị khởi tạo seq. # của B
- Bước 3: A nhận SYNACK, trả lời ACK, có thể kèm theo dữ liệu

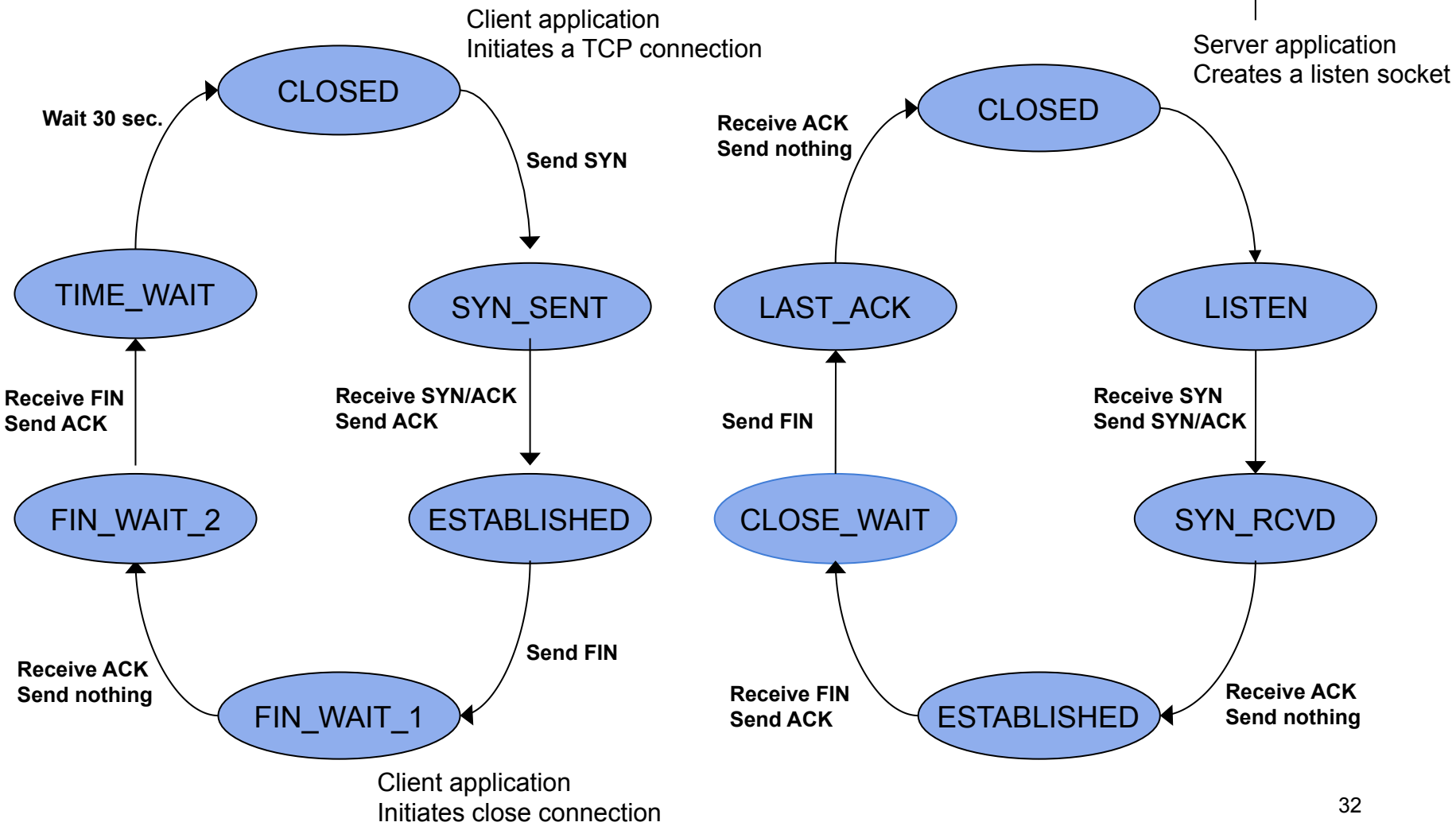
Ví dụ về việc đóng liên kết

- Bước 1: Gửi FIN cho B
- Bước 2: B nhận được FIN, trả lời ACK, đồng thời đóng liên kết và gửi FIN.
- Bước 3: A nhận FIN, trả lời ACK, vào trạng thái "chờ".
- Bước 4: B nhận ACK. đóng liên kết.

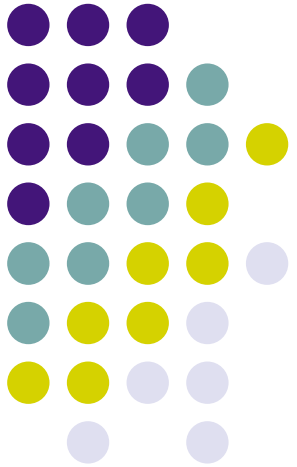
Lưu ý: Cả hai bên đều có thể chủ động đóng liên kết



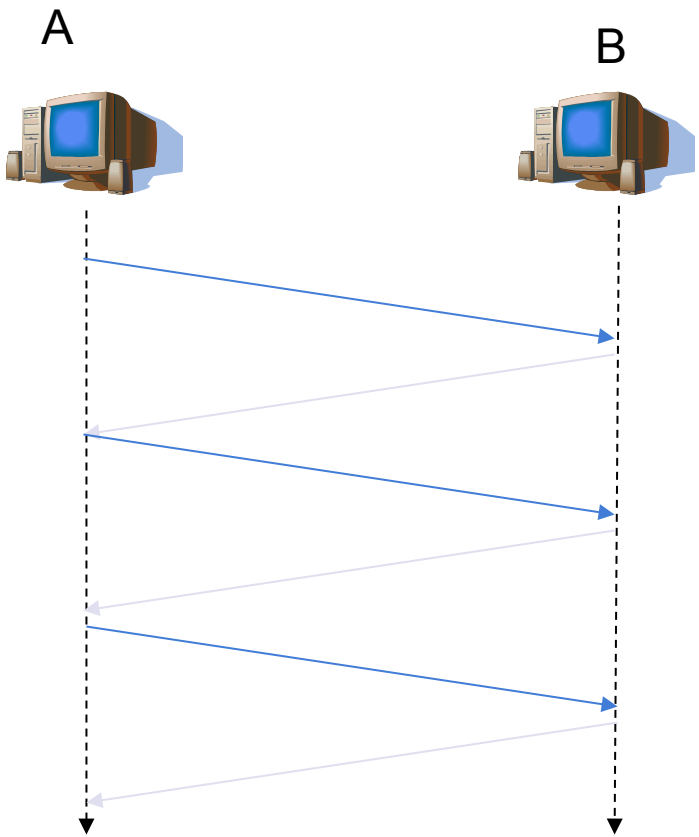
Chu trình sống của TCP (đơn giản hóa)



