Chapter 3

The Data Link Layer

3.1 Data Link Layer Design Issues

- Services Provided to the Network Layer
- Framing
- Error Control
- Flow Control

3.1.1 Services provided to the network layer

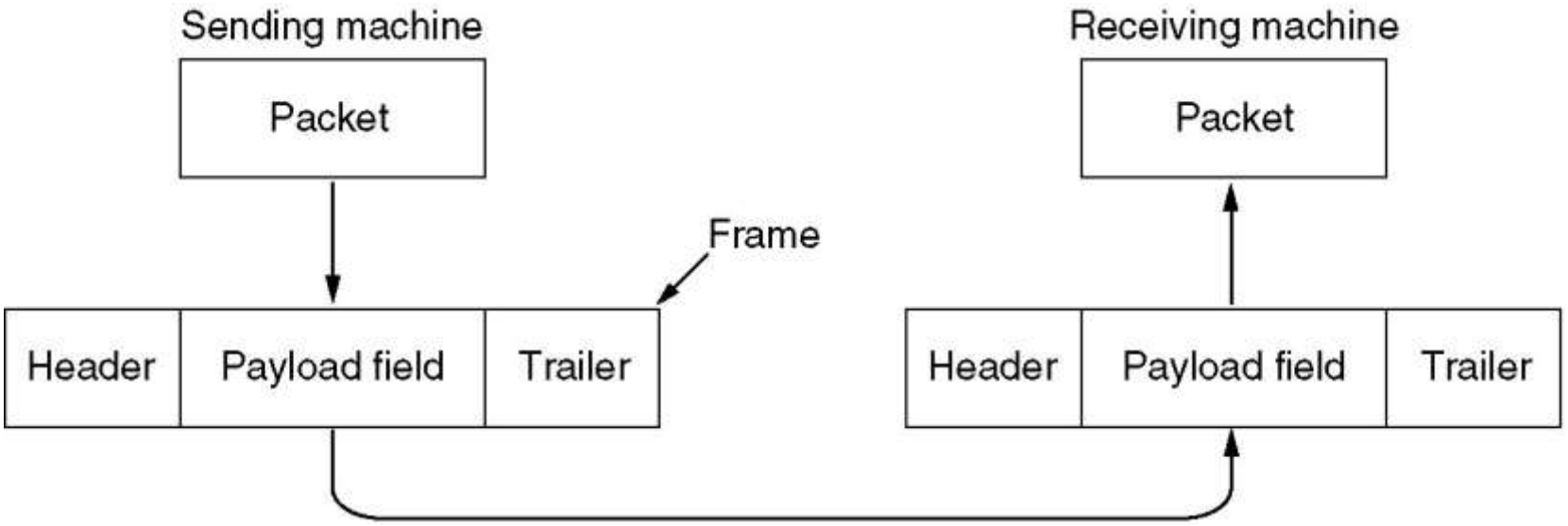
- Unacknowledged connectionless service
- Acknowledged connectionless service
- Acknowledged connection-oriented service

Functions of the Data Link Layer

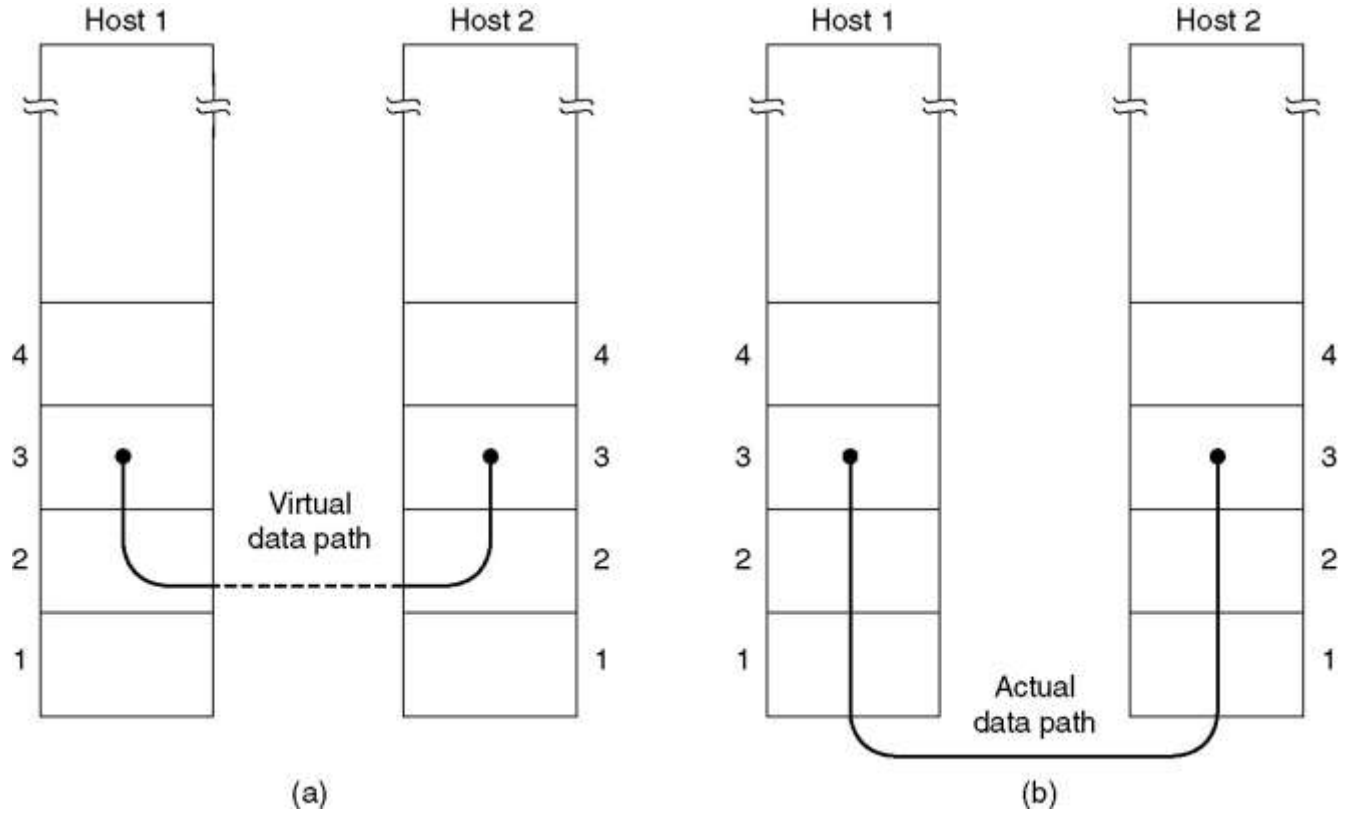
- Provide service interface to the network layer
- Dealing with transmission errors
- Regulating data flow
  - Slow receivers not swamped by fast senders

Functions of the Data Link Layer (2)

- **Relationship between packets and frames.**



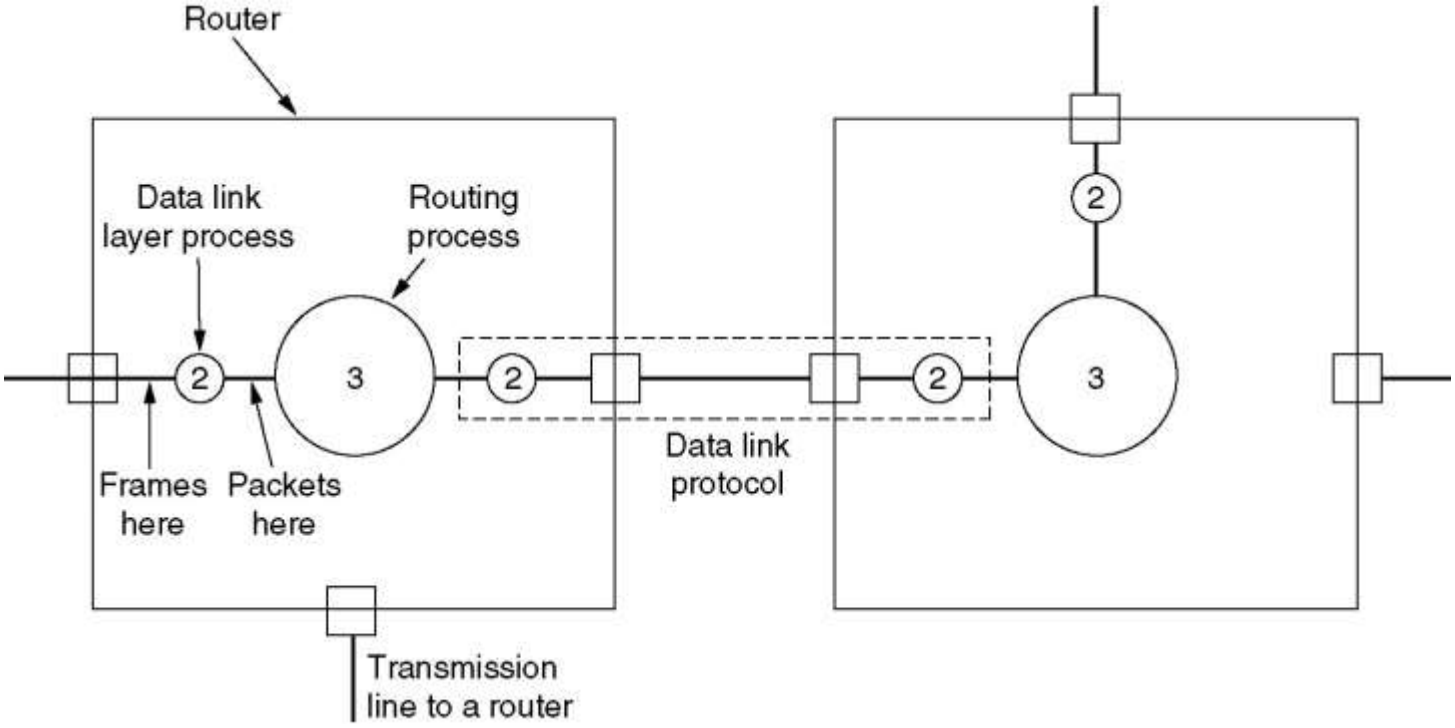
Services Provided to Network Layer



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

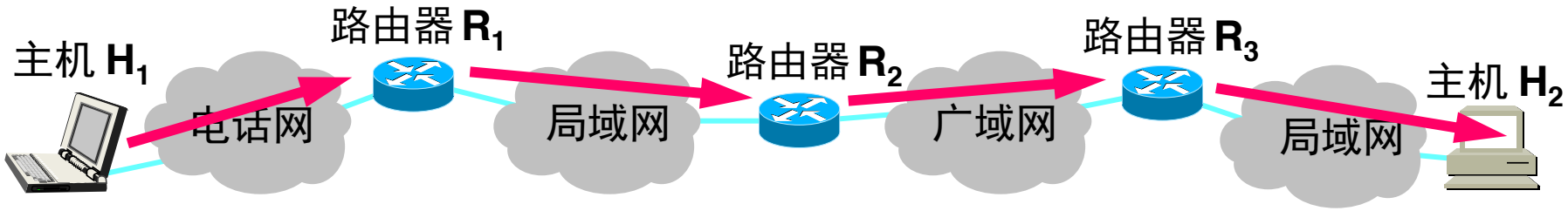
Services Provided to Network Layer (2)

