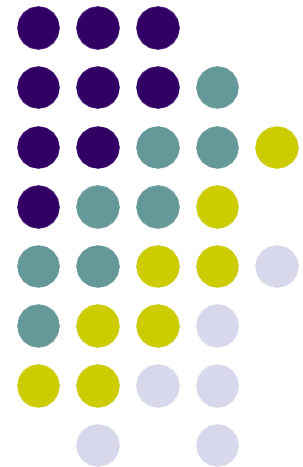


Chương 4: Liên kết dữ liệu - Data Link

Giảng viên: Nguyễn Đức Toàn

Bộ môn Truyền thông và Mạng máy tính
Viện CNTT&TT - ĐHBK Hà Nội

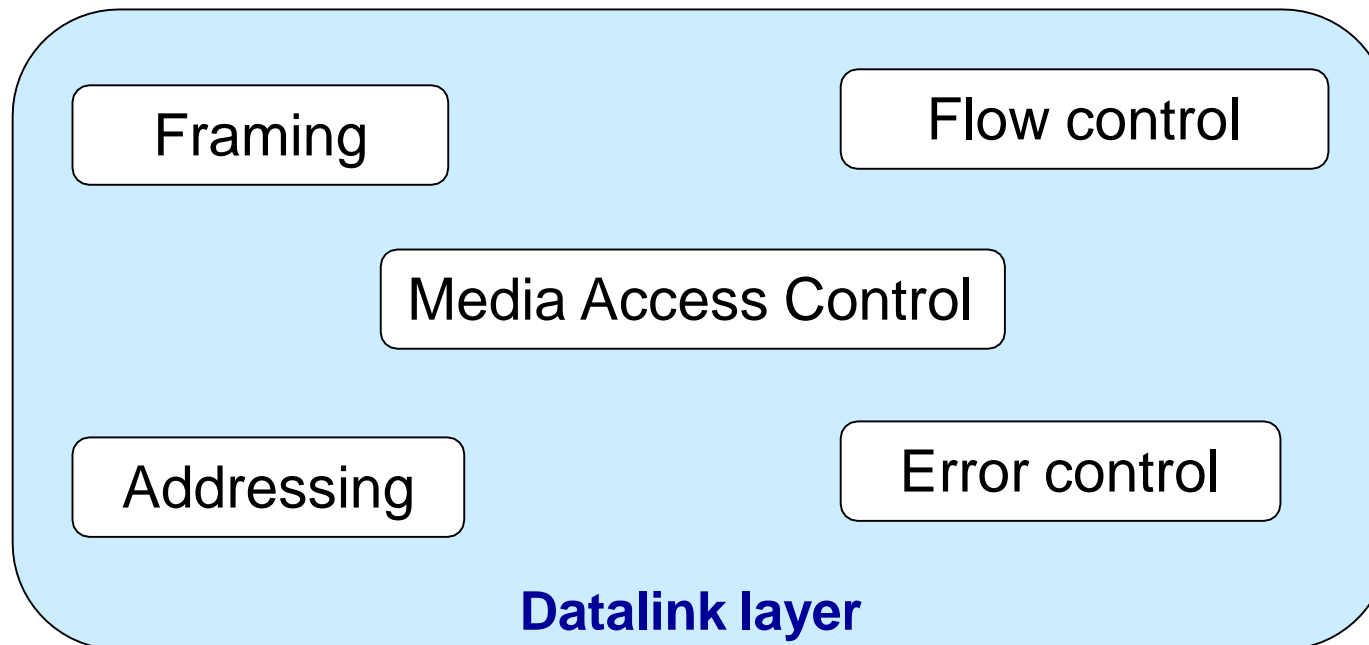




Tổng quan

- Tuần trước
 - Tầng vật lý
 - Các loại đường truyền
 - Mã đường truyền
- Tuần này: Tầng liên kết dữ liệu
 - Đóng gói khung
 - Kiểm soát lỗi
 - Kiểm soát luồng

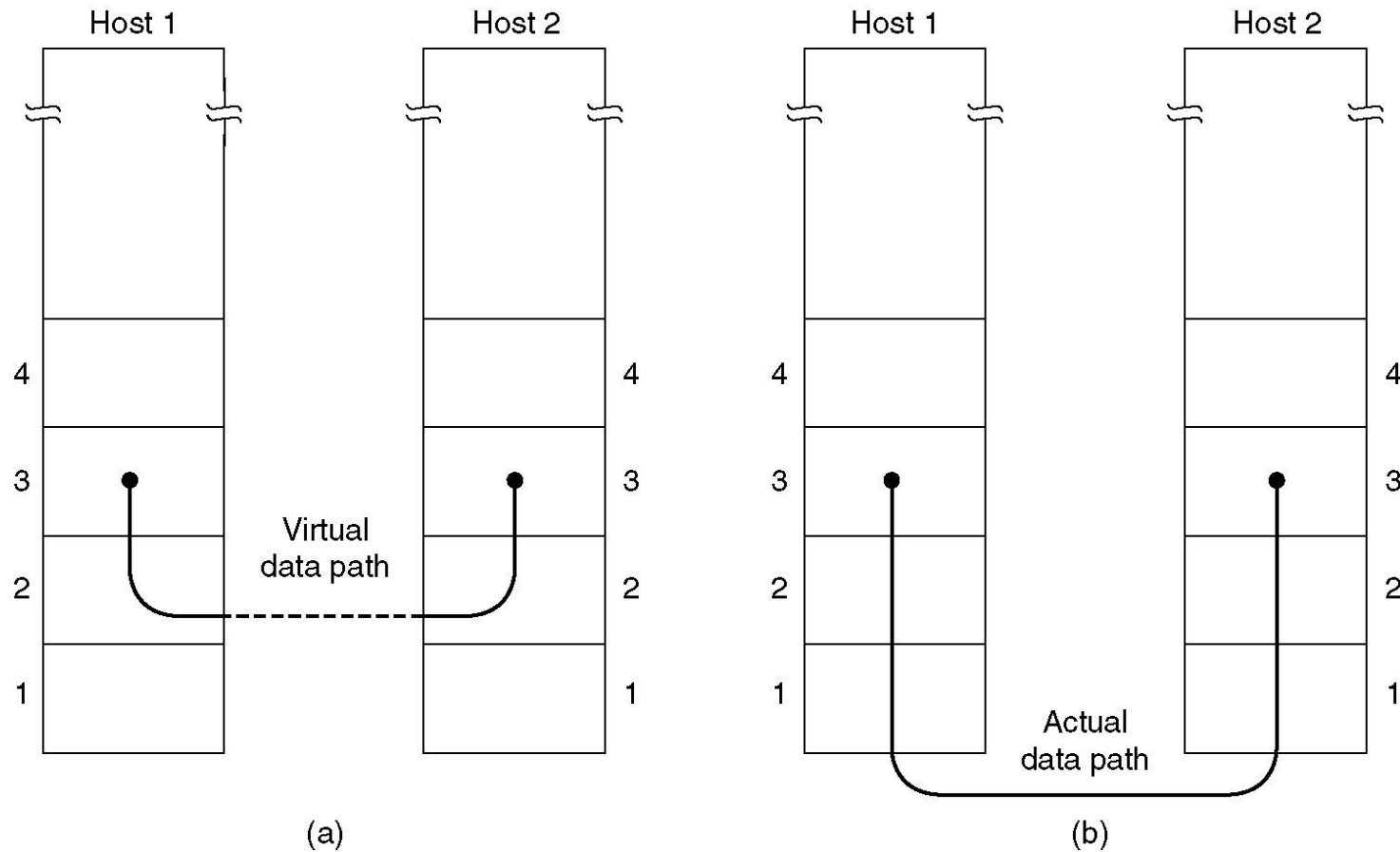
Tổng quan về các chức năng



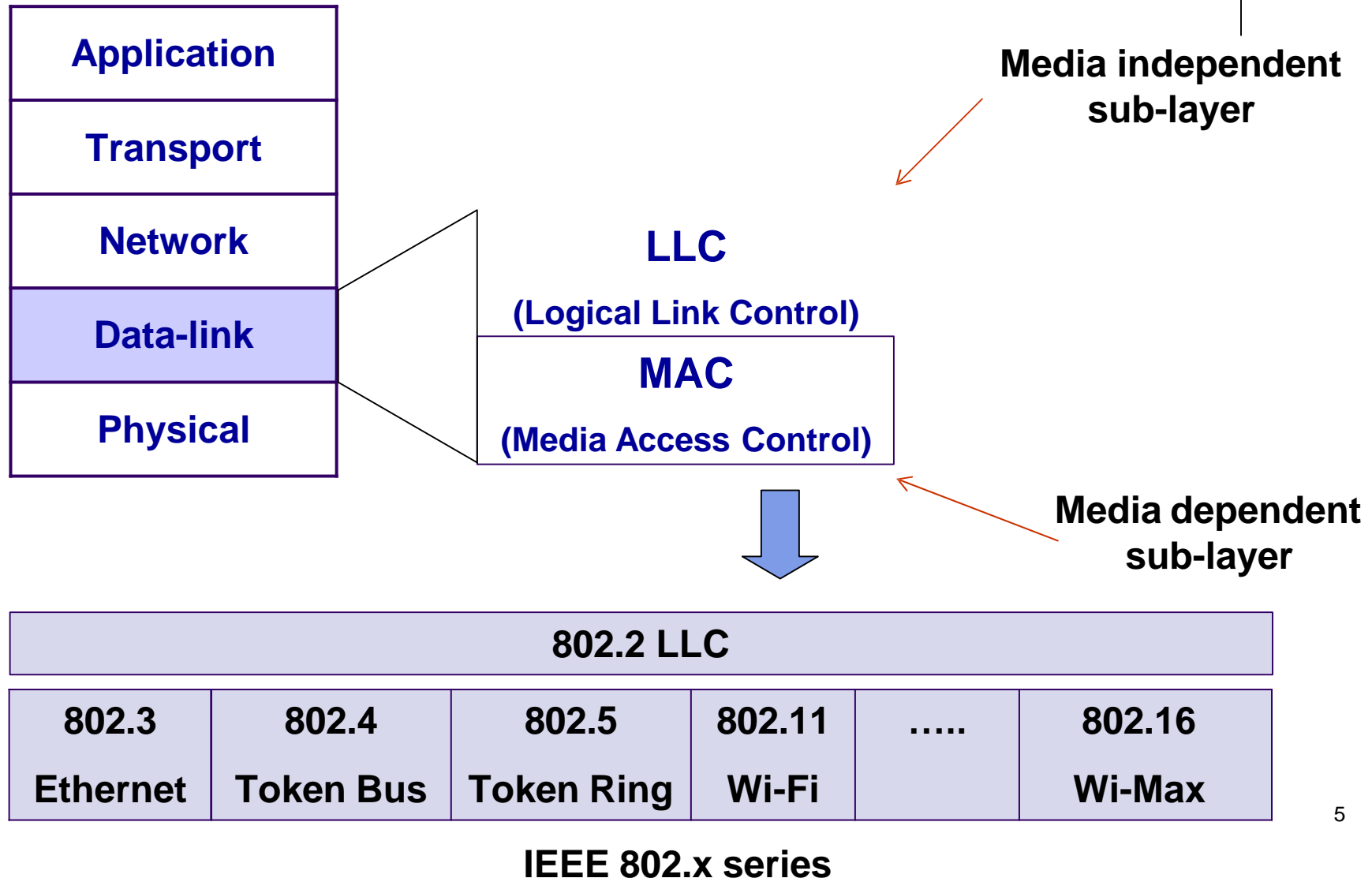
Chức năng của tầng Data Link

- Cung cấp giao diện dịch vụ cho tầng mạng
- Xác định cách nhóm các bit thành khung và truyền trên đường truyền vật lý.
- Xử lý lỗi khi truyền
- Kiểm soát luồng dữ liệu
 - *Máy có tốc độ chậm không bị quá tải khi giao tiếp với máy tốc độ cao*

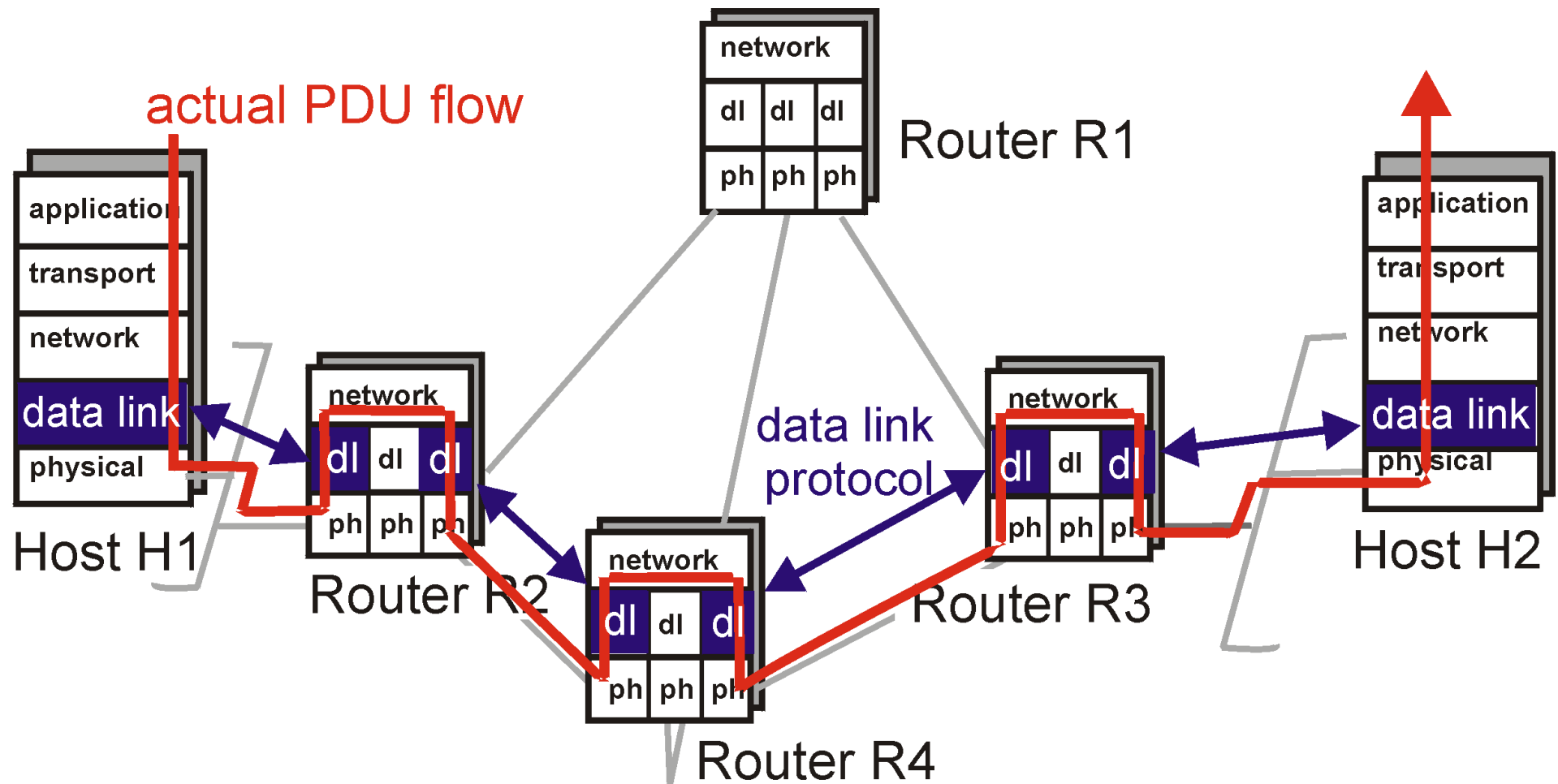
Tầng liên kết dữ liệu trong mô hình



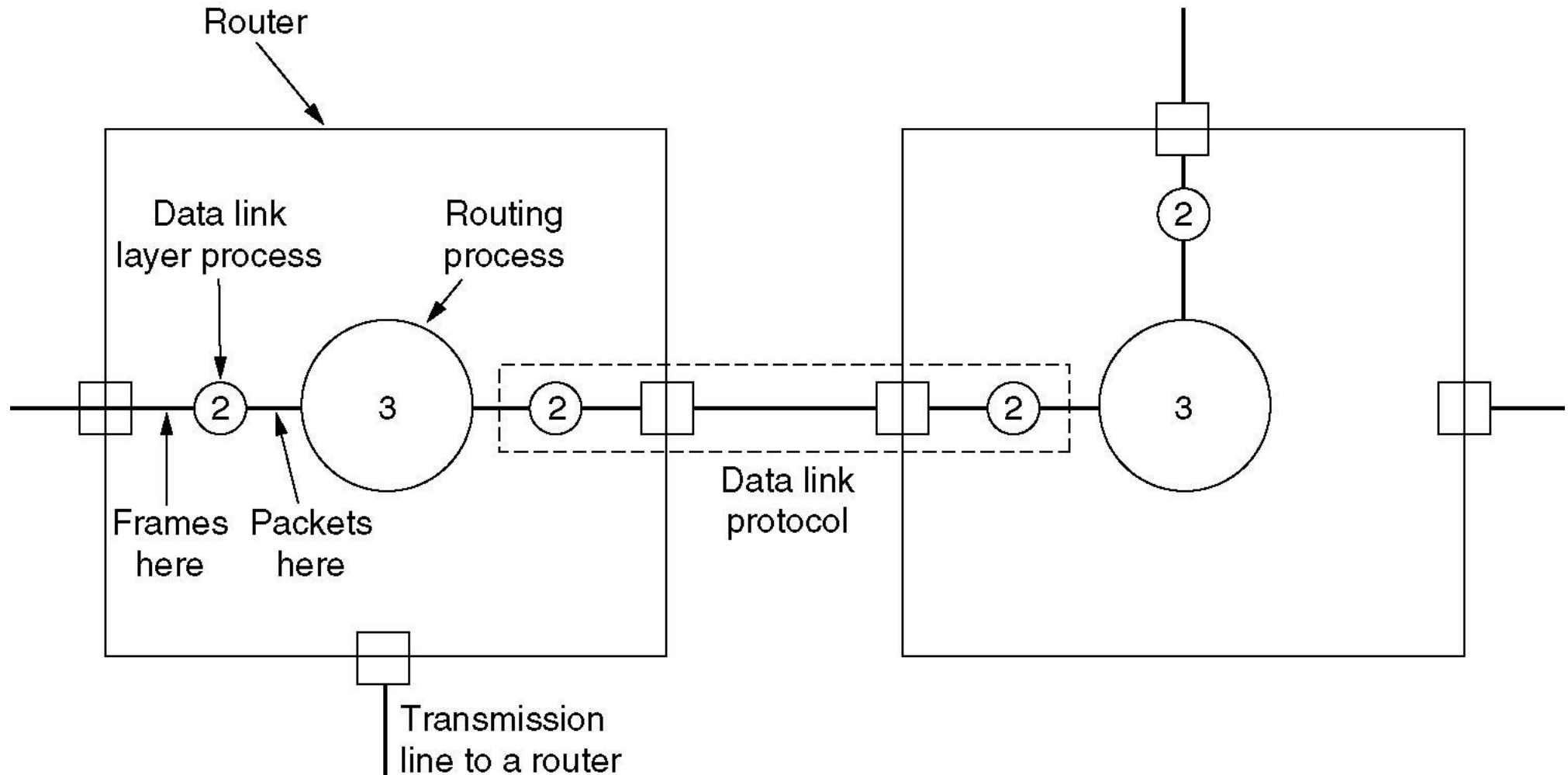
Tầng liên kết dữ liệu và kiến trúc phân tầng



