TRƯỜNG ĐẠI HỌC THỦY LỢI
KHOA CÔNG NGHỆ THÔNG TIN

MẠNG MÁY TÍNH

Chương 3: Tầng giao vận

ThS. Nguyễn Thị Phương Thảo
Email: thaont@tlu.edu.vn

NỘI DUNG

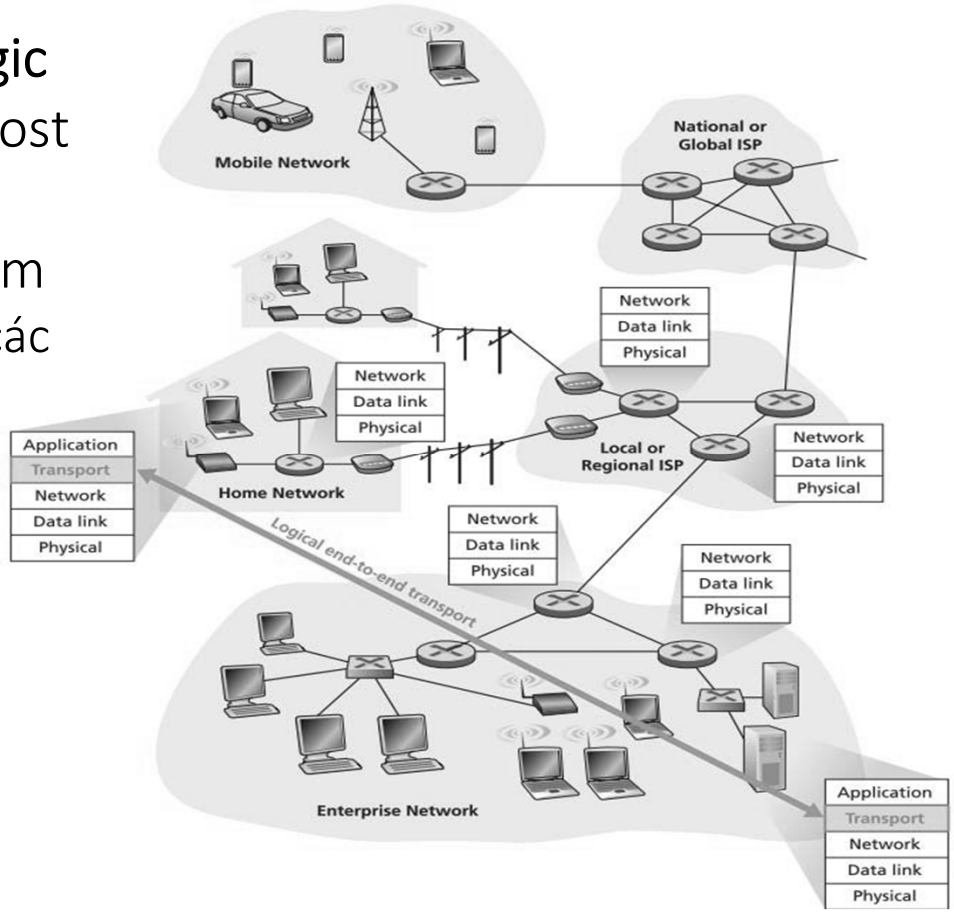
1. **Tổng quan về tầng giao vận**
2. Dồn kênh và phân kênh
3. Giao thức UDP
4. Truyền dữ liệu tin cậy
5. Giao thức TCP

1. TỔNG QUAN VỀ TẦNG GIAO VẬN

a. Dịch vụ và giao thức tầng giao vận
b. Quan hệ giữa tầng giao vận và tầng mạng
c. Tầng giao vận trong mạng Internet

a. Dịch vụ và giao thức tầng giao vận

- Giao thức giao vận cung cấp **kênh truyền logic** giữa các *tiến trình ứng dụng* chạy trên các host khác nhau
- Giao thức giao vận chạy trong các end system
  - Phía gửi: chia thông điệp (message) thành các **segment,** chuyển tới tầng mạng
  - Phía nhận: ghép các segment lại thành message, chuyển tới tầng ứng dụng
- Có nhiều giao thức tầng giao vận
  - Mạng Internet: TCP và UDP

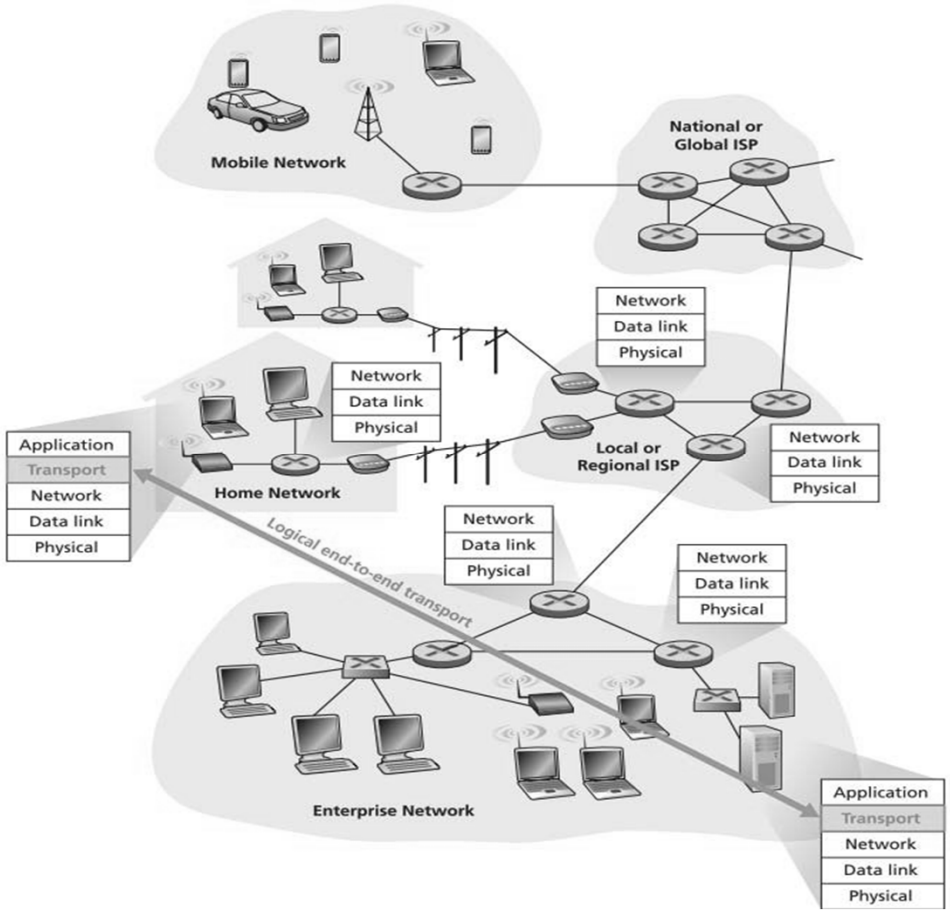


b. Quan hệ giữa tầng giao vận và tầng mạng

- Tầng giao vận: cung cấp đường truyền
  Logic giữa các tiến trình chạy trên các host
- Tầng mạng: cung cấp đường truyền
  Logic giữa các host
- Ví dụ:
  - *2 trại trẻ mồ côi A và B kết giao với nhau. Mỗi trại có 10 trẻ. Quy ước: mỗi CN các trẻ có thể viết thư cho bạn mình ở nhà trẻ bên còn lại.*
  - *Có 2 bạn nhỏ chịu trách nhiệm thu thập thư của cả trại và gửi ra bưu điện: Ann (trại A) và Bill (trại B) đồng thời nhận thư và chia lại cho các bạn*
- Liên hệ
  - *Host: trại A, trại B*
  - *Tiến trình: trẻ mồ côi*
  - *Thông điệp: thư*
  - *Giao thức tầng giao vận: Ann và Bill → chỉ chạy trên đầu cuối*
  - *Giao thức tầng mạng: dịch vụ bưu điện*

c. Tầng giao vận trong mạng Internet

- TCP (Transmission Control Protocol):
  - Truyền dữ liệu tin cậy
  - Thiết lập kết nối trước khi truyền
  - Điều khiển luồng
  - Điều khiển tắc nghẽn
- UDP (User Datagram Protocol):
  - Truyền dữ liệu không tin tin cậy
  - Không thiết lập kết nối trước khi truyền
  - Có kiểm soát lỗi
  - Không có điều khiển luồng/tắc nghẽn
- Các dịch vụ không cung cấp ở tầng giao vận:
  - Đảm bảo độ trễ (thời gian truyền)
  - Đảm bảo băng thông



c. Tầng giao vận của Internet

- Ứng dụng và dịch vụ tầng giao vận

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

NỘI DUNG

1. Tổng quan về tầng giao vận
2. **Dồn kênh và phân kênh**
3. Giao thức UDP
4. Truyền dữ liệu tin cậy
5. Giao thức TCP

DỒN KÊNH và PHÂN KÊNH

- Dịch vụ truyền tin tiến trình – tiến trình
- Socket và port number

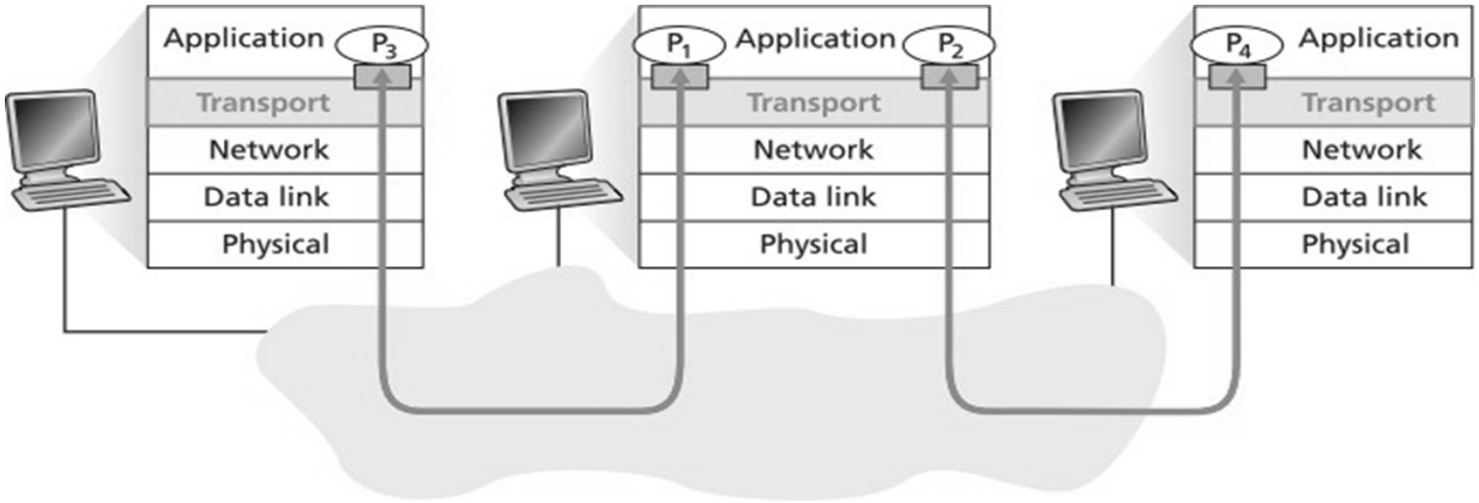
Dồn kênh – Phân kênh

Dồn kênh (multiplexing) tại nút gửi

Tầng giao vận nhận dữ liệu từ nhiều tiến trình khác nhau, tạo segment chứa dữ liệu và thêm thông tin số hiệu cổng nguồn và số hiệu cổng đích vào tiêu đề segment và gửi segment xuống tầng mạng

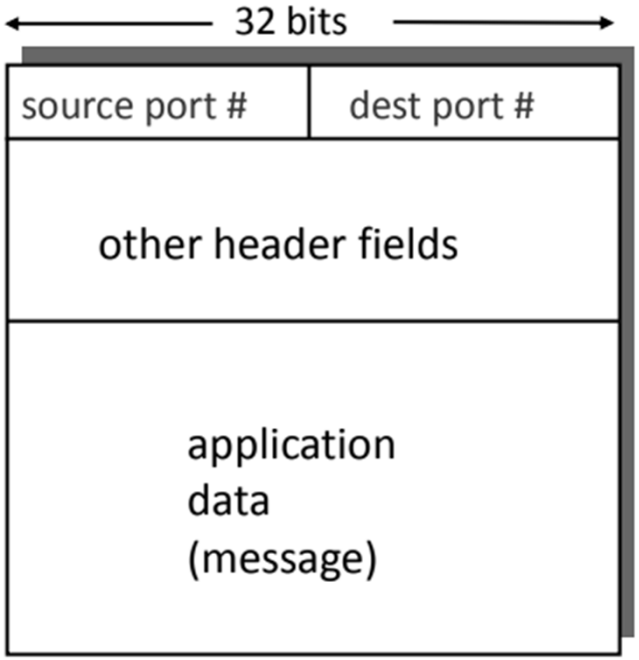
Phân kênh (demultiplexing) tại nút nhận

Tầng giao vận sử dụng thông tin header để chuyển các segment đã nhận tới đúng tiến trình ứng dụng nhận



