

The Data Link Layer

Chapter 3

3.1 Data Link Layer Design Issues

- Network layer services
- Framing
- Error control
- Flow control

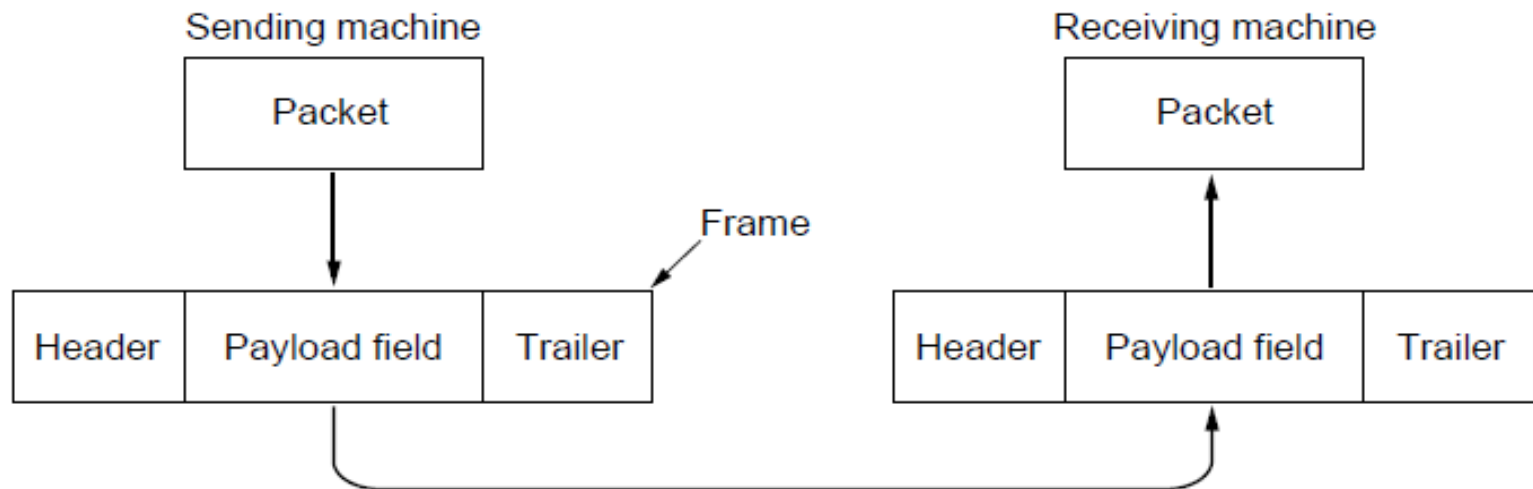
每个帧如果独立确认和必要时重传。
出现差错可以很快的恢复。不可靠的
信道，还是很有必要的。



为啥需要这些功能，
能不能在上层处理？

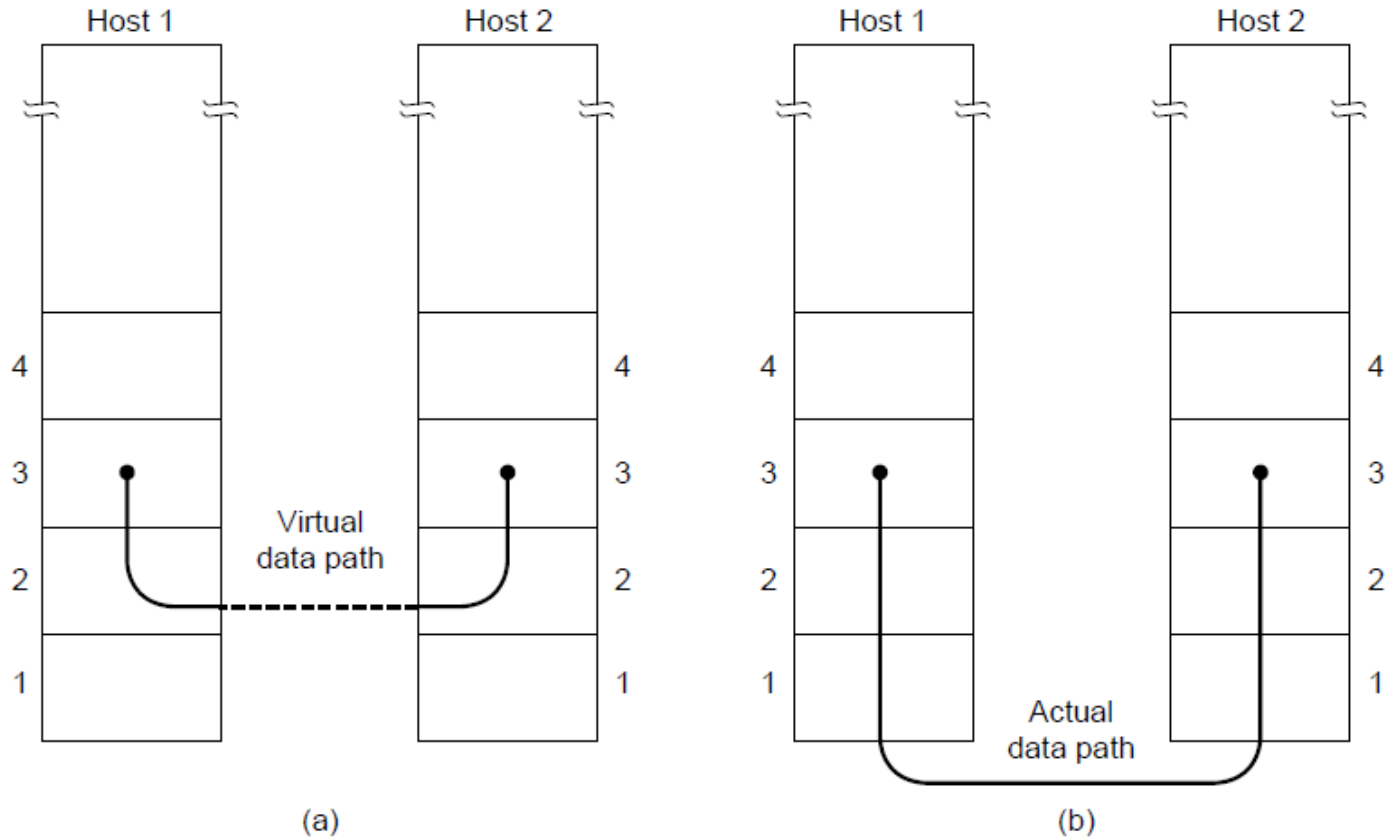
Packets and Frames

- Data link layer takes the packets it gets from the network layer and encapsulates them into **frames** for transmission. Each frame contains a frame header, a payload field for holding the packet, and a frame trailer. Frame management forms the heart of what the data link layer does.



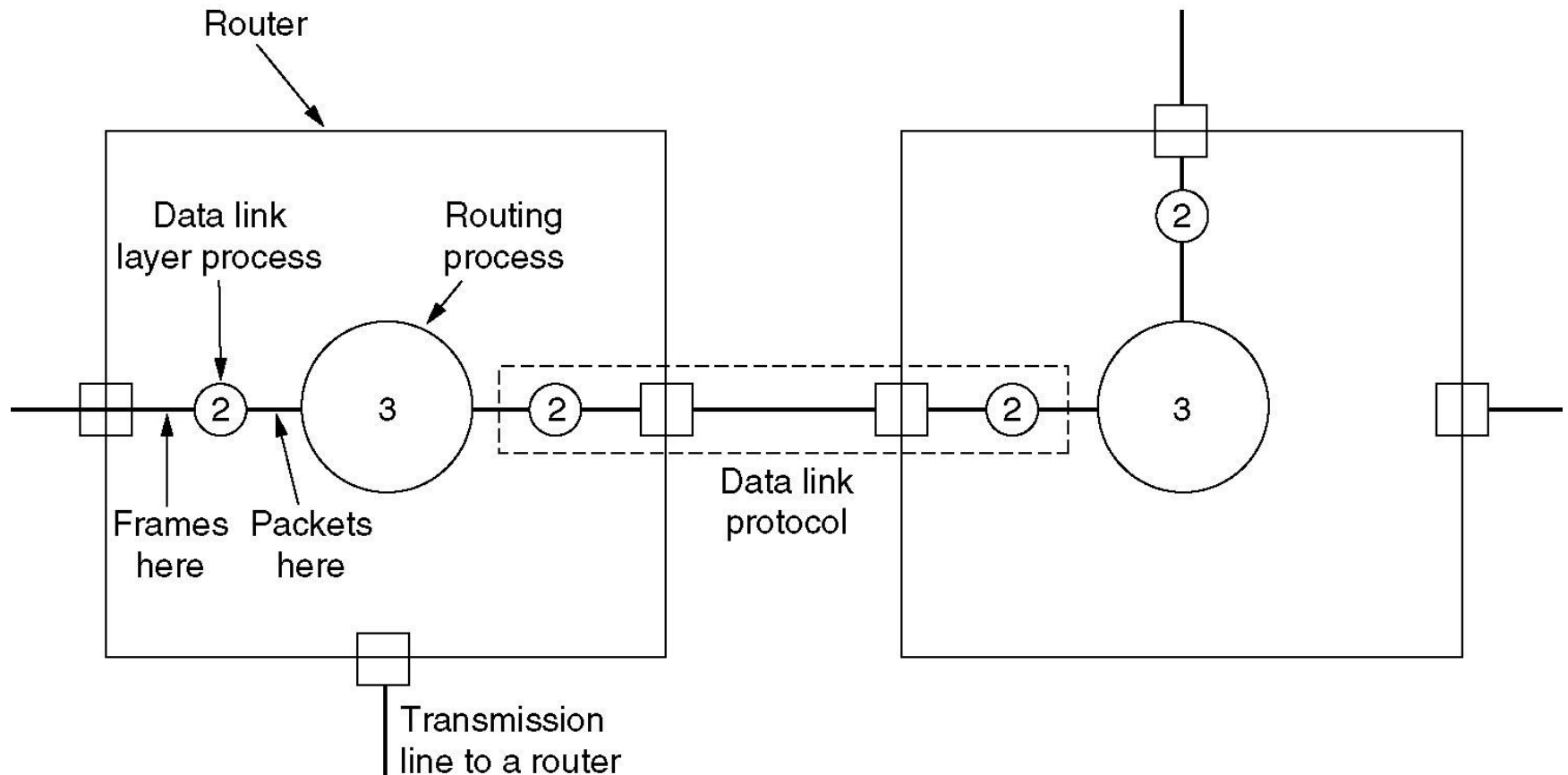
Relationship between packets and frames.

Network Layer Services



(a) Virtual communication. (b) Actual communication.

Network Layer Services(plus)



Possible Services Offered (1)

- **Unacknowledged connectionless service :**

- Source machine sends independent frames to the destination without having the destination acknowledge them. No connection is established beforehand or released afterward. No attempt is made to recover lost data.

适用于误码率很低的线路，错误恢复留给高层；实时业务及大多数局域网采用。

- **Acknowledged connectionless service**

- No logical connection used, each frame sent is individually acknowledged. If a frame has not arrived within a specified time interval, it can be sent again.

适用于不可靠的信道，如无线网。

- **Acknowledged connection-oriented service**

- Transfers go through three distinct phases, each frame sent over

适用于对链路可靠性要求较高的场合。

Possible Services Offered(2)

1. Unacknowledged connectionless service.
2. Acknowledged connectionless service.
3. Acknowledged connection-oriented service.