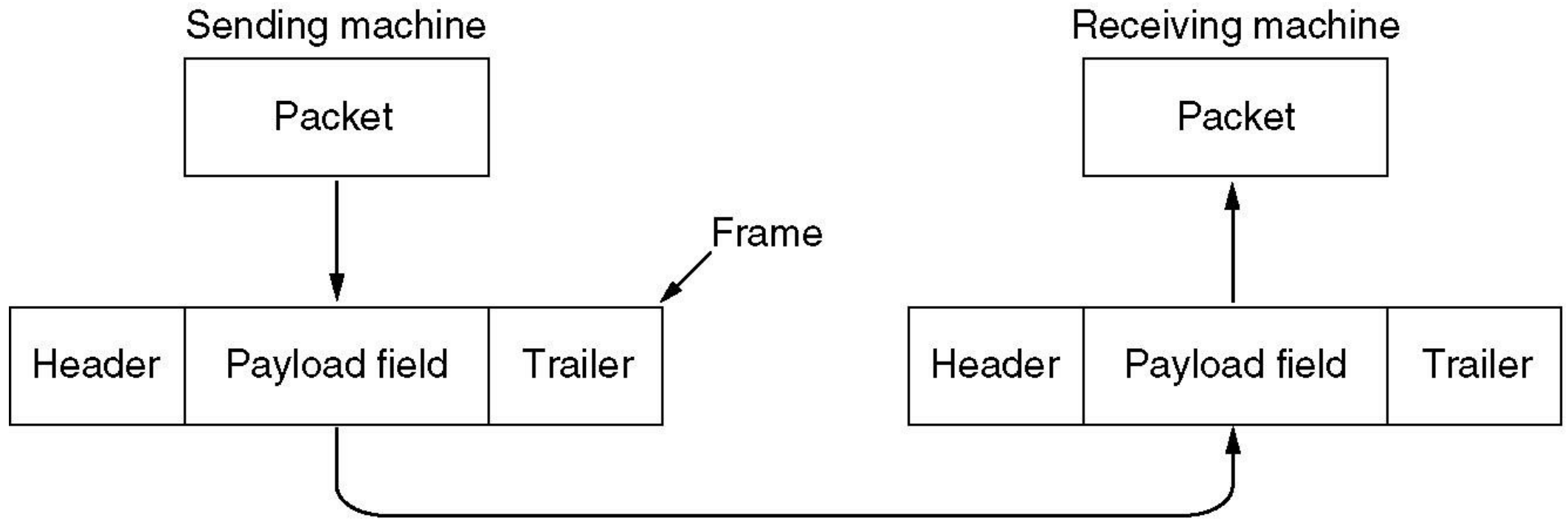
Tầng liên kết dữ liệu ở đâu trong mạng?



Tầng liên kết dữ liệu ở đâu trong Router?



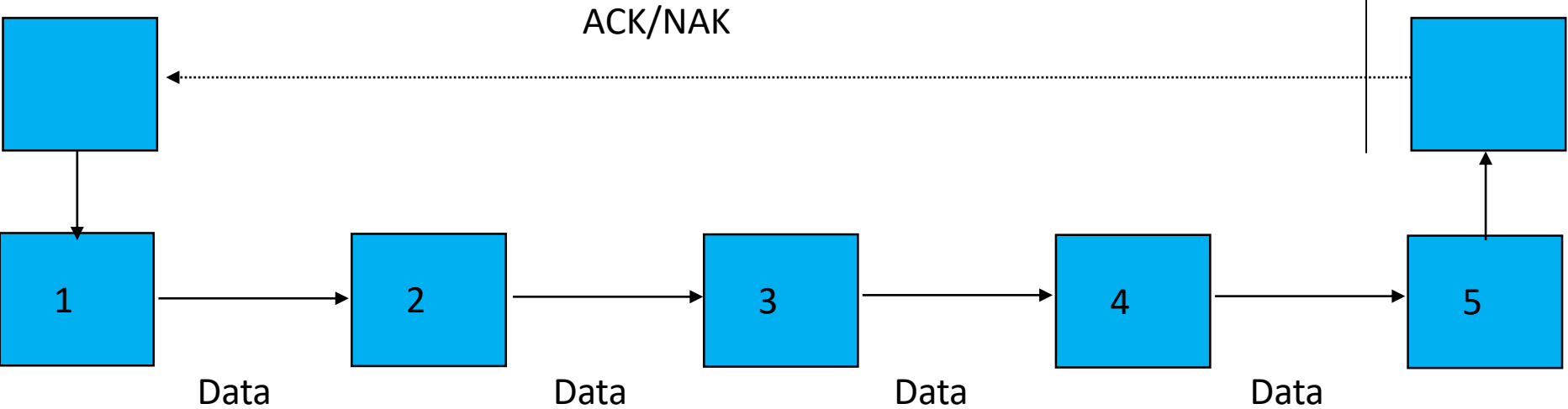
Gói tin (Packet) và khung (Frame)



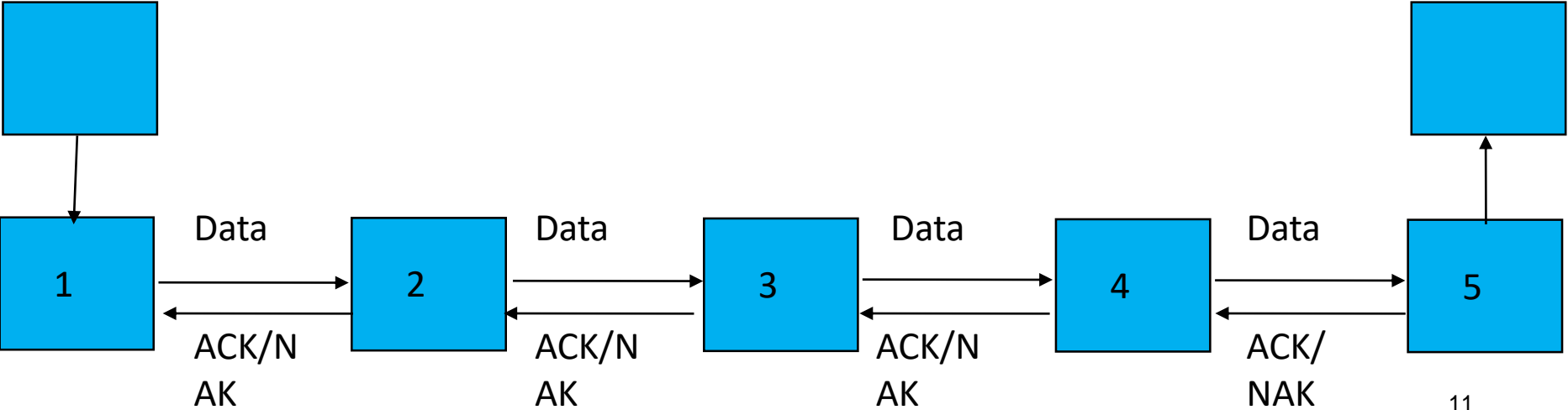
Dịch vụ cung cấp cho tầng mạng

- Dịch vụ không hướng liên kết, không đảm bảo
- Dịch vụ không hướng liên kết, có đảm bảo
- Dịch vụ hướng liên kết, có đảm bảo

End to End



Hop by Hop



Dịch vụ không hướng liên kết, không đảm bảo

- Việc xử lý mất mát dữ liệu được thực hiện ở các tầng cao hơn
- Thường được dung trên các phương tiện truyền dẫn có tỉ lệ lỗi thấp: cáp đồng trục, cáp quang,...
- Thích hợp với truyền thông thoại, tốc độ truyền dẫn cần nhanh (độ trễ nhỏ)

Dịch vụ không hướng liên kết, có đảm bảo

- Thích hợp với các đường truyền có độ tin cậy thấp như truyền dẫn không dây.
- Việc đảm bảo (Acknowledgements) sẽ tăng độ trễ.
- Việc thêm mức đảm bảo ở tầng này chỉ với mục đích tối ưu chứ không bắt buộc.
- Nếu đề tầng Mạng xử lý việc mất/ truyền lại gói tin (Packet) → Các gói tin kích thước lớn sẽ được gửi lại. Trong khi ở tầng này, các khung (Frame) có kích thước nhỏ hơn.

Chú ý: ở các đường truyền có độ tin cậy cao như cáp quang, sự khác biệt trên là không đáng kể.

Dịch vụ hướng liên kết, có đảm bảo

- Có độ tin cậy cao nhất
- Các dịch vụ đảm bảo
 - Từng khung được gửi đi sẽ được nhận
 - Mỗi khung được nhận 1 lần
 - Các khung được nhận có thứ tự



Các chức năng (1)

- Đóng gói - Framing:
 - Đơn vị dữ liệu: Frame (khung tin)
 - Bên gửi: đặt gói tin tầng mạng vào khung tin, thêm phần đầu, phần đuôi
 - Bên nhận: Bỏ phần đầu, phần đuôi và lấy gói tin truyền lên tầng mạng
- Địa chỉ hóa - Addressing:
 - Địa chỉ vật lý đặt trong phần đầu gói tin để định danh nút nguồn, nút đích

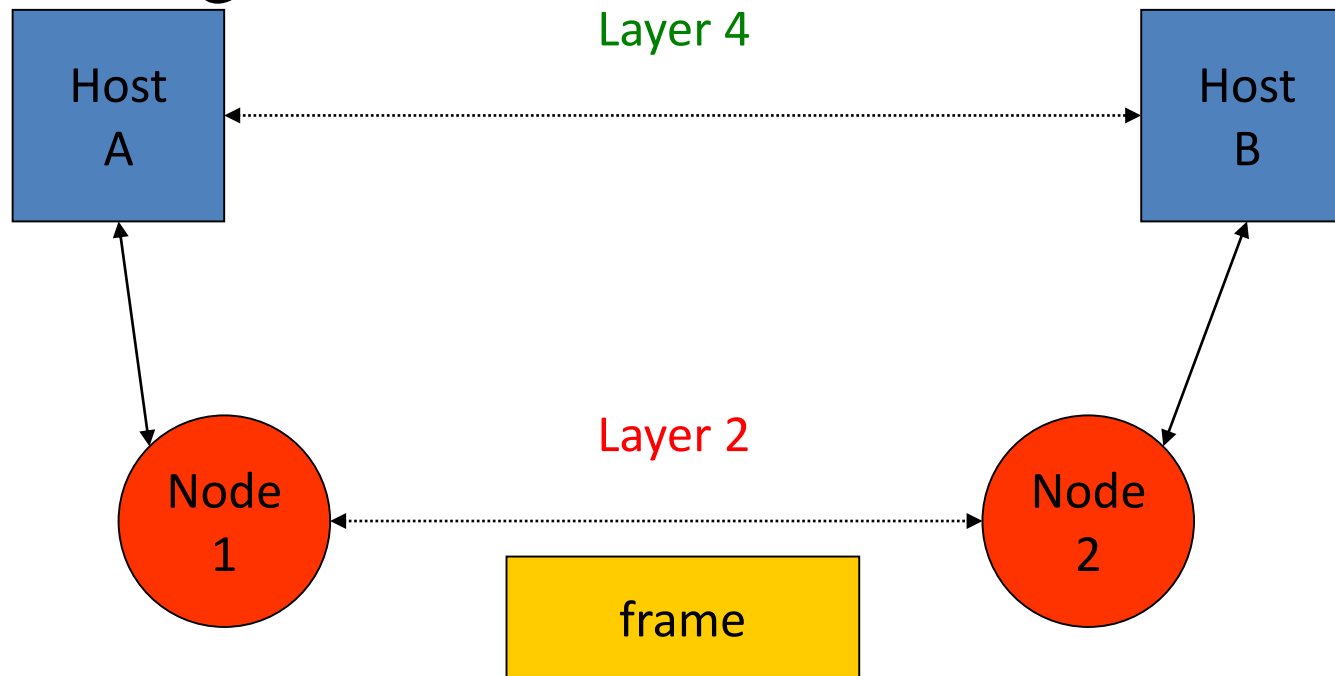


Các chức năng (2)

- Điều khiển truy nhập đường truyền
 - Nếu là mạng đa truy nhập, cần có các giao thức truy nhập đường truyền cho nhiều máy trạm
- Kiểm soát luồng:
 - Kiểm soát tốc độ truyền của bên gửi sao cho bên nhận hoạt động tốt, không bị quá tải
- Kiểm soát lỗi:
 - Phát hiện và sửa các lỗi bit
 - e.g. parity check, checksum, CRC check

Các kỹ thuật lập Khung

- Xác định khung bằng Đếm ký tự
- Xác định khung bằng các byte cờ có kiểm tra byte (byte stuffing)
- Xác định khung bằng các byte cờ có kiểm tra bit (bit stuffing)



Xác định khung

DLE	STX	A	B	C	D	DLE	ETX
-----	-----	---	---	---	---	-----	-----

Khung bắt đầu/ kết thúc với các chuỗi ký tự đặc biệt

A	DLE	C
---	-----	---

DLE	STX	A	DLE	DLE	C	DLE	ETX
-----	-----	---	-----	-----	---	-----	-----

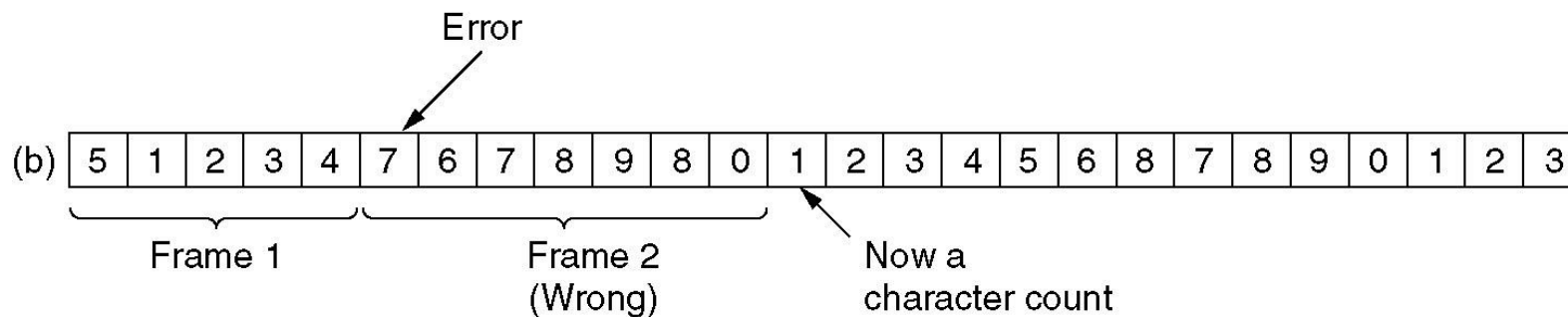
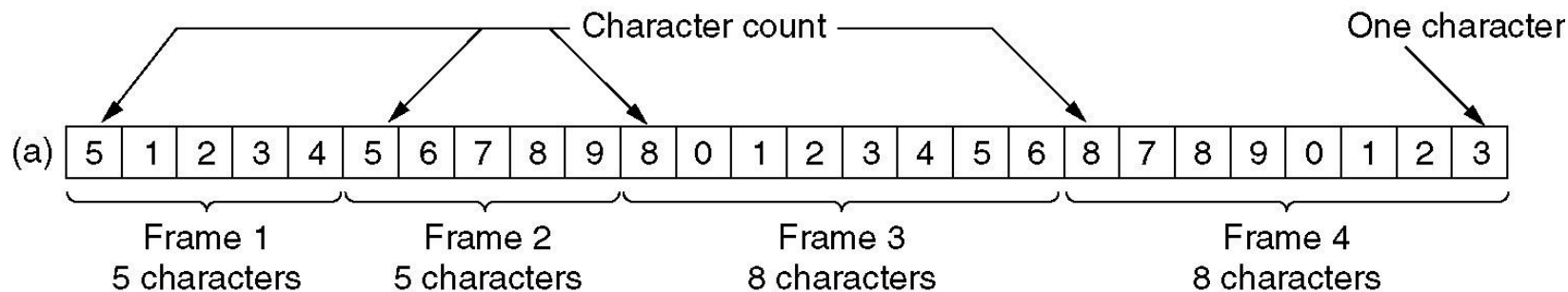
Ký hiệu

DLE: Data link escape

STX: Start Text

ETX: End Text

Đếm ký tự



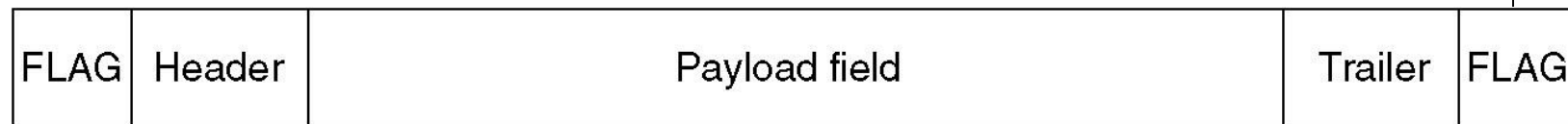
Chuỗi ký tự

(a) Không có lỗi. (b) Có 1 lỗi.