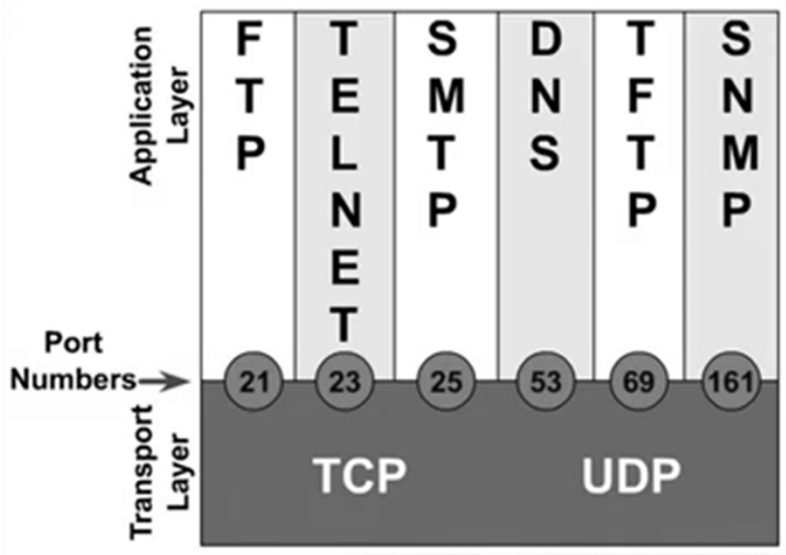
Dồn kênh – Phân kênh: hoạt động

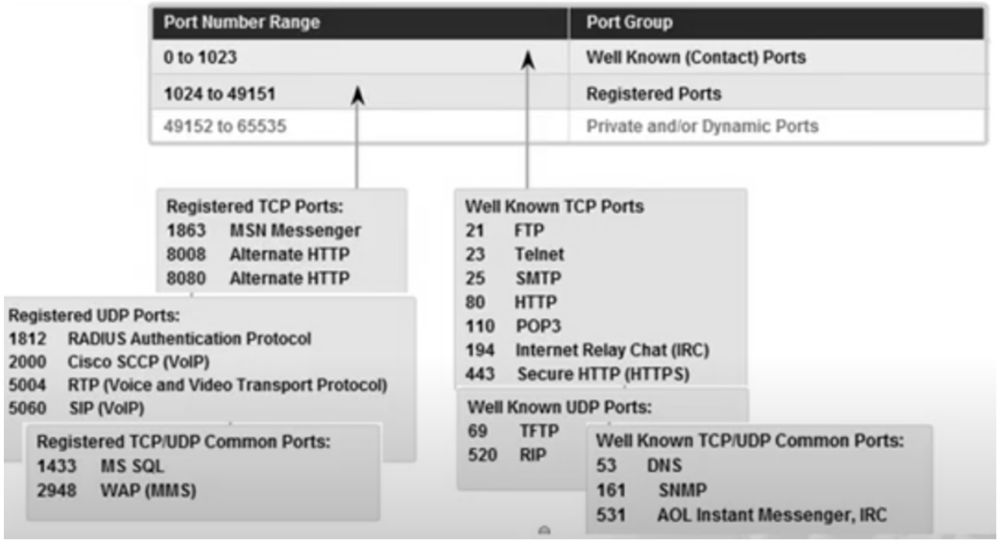
- Dồn kênh và phân kênh nhờ hai trường đặc biệt ở đầu segment
  - Định danh cổng tiến trình gửi (source port number): là một giá trị chưa được sử dụng bởi tiến trình nào
  - Định danh cổng tiến trình nhận (dest port number): là số hiệu cổng của ứng dụng, ví dụ: Web là 80
- Mỗi datagram ở tầng mạng có: Địa chỉ IP nguồn và IP đích để xác định host gửi và host nhận
- Host sử dụng địa chỉ IP và port number để chuyển segment tới socket thích hợp



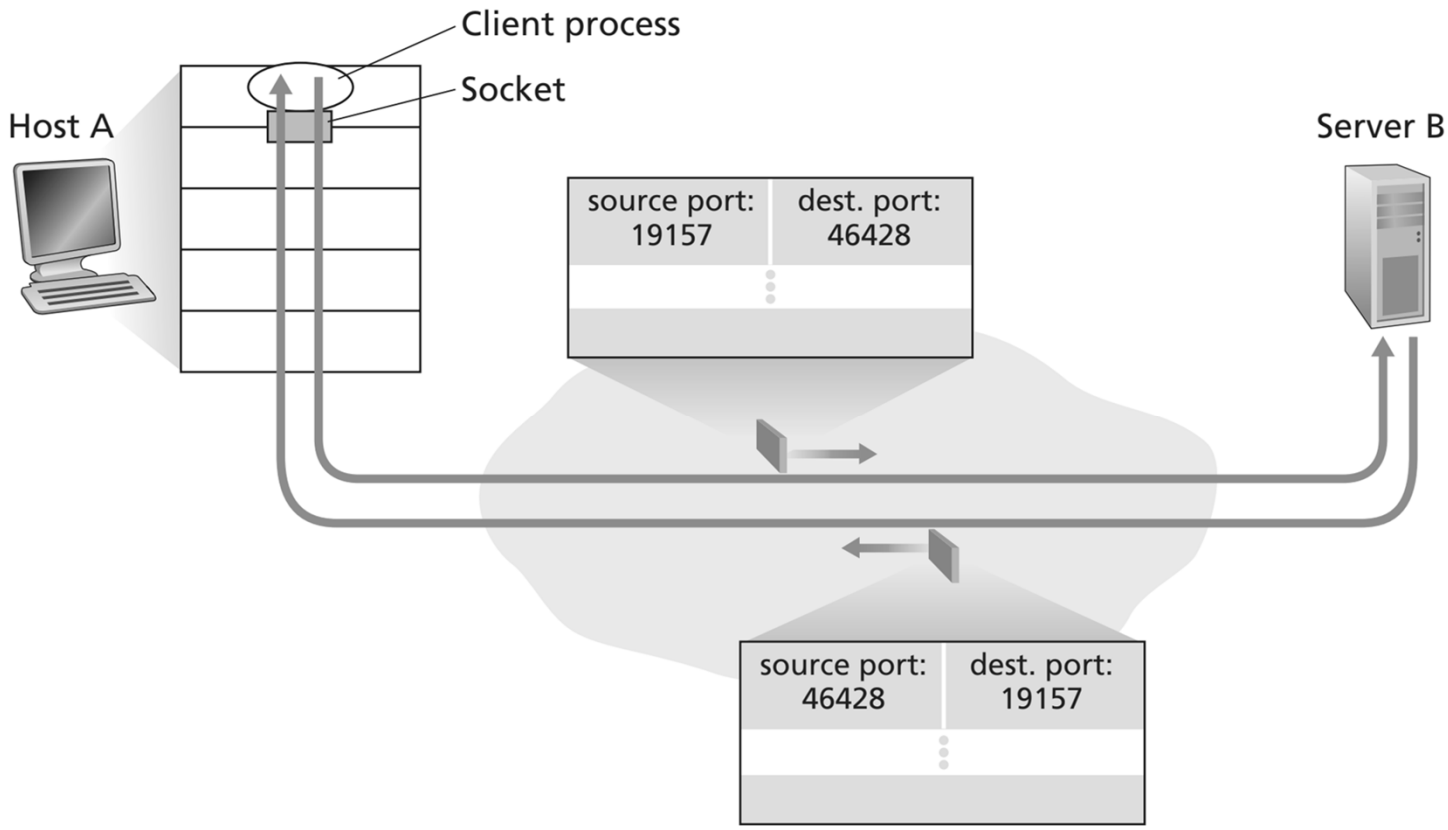
Port number (#)

- Địa chỉ cổng gồm 16b: từ 0 đến **???**.
  Trong đó:
  - Từ 0 đến 1023: số cổng được quy định sẵn (RFC 1700, http://www.iana.org [RFC 3232]) cho một số dịch vụ mạng: ví dụ: HTTP, FTP, DNS, SMTP: ???
  - Từ 1024 đến 49151: số hiệu cổng cho phép đăng ký
  - Từ 49152 đến 65535: số hiệu cổng được client lấy tự động khi thiết lập kết nối TCP hoặc UDP đến server

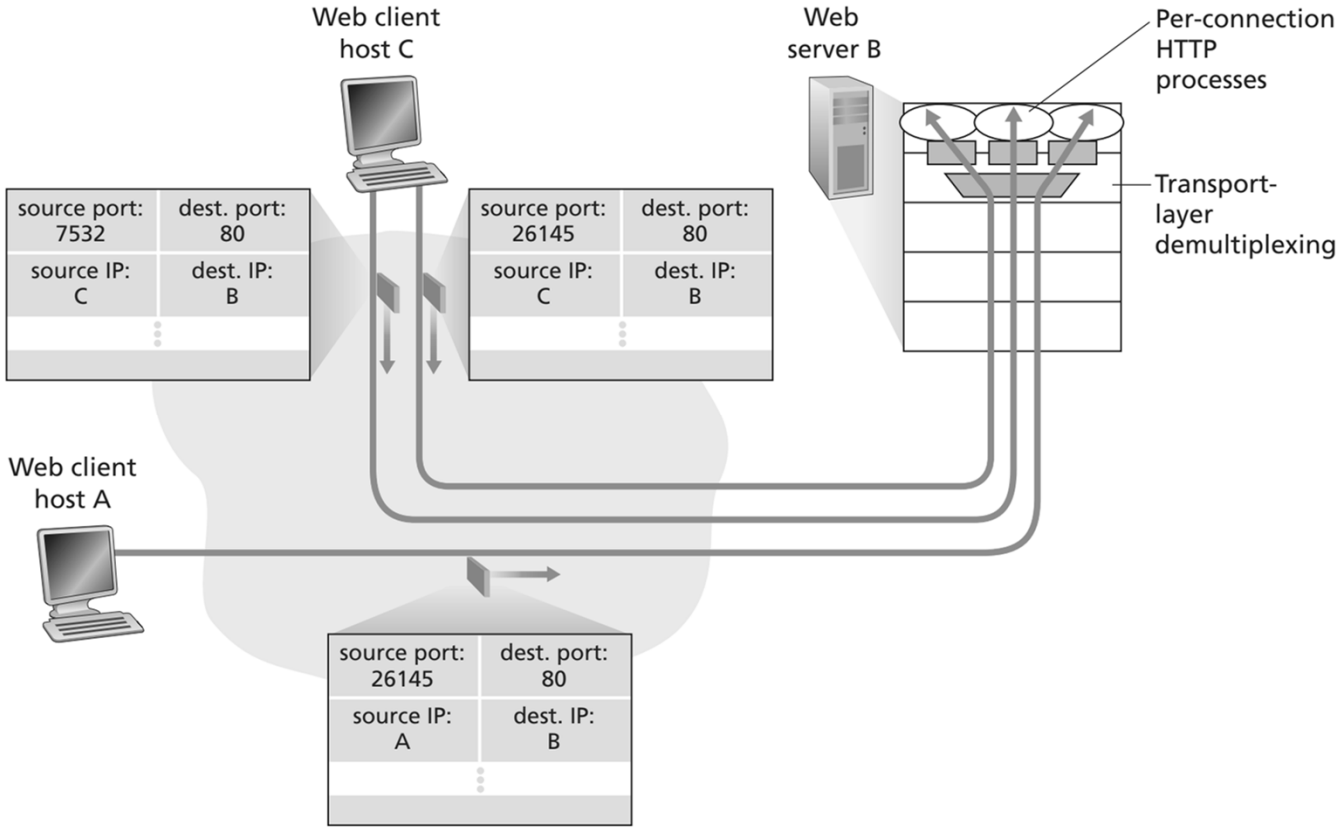




Port number


Client process
Socket
Host A
Server B
source port: 19157
dest. port: 46428
source port: 46428
dest. port: 19157

Minh họa về dồn kênh và phân kênh



3 segments, gửi tới IP address: B,
dest port: 80 được tách kênh tới các socket khác nhau

NỘI DUNG

1. Tổng quan về tầng giao vận
2. Dồn kênh và phân kênh
3. **Giao thức UDP**
4. Truyền dữ liệu tin cậy
5. Giao thức TCP

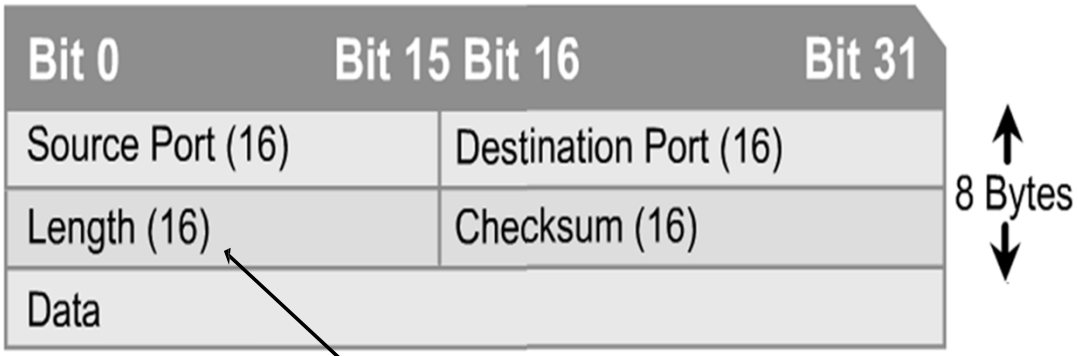
GIAO THỨC UDP

a. Cấu trúc UDP segment
b. UDP checksum

GIAO THỨC UDP

- RFC 768
- Giao thức không hướng kết nối (connectionless)
  - Không cần thiết lập kết nối
- Tiêu đề gói tin nhỏ: 8 bytes (so với 20 bytes của TCP)
- UDP có những chức năng cơ bản gì?
  - Dồn kênh/ phân kênh
  - Phát hiện lỗi bit đơn giản (dùng checksum)

Cấu trúc UDP segment



Length: độ dài toàn bộ gói UDP segment (bao gồm cả header), tính theo bytes

UDP checksum

Mục đích: phát hiện lỗi bit trong segment đã gửi

Bên gửi: coi dữ liệu của segment, bao gồm cả header gồm các từ 16 bít

- Tính giá trị bù một của tổng các từ 16 bit trong segment (RFC 1071)
- Giá trị checksum nhận được được đặt vào trường checksum trong gói dữ liệu UDP

Bên nhận:

- Tính checksum của segment đã nhận được
- Kiểm tra xem checksum đã tính có bằng với giá trị của trường checksum không:
  - Không - phát hiện lỗi
  - Có - không phát hiện lỗi.

