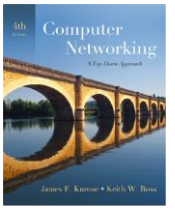
Chapter 3
Transport Layer



Computer Networking: A Top Down Approach 4th edition. Jim Kurose, Keith Ross Addison-Wesley, July 2007.

A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Chapter 3: tầng vận chuyển

Mục đích:

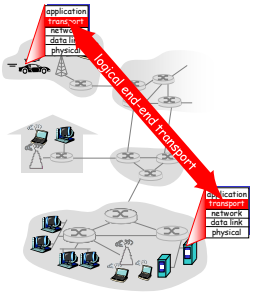
- Hiểu được nguyên lý dịch vụ vận chuyển:
  - Dồn kênh/phân kênh
  - Truyền dữ liệu tin cậy
  - Điều khiển luồng
  - Điều khiển tắc nghẽn
- Nghiên cứu các giao thức tầng vận chuyển trong Internet:
  - UDP: không hướng kết nối
  - TCP: hướng kết nối
  - TCP điều khiển tắc nghẽn

Chapter 3 Tổng quan

- 3.1 Các dịch vụ tầng vận chuyển
- 3.2 Dồn kênh và phân kênh
- 3.3 Vận chuyển không hướng kết nối: UDP
- 3.4 Nguyên lý về truyền dữ liệu tin cậy
- 3.5 Vận chuyển hướng kết nối: TCP
  - Cấu trúc phân đoạn
  - Truyền dữ liệu tin cậy
  - Điều khiển luồng
  - Quản trị kết nối
- 3.6 Nguyên lý điều khiển tắc nghẽn
- 3.7 TCP điều khiển tắc nghẽn

Giao thức và dịch vụ tầng vận chuyển

- Cung cấp *kết nối logic* giữa các tiến trình ứng dụng chạy trên các trạm khác nhau
- Các giao thức vận chuyển chạy trên các hệ thống cuối
  - Bên gửi: phân đoạn message và gửi qua tầng mạng
  - Bên nhận: lắp ráp các đoạn lại thành messages và chuyển qua tầng ứng dụng
- Có nhiều hơn một giao thức vận chuyển có hiệu lực cho tầng ứng dụng
  - Internet: TCP và UDP



Tầng vận chuyển và tầng mạng

- *Tầng mạng:* truyền tin mức logic giữa các trạm
- *Tầng vận chuyển:* truyền tin mức logic giữa các tiến trình
  - Tin cậy và làm tăng chất lượng các dịch vụ tầng mạng

Household analogy:

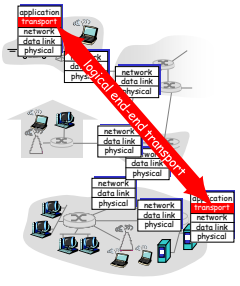
12 kids sending letters to 12 kids

- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

Các giao thức tầng vận chuyển Internet

- Xác thực để thực hiện (TCP)
  - Điều khiển tắc nghẽn
  - Điều khiển luồng
  - Thiết lập kết nối
- Không xác thực: UDP

- Các dịch vụ không hiệu lực:
  - Đảm bảo độ trễ
  - Đảm bảo băng thông



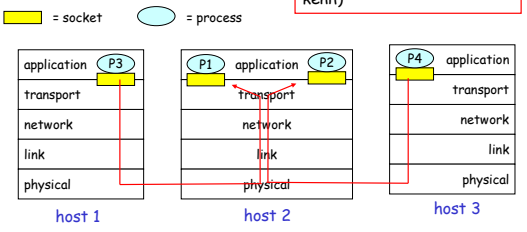
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Dồn kênh/phân kênh

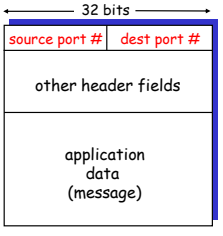
Phân kênh tại trạm nhận:
Phân phát chính xác các phân đoạn nhận được tới socket

Dồn kênh tại trạm gửi:
Tập hợp dữ liệu từ nhiều sockets và đóng gói dữ liệu với header (dùng cho phân kênh)



Phân kênh

- Trạm nhận các gói dữ liệu IP
  - Mỗi gói dữ liệu có địa chỉ IP nguồn và đích
  - Mỗi gói dữ liệu có mang 1 phân đoạn của tầng vận chuyển
  - Mỗi phân đoạn có số hiệu cổng của nguồn và đích
- Trạm dùng địa chỉ IP và số hiệu cổng phân đoạn trực tiếp tới socket tương ứng