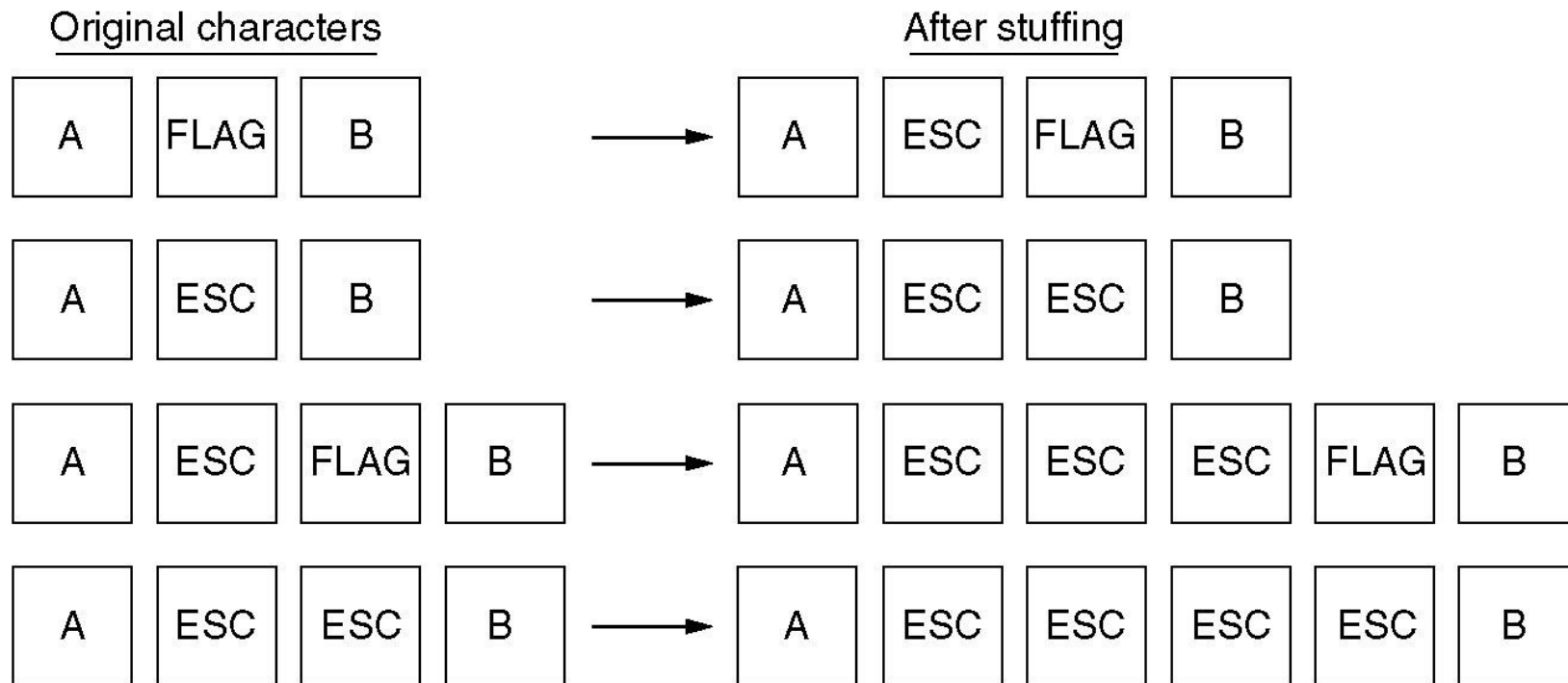
Vấn đề của kỹ thuật đếm ký tự

- Có thể đếm sai
 - Ngay cả khi dữ liệu có mã kiểm tra lỗi, và bên nhận biết rằng khung có lỗi nhưng không có cách nào biết khung tiếp theo bắt đầu từ đâu!
 - Yêu cầu truyền lại không tác dụng vì điểm bắt đầu của khung truyền lại là không xác định!
- ➔ Không còn được sử dụng

Xác định khung bằng cờ (Flag)



(a)



(b)

Vấn đề của kỹ thuật byte stuffing


Giả thiết kích thước ký tự là cố định - 8 bits

➔ Không xử lý được các trường hợp khác!

Các byte cờ có kiểm tra bit

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0



Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

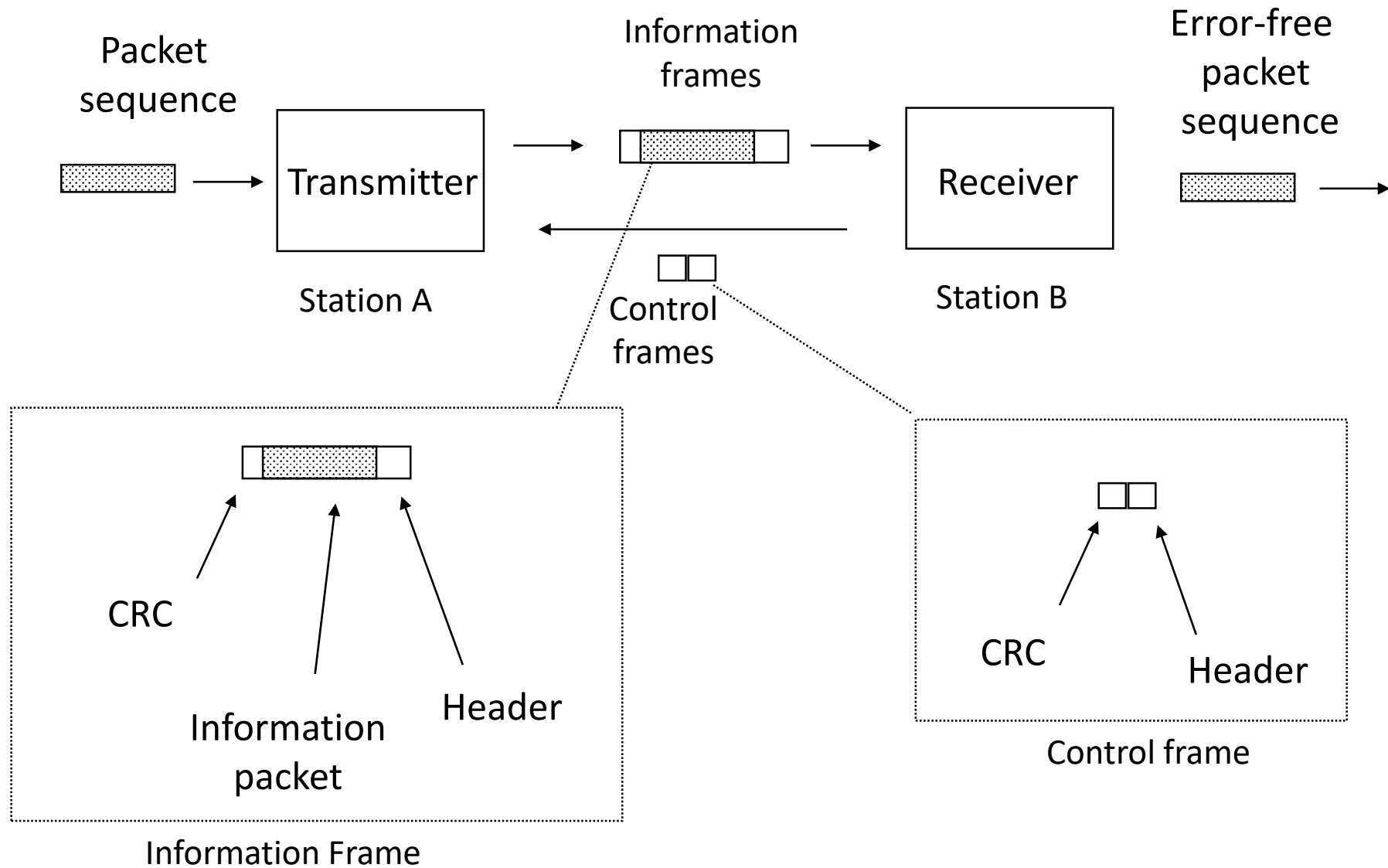
Kiểm tra bit

(a) Dữ liệu gốc.

(b) Dữ liệu truyền trên đường truyền.

(c) Dữ liệu ở bên nhận.

Các thành phần trong quá trình truyền và nhận



Khai báo trong lập trình mạng

```
#define MAX_PKT 1024                                /* determines packet size in bytes */

typedef enum {false, true} boolean;                  /* boolean type */
typedef unsigned int seq_nr;                          /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet; /* packet definition */
typedef enum {data, ack, nak} frame_kind;            /* frame_kind definition */

typedef struct {                                      /* frames are transported in this layer */
    frame_kind kind;                                  /* what kind of a frame is it? */
    seq_nr seq;                                       /* sequence number */
    seq_nr ack;                                       /* acknowledgement number */
    packet info;                                      /* the network layer packet */
} frame;
```

Còn tiếp →

```
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

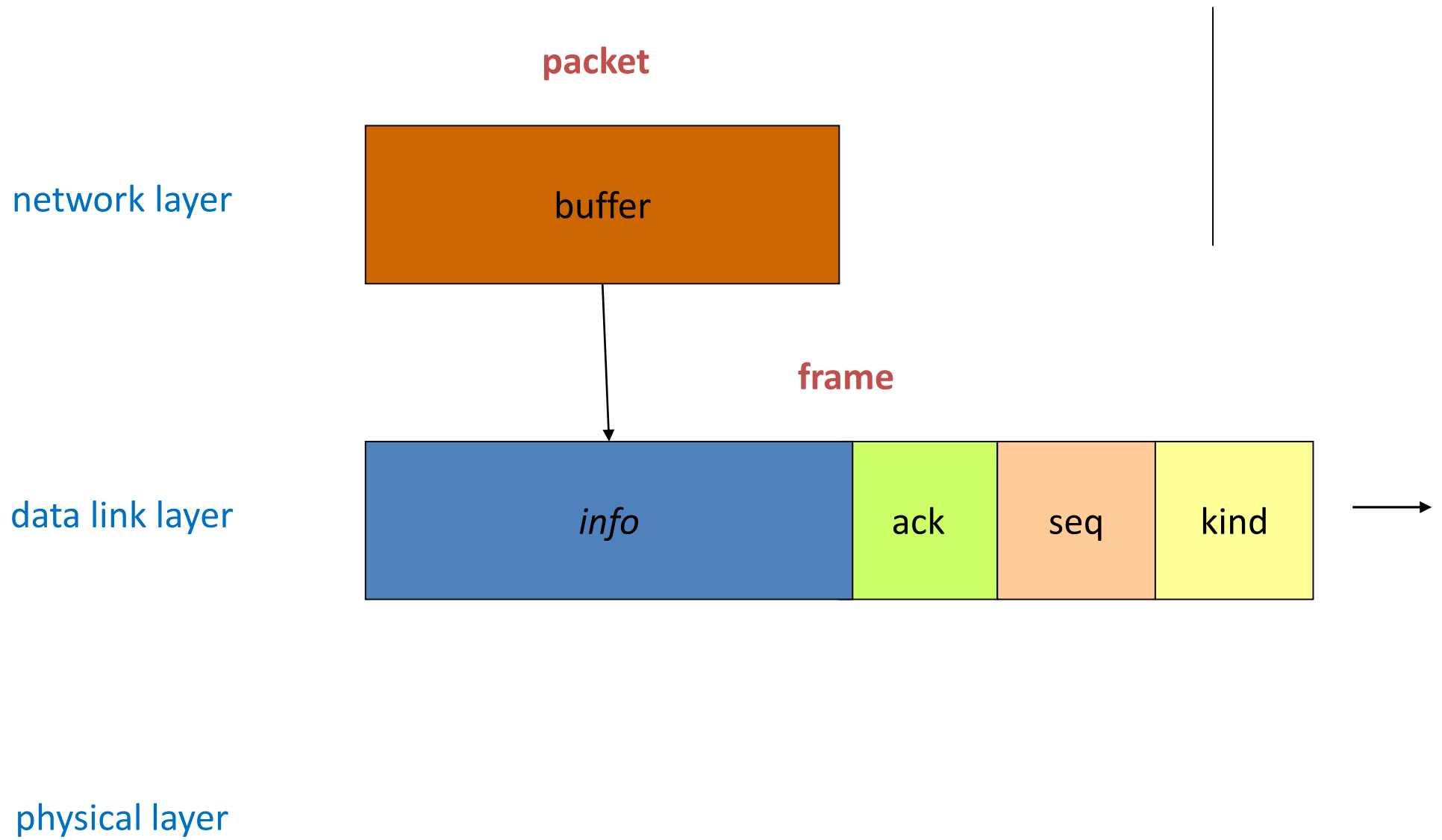
/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```



/* Protocol 1 (utopia) provides for data transmission in one direction only, from sender to receiver. The communication channel is assumed to be error free, and the receiver is assumed to be able to process all the input infinitely quickly. Consequently, the sender just sits in a loop pumping data out onto the line as fast as it can. */

```
typedef enum {frame arrival} event type;
#include "protocol.h"
```

```
void sender1(void)
{
    frame s;                                /* buffer for an outbound frame */
    packet buffer;                          /* buffer for an outbound packet */

    while (true) {
        from_network_layer(&buffer); /* go get something to send */
        s.info = buffer;             /* copy it into s for transmission */
        to_physical_layer(&s);       /* send it on its way */
    }                                /* Tomorrow, and tomorrow, and tomorrow,
                                   Creeps in this petty pace from day to day
                                   To the last syllable of recorded time
                                   - Macbeth, V, v */
}

void receiver1(void)
{
    frame r;
    event_type event;                      /* filled in by wait, but not used here */

    while (true) {
        wait_for_event(&event);           /* only possibility is frame_arrival */
        from_physical_layer(&r);          /* go get the inbound frame */
        to_network_layer(&r.info);       /* pass the data to the network layer */
    }
}
```

/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from sender to receiver. The communication channel is once again assumed to be error free, as in protocol 1. However, this time, the receiver has only a finite buffer capacity and a finite processing speed, so the protocol must explicitly prevent the sender from flooding the receiver with data faster than it can be handled. */

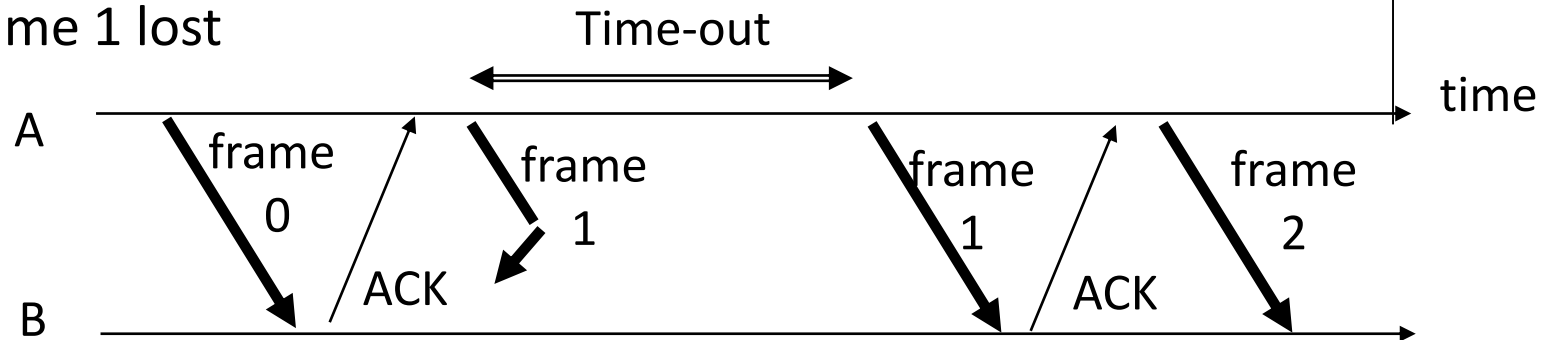
```
typedef enum {frame_arrival} event_type;  
#include "protocol.h"
```

```
void sender2(void)  
{  
    frame s;                                /* buffer for an outbound frame */  
    packet buffer;                          /* buffer for an outbound packet */  
    event_type event;                       /* frame_arrival is the only possibility */  
  
    while (true) {  
        from_network_layer(&buffer);        /* go get something to send */  
        s.info = buffer;                    /* copy it into s for transmission */  
        to_physical_layer(&s);              /* bye bye little frame */  
        wait_for_event(&event);             /* do not proceed until given the go ahead */  
    }  
}  
  