Ví dụ: Internet Checksum

- Bên gửi: Giả sử dữ liệu truyền (48b): 0110011001100000 0101010101010101 1000111100001100
  - Chia thành 3 nhóm 16b:

    W1: 0110011001100000
    W2: 0101010101010101
    W3: 1000111100001100
  - W1+W2

    W1: 0110011001100000
    W2: 0101010101010101
    Sum: 1011101110110101
  - Cộng W3 vào Sum

    Sum: 1011101110110101
    W3: 1000111100001100
    Sum: 10100101011000001

  Chú ý: khi có 1 bit nhớ thì phải cộng vào kết quả

    Sum: 0100101011000001
    1
    Sum: 0100101011000010
    Bù 1: 1011010100111101 → ghi vào trường **Checksum**
- Bên nhận: dữ liệu nhận (chia thành các nhóm 16b) + checksum
  - Tổng toàn các bit 1: không lỗi
  - Tổng có từ 1 bit 0: có lỗi

Bài tập

1. Dữ liệu truyền đi gồm 2 từ 16b sau:

W1: 1110 1011 0001 1100

W2: 1010 0111 0110 0111

Tính checksum của từ này

2. UDP và TCP sử dụng bù 1 để tính checksum. Giả sử có các ba byte sau: 01010101, 01110000, 01001100

a) Tính giá trị bù 1 của tổng ba byte trên. Chú ý là UDP và TCP sử dụng các word 16-bit để tính checksum, nhưng bài tập này yêu cầu tính cho các byte 8-bit.

b) Khi dùng bù 1, phía nhận phát hiện lỗi như thế nào? Tại sao UDP dùng bù 1 để tính tổng, tại sao không chỉ tính tổng?

c) Lỗi 1 bít có thể không bị phát hiện không?

d) Lỗi 2 bít có thể không bị phát hiện không?

3. Giả sử rằng với UDP, phía nhận tính Internet checksum của segment gửi tới và thấy rằng kết quả trùng với giá trị trong trường checksum. Phía nhận có chắc chắn rằng không có bít nào bị lỗi không? Giải thích.

4. Cho hai từ 16b dạng Hexa: 5E8C và 9D7A, tính giá trị checksum.

5. UDP và TCP sử dụng bù 1 để tính checksum. Giả sử có các byte sau: 253A17B4H. Tính giá trị checksum của các byte đã cho.

Các vấn đề của UDP

- Không có kiểm soát tắc nghẽn
  - UDP cứ gửi dữ liệu nhanh nhất, nhiều nhất có thể

    ⇒làm Internet quá tải
- Không đảm bảo 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
- Tại sao UDP vẫn được sử dụng:
  - UDP không cần truyền lại gói tin khi bị lỗi, mất gói
  - UDP không thiết lập kết nối → không bị mất thời gian thiết lập kết nối
  - Không kiểm soát trạng thái kết nối → không cần theo dõi các thông số kết nối
  - Tiêu đề gói tin nhỏ (header): UDP 8B, TCP 20B

  → Phù hợp với các ứng dụng thời gian thực

Các ứng dụng sử dụng UDP

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

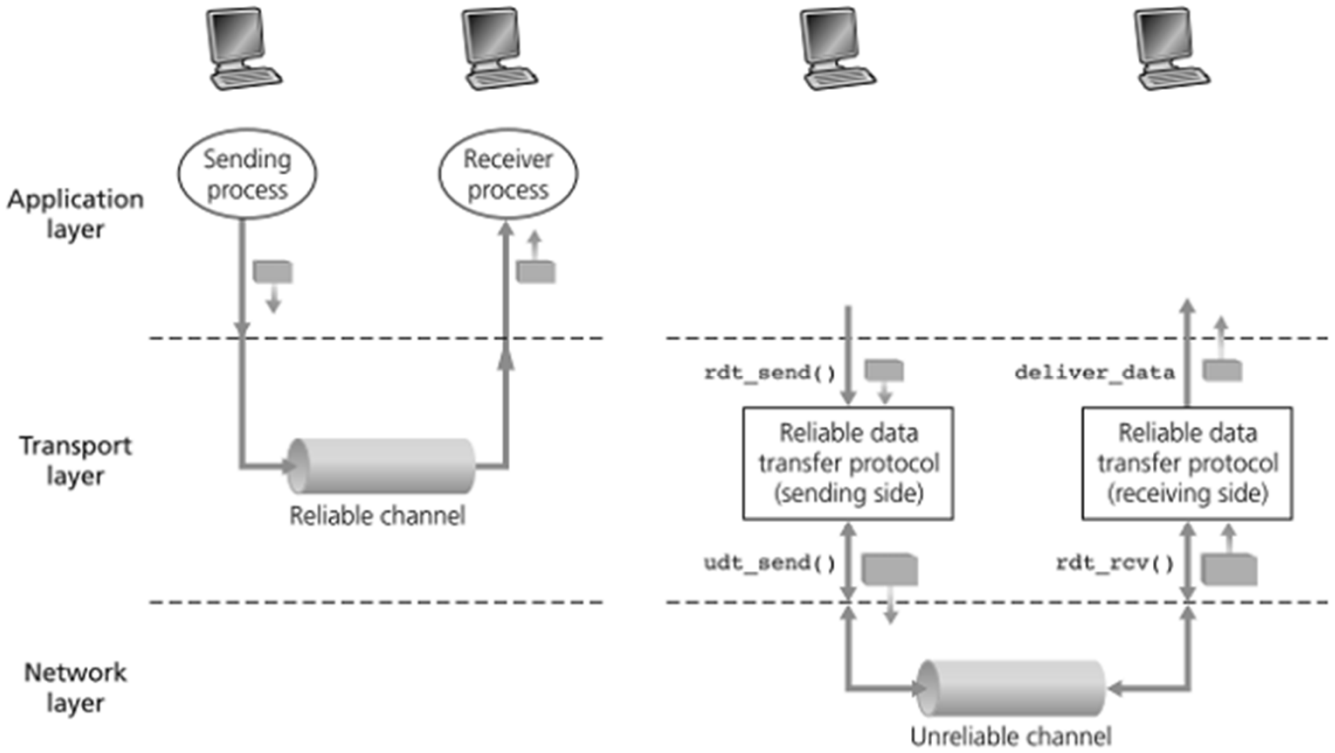
NỘI DUNG

1. Tổng quan về tầng giao vận
2. Dồn kênh và phân kênh
3. Giao thức UDP
4. **Truyền dữ liệu tin cậy**
5. Giao thức TCP

TRUYỀN DỮ LIỆU TIN CẬY

a. Xây dựng giao thức truyền dữ liệu tin cậy
b. Truyền dữ liệu theo pipeline

Sơ đồ cấu trúc của quá trình truyền dữ liệu tin cậy



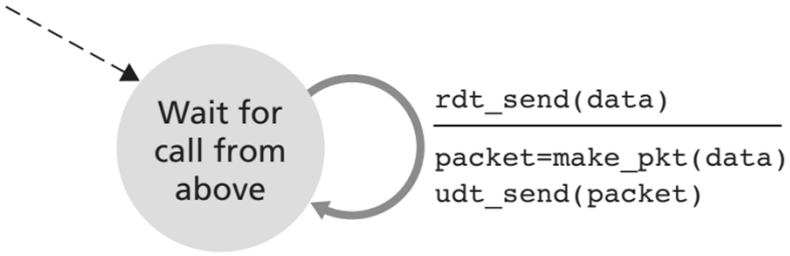
a) Dịch vụ cung cấp

b) Cài đặt dịch vụ

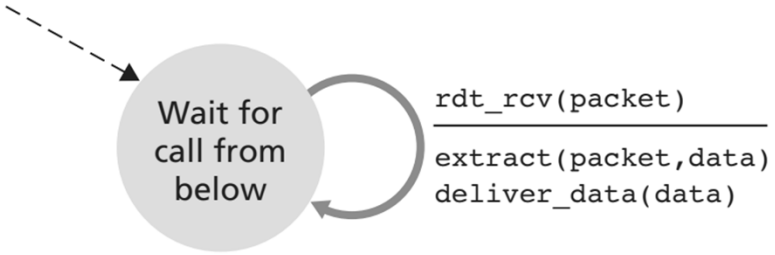
rdt - reliable data transfer protocol
udt - unreliable data transfer protocol

a. Xây dựng giao thức truyền dữ liệu tin cậy

- **Cách tiếp cận**: Từng bước phát triển giao thức truyền dữ liệu tin cậy giữa bên gửi và bên nhận
- **Với kênh truyền tin cậy hoàn toàn (rdt1.0)**:
  - rdt_sent: gửi dữ liệu
  - rdt_rcv: nhận dữ liệu
  - Vì là kênh truyền tin cậy nên bên gửi cứ gửi, bên nhận cứ nhận, gói tin chắc chắn truyền chính xác



a. rdt1.0: sending side

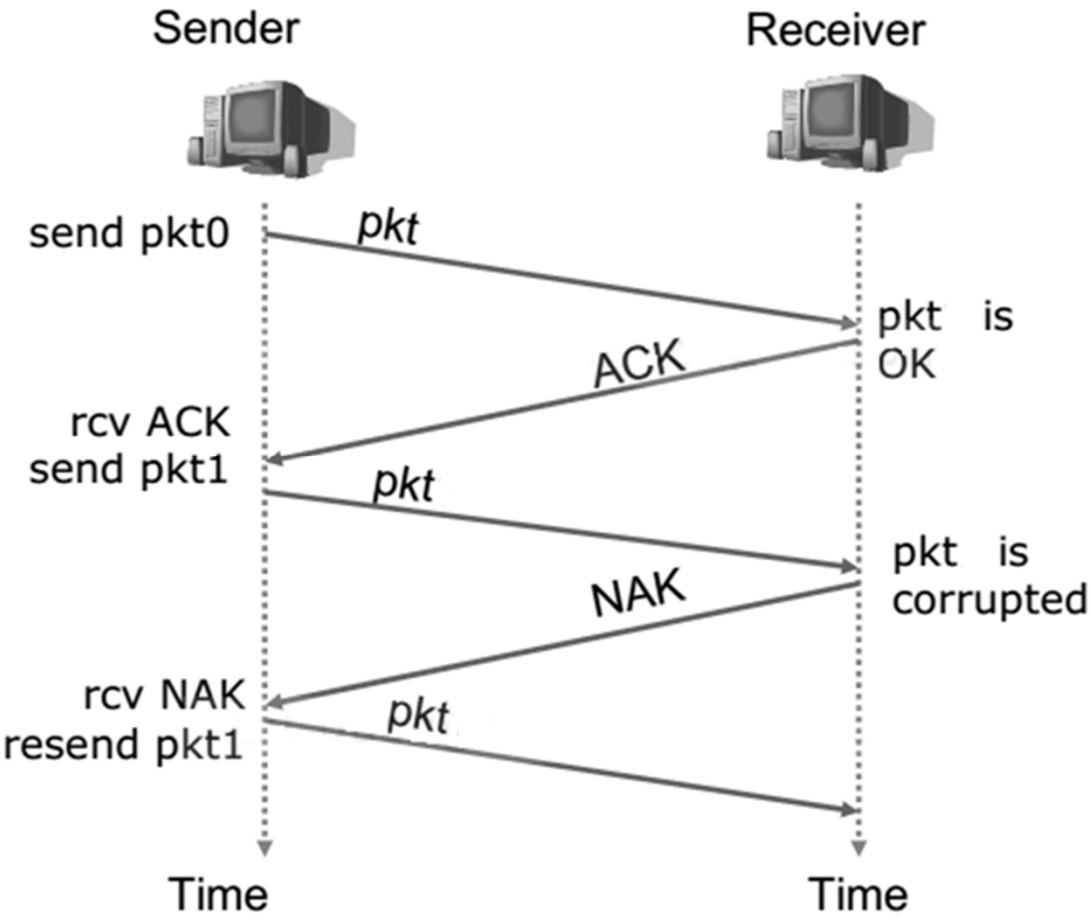


b. rdt1.0: receiving side

a. Xây dựng giao thức truyền dữ liệu tin cậy (tiếp)

- **Với kênh truyền có lỗi bit (rdt2.0)**: ba vấn đề
  - Phát hiện lỗi: checksum
  - Phản hồi từ bên nhận:
    - ACKnowledgements (ACKs): bên nhận thông báo cho bên gửi là gói tin đã nhận không có lỗi
    - Negative Acknowledgements (NAKs): bên nhận thông báo cho bên gửi là gói tin có lỗi
    - Chỉ cần bản tin 1b
  - Bên gửi truyền lại gói tin khi nhận được NAK