- **Placement of the data link protocol.**

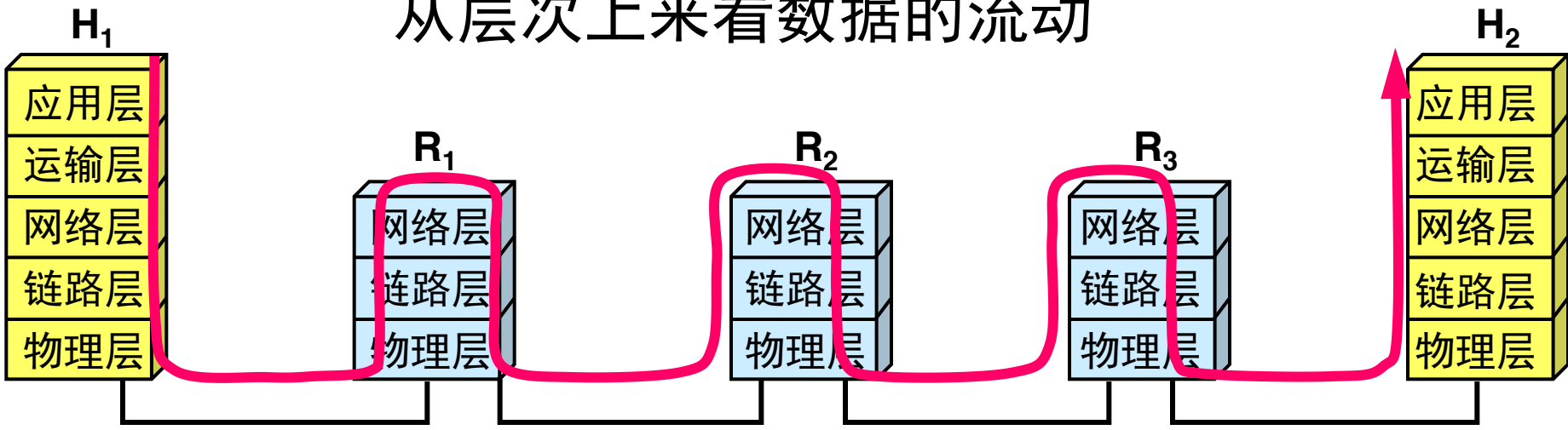


数据链路层的简单模型

主机 H_1 向 H_2 发送数据

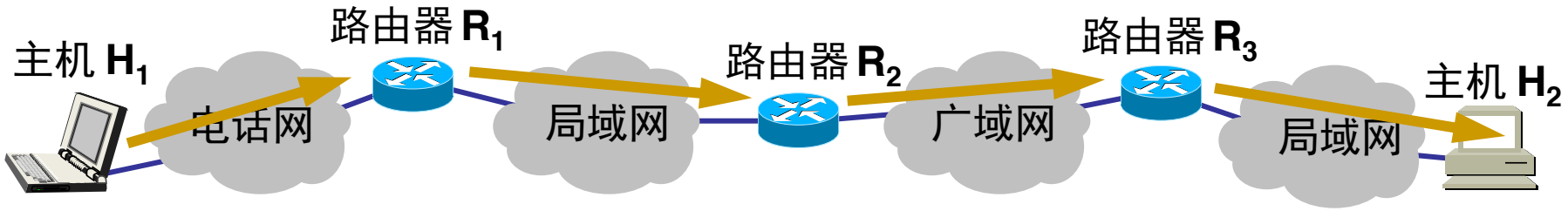


从层次上来看数据的流动

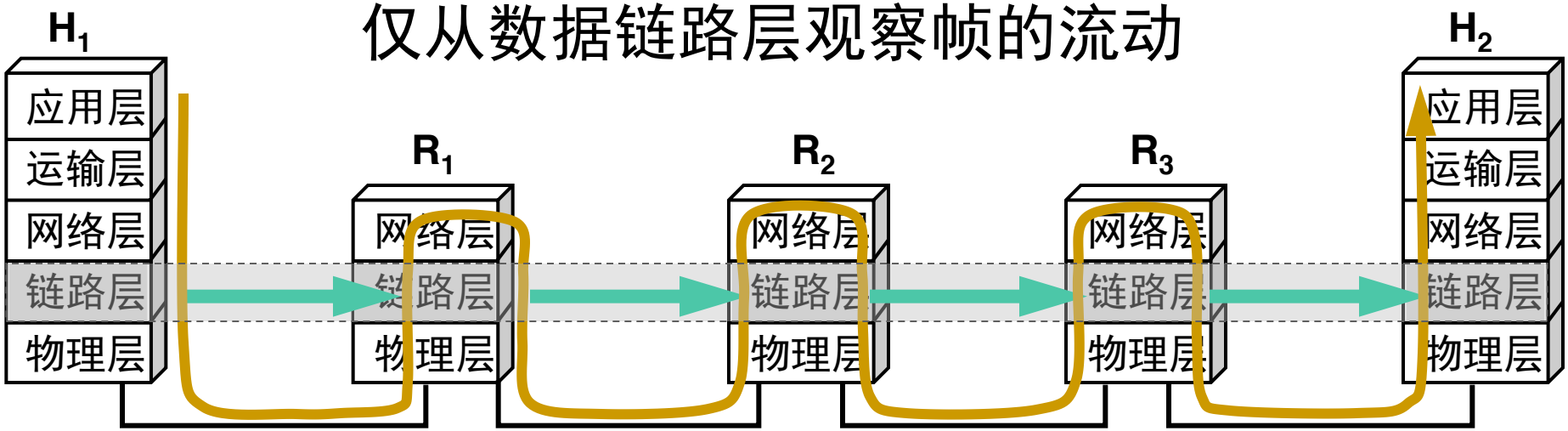


数据链路层的简单模型(续）

主机 H_1 向 H_2 发送数据



仅从数据链路层观察帧的流动

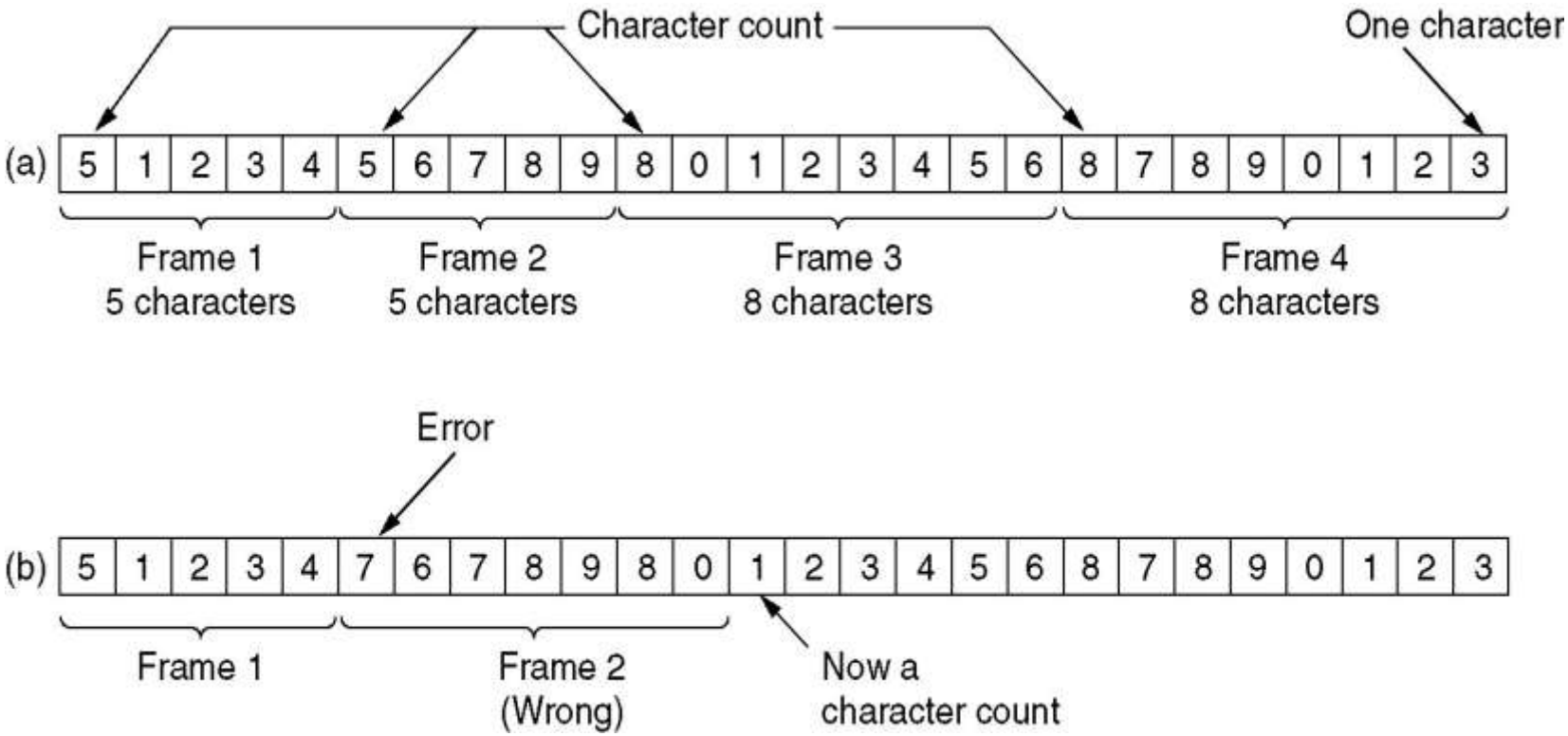


3.1.2 Framing Methods

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

Framing (2)

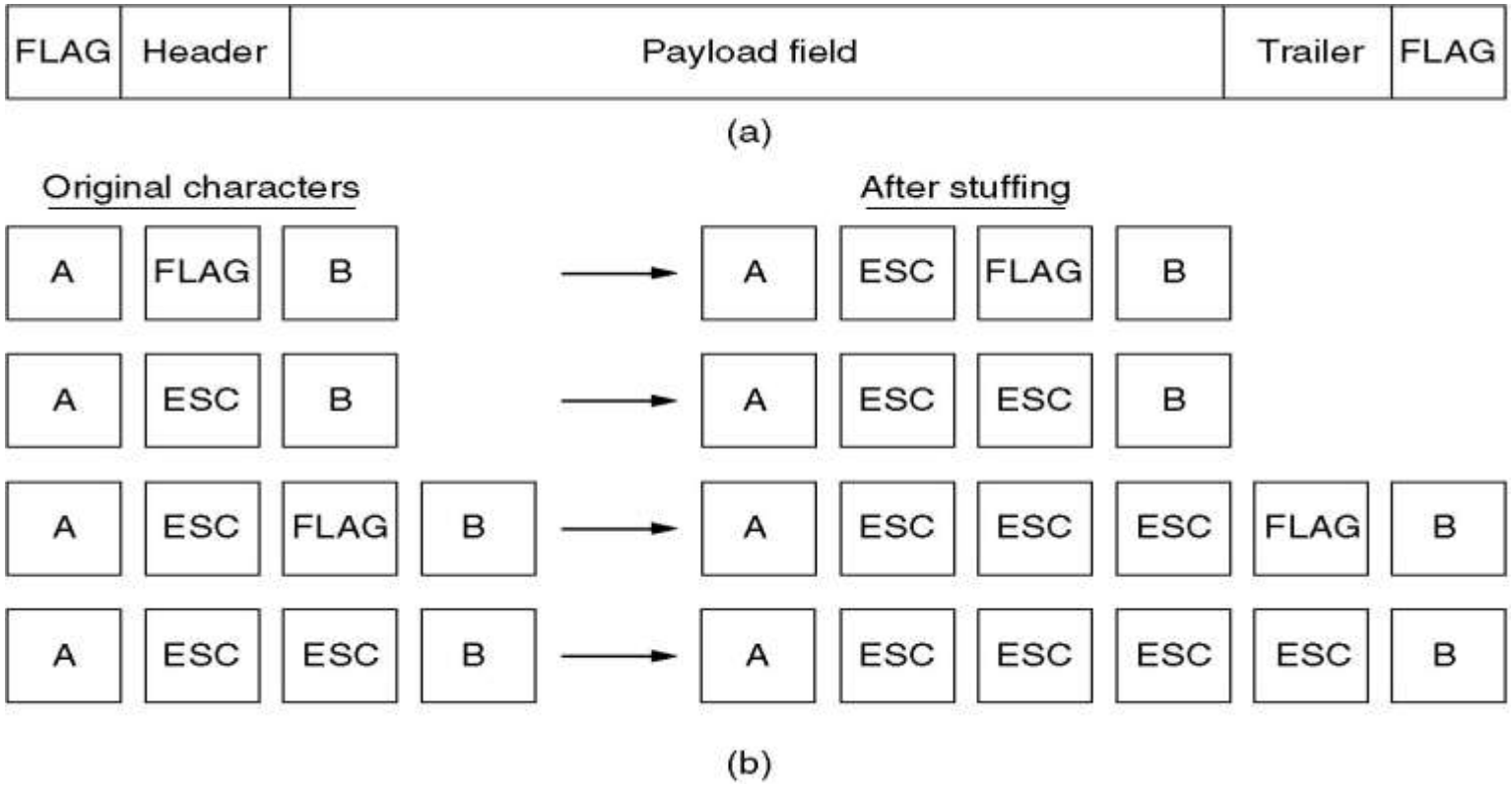
- **Byte count（字节计数法 ）**



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

Framing (3)

- **Flag bytes with byte stuffing**



(a) A frame delimited by flag bytes.

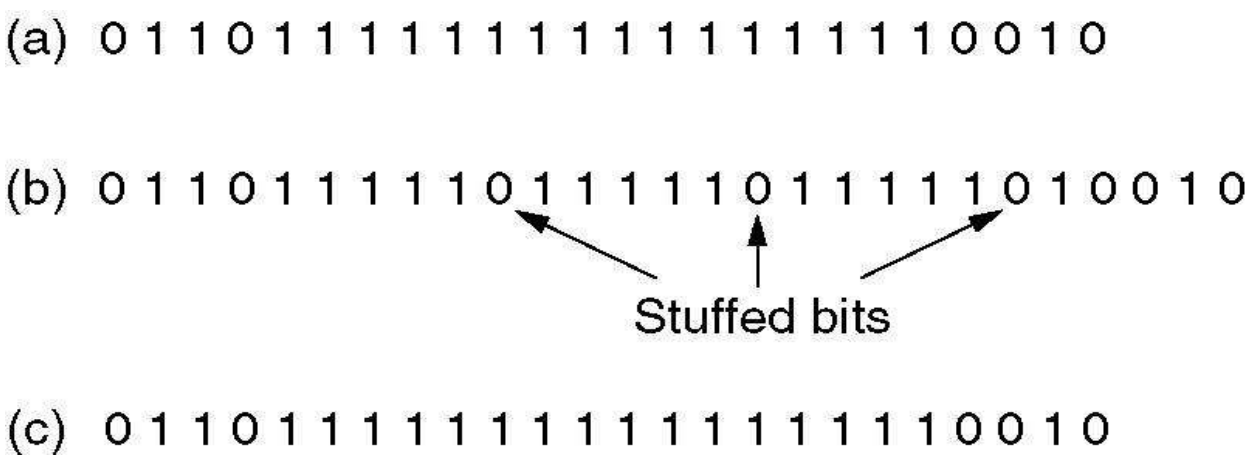
(b) Four examples of byte sequences before and after stuffing.

Framing (4)

- **Flag bytes with byte stuffing**
  - **Starting and ending character, with character stuffing**（带字符填充的首尾界符法 ）
    - Frame flags: DLE STX(Start Text), DLE ETX(End Text)
    - DLE in data(between flags): DLE-->DLE DLE, so that: DLE ETX--> DLE DLE ETX
    - ESC=0x1B，SOH(Start of Header)=0x01，EOT(End of Transmission)=0x04
  - **Application Example：**
    - **PPP（Point-to-Point Protocol）**

Framing (5)

- **Flag bits with bit stuffing.**
  - **Starting and ending flags,with bit stuffing（带位填充的首尾界符法 ）.-（01111110）**



  - **Application Example：HDLC**

Bit stuffing

(a) The original data.

(b) The data as they appear on the line.

(c) The data as they are stored in receiver's memory after destuffing.