- 正向应答：只对正确的信息应答。
- 负向应答：只对错误的信息应答。
- 双向应答：既对正确的信息应答，也对错误的信息应答。



在数据传输过程中，数据会出现哪些情况？

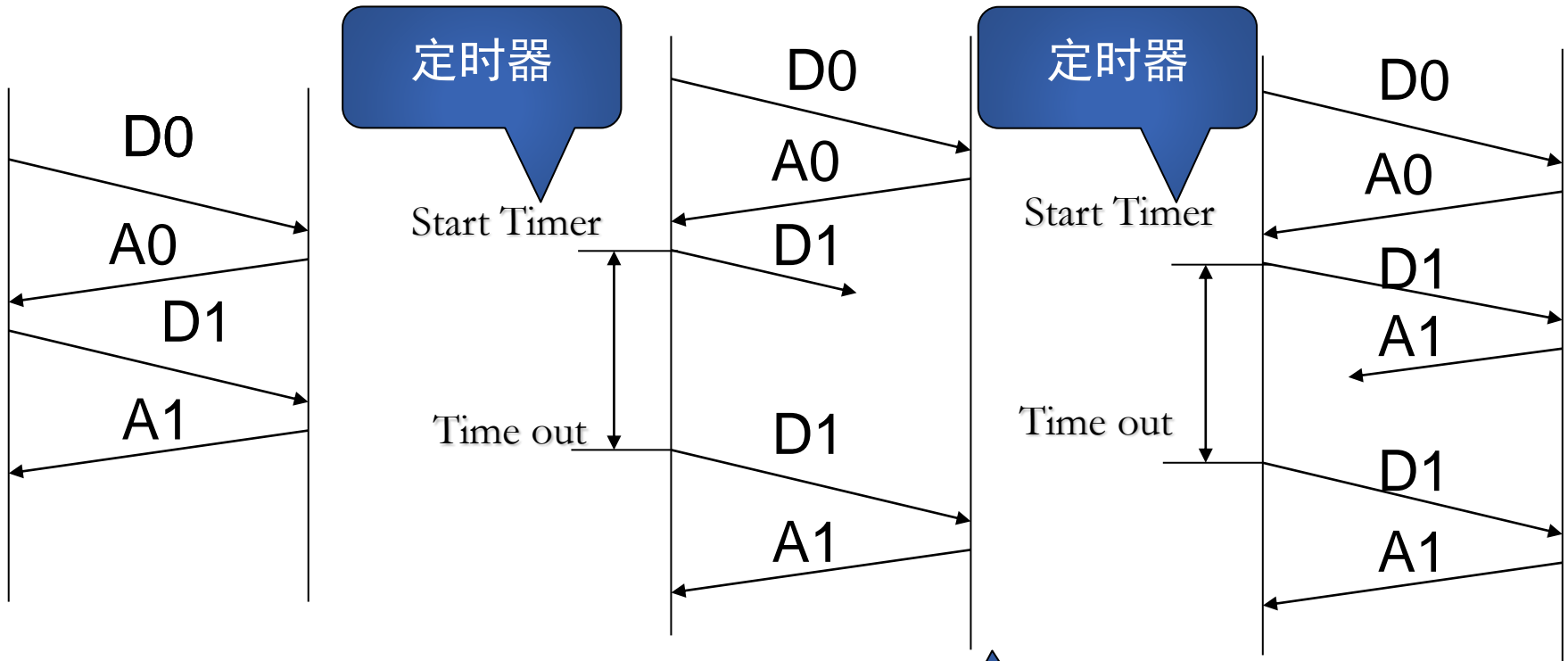
正确、出错、丢失。



在数据链路层，最常采用哪种应答方式？

正向应答

Possible Services Offered(3)



(1) Normal

(2) data loss or error

(3) Ack loss

超时时间, 序号等问题

Framing Methods

1. Byte count.
2. Flag bytes with byte stuffing.
3. Flag bits with bit stuffing.
4. Physical layer coding violations.

Framing Methods

- The physical layer doesn't do much: it just pumps bits from one end to the other. But things may go wrong, that is the data link layer needs a means to do retransmissions. The unit of retransmission is a **frame** (which is just a fixed number of bits).
- **Problem:** How can we break up a bit stream into frames?

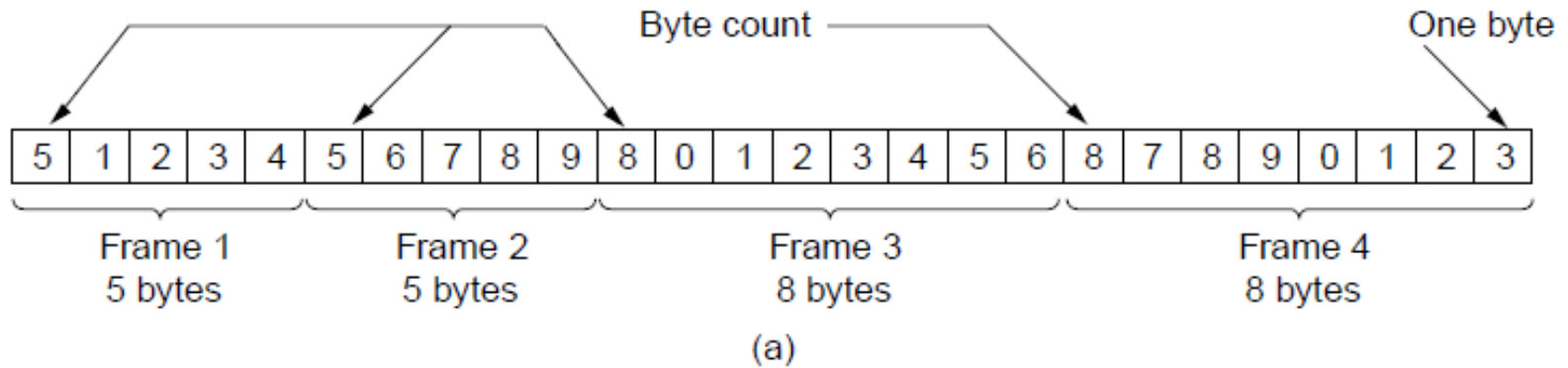


构造的原则？

易区分，占用的
信道带宽少

Byte Count

字节计数法



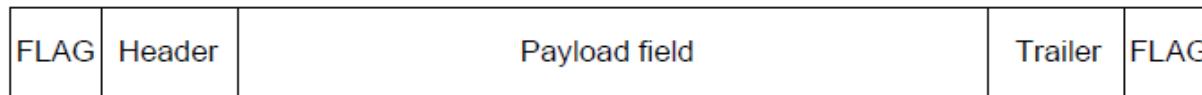
Problem?

不是独立定界，一个错都错

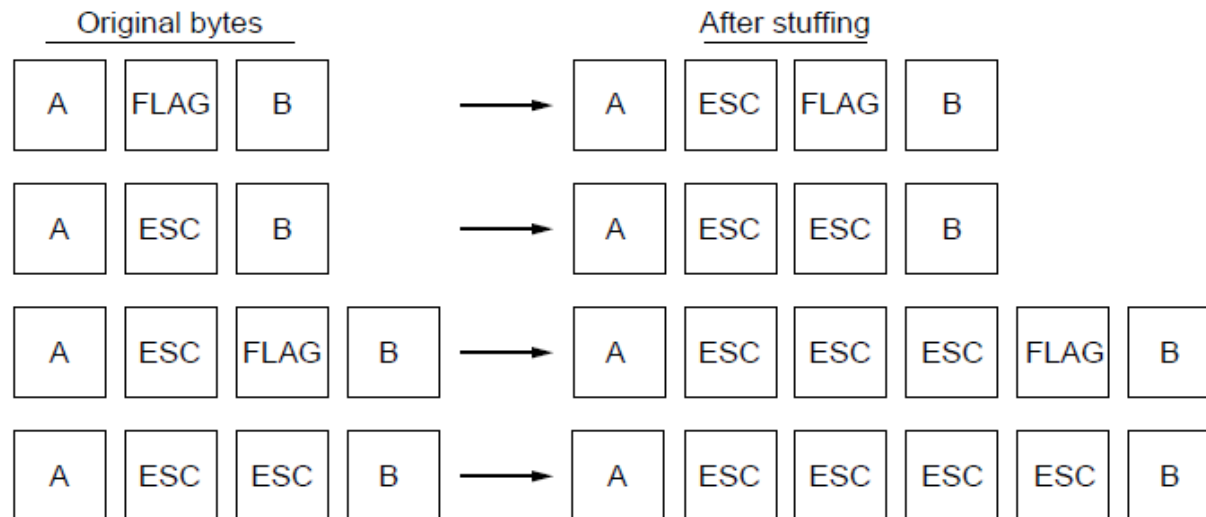
A byte stream. (a) Without errors. (b) With one error.

Flag byte with byte stuffing

- Mark the beginning and end of a frame with a **flag byte** – a special byte or character. Insert a special escape byte (ESC) just before each “accidental” flag byte in the data.



(a)



(b)

A frame delimited by flag bytes.

Four examples of byte sequences before and after byte stuffing.

Flag byte with byte stuffing

用DLE STX标示
帧的开始
用DLE ETX标示
帧的结束

标准表

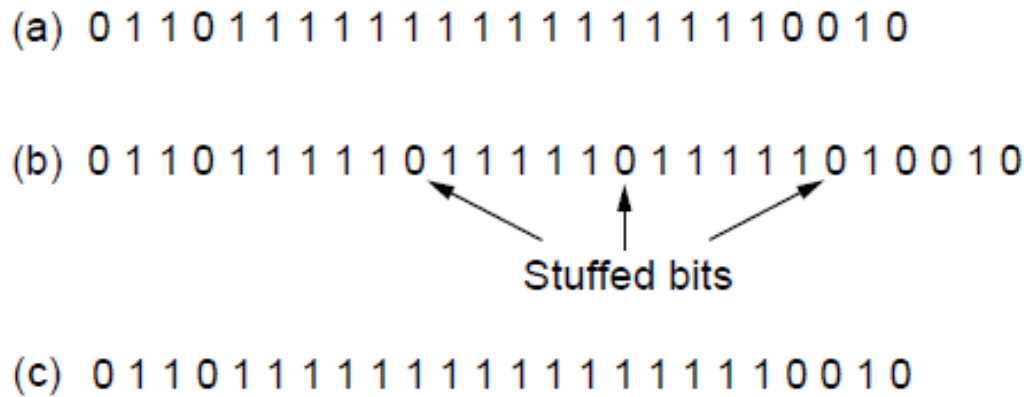
Bin(二进制)	Oct(八进制)	Dec(十进制)	Hex(十六进制)	缩写/字符	解释
0000 0000	0	0	00	NUL(null)	空字符
0000 0001	1	1	01	SOH(start of headline)	标题开始
0000 0010	2	2	02	STX (start of text)	正文开始
0000 0011	3	3	03	ETX (end of text)	正文结束
0000 0100	4	4	04	EOT (end of transmission)	传输结束
0000 0101	5	5	05	ENQ (enquiry)	请求
0000 0110	6	6	06	ACK (acknowledge)	收到通知
0000 0111	7	7	07	BEL (bell)	响铃
0000 1000	10	8	08	BS (backspace)	退格
0000 1001	11	9	09	HT (horizontal tab)	水平制表符
0000 1010	12	10	0A	LF (NL line feed, new line)	换行键
0000 1011	13	11	0B	VT (vertical tab)	垂直制表符
0000 1100	14	12	0C	FF (NP form feed, new page)	换页键
0000 1101	15	13	0D	CR (carriage return)	回车键
0000 1110	16	14	0E	SO (shift out)	不用切换
0000 1111	17	15	0F	SI (shift in)	启用切换
0001 0000	20	16	10	DLE (data link escape)	数据链路转义

- 每帧单独定界，每帧定界的错误不会传递给后续帧
- 局限于8位字符和ASCII字符传送，也不被普遍采用。
 - ✓ 字符串，信道利用率
 - ✓ 监视着位串，处理复杂，可能要修改有效载荷
 - ✓ 使用相同的字符集

Starting and ending flags with bit stuffing

含位填充的分界标志法

- Mark the beginning and end of a frame with a special bit sequence (01111110). Escape the flag byte through an additional bit “0”:



Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

Physical layer coding violations

物理层编码违例法

- Manchester encoding:
 - HL represents “1”, LH represents “0”
 - HH and LL are coding violations and can be used for flags
- 4B/5B



- 只有有冗余的编码才能使用
- 不需要填充，信道利用率高

实际情况，往往不单独使用
以太网，前导码，帧间隙，帧长字段

Error Control

- Ensuring all frames are eventually delivered:
 - To the network layer at the destination
 - In the proper order
- Ensures reliable, connection-oriented service
- Requires acknowledgement frames and timers

Flow Control

流量控制：收发方不匹配。不同层次视角不同，但是策略相当。

- Controlling the sending of transmission frames at a faster pace than they can be accepted
- Feedback-based flow control
 - Receiver sends back information to the sender giving it permission to send more data
 - Or receiver tells the sender how the receiver is doing
- Rate-based flow control
 - Protocol has a built-in mechanism
 - Mechanism limits the rate at which senders may transmit data
 - No feedback from the receiver is necessary

3.2 Error Detection and Correction

- 自动纠错机制 (FEC)
Forward Error Correction
- 检错反馈重发机制 (ARQ)
Automatic Repeat-reQuest

- Error-correcting codes

- Referred to as FEC (Forward Error Correction)
- Include enough redundant information to enable the receiver to deduce what the transmitted data must have been

- Error-detecting codes

- Include only enough redundancy to allow the receiver to deduce that an error has occurred (but not which error) and have it request a retransmission

随机，连续突发 (burst)；出错、丢失

- Key consideration is the type of errors likely to occur

- 信道的电气特性引起信号幅度、频率、相位的畸变；
- 信号反射；串扰；
- 闪电、大功率电机的启停等。

Error Correcting Codes (1)

1. **Hamming codes.**
2. Binary convolutional codes.
3. Reed-Solomon codes.
4. Low-Density Parity Check codes.