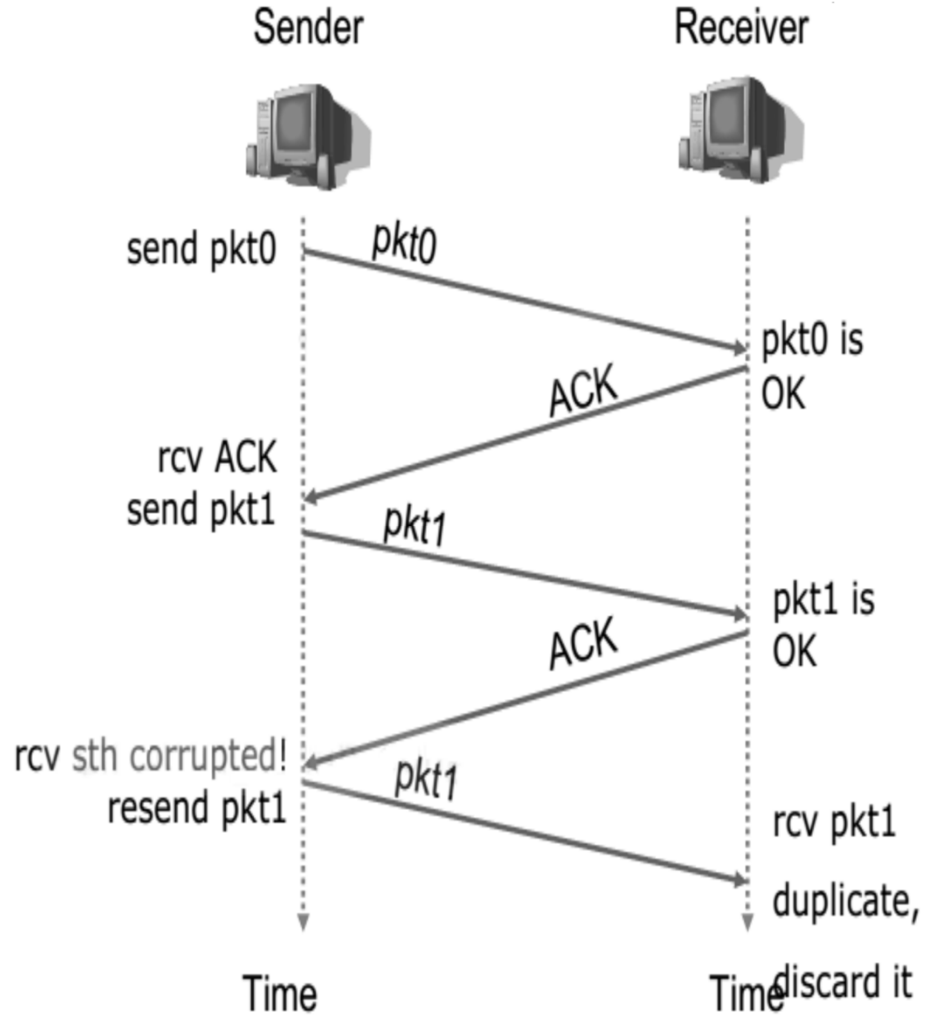
→ Tuy nhiên, nếu NAK và ACK truyền lỗi thì sẽ xử lý ntn?



a. Xây dựng giao thức truyền dữ liệu tin cậy (tiếp)

Lỗi ACK/NAK: giải pháp (rdt2.1)

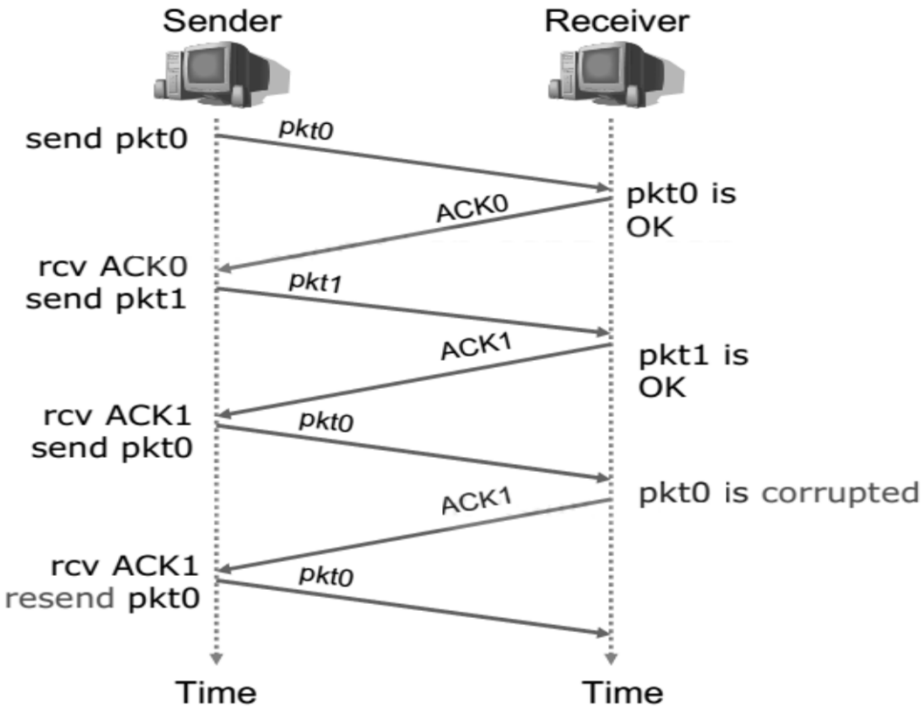
- Có 3 giải pháp:
  - Hỏi lại → dễ đi vào ngõ cụt
  - Sử dụng checksum để phát hiện và sửa lỗi → không khả thi nếu cả gói bị mất
  - Truyền lại khi ACK/NAK bị lỗi → lặp gói
- Xử lý việc lặp gói như thế nào?
  - Bên gửi: thêm số thứ tự gói
  - Bên nhận: hủy packet trùng lặp



a. Xây dựng giao thức truyền dữ liệu tin cậy (tiếp)

Giải pháp không dùng NAK (rdt2.2)

- Dùng ACK cho gói tin cuối cùng đã nhận được đúng
- Nếu nhận 2 ACK cho cùng một gói dữ liệu=> bên gửi gửi lại gói dữ liệu

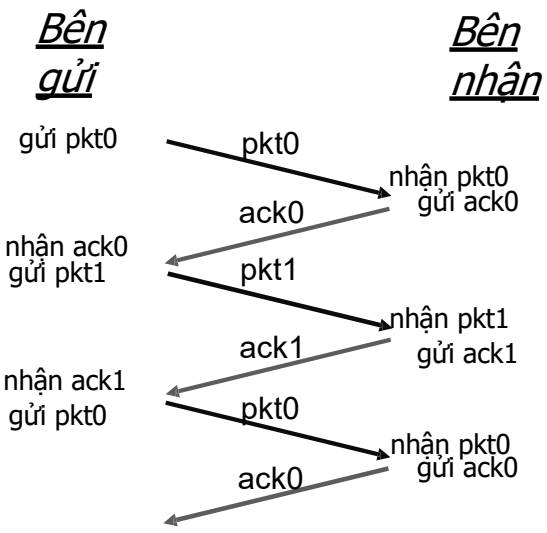


a. Xây dựng giao thức truyền dữ liệu tin cậy (tiếp)

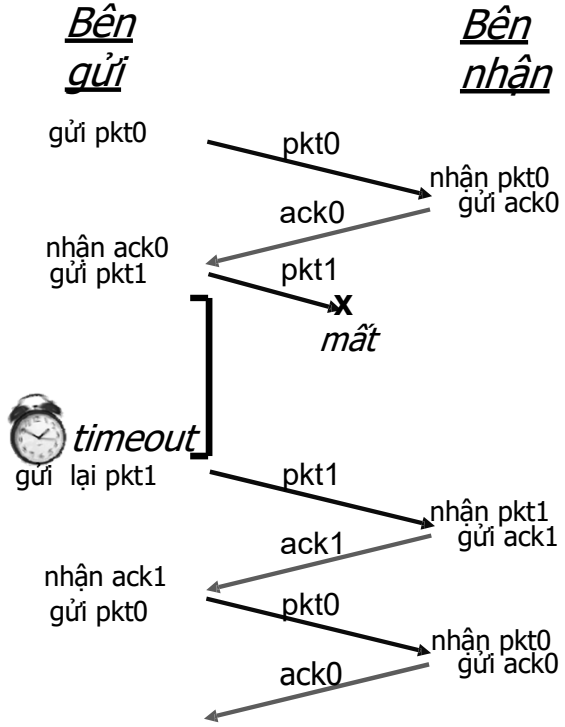
Kênh có lỗi và mất gói tin (rdt3.0)

- Với phiên bản rdt2.0, việc gói tin bị lỗi đã được giải quyết. Phiên bản rdt3.0 tiếp tục giải quyết vấn đề mất gói trên đường truyền
- Dữ liệu và ACK có thể bị mất
  - Bên nhận đợi một khoảng thời gian hợp lý (thời gian chờ -timeout) cho ACK
  - Truyền lại nếu không có ACK tới trong thời gian này
- Thời gian chờ là bao lâu?
  - Ít nhất là 1 RTT (Round Trip Time)
  - Mỗi gói tin gửi đi cần 1 timer → quá thời gian chờ, bên gửi sẽ gửi lại gói tin
- Nếu gói tin vẫn đến đích và ACK bị mất?
  - Truyền lại gói tin và xảy ra trùng lặp gói tin => rdt2.1 đã giải quyết được
- Nếu ACK đến trễ
  - Làm thế nào để xác định ACK đó là của gói tin nào? → thêm số thứ tự vào ACK