Framing (6)

- **Physical layer coding violation**
  - **a 1 bit ->high-low pair,**
  - **a 0 bit ->low-high pair**

3.1.3 Error Control（差错控制）

- Error detection
- Error correction
- Repeated data frame
- Lost frame
- Timer

3.1.4 Flow Control（流量控制）

- To solve the problem that a sender wants to transmit frames faster than the receiver can accept them.
- Feedback-based flow control
- Rate-based flow control

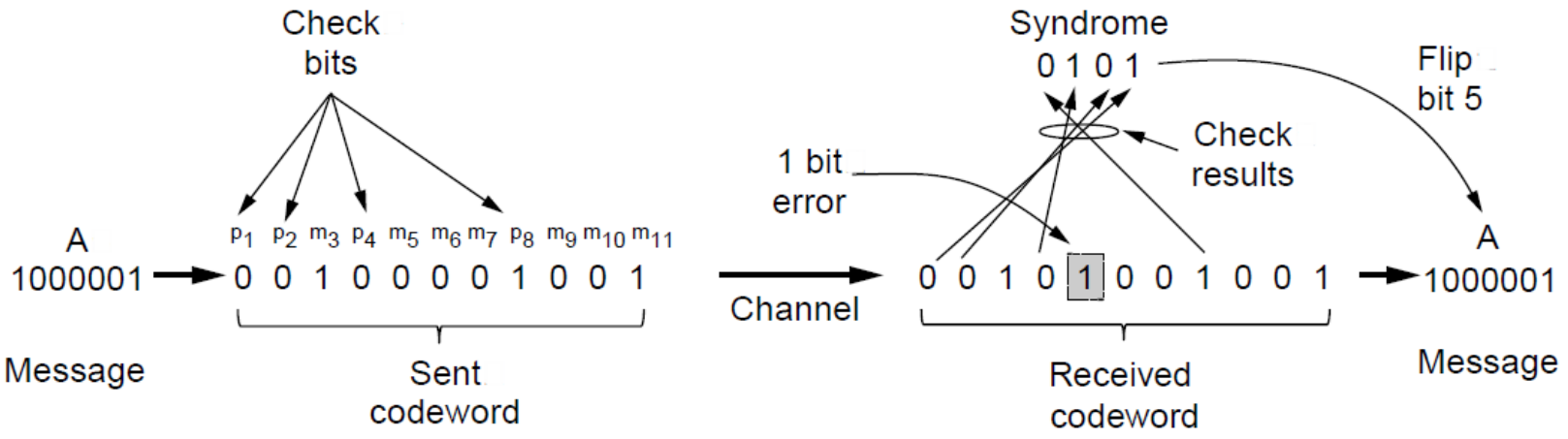
3.2 Error Detection and Correction

- Error-Correcting Codes（纠错码）
  - codeword （码字）
  - Hamming distance (10001001,10110001=>00111000,3)
  - parity bit
  - 纠正d比特错
    - 需距离为2d+1比特的编码
- Error-Detecting Codes（检错码）
  - polynomial code (cyclic redundancy code, CRC code)
  - 检测d比特错
    - 需距离为d+1比特的编码

3.2.1Error-Correcting Codes (1)

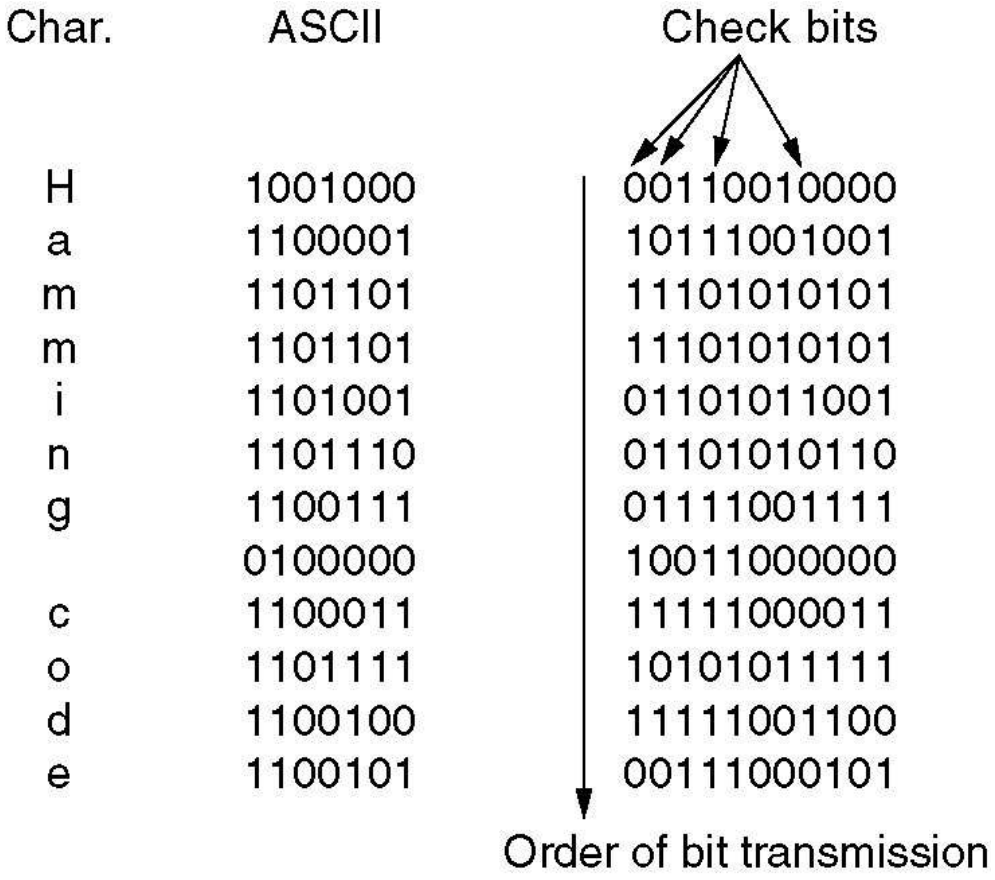
1.Hamming codes.
2.Binary convolutional codes.二进制卷积码
3.Reed-Solomon codes.
4.Low-Density Parity Check codes.

Error-Correcting Codes (2)



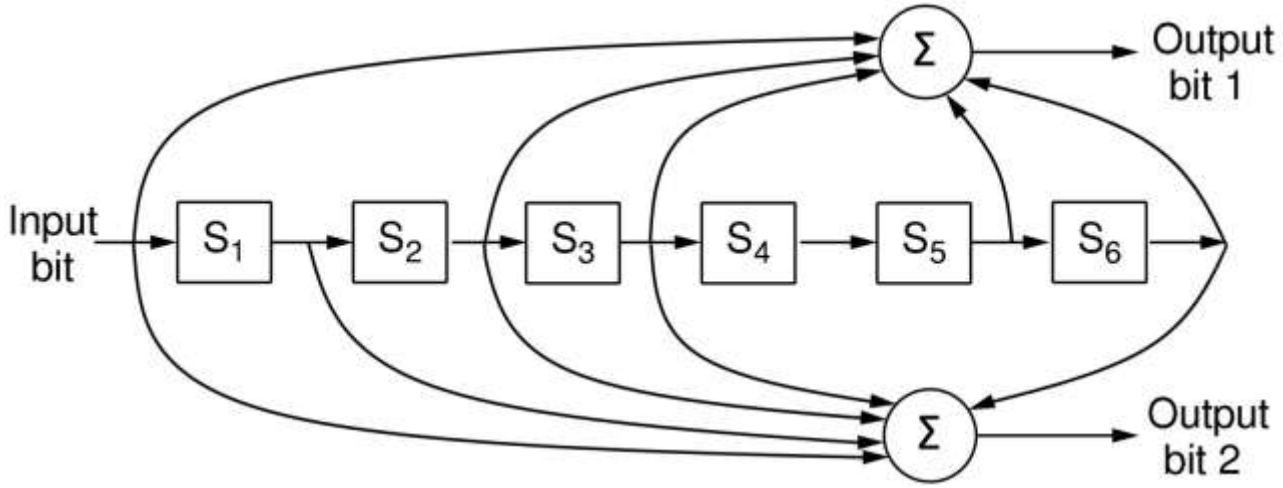
Example of an (11, 7) Hamming code correcting a single-bit error.

Error-Correcting Codes (3)



Use of a Hamming code to correct burst errors.

Error-Correcting Codes (4)



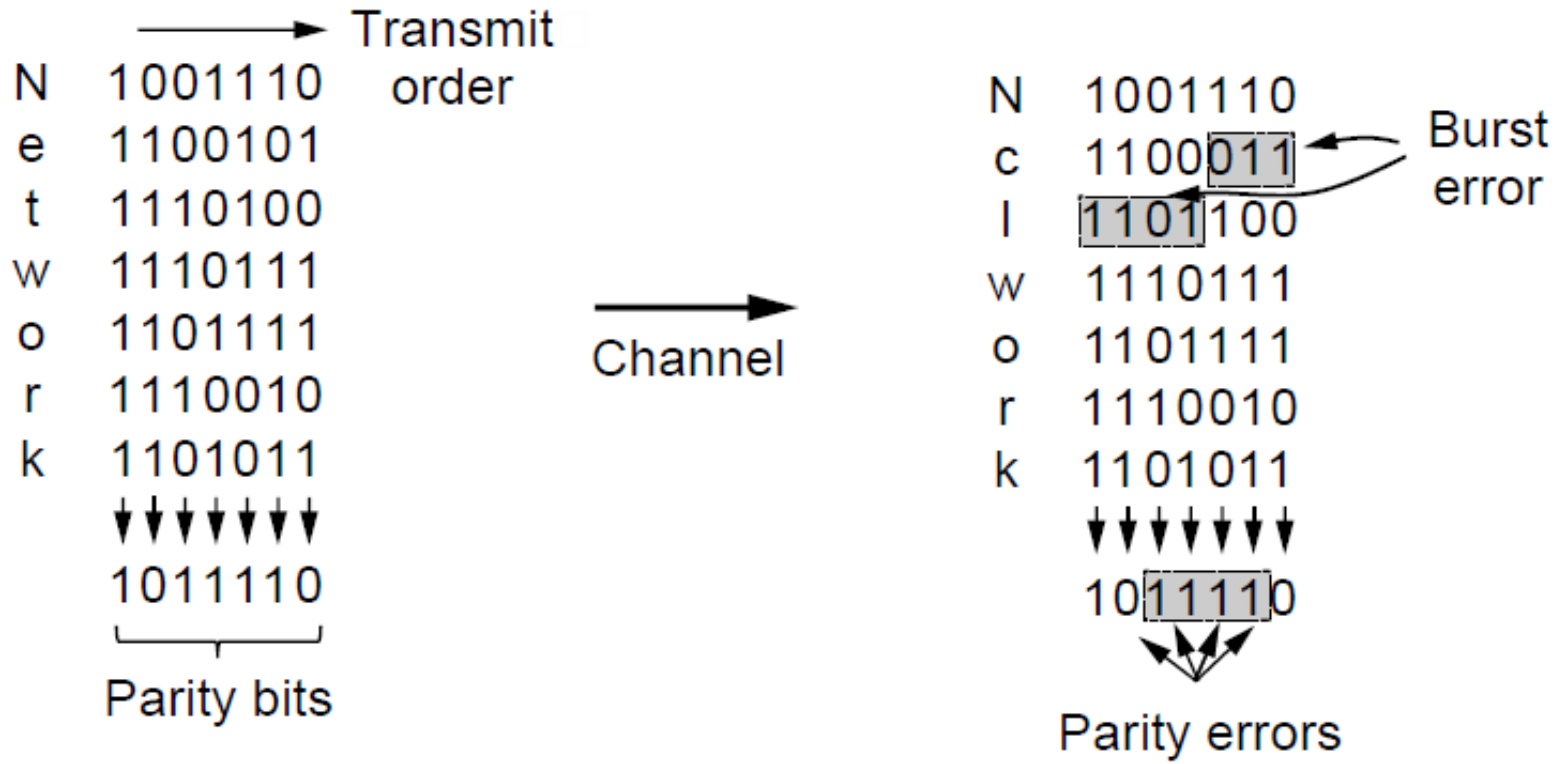
The NASA binary convolutional code used in 802.11.

3.2.2 Error-Detecting Codes (1)

Linear, systematic block codes

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

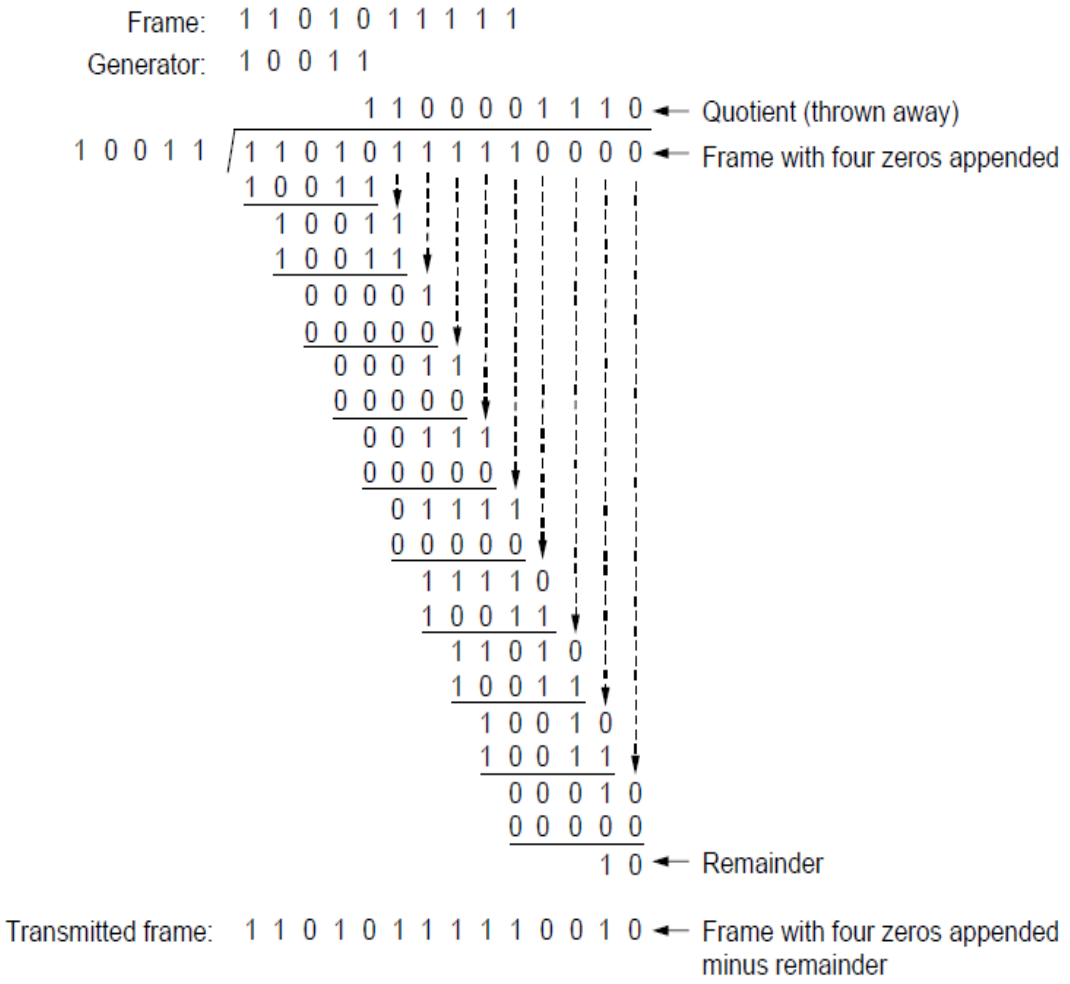
Error-Detecting Codes (2)