void receiver2(void)  
{  
    frame r, s;                             /* buffers for frames */  
    event_type event;                       /* frame_arrival is the only possibility */  
    while (true) {  
        wait_for_event(&event);             /* only possibility is frame_arrival */  
        from_physical_layer(&r);            /* go get the inbound frame */  
        to_network_layer(&r.info);          /* pass the data to the network layer */  
        to_physical_layer(&s);              /* send a dummy frame to awaken sender */  
    }  
}
```

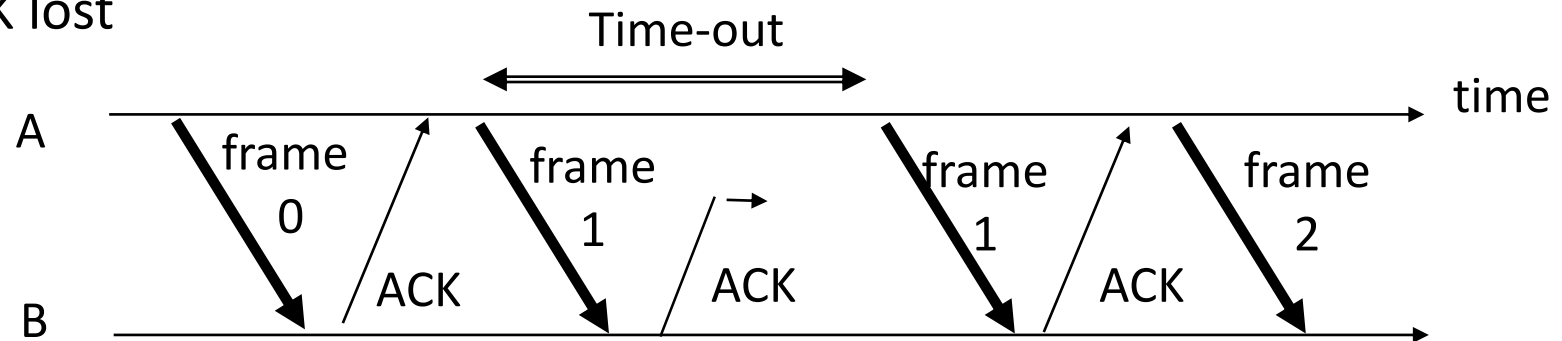
Ambiguities with Stop-and-Wait

[Không đánh số khung]

(a) Frame 1 lost

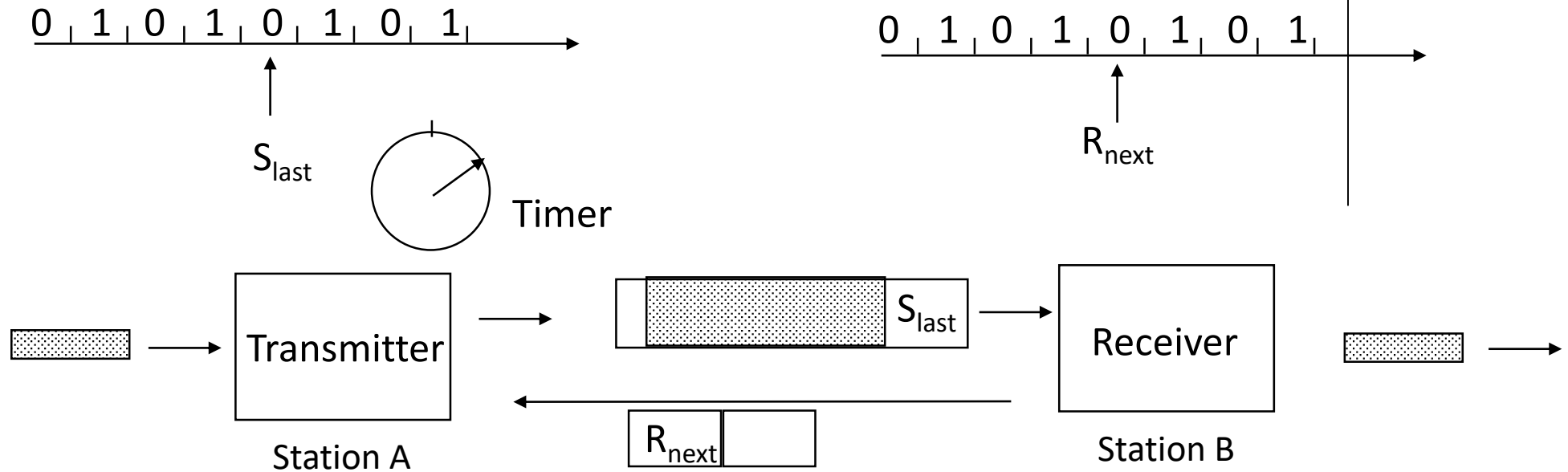


(b) ACK lost

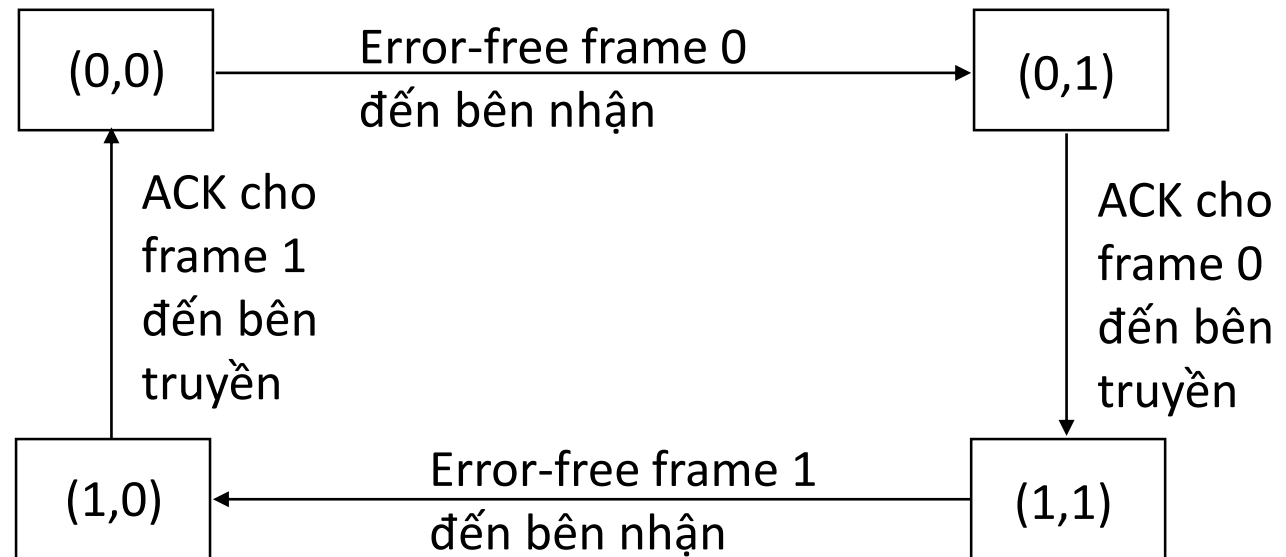


Ở (a) và (b) bên truyền A truyền cùng 1 cách, nhưng (b) bên nhận B nhận frame 1 hai lần.

State Machine for Stop-and-Wait



Global State:
 (S_{last}, R_{next})



Protocol 3 (PAR) Positive ACK with Retransmission

```
#define MAX_SEQ 1
typedef enum {frame_arrival, cksum_err, timeout} event_type;
include "protocol.h"
void sender_par (void)
{
    seq_nr next_frame_to_send;
    frame s;
    packet buffer;
    event_type event;
    next_frame_to_send = 0;
    from_network_layer (&buffer);
    while (true)
    {
        s.info = buffer;
        s.seq = next_frame_to_send;
        to_physical_layer (&s);
        start_timer (s.seq);
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_network_layer (&buffer);
            inc (next_frame_to_send);
        }
    }
}
```

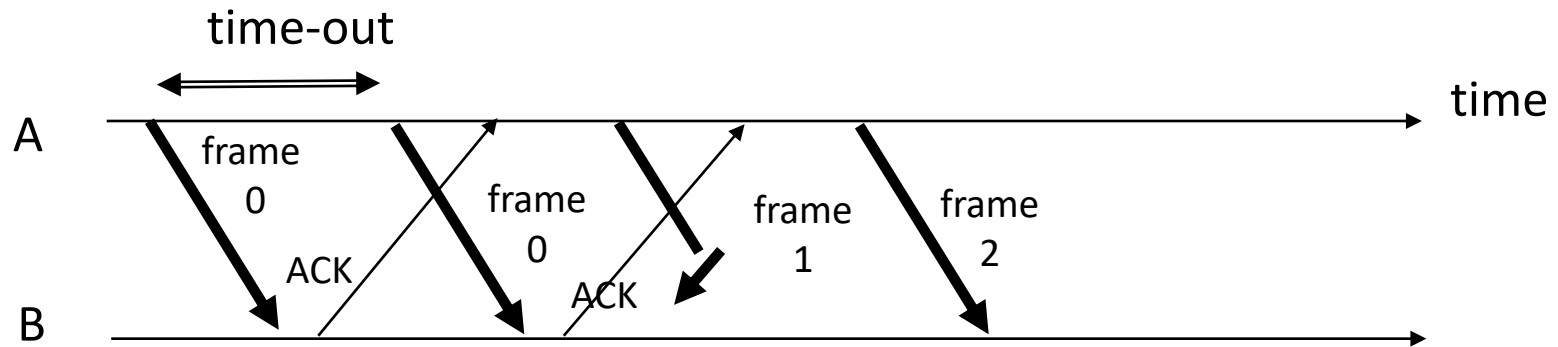

Protocol 3 (PAR) Positive ACK with Retransmission

```
void receiver_par (void)
{
    seq_nr next_frame_to_send;
    frame r, s;
    event_type event;
    frame_expected = 0;
    while (true)
    {
        wait_for_event (&event);
        if (event == frame_arrival)
        {
            from_physical_layer (&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc (frame_expected);
            }
            to_physical_layer (&s);
        }
    }
}
```

on ACK */

/* Note – no sequence number

Vấn đề của PAR Khi **Không đánh số** ACKs



Bên truyền A không xử lý trường hợp có 2 ACKs giống nhau

```

/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */
#define MAX_SEQ 1 /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
    seq_nr next_frame_to_send; /* seq number of next outgoing frame */
    frame s; /* scratch variable */
    packet buffer; /* buffer for an outbound packet */
    event_type event;

    next_frame_to_send = 0; /* initialize outbound sequence numbers */
    from_network_layer(&buffer); /* fetch first packet */
    while (true) {
        s.info = buffer; /* construct a frame for transmission */
        s.seq = next_frame_to_send; /* insert sequence number in frame */
        to_physical_layer(&s); /* send it on its way */
        start_timer(s.seq); /* if answer takes too long, time out */
        wait_for_event(&event); /* frame_arrival, cksum_err, timeout */
        if (event == frame_arrival) {
            from_physical_layer(&s); /* get the acknowledgement */
            if (s.ack == next_frame_to_send) {
                stop_timer(s.ack); /* turn the timer off */
                from_network_layer(&buffer); /* get the next one to send */
                inc(next_frame_to_send); /* invert next_frame_to_send */
            }
        }
    }
}

```

Còn tiếp →

```

void receiver3(void)
{
    seq_nr frame_expected;
    frame r, s;
    event_type event;

