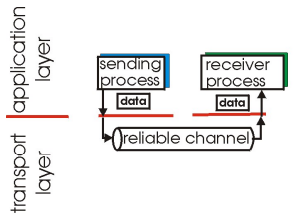
Chương 3: Nội dung

Các nguyên lý của truyền dữ liệu tin cậy

- Quan trọng trong các tầng ứng dụng, giao vận và liên kết
  - Thuộc danh sách 10 vấn đề quan trọng nhất của mạng!

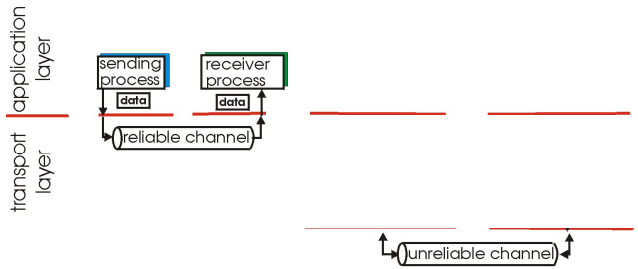


(a) provided service

- Các đặc tính của kênh truyền không tin cậy sẽ xác định sự phức tạp của giao thức truyền dữ liệu tin cậy (reliable data transfer protocol – rdt)

Các nguyên lý của truyền dữ liệu tin cậy

- Quan trọng trong các tầng ứng dụng, giao vận và liên kết
  - Thuộc danh sách 10 vấn đề quan trọng nhất của mạng!

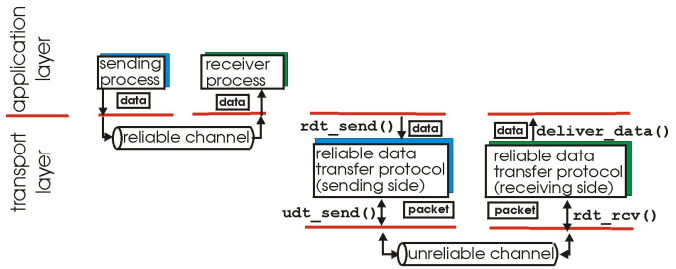


(a) provided service (b) service implementation

- Các đặc tính của kênh truyền không tin cậy sẽ xác định sự phức tạp của giao thức truyền dữ liệu tin cậy (reliable data transfer protocol – rdt)

Các nguyên lý của truyền dữ liệu tin cậy

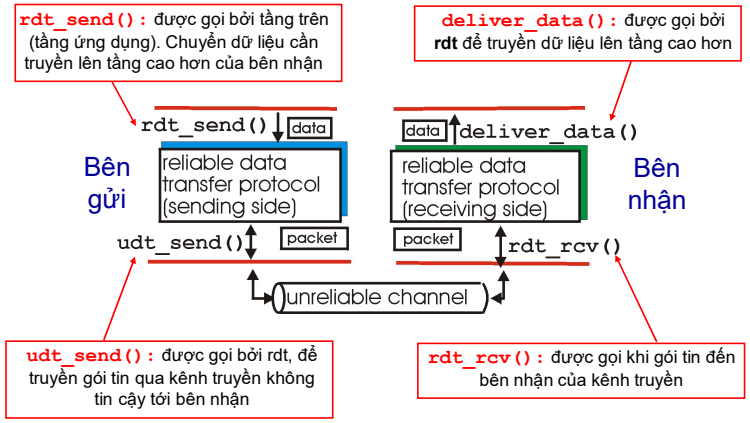
- Quan trọng trong các tầng ứng dụng, giao vận và liên kết
  - Thuộc danh sách 10 vấn đề quan trọng nhất của mạng!