Minh họa

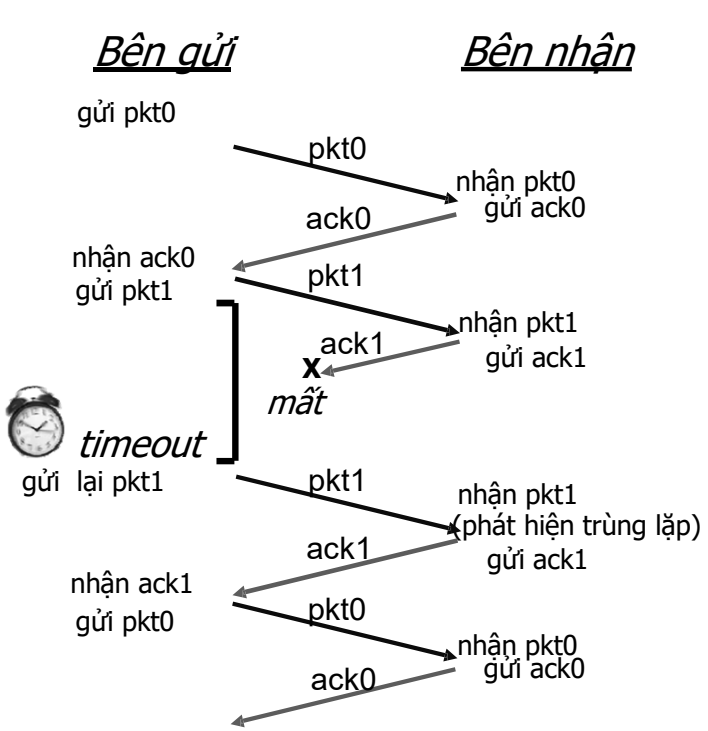


(a) không mất gói

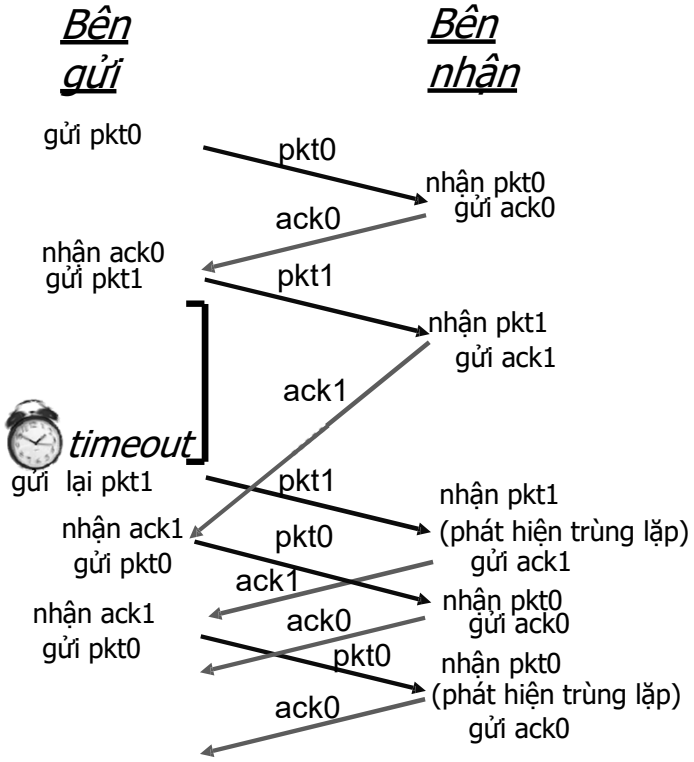


(b) có mất gói

Minh họa

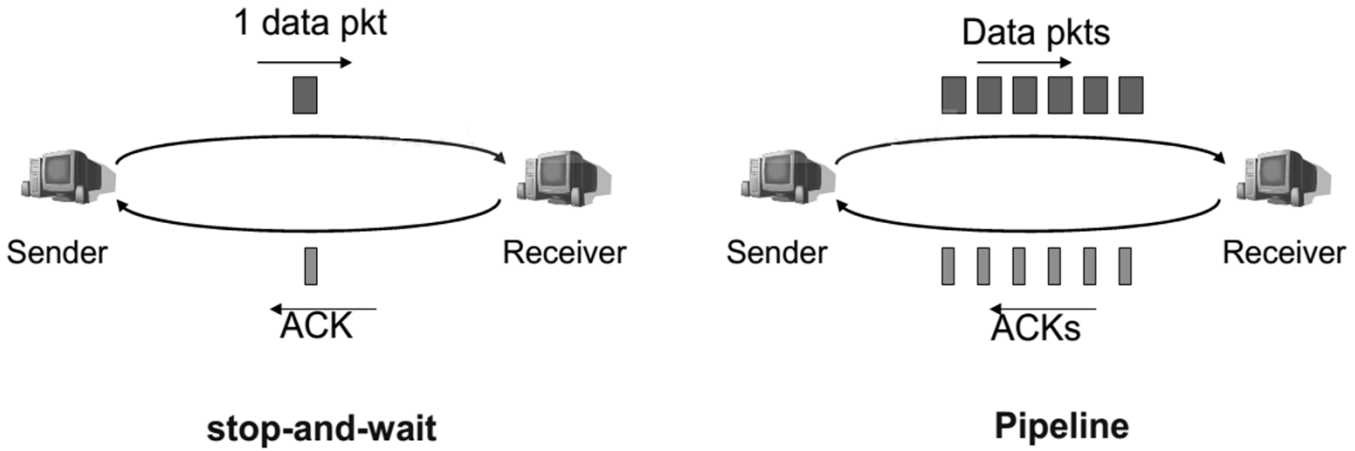


(c) mất ACK



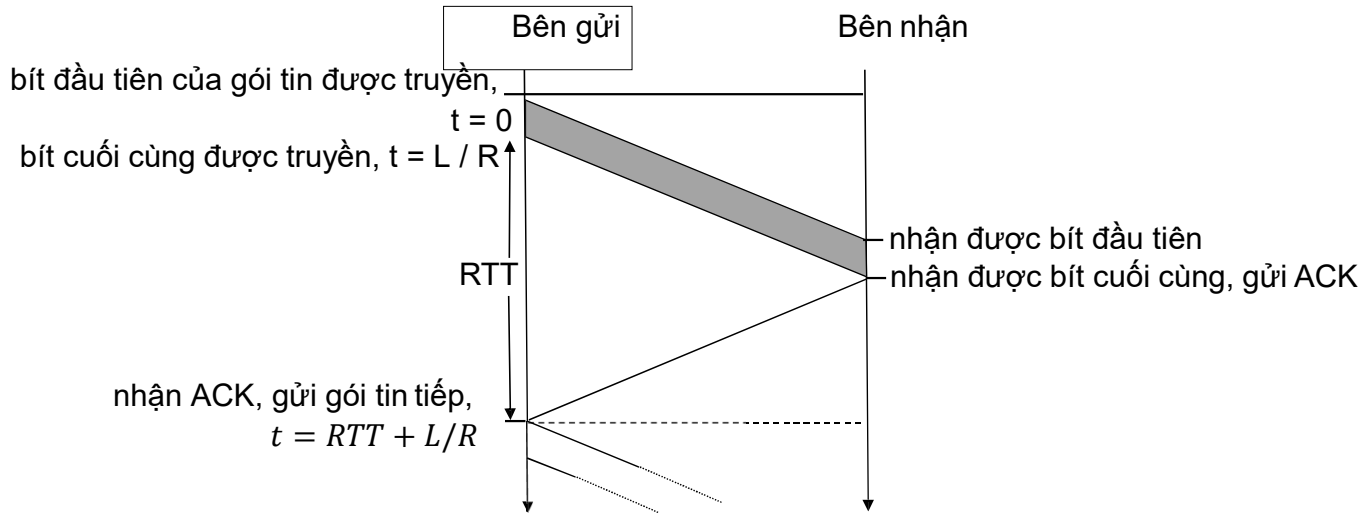
(d) timeout quá ngắn/ ACK tới trễ

b. Truyền dữ liệu theo pipeline



- Việc truyền stop-n-wait (truyền gói rồi đợi trả lời) làm lãng phí hiệu năng đường truyền
- Sử dụng giao thức truyền dữ liệu liên tục (Pipelined protocols): bên gửi gửi liên tục các gói tin mà không cần đợi ACK
  - Cần tăng dải đánh số thứ tự gói tin
  - Cần có vùng đệm tại bên gửi và bên nhận
- Hai dạng của pipelined protocols: *go-Back-N, selective repeat*

Hiệu năng của kiểu xử lý stop-and-wait



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

$$U_{sender} = \frac{L/R}{RTT + L/R}$$

Hiệu năng của kiểu xử lý stop-and-wait

- Liên kết 1 Gbps, độ trễ lan truyền 15 ms, gói tin 8000 bit

$$D_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9 bits\ /sec} = 8\ \mu s = 0.008\ ms$$

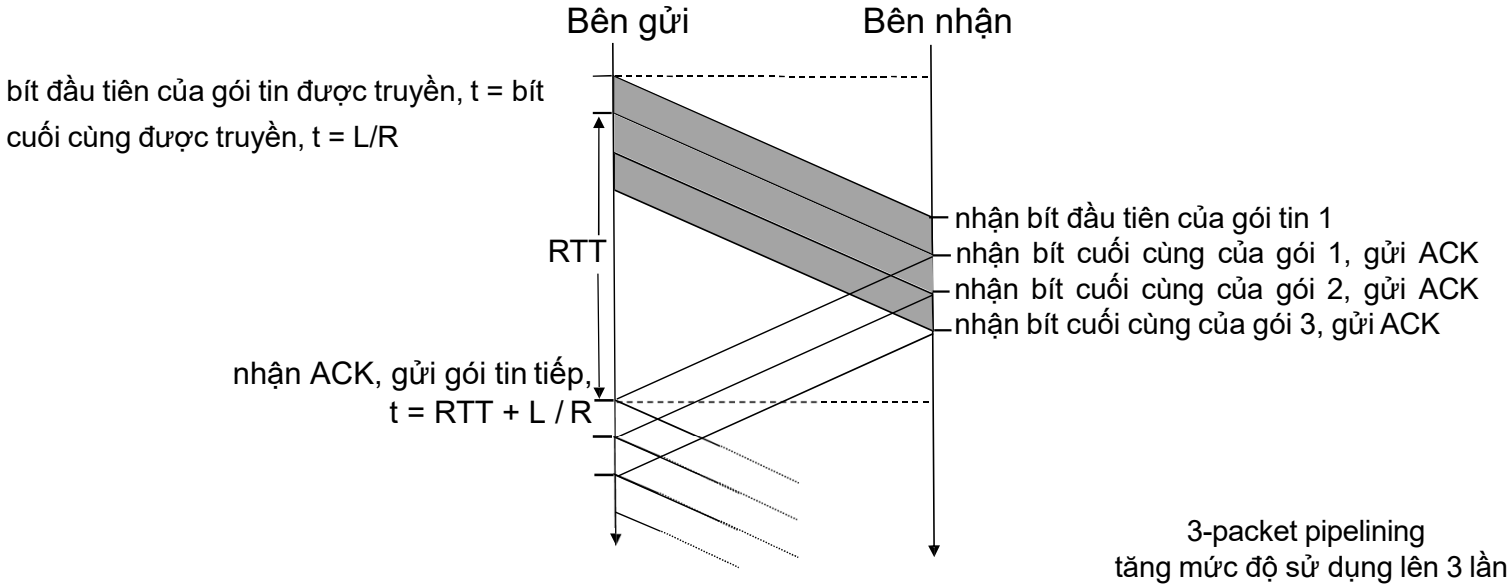
  - *RTT =15 +15 = 30 ms*
  - U_{sender}: *utilization* – phần thời gian bên gửi thực hiện gửi

$$U_{sender} = \frac{L\ /R}{RTT + L\ /R} \approx \frac{0.008}{30.008} \approx 0.000267$$

  - Nếu *RTT=30 ms*, gói tin 8000 bit mất *30 ms*:
  - 1s truyền được $8000/30.10^{-3} \approx 267$ Kbit trên liên kết 1Gbps = 10^9 bit/s

➔ Kiểu stop-and-wait hạn chế việc sử dụng tài nguyên băng thông của đường truyền

Hiệu năng của Pipeline: Tăng mức độ sử dụng

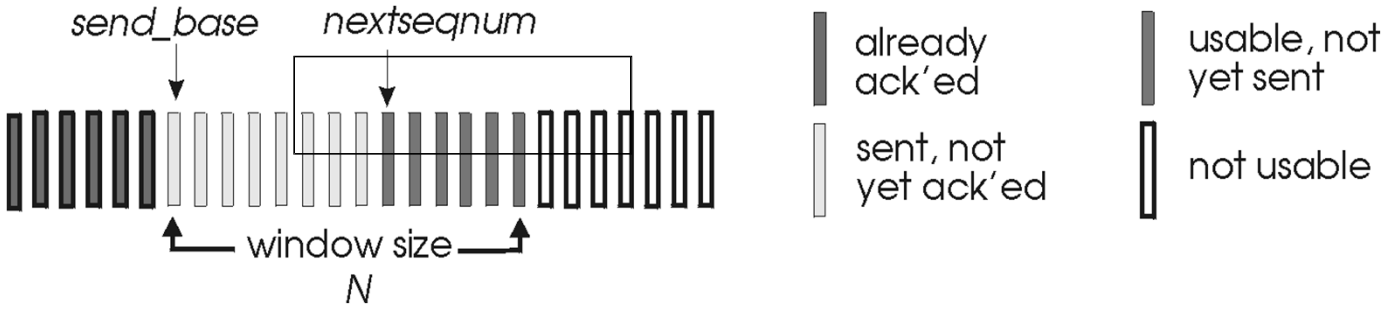


3-packet pipelining
tăng mức độ sử dụng lên 3 lần

$$U_{sender} = \frac{3L\ /R}{RTT + L\ /R} \approx \frac{0.024}{30.008} \approx 0.000801$$

Go-Back-N: Bên gửi

- "window" kích thước N,: có thể gửi N gói tin liên tục chưa được ack

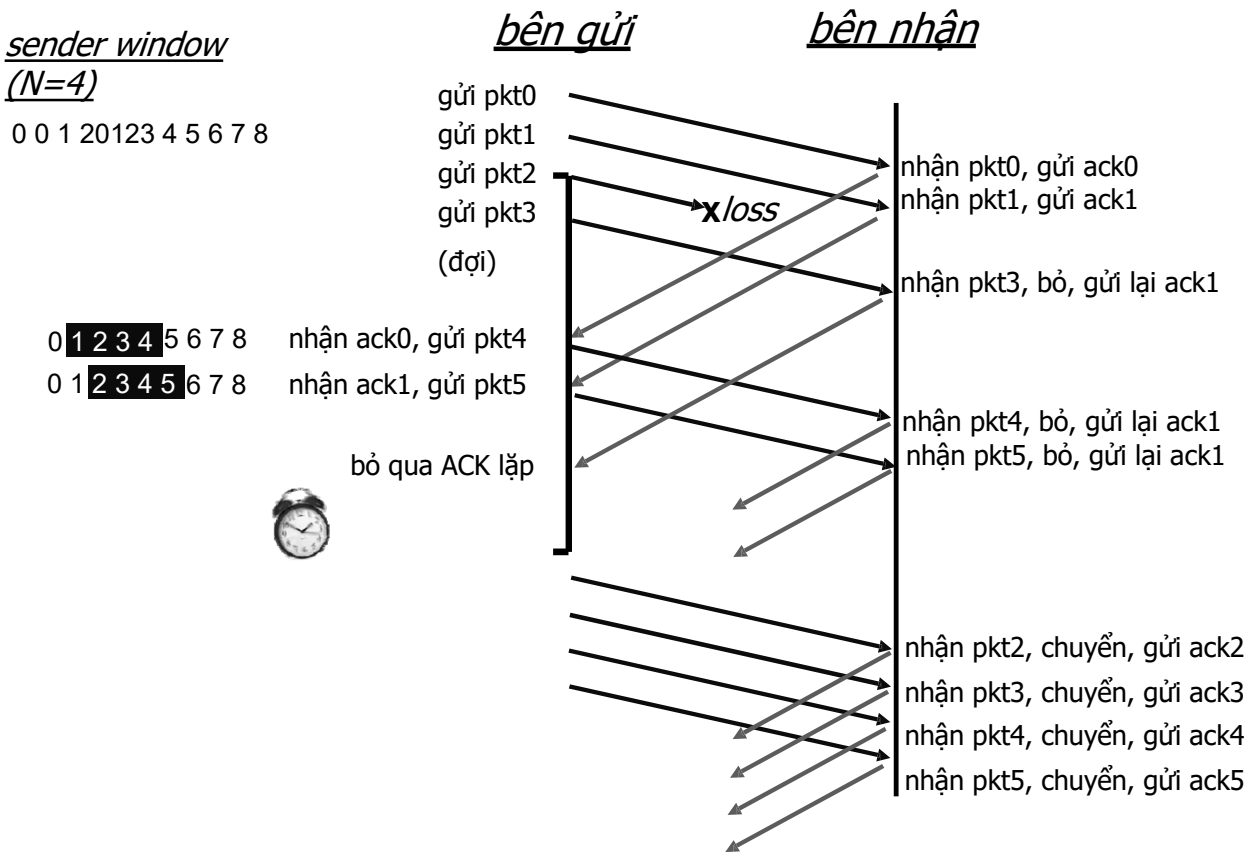


- Chỉ gửi các gói có số hiệu trong cửa sổ (gói 0 đến gói N-1), sau đó đợi xác nhận cho các gói tin này trước khi tiếp tục gửi gói, cửa sổ "dịch" sang phải mỗi khi nhận được ACK
- ACK(n): xác nhận tất cả các gói tin tới seq # n đã được nhận - *"cumulative ACK"*
- Đặt đồng hồ cho gói tin cũ nhất chờ ack
  - Khi hết *timeout(n):* gửi lại gói tin n và tất cả gói tin có seq # lớn hơn trong window

Go-Back-N: Bên nhận

- Chỉ gửi 1 xác nhận ACK cho gói tin có số hiệu lớn nhất đã nhận được theo đúng thứ tự.
- Với các gói tin không theo thứ tự:
  - Hủy bỏ -> không lưu vào vùng đệm
  - Xác nhận lại gói tin với số hiệu lớn nhất còn đúng thứ tự