Error-Detecting Codes （3）

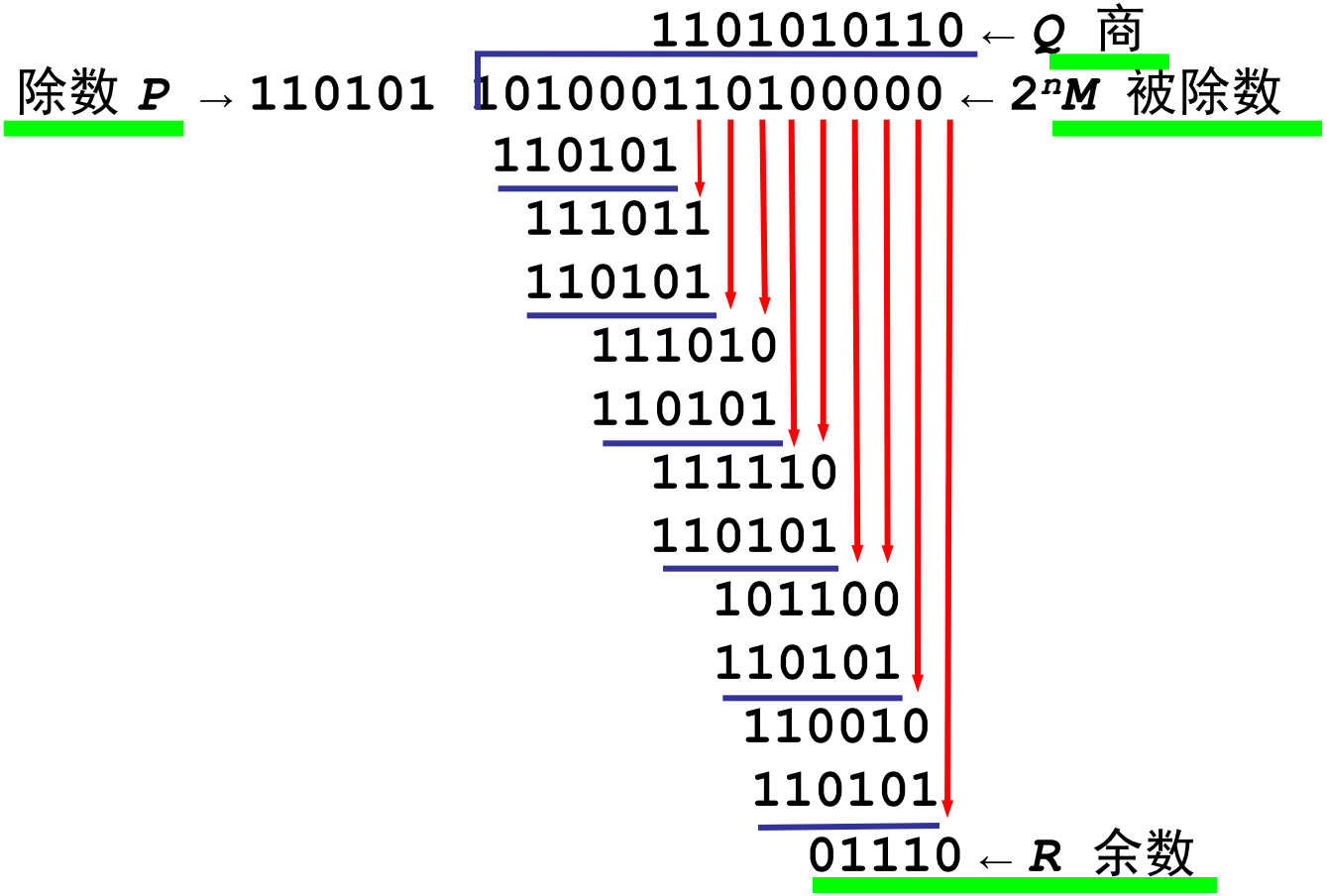
Calculation of the polynomial code checksum

- 多项式编码
  - （循环冗余码，CRC码）
- 生成多项式 G(x)
- 计算校验和的算法：
  - -[see page 197]
- CRC-12 = $x^{12}+x^{11}+x^3+x^2+x^1+1$
- CRC-16 = $x^{16}+x^{15}+x^2+1$
- CRC-CCITT = $x^{16}+x^{12}+x^5+1$
- 结论：
  - 具有r检测位的多项式能够检测所有长度<=r的突发错误
  - 长度大于r+1的错误逃脱的概率为$1/2^r$.



Example calculation of the CRC

循环冗余检验的原理说明

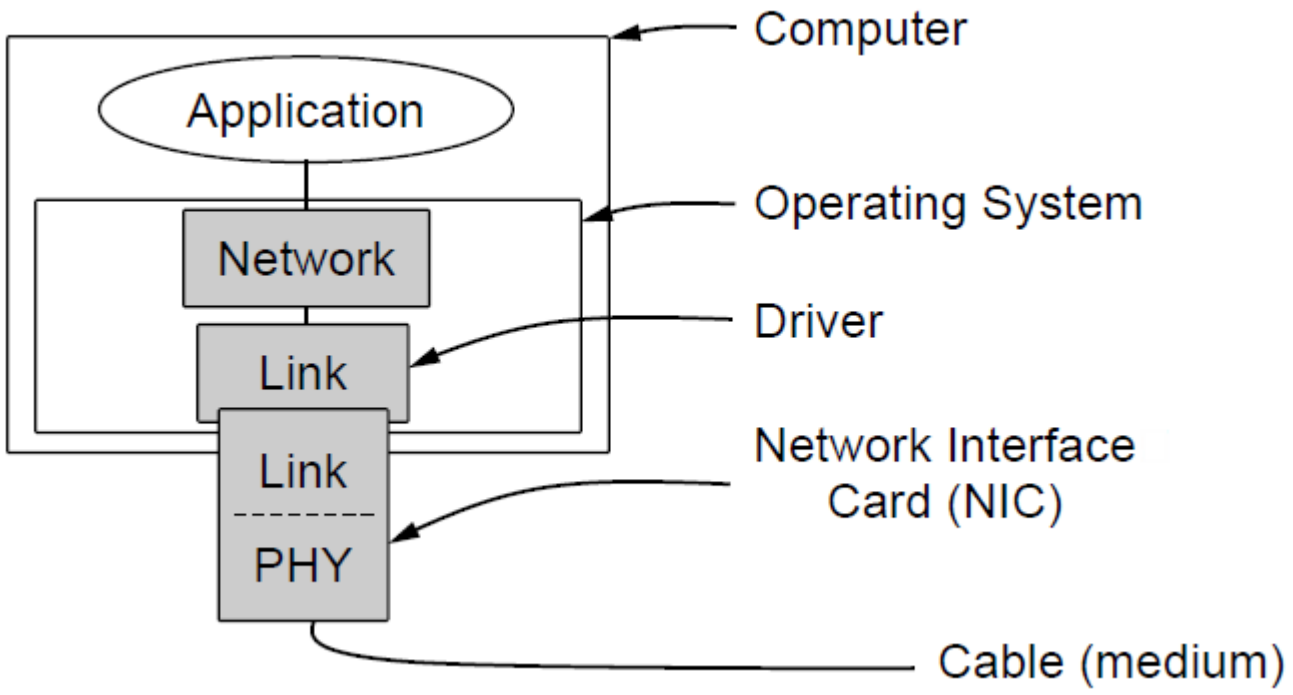


3.3 Elementary Data Link Protocols

- A utopian Simplex Protocol（一种无限制的单工协议）
- A Simplex Stop-and-Wait Protocol（一种单工的停-等协议）
  - Error-Free Channel
- A Simplex Protocol for a Noisy Channel（有噪声信道的单工协）
  - Noisy Channel

- 进程(process)、分组(packet)和帧(frame)
- wait_for_event(&event), 3 events:
  - frame_arrival: Received correct frame
  - cksum_err: Received frame with checksum error
  - timeout: Timeout when waiting acknowledgement

Elementary Data Link Protocols



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

Protocol Definitions

```
#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;
```

Continued →

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

Protocol Definitions (ctd.)

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

```
/* 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
```

3.3.1 Utopian Simplex Protocol(一种无限制的单工协议)

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

3.3.2 Simplex Stop-and-Wait Protocol (一种单工的停–等协议)

- stop-and-wait, 单向数据传输, 发送方网络层一直有无限的数据要发送, 从不损坏或丢失帧.

- 抛弃约束条件: 接收方网络层一直可以接收, 接收方缓冲空间无限大.

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

3.3.3 A Simplex Protocol for a Noisy Channel

- 一个肯定确认的重传协议(PAR)
- 计时器(timer)、序号(sequence)、PAR/ARQ、时间间隔(timeout)
- network layer of sender has unlimited data to transmit.
- Discard restricted condition: channel never damages or loses frame.

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

PAR:Positive Acknowledgement with Retransmission

ARQ:Automatic Repeat reQuest

Continued →

A Simplex Protocol for a Noisy Channel (ctd.)

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

 frame_expected = 0;
 while (true) {
     wait_for_event(&event);                 /* possibilities: frame_arrival, cksum_err */
     if (event == frame_arrival) {           /* a valid frame has arrived. */
          from_physical_layer(&r);           /* go get the newly arrived frame */
          if (r.seq == frame  expected) {    /* this is what we have been waiting for. */
               to_network_layer(&r.info);    /* pass the data to the network layer */
               inc(frame_expected);          /* next time expect the other sequence nr */
          }
          s.ack = 1 – frame_expected;        /* tell which frame is being acked */
          to_physical_layer(&s);             /* send acknowledgement */
     }
 }
}
```

A positive acknowledgement with retransmission protocol.

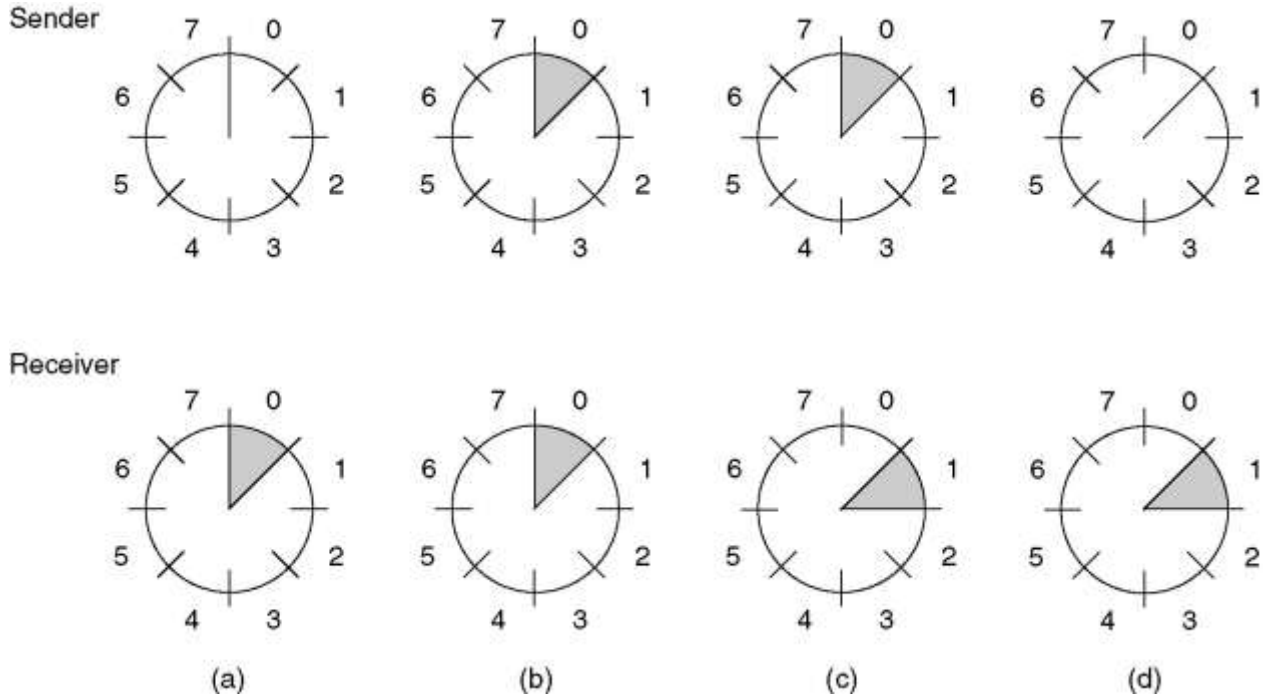
3.4 Sliding Window Protocols
（滑动窗口协议）

- Bidirectional transmission, **piggybacking**（捎带）, better use of available bandwidth
- **slide window**
  - **sending window**: a set of sequence numbers corresponding to frame it is permitted to send for sender
  - **receiving window**: corresponding to the set of frames which is permitted to accept for receiver

Sliding Window Protocols (2)

- A One-Bit Sliding Window Protocol
- A Protocol Using Go Back N（后退n帧）
- A Protocol Using Selective Repeat（选择性重传）

Sliding Window Protocols (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.

3.4.1 A One-Bit Sliding Window Protocol

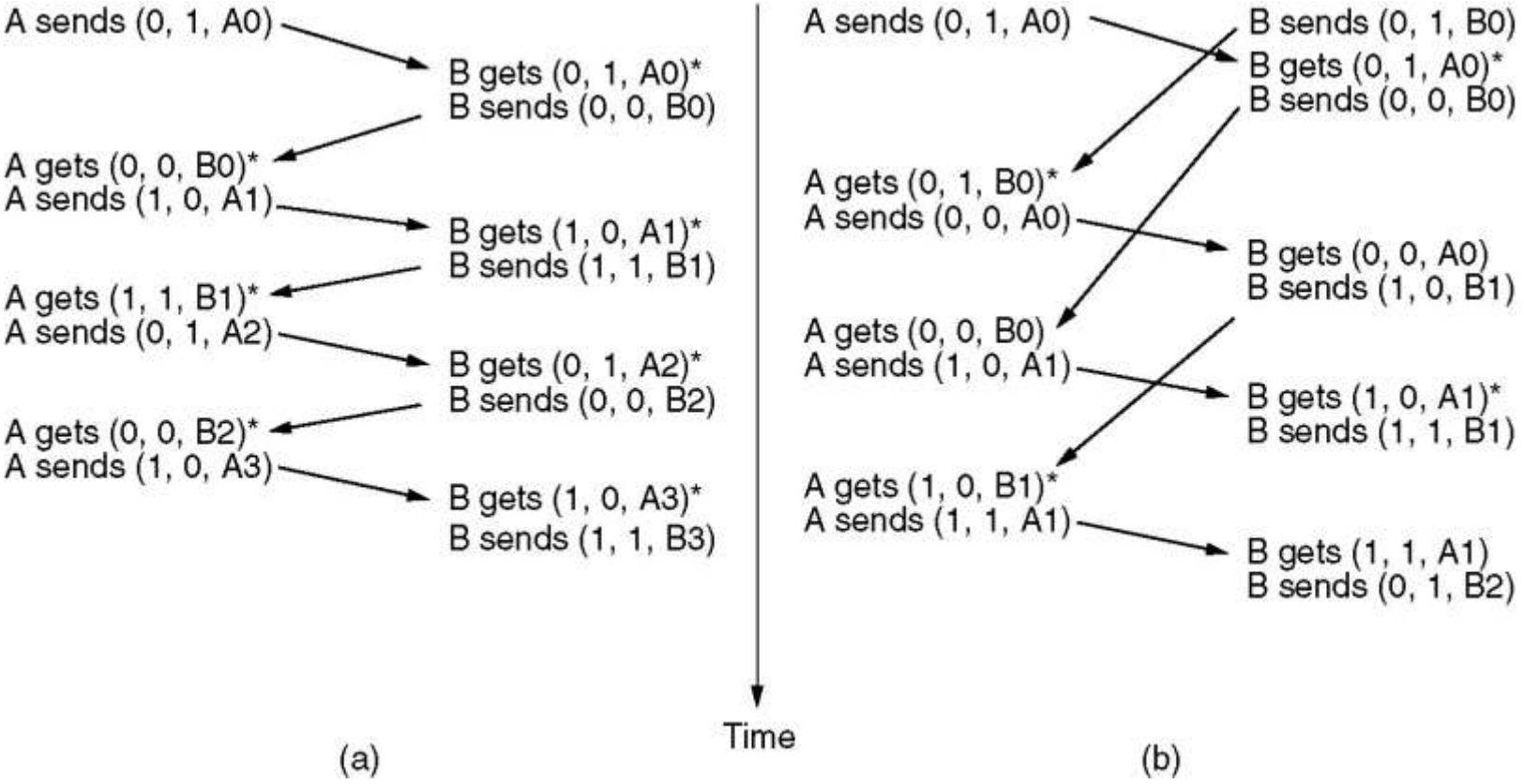
```
/* Protocol 4 (sliding window) is bidirectional. */
#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                        /* 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 */
```

Continued →

A One-Bit Sliding Window Protocol (ctd.)