(a) provided service (b) service implementation

- Các đặc tính của kênh truyền không tin cậy sẽ xác định sự phức tạp của giao thức truyền dữ liệu tin cậy (reliable data transfer protocol – rdt)

Truyền dữ liệu tin cậy

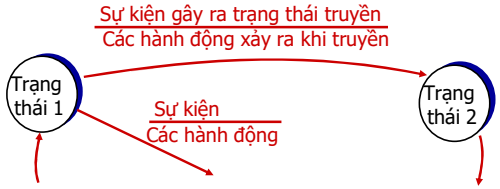


Truyền dữ liệu tin cậy

Việc cần làm:

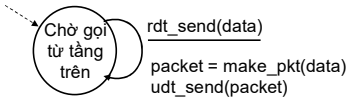
- Phát triển dần giao thức truyền dữ liệu tin cậy (reliable data transfer protocol - rdt) cho cả bên gửi và bên nhận
- Chỉ xem xét truyền dữ liệu theo một hướng
  - Nhưng thông tin điều khiển vẫn được truyền theo cả hai hướng
- Dùng máy trạng thái hữu hạn (finite state machines - FSM) để xác định bên gửi, bên nhận

Trạng thái: khi đang ở một "trạng thái" thì trạng thái duy nhất kế tiếp sẽ được xác định khi có sự kiện tiếp theo

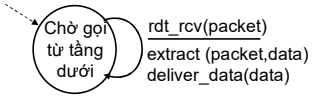


rdt1.0: truyền dữ liệu tin cậy qua một kênh truyền tin cậy

- Kênh truyền cơ bản hoàn toàn tin cậy
  - Không có lỗi bit
  - Không có mất mát gói tin
- Phân biệt các FSM cho bên gửi, bên nhận:
  - Bên gửi gửi dữ liệu vào kênh truyền cơ bản
  - Bên nhận đọc dữ liệu từ kênh truyền cơ bản



Bên gửi



Bên nhận

rdt2.0: Kênh truyền có lỗi bit

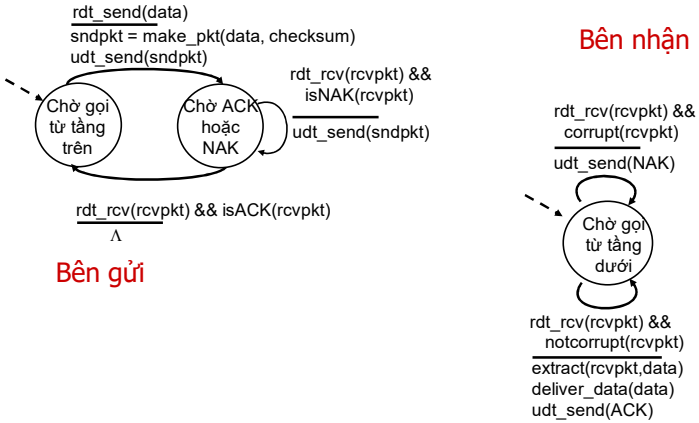
- Kênh cơ bản có thể bật một vài bit trong gói tin
  - Kiểm tra (checksum) để phát hiện các lỗi bit
- Câu hỏi: Làm thế nào để khôi phục lại các lỗi?

Làm thế nào con người khôi phục được "lỗi" trong suốt quá trình thực hiện cuộc hội thoại?