TCP/UDP segment format

Phân kênh giao thức không kết nối

- Tạo các sockets với số hiệu cổng:

```
DatagramSocket mySocket1 = new
   DatagramSocket(12534);
DatagramSocket mySocket2 = new
   DatagramSocket(12535);
```

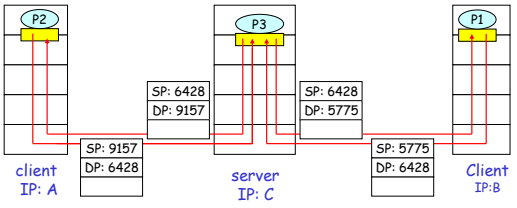
- UDP socket xác định bởi 2 thành phần:

(dest IP address, dest port number)

- Khi trạm nhận phân đoạn UDP:
  - Kiểm tra số hiệu cổng đích trong phân đoạn
  - Phân đoạn trực tiếp UDP tới socket theo số hiệu cổng
- Các gói dữ liệu IP với địa chỉ IP, số hiệu cổng nguồn khác nhau chuyển tới cùng socket

Phân kênh giao thức không kết nối (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

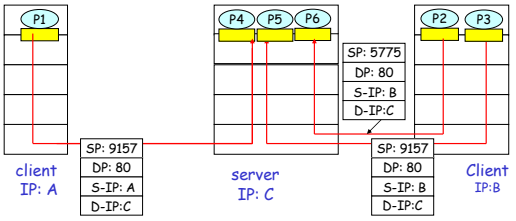


SP provides "return address"

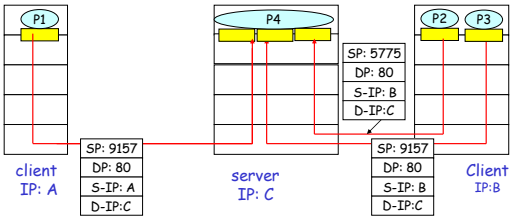
Phân kênh giao thức hướng kết nối

- TCP socket xác định bởi 4 thành phần:
  - Địa chỉ IP nguồn
  - Số hiệu cổng nguồn
  - Địa chỉ IP đích
  - Số hiệu cổng đích
- Trạm nhận sử dụng cả 4 giá trị để chuyển các phân đoạn tới socket tương ứng
- Trạm Server hỗ trợ đồng thời nhiều TCP sockets:
  - Mỗi socket xác định bởi 4 thành phần trong đó
- Web servers có các sockets khác nhau cho mỗi kết nối với client
  - HTTP không xác thực sẽ có socket khác nhau cho mỗi yêu cầu

Phân kênh giao thức hướng kết nối (cont)



Phân kênh giao thức hướng kết nối : phân luồng Web Server



Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

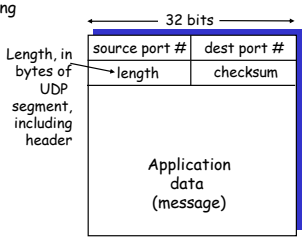
UDP: giao thức gói dữ liệu người sử dụng [RFC 768]

- Giao thức vận chuyển Internet không rườm rà, tinh giản tối thiểu
- Dịch vụ với nỗ lực cao nhất, UDP segments có thể:
  - Mất
  - Phân phát quá yêu cầu của ứng dụng
- *Không hướng kết nối:*
  - Không có thủ tục bắt tay giữa UDP gửi, nhận
  - Mỗi phân đoạn UDP thao tác độc lập với những phân đoạn khác

Ưu điểm của UDP:
- Không tạo ra kết nối (tăng độ trễ)
- Đơn giản: không trạng thái kết nối tại người gửi, người nhận
- Segment header bé
- Không điều khiển tắc nghẽn: UDP có thể xóa đi bất cứ lúc nào

UDP: (cont)

- Thường sử dụng cho các ứng dụng luồng đa phương tiện
  - Chấp nhận mất mát
  - Tỷ lệ chính xác
- UDP sử dụng cho
  - DNS
  - SNMP
- Vận chuyển tin cậy thông qua UDP: thêm sự tin cậy tại tầng ứng dụng
  - Phục hồi lỗi cụ thể tại tầng ứng dụng



UDP segment format

UDP checksum

Mục đích: phát hiện "lỗi" (e.g., lỗi bits) trong phân đoạn nhận được

Sender:
- Xử lý nội dung của phân đoạn như chuỗi 16-bit
- checksum: thêm 1 bit phần bù của 16-bit
- sender ghi giá trị của checksum vào trường UDP checksum

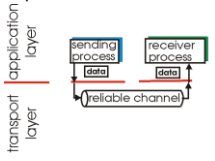
Receiver:
- Tính checksum của segment nhận được
- Kiểm tra nếu giá trị tính được bằng giá trị trong trường checksum :
  - NO – phát hiện có lỗi
  - YES – không phát hiện lỗi.

