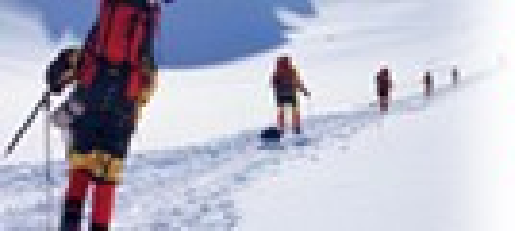


Chapter 11

Data Link Control
数据链路控制

数据链路控制的功能包括成帧、流量控制和差错控制，以及能提供帧在节点之间流畅且可靠传输的软件实现协议。差错检测已在第十章介绍了。

数据链路层两个主要功能：

■数据链路控制，用来处理两个相邻节点之间的通信；

■介质访问控制， 如何共享媒介进行访问控制。

11-1 FRAMING 成帧

数据链路层需要将一组比特位组成帧，以便帧和帧之间是可以识别的。成帧是将一条从源端到目的端的报文分离开，或者将到不同目的端的报文分离开。邮局系统就实现了成帧方式，信封就是一种分隔符。

Topics discussed in this section:

Fixed-Size Framing 固定长度成帧 例如第 18 章的 ATM 信元

Variable-Size Framing 可变长度成帧 面向字符和面向比特位

成帧的方式