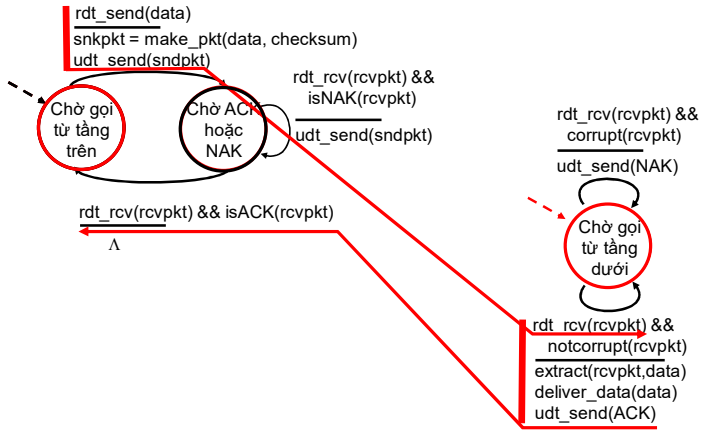
rdt2.0: Kênh truyền có lỗi bit

- Kênh truyền cơ bản có thể bật một vài bit trong gói tin
  - Kiểm tra (checksum) để phát hiện các lỗi bit
- Câu hỏi: Làm thế nào để khôi phục lại các lỗi?
  - *Báo nhận ACK (acknowledgement):* bên nhận thông báo rõ cho cho bên gửi là gói tin nhận được tốt
  - *Báo nhận NAK (negative acknowledgement):* bên nhận thông báo rõ cho bên gửi là gói tin nhận được có lỗi
  - Bên gửi truyền lại gói tin có báo nhận là NAK
- Các cơ chế mới trong rdt2.0 (ngoài rdt1.0):
  - Phát hiện lỗi
  - Phản hồi: các thông điệp điều khiển (ACK,NAK) từ bên nhận gửi về bên gửi

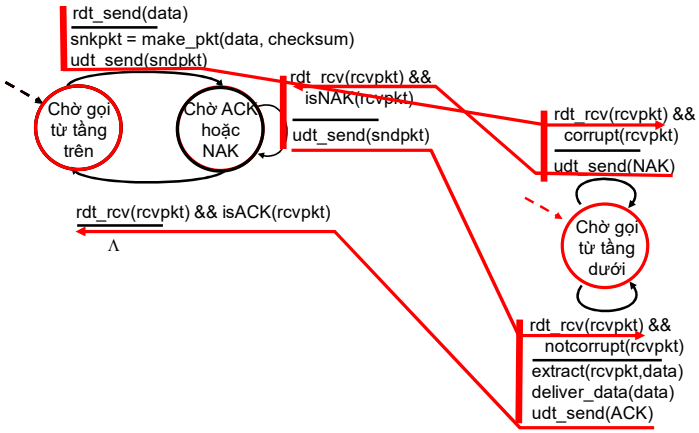
rdt2.0: Đặc tả FSM



rdt2.0: Hoạt động khi không có lỗi



rdt2.0: Kịch bản khi có lỗi



rdt2.0 có lỗ hổng nghiêm trọng!

Điều gì xảy ra khi ACK/NAK bị hỏng?

- Bên gửi không biết được điều gì đã xảy ra tại bên nhận!
- Không thể đơn phương truyền lại: có thể bị trùng lặp

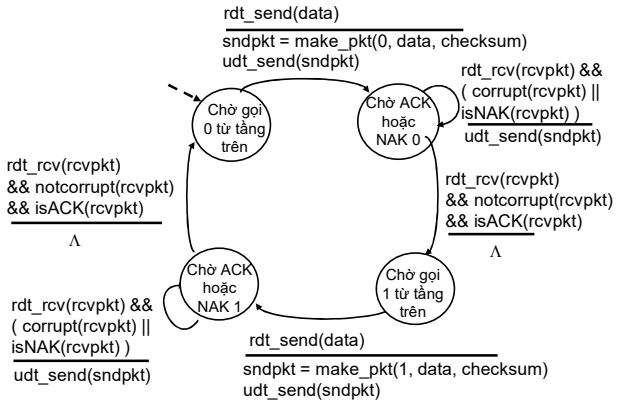
Xử lý trùng lặp:

- Bên gửi truyền lại gói tin hiện tại nếu ACK/NAK bị hỏng
- Bên gửi thêm *số thứ tự* vào trong mỗi gói tin
- Bên nhận bỏ qua (không nhận) gói bị trùng lặp