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Nguyên lý truyền dữ liệu tin cậy

- Quan trọng trong các tầng ứng dụng, vận chuyển, liên kết DL
- top-10 danh sách các chủ đề quan trọng của mạng máy tính!

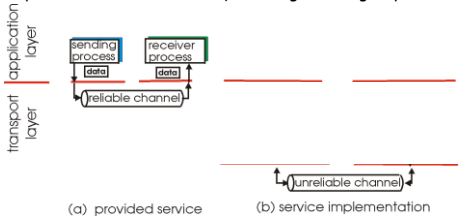


(a) provided service

- Các thuộc tính của kênh không tin cậy sẽ xác định sự phức tạp của giao thức truyền dữ liệu tin cậy (rdt)

Nguyên lý truyền dữ liệu tin cậy

- Quan trọng trong các tầng ứng dụng, vận chuyển, liên kết DL
- top-10 danh sách các chủ đề quan trọng của mạng máy tính!

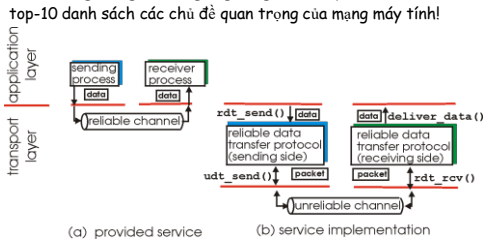


(a) provided service (b) service implementation

- Các thuộc tính của kênh không tin cậy sẽ xác định sự phức tạp của giao thức truyền dữ liệu tin cậy (rdt)

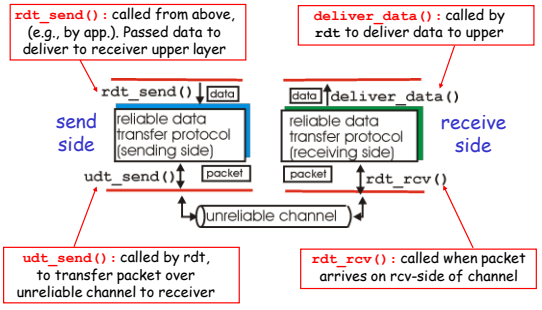
Nguyên lý truyền dữ liệu tin cậy

- Quan trọng trong các tầng ứng dụng, vận chuyển, liên kết DL
- top-10 danh sách các chủ đề quan trọng của mạng máy tính!



- Các thuộc tính của kênh không tin cậy sẽ xác định sự phức tạp của giao thức truyền dữ liệu tin cậy (rdt)

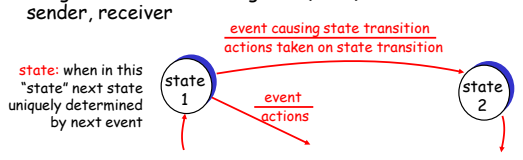
Truyền dữ liệu tin cậy: nhập môn



Truyền dữ liệu tin cậy: nhập môn

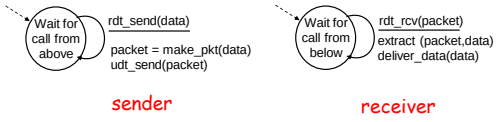
Công việc sẽ thực hiện:

- Tăng độ tin cậy phía gửi, nhận của giao thức truyền dữ liệu tin cậy (rdt)
- Xem xét chỉ một hướng truyền duy nhất
  - Nhưng thông tin điều khiển sẽ lưu thông cả hai hướng!
- Dùng một số hữu hạn trạng thái (FSM) xác định sender, receiver



Rdt1.0: truyền tin cậy thông qua một kênh tin cậy

- Dựa trên kênh có độ tin cậy hoàn hảo
  - Không lỗi bit
  - Không mất gói tin
- Phân tách thành FSMs cho sender, receiver:
  - sender gửi dữ liệu vào kênh
  - receiver đọc dữ liệu từ kênh