Error Correcting Codes (2)

- A frame consists of m data (i.e., message) bits and r redundant, or check, bits. Let the total length be n (i.e., $n = m + r$). An n -bit unit containing data and check bits is often referred to as an n -bit **codeword**.

- **Definition:** The **Hamming distance** between two frames a and b is the number of bits at the same position that differ.

1	0	0	0	1	0	0	1
1	0	1	1	0	0	0	1
<hr/>							
0	0	1	1	1	0	0	0

Hamming distance 3



Error Correcting Codes (2)

- To *detect* all sets of d or fewer errors, it is necessary and sufficient that the Hamming distance between any two frames is $d+1$ or more.

d个1位错误不会让他变成另一个合法的信息



- To *correct* all sets of d or fewer errors, it is necessary that the Hamming distance between any two frames is $2d+1$ or more.

d个1位错误还是离原来的近，可以知道

```
0000000000
0000011111
1111100000
1111111111
```

- Parity Check:** Add a bit to a bit string such that the total number of 1-bits is even (or odd), that the distance between all frames is at least 2.
- Conclusion:** We can *detect* a single error by this.

Hamming Code

- Imagine that we want to design a code with m message bits and r check bits that will allow all **single errors** to be corrected.

Every bit at position 2^k , $k \geq 0$ is used as a parity bit for those positions to which it contributes:

	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11
1	X		X		X		X		X		X
2		X	X			X	X			X	X
4				X	X	X	X				
8								X	X	X	X

So that 1100001 is encoded as (check bits are in boldface)

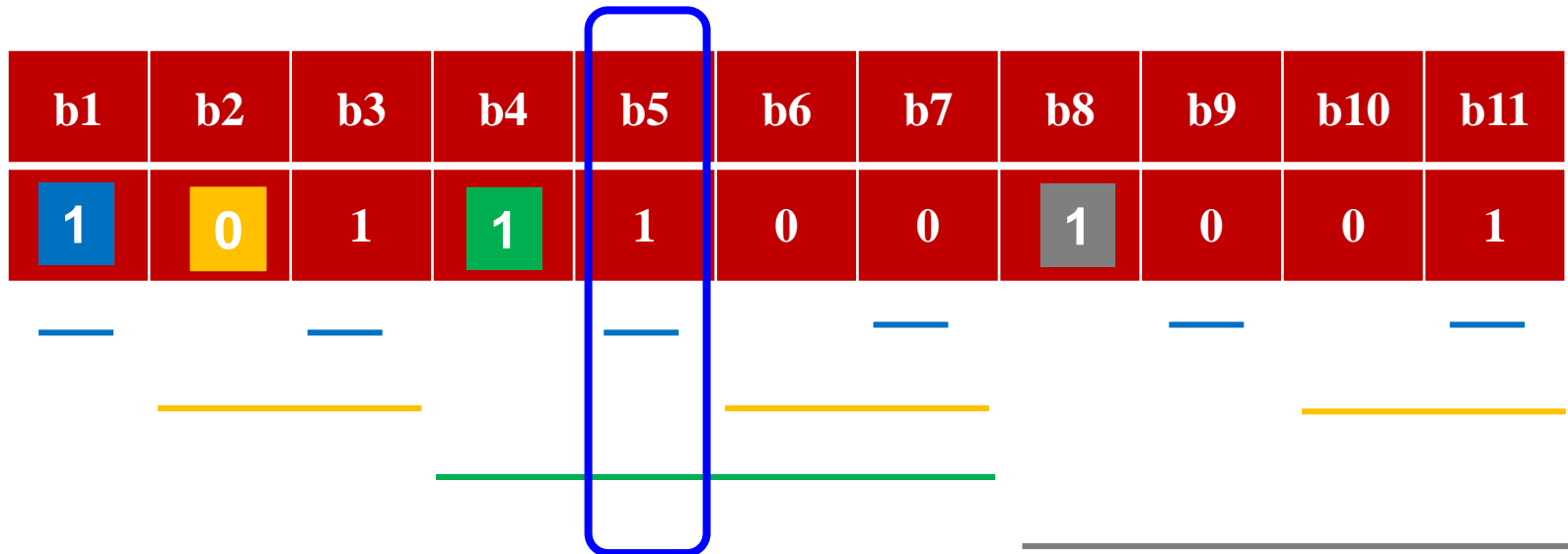
	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11
1	X		X		X		X		X		X
2		X	X			X	X			X	X
4				X	X	X	X				
8								X	X	X	X
	1	0	1	1	1	0	0	1	0	0	1

Hamming Code

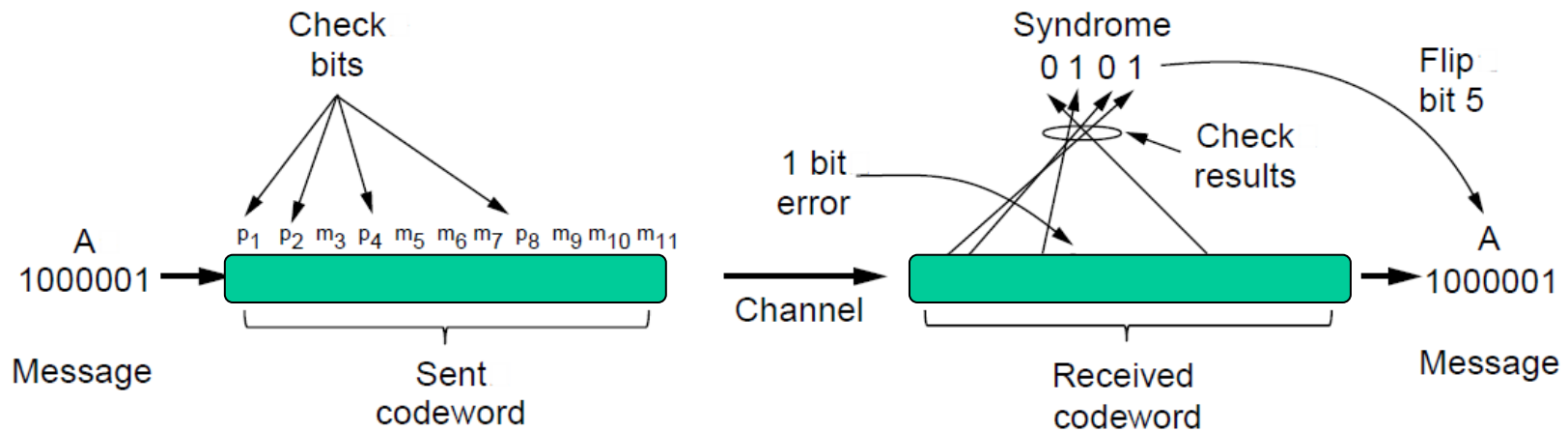
- Imagine that we want to design a code with m message bits and r check bits that will allow all **single errors** to be corrected.

So that 1100001 is encoded as (check bits are in boldface)

	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11
1	X		X		X		X		X		X
2		X	X			X	X			X	X
4				X	X	X	X				
8								X	X	X	X
	1	0	1	1	1	0	0	1	0	0	1



Error Correcting Codes (2)



Example of an (11, 7) Hamming code
correcting a single-bit error.

Error-Detecting Codes (1)

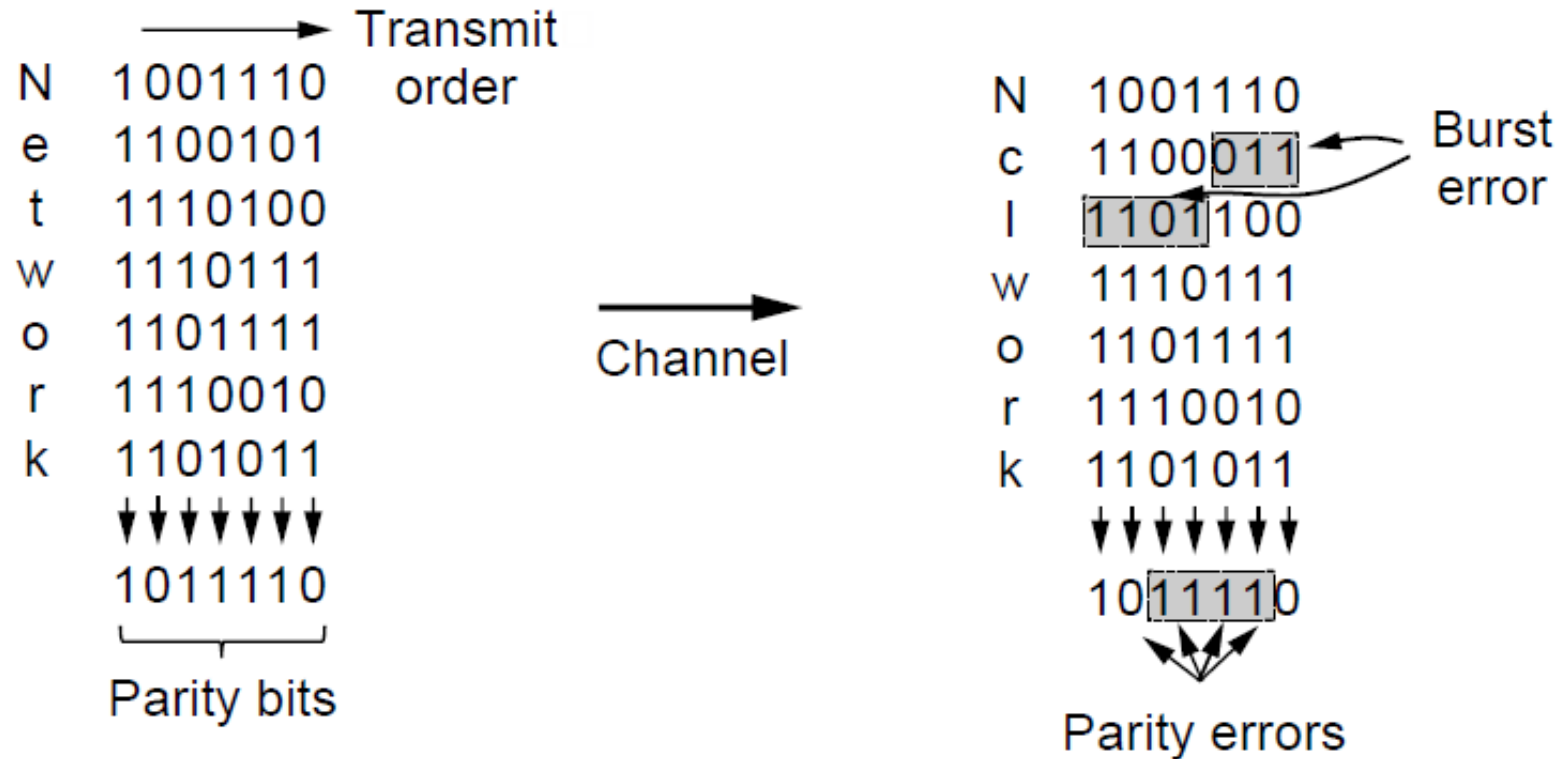
Problem: Error correcting codes are simply too expensive, so we only use error detection combined with retransmissions.

Linear, systematic block codes

1. **Parity.**
2. **Checksums.**
3. **Cyclic Redundancy Checks (CRCs).**

一组校验位。e. g. 一组奇偶校验位。
对码字求和检测错误。

Error-Detecting Codes (2)



Interleaving of parity bits to detect a burst error.

Cyclic Redundancy Checks (CRCs).

将位串看成系数为0
或者1的多项式

All arithmetic operation is done modulo 2, no carries
for addition or borrows for subtraction:

EXCLUSIVE OR.

Ex: $10011011 + 11001010 = 01010001$

$01010101 - 10101111 = 11111010$

For division, is done at same length.

模2算法：加法不进
位，减法不借位

Error-Detecting Codes (3)

- $G(x)$ is an irreducible polynomial with both the high- and low- order bits of the generator is 1. Compute the checksum by divide $G(x)$, and then append the checksum to the end of the frame. Receiver divides the frame by $G(x)$ again to see if there is a remainder.