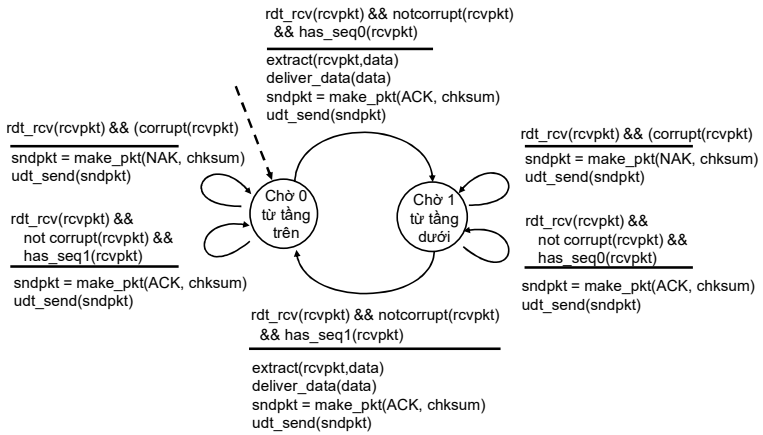
Dừng và chờ

Bên gửi gửi một gói tin, sau đó dừng lại chờ bên nhận phản hồi

rdt2.1: Bên gửi xử lý các ACK/NAK bị hỏng



rdt2.1: Bên nhận xử lý các ACK/NAK bị hỏng



rdt2.1: Thảo luận

Bên gửi:

- Số thứ tự được bổ sung vào gói tin
- Chỉ cần hai số thứ tự (0,1) là đủ. Vì sao?
- Phải kiểm tra lại nếu việc nhận ACK/NAK bị hỏng
- Số trạng thái tăng lên 2 lần
  - Trạng thái phải "nhớ" xem gói tin đang "dự kiến" đến sẽ có số thứ tự là 0 hay 1

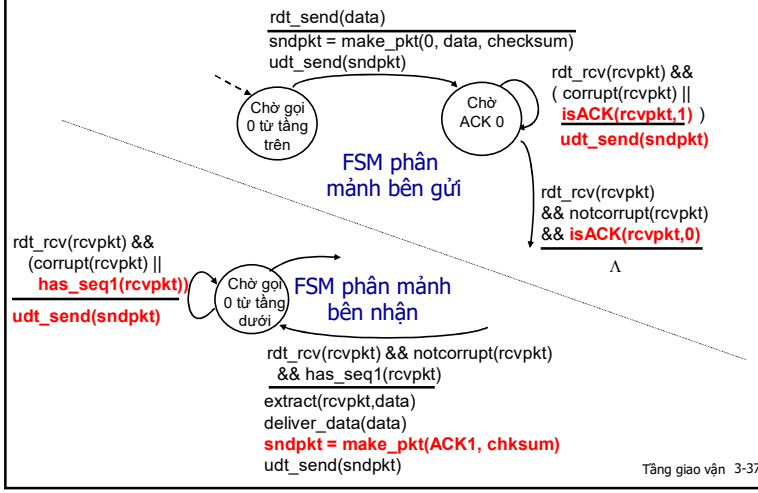
Bên nhận:

- Phải kiểm tra xem gói tin nhận được có bị trùng lặp hay không
  - Trạng thái chỉ rõ gói tin đang chờ đến có số thứ tự là 0 hay 1
- Chú ý: bên nhận *không thể* biết được ACK/NAK cuối cùng gửi đi có được nhận tốt hay không tại bên gửi

rdt2.2: Một giao thức không cần NAK

- Chức năng giống như trong rdt2.1, nhưng chỉ dùng báo nhận ACK
- Thay vì sử dụng NAK, bên nhận sẽ gửi ACK cho gói tin cuối cùng nhận tốt
  - Bên nhận phải thêm số thứ tự của gói tin đang được báo nhận
- ACK bị trùng lặp tại bên gửi sẽ dẫn đến cùng hành động như NAK: *truyền lại gói tin hiện tại*