    frame_expected = 0;
    while (true) {
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            s.ack = 1 - frame_expected;
            to_physical_layer(&s);
        }
    }
}

```

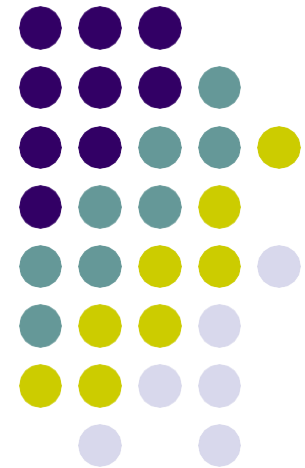
/* possibilities: frame_arrival, cksum_err */
 /* a valid frame has arrived. */
 /* go get the newly arrived frame */
 /* this is what we have been waiting for. */
 /* pass the data to the network layer */
 /* next time expect the other sequence nr */

 /* tell which frame is being acked */
 /* send acknowledgement */

Kiểm soát lỗi

Phát hiện lỗi

Phát hiện và sửa lỗi



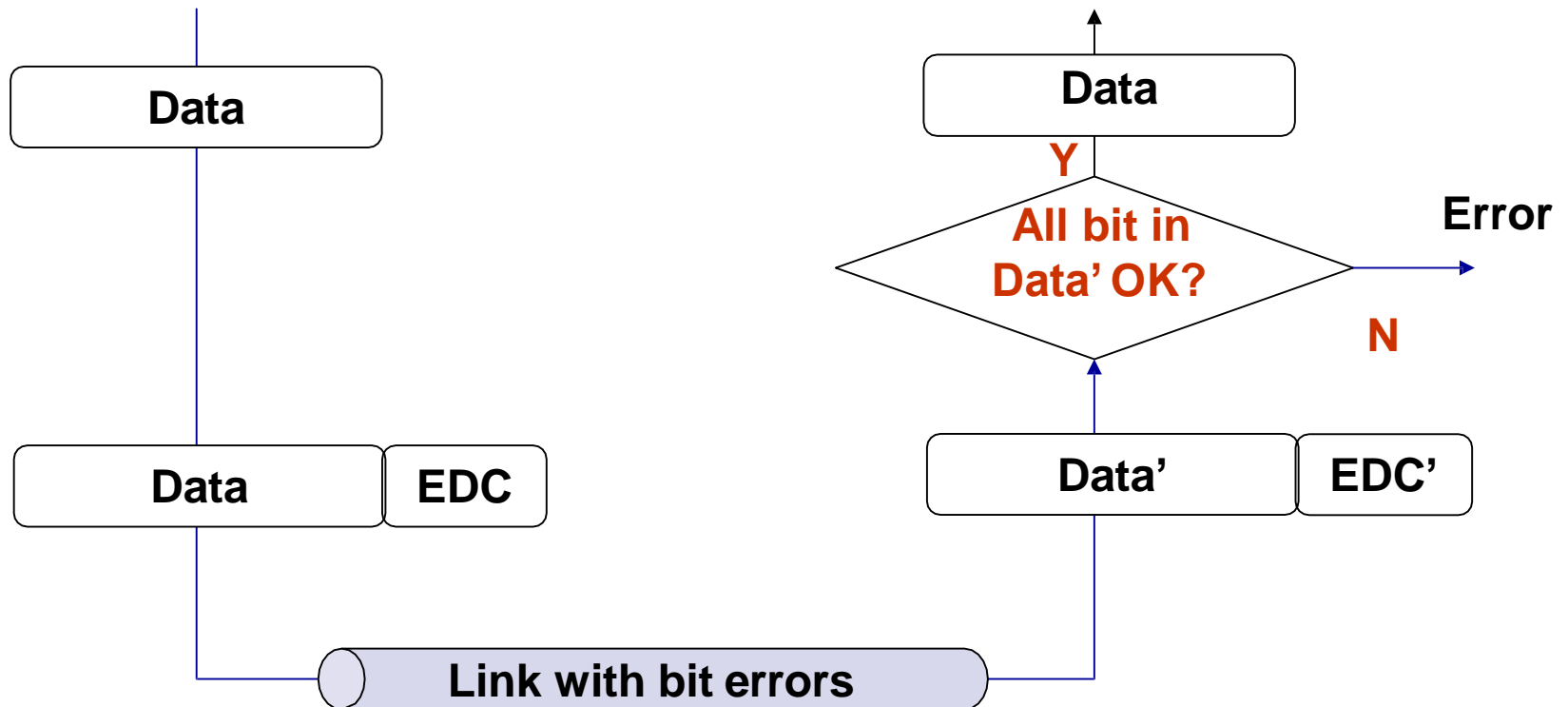
Phát hiện và sửa lỗi

- Trong một vài trường hợp, hệ thống phát hiện lỗi và yêu cầu gửi lại. Tuy vậy, nhiều trường hợp yêu cầu sửa lỗi.
- Ví dụ
 - Trên đường truyền độ tin cậy cao: Phát hiện lỗi là đủ vì tỉ lệ lỗi nhỏ, việc gửi lại thông tin bị lỗi sau đó không ảnh hưởng quá nhiều đến đường truyền.
 - Ngược lại, trên các đường truyền độ tin cậy thấp: gửi lại thông tin có thể vẫn bị lỗi, do vậy cần có cơ chế sửa lỗi.

Nguyên lý phát hiện lỗi



EDC= Error Detection Code (redundancy)
Mã phát hiện lỗi



Mã chẵn lẻ



- Mã đơn
 - **Phát hiện** lỗi bit đơn
- Mã hai chiều
 - **Phát hiện** và **sửa** lỗi bit đơn
- Khái niệm về checksum của Internet?

0111000110101011	0
------------------	---

101011	1	101011	1
111100	0	101100	0
011101	1	011101	1
001010	0	001010	0



Checksum

- Mã kiểm tra lỗi độ dài 16 bit
- Tại bên gửi
 - Đặt 16 bit của checksum = 0
 - Tổng theo các số 16 bits
 - Đảo bit tất cả
- Tại bên nhận
 - Tổng tất cả theo các số 16 bit
 - Phải thu được toàn các bit 1
 - Nếu không, gói tin bị lỗi



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
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

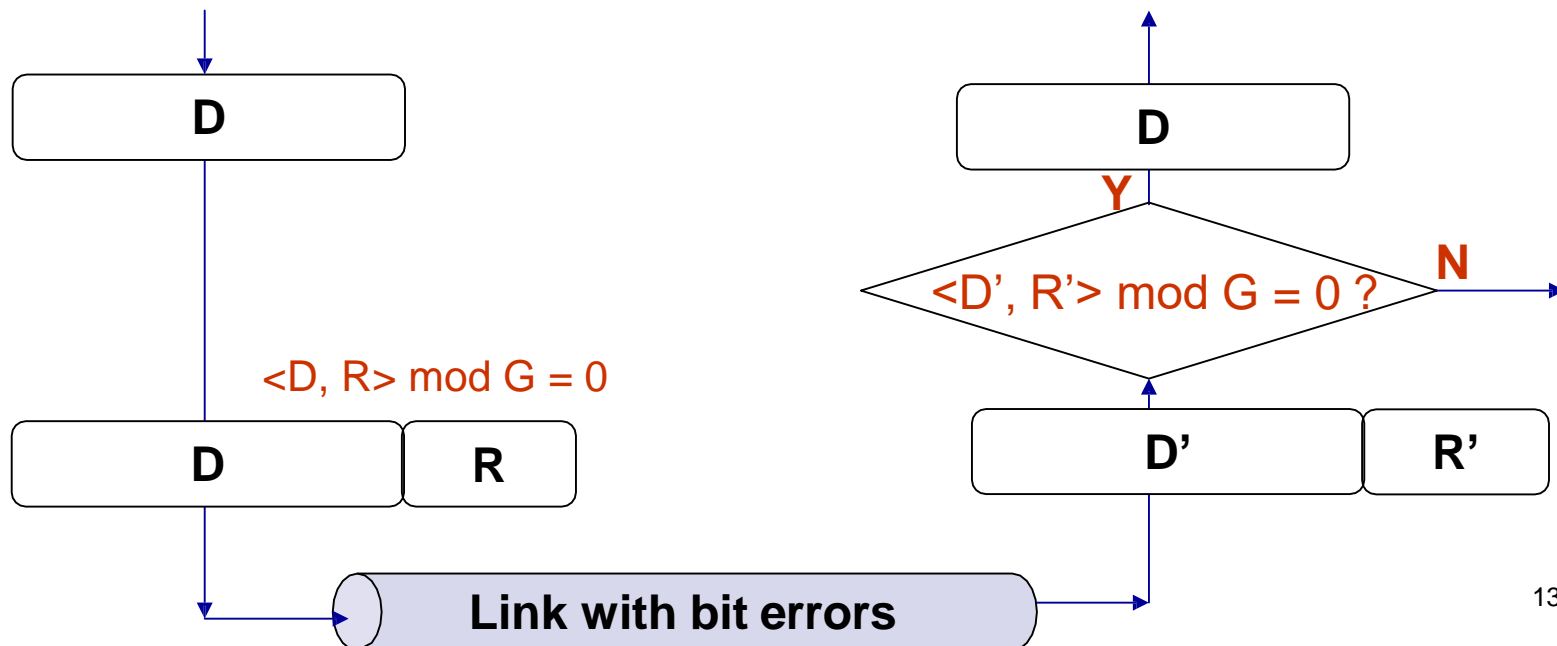
CRC - Cyclic Redundancy Check

- Dữ liệu cần gửi m gồm các bits thông tin biểu diễn bằng đa thức $M(x)$.
- r là bits kiểm tra biểu diễn bằng đa thức $R(x)$.
- Dữ liệu truyền $T(x) = M(x) + R(x)$
- Đa thức sinh $G(x)$ chia hết cho $T(x)$

CRC: Mã vòng



- Dữ liệu được xem như một số nhị phân: D
- Chọn một chuỗi $r+1$ bit, G (chuỗi sinh – Generator)
- Tìm một chuỗi R độ dài r bit, sao cho chuỗi ghép của D và R là một số nhị phân chia hết cho G (chia modulo 2)
 - $\langle D, R \rangle$ chia hết cho G



CRC: Cách tìm R



- $\langle D, R \rangle$ có thể viết dưới dạng
 - $D \cdot 2^r \text{ XOR } R$
- $\langle D, R \rangle$ chia hết cho G
 - $D \cdot 2^r \text{ XOR } R = n \cdot G$
 - $D \cdot 2^r = n \cdot G \text{ XOR } R$
- Có nghĩa là R là số dư khi chia $D \cdot 2^r$ cho G (phép chia modulo 2)

$$R = D \cdot 2^r \bmod G$$

$R=110$, chuỗi bit gửi đi là
 $\underbrace{10101001}_D \underbrace{110}_R$

• Ví dụ

$ \begin{array}{r} 10101001000 \\ \underline{1001} \quad \mathbf{D} \\ 1110 \\ 1001 \\ \underline{1110} \\ 1001 \\ \underline{1111} \\ 1001 \\ \underline{1100} \\ 1001 \\ \underline{1010} \\ 1001 \\ \underline{110} \\ \mathbf{R} \end{array} $	$ \begin{array}{r} \mathbf{G} \\ \underline{1001} \\ 1011110 \end{array} $