- 固定大小成帧
- 可变长度成帧
- 面向字符方式
- 面向位的方式

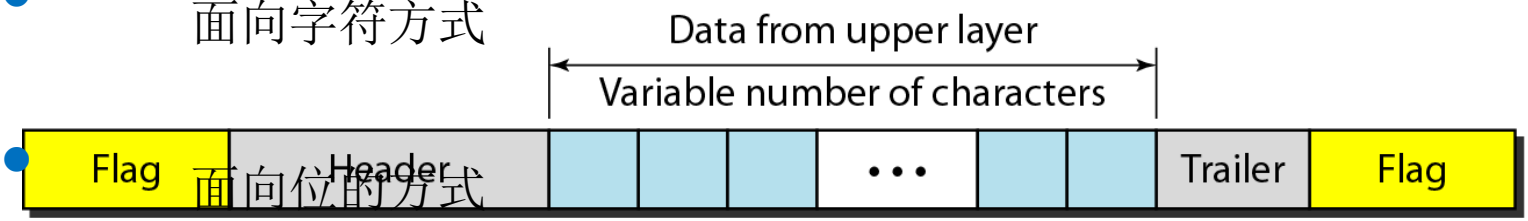


Figure 11.1　面向字符的帧

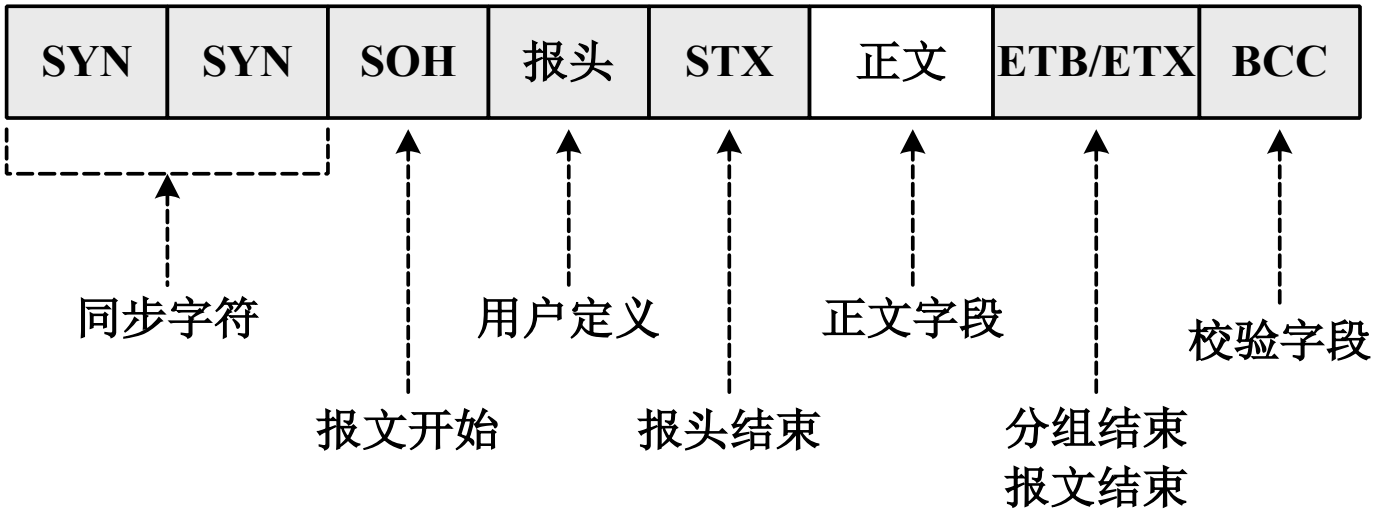
数据链路层协议的分类

- 数据链路层协议可分为两类：面向字符型与面向比特型。
- 最早出现的数据链路层协议是面向字符型协议，典型的面向字符型数据链路层协议是二进制同步通信（ BSC ）协议。
- 面向字符型使用已定义的某种编码（例如 ACSII 码）的子集来执行通信控制，可利用 ASCII 码中的 10 个控制字符完成通信控制，并规定了数据与控制报文的格式，以及协议操作过程。
- 面向字符型的缺点是采用不同字符集的计算机之间难以通信，而且控制字符编码不能出现在数据字段中，否则会引起通信控制错误。
- 面向比特型协议主要包括：同步数据链路控制（ SDLC ）协议与高级数据链路控制（ HDLC ）协议。