rdt2.2: Phân mảnh tại bên gửi, bên nhận

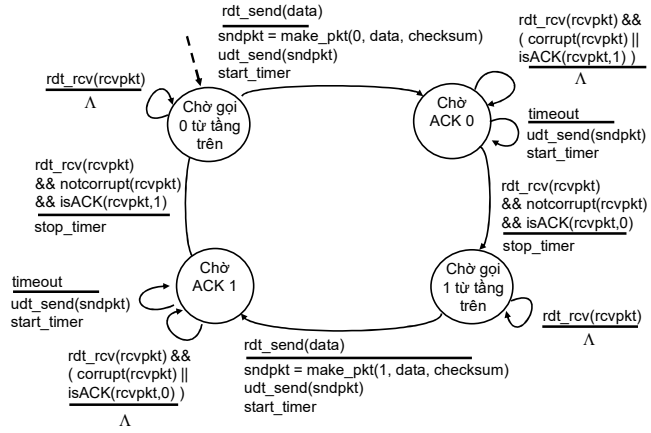


rdt3.0: Kênh truyền có lỗi và mất mát

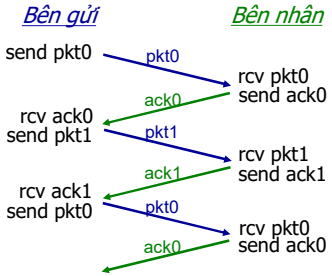
Giả thiết mới: Kênh cơ bản cũng có thể làm mất các gói tin (dữ liệu, ACK)
- checksum, số thứ tự, báo nhận ACK, truyền lại sẽ hỗ trợ… nhưng chưa đủ

Tiếp cận: Bên gửi chờ ACK trong khoảng thời gian "chấp nhận được"
- Truyền lại nếu không nhận được ACK trong khoảng thời gian này
- Nếu gói tin (hoặc ACK) chỉ đến trễ (chứ không bị mất):
  - Việc truyền lại sẽ gây trùng lặp, nhưng số thứ tự sẽ xử lý việc này
  - Bên nhận phải chỉ rõ số thứ tự của gói tin đang được báo nhận
- Cần bộ định thời đếm ngược

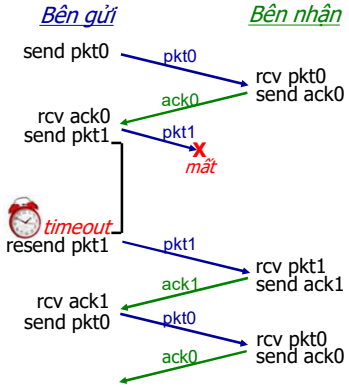
rdt3.0 bên gửi



Hoạt động của rdt3.0

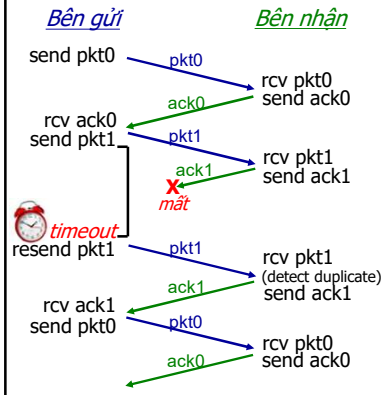


(a) Không mất mát

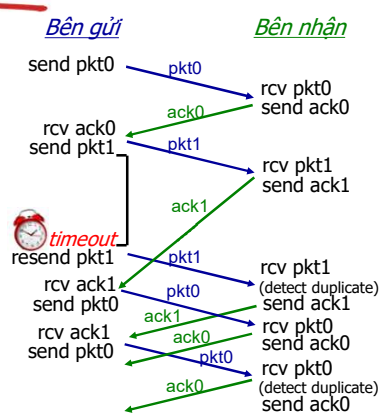


(b) Mất gói tin

Hoạt động của rdt3.0



(c) Mất ACK



(d) Timeout sớm/ ACK bị trễ

Hiệu suất của rdt3.0

- rdt3.0 hoạt động tốt, nhưng không hiệu quả
- Ví dụ: Liên kết 1 Gbps, trễ lan truyền 15 ms, gói tin 8000 bit:

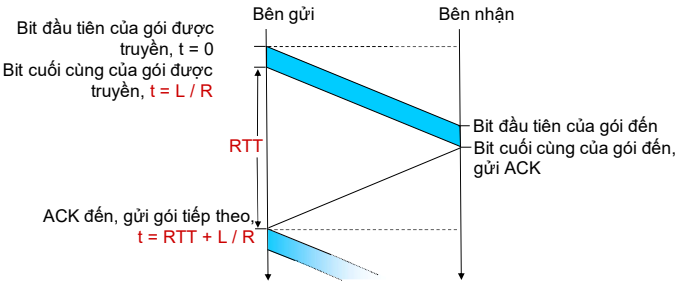
$$D_{trans} = \frac{L}{R} = \frac{8000\ bit}{10^9\ bit/sec} = 8\ microsecs$$