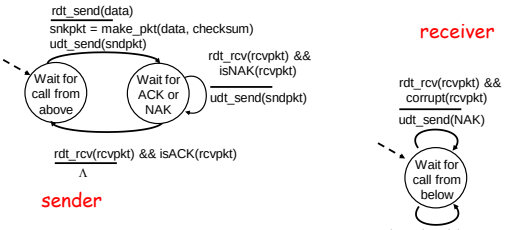


sender receiver

Rdt2.0: kênh với lỗi bit

- Truyền qua kênh bit trong gói dữ liệu có thể bị lật
  - checksum phát hiện lỗi bit
- Phục hồi lỗi:
  - *acknowledgements (ACKs):* receiver báo cho bên sender rằng gói nhận được tốt
  - *negative acknowledgements (NAKs):* receiver báo cho bên sender rằng gói nhận được có lỗi
  - sender gửi lại gói theo thông báo của NAK
- Kỹ thuật mới trong `rdt2.0` (beyond `rdt1.0`):
  - Phát hiện lỗi
  - receiver gửi lại: control msgs (ACK,NAK) rcvr->sender

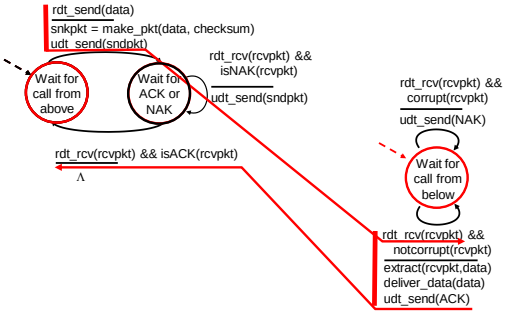
rdt2.0: đặc tả FSM



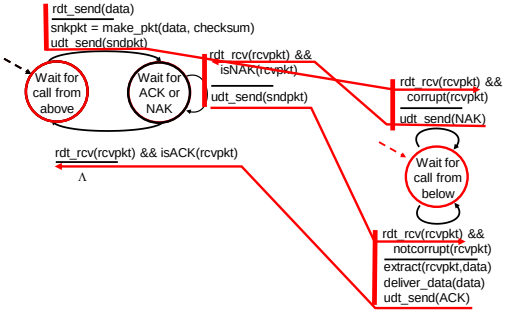
sender

receiver

rdt2.0: quá trình hoạt động không lỗi



rdt2.0: trường hợp có lỗi



rdt2.0 lỗi nghiêm trọng!

Điều gì xảy ra nếu ACK/NAK bị hỏng?

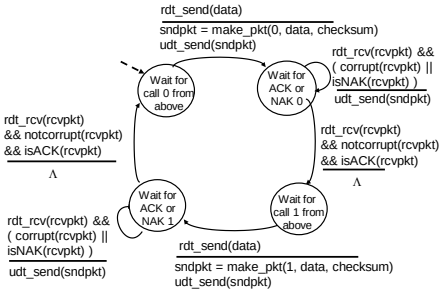
- sender không biết điều gì xảy ra với receiver!
- Không thể tiếp tục truyền lại: có thể truyền thêm bản sao

Điều khiển bản sao:

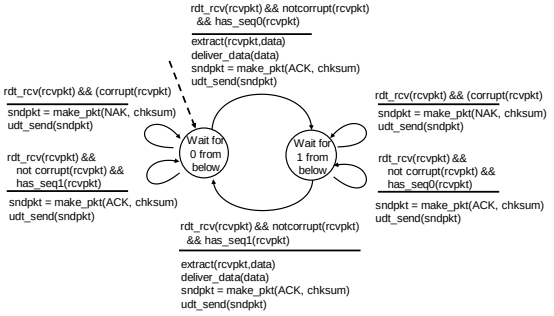
- sender gửi lại gói hiện thời nếu ACK/NAK bị hỏng
- sender thêm *số hiệu chuỗi* vào mỗi gói
- receiver loại bỏ (không chuyển lên) gói bản sao

stop and wait
Sender gửi một packet, Sau đó chờ receiver hồi đáp

rdt2.1: sender, điều khiển dùng ACK/NAKs



rdt2.1: receiver, điều khiển dùng ACK/NAKs



rdt3.0: kênh có lỗi và mất gói tin

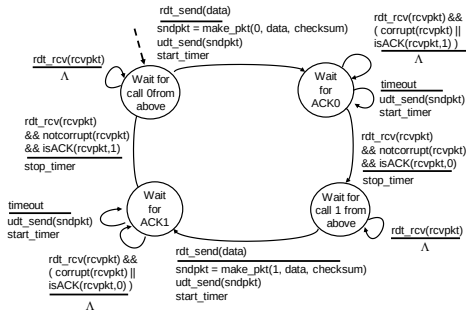
Giả thiết mới: qua kênh truyền có thể mất các gói tin (dữ liệu hoặc ACKs)