BSC 协议使用的控制字符

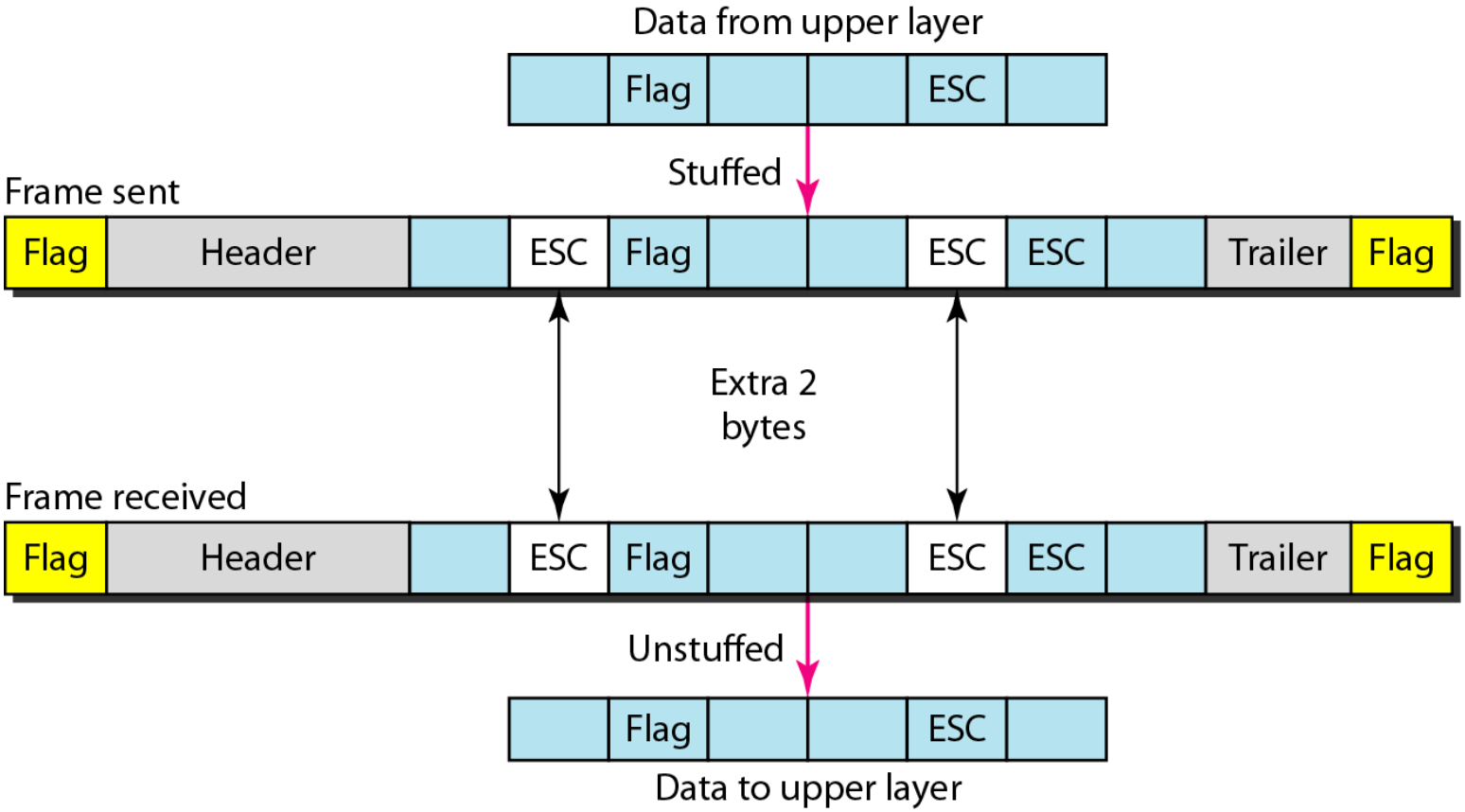
控制字符	功能
SOH （ Start of Head ）	报头开始
STX （ Start of Text ）	正文开始
ETX （ End of Text ）	正文结束
EOT （ End of Transmission ）	传输结束
ENQ （ Enquiry ）	询问对方，要求回答
ACK （ Acknowledge ）	肯定应答
NAK （ Negative Acknowledge ）	否定应答
DLE （ Data link Escape ）	转义字符
SYN （ Synchronous ）	同步
ETB （ End of Transmission Block ）	正文信息组结束