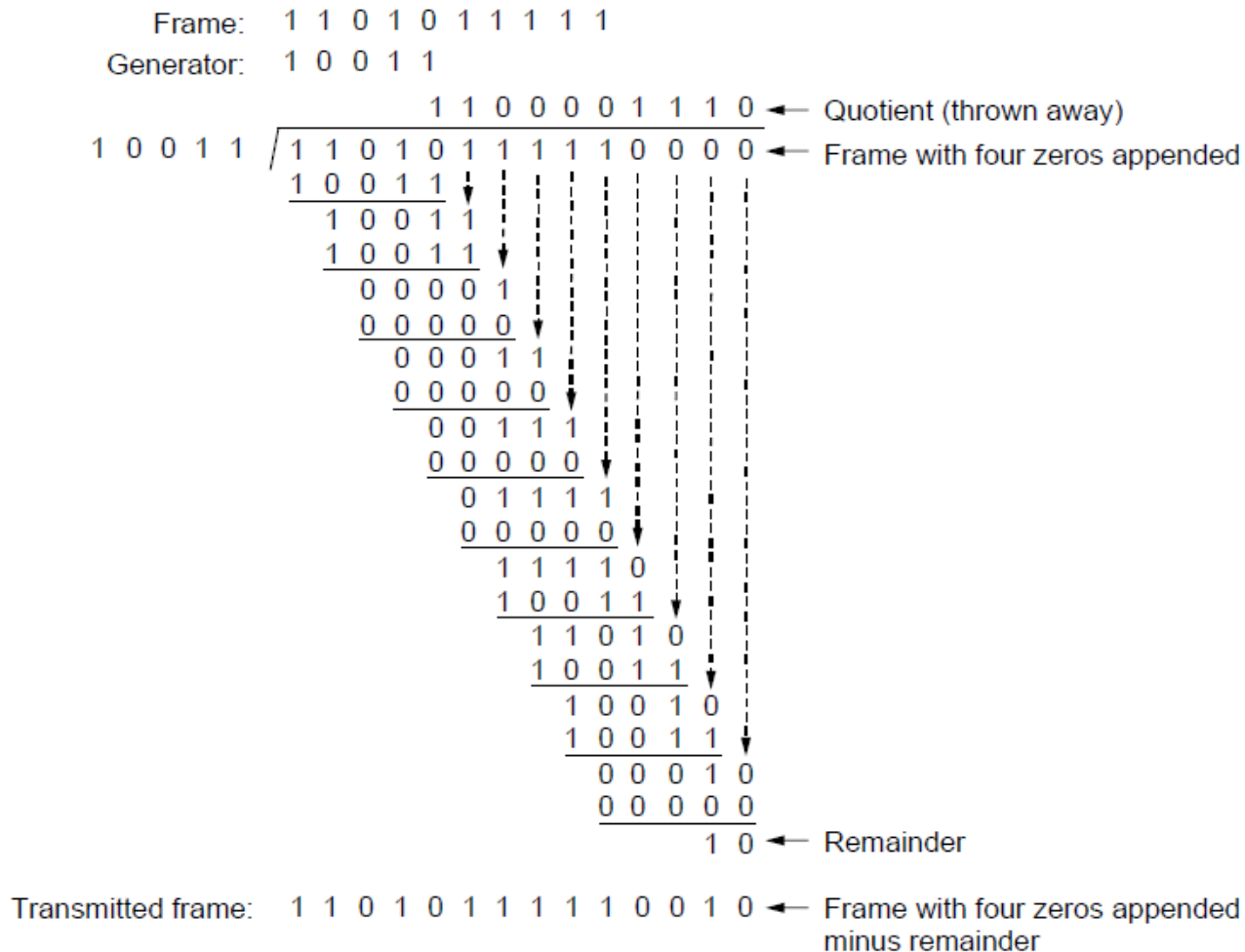
- 生成多项式 $G(x)$

- 发方、收方事前商定；
- 生成多项式的高位和低位必须为1
- 生成多项式必须比传输信息对应的多项式短。

- CRC码基本思想：

- 校验和（checksum）加在帧尾，使带校验和的帧的多项式能被 $G(x)$ 除尽；
收方接收时，用 $G(x)$ 去除它，若有余数，则传输出错。

Error-Detecting Codes (3)



Example calculation of the CRC

International CRC Standards

- CRC-12 $=x^{12}+x^{11}+x^3+x^2+x^1+1$
- CRC-16 $=x^{16}+x^{15}+x^2+1$
- CRC-CCITT $=x^{16}+x^{12}+x^5+1$
- CRC-32 $=x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$



IEEE 802使用



- 要发送的数据是101110, 生成多项式是 $G(x) = x^3 + 1$, 添加的余数是?



失效?

• CRC的检错能力

- 发送: $T(x)$; 接收: $T(x) + E(x)$;
- 余数 $((T(x) + E(x)) / G(x)) = 0 + \text{余数}(E(x) / G(x))$
- 若 余数 $(E(x) / G(x)) = 0$, 则差错不能发现; 否则, 可以发现。

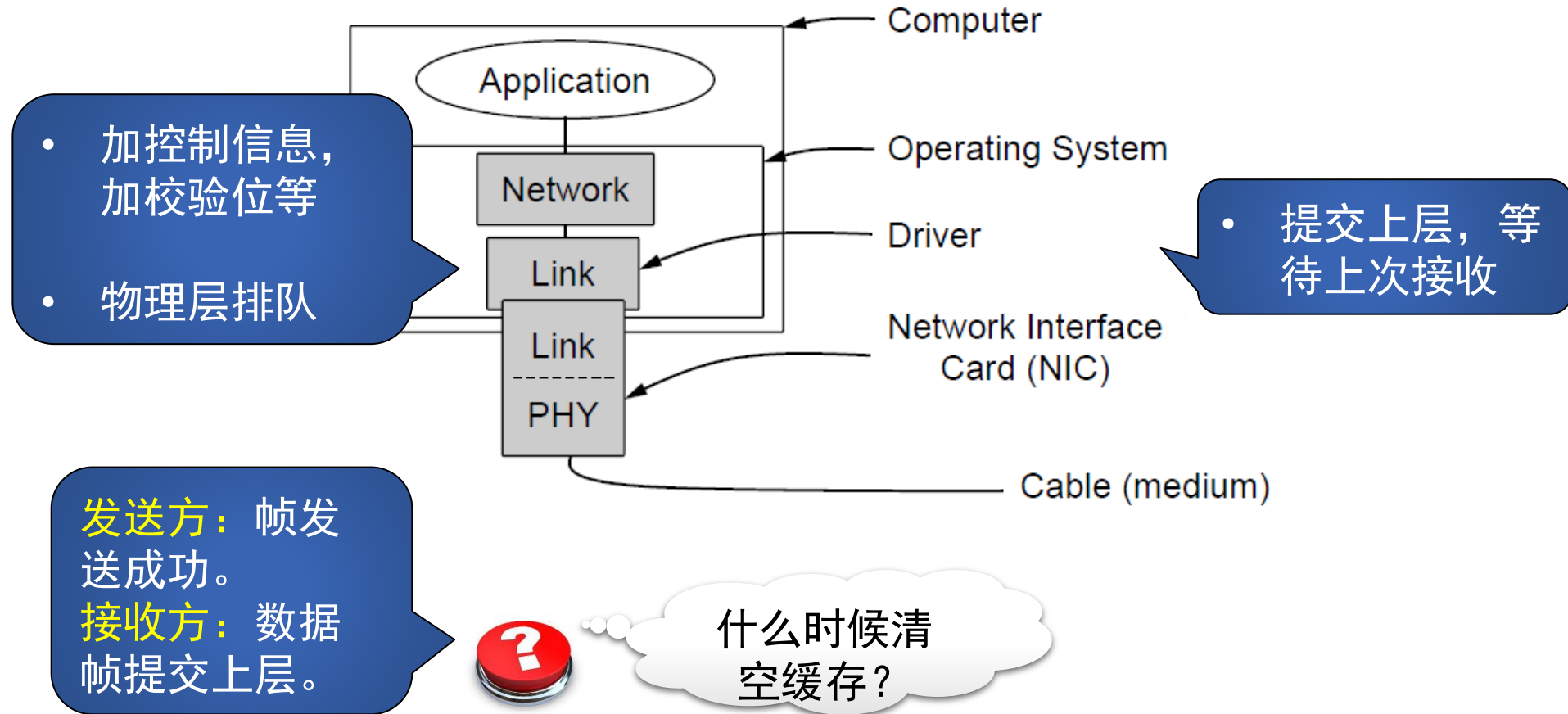
3.3 Elementary Data Link Protocols (1)

- Assumptions underly the communication model
- Three simplex link-layer protocols
 - Utopia: No Flow Control or Error Correction
 - Adding Flow Control: Stop-and-Wait
 - Adding Error Correction: Sequence Numbers and ARQ

Initial Simplifying Assumptions (1 of 2)

- Independent processes
 - Physical, data link, and network layers are independent
 - Communicate by passing messages back and forth
- Unidirectional communication
 - Machines use a reliable, connection-oriented service
- Reliable machines and processes
 - Machines do not crash

Initial Simplifying Assumptions (2of 2)



Implementation of the physical, data link, and network layers.

Basic Transmission And Receipt (1 of 3)

```
#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 frame is it? */
    seq_nr seq;                                       /* sequence number */
    seq_nr ack;                                       /* acknowledgement number */
    packet info;                                      /* the network layer packet */
} frame;
. . .
```

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h*.

Basic Transmission And Receipt (2 of 3)

```
/* 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);    . . .
```

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h*.

Basic Transmission And Receipt (3of 3)

```
/* 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
```

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h*.

Utopian Simplex Protocol (1)

/* 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 */  
}
```

...

无错、理想信道
下的单工协议

A utopian simplex protocol.

Utopian Simplex Protocol (2)

```
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 */
    }
}
```

A utopian simplex protocol.

Utopian Simplex Protocol (3)

- 《乌托邦》全名是《关于最完美的国家制度和乌托邦新岛的既有益又有趣的金书》
- 用拉丁语写成。书中叙述一个虚构的航海家航行到一个奇乡异国乌托邦的旅行见闻。“乌托邦”一词来自希腊文，意即“乌有之乡”。莫尔第一次用它来表示一个幸福的、理想的国家。



Thomas More

Simplex Stop-and-Wait Protocol (1)

/* 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;  
    packet buffer;  
    event_type event;
```

```
    while (true) {  
        from_network_layer(&buffer);  
        s.info = buffer;  
        to_physical_layer(&s);  
        wait_for_event(&event);
```

```
    }  
}
```

...

无错信道下的单工
停等协议

```
/* buffer for an outbound frame */
```

```
/* buffer for an outbound packet */
```

```
/* frame_arrival is the only possibility */
```

```
/* go get something to send */
```

```
/* copy it into s for transmission */
```

```
/* bye-bye little frame */
```

```
/* do not proceed until given the go ahead */
```

A simplex stop-and-wait protocol.

Simplex Stop-and-Wait Protocol (2)



应答的含义？

等接收方确认，流量控制，无错无序号。

```
void receiver2(void)
{
    frame r, s;
    event_type event;
    while (true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
        to_physical_layer(&s);
    }
}
```

/* buffers for frames */
/* frame_arrival is the only possibility */

/* only possibility is frame_arrival */
/* go get the inbound frame */
/* pass the data to the network layer */
/* send a dummy frame to awaken sender */


数据流量是单工的，物理信道需要具备双向通信能力，是半双工的即可

A simplex stop-and-wait protocol.

Positive Acknowledgement with Retransmission Protocols (1)

Protocol 3: Let's drop the assumption that the channel is **error free**. We do assume that damaged frames can be detected, but also that frames can get lost entirely.

有错信道下的单工
停等协议



• **Problem 1:** A sender doesn't know whether a frame has made it (correctly) to the receiver.

正向应答

– **Solution:** Let the receiver acknowledge undamaged frames.

Positive Acknowledgement with Retransmission Protocols(2)

 **Problem 2:** Acknowledgments may get lost.

- **Solution:** let the sender use a timer by which it simply retransmits unacknowledged frames after some time.

ARQ, Automatic Retransmit reQuest.

定时、重传

Positive Acknowledgement with Retransmission Protocols(3)



Problem 3: The receiver cannot distinguish duplicate transmissions.

- **Solution:** use sequence numbers

序列号



Problem 4: Sequence numbers cannot go on forever. .

- **Solution:** We can (and need) only use a few of them.

In our example, we need only two (0 & 1).

循环序号

Positive Acknowledgement with Retransmission Protocols(4)

```
/* 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;  
    . . .  
}
```

有错信道下的单工
停等协议

A positive acknowledgement with retransmission protocol.

Positive Acknowledgement with Retransmission Protocols (5)

```
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 */
        }
    }
}
}
```

...

A positive acknowledgement with retransmission protocol.

Positive Acknowledgement with Retransmission Protocols(6)

```
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);  
        }  
    }  
}
```



应答的含义？

- 数据正确到达
- 可以接收

```
/* 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 */
```

A positive acknowledgement with retransmission protocol.