```
while (true) {
    wait_for_event(&event);                 /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival) {           /* a frame has arrived undamaged. */
        from_physical_layer(&r);            /* go get it */

        if (r.seq == frame_expected) {      /* handle inbound frame stream. */
            to_network_layer(&r.info);      /* pass packet to network layer */
            inc(frame_expected);            /* invert seq number expected next */
        }

        if (r.ack == next_frame_to_send) {  /* handle outbound frame stream. */
            stop_timer(r.ack);              /* turn the timer off */
            from_network_layer(&buffer);    /* fetch new pkt from network layer */
            inc(next_frame_to_send);        /* invert senderís sequence number */
        }
    }
    s.info = buffer;                        /* construct outbound frame */
    s.seq = next_frame_to_send;             /* insert sequence number into it */
    s.ack = 1 – frame_expected;             /* seq number of last received frame */
    to_physical_layer(&s);                  /* transmit a frame */
    start_timer(s.seq);                     /* start the timer running */
  }
}
```

A One-Bit Sliding Window Protocol (2)



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.

3.4.2 A Protocol Using Go Back N

- The long round-trip time can have important implications for the efficiency of the bandwidth utilization.
  - Consider a 50kbps satellite channel with a 500msec round-trip propagation delay.
  - Assume using protocol 4 to send 1000bit frames via satellite.
    - At *t= 0* msec: sending the first frame.
    - At *t= 20* msec: the frame being completely sent.
    - At *t=270* msec: the frame arrived at the receiver.
    - At *t=520* msec: the acknowledgement arrived back at the sender
  - The sender was blocked during 500/520 or 96 percent of the time.

A Protocol Using Go Back N

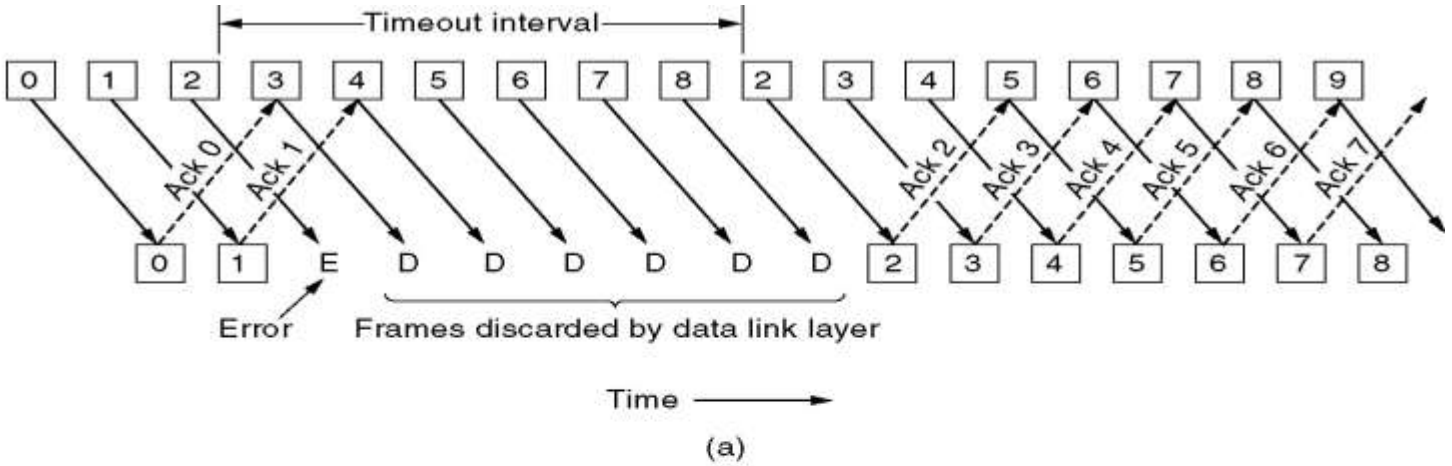
- Why? *For protocol 4, a sender is required to wait for an acknowledgement before sending another frame.*
- The solution lies in allowing the sender to transmit up to *w* frames before blocking instead of just *1*. This technique is known as pipelining.

- Pipelining frames over an unreliable communication channel raises some serious issues.
- There are two basic approaches to dealing with errors in the presence of pipelining.
  - Go back N
  - Selective repeat

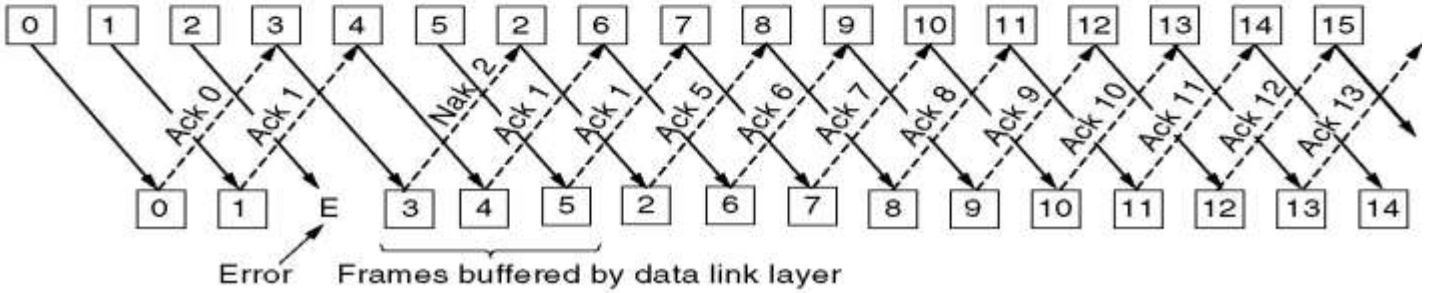
Line utilization (线路利用率)

- This technique is known as pipelining. If the channel capacity is *b* bits/sec, the frame size *l* bits, and the round-trip propagation time *R* sec, the time required to transmit a single frame is l/b sec. After the last bit of a data frame has been sent, there is a delay of R/2 before that bit arrives at the receiver and another delay of at least R/2 for the acknowledgement to come back, for a total delay of R. In stop-and-wait the line is busy for l/band idle for R, giving
- **Line utilization = *l*/(*l*+*bR*)**
- If l < bR, the efficiency will be less than 50 percent.

A Protocol Using Go Back N



(a)



(b)

Pipelining and error recovery. Effect on an error when

(a) Receiver's window size is 1.

(b) Receiver's window size is large.

Sliding Window Protocol Using Go Back N (2)

- 接收窗口大小为W=1
- 抛弃所有后续的帧
- 最多允许有MAX_SEQ帧未被确认
- Stop-and-wait is not fitful for channel which has long round-trip delay, e.g. Satellite channel,270 ms
- Bidirectional transmission, noisy channel,
- Discard restricted condition: network layer of sender has unlimited data to transmit
- size of receiving window = 1, receiver discards all subsequent frames after a damaged or lost data frame.

Sliding Window Protocol Using Go Back N (3)

思考题

- MAX_SEQ+1 distinct sequence numbers(0,1,2,... MAX_SEQ),no more than MAX_SEQ unacknowledged frames,the sender window<=MAX_SEQ. Why? Consider the following scenario with MAX_SEQ=7(a sender window of size eight):
  - The sender sends frames 0 through 7.
  - A piggybacked acknowledgement for frame 7 eventually comes back to the sender.
  - The sender sends another eight frames,again with sequence numbers 0 through 7.
    - All eight frames belonging to the second batch get lost
    - All eight frames belonging to the second batch arrive successfully.
  - Another piggybacked acknowledgement for frame 7 come in, ambiguity will arise to sender:
    - All eight frames belonging to the second batch get lost,ack=7
    - All eight frames belonging to the second batch arrive successfully,ack=7

Sliding Window Protocol Using Go Back N (4)

- A Protocol Using Go Back n(when the size of sender window=7,no ambiguity:
  - The sender sends frames 0 through 6
  - A piggybacked acknowledgement for frame 6 eventually comes back to the sender
  - The sender sends another 7 frames:7,0,1,2,3,4,5,
    - Frame 7,0,1,2,3,4,5 belonging to the second batch get lost
    - Frame 7,0,1,2,3,4,5 belonging to the second batch arrive
  - Another piggybacked ack nowledgement comes in,no ambiguity
    - Frame 7,0,1,2,3,4,5 belonging to the second batch get lost,ack=6
    - Frame 7,0,1,2,3,4,5 belonging to the second batch arrive, ack=5
- the problem :
  - Since a sender may have to retransmit all the unacknowledged frames at a future time ,it must hang on to all transmitted frames until it knows for sure that they have been accepted by the receiver.
  - No acknowledged frames without data.

Sliding Window Protocol Using Go Back N (5)

```
/* Protocol 5 (pipelining) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike 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                       /* should be 2^n – 1 */
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);
}

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 */
}
```

Continued →

Sliding Window Protocol Using Go Back N (6)

```
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;                   /* # output buffers currently in use */
  seq_nr i;                           /* used to index into the buffer array */
  event_type event;

  enable_network_layer();             /* allow network_layer_ready events */
  ack_expected = 0;                   /* next ack expected inbound */
  next_frame_to_send = 0;             /* next frame going out */
  frame_expected = 0;                 /* number of frame expected inbound */
  nbuffered = 0;                      /* initially no packets are buffered */
```

Continued →

Sliding Window Protocol Using Go Back N (7)

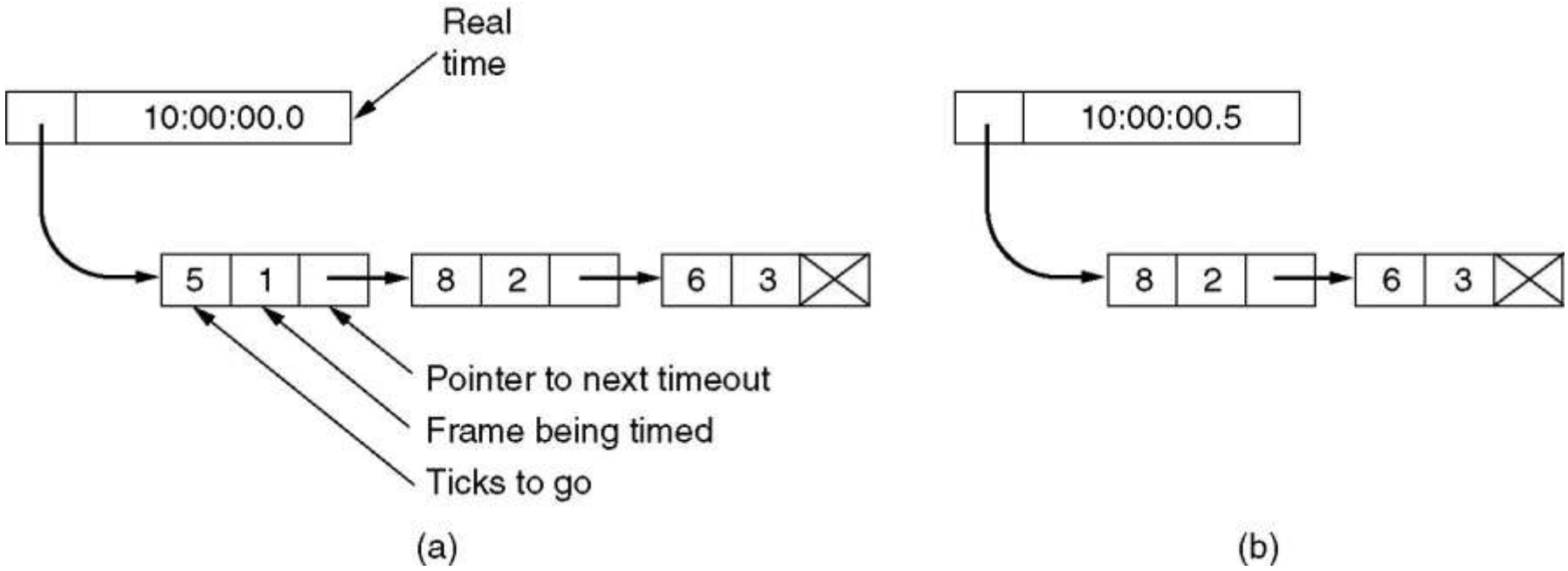
```
while (true) {
  wait_for_event(&event);              /* four possibilities: see event_type above */