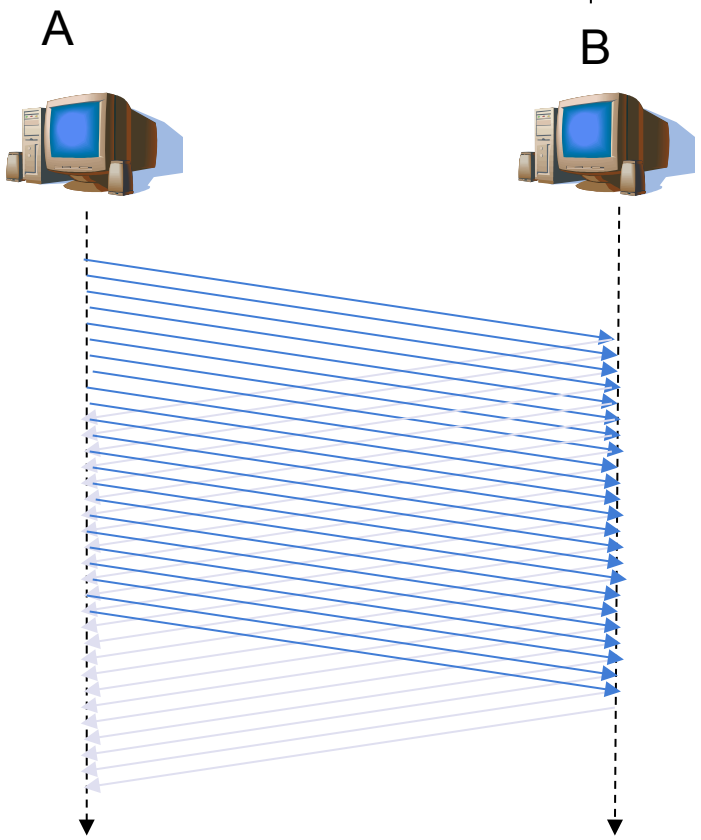
Kiểm soát luồng



Kiểm soát luồng (1)

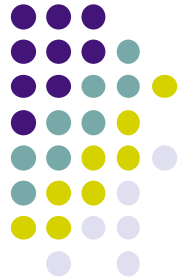


Chậm



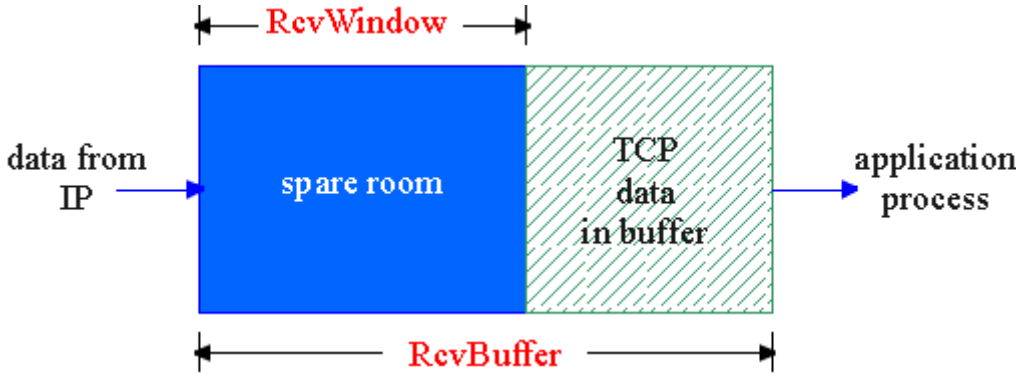
Quá tải

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

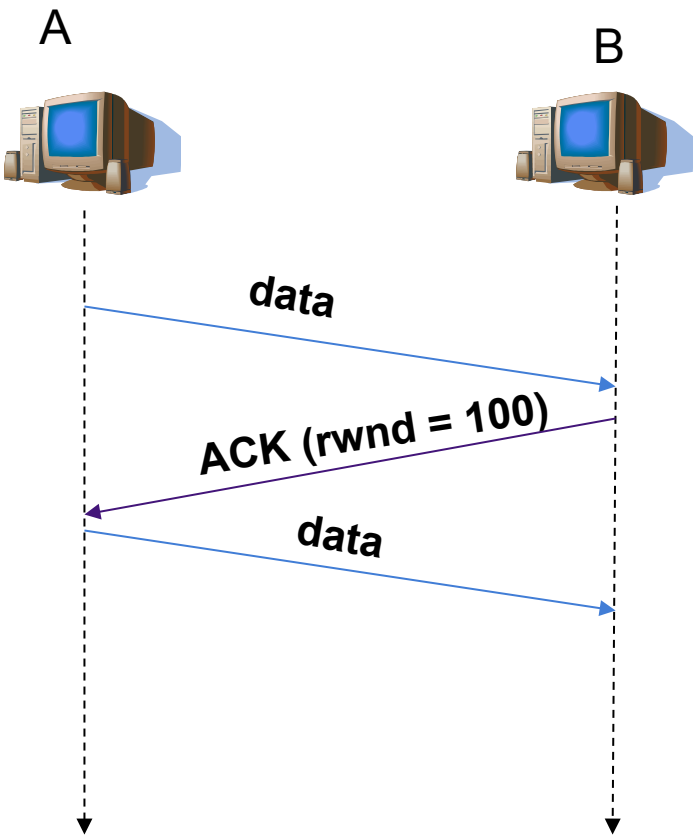


- Kích thước vùng đệm trống

= Rwnd

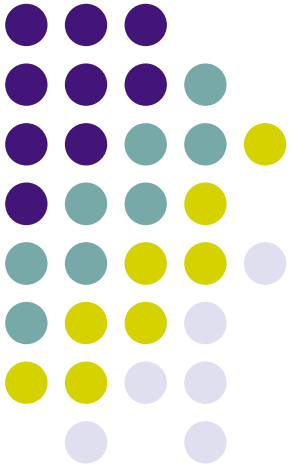
= **RcvBuffer-[LastByteRcvd - LastByteRead]**