3.4 Improving Efficiency

- Need bidirectional data transmission
- Link layer efficiency improvement
 - Send multiple frames simultaneously before receiving an acknowledgement

From simplex to duplex

- Use more realistic Two-way communication
 - use the same circuit for data in both directions
 - the reverse channel has the same capacity as the forward channel.

Bidirectional Transmission, Multiple Frames in Flight (1 of 3)

- Bidirectional transmission: piggybacking
 - Use the same link for data in both directions
 - Interleave data and control (ACK) frames on the same link
 - Temporarily delay outgoing acknowledgements so they can be hooked onto the next outgoing data frame
- Piggybacking advantages
 - A better use of the available channel bandwidth
 - Lighter processing load at the receiver
- Piggybacking issue
 - Determining time data link layer waits for a packet to piggyback the acknowledgement



坏处

Bidirectional Transmission, Multiple Frames in Flight (2 of 3)



停等协议的发送窗口大小？

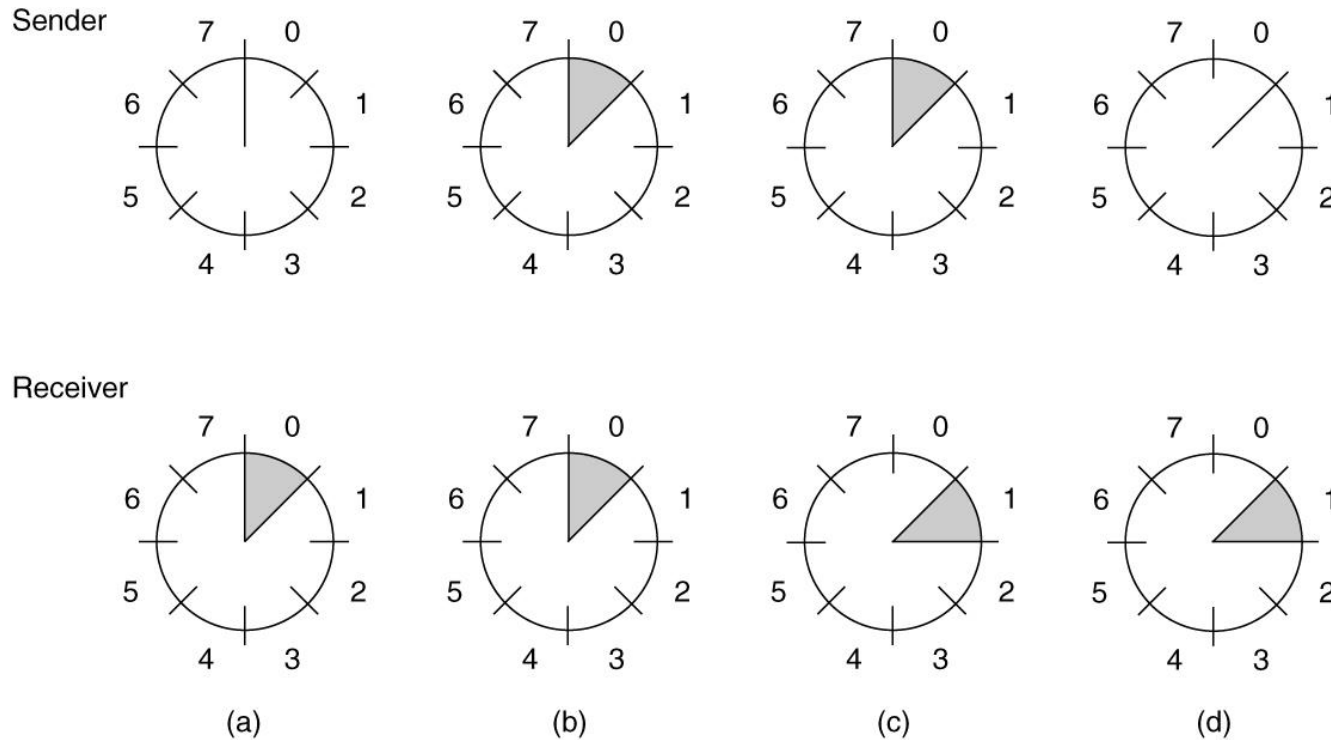
- Three bidirectional sliding window protocols
 - One-bit sliding window, go-back-n, selective repeat
- Consider any instant of time
 - Sender maintains a set of sequence numbers corresponding to frames it is permitted to send
 - Frames are said to fall within the sending window
 - Receiver maintains a receiving window corresponding to the set of frames it is permitted to accept
- Differ among themselves in terms of efficiency, complexity, and buffer requirements

发送窗口：允许连续发送未应答帧的序号

接收窗口：允许连续接收未处理帧的序号

- 确定能存几个后续帧；
- 区分后序帧，重复帧。

Bidirectional Transmission, Multiple Frames in Flight (3 of 3)



A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

One-Bit Sliding Window Protocol (1)

- **Protocol 4:** A stop-and-wait sliding window protocol with a maximum window size of 1. The sender transmits a frame and waits for its acknowledgment before sending the next one.

One-Bit Sliding Window Protocol (1)

```
/* Protocol 4 (Sliding window) is bidirectional. */
```

```
#define MAX_SEQ 1
```

```
/* must be 1 for protocol 4 */
```

```
typedef enum {frame_arrival, c
```

```
#include "protocol.h"
```

```
void protocol4 (void)
```

```
{
```

```
    seq_nr next_frame_to_send;
```

```
/* 0 or 1 only */
```

```
    seq_nr frame_expected;
```

```
/* 0 or 1 only */
```

```
    frame r, s;
```

```
/* scratch variables */
```

```
    packet buffer;
```

```
/* current packet being sent */
```

```
    event_type event;
```

```
    next_frame_to_send = 0;
```

```
/* next frame on the outbound stream */
```

```
    frame_expected = 0;
```

```
/* frame expected next */
```

```
    from_network_layer(&buffer);
```

```
/* fetch a packet from the network layer */
```

```
    s.info = buffer;
```

```
/* prepare to send the initial frame */
```

```
    s.seq = next_frame_to_send;
```

```
/* insert sequence number into frame */
```

```
    s.ack = 1 - frame_expected;
```

```
/* piggybacked ack */
```

```
    to_physical_layer(&s);
```

```
/* transmit the frame */
```

```
    start_timer(s.seq);
```

```
/* start the timer running */
```

```
    . . .
```

有错信道下的双工
停等协议

A 1-bit sliding window protocol.

One-Bit Sliding Window Protocol (2)

```
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);  
        }  
        if (r.ack == next_frame_to_send) {  
            stop_timer(r.ack);  
            from_network_layer(&buffer);  
            inc(next_frame_to_send);  
        }  
    }  
}
```

...

A 1-bit sliding window protocol.

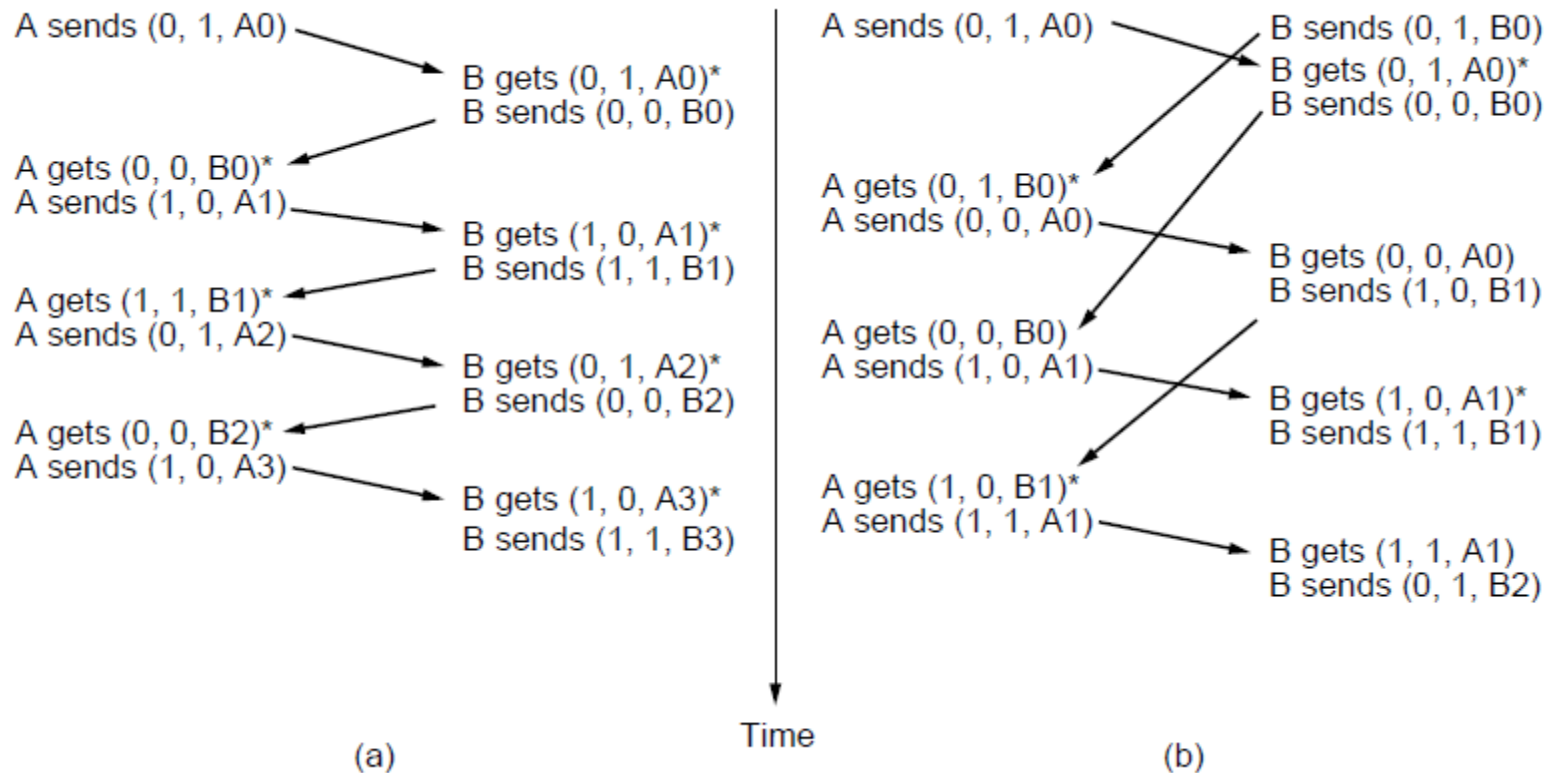
One-Bit Sliding Window Protocol (3)

```
s.info = buffer;
s.seq = next_frame_to_send;
s.ack = 1 - frame_expected;
to_physical_layer(&s);
start_timer(s.seq);
}
}
```

```
/* construct outbound frame */
/* insert sequence number into it */
/* seq number of last received frame */
/* transmit a frame */
/* start the timer running */
```

A 1-bit sliding window protocol.