- checksum, seq. #, ACKs, giúp xác định việc truyền lại nhưng chưa đủ

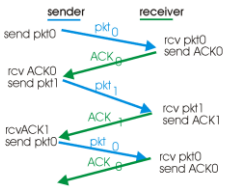
Phương pháp: sender chờ ACK một khoảng thời gian hợp lý

- Sẽ truyền lại nếu không nhận được ACK trong thời gian này
- Nếu pkt (or ACK) chỉ bị trễ (không mất):
  - Truyền lại sẽ bị trùng lặp, nhưng sử dụng seq. # giải quyết được vấn đề này
  - receiver định rõ seq # của pkt đã truyền đúng
- Yêu cầu đếm lùi thời gian

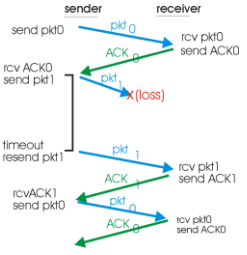
rdt3.0 sender



rdt3.0 hoạt động

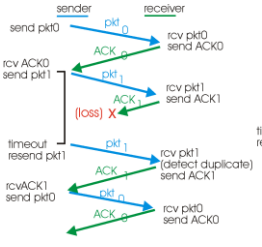


(a) operation with no loss

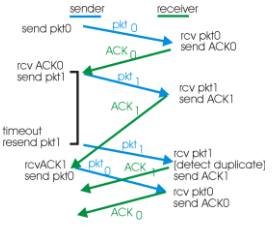


(b) lost packet

rdt3.0 hoạt động



(c) lost ACK



(d) premature timeout

Hiệu năng của rdt3.0

- rdt3.0 có hiệu năng kém
- ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

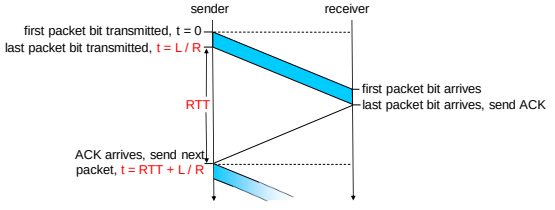
$$d_{trans} = \frac{L}{R} = \frac{8000\,\text{bits}}{10^9\,\text{bps}} = 8\,\text{microseconds}$$

  - U_{sender}: sử dụng – tỷ số của thời gian người gửi bận gửi đi

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

  - 1KB pkt cho mỗi 30 msec -> 33kB/sec thông lượng qua 1 Gbps link
  - Giao thức mạng giới hạn sử dụng tài nguyên vật lý.

rdt3.0: thao tác stop-and-wait

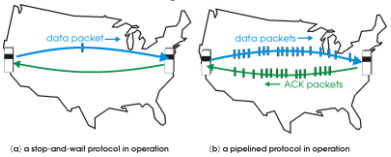


$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

Giao thức dạng Pipeline

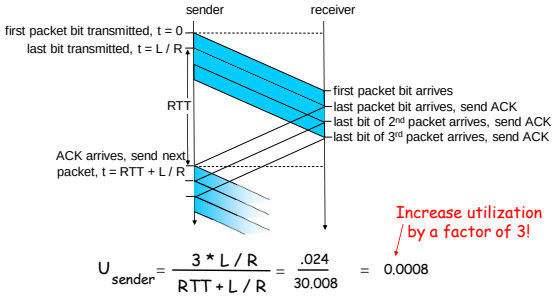
Pipelining: cho phép nhiều gói tin cùng gửi tin trong cùng một chu kỳ RTT.