Trao đổi thông tin về Rwnd



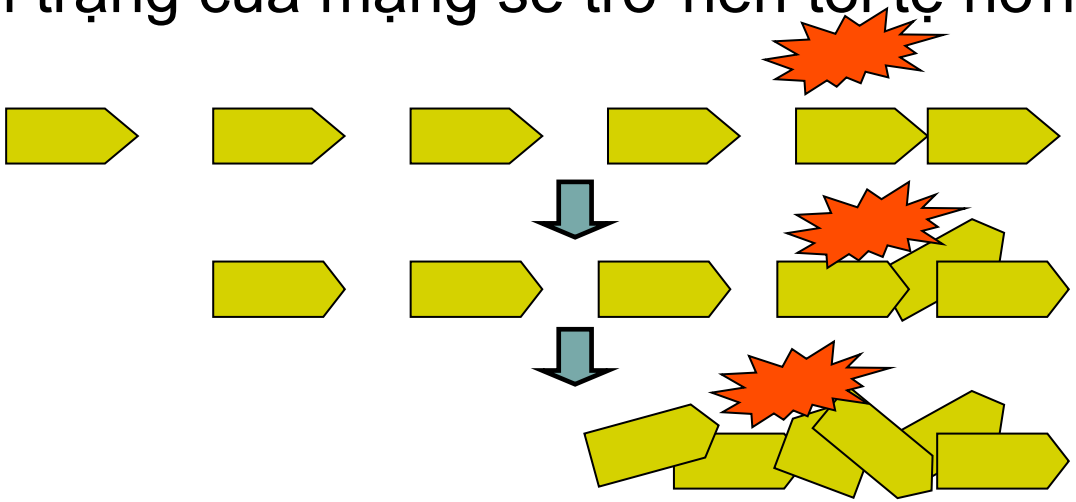
- Bên nhận sẽ báo cho bên gửi biết Rwnd trong các đoạn tin
- Bên gửi đặt kích thước cửa sổ gửi theo Rwnd

Điều khiển tắc nghẽn trong TCP



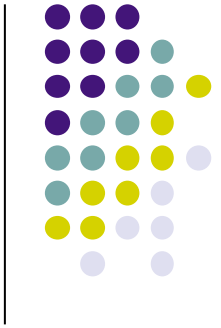
Tổng quan về tắc nghẽn

- 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.



Congestion occur

TCP Kiểm soát tắc nghẽn



- Chiến lược cơ bản
  - Trạm đầu cuối gửi các gói tin TCP vào trong mạng và các trạm này điều chỉnh theo tình trạng mạng quan sát được.
- ACK được sử dùng để điều hòa tốc độ gửi dữ liệu của các trạm đầu cuối TCP

AIMD (Additive Increase / Multiplicative Decrease)

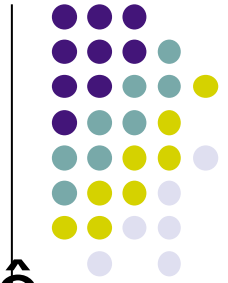
- Mỗi trạm nguồn TCP quản lý một biến CongestionWindow (cwnd)
- Mỗi trạm nhận TCP quản lý một biến ReceiveWindow là kích thước vùng đệm của trạm nhận.

MaxWindow :: min (CongestionWindow , ReceiveWindow)

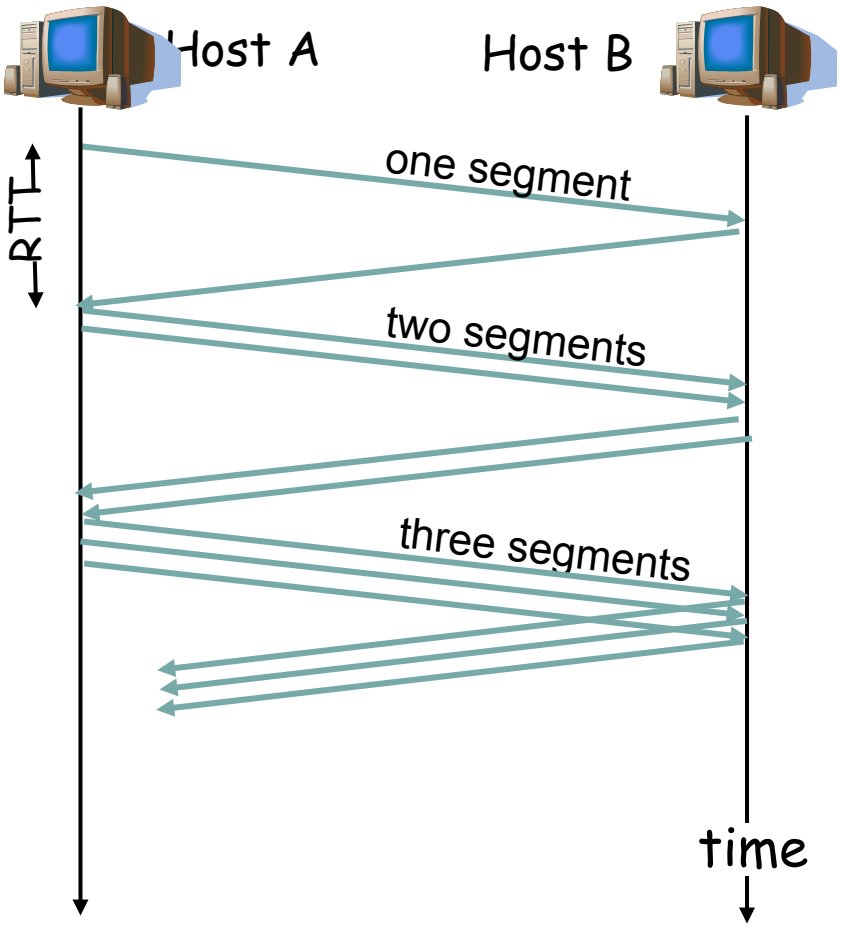
EffectiveWindow = MaxWindow – (LastByteSent -LastByteAcked)