One-Bit Sliding Window Protocol (4)

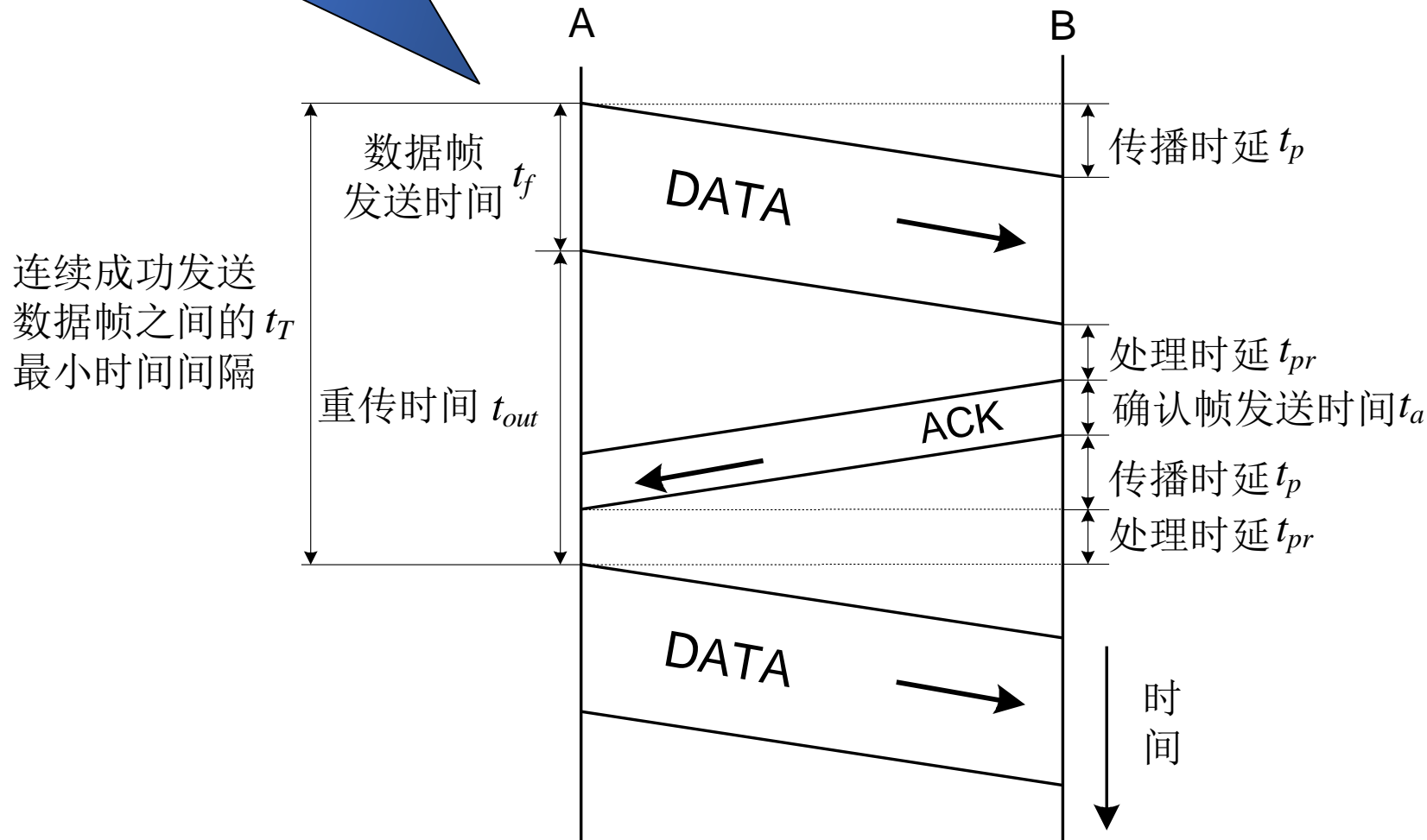


Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet

One-Bit Sliding Window Protocol (5)

发送效率: $\frac{t_f}{t_T} = \frac{l_f}{v \cdot t_T}$

长距离, 高带宽, 短帧的组合
-悲剧
(带宽延时积)

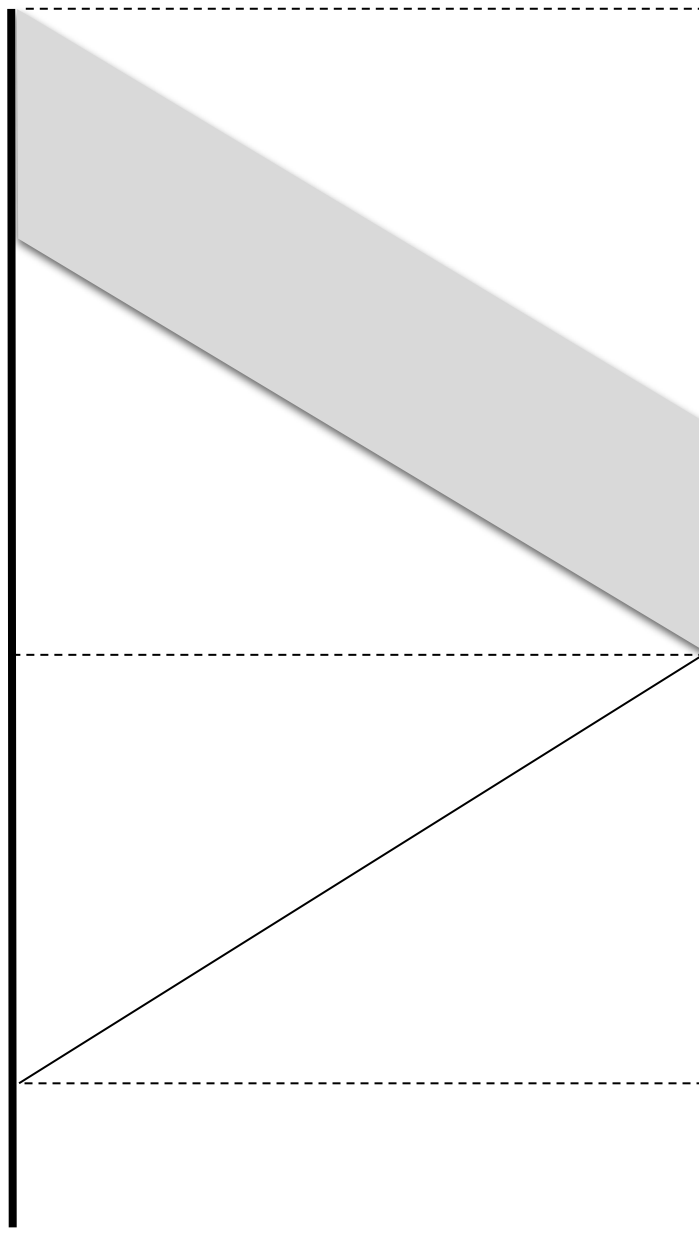


One-Bit Sliding Window Protocol (5)

流水线,
发多帧



但是有可能出错。
后面已经发出去
了，咋办呢？



A Protocol Using Go Back N or Selective repeat

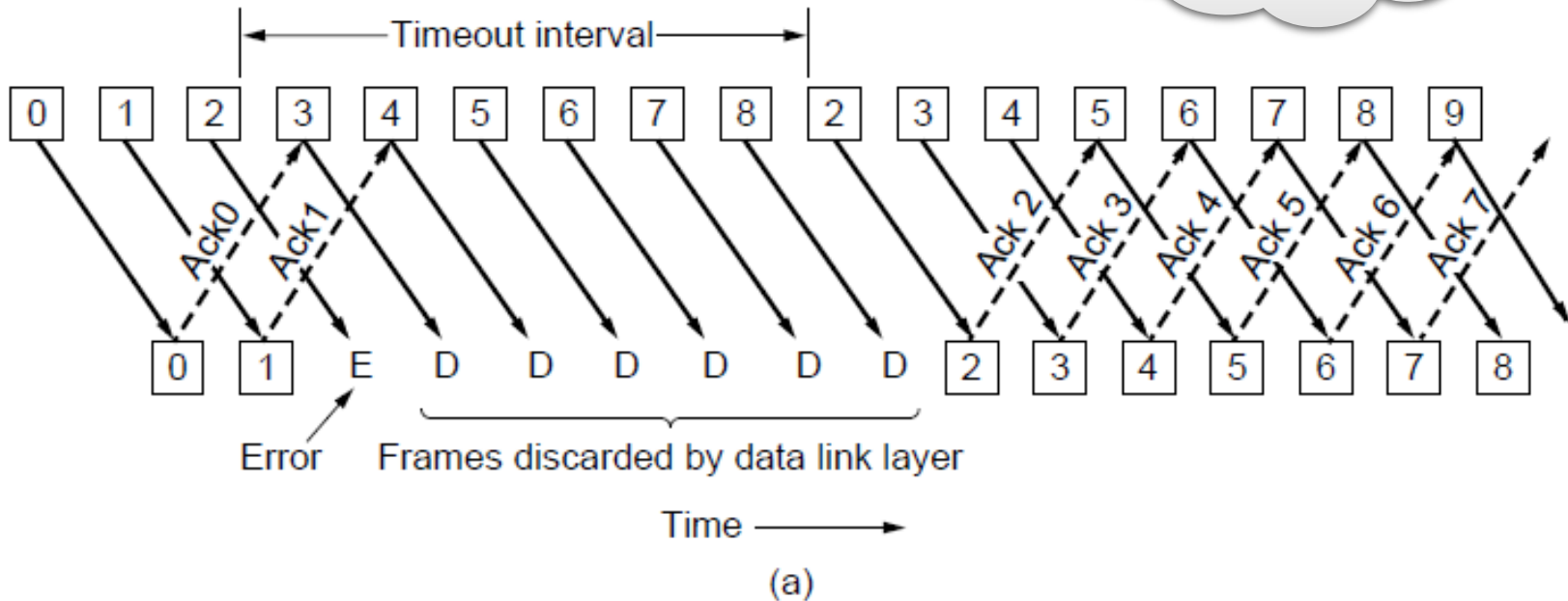
Problem: What should the receiver do if a frame is damaged?

1. Simply request retransmission of all frames starting from frame #N. If any other frames had been received in the meantime (and stored in the receiver's window), they'll just be ignored, that is **go back n**.
2. Request just retransmission of the damaged frame, and wait until it comes in before delivering any frames after that:
selective repeat.

Protocol Using Go-Back-N (1)

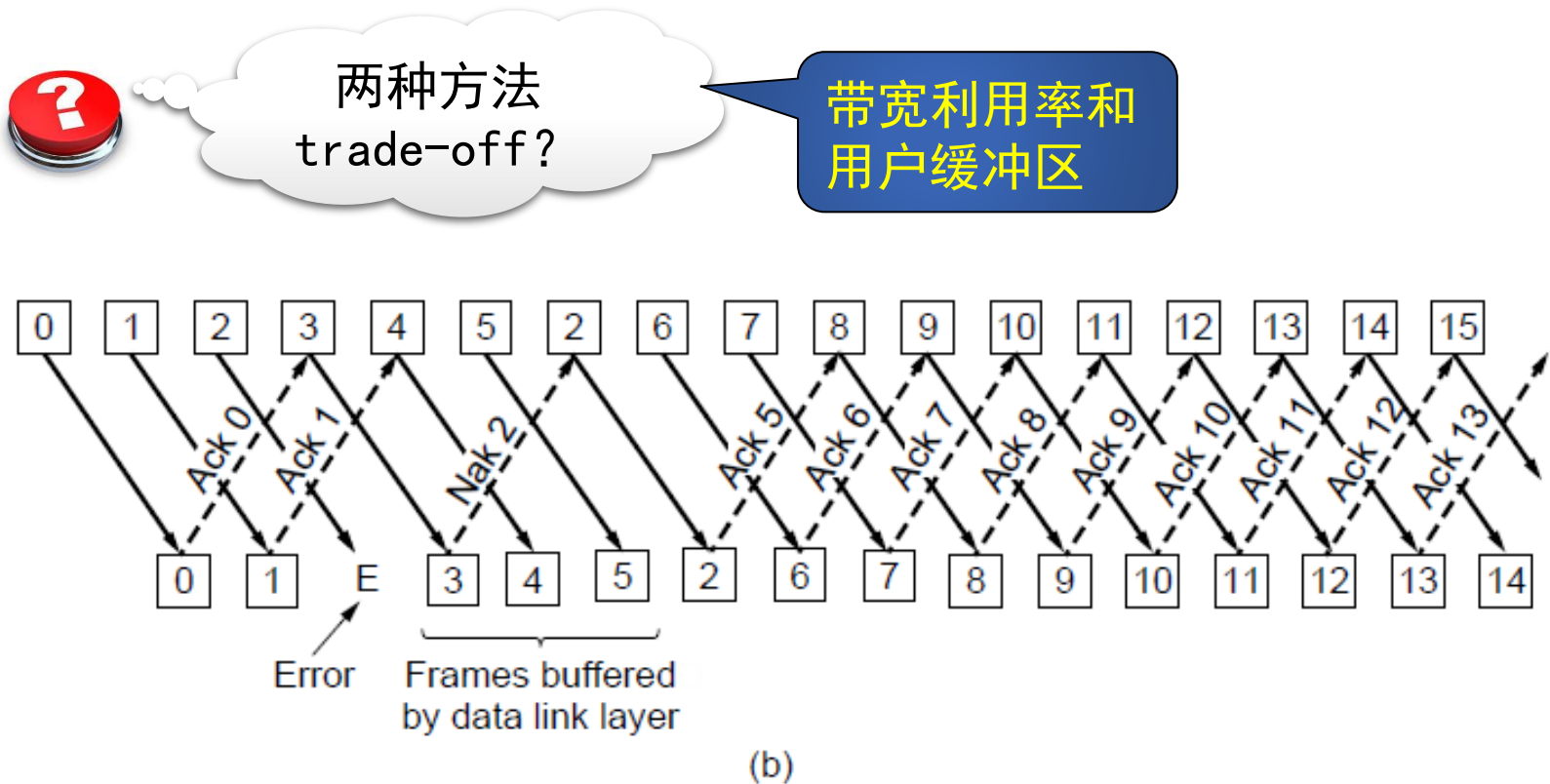


接收窗口
多大?