  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 */
         }
```

Continued →

Sliding Window Protocol Using Go Back N (8)

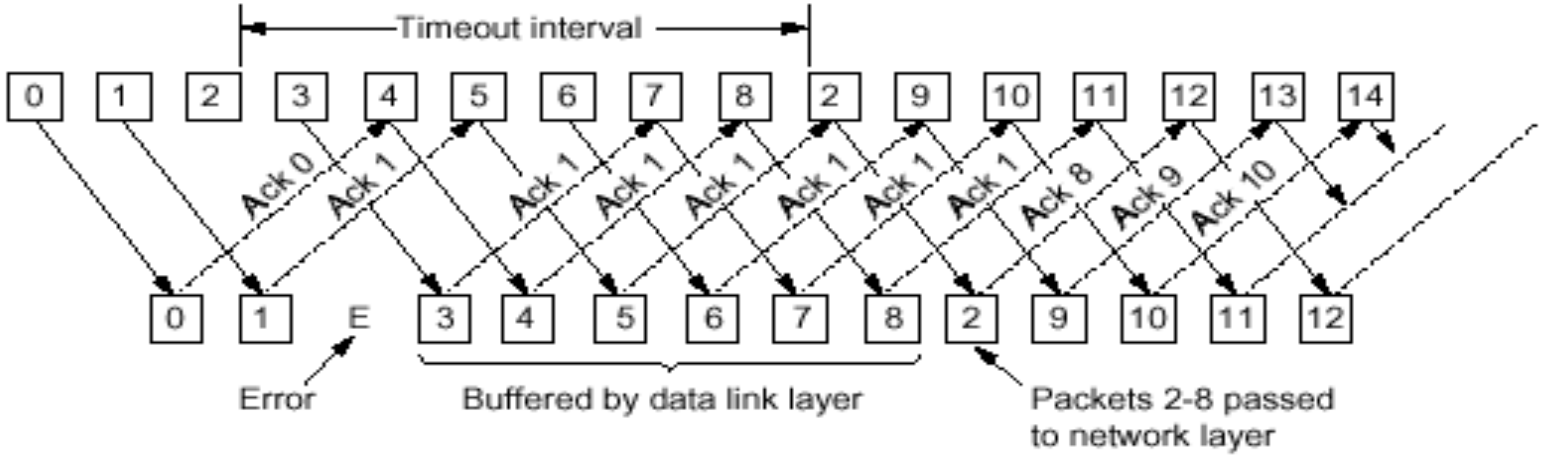
```
            /* Ack n implies n – 1, n – 2, etc.  Check for this. */
            while (between(ack_expected, r.ack, next_frame_to_send)) {
                /* Handle piggybacked ack. */
                nbuffered = nbuffered    1; /* one frame fewer buffered */
                stop_timer(ack_expected); /* 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 1 frame */
                inc(next_frame_to_send);  /* prepare to send the next one */
            }

    }

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

Sliding Window Protocol Using Go Back N (9)



Simulation of multiple timers in software.

3.4.3 A Sliding Window Protocol Using Selective Repeat



(b)

- 允许接收后续的帧（接收窗口 >1）
- **窗口大小为W<=(MAX_SEQ+1)/2**
- 所需缓冲器的数量=W
- 所需计时器的数量=W
- 否定性确认帧-NAK

思考题

A Sliding Window Protocol Using Selective Repeat (2)

```
/* Protocol 6 (nonsequential receive) 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 protocol5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

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 */
}
```

Continued →

A Sliding Window Protocol Using Selective Repeat (3)

```
void protocol6(void)
{
  seq_nr ack_expected;                          /* lower edge of sender's window */
  seq_nr next_frame_to_send;                    /* upper edge of sender's window + 1 */
  seq_nr frame_expected;                        /* lower edge of receiver's window */
  seq_nr too_far;                               /* upper edge of receiver's window + 1 */
  int i;                                        /* index into buffer pool */
  frame r;                                      /* scratch variable */
  packet out_buf[NR_BUFS];                      /* buffers for the outbound stream */
  packet in_buf[NR_BUFS];                       /* buffers for the inbound stream */
  boolean arrived[NR_BUFS];                     /* inbound bit map */
  seq_nr nbuffered;                             /* how many output buffers currently used */
  event_type event;

  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;
```

Continued →

A Sliding Window Protocol Using Selective Repeat (4)

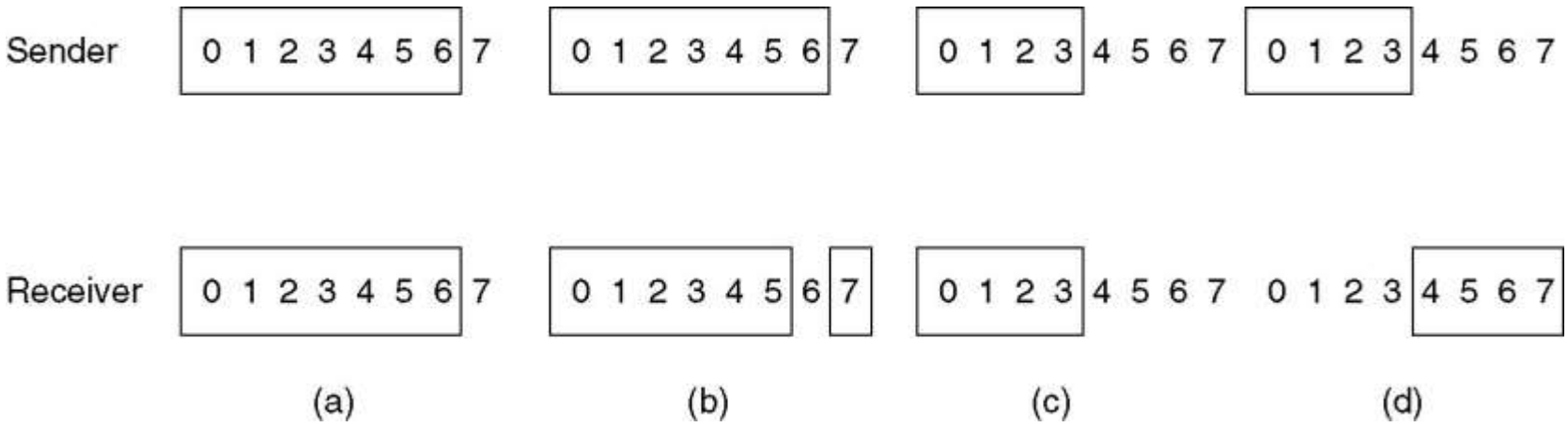
```
while (true) {
  wait_for_event(&event);                          /* five possibilities: see event_type above */
  switch(event) {
    case network_layer_ready:                      /* accept, save, and transmit a new frame */
         nbuffered = nbuffered + 1;                /* expand the window */
         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;

    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 */
                   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 */
                   }
              }
         }
```