- **cwnd** được thiết lập dựa vào quan sát về mức độ tắc nghẽn:
  - Dấu hiệu không tường minh: lượng gói rớt.
  - Dấu hiệu tường minh: gói điều khiển thông báo tình trạng tắc nghẽn

Additive Increase (AI)



- Additive Increase là phản ứng tăng tốc độ truyền theo cấp số cộng (trong giai đoạn chống tắc nghẽn).
- Thông thường additive increase được định nghĩa bằng tham số **α** (mặc định **α = 1**).
- Mỗi RTT tăng cửa sổ tắc nghẽn lên một lượng cố định
  - Tăng cửa sổ tắc nghẽn thêm tương đương 1 gói tin.



Tăng thêm
1 gói mỗi RTT

Additive Increase

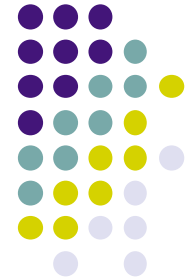
Multiplicative Decrease (MD)

* Giả thiết cơ bản:
  * Mỗi gói bị mất hoặc timeout xảy ra tại bên gửi là do tắc nghẽn tại môt router.
- Multiplicative decrease được định nghiã bằng tham số **β** (mặc định **β = 0.5**)

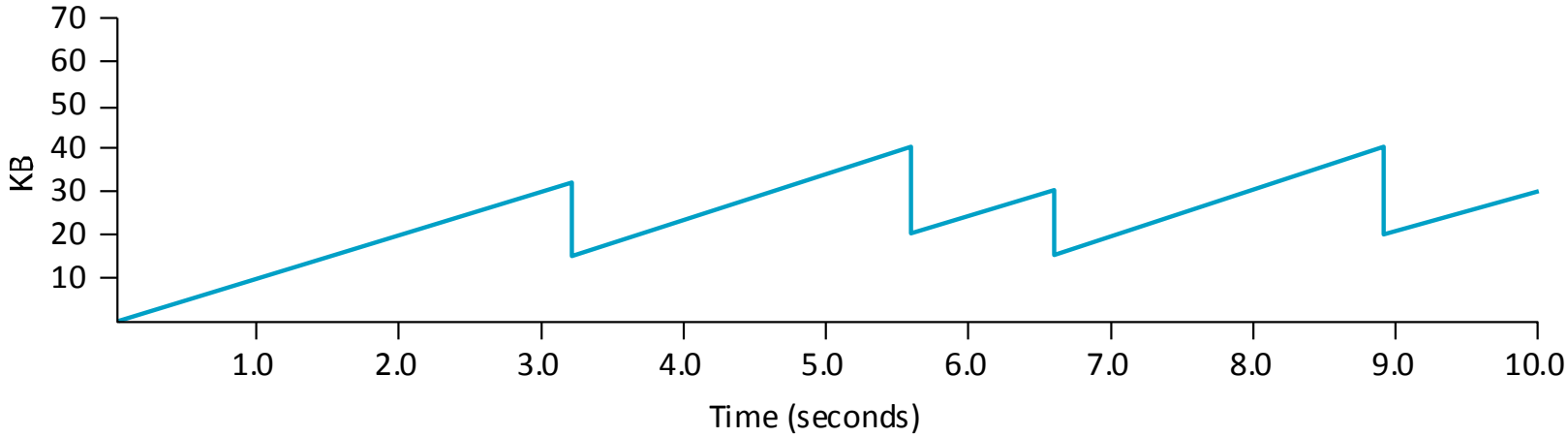
Multiplicative Decrease

- Với mỗi lần timeout TCP giảm kích thước cửa sổ tắc nghẽn **cwnd** đi một hệ số nhân **β**.
- **cwnd** không nhỏ hơn kích thước 1 gói.

AIMD (Additive Increase / Multiplicative Decrease)



- Người ta thấy rằng cần sử dụng AIMD để kiểm soát tình trạng tắc nghẽn của TCP ở trạng thái ổn định.
- Vì các cơ chế timeout gây truyền lại và đòi hỏi ước lượng ngưỡng timeout chính xác
- Timeout được đặt là hàm của RTT trung bình và độ lệch chuẩn của RTT.
- Tuy vậy các trạm đầu cuối TCP chỉ lấy mẫu RTT một lần/ thời gian RTT sử dụng đồng hồ với độ chính xác thấp.