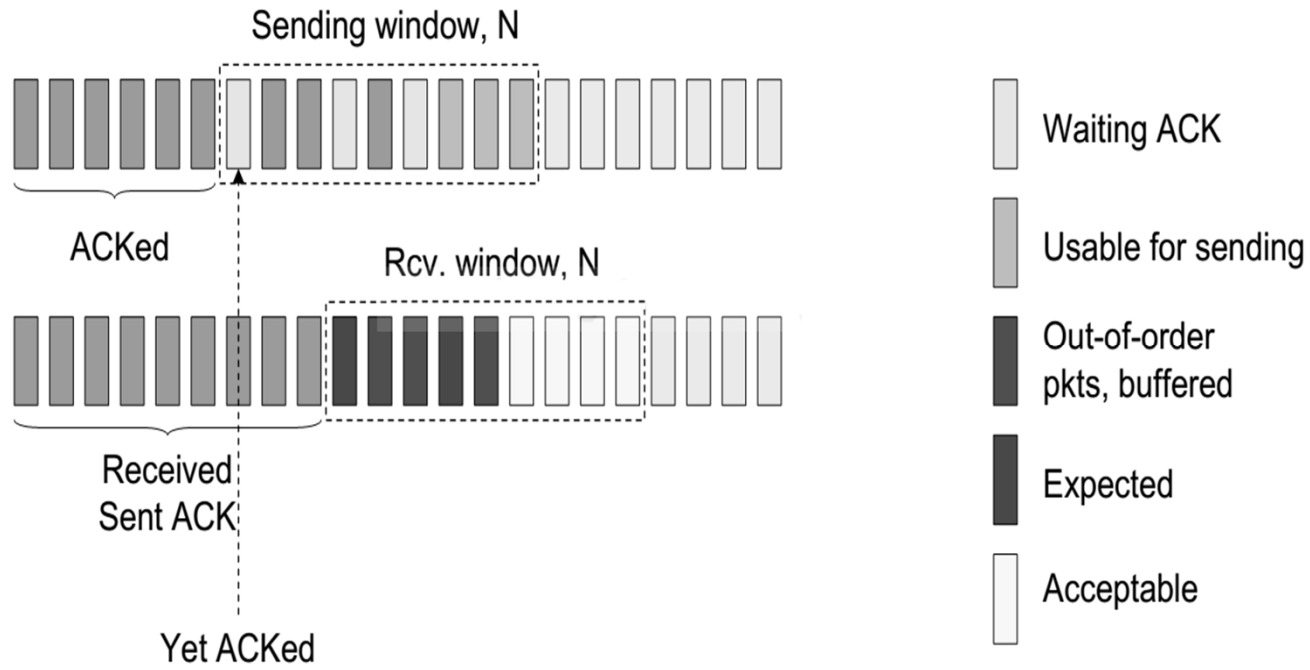
GBN: Ví dụ


sender window
(N=4)
0 0 1 20123 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
bỏ qua ACK lặp
bên gửi
bên nhận
gửi pkt0
gửi pkt1
gửi pkt2
gửi pkt3
(đợi)
nhận ack0, gửi pkt4
nhận ack1, gửi pkt5
x loss
nhận pkt0, gửi ack0
nhận pkt1, gửi ack1
nhận pkt3, bỏ, gửi lại ack1
nhận pkt4, bỏ, gửi lại ack1
nhận pkt5, bỏ, gửi lại ack1
nhận pkt2, chuyển, gửi ack2
nhận pkt3, chuyển, gửi ack3
nhận pkt4, chuyển, gửi ack4
nhận pkt5, chuyển, gửi ack5

Selective repeat

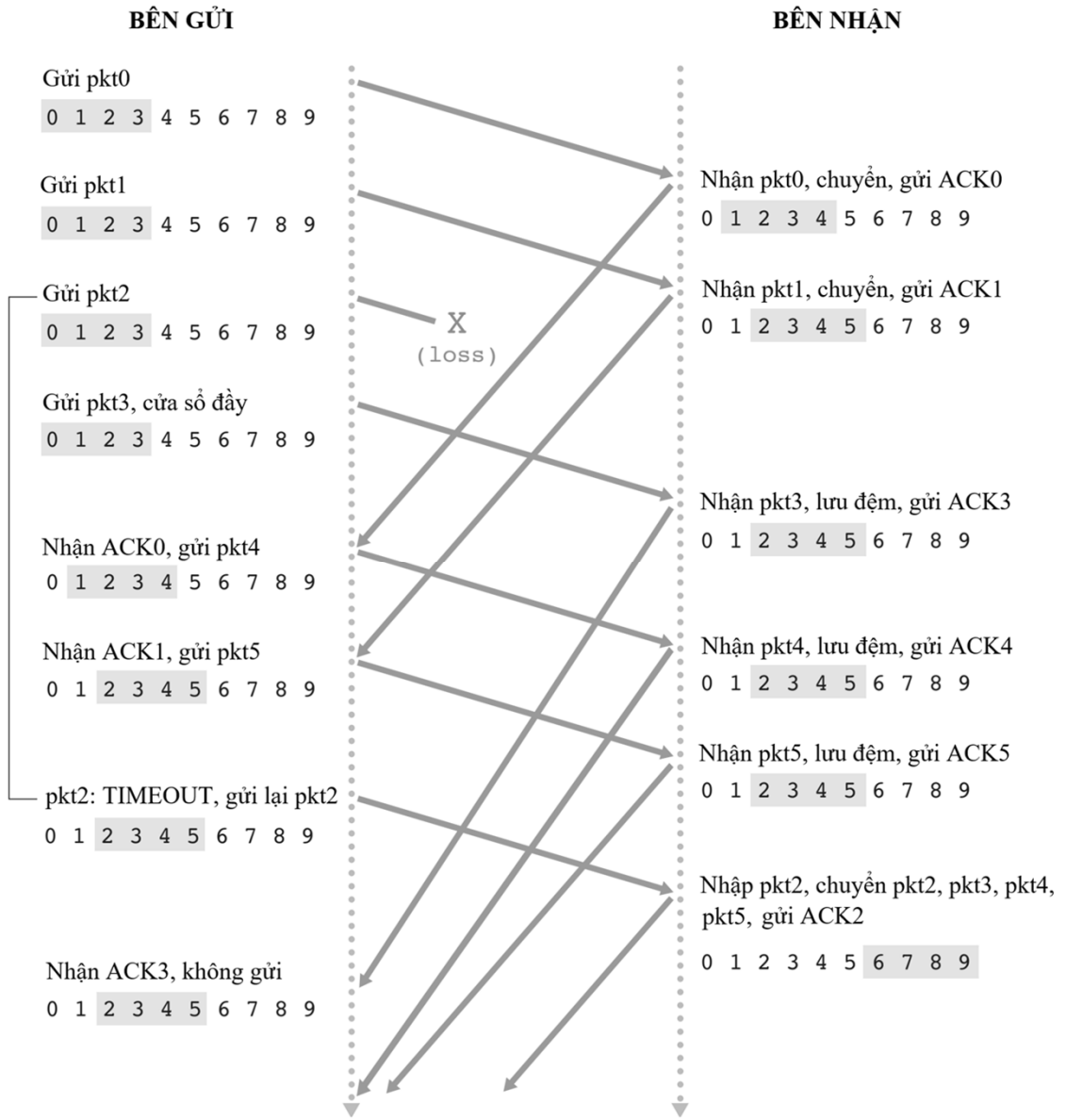
- Bên gửi: có thể gửi đến N gói tin chưa ack, sau đó chờ xác nhận
- Bên nhận: gửi xác nhận ack riêng lẻ cho từng gói tin nhận đúng
  - Chứa gói tin vào vùng đệm, khi cần, để chuyển đảm bảo thứ tự cho tầng trên
- Bên gửi: khi nhận được ACK cho từng gói sẽ dịch cửa sổ sang phải, sau đó truyền gói tiếp theo
- Bên gửi: không nhận được ACK
  - Đặt đồng hồ cho từng gói tin chưa được ack
  - Sau pkt n timeout thì truyền lại pkt n

Selective repeat



- Tổ chức vùng đệm để sắp xếp các gói tin theo đúng thứ tự để truyền cho tầng trên

Selective repeat



Truyền tin tin cậy – Tổng kết

CƠ CHẾ	MỤC ĐÍCH	DỊCH VỤ
Checksum	Phát hiện lỗi bít	Đảm bảo dữ liệu không lỗi
Đánh số gói tin	Giúp cho bên nhận xác định được gói tin bị mất hoặc trùng lặp nhau	Gói tin đến đúng thứ tự, không trùng lặp
ACK	Bản tin được gửi từ bên nhận về bên gửi để báo cho bên gửi biết gói tin đã đến nơi	
NAK	Bản tin được gửi từ bên nhận về bên gửi để báo cho bên gửi biết là gói tin truyền đến không chính xác và cần được truyền lại.	
Timer	Sử dụng thời gian chờ (timeout) để truyền lại gói tin trong trường hợp gói tin hoặc ACK bị mất trên đường truyền. Tuy nhiên, trong trường hợp gói tin hoặc ACK đến muộn, việc truyền lại gói của bên gửi sẽ sinh ra hai gói giống hệt nhau ở bên nhận	Xử lý mất gói
Cửa sổ, đường ống	Cho phép truyền nhiều gói tin liên tục, cải thiện hiệu quả của đường truyền	

NỘI DUNG

1. Tổng quan về tầng giao vận
2. Dồn kênh và phân kênh
3. Giao thức UDP
4. Truyền dữ liệu tin cậy
5. **Giao thức TCP**

GIAO THỨC TCP

a. Tổng quan về TCP
b. Cấu trúc của TCP segment
c. Truyền dữ liệu tin cậy của TCP
d. Điều khiển luồng
e. Điều khiển tắc nghẽn

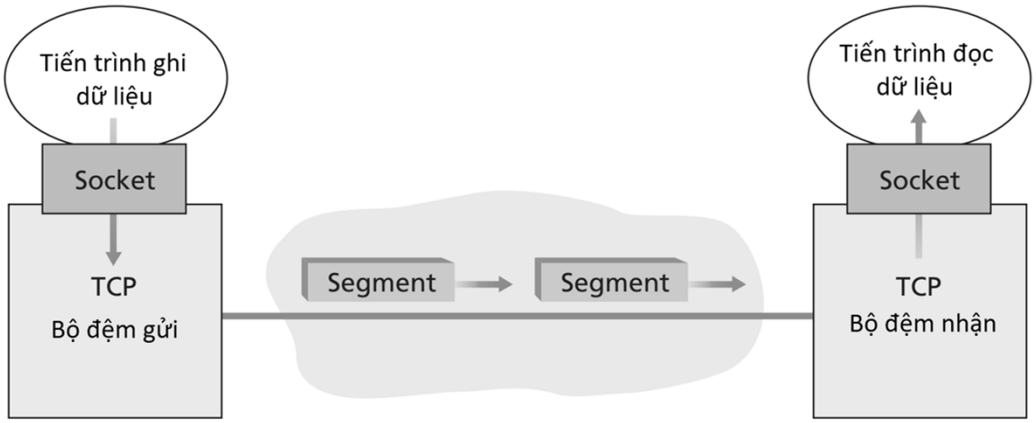
a. Tổng quan về TCP

RFCs: 793,1122,1323, 2018, 2581

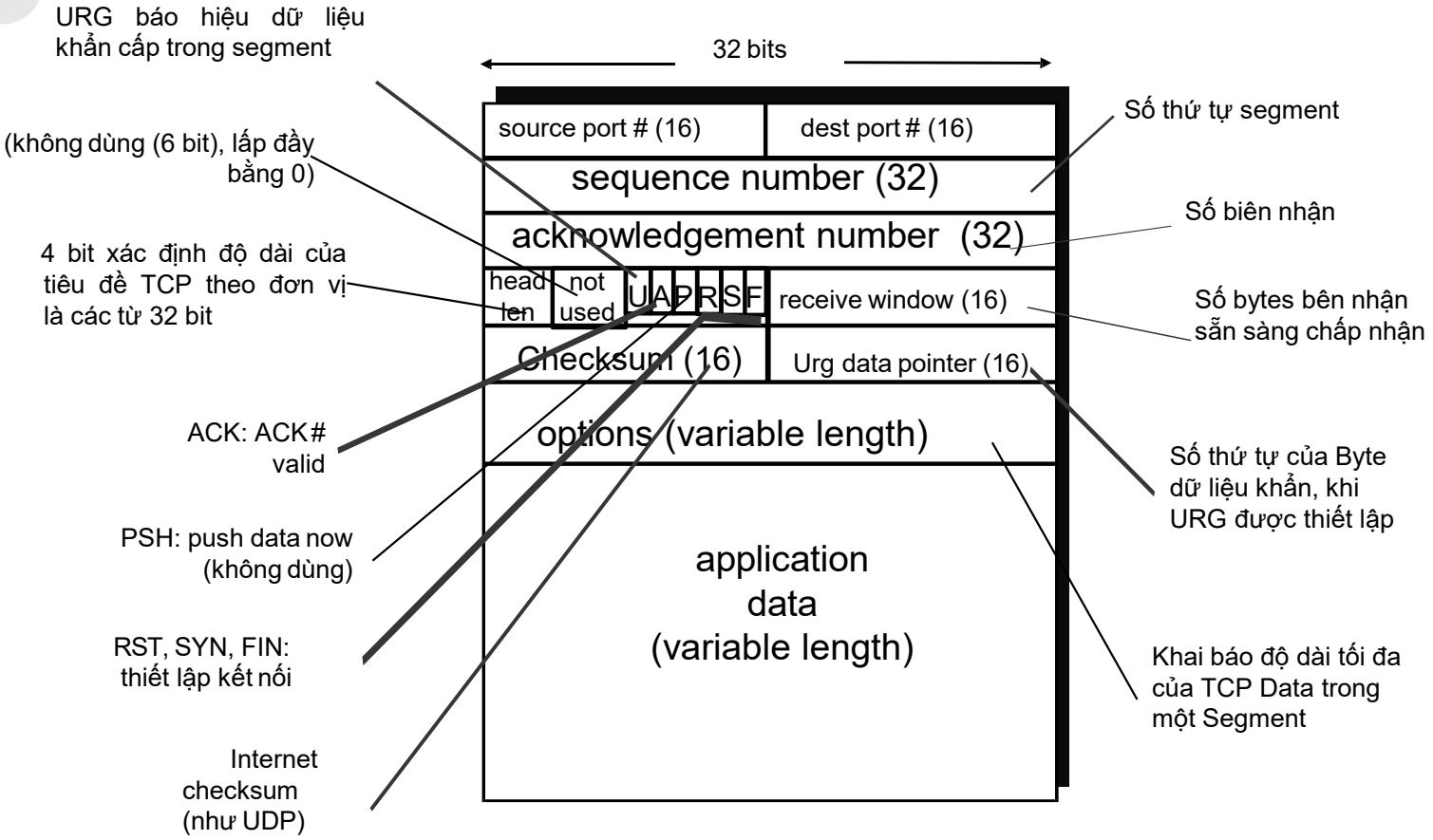
- TCP là giao thức truyền hướng kết nối (connection-oriented):
  - Bắt tay ba bước (handshaking):
- Giao thức truyền dữ liệu tin cậy, truyền theo chuỗi byte đảm bảo thứ tự*:*
  - Sử dụng vùng đệm
- Truyền theo kiểu liên tục (pipelined):
  - Tăng hiệu quả
- Điều khiển luồng:
  - Bên gửi không làm quá tải bên nhận
- Điều tắc nghẽn:
  - Việc truyền dữ liệu không nên làm tắc nghẽn mạng