  - U sender: độ khả dụng – tỷ lệ về mặt thời gian bên gửi liên tục phải gửi

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

  - Nếu RTT=30 msec, gói tin 1KB được truyền sau mỗi 30 msec: thông lượng trên liên kết 1 Gbps là 33kB/sec
- Giao thức mạng giới hạn việc sử dụng các tài nguyên vật lý!

rdt3.0: Hoạt động dừng-và-chờ

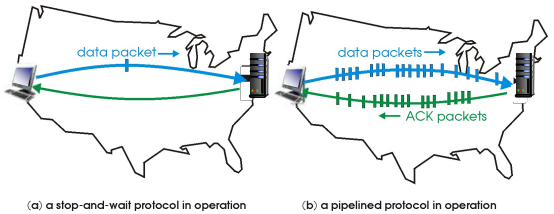


$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

Các giao thức Pipeline

Pipelining: bên gửi cho phép gửi nhiều gói "đồng thời", mà không cần chờ gói báo nhận

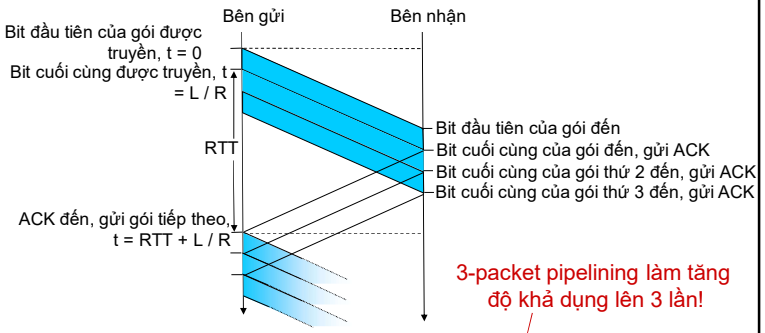
- Dãy các số thứ tự sẽ được tăng dần
- Cần có bộ đệm tại bên gửi và/hoặc bên nhận



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

- Hai dạng thức chung của các giao thức pipeline : *go-Back-N, lặp có lựa chọn (selective repeat)*

Pipelining: tăng độ khả dụng



3-packet pipelining làm tăng độ khả dụng lên 3 lần!

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Các giao thức pipeline

Go-back-N:

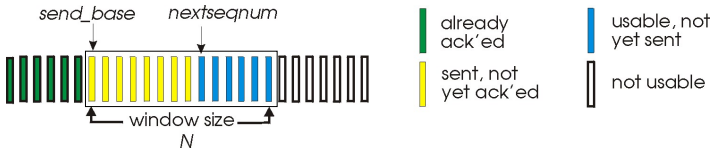
- Bên gửi có thể có đến N gói chưa được báo nhận trong pipeline
- Bên nhận chỉ gửi *ack tích lũy*
  - Không báo nhận cho gói tin cho đến khi có một khoảng trống
- Bên gửi có bộ định thời cho các gói tin gửi đi mà chưa được báo nhận
  - Khi bộ định thời hết hạn, truyền lại tất cả các gói tin chưa được báo nhận

Lặp có lựa chọn:

- Bên gửi có thể có đến N gói chưa được báo nhận trong pipeline
- Bên nhận gửi *ack riêng* cho mỗi gói tin
- Bên gửi duy trì bộ định thời cho mỗi gói tin chưa được báo nhận
  - Khi bộ định thời hết hạn, chỉ truyền lại gói tin chưa được báo nhận