Continued →

A Sliding Window Protocol Using Selective Repeat (5)

```
        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;
    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 (6)



(a) Initial situation with a window size 7.

(b) After 7 frames sent and received, but not acknowledged.

(c) Initial situation with a window size of 4.

(d) After 4 frames sent and received, but not acknowledged.

3.5 Example Data Link Protocols

Content in 4th edition:

- HDLC – High-Level Data Link Control
- The Data Link Layer in the Internet （PPP）

Content in 5th edition:

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

Content in 4th edition:

3.5.1 High-Level Data Link Control

- SDLC(IBM' SNA)→ADCCP(ANSI)
  - →HDLC(ISO)→LAP(CCITT)→LAPB(X.25)
  - bit-oriented protocol
  - frame format-[see fig 3-24]
  - flag sequence (01111110)、the minimum frame: three fields、32 bits
  - three kinds of frames: Information、Supervisory、Unnumbered-[see fig 3-25]

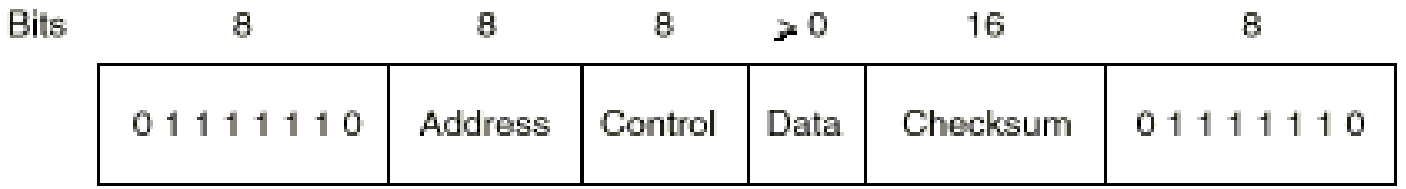


Fig. 3-24. Frame format for bit-oriented protocols.

Content in 4th edition:

High-Level Data Link Control (2)

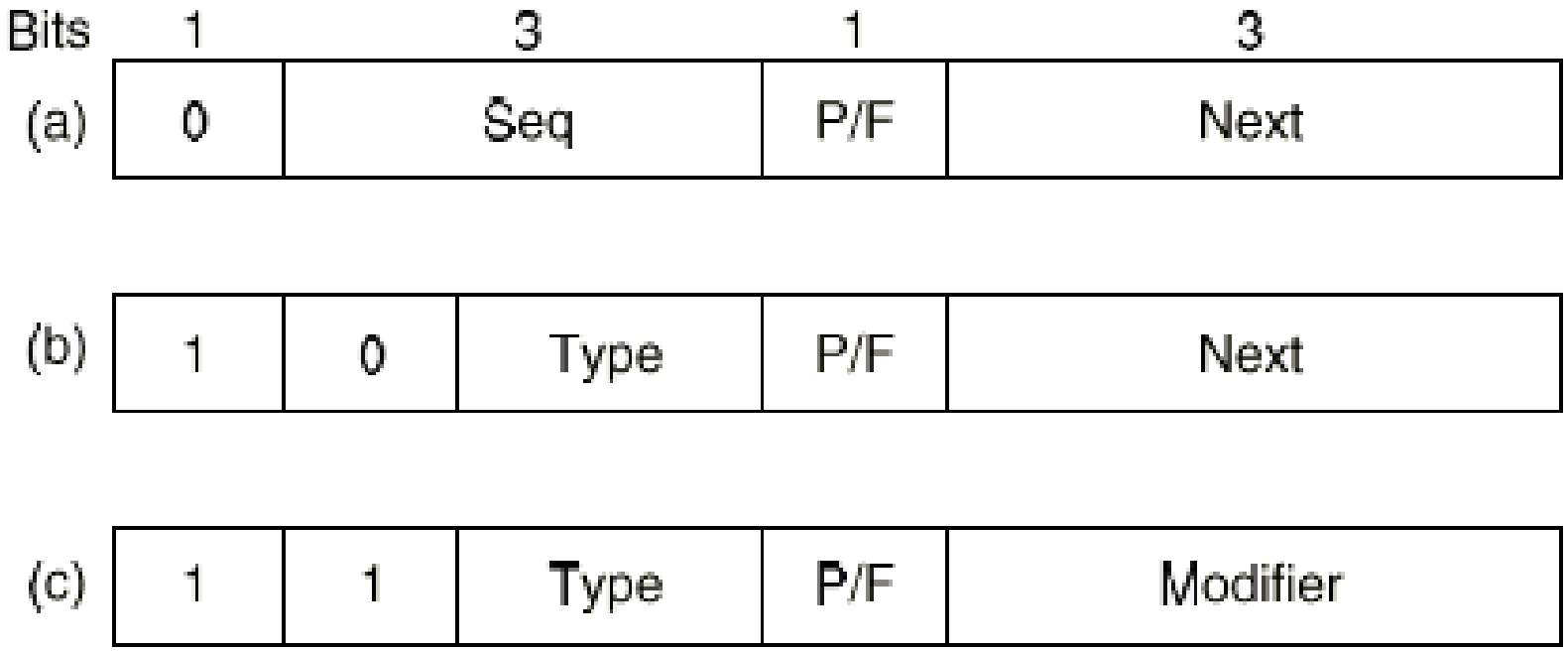


Fig. 3-25. Control field of (a) an information frame, (b) a supervisory frame, (c) an unnumbered frame.

Seq --- sender seq(like former protocol 5)

Next --- expected frame seq, different from former ack,it is (ack+1) mod 8

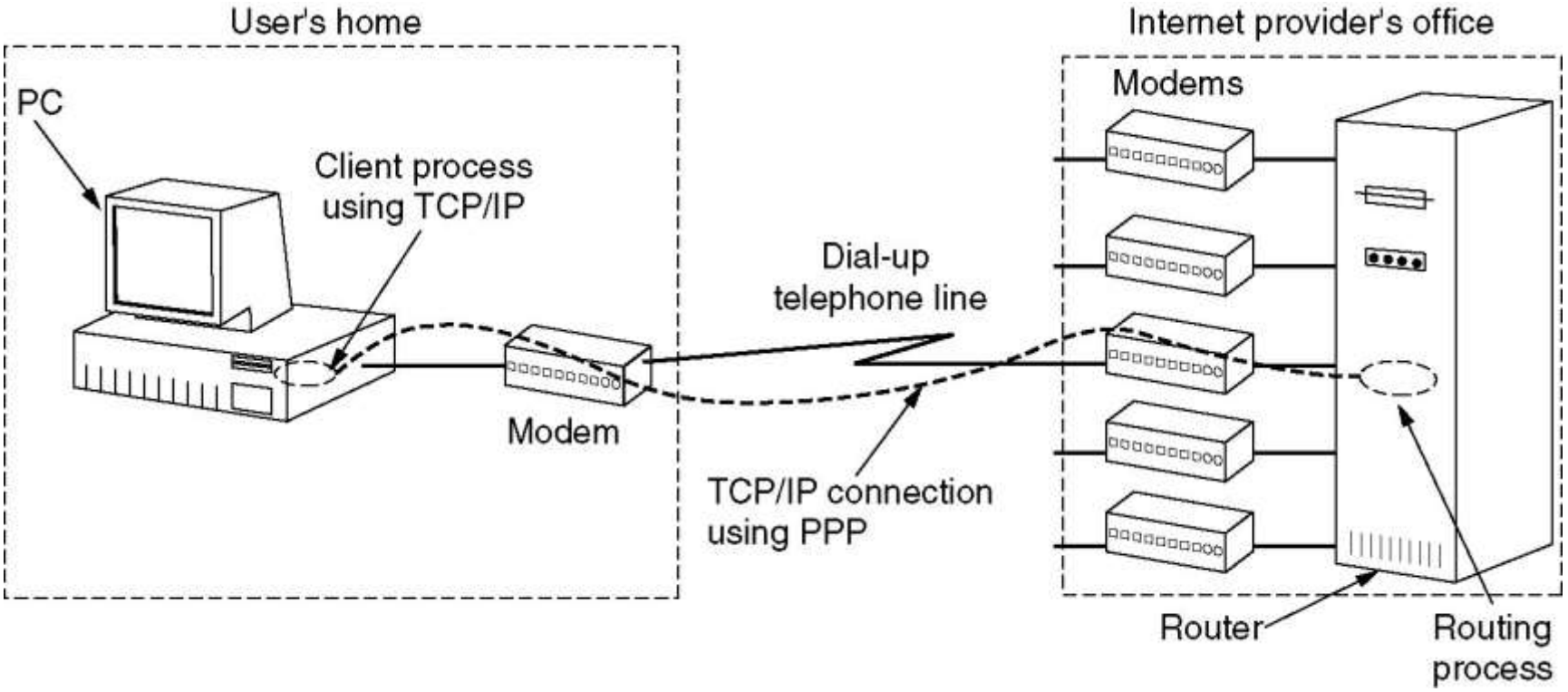
Content in 4th edition:

High-Level Data Link Control (3)

- Information frame(信息帧):
- Supervisory frame(监控帧):
  - RR(Receive Ready),接收准备就绪
  - RNR(Receive Not Ready): to implement flow control
  - REJ(Reject): like NAK frame,从Next起的所有帧都被否认,但确认序号为Next及其以前的各帧
  - SREJ(Selective Reject),选择性拒绝
- Unnumbered frame (无序号帧):
  - request to establish connection: SNRM(Set Normal Response Mode), SABM(set asynchronous balanced mode)
  - request to disconnect: DISC(Disconnect)
  - response frame: UA(unnumbered acknowledgement)
  - report severe error: FRMR(FRaMe Reject)

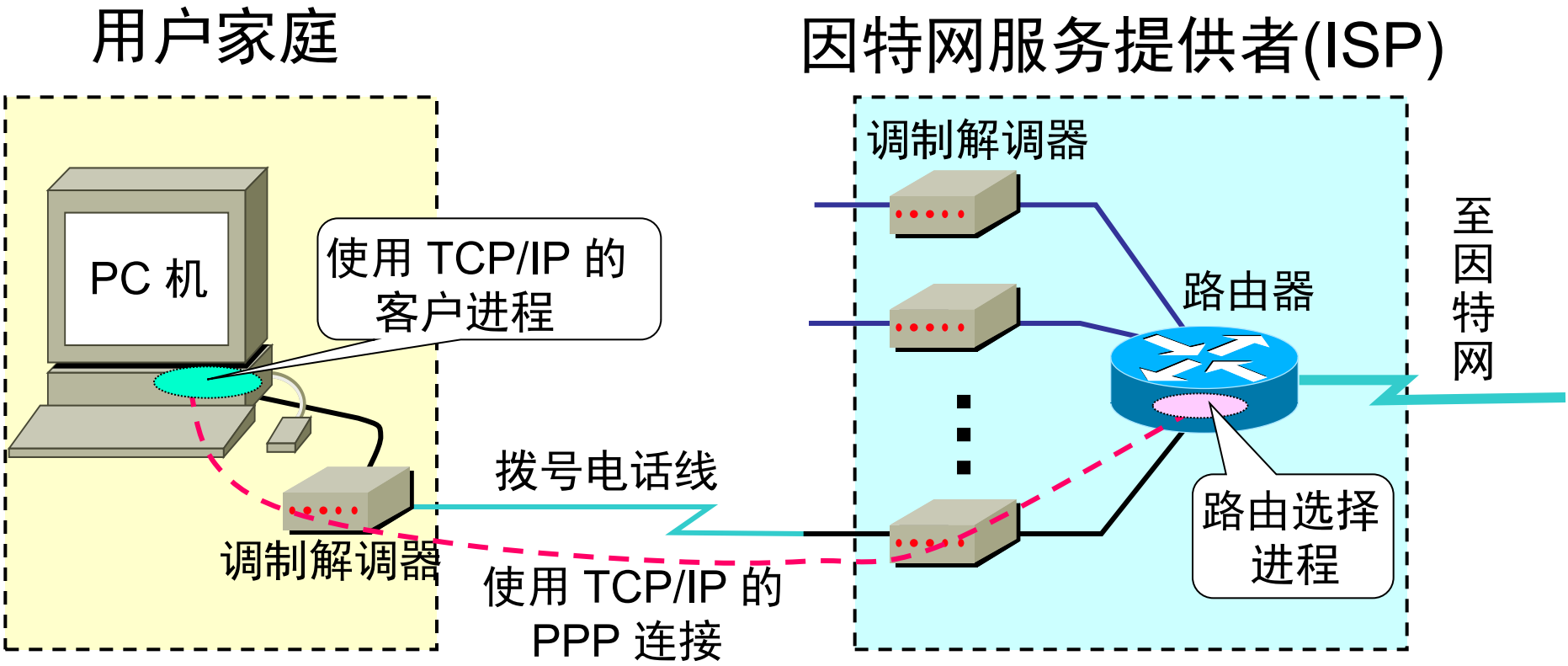
Content in 4th edition:

3.5.2 The Data Link Layer in the Internet



A home personal computer acting as an internet host.

用户拨号入网的示意图


用户家庭
因特网服务提供者(ISP)
调制解调器
PC 机
使用 TCP/IP 的
客户进程
路由器
至
因
特
网
拨号电话线
调制解调器
使用 TCP/IP 的
PPP 连接
路由选择
进程

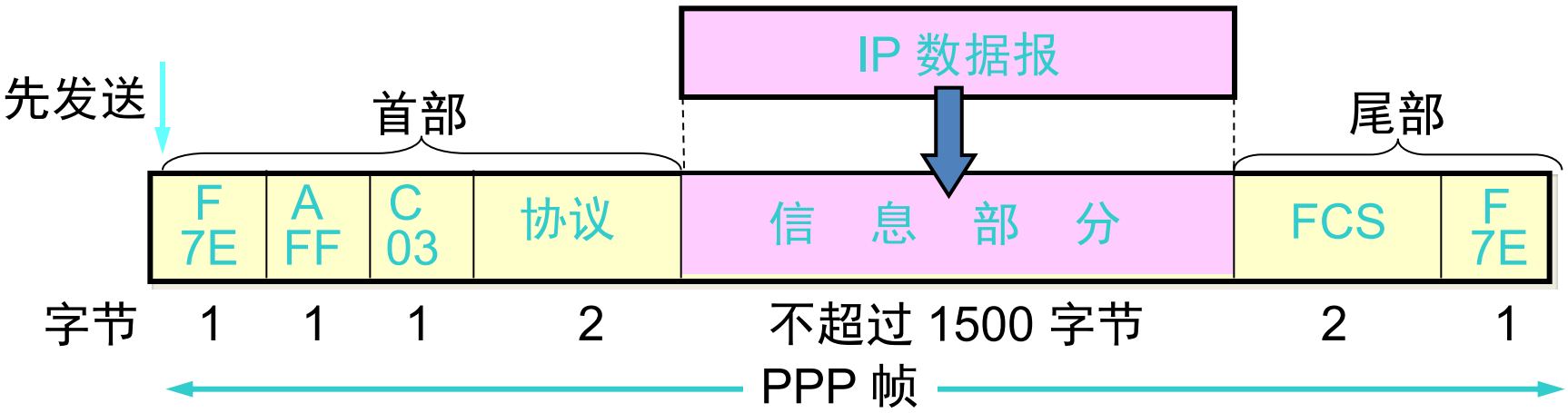
PPP 协议

- 1992 年制订了 PPP 协议。经过 1993 年和 1994 年的修订，现在的 PPP 协议已成为因特网的正式标准[RFC 1661]。
- PPP协议有三个组成部分
  - 一个将 IP 数据报封装到串行链路的方法。
  - 链路控制协议 LCP (Link Control Protocol)。
  - 网络控制协议 NCP (Network Control Protocol)。

PPP 协议的帧格式

- PPP 的帧格式和 HDLC 的相似。
- 标志字段 F 仍为 0x7E （符号“0x”表示后面的字符是用十六进制表示。十六进制的 7E 的二进制表示是 01111110）。
- 地址字段 A 只置为 0xFF。地址字段实际上并不起作用。
- 控制字段 C 通常置为 0x03。
- PPP 是面向字节的，所有的 PPP 帧的长度都是整数字节。

PPP 协议的帧格式



- PPP 有一个 2 个字节的协议字段。
  - 当协议字段为 0x0021 时，PPP 帧的信息字段就是IP 数据报。
  - 若为 0xC021, 则信息字段是 PPP 链路控制数据。
  - 若为 0x8021，则表示这是网络控制数据。

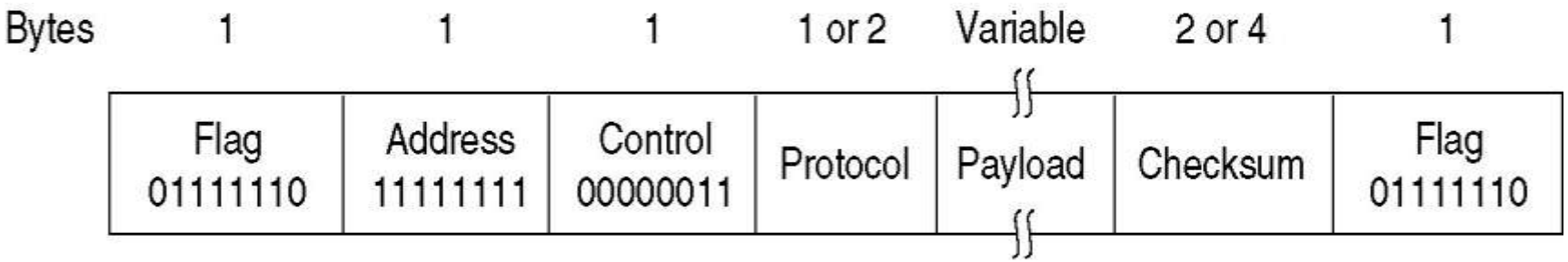
Content in 4th edition:

PPP—Point to Point Protocol (2)
（点到点协议）

- SLIP → PPP