Các bước truyền dữ liệu:

- Tiến trình gửi dl qua socket
- TCP lưu và bộ đệm (buffer)
- TCP lấy dần dữ liệu ra để gửi, kích thước dữ liệu tối đa mỗi lần gửi là MMS (maximum segment size)
- Khi gửi, TCP thêm Header và trước dữ liệu → **segment**
- Chuyển Segment xuống tầng mạng



b. Cấu trúc của TCP segment



Sequence number, acknowledgement number

- Giả sử rằng một tiến trình trong máy A muốn gửi một luồng dữ liệu đến một tiến trình trong máy B qua kết nối TCP. TCP trong máy A sẽ đánh số thứ tự cho từng byte của luồng dữ liệu. Mỗi luồng chia thành nhiều segment, mỗi segment có số thứ tự là số thứ tự của byte đầu tiên của segment
- Giả sử luồng dữ liệu gồm 500.000 byte, mỗi segment là 1.000 byte. TCP xây dựng 500 segment dữ liệu.
  - Segment đầu tiên được gán số thứ tự 0
  - Segment thứ hai được gán số thứ tự 1.000
  - Segment thứ ba được gán số thứ tự 2.000, v.v.
  - Mỗi số thứ tự như vậy được chèn vào trường **Sequence number** trong tiêu đề của phân đoạn TCP thích hợp.

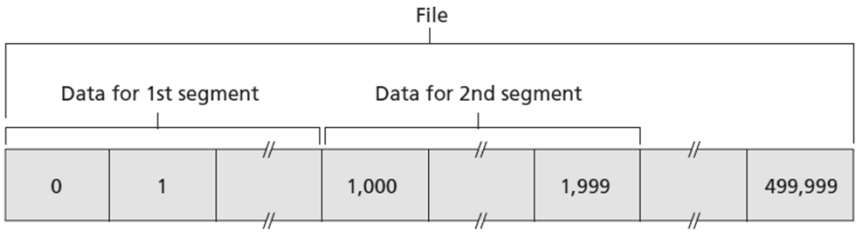


Figure 3.30 ♦ Dividing file data into TCP segments

Sequence number, acknowledgement number

- TCP là kênh truyền song công (full-duplex): bên A có thể gửi dữ liệu đến B đồng thời với bên B gửi dữ liệu cho bên A (trong cùng kết nối TCP)
- Giả sử tình huống: Bên B gửi dữ liệu đến bên A
- **Acknowledgement number (Ack #)**: là số thứ tự của byte tiếp theo mà máy A đang chờ máy B gửi tới
- Ví dụ 1:
  - A đã nhận được tất cả các byte được đánh số từ 0 đến 535 từ B và giả sử rằng A sắp gửi một segment đến B.
  - A đang chờ byte từ 536 và các byte tiếp theo trong luồng dữ liệu của B. Vì vậy, trường Ack # của segment A gửi đến B sẽ có giá trị = 536.
- Ví dụ 2:
  - A đã nhận được segment chứa byte 0 đến 535 và một segment khác chứa byte 900 đến 1.000 từ B (A chưa nhận được byte 536 đến 899) Khi đó, A vẫn đang chờ segment từ byte 536 để tạo lại luồng dữ liệu liên tục. Do đó, A sẽ gửi một segment tới B và thiết lập trường Ack #=536
  - Mở rộng: A đã nhận được segment thứ ba (byte 900 đến 1.000) trước khi nhận được segment thứ hai (byte 536 đến 899). Vì vậy, segment thứ ba không đúng thứ tự.

  → ***Máy nhận sẽ làm gì khi nhận được các segment không theo thứ tự trong kết nối TCP?***

  TCP không mô tả → người lập trình sẽ giải quyết

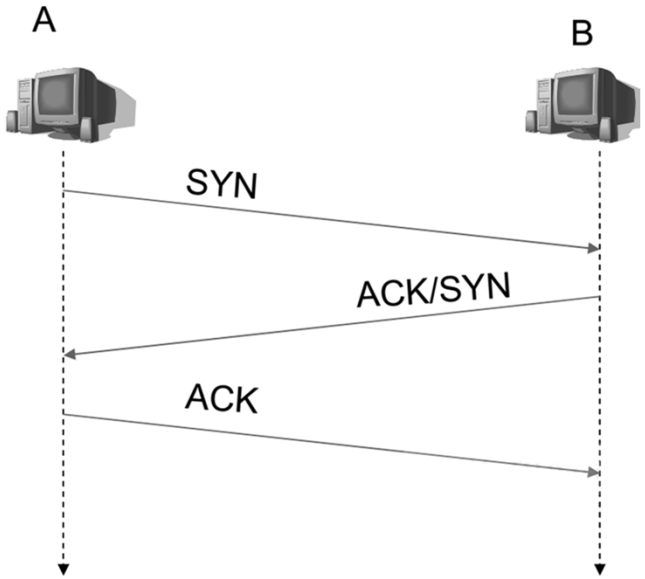
c. Truyền dữ liệu tin cậy của TCP

- TCP kiểm soát dữ liệu đã được nhận chưa (giao thức truyền tin tin cậy mục 4)
  - Seq. #
  - ACK (Acknowledgement number)
- Chu trình làm việc của TCP:
  - Thiết lập kết nối

    → Bắt tay ba bước
  - Truyền/nhận dữ liệu
  - Đóng kết nối

Thiết lập kết nối TCP: Giao thức bắt tay ba bước

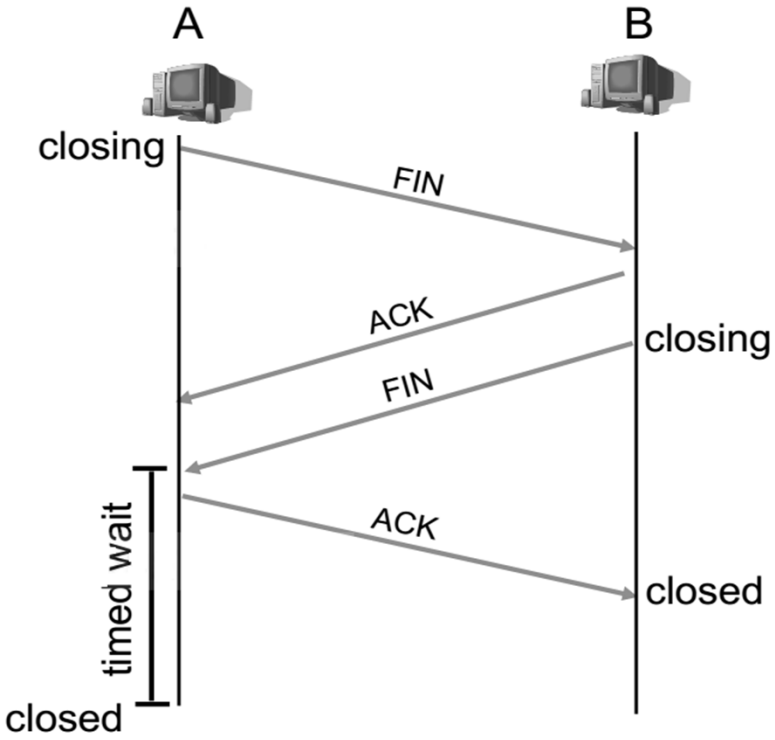
- Bước 1: A gửi TCP segment với SYN=1 cho B
  - Chỉ ra giá trị khởi tạo seq # của A
  - Không có dữ liệu
- Bước 2: B nhận SYN, trả lời bằng SYNACK
  - B khởi tạo vùng đệm
  - Chỉ ra giá trị khởi tạo seq. # của B
- Bước 3: A nhận SYNACK, trả lời ACK, có thể kèm theo dữ liệu



Ví dụ về đóng kết nối TCP

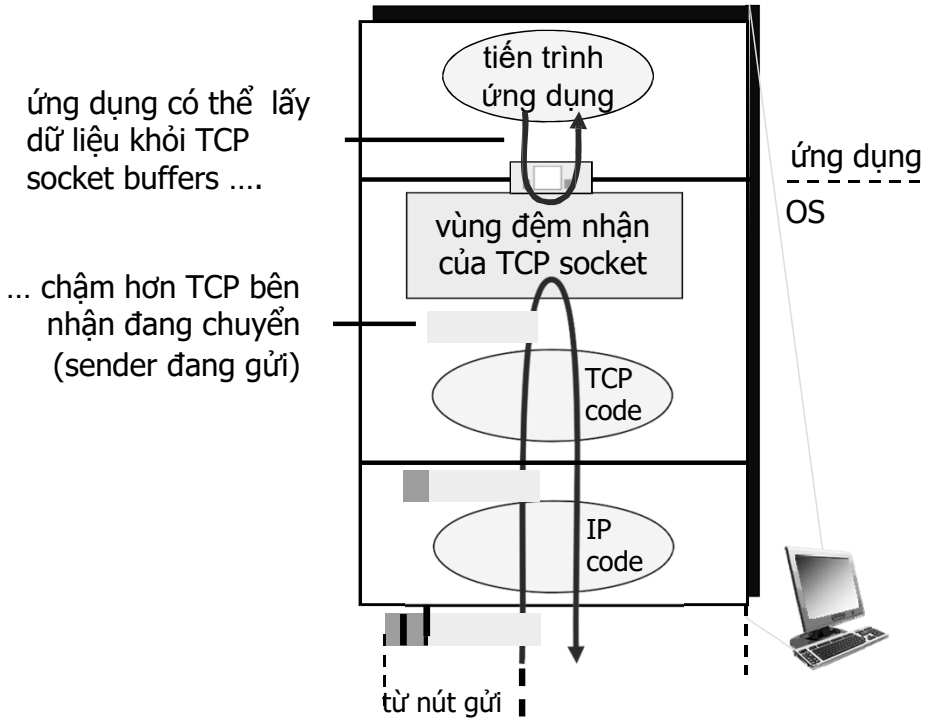
- Bước 1: A Gửi FIN cho B
- Bước 2: B nhận được FIN, trả lời ACK, đồng thời đóng liên kết và gửi FIN.
- Bước 3: A nhận FIN, trả lời ACK, vào trạng thái "chờ".
- Bước 4: B nhận ACK. Đóng liên kết.