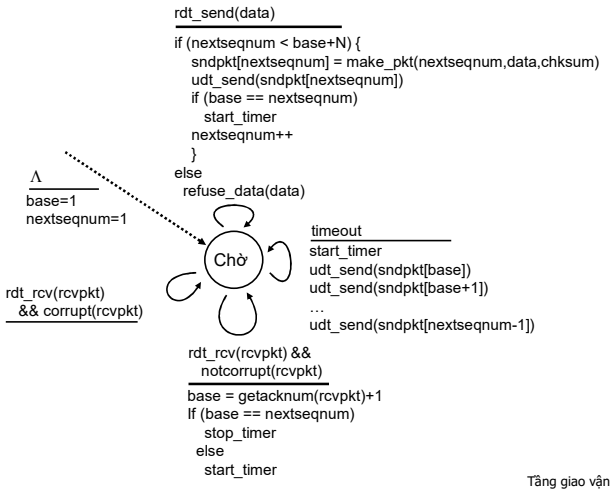
Go-Back-N: bên gửi

- k-bit số thứ tự trong phần tiêu đề của gói tin
- "Cửa sổ" tăng lên đến N, cho phép gửi gói liên tục không cần báo nhận

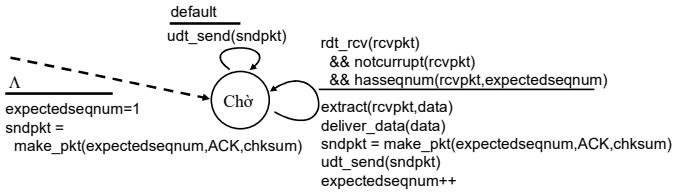


- ACK(n): báo nhận ACK cho tất cả các gói đến, chứa số thứ tự n- *"ACK tích lũy"*
  - Có thể nhận được ACK trùng lặp (xem bên nhận)
- Đặt bộ định thời cho các gói tin truyền đi
- *timeout(n):* truyền lại gói n và tất cả các gói có số thứ tự lớn hơn trong cửa sổ

GBN: FSM mở rộng tại bên gửi



GBN: FSM mở rộng tại bên nhận



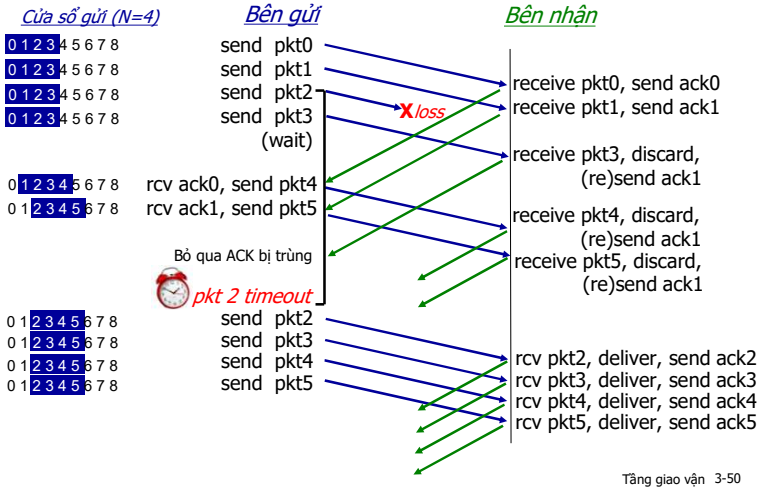
ACK-duy nhất: luôn gửi ACK cho gói đã nhận đúng với số thứ tự xếp hạng cao nhất

- Có thể sinh ra ACK trùng nhau
- Chỉ cần nhớ số thứ tự của gói dự kiến đến (**expectedseqnum**)

❖ Gói không theo đúng thứ tự:

- Hủy: *không nhận vào vùng đệm!*
- Gửi lại ACK với số thứ tự (xếp hạng) cao nhất