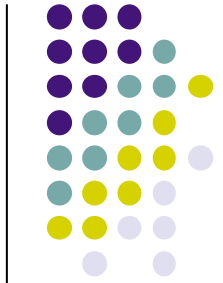


Kiểu răng cưa của TCP

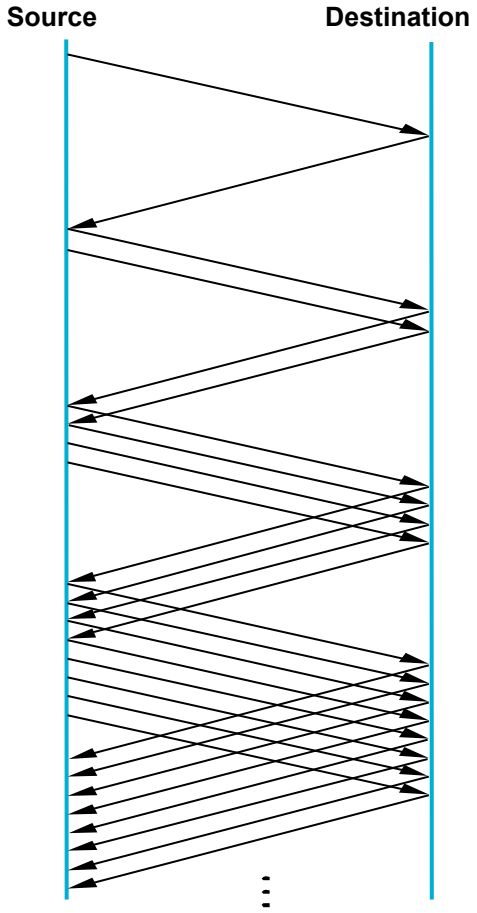
Cơ chế Slow Start (1)

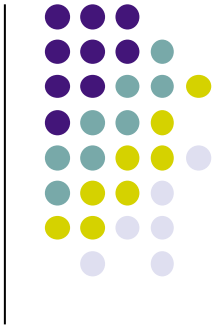
- Tăng tuyến tính có thể mất nhiều thời gian để kết nối TCP đạt tốc độ.
- Kỹ thuật slow start được giới thiệu trong TCP Tahoe
- Ý tưởng cơ bản
  - Đặt cwnd bằng 1 MSS (Maximum segment size)
  - Tăng cwnd lên 1 mỗi khi nhận được ACK
    - Cửa sổ được tăng gấp đôi sau mỗi RTT
  - Bắt đầu thấp, nhưng tăng theo hàm mũ
- Tăng cho đến một ngưỡng: ssthresh
  - Sau đó, TCP chuyển sang trạng thái tránh tắc nghẽn

Cơ chế Slow Start (2)



- 2 trường hợp kích hoạt slow start:
  - Khi mới khởi tạo kết nối.
  - Khi phát hiện tắc nghẽn, sau khi kích thước cửa sổ giảm về 0
    - ½ kích thước cửa sổ cwnd trước tắc nghẽn được dùng làm ngưỡng tắc nghẽn **ssthresh**



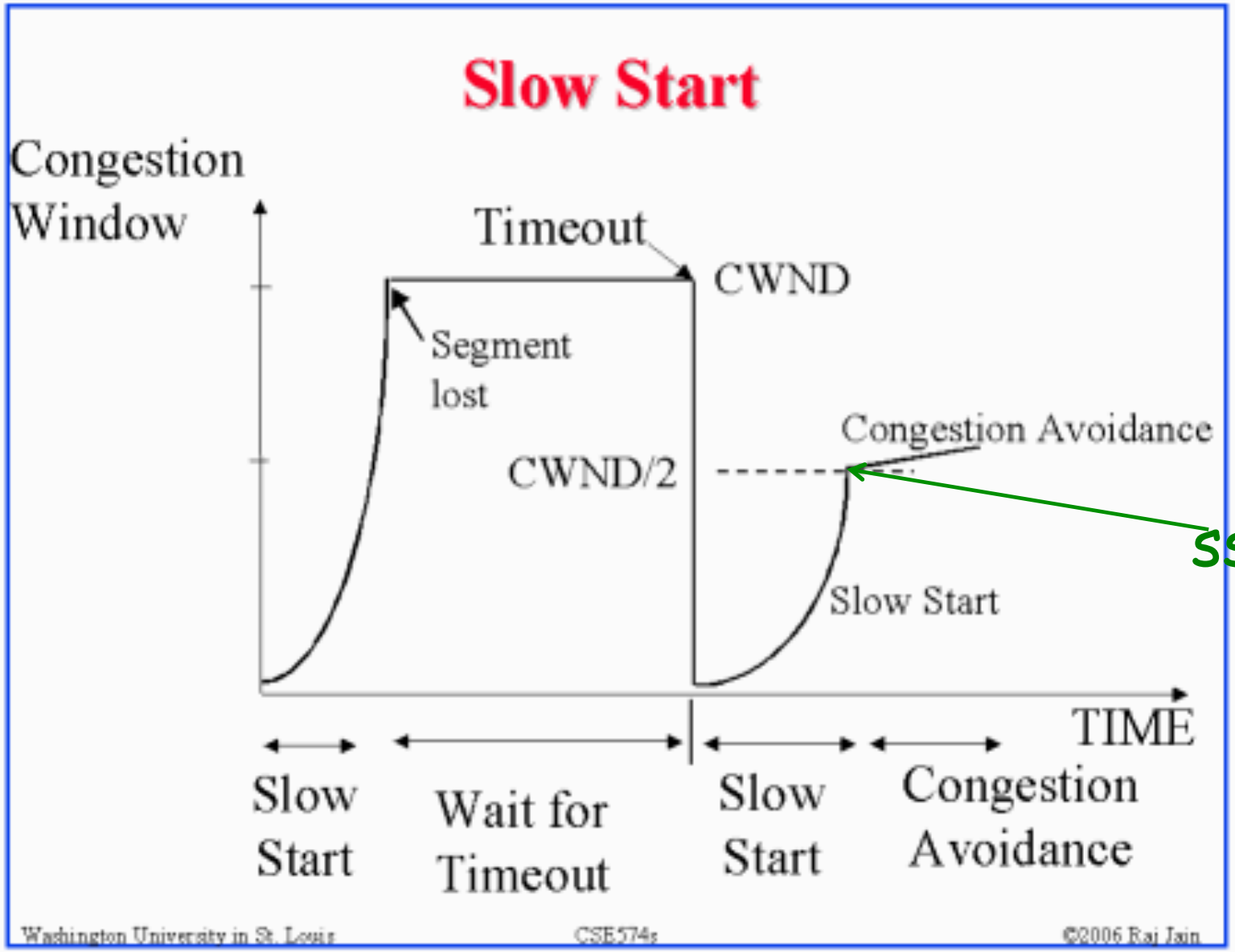


Slow Start
Add one packet per ACK

Figure 6.10 Slow Start

Kết hợp các cơ chế trong TCP

- Thường dùng slow start trong giai đoạn đầu để nhanh chóng đạt tốc độc cao, cho đến ngưỡng ssthresh
- Dùng AIMD trong giai đoạn tránh tắc nghẽn sau khi cwnd đạt ssthresh