- Phạm vi của số hiệu chuỗi phải được tăng lên
- Kỹ thuật bộ nhớ đệm bên gửi và/hoặc nhận



- Có 2 dạng giao thức pipeline: *go-Back-N, selective repeat*

Pipelining: tăng khả năng sử dụng



$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Hai dạng giao thức Pipeline

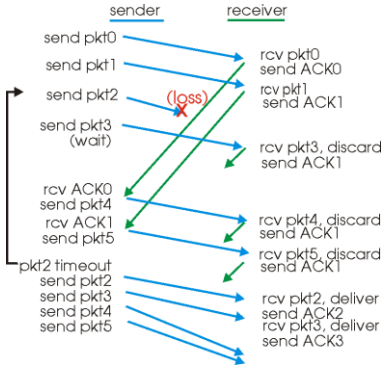
Go-back-N: tổng quan:

- Bên gửi có thể có đến N gói chưa xác thực trong pipeline
- Bên nhận chỉ gửi xác thực tích lũy
  - Chưa xác thực gói nếu vẫn còn khoảng trống
- Bên gửi thiết lập thời gian cho gói tin chưa xác thực lâu nhất
  - Nếu hết thời gian chờ, truyền lại tất cả các gói chưa xác thực

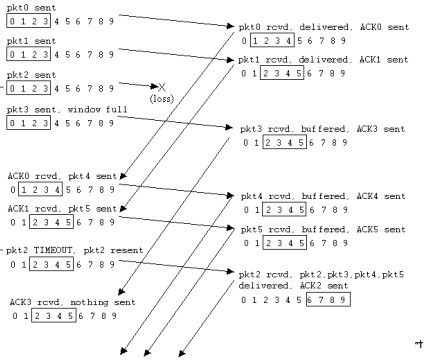
Selective Repeat: tổng quan

- Bên gửi có thể có đến N gói chưa xác thực trong pipeline
- Bên nhận xác thực từng gói riêng biệt
- Bên gửi duy trì thời gian cho mỗi gói chưa xác thực
  - Khi kết thúc thời gian chờ, chỉ truyền lại gói tin chưa xác thực

Hoạt động của GBN



Hoạt động của Selective repeat



Chapter 3 outline

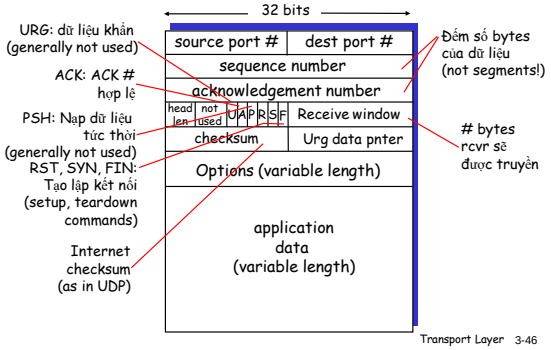
- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP: Tổng quan RFCs: 793, 1122, 1323, 2018, 2581

- Điểm-tới-Điểm:
  - Bên gửi, bên nhận
- Tin cậy, theo chuỗi byte:
  - không "ranh giới bản tin"
- Dạng pipeline:
  - Điều khiển tắc nghẽn và luồng TCP bằng cách thiết lập kích thước cửa sổ
- *send & receive buffers*

- Song công dữ liệu:
  - Luồng dữ liệu hai chiều trong cùng một kết nối
  - MSS: kích thước tối đa của phân đoạn
- Hướng kết nối:
  - Dùng thủ tục bắt tay khởi tạo trạng thái người gửi, người nhận trước khi trao đổi dữ liệu
- Điều khiển luồng:
  - Bên gửi không làm quá tải bên nhận

Cấu trúc phân đoạn TCP



Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP truyền dữ liệu tin cậy

- TCP tạo dịch vụ truyền dữ liệu tin cậy ở mức cao của dịch vụ không tin cậy IP
- Phân đoạn Pipeline
- Tính tích lũy các acks
- TCP sử dụng bộ thiết lập thời gian truyền lại đơn
- Truyền lại được xác định bởi:
  - Các sự kiện timeout
  - Sao chép kép acks
- Khởi tạo ban đầu với TCP đơn giản bên gửi:
  - Bỏ qua sao chép lặp acks
  - Bỏ qua điều khiển luồng, điều khiển tắc nghẽn

TCP các sự kiện bên gửi:

Dữ liệu nhận từ ứng dụng:

- Tạo phân đoạn với số hiệu phân đoạn seq #
- seq # là số hiệu của byte dữ liệu đầu tiên trong phân đoạn
- Khởi động hệ thống tính giờ nếu nó chưa được khởi động
- Thời gian hết hiệu lực: `TimeOutInterval`

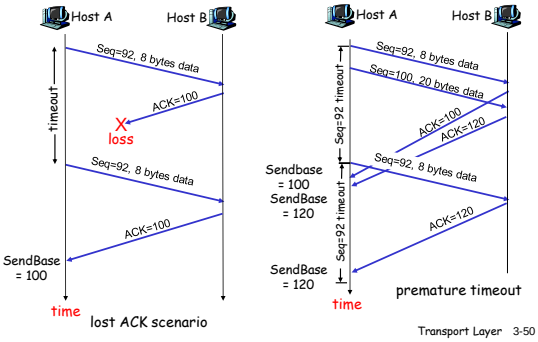
Thời gian chờ:

- Truyền lại phân đoạn quá thời gian chờ
- Khởi tạo lại bộ đếm thời gian

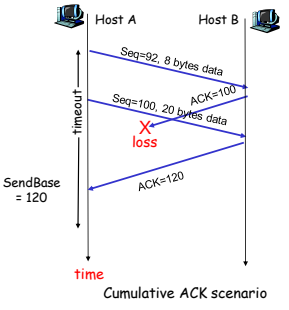
xác thực đã nhận dữ liệu:

- Nếu phân đoạn gửi trước chưa được xác nhận
  - Cập nhật thông tin xác thực đã có
  - Bật bộ đếm thời gian nếu có các phân đoạn quá hạn thời gian

TCP: các tình huống truyền lại



TCP các tình huống truyền lại (more)



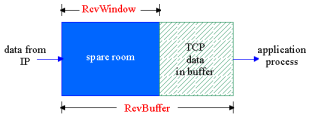
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP điều khiển luồng

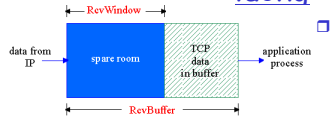
Điều khiển luồng
Người gửi không làm quá tải bộ đệm bên nhận bằng cách gửi quá nhanh, quá nhiều

- Bên nhận của kết nối TCP có bộ đệm nhận:



- Dịch vụ so khớp tốc độ: so khớp tốc độ bên gửi và bên nhận
- Tiến trình ứng dụng có thể đọc từ bộ đệm chậm

TCP hoạt động của điều khiển luồng



(giả sử TCP bên nhận loại bỏ các phân đoạn ngoài phạm vi truyền-nhận)

- Khoảng trống của bộ đệm

```
= RcvWindow
= RcvBuffer-[LastByteRcvd -
  LastByteRead]
```

- Bên nhận thông báo khoảng trống bộ đệm bằng giá trị của **`RcvWindow`** trong phân đoạn.
- Bên gửi giới hạn các phân đoạn để chuyển theo **`RcvWindow`**
  - Đảm bảo bộ đệm bên nhận không bị quá tải

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP quản trị kết nối

Recall: bên gửi và bên nhận tạo lập kết nối trước khi trao đổi các phân đoạn dữ liệu

- Khởi tạo các tham số TCP:
  - seq. #s
  - Bộ đệm, thông tin điều khiển luồng (e.g. `RcvWindow`)
- *client:* tạo lập kết nối

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

- *server:* kết nối bởi client

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

Three way handshake:

Bước 1: trạm client gửi TCP SYN tới server
- Khởi tạo seq #
- Không dữ liệu

Bước 2: server nhận SYN, và trả lời bằng SYNACK
- server phân bổ bộ đệm
- Khởi tạo seq. #

Bước 3: client nhận SYNACK, trả lời bằng ACK, có thể chứa dữ liệu

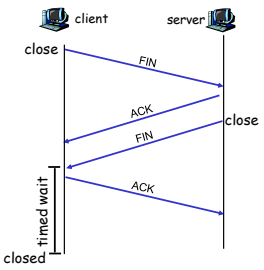
TCP quản trị kết nối (cont.)

Đóng một kết nối:

client closes socket:
`clientSocket.close();`

Bước 1: hệ thống cuối client gửi TCP FIN tới server

Bước 2: server nhận FIN, trả lời bằng ACK. Đóng kết nối, gửi FIN.

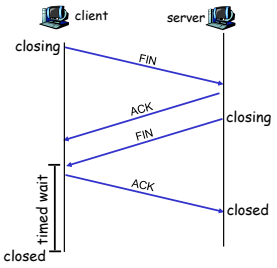


TCP quản trị kết nối (cont.)

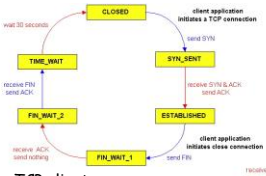
Bước 3: client nhận FIN, trả lời bằng ACK.

- Chuyển sang "thời gian chờ" – sẽ hồi đáp bởi ACK đối với FINs

Bước 4: server, nhận ACK. Đóng kết nối.