Pipelining and error recovery. Effect of an error when
(a) receiver's window size is 1

Protocol Using Go-Back-N (2)



Pipelining and error recovery. Effect of an error when
(b) receiver's window size is large.

Protocol Using Go-Back-N (3)

/* Protocol 5 (Go-back-n) allows multiple outstanding frames. The sender may transmit up to MAX_SEQ frames without waiting for an ack. In addition, unlike in the previous protocols, the network layer is not assumed to have a new packet all the time. Instead, the network layer causes a network_layer_ready event when there is a packet to send. */

```
#define MAX_SEQ 7
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
    /* Return true if a <= b < c circularly; false otherwise. */
    if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
        return(true);
    else
        return(false);
}
. . .
```

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (4)

```
static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
    /* Construct and send a data frame. */
    frame s;                                /* scratch variable */

    s.info = buffer[frame_nr];              /* insert packet into frame */
    s.seq = frame_nr;                       /* insert sequence number into frame */
    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1); /* piggyback ack */
    to_physical_layer(&s);                  /* transmit the frame */
    start_timer(frame_nr);                  /* start the timer running */
}

. . .
```

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (5)

```
void protocol5(void)
{
    seq_nr next_frame_to_send;           /* MAX_SEQ > 1; used for outbound stream */
    seq_nr ack_expected;                 /* oldest frame as yet unacknowledged */
    seq_nr frame_expected;               /* next frame expected on inbound stream */
    frame r;                             /* scratch variable */
    packet buffer[MAX_SEQ + 1];          /* buffers for the outbound stream */
    seq_nr nbuffered;                    /* number of output buffers currently in use */
    seq_nr i;                             /* used to index into the buffer array */
    event_type event;

    . . .
}
```

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (6)

```
enable_network_layer();  
ack_expected = 0;  
next_frame_to_send = 0;  
frame_expected = 0;  
nbuffered = 0;  
  
while (true) {  
    wait_for_event(&event);  
  
    . . .  
}
```

/* allow network_layer_ready events */
/* next ack expected inbound */
/* next frame going out */
/* number of frame expected inbound */
/* initially no packets are buffered */

/* four possibilities: see event_type above */

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (7)

```
switch(event) {  
  case network_layer_ready:          /* the network layer has a packet to send */  
    /* Accept, save, and transmit a new frame. */  
    from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */  
    nbuffered = nbuffered + 1;        /* expand the sender's window */  
    send_data(next_frame_to_send, frame_expected, buffer); /* transmit the frame */  
    inc(next_frame_to_send);          /* advance sender's upper window edge */  
    break;  
  
  case frame_arrival:                /* a data or control frame has arrived */  
    from_physical_layer(&r);          /* get incoming frame from physical layer */  
  
    if (r.seq == frame_expected) {  
      /* Frames are accepted only in order. */  
      to_network_layer(&r.info);      /* pass packet to network layer */  
      inc(frame_expected);            /* advance lower edge of receiver's window */  
    }  
  
  . . .
```

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (8)

```
/* Ack n implies n - 1, n - 2, etc. Check for this. */
while (between(ack_expected, r.ack, next_frame_to_send)) {
    /* Handle piggybacked ack. */
    nbuffered = nbuffered - 1;      /* one frame fewer buffered */
    stop_timer(ack_expected);      /* frame arrived intact; stop timer */
    inc(ack_expected);              /* contract sender's window */
}
break;

case cksum_err: break;             /* just ignore bad frames */

case timeout:                      /* trouble; retransmit all outstanding frames */
    next_frame_to_send = ack_expected; /* start retransmitting here */
    for (i = 1; i <= nbuffered; i++) {
        send_data(next_frame_to_send, frame_expected, buffer); /* resend frame */
        inc(next_frame_to_send); /* prepare to send the next one */
    }
}

...
```

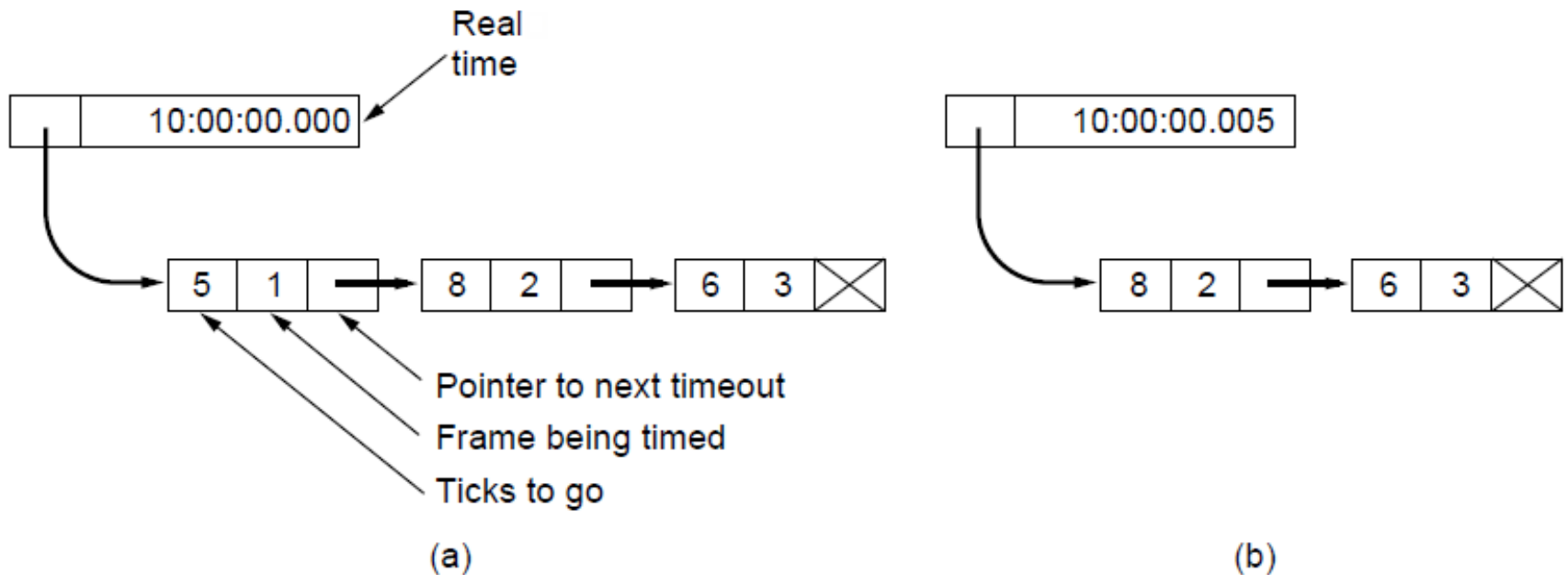
A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (9)

```
if (nbuffered < MAX_SEQ)
    enable_network_layer();
else
    disable_network_layer();
}
```

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (10)



Simulation of multiple timers in software. (a) The queued timeouts (b) The situation after the first timeout has expired.

Protocol Using Selective Repeat (1)

```
/* Protocol 6 (Selective repeat) accepts frames out of order but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7                                /* should be  $2^n - 1$  */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                          /* no nak has been sent yet */
seq_nr oldest_frame = MAX_SEQ + 1;              /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol 5, but shorter and more obscure. */
return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

. . .
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (2)

```
static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Construct and send a data, ack, or nak frame. */
    frame s;                                /* scratch variable */

    s.kind = fk;                            /* kind == data, ack, or nak */
    if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
    s.seq = frame_nr;                       /* only meaningful for data frames */
    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
    if (fk == nak) no_nak = false;          /* one nak per frame, please */
    to_physical_layer(&s);                  /* transmit the frame */
    if (fk == data) start_timer(frame_nr % NR_BUFS);
    stop_ack_timer();                       /* no need for separate ack frame */
}
```

...

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (3)

```
void protocol6(void)
{
    seq_nr ack_expected;
    seq_nr next_frame_to_send;
    seq_nr frame_expected;
    seq_nr too_far;
    int i;
    frame r;
    packet out_buf[NR_BUFS];
    packet in_buf[NR_BUFS];
    boolean arrived[NR_BUFS];
    seq_nr nbuffered;
    event_type event;

    /* lower edge of sender's window */
    /* upper edge of sender's window + 1 */
    /* lower edge of receiver's window */
    /* upper edge of receiver's window + 1 */
    /* index into buffer pool */
    /* scratch variable */
    /* buffers for the outbound stream */
    /* buffers for the inbound stream */
    /* inbound bit map */
    /* how many output buffers currently used */
}
```

...

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (4)

```
enable_network_layer();           /* initialize */
ack_expected = 0;                  /* next ack expected on the inbound stream */
next_frame_to_send = 0;           /* number of next outgoing frame */
frame_expected = 0;
too_far = NR_BUFS;
nbuffered = 0;                    /* initially no packets are buffered */
for (i = 0; i < NR_BUFS; i++) arrived[i] = false;

. . .
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (5)

```
while (true) {  
    wait_for_event(&event);           /* five possibilities: see event_type above */  
    switch(event) {  
        case network_layer_ready:     /* accept, save, and transmit a new frame */  
            nbuffered = nbuffered + 1; /* expand the window */  
            from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */  
            send_frame(data, next_frame_to_send, frame_expected, out_buf); /* transmit the frame */  
            inc(next_frame_to_send);   /* advance upper window edge */  
            break;  
        . . .  
    }  
}
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (6)

```
case frame_arrival:                /* a data or control frame has arrived */
    from_physical_layer(&r);        /* fetch incoming frame from physical layer */
    if (r.kind == data) {
        /* An undamaged frame has arrived. */
        if ((r.seq != frame_expected) && no_nak)
            send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
        if (between(frame_expected, r.seq, too_far) && (arrived[r.seq % NR_BUFS] == false)) {
            /* Frames may be accepted in any order. */
            arrived[r.seq % NR_BUFS] = true;    /* mark buffer as full */
            in_buf[r.seq % NR_BUFS] = r.info;  /* insert data into buffer */
        }
    }
    . . .
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (7)

```
while (arrived[frame_expected % NR_BUFS]) {  
    /* Pass frames and advance window. */  
    to_network_layer(&in_buf[frame_expected % NR_BUFS]);  
    no_nak = true;  
    arrived[frame_expected % NR_BUFS] = false;  
    inc(frame_expected);    /* advance lower edge of receiver's window */  
    inc(too_far);           /* advance upper edge of receiver's window */  
    start_ack_timer();      /* to see if a separate ack is needed */  
}  
}  
...  
}
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (8)

```
if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next_frame_to_send))
    send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);

while (between(ack_expected, r.ack, next_frame_to_send)) {
    nbuffered = nbuffered - 1;          /* handle piggybacked ack */
    stop_timer(ack_expected % NR_BUFS); /* frame arrived intact */
    inc(ack_expected);                  /* advance lower edge of sender's window */
}
break;

case cksum_err:
    if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* damaged frame */
    break;
    . . .
```

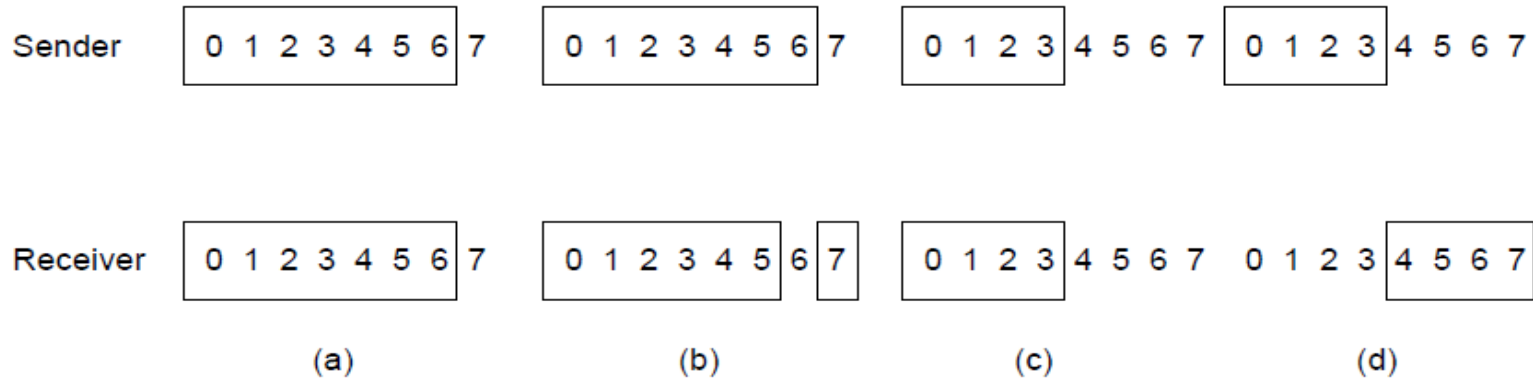
A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (9)

```
case timeout:
    send_frame(data, oldest_frame, frame_expected, out_buf); /* we timed out */
    break;
case ack_timeout:
    send_frame(ack, 0, frame_expected, out_buf); /* ack timer expired; send ack */
}
if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
}
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (10)



- (a) Initial situation with a window size seven.
- (b) After seven frames sent and received, but not acknowledged.
- (c) Initial situation with a window size of four.
- (d) After four frames sent and received, but not acknowledged.