Slow Start
Congestion Window
Timeout
CWND
Segment lost
CWND/2
Congestion Avoidance
Slow Start
TIME
Slow Start
Wait for Timeout
Slow Start
Congestion Avoidance
Washington University in St. Louis
CSE574s
©2006 Raj Jain
10-8

ssthresl

Xử lý tắc nghẽn TCP Tahoe (1)

- Sử dụng slowstart
- Phát hiện tắc nghẽn?
  - Nếu như phải truyền lại
    - Có thể suy ra là mạng “tắc nghẽn”
- Khi nào thì phải truyền lại?
  - Timeout!
  - Cùng một gói tin số hiệu gói tin trong ACK

Xử lý tắc nghẽn TCP Tahoe khi timeout (2)

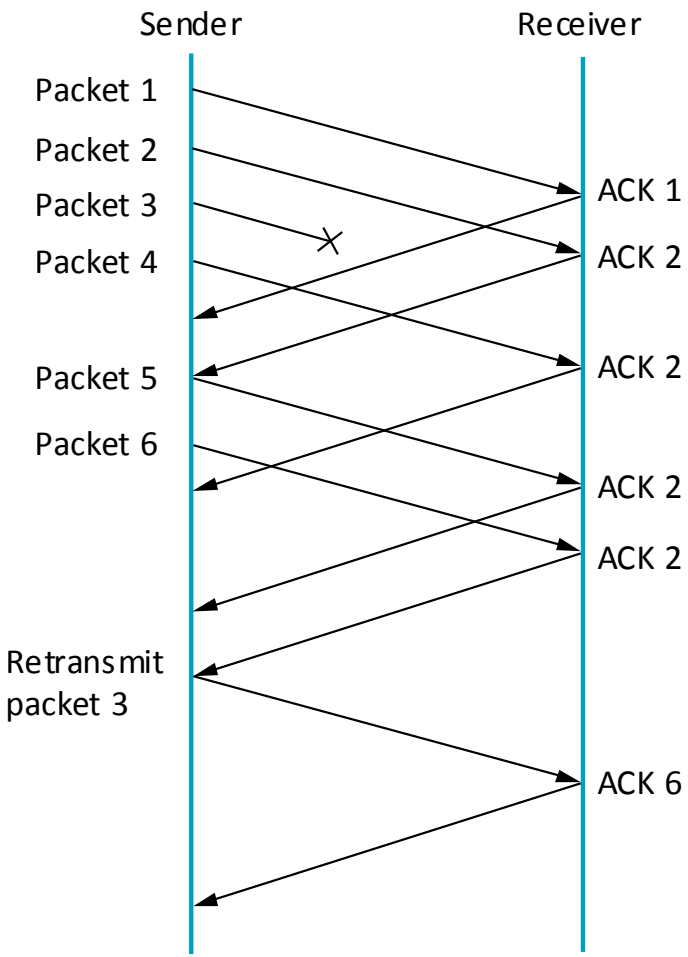
- Khi có timeout của bên gửi
  - TCP đặt ngưỡng **ssthresh** xuống còn một nửa giá trị hiện tại của cwnd
  - TCP đặt cwnd về 1 MSS
  - TCP chuyển về slow start

Fast Retransmit

- Thông thường, việc truyền lại gói tin chỉ được thực hiện khi qua timeout mà chưa nhận được ACK.
- Timeout được tính căn cứ RTT, RTT được đo 1 lần.
- Giá trị RTT thay đổi → timeout không chính xác.
- Fast retransmit cho phép truyền lại nhanh:
  - *Coi việc nhận được **dupllcate ACKs** là biểu hiện mất gói tin và thực hiện truyền lại gói bị mất ngay mà không chờ hết time out.*

Fast Retransmit

Khi nhận được 3 ACK trùng nhau,
TCP gửi lại các gói bị mất.

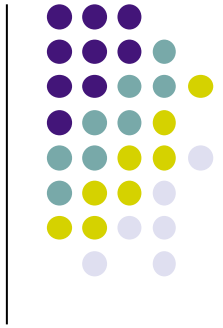


Fast Retransmit

Based on three duplicate ACKs

Fast Retransmit

Fast Retransmit



- Được dùng trong TCP Tahoe
- Nếu nhận được 3 ACK giống nhau
  - TCP đặt ngưỡng **ssthresh** xuống còn một nửa giá trị hiện tại của cwnd
  - TCP Tahoe: đặt cwnd =1 → **quay lại slowstart**