TCP quản trị kết nối (cont)



TCP client lifecycle

TCP server lifecycle

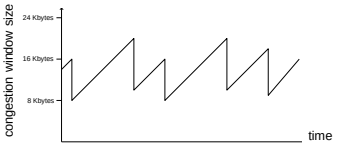
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP điều khiển tắc nghẽn: tăng cộng tính, tăng nhân tính

- *Phương pháp:* tăng tốc độ truyền (window size), thăm dò khả năng sử dụng của bang thông cho đến khi xảy ra mất mát
  - *Tăng cộng tính:* tăng **CongWin** lên 1 MSS cho mỗi RTT cho đến khi phát hiện mất mát
  - *Tăng nhân tính:* chia đôi **CongWin** sau khi lỗi mất dữ liệu

Chế độ răng cưa: dò độ rộng băng thông dùng được



TCP điều khiển tắc nghẽn(cont)

- Giới hạn truyền bên gửi:
  `LastByteSent-LastByteAcked ≤ CongWin`
- Xấp xỉ,

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- **CongWin** thay đổi, là hàm nhận biết độ tắc nghẽn mạng

Cách phát hiện quá tải?

- Sự kiện thất bại = quá thời gian chờ hoặc dư thừa bản sao acks
- TCP sender giảm tỷ lệ (**CongWin**) sau khi phát hiện sự kiện thất bại

Ba kỹ thuật:

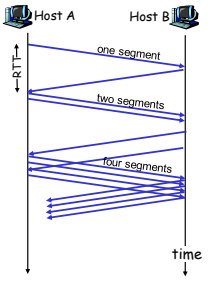
- AIMD
- Khởi động chậm
- Bảo toàn sau sự kiện timeout

TCP khởi động chậm

- Kết nối ban đầu, **CongWin** = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- Băng thông dùng được có thể >> MSS/RTT
  - Mục đích nhanh chóng đạt được tốc độ truyền khá lớn
- Khi kết nối bắt đầu sẽ tăng tốc độ theo hàm mũ cho đến khi sự kiện mất mát dữ liệu xuất hiện

TCP khởi động chậm (more)

- Khi kết nối bắt đầu sẽ tăng tốc độ theo hàm mũ cho đến khi sự kiện mất mát dữ liệu đầu tiên xuất hiện:
  - Gấp đôi **CongWin** cho mỗi RTT
  - tăng **CongWin** cho mỗi ACK nhận được
- Summary: khởi tạo tốc độ chậm nhưng tăng nhanh theo hàm mũ



Lọc: suy diễn dự đoán mất mát

- Sau 3 lần lặp ACKs:
  - **CongWin** được chia đôi
  - Sau đó kích thước window tăng tuyến tính
- Nếu sau sự kiện timeout:
  - **CongWin** thiết lập tới 1 MSS;
  - Kích thước window sau đó tăng theo hàm mũ
  - Tới ngưỡng, sau đó tăng tuyến tính

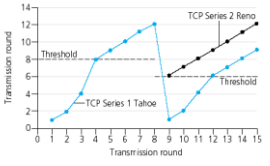
Quan điểm:

- lặp 3 ACKs chỉ ra rằng mạng đang phân phát một số phân đoạn
- timeout cho biết rằng "cảnh báo" về tình huống quá tải

Làm mịn

Q: Lúc nào thì quá trình tang theo hàm mũ sẽ chuyển sang tuyến tính?

A: Khi **CongWin** đạt được 1/2 giá trị của nó trước thời gian timeout.



Thực hiện:

- Thay đổi ngưỡng
- Tại sự kiện mất mátngưỡng thiết lập bằng 1/2 của CongWin trước khi xảy ra lỗi

Summary: TCP điều khiển tắc nghẽn

- Khi **CongWin** ở dưới ngưỡng, bên gửi trong thời kỳ khởi động chậm, window tang theo hàm mũ.
- Khi **CongWin** trên ngưỡng, bên gửi trong thời kỳ chống tắc nghẽn, window tang tuyến tính.
- Khi xảy ra bộ ba lặp ACK, **ngưỡng thiết lập bằng CongWin/2** và **CongWin** thiết lập bằng ngưỡng.
- Khi timeout xảy ra, **Ngưỡng** thiết lập bằng **CongWin/2** và **CongWin** thiết lập bằng 1 MSS.

Chapter 3: Summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation and implementation in the Internet
  - UDP
  - TCP

Next:

- leaving the network "edge" (application, transport layers)
- into the network "core"