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



d. Điều khiển luồng của TCP

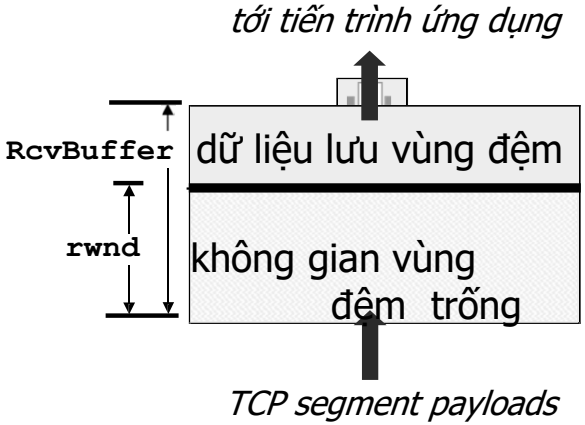
- *Điều khiển luồng:* nút nhận điều khiển nút gửi, để nút gửi không làm tràn vùng đệm của nút nhận bởi truyền quá nhiều và quá nhanh



ngăn xếp giao thức tại nút nhận

Điều khiển luồng của TCP

- Nút nhận thông báo không gian vùng đệm còn trống bằng cách đưa giá trị **rwnd (receive window)** trong TCP header của segment gửi từ nút nhận tới nút gửi
  - **Kích thước của RcvBuffer** được đặt thông qua tùy chọn của socket (thường mặc định 4096 byte)
  - Nhiều hệ điều hành tự động điều chỉnh **RcvBuffer**
- Nút gửi giới hạn dữ liệu chưa được ack bằng giá trị **rwnd** của nút nhận
- Đảm bảo vùng đệm nhận không bị tràn



Bộ đệm phía nút nhận

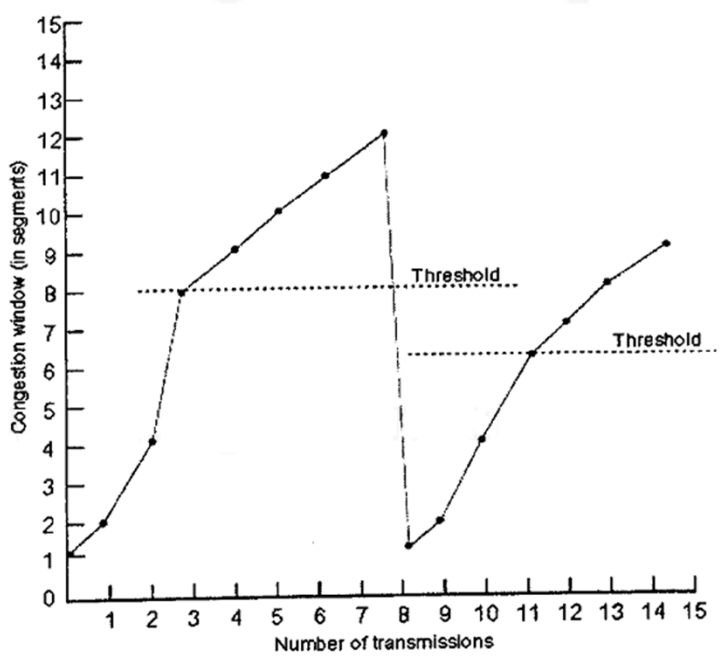
e. Điều khiển tắc nghẽn

- *Nguyên nhân tắc nghẽn*?
  - Quá nhiều cặp gửi - nhận trên mạng
  - Truyền quá nhiều làm cho mạng quá tải
  - Khác với điều khiển luồng!
- *Hậu quả của việc nghẽn mạng:*
  - Mất gói tin (tràn vùng đệm tại router)
  - Độ trễ lớn (đợi trong vùng đệm của router)
- *Giải pháp:* Bổ sung thêm thông tin cho điều khiển tắc nghẽn
  - Congestion window (cwnd): số lượng dữ liệu tối đa mà người gửi có thể gửi qua kết nối
  - Threshold (ngưỡng)

- *Chiến lược:*
  - Cwnd: là kích thước cửa sổ tối đa của TCP
  - Cwnd ban đầu được gán giá trị nhỏ → lượng thông tin gửi đi và chờ phản hồi (ACK) nhỏ
  - Khi nhận được ACK trong timeout → tăng Cwnd (**2 cách**)
  - Khi nghẽn mạng (không nhận đc ACK hoặc 3 ACK giống nhau) → giảm Cwnd

Nguyên tắc điều khiển tắc nghẽn

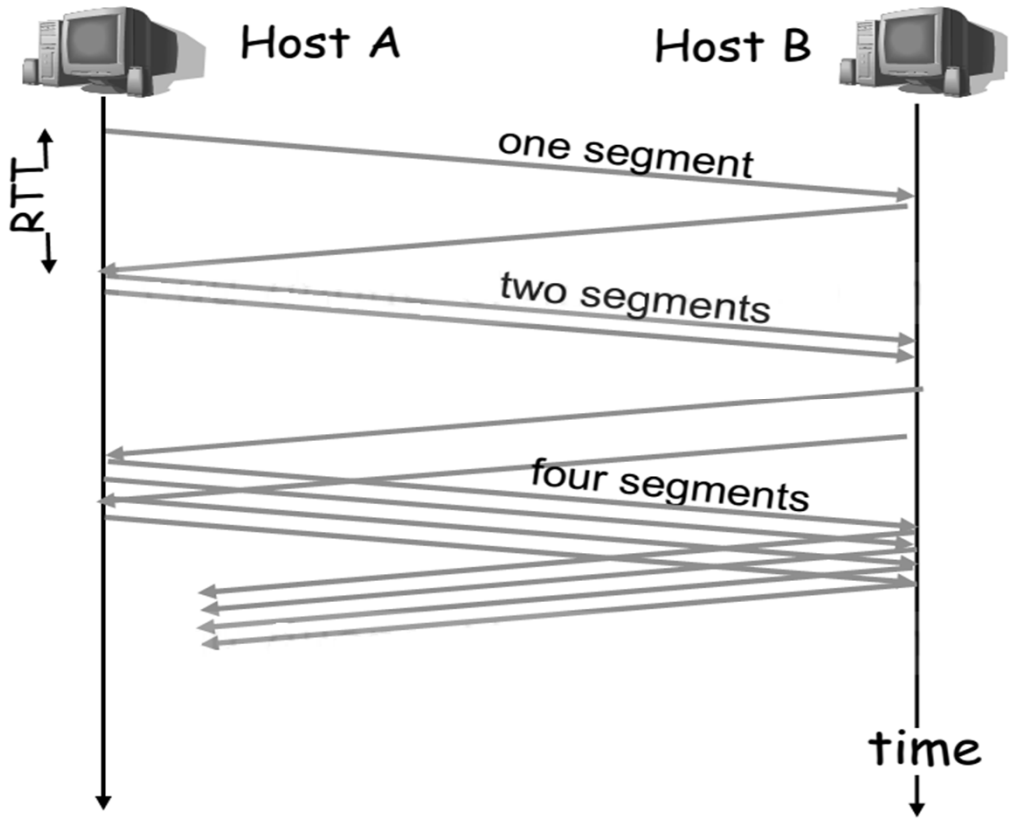
- Cách 1: Tăng dần tốc độ (slow-chat)
  - Tăng tốc độ theo hàm số mũ
  - Tiếp tục tăng đến một ngưỡng nào đó
- Cách 2: Tránh tắc nghẽn
  - Tăng dần tốc độ theo hàm tuyến tính cho đến khi phát hiện tắc nghẽn



Cách 1: TCP slow-start

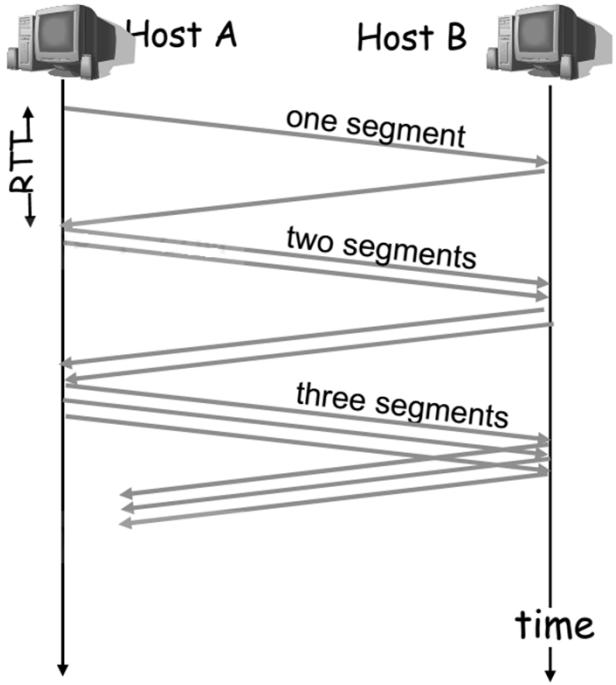
- Ý tưởng cơ bản
  - Đặt cwnd bằng 1 MSS (Maximum segment size)
  - Tăng cwnd lên gấp đôi
    - ✓ Khi nhận được ACK
  - Bắt đầu chậm, nhưng tăng theo hàm mũ
- Tăng cho đến một ngưỡng: threshold
  - Sau đó, TCP chuyển sang trạng thái tránh tắc nghẽn

Cách 1: TCP slow-start: minh họa



Cách 2: TCP tránh tắc nghẽn

- Ý tưởng cơ bản
  - Đặt cwnd theo cấp số cộng sau khi nó đạt tới **threshold**
  - Khi bên gửi nhận được ACK
    - ✓ Tăng cwnd thêm 1 MSS (Maximum segment size)



TCP phát hiện tắc nghẽn

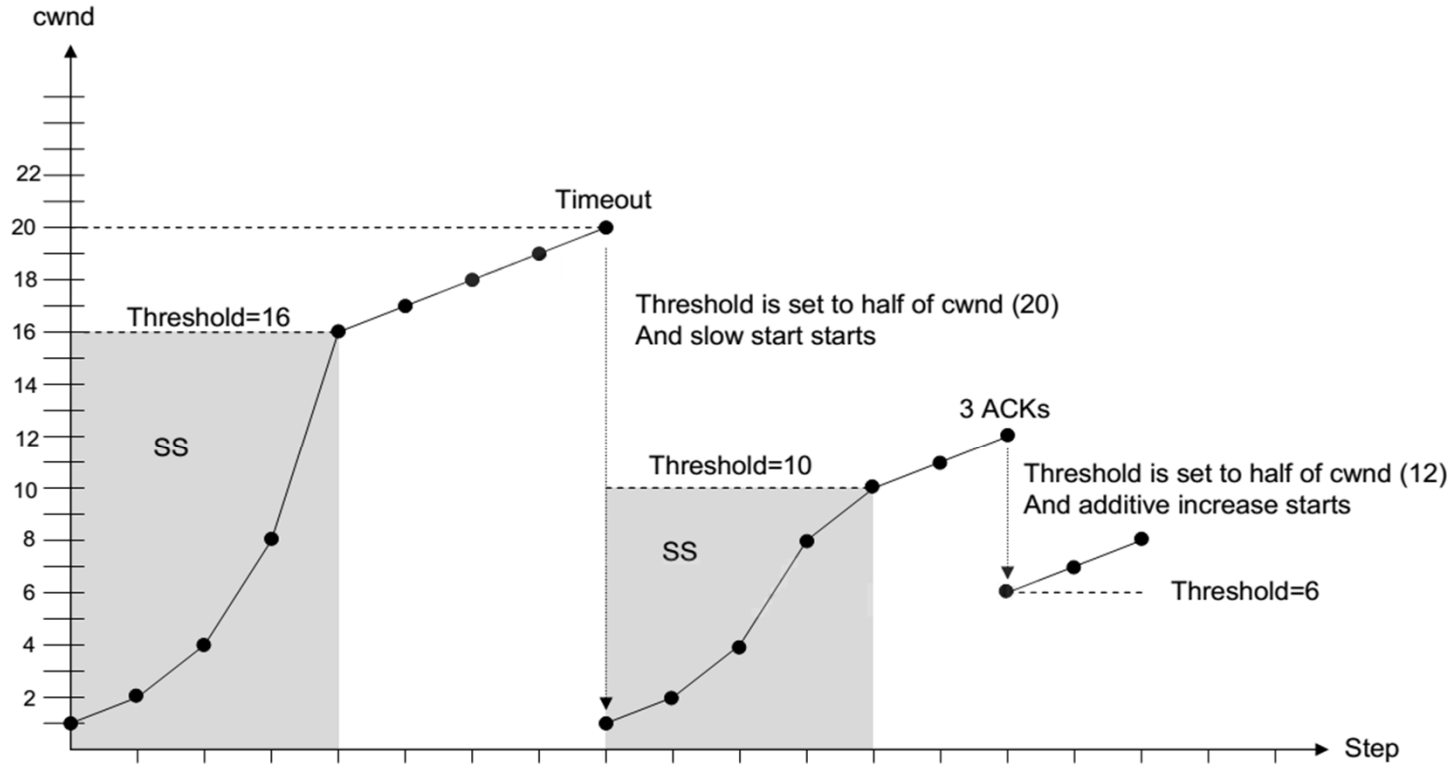
- Giảm tốc độ gửi
- Phát hiện tắc nghẽn?
  - Nếu như phải truyền lại

    → Có thể suy ra là mạng “tắc nghẽn”
- Khi nào thì phải truyền lại?
  - Timeout!
  - Cùng một số hiệu gói tin trong ACK

TCP phát hiện tắc nghẽn (tiếp)

- Khi có timeout của bên gửi
  - TCP đặt ngưỡng xuống còn một nửa giá trị hiện tại của cwnd
  - TCP đặt cwnd về 1 MSS
  - TCP chuyển về slow-start
- Nếu nhận được 3 ACK giống nhau
  - TCP đặt ngưỡng xuống còn một nửa giá trị hiện tại của cwnd
  - TCP đặt cwnd về giá trị hiện tại của ngưỡng cũ
  - TCP chuyển trạng thái “tránh tắc nghẽn”

Kiểm soát tắc nghẽn: minh họa



Chương 3: Tóm tắt

- Có hai dạng giao thức giao vận trong Internet
  - UDP: truyền dữ liệu không tin cậy
  - TCP: truyền dữ liệu tin cậy
- Các cơ chế bên trong các dịch vụ của tầng giao vận:
  - Multiplexing, demultiplexing
  - Truyền dữ liệu tin cậy
    - Báo nhận
    - Truyền lại
    - Điều khiển luồng
    - Điều khiển tắc nghẽn

Bài tập

Bài 1. Client A khởi tạo một phiên Telnet tới Server S. Tại cùng một thời điểm, Client B cũng khởi tạo một phiên Telnet tới Server S. Chỉ ra các giá trị có thể của cổng nguồn và cổng đích của

a) Segment gửi từ A tới S

b) Segment gửi từ B tới S

c) Segment gửi từ S tới A

d) Segments gửi từ S tới B

e) Nếu A và B là các host khác nhau, giá trị cổng nguồn trong segment gửi từ A tới S và giá trị cổng nguồn gửi từ B tới S có thể giống nhau được không?

f) Tương tự câu hỏi e) trong trường hợp A và B là cùng một host

MẠNG MÁY TÍNH

Hình ảnh và nội dung trong bài giảng này có tham khảo từ sách và bài giảng của TS. J.F. Kurose and GS. K.W. Ross