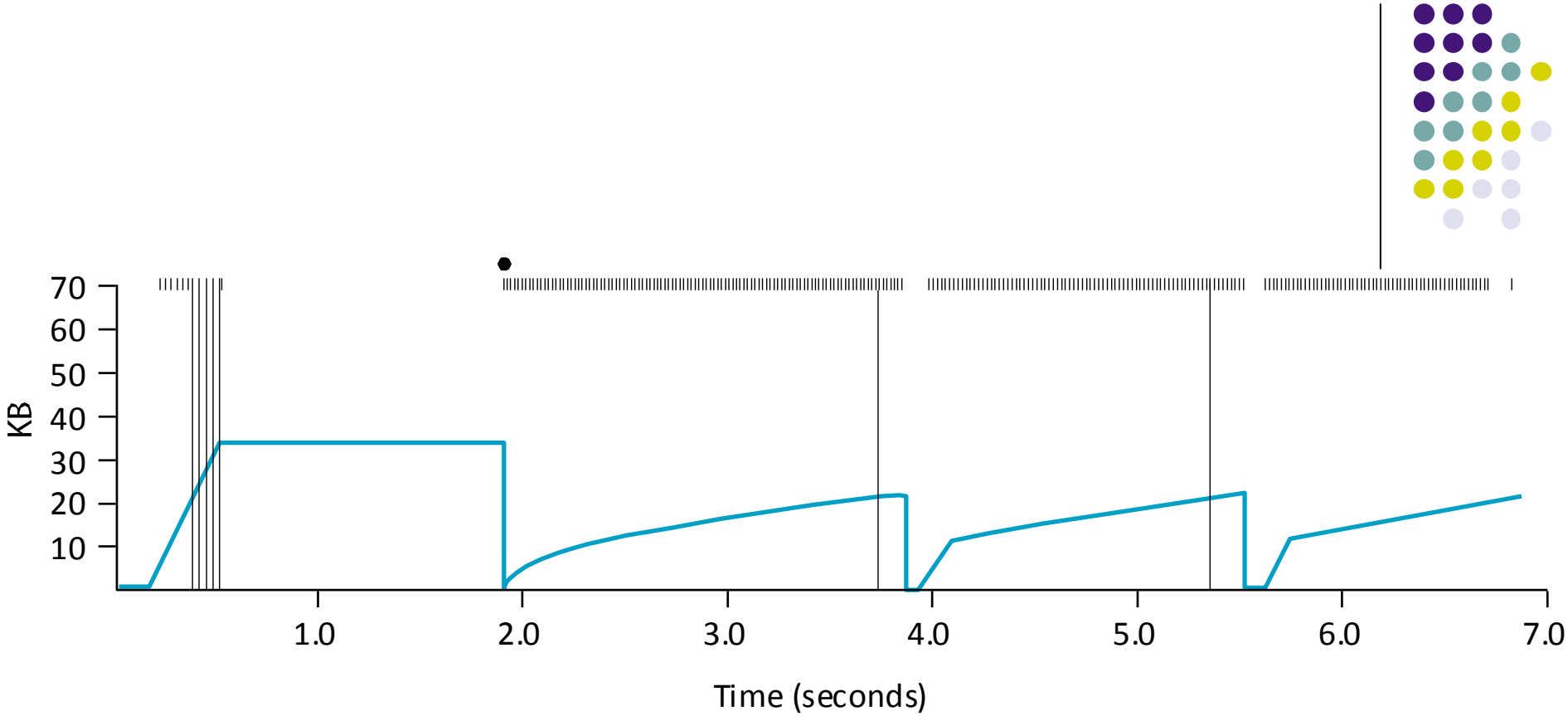
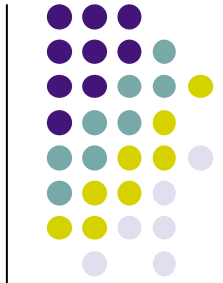