---	--

CRC biểu diễn dưới dạng đa thức



- 1011 : $x^3 + x + 1$
- Ví dụ một số mã CRC được sử dụng trong thực tế:
 - CRC-8 = $x^8 + x^2 + x + 1$
 - CRC-12 = $x^{12} + x^{11} + x^3 + x^2 + x$
 - CRC-16-CCITT = $x^{16} + x^{12} + x^5 + 1$
 - CRC-32 = $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- G càng dài, mã CRC phát hiện lỗi càng hiệu quả
- CRC được sử dụng rộng rãi trong thực tế
 - Wi-fi, ATM, Ethernet...
 - Phép toán XOR được cài đặt bởi phần cứng
 - Phát hiện chuỗi bit bị lỗi có độ dài nhỏ hơn $r+1$ bit

Kiểm soát truy nhập đường truyền MAC (Media Access Control)



Các dạng liên kết

- Điểm-nối-điểm
 - ADSL
 - Telephone modem
 - Leased Line....
- Quảng bá
 - Mạng LAN truyền thống với hình trạng bus hay mạng hình sao dùng hub (công nghệ lỗi thời)
 - Wireless LAN, radio network, mobile network
 - HFC:
 - ...
- Các mạng quảng bá cần giao thức điều khiển truy nhập để tránh xung đột -> Giao thức đa truy nhập

Phân loại các giao thức đa truy nhập



- Chia kênh:
 - Chia tài nguyên của đường truyền thành nhiều phần nhỏ (Thời gian - TDMA, Tần số - FDMA, Mã - CDMA)
 - Chia từng phần nhỏ đó cho các nút mạng
- Truy nhập ngẫu nhiên:
 - Kênh không được chia, cho phép đồng thời truy nhập, chấp nhận là có xung đột (collision)
 - Cần có cơ chế để phát hiện và tránh xung đột
 - e.g. Pure Aloha, Slotted Aloha, CSMA/CD, CSMA/CA...
- Làn lượt:
 - Theo hình thức quay vòng
 - Token Ring, Token Bus....

Các phương pháp chia kênh



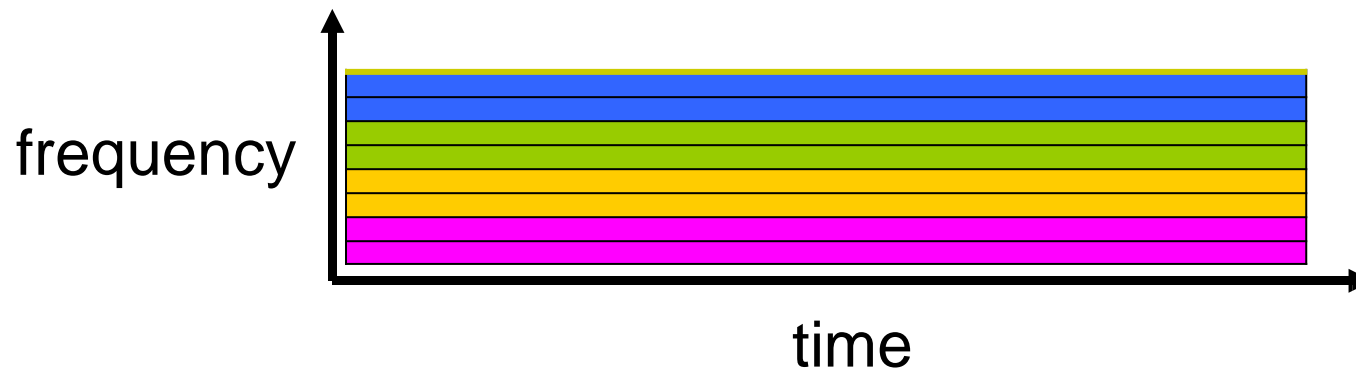
- FDMA: frequency division multiple access
- TDMA: time division multiple access
- CDMA: code division multiple access

TDMA và FDMA

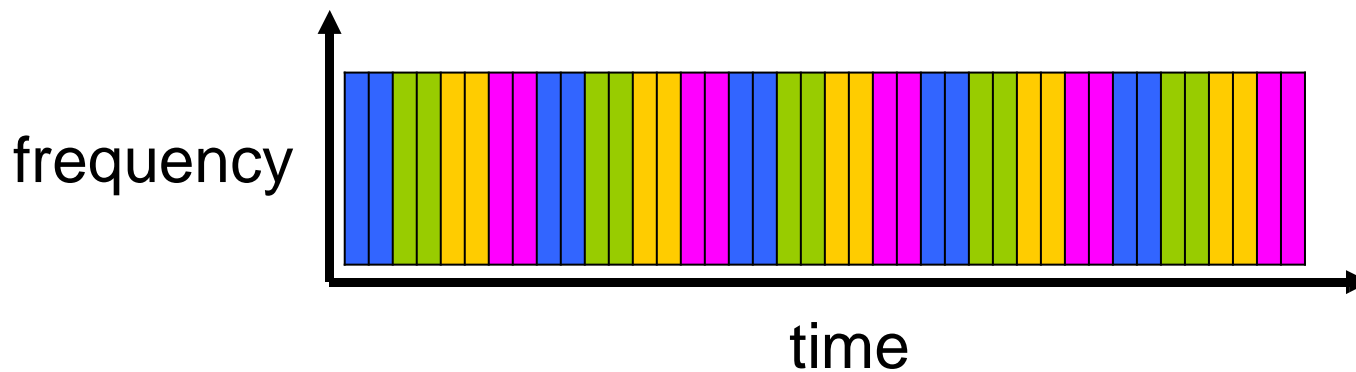
Ví dụ:
4 kênh



FDMA



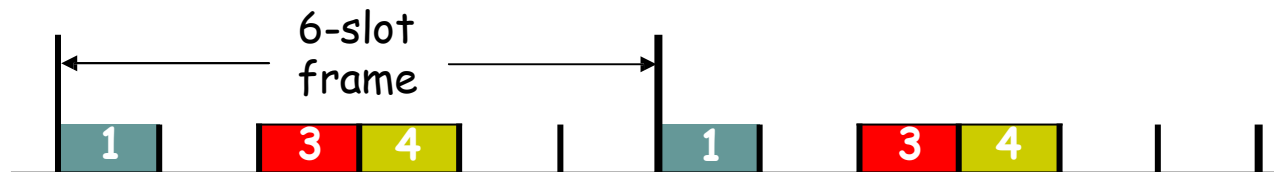
TDMA:



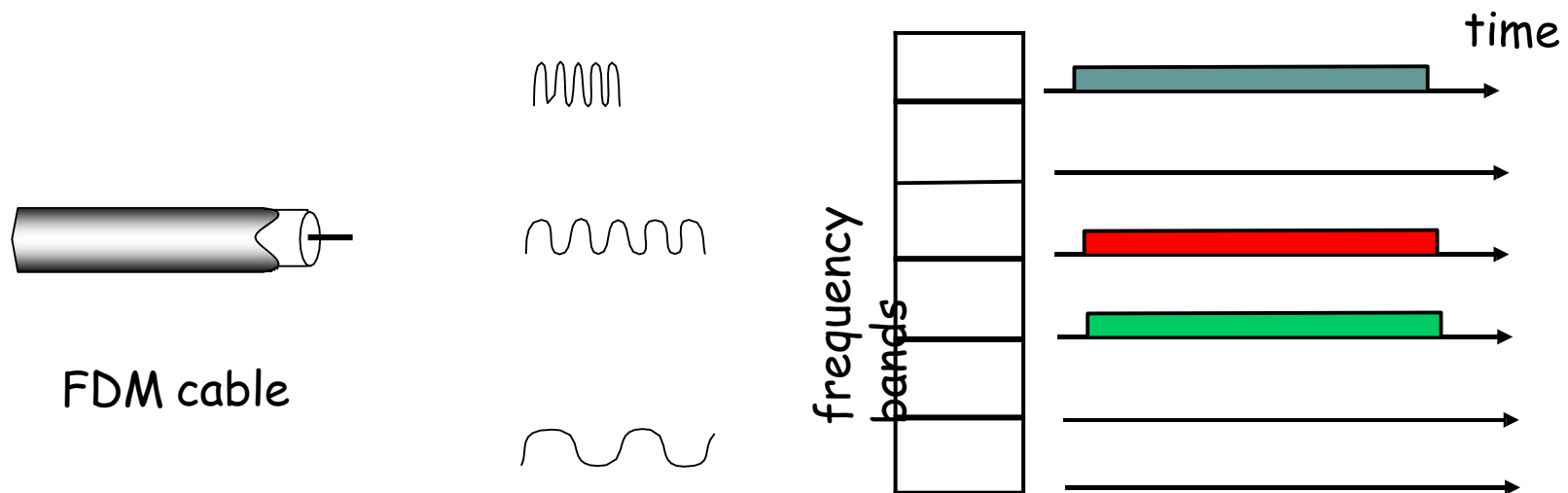


TDMA: Ví dụ

- Mạng LAN có 6 máy, 1,3,4 hoạt động. 2, 5, 6 nghỉ



FDMA: Ví dụ



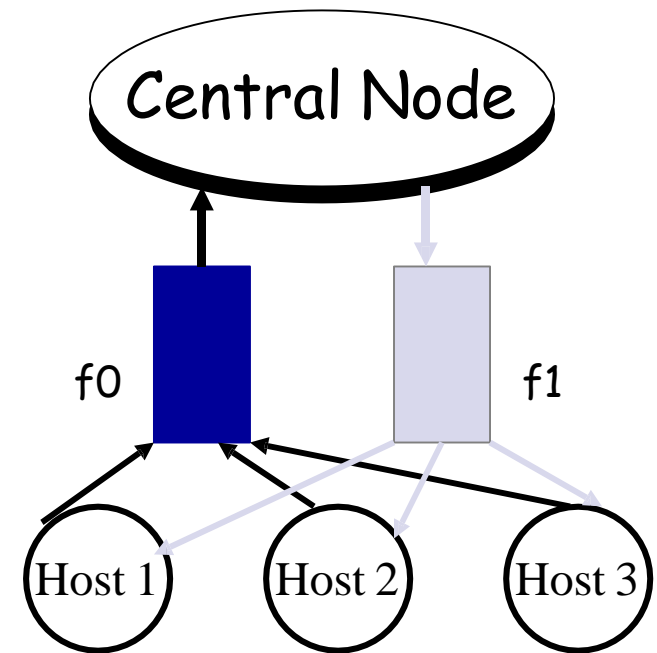
Các phương pháp truy cập ngẫu nhiên

Aloha

- Packet-Switched Radio Network
- Các nút truyền trên một tần số (f_0)
- Nút trung tâm nhận và truyền lại một tần số khác (f_1)

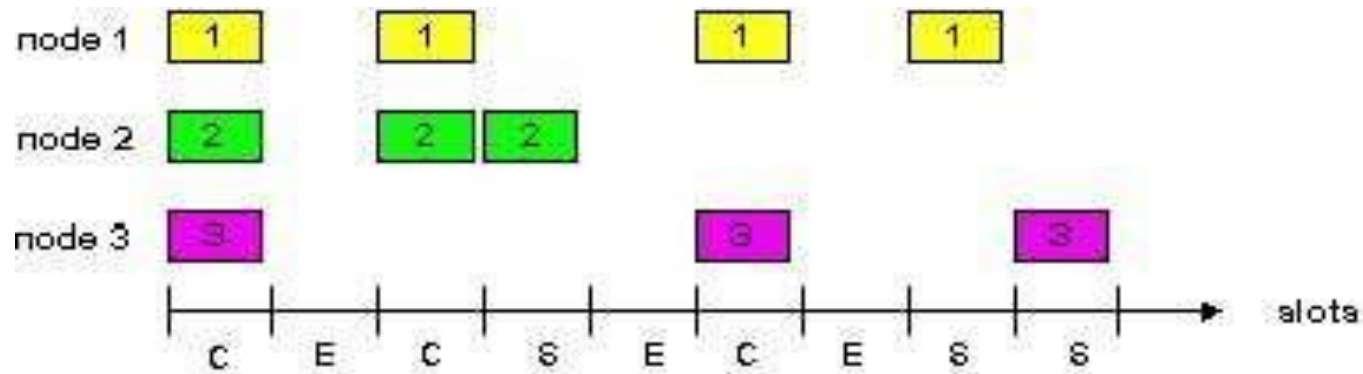
Nếu có hai nút cùng truyền: Xung đột

- Nếu có xung đột, nút vừa truyền sẽ nhận được một gói tin bị lỗi, nó sẽ đợi một thời gian ngẫu nhiên trước khi truyền lại





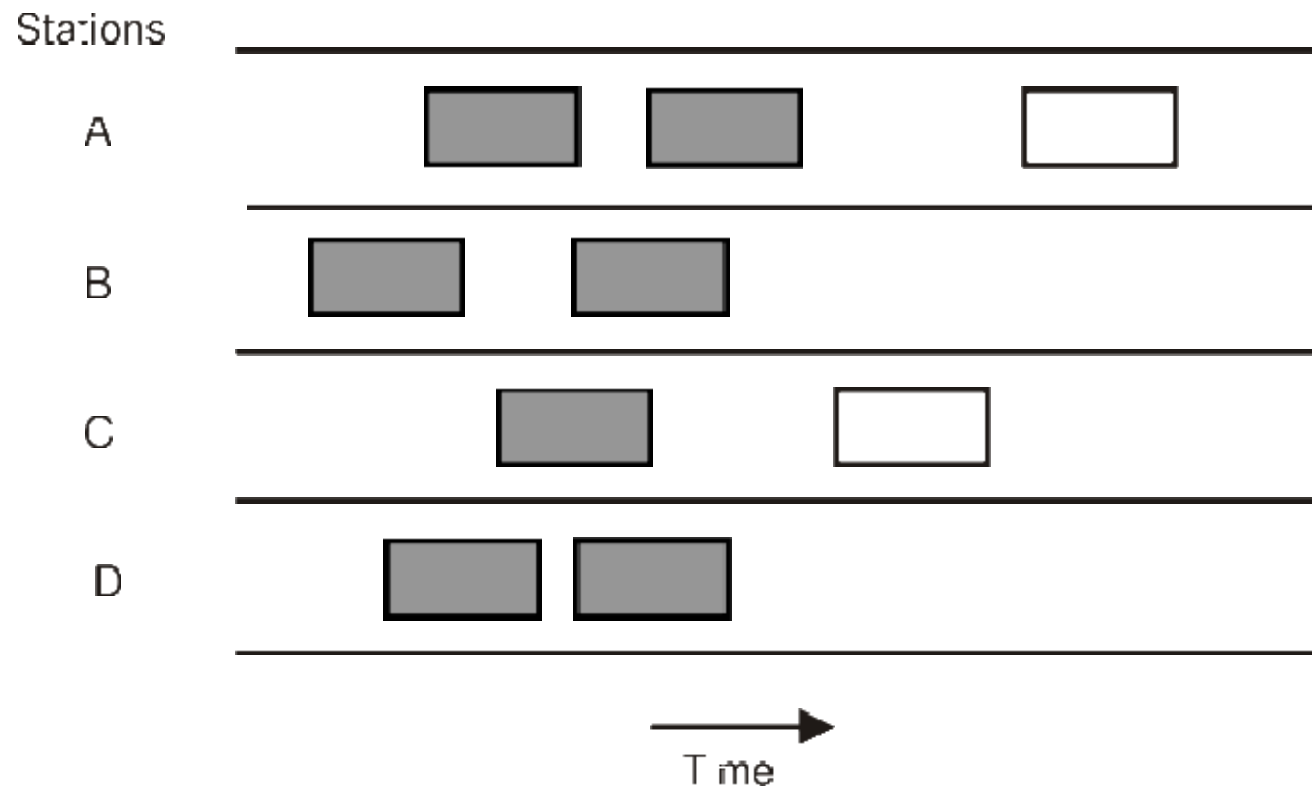
Slotted ALOHA



- Thời gian được chia làm các khe (slot) bằng nhau
- Dữ liệu có cùng kích thước (1 slot)
- Các nút phải đồng bộ hóa thời gian

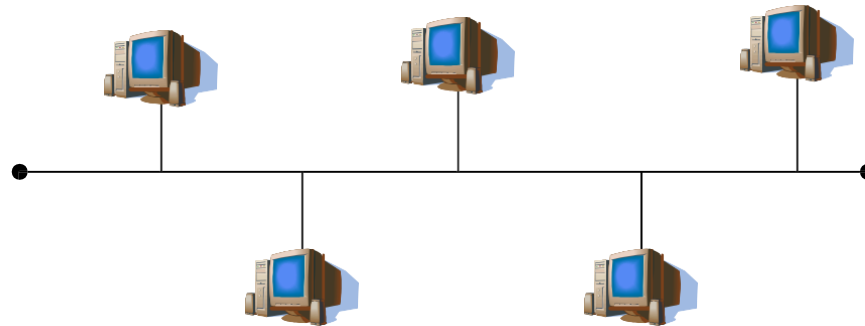


Pure ALOHA



Hãy qua kém hơn Slotted ALOHA!

CSMA/CD

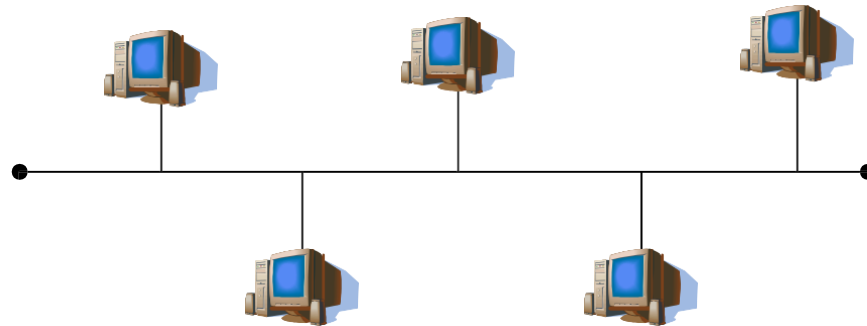


- Carrier Sense Multiple Access with Collision Detection (đồng truy nhập, có phát hiện xung đột)
- Thế nào là CSMA/CD: trong một cuộc họp
 - Multiple Access:
 - Collision:
 - CSMA: “Listen before talk”
 - CD
 - “Listen while talking”





CSMA/CD

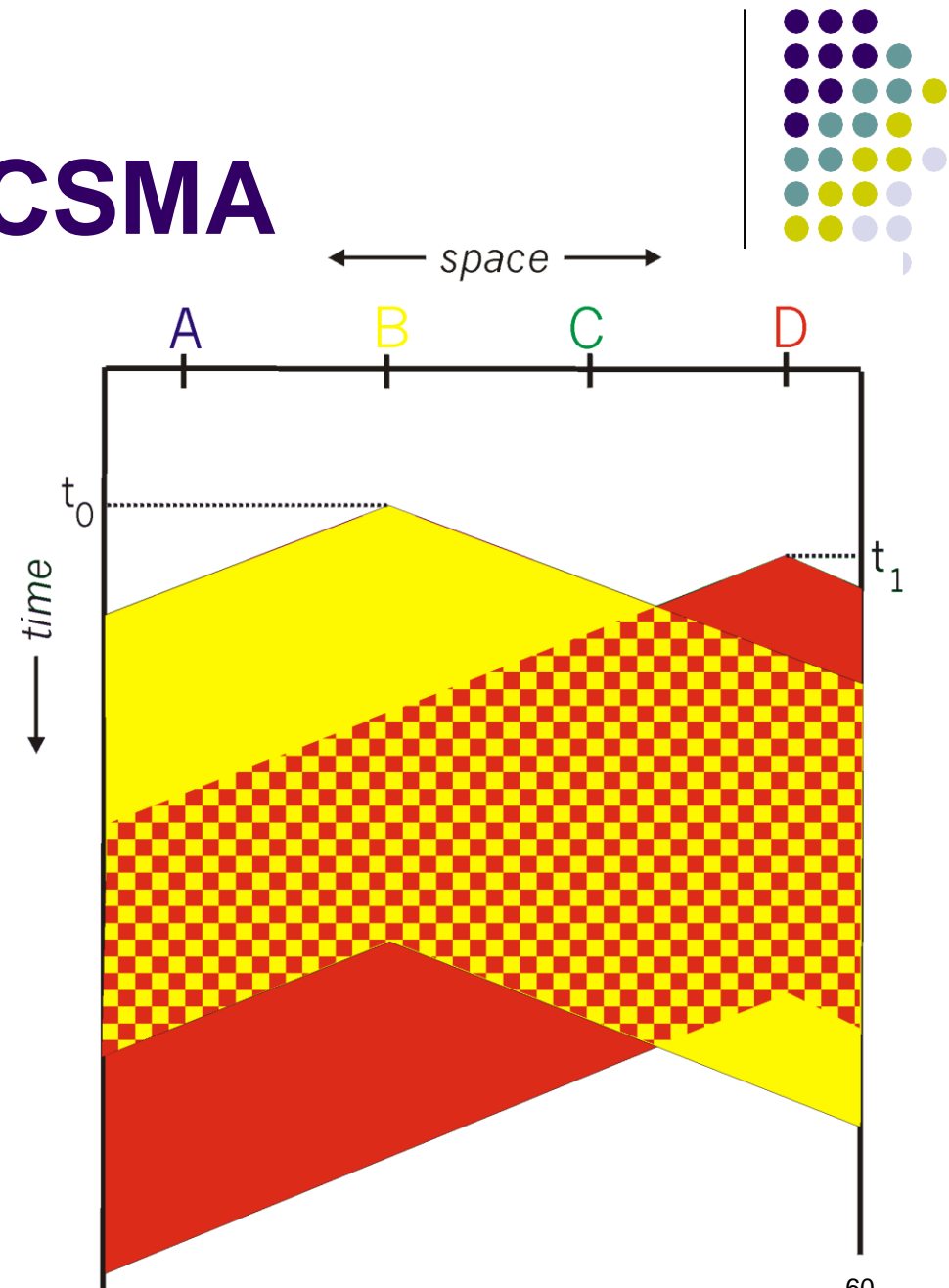


- **CSMA**: Các máy nghe trước muốn truyền:
 - Nếu kênh rỗi, truyền toàn bộ dữ liệu
 - Nếu kênh bận, chờ (rút lui và quay lại)
- Tại sao lại có xung đột?