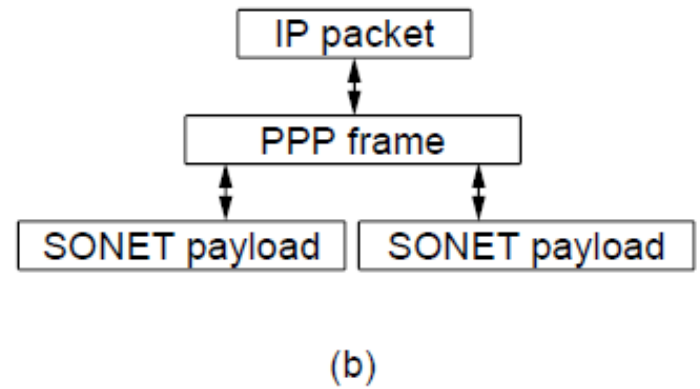
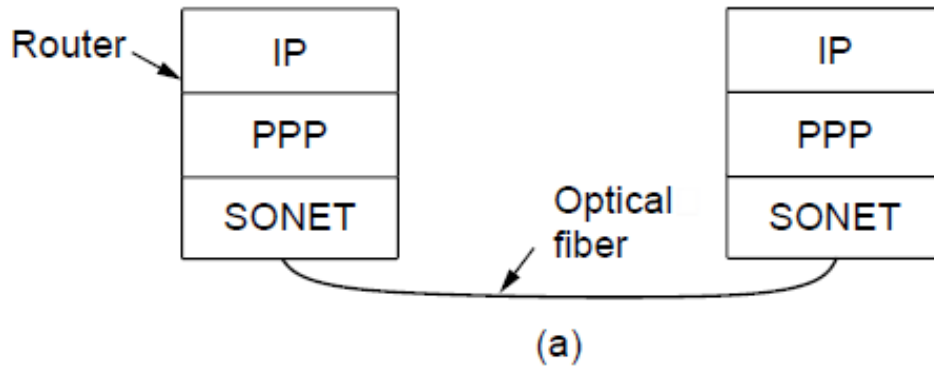
Protocol Using Selective Repeat (10)

- So, we must **avoid overlapping** send and receive windows, the highest sequence number must be at least twice the window size.

3.5 Example Data Link Protocols

- Packet over SONET
- ADSL (Asymmetric Digital Subscriber Loop)

Packet over SONET (1)



Packet over SONET. (a) A protocol stack. (b) Frame relationships

Packet over SONET (2)

PPP Features

Separate packets, error detection

Link Control Protocol

Network Control Protocol

成帧

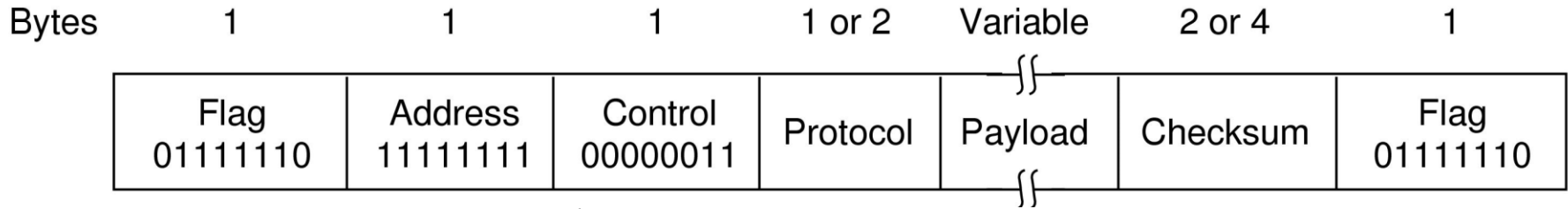
链路控制：启动线路、测试线路、协商参数…

协商网络层选项

Packet over SONET (2)

- **PPP 的需求 (RFC 1547)**
 - 分组成帧
 - 透明性
 - 多种网络层协议
 - 链路的各种类型
 - 错误检测
 - 连接的存活
 - 网络地址的协商
 - 简单性 (目前, 已有50多个RFC定义这个“简单”协议)
- **PPP 不需要做**
 - 错误纠正
 - 流量控制
 - 有序性
 - 多点链路

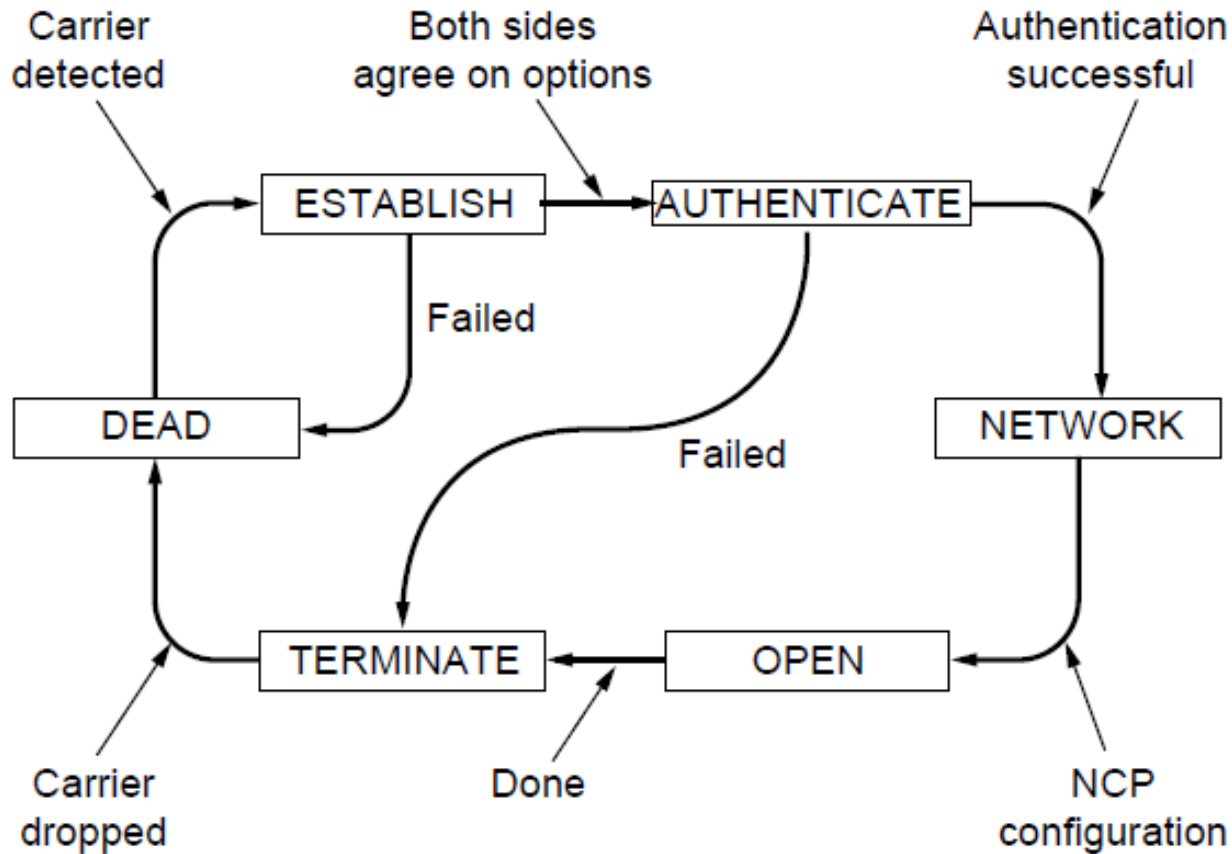
Packet over SONET (3)



- 标记域：01111110，字符填充；
- 地址域：11111111
- 控制域：缺省值为00000011，表示无序号帧，缺省情况下，PPP不提供使用序号和确认的可靠传输；但是在不可靠线路上，也可使用有序号的可靠传输。
- 协议域：指示净负荷中是何种包，比如IP，IPX等。缺省大小为2个字节。
- 净负荷域：变长，缺省为1500字节；
- 校验和域：2或4个字节

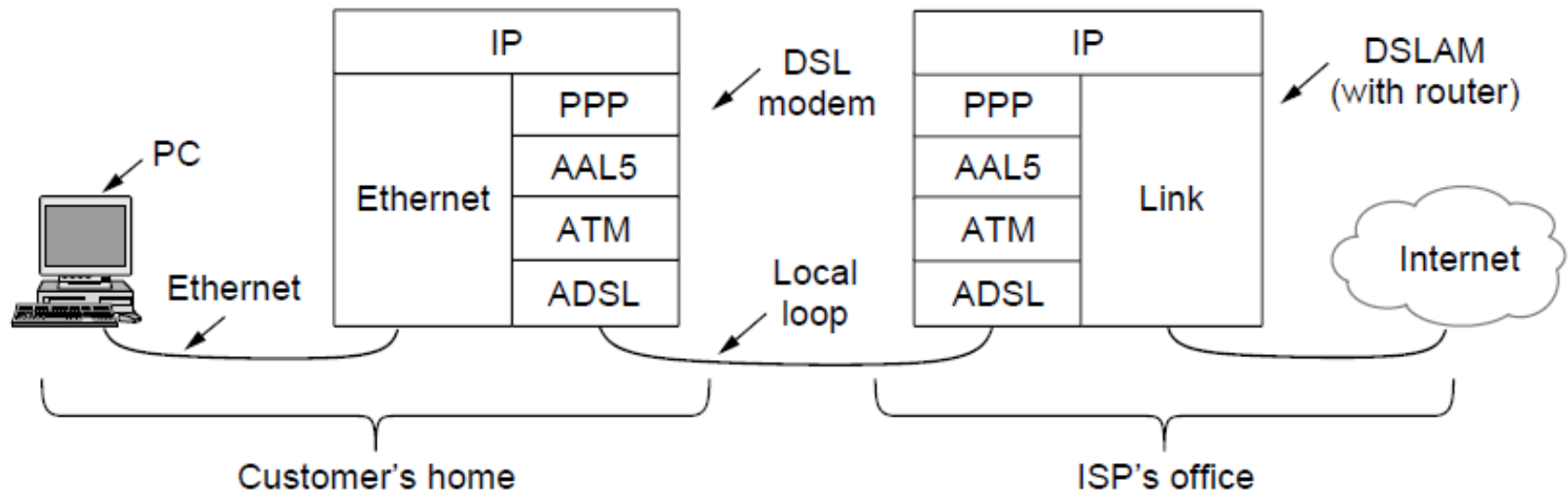
The PPP full frame format for unnumbered mode operation

Packet over SONET (4)



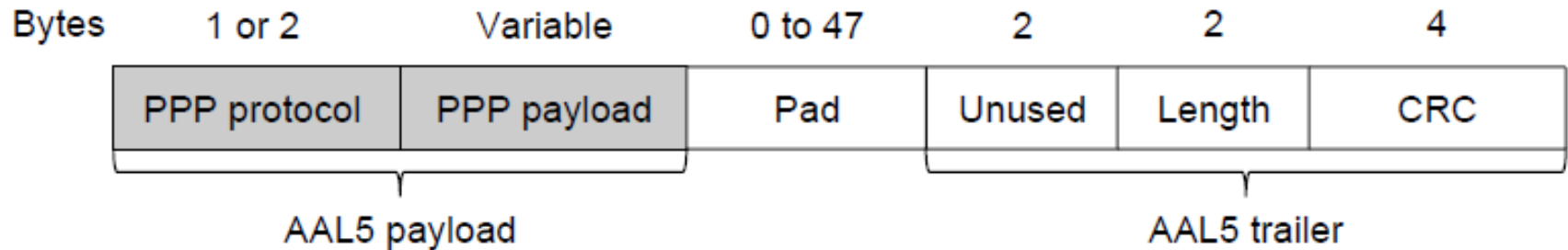
State diagram for bringing a PPP link up and down

ADSL (Asymmetric Digital Subscriber Loop) (1)



ADSL protocol stacks.

ADSL (Asymmetric Digital Subscriber Loop) (1)



AAL5 frame carrying PPP data

End

Chapter 3