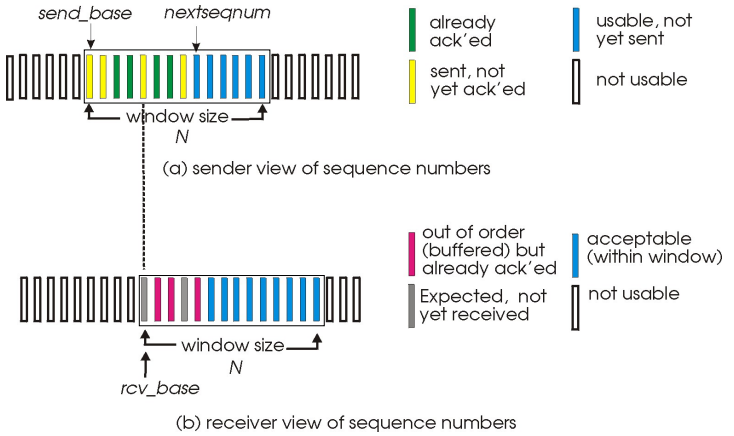
Hoạt động của GBN



Lặp có lựa chọn

- Bên nhận báo nhận *riêng* cho tất cả các gói tin đã nhận đúng.
  - Đặt các gói vào bộ đệm (nếu cần), cho đúng thứ tự để chuyển lên tầng cao hơn
- Bên gửi chỉ gửi lại các gói tin nào mà không nhận được ACK
  - Có bộ định thời bên gửi cho mỗi gói tin không gửi ACK
- Cửa sổ bên gửi
  - *N* số thứ tự liên tục
  - Hạn chế số thứ tự các gói không gửi ACK

Lặp có lựa chọn: cửa sổ bên gửi, bên nhận



Lặp có lựa chọn

Bên gửi

Dữ liệu từ tầng trên:
- Nếu số thứ tự kế tiếp sẵn sàng trong cửa sổ, thì gửi gói tin

timeout(n):
- Gửi lại gói n, khởi tạo lại bộ định thời

ACK(n) trong [sendbase,sendbase+N]:
- Đánh dấu gói n là đã nhận
- Nếu gói có số thứ tự n thấp nhất mà chưa được ACK, thì dịch chuyển cửa sổ cơ sở đến số thứ tự kế tiếp chưa được ACK.

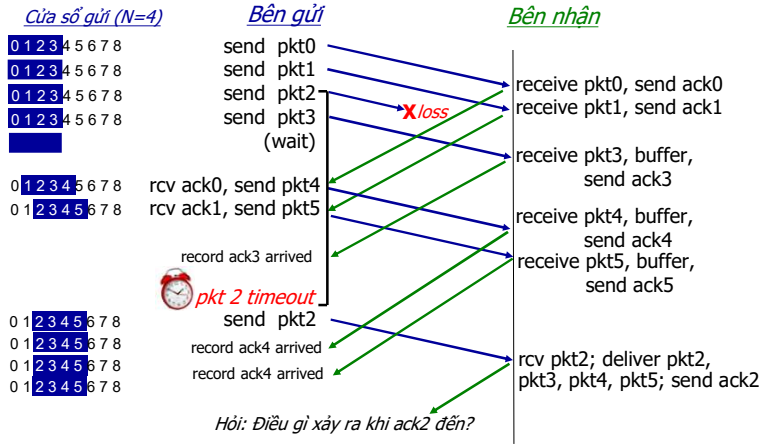
Bên nhận

Gói n trong [rcvbase, rcvbase+N-1]
- Gửi ACK(n)
- Không đúng thứ tự: đệm
- Đúng thứ tự: truyền (cũng truyền các gói đã đệm, đúng thứ tự), dịch chuyển cửa sổ đến gói chưa nhận được kế tiếp

Gói n trong [rcvbase-N,rcvbase-1]
- ACK(n)

Ngược lại:
- Bỏ qua

Hoạt động trong lặp có lựa chọn



Lặp có lựa chọn: tình trạng khó giải quyết

Ví dụ:
- Các số thứ tự: 0, 1, 2, 3
- Kích thước cửa sổ =3
- Bên nhận không nhận ra sự khác biệt giữa 2 kịch bản!
- Chấp nhận dữ liệu bị trùng lặp như là dữ liệu mới trong (b)

Hỏi: Quan hệ giữa kích thước số thứ tự và kích thước cửa sổ như thế nào để tránh vấn đề như trong (b)?