độ trễ lan truyền

Xung đột trong CSMA

- Giả sử kênh truyền có 4 nút
- Tín hiệu điện từ lan truyền từ nút này đến nút kia mất một thời gian nhất định (trễ lan truyền)
- Ví dụ:





CSMA/CD: Tóm tắt

- Máy trạm nghe trước khi muốn truyền
 - Bận: Rút lui, sau đó quay lại tiếp tục nghe
 - Rỗi: Bắt đầu truyền, vừa truyền vừa “nghe ngóng” xem có xung đột hay không
 - Nghe trong thời gian bao lâu?
 - Nếu phát hiện thấy xung đột: Hủy bỏ quá trình truyền và quay lại trạng thái rút lui
- Sau khi rút lui, khi nào thì quay lại
 - *Exponential back-off*

So sánh chia kênh và truy nhập ngẫu nhiên

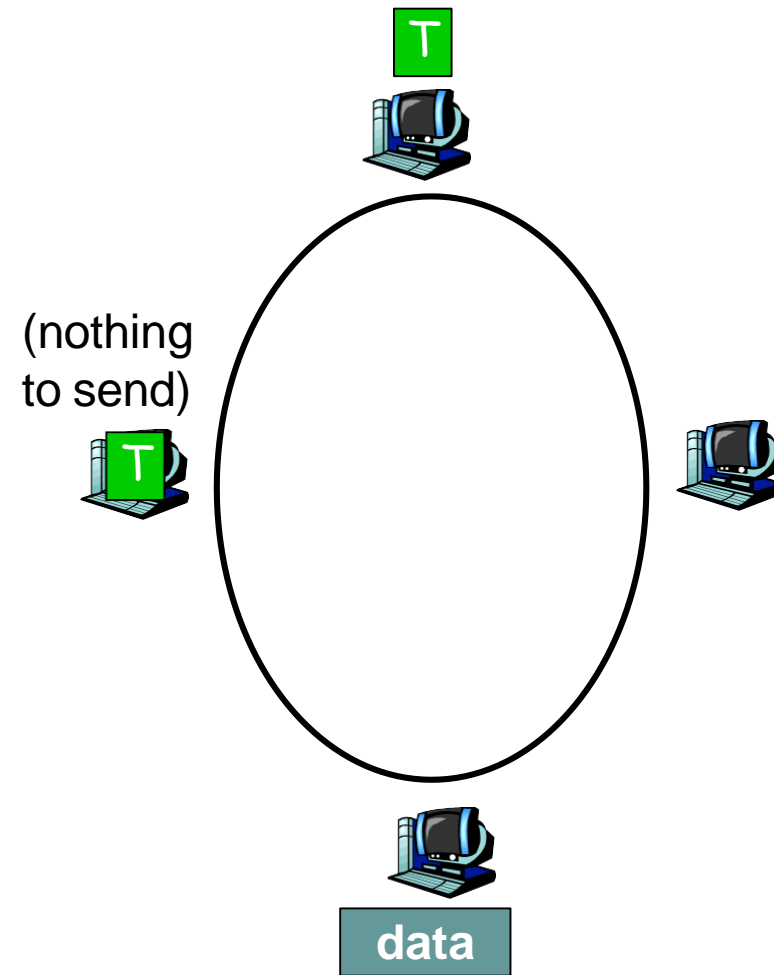


- Chia kênh
 - Hiệu quả, công bằng cho đường truyền với lưu lượng lớn
 - Lãng phí nếu chúng ta cấp kênh con cho một nút chỉ cần lưu lượng nhỏ
- Truy nhập ngẫu nhiên
 - Khi tải nhỏ: Hiệu quả vì mỗi nút có thể sử dụng toàn bộ kênh truyền
 - Tải lớn: Xung đột tăng lên
- Phương pháp quay vòng: Có thể dung hòa ưu điểm của hai phương pháp trên

Token Ring – Mạng vòng dùng thẻ bài



- Một “thẻ bài” luân chuyển lần lượt qua từng nút mạng
- Nút nào giữ thẻ bài sẽ được gửi dữ liệu
- Gửi xong phải chuyển thẻ bài đi
- Một số vấn đề
 - Tốn thời gian chuyển thẻ
 - Trễ
 - Mất thẻ bài....



Tổng kết các phương pháp kiểm soát đa truy nhập



- Chia kênh
- Truy nhập ngẫu nhiên
- Quay vòng
- Phân tích ưu, nhược điểm

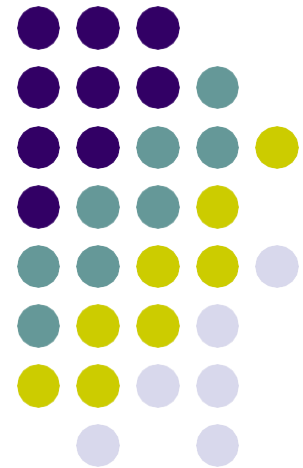


Thảo luận

- Trong phương pháp CSMA/CD, khi lượng dữ liệu cần gửi tăng lên thì:
 - Xung đột tăng lên?
 - Thông lượng tăng lên?
- Trong phương pháp TDMA, xung đột sẽ tăng lên khi lượng dữ liệu cần gửi tăng lên?
- Khi lượng dữ liệu cần gửi là rất nhiều, phương pháp Token Ring là kém hơn so với CSMA/CD
- Câu hỏi: Giải thích một cách định lượng hiệu quả của các phương pháp truy cập đường truyền (Bài tập lớn)

Điều khiển luồng

Khái niệm về truyền thông tin cậy
Các giải thuật kiểm soát luồng

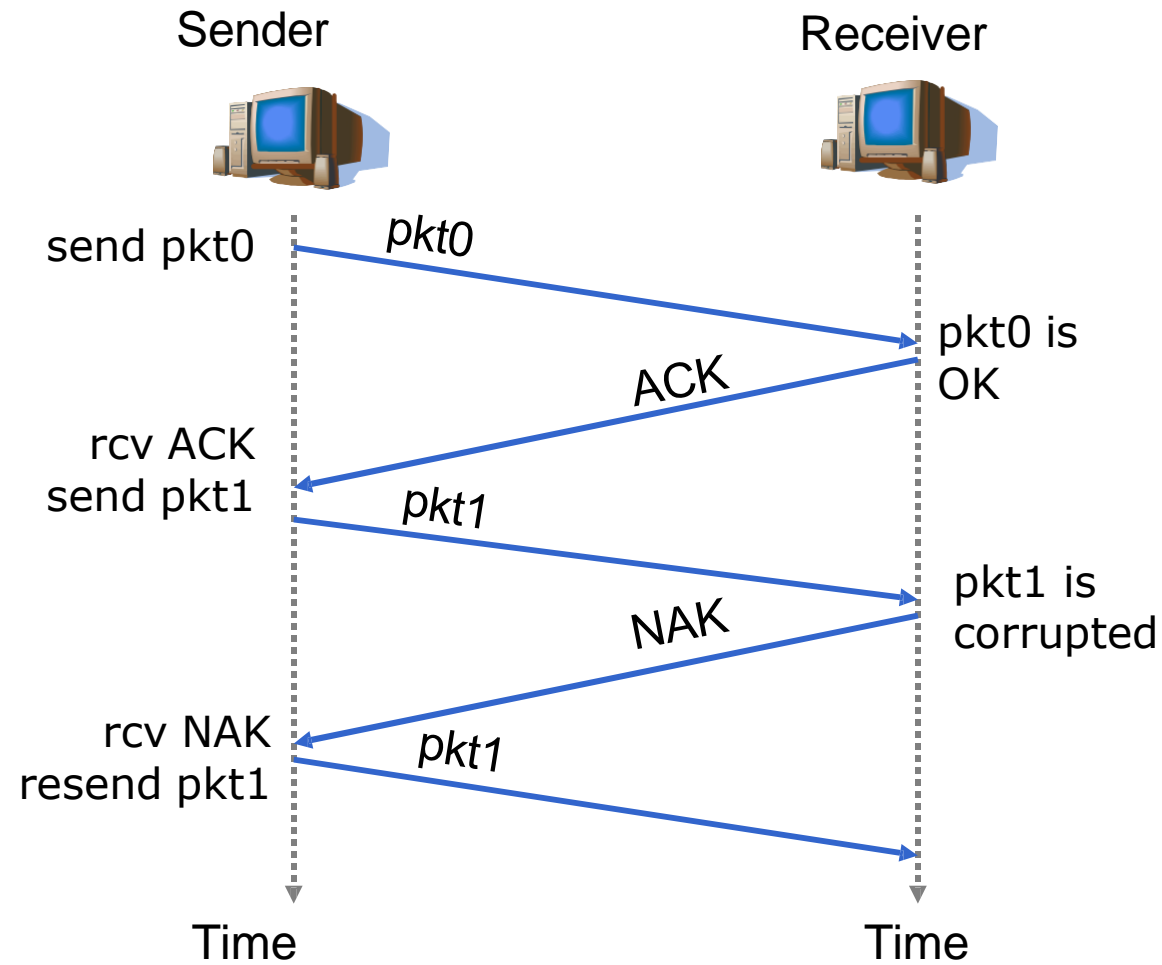


Kênh có lỗi bit, không bị mất tin



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

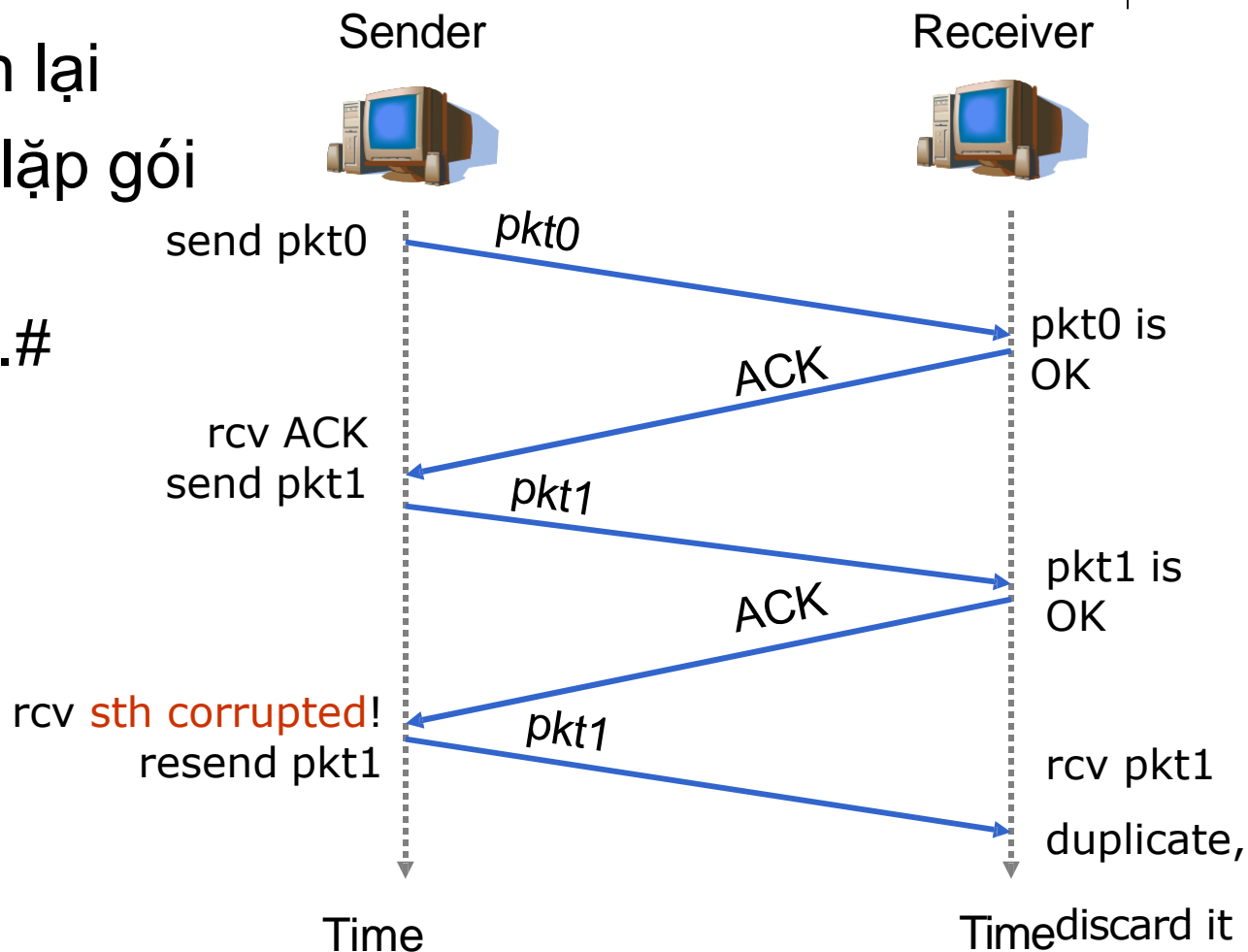
Hoạt động



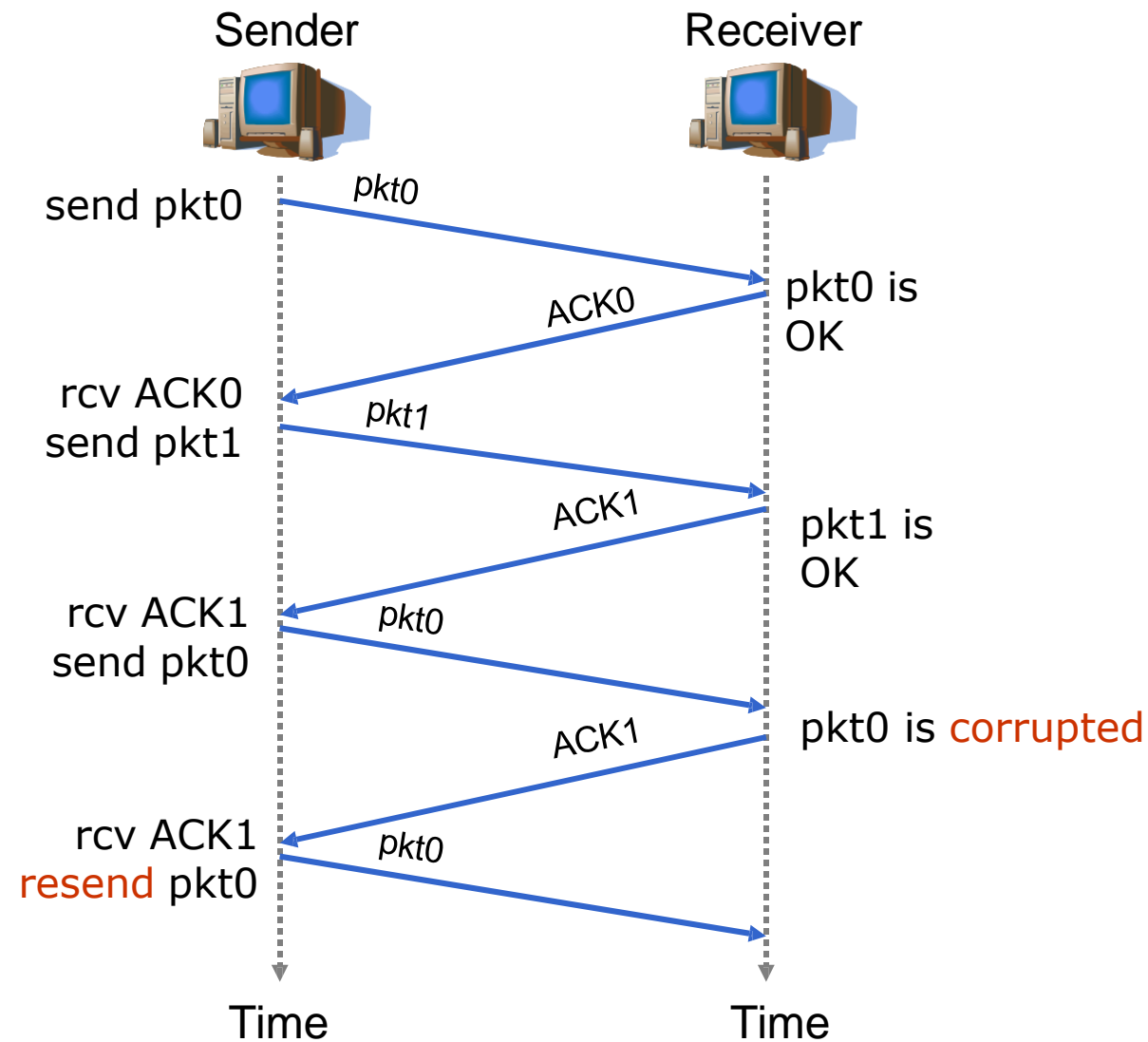
Lỗi ACK/NAK



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



Giải pháp không dùng NAK

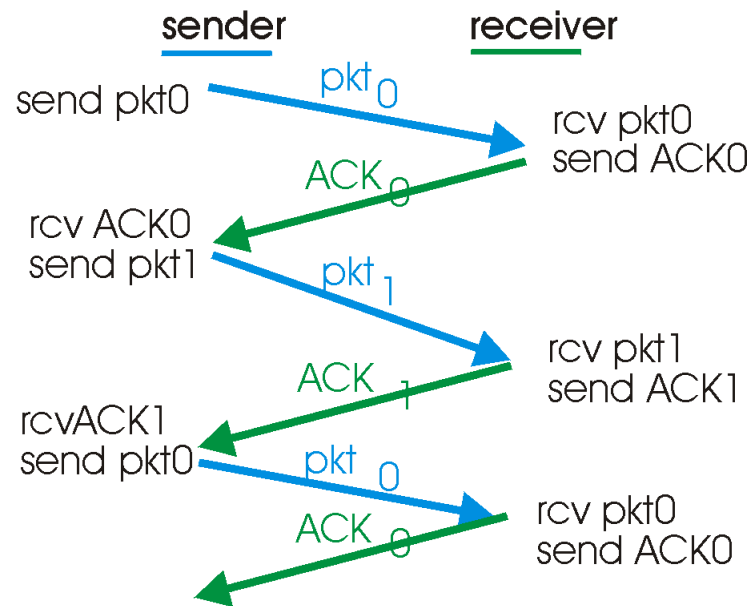




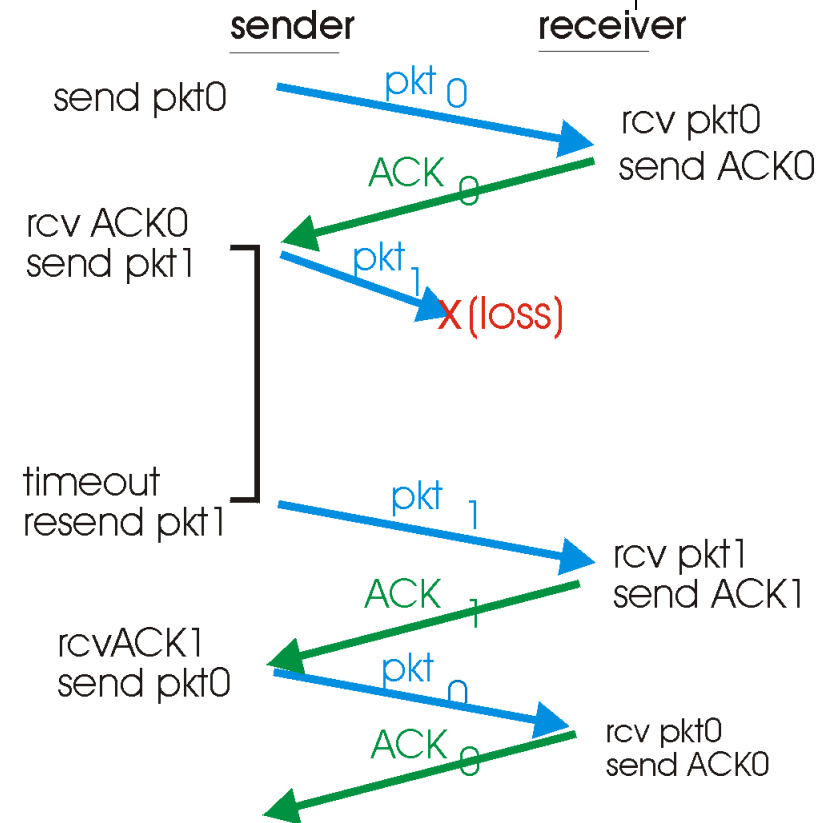
Kênh có lỗi bit và mất gói tin

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

Minh họa

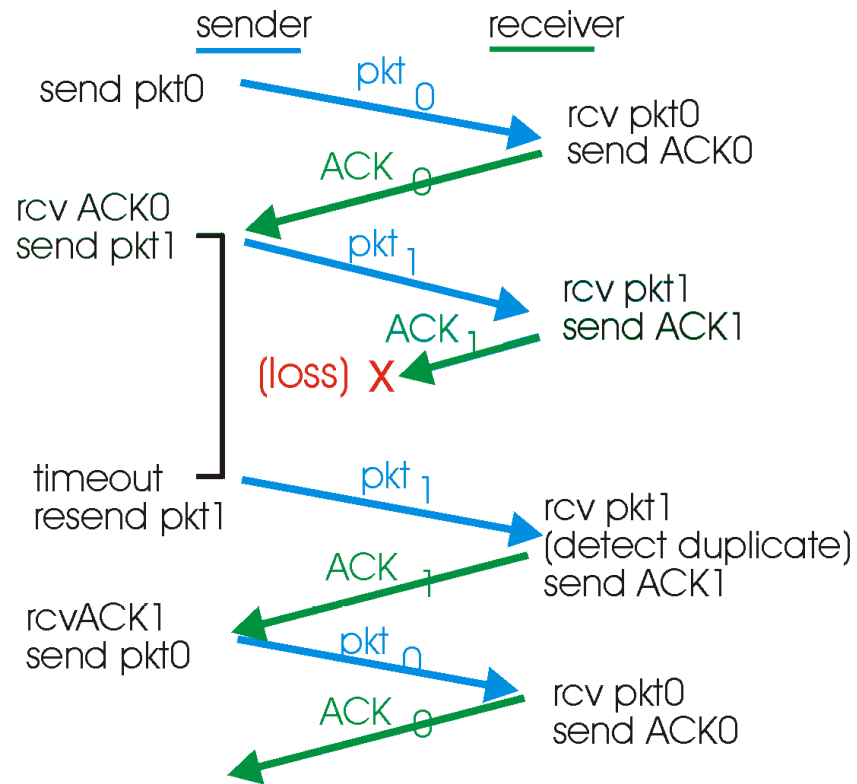


(a) operation with no loss

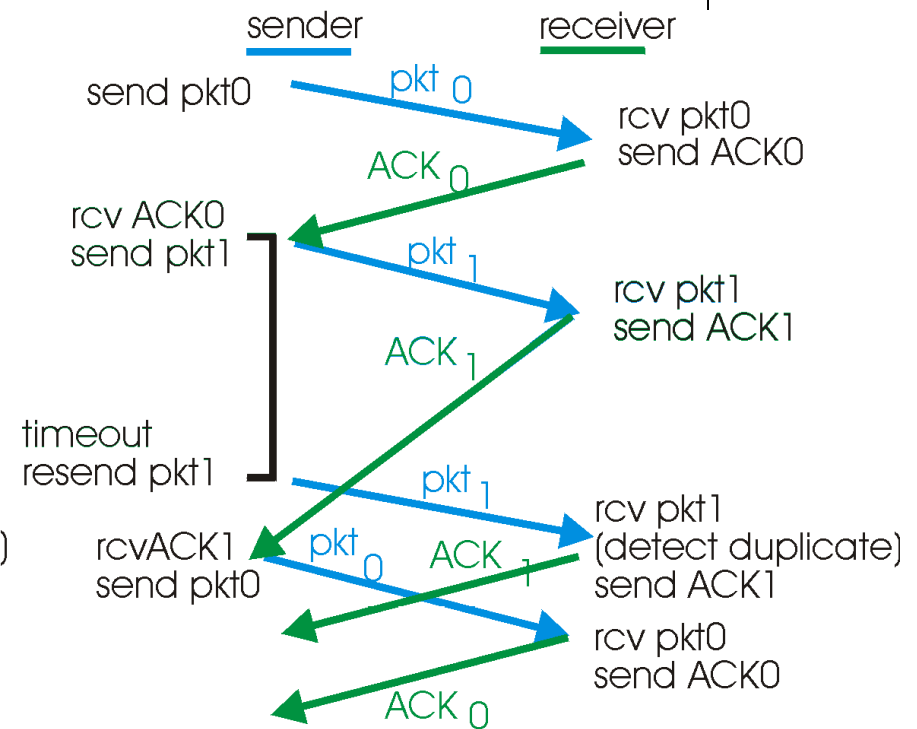


(b) lost packet

Minh họa

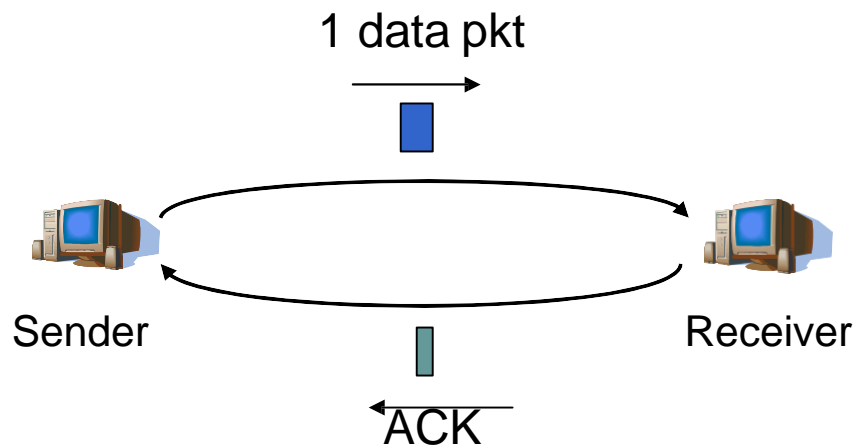


(c) lost ACK

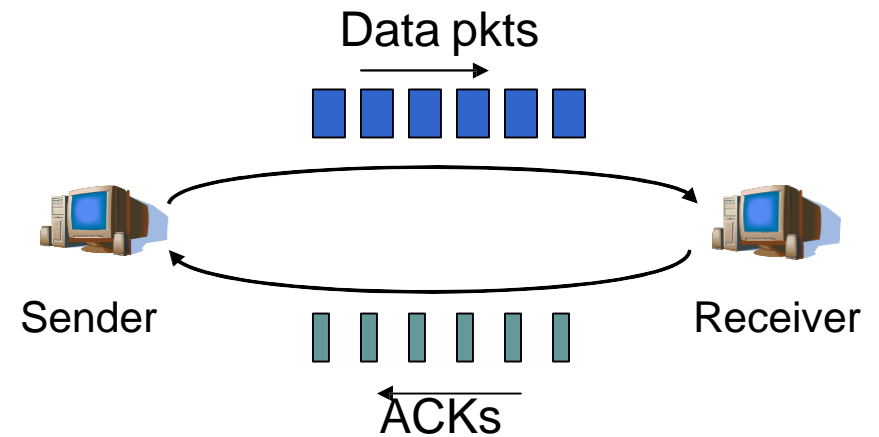


(d) premature timeout

Truyền theo kiểu pipeline



stop-and-wait

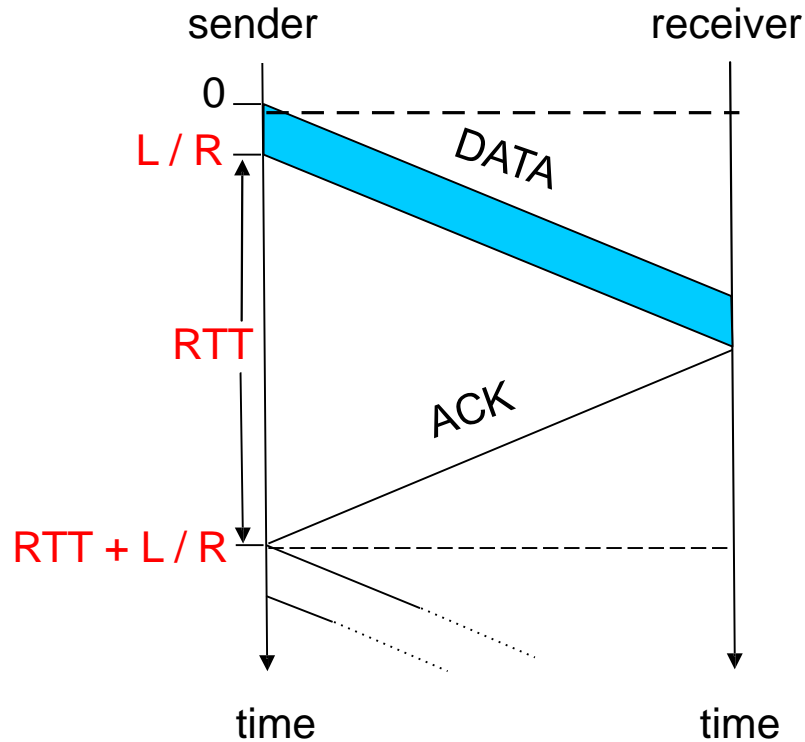


Pipeline

So sánh hiệu quả



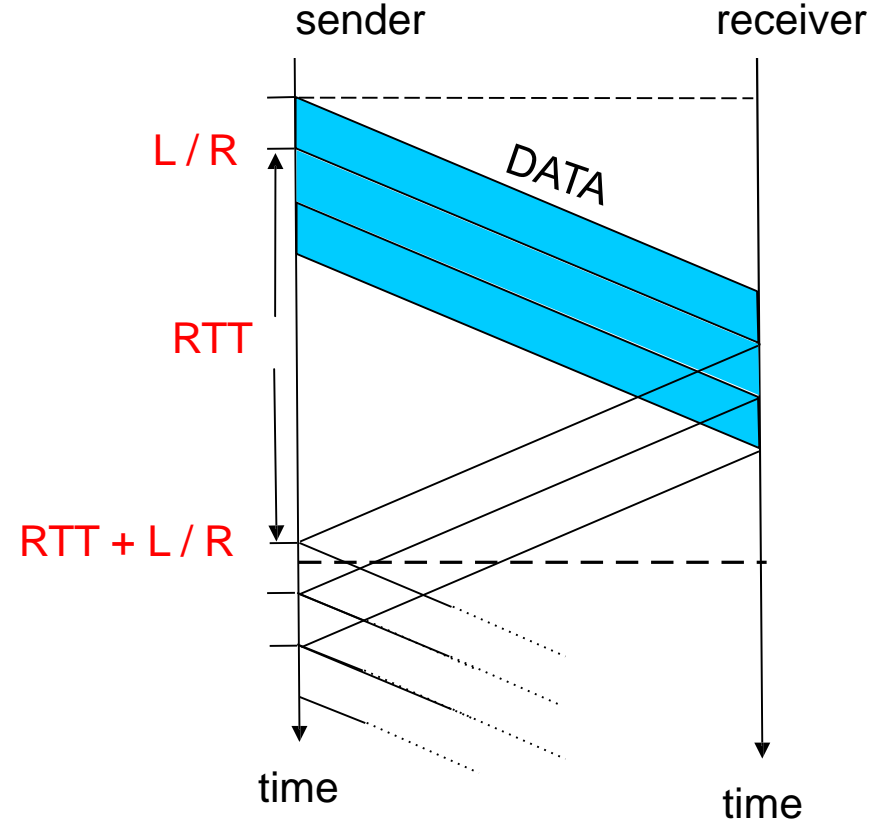
stop-and-wait



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

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

Pipeline



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

Các phương pháp gửi ACK