Figure 6.13 TCP Fast Retransmit Trace

Fast Recovery

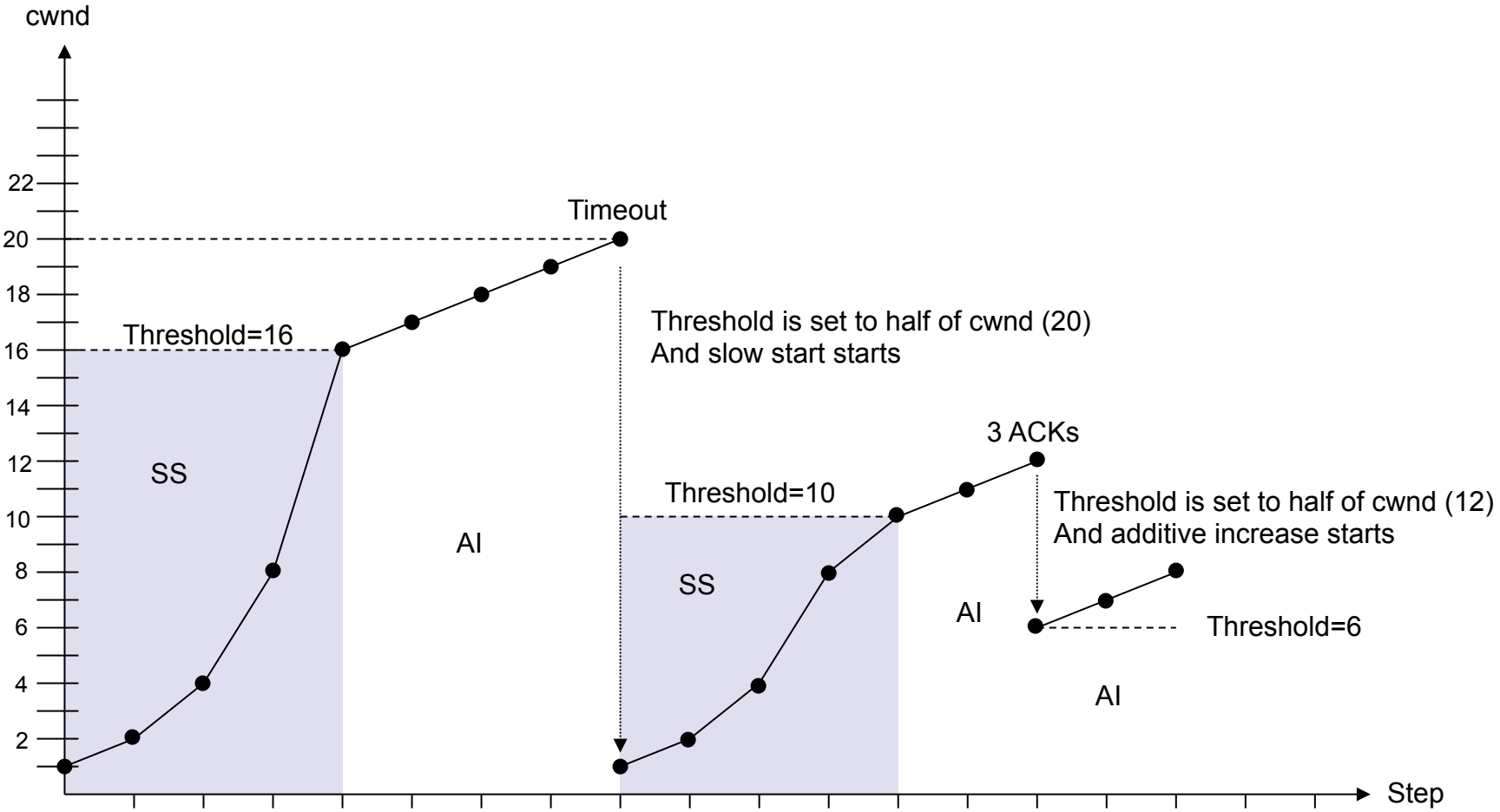


- Fast recovery sử dụng trong **TCP Reno**.
- Ý tưởng chính:
  - Truyền lại ngay các gói đã mất khi nhận được 3 ACKs giống nhau
  - Sau khi nhận được ACK của tất cả các gói đã truyền lại thì chuyển sang giai **đoạn congestion avoidance**
    - → **bỏ qua slow start trong Fast Retransmit.**
  - Sau đó nếu timeout: quay về slowstart

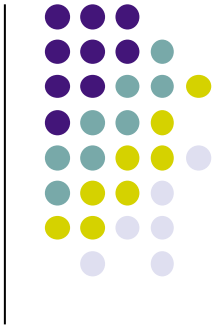
Fast Recovery

Sau Fast Retransmit, giảm cwnd đi ½ và bắt đầu tăng cửa sổ tuyến tính

Kiểm soát tắc nghẽn TCP Reno

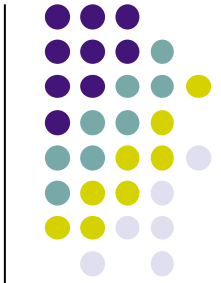


TCP Tahoe vs Reno



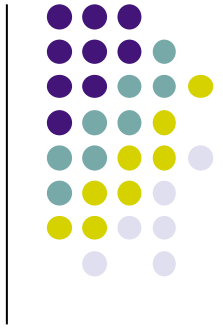
- TCP Tahoe:
  - Slow start,
  - congestion avoidance
  - Timeout: Slowstart.
  - Fast retransmit nếu nhận được 3 ACK trùng nhau:
    - **giảm cwnd =1,**
    - bắt đầu slow start mới với ssthresold bằng ½ cwnd cũ.
- TCP Reno:
  - Slow start
  - Congestion avoidance
  - Timeout: slow start
  - Fast Recovery nếu nhận được 3 ACK trùng nhau:
    - **giảm cwnd đi** ½ → bỏ qua slow start
    - Đặt ssthresold =cwnd mới
    - Truyền lại các gói mất, chờ đủ ACK rồi Thực hiện Congestion avoidance.

Nhiều biến thể TCP khác



- TCP New Reno
  - Sau ACK trùng lặp, gửi thêm 1 gói ở cuối cửa sổ gửi.
  - Điều chỉnh timeout
  - Gây nhiều gói chưa được ACK trong chuỗi gửi
  - Cho tốc độ gửi cao hơn TCP Tahoe và Reno
- TCP SACK
  - Cả bên gửi và nhận đều phải hỗ trợ TCP SACK
- TCP Vegas
  - Điều chỉnh kích thước cửa sổ theo sự khác biệt giữa RTT kỳ vọng và thực tế
  - Chủ yếu dùng trong phòng thí nghiệm
- TCP Cubic
  - CUBIC TCP được cài đặt mặc định trong hạt nhân Linux phiên bản 2.6.19 trở lên, và trong Windows 10.1709 Fall Creators Update, Windows Server 2016 1709 update

Bài tập

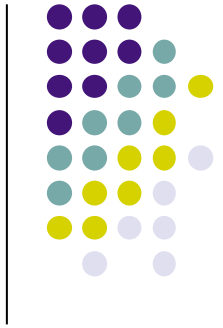


- Giả sử cần truyền 1 file
  - Kích thước *O* =100KB trên kết nối TCP
  - S là kích thước mỗi gói TCP, *S* = 536 byte
  - *RTT* = 100 *ms*.

  Giả sử cửa tốc độ truyền của TCP chỉ bị giới hạn bởi cửa số nhận Rwnd theo cơ chế sliding windows.

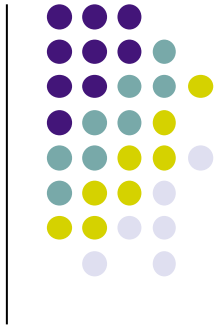
  Hỏi thời gian chờ đợi ít nhất để truyền hết file là bao nhiêu? Nếu tốc độ đường truyền là
  - R = 10 Mbit/s;
  - R= 100 Mbits/s.

Bài tập



- Giả sử cần truyền 1 file
  - Kích thước *O* =100KB trên kết nối TCP
  - S là kích thước mỗi gói TCP, *S* = 536 byte
  - *RTT* = 100 *ms*.
- Giả sử tốc độ truyền của TCP chị bị giới hạn bởi cửa sổ kiểm soát tắc nghẽn Cwnd và cửa sổ hoạt đông theo cơ chế slow-start, chưa tính đến giai đoạn congestion avoidance.
- Cửa sổ đạt đến giá trị bao nhiêu thì truyền hết file.
- Hỏi thời gian chờ đợi ít nhất để truyền hết file là bao nhiêu? Nếu R = 10 Mbit/s; R= 100 Mbits/s.

Bài tập



- 2 người 1 nhóm

1. Trình bày tìm hiểu về cơ chế kiểm soát tắc nghẽn trong một trong các phiên bản TCP New Reno, TCP SACK, TCP Vegas, TCP Cubic. So sánh với TCP Tahoe và TCP Reno.
2. Triển khai OSPF multi-area và thực hiện các thử nghiệm minh họa hoạt động sử dụng quagga.
3. Triển khai RIP v2 với cơ chế bảo mật và minh họa hoạt động sử dụng quagga.
4. Triển khai BGP sử dụng quagga và minh họa hoạt động.

Báo cáo 16/4, nộp báo cáo + trình bày 15 phút/ nhóm