- Support： error detection、multiple protocols(IP,IPX,Apple Talk,XNS,...),allows IP address to be negotiated、permits authentication…
- LCP(Link Control Protocol): byte oriented and bit-oriented encodings
- NCP(Network Control Protocol).
- Frame format(as HDLC):-[see fig 3-27]
- A simplified phase diagram for bringing a line up and down-[see fig 3-28]

Content in 4th edition:

PPP – Point to Point Protocol (3)



The PPP full frame format for unnumbered mode operation.

Flag: 01111110

Address: always 11111111

Control: default 00000011 (unnumbered mode)

Protocol: default size is 2 bytes, tell what kind of packet is in the *Playload* field

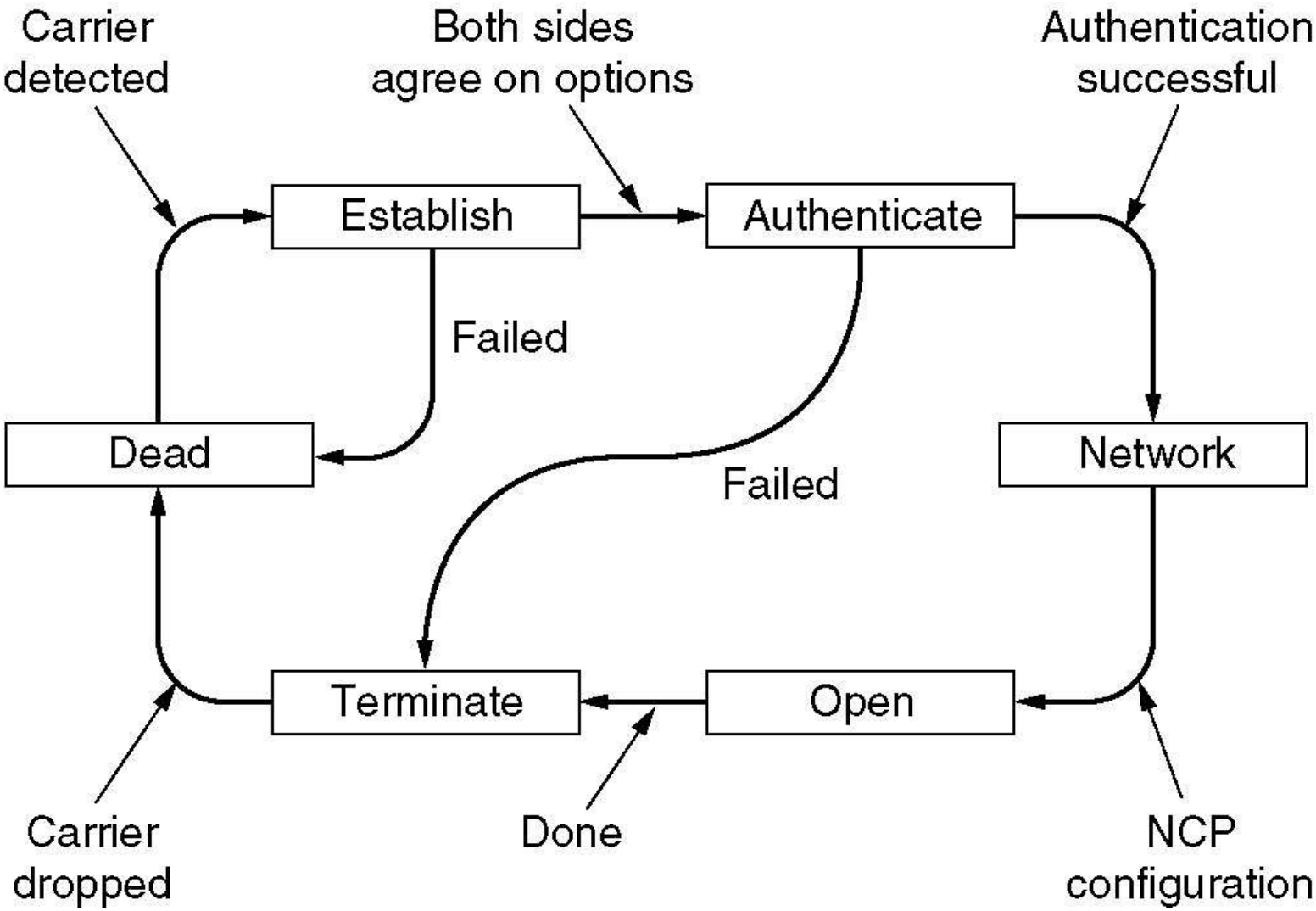
protocols starting with a 0 bit are network lay protocols such as IP, IPX,OSL CLNP, XNS; Those starting with a 1 bit are used to negotiate other protocols: LCP or other NCP

Payload: variable length, up to some negotiated maximum; default length of 1500 bytes is used for LCP line setup

Checksum: 2 or 4 bytes

Content in 4th edition:

PPP – Point to Point Protocol (4)



A simplified phase diagram for bring a line up and down.

Content in 4th edition:

PPP – Point to Point Protocol (5)

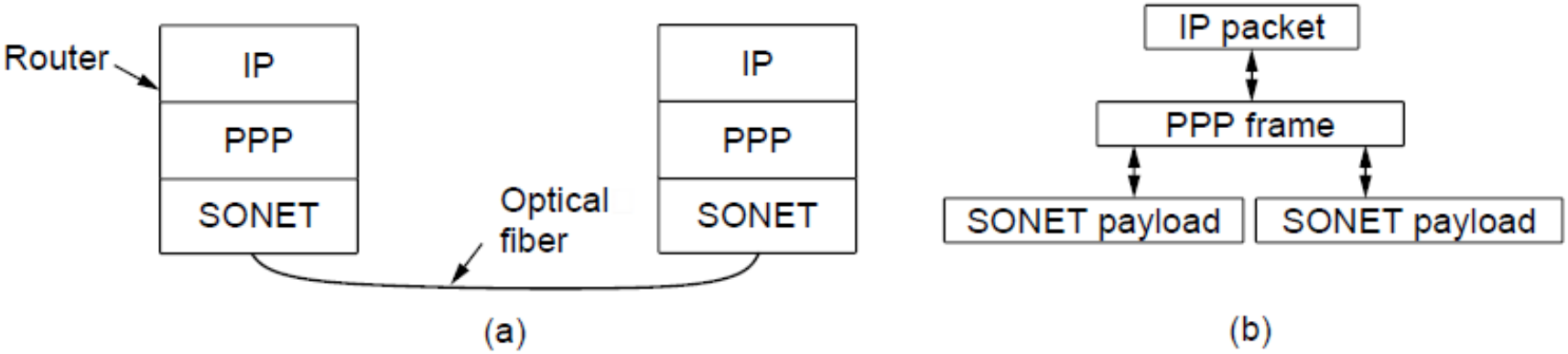
Name	Direction	Description
Configure-request	I → R	List of proposed options and values
Configure-ack	I ← R	All options are accepted
Configure-nak	I ← R	Some options are not accepted
Configure-reject	I ← R	Some options are not negotiable
Terminate-request	I → R	Request to shut the line down
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request received
Protocol-reject	I ← R	Unknown protocol requested
Echo-request	I → R	Please send this frame back
Echo-reply	I ← R	Here is the frame back
Discard-request	I → R	Just discard this frame (for testing)

The LCP frame types.

Note: Initiator(I) Responder(R)

Content in 5th edition:

3.5.1 Packet over SONET (1)



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

Content in 5th edition:

Packet over SONET (2)

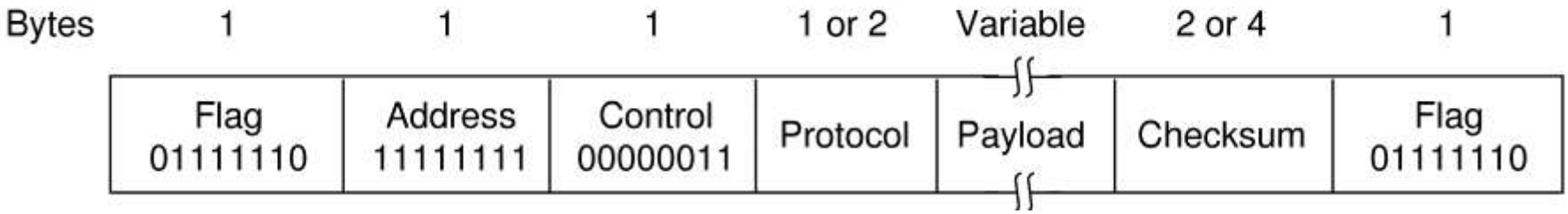
PPP Features

1. Separate packets, error detection
2. Link Control Protocol
3. Network Control Protocol

Content in 5th edition:

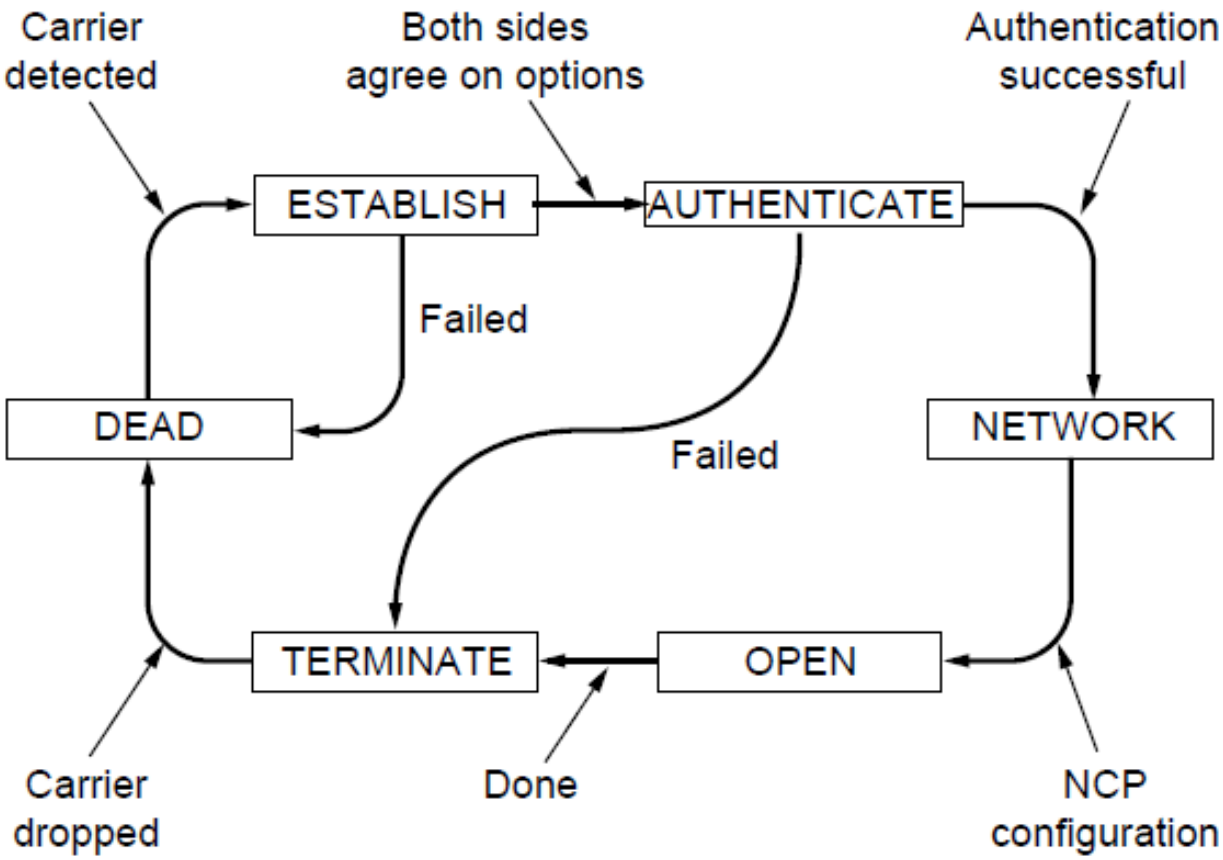
Packet over SONET (3)

The PPP full frame format for unnumbered mode operation



Content in 5th edition:

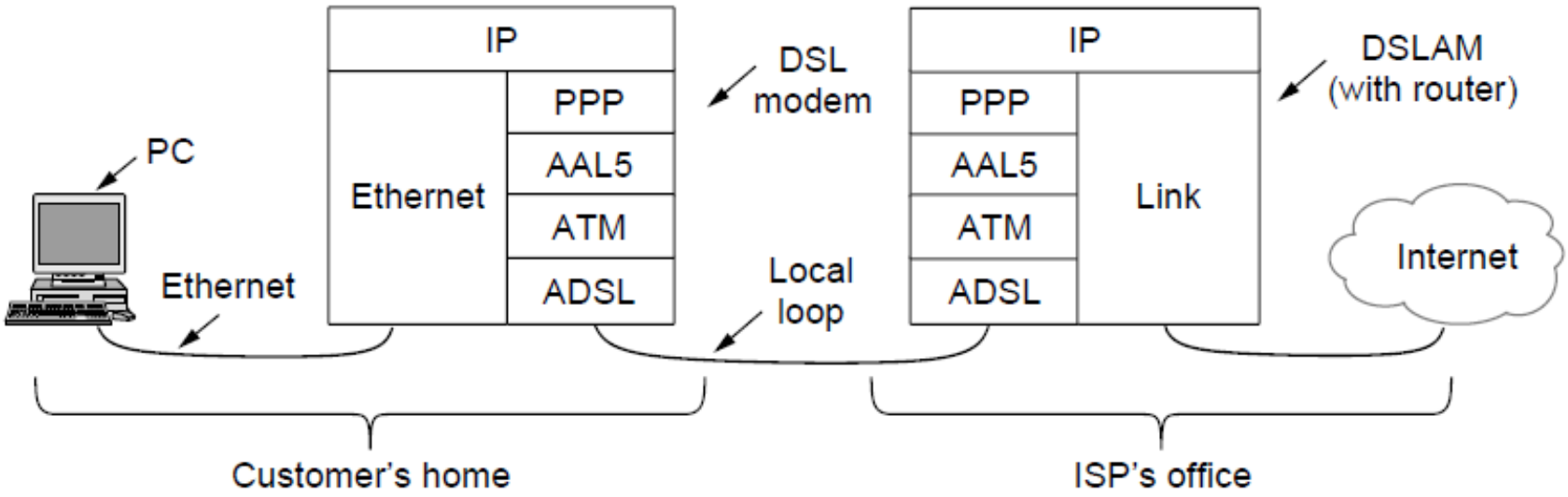
Packet over SONET (4)



State diagram for bringing a PPP link up and down

Content in 5th edition:

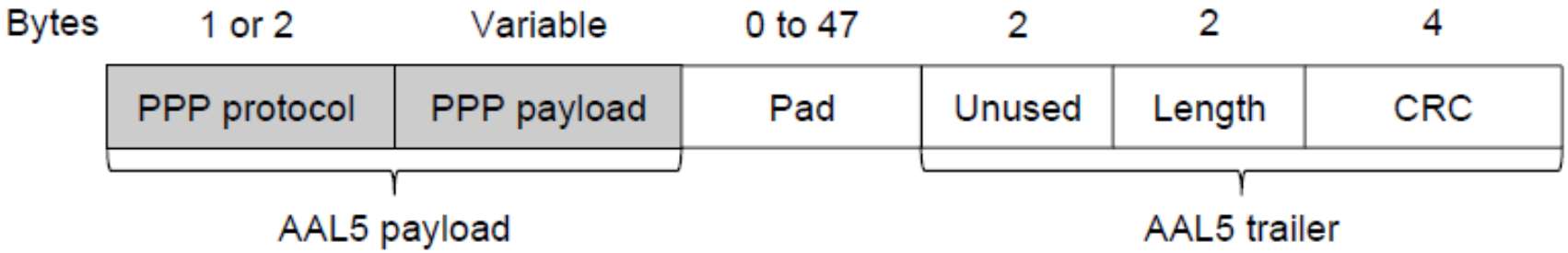
3.5.2 ADSL (Asymmetric Digital Subscriber Loop) (1)



ADSL protocol stacks.

Content in 5th edition:

ADSL (Asymmetric Digital Subscriber Loop) (1)



AAL5 frame carrying PPP data

3.6 Summary

- The task of the data link layer
- The size of window、 the sliding window
- stop-wait protocol、sliding window protocol
- Example of the data link layer：
  - **bit-oriented**：SDLC、HDLC、ADCCP or LAPB
  - **byte-oriented and bit-oriented**：SLIP and PPP（Internet）

Exercises

In 4th Edition：

- 2，3，5，14-18，
- 27，29，32，37

In 5th Edition：

- 3，7，13-16，18，20，
- 27，29，31，33