1. Số ACK xác định khung **cuối cùng** đã nhận thành công.

- HOẶC -

2. Số ACK xác định khung **tiếp theo** bên nhận muốn được nhận.

Giao thức cửa sổ trượt

Sliding Window Protocols

- Dữ liệu cần truyền theo cả 2 chiều
 - Có thể gửi ACK cùng với dữ liệu không?
- Phía nhận cần chờ bao lâu cho việc xác nhận là đã nhận thông tin qua gói tin ACKs ?
 - ➔ Cần có *ACK Timer*!!
- Nếu đường truyền tắc nghẽn, thời gian truyền lâu hơn bình thường?
 - ➔ Cần tăng khoảng thời gian giới hạn phía gửi

Giao thức cửa sổ trượt (2)

- Mỗi khung thông tin cần được đánh số.
- *Số thứ tự (sequence number) có n bits thì số lớn nhất là $\text{maxseq} = 2^n - 1$.*

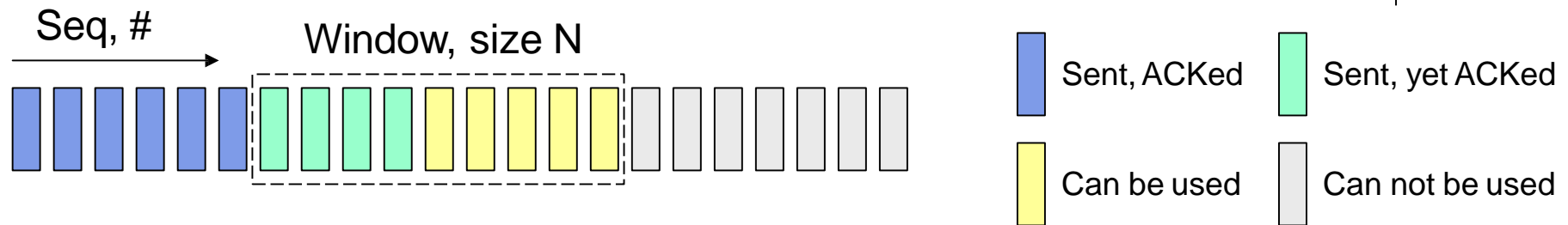
Sliding window ::

- **Bên gửi có cửa sổ** các khung thông tin và danh sách số thứ tự liên tiếp của các khung được phép gửi mà không cần chờ xác nhận ACKs.
- **Bên nhận cũng có cửa** chứa danh sách số thứ tự của các khung thông tin được phép phê duyệt ACK.
- **Chú ý**
 - *Cửa sổ bên nhận và bên gửi không nhất thiết phải bằng nhau.*
 - *Giá trị các cửa sổ có thể cố định hoặc tăng/ giảm linh hoạt*

Sliding Window Protocols

- Ở máy trạm, thứ tự gói tin được xác định ở tầng giao vận.
- Tất cả các khung ở trong cửa sổ bên gửi phải được lưu để phòng trường hợp cần gửi lại.
 - Nếu kích thước cửa sổ lớn nhất bên gửi là **B**, bên gửi cần bộ nhớ đệm có kích thước tối thiểu là **B**.
 - Nếu **cửa sổ bên gửi** đầy, giao thức cửa sổ trượt phải đóng cửa sổ (ở tầng mạng) cho đến khi bộ nhớ đệm được giải phóng.

Go-back-N



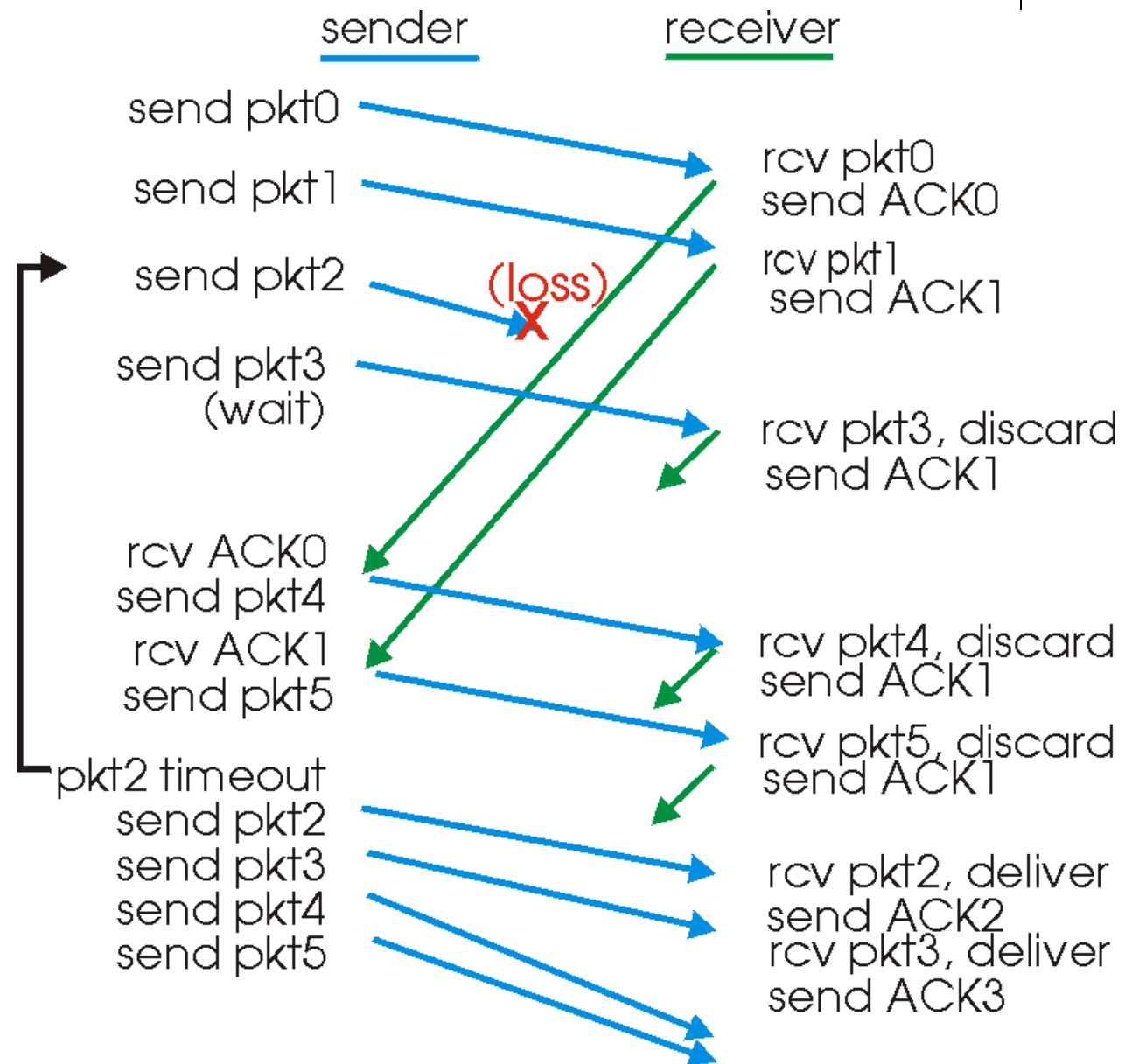
Sender

- Chỉ gửi các gói tin với số hiệu trong cửa sổ, “dịch” cửa sổ sang phải mỗi khi nhận được ACK
- ACK(n): xác nhận cho các gói tin với số hiệu cho đến n
- Khi có timeout: truyền lại tất cả các gói tin có số hiệu lớn hơn n trong cửa sổ.

Receiver

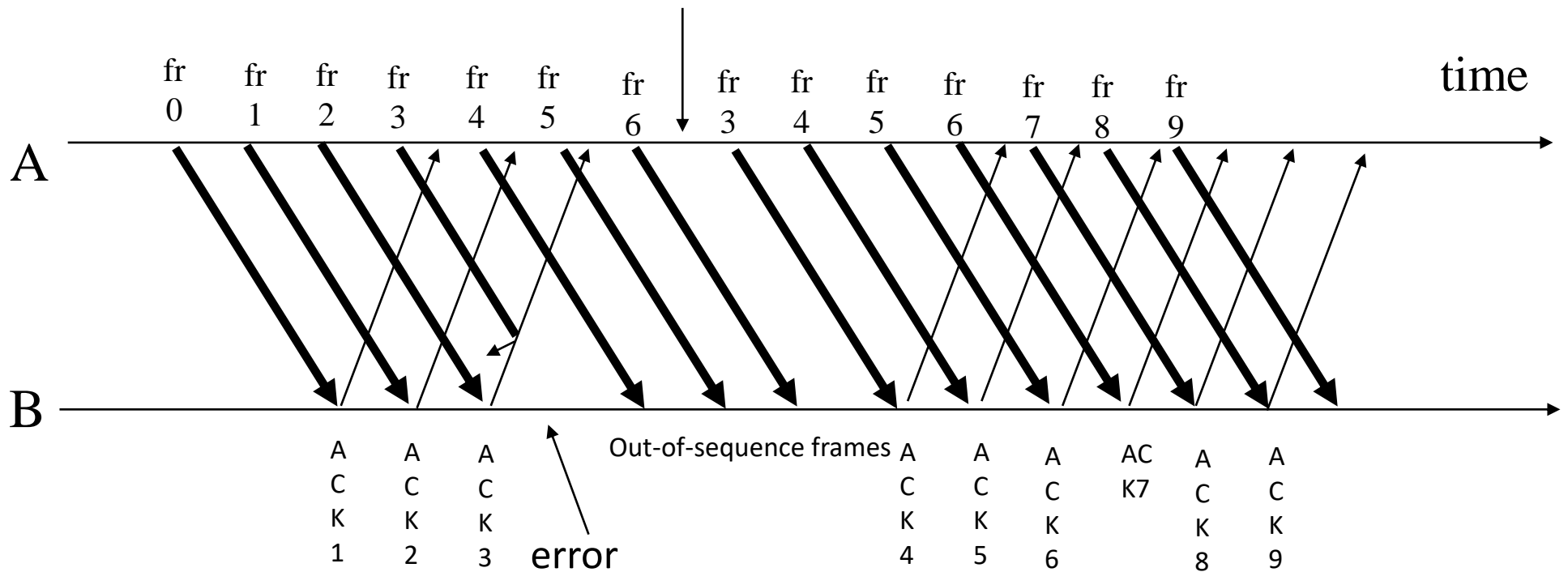
- Chỉ gửi 1 xác nhận ACK cho gói tin có số hiệu lớn nhất đã nhận được theo đúng thứ tự.
- Với các gói tin không theo thứ tự:
 - Hủy bỏ -> không lưu vào vùng đệm
 - Xác nhận lại gói tin với số hiệu lớn nhất còn đúng thứ tự

Ví dụ về GBN



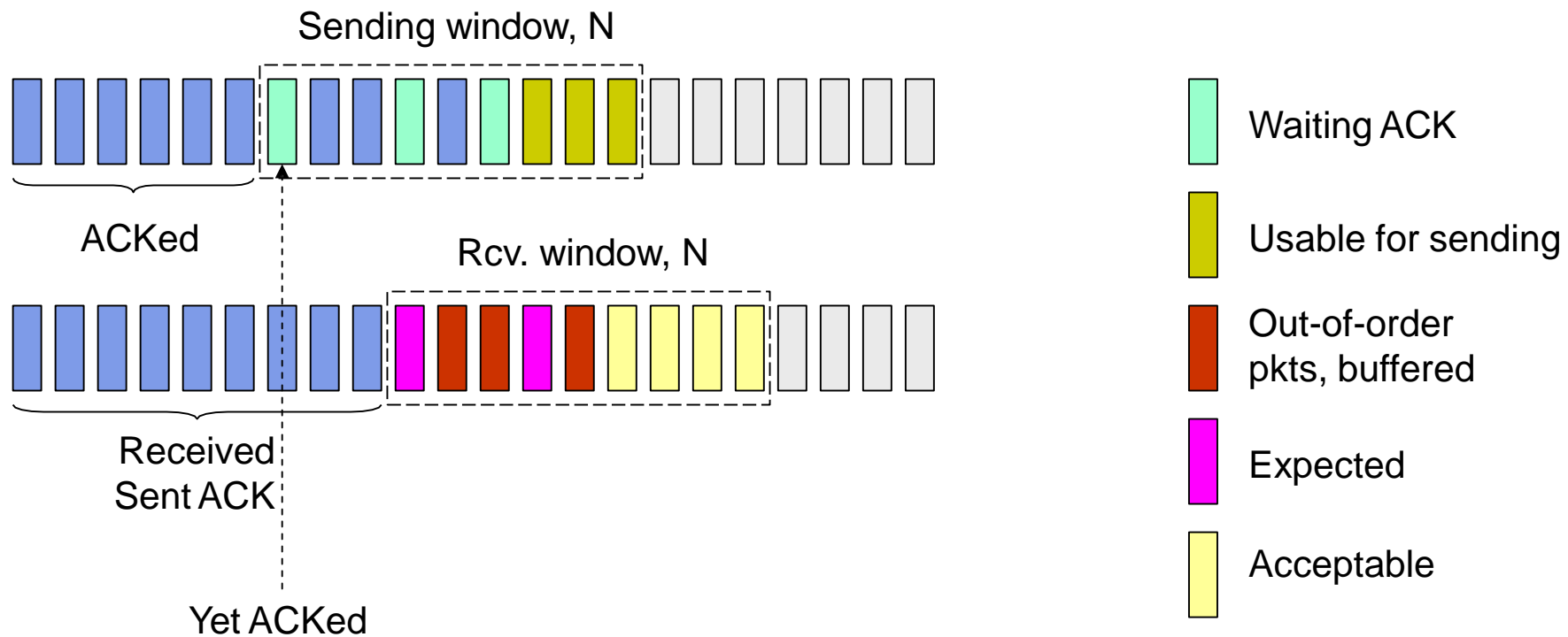
Go Back N

4 frames truyền sai → Go back 4



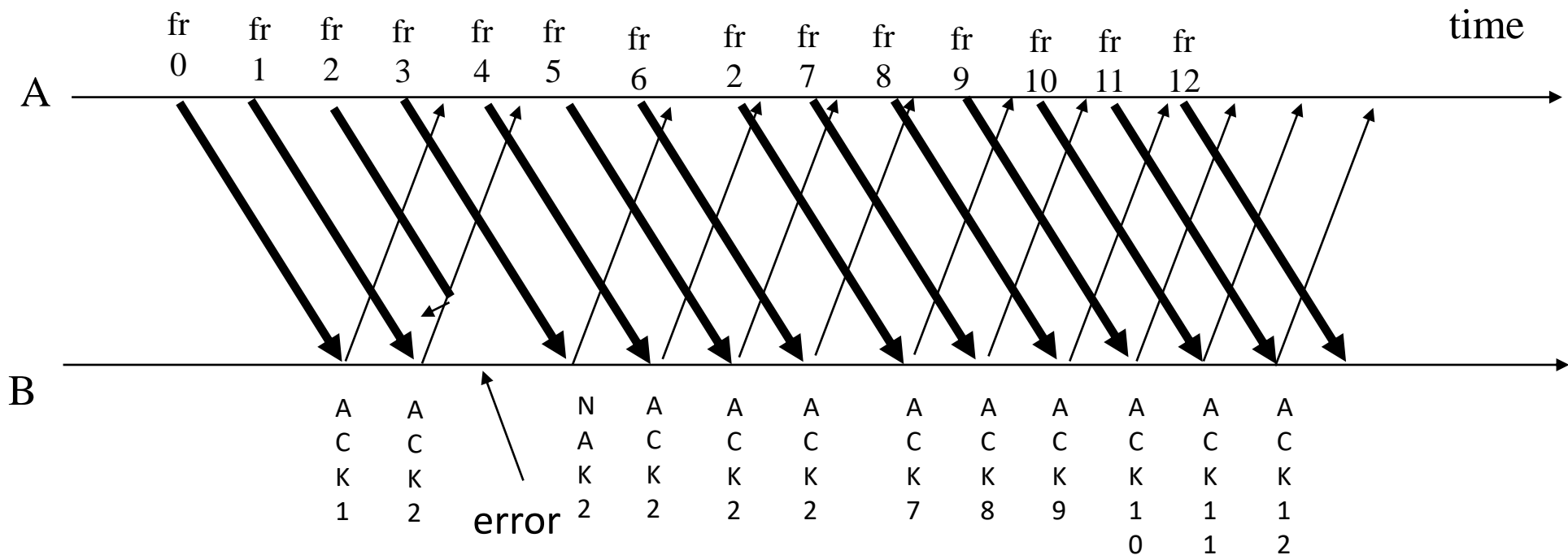
ACKing next frame expected

Selective Repeat

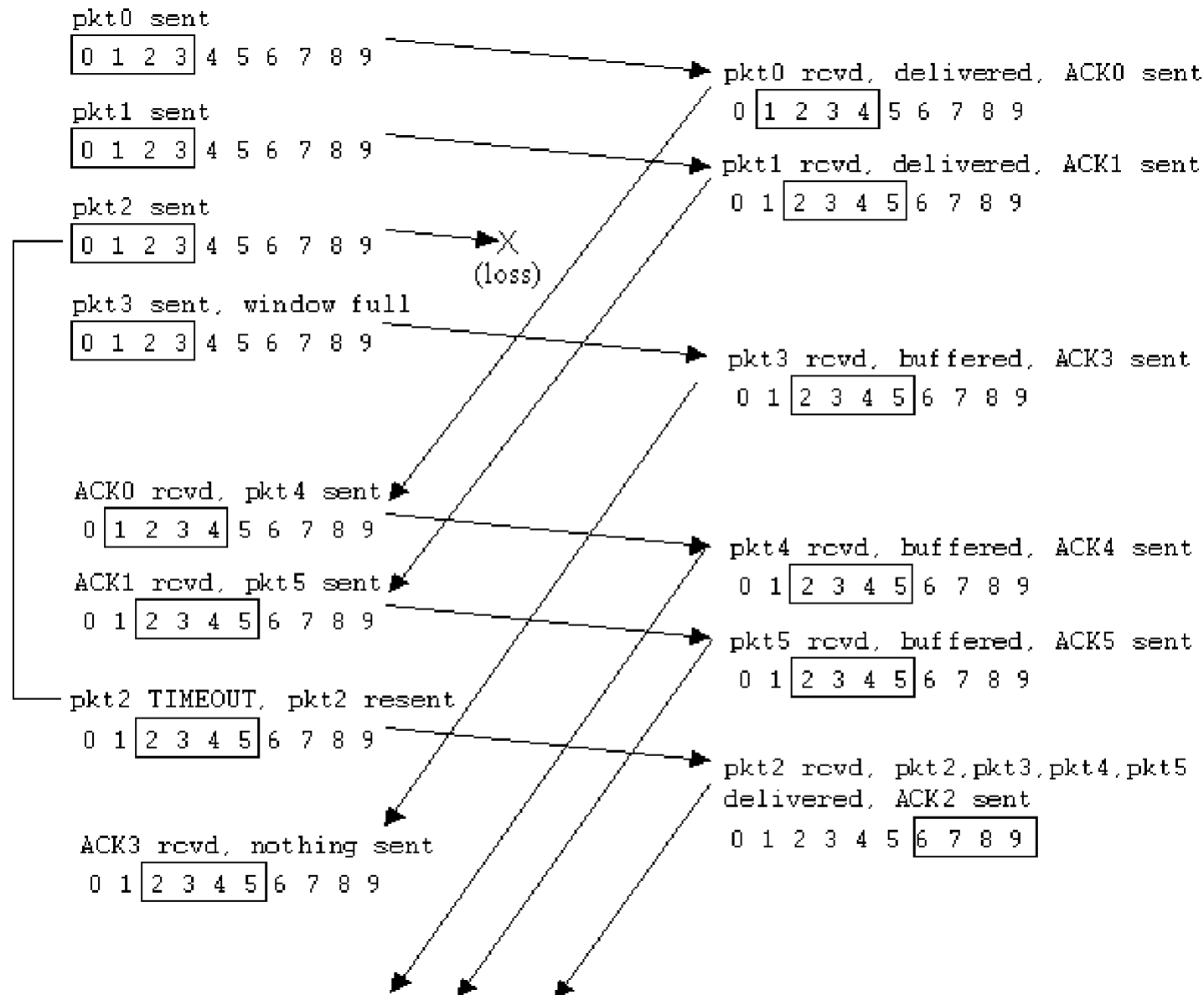


- Bên nhận xác nhận riêng rẽ cho từng gói tin
- Chỉ gửi lại các gói tin chưa được xác nhận bị timeout
- Tổ chức vùng đệm để sắp xếp các gói tin theo đúng thứ tự để chuyển cho tầng trên

Selective Repeat with NAK error recovery



Ví dụ về Selective Repeat



So sánh 2 phương pháp



GO-BACK-N

SELECTIVE REPEAT

Nguyên lý Truyền lại tất cả frame sau frame lớn nhất

Chỉ truyền lại frame được cho là bị mất

Bandwidth Tốn băng thông

Ít tốn băng thông

Complexity Đơn giản

Phức tạp

Window size N-1

$\leq (N+1)/2$

Sorting Không cần sắp xếp ở cả 2 phía truyền và nhận.
Không cần lưu thông tin frames sau frame bị mất

Bên nhận phải sắp xếp được.
Lưu thông tin frame trong buffer

Searching Không có tính năng tìm kiếm

Bên gửi phải có tính năng tìm kiếm