BSC 报文格式



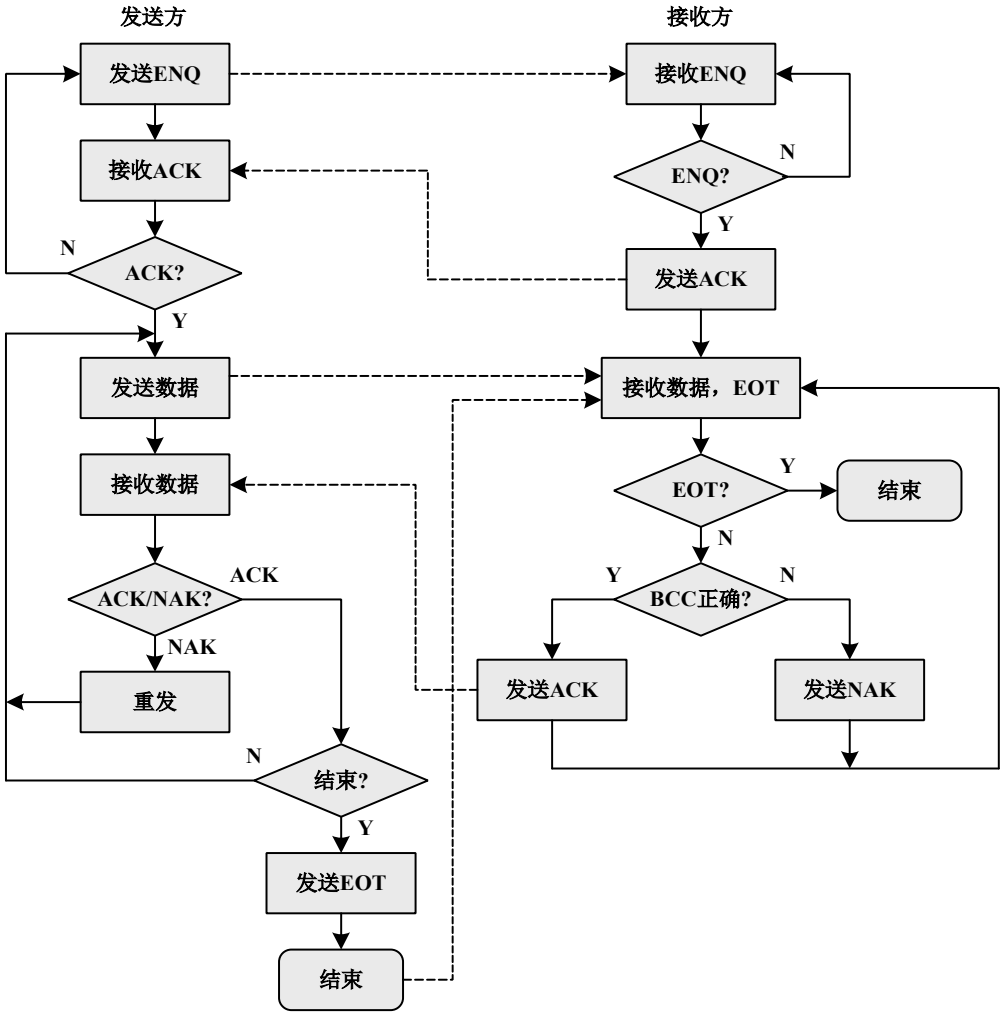
字节填充：当数据中出现与控制字符相同的编码时，添加一个额外字节的过程。

Figure 11.2 字节填充和移除



BSC 协议执行过程

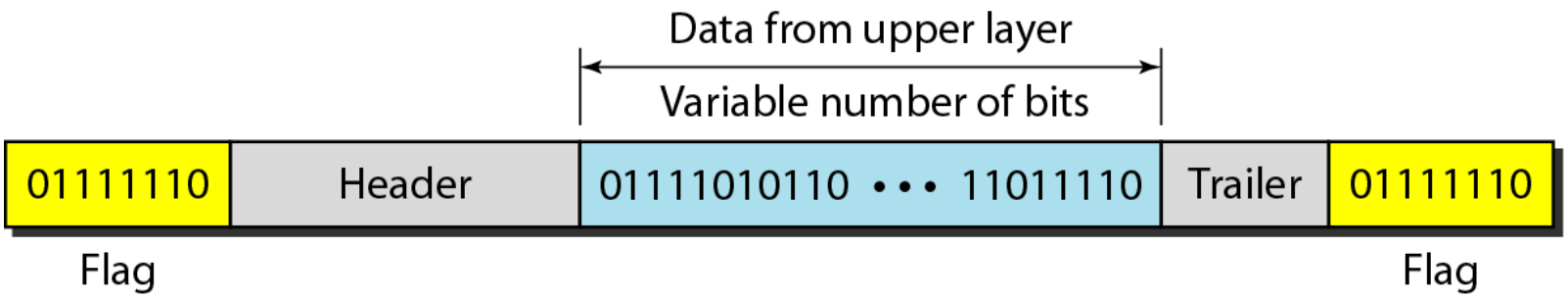
控制字符	功能
SOH	报头开始
STX	正文开始
ETX	正文结束
EOT	传输结束
ENQ	询问对方，要求回答
ACK	肯定应答
NAK	否定应答
DLE	转义字符
SYN	同步
ETB	正文信息组结束



面向字符型链路控制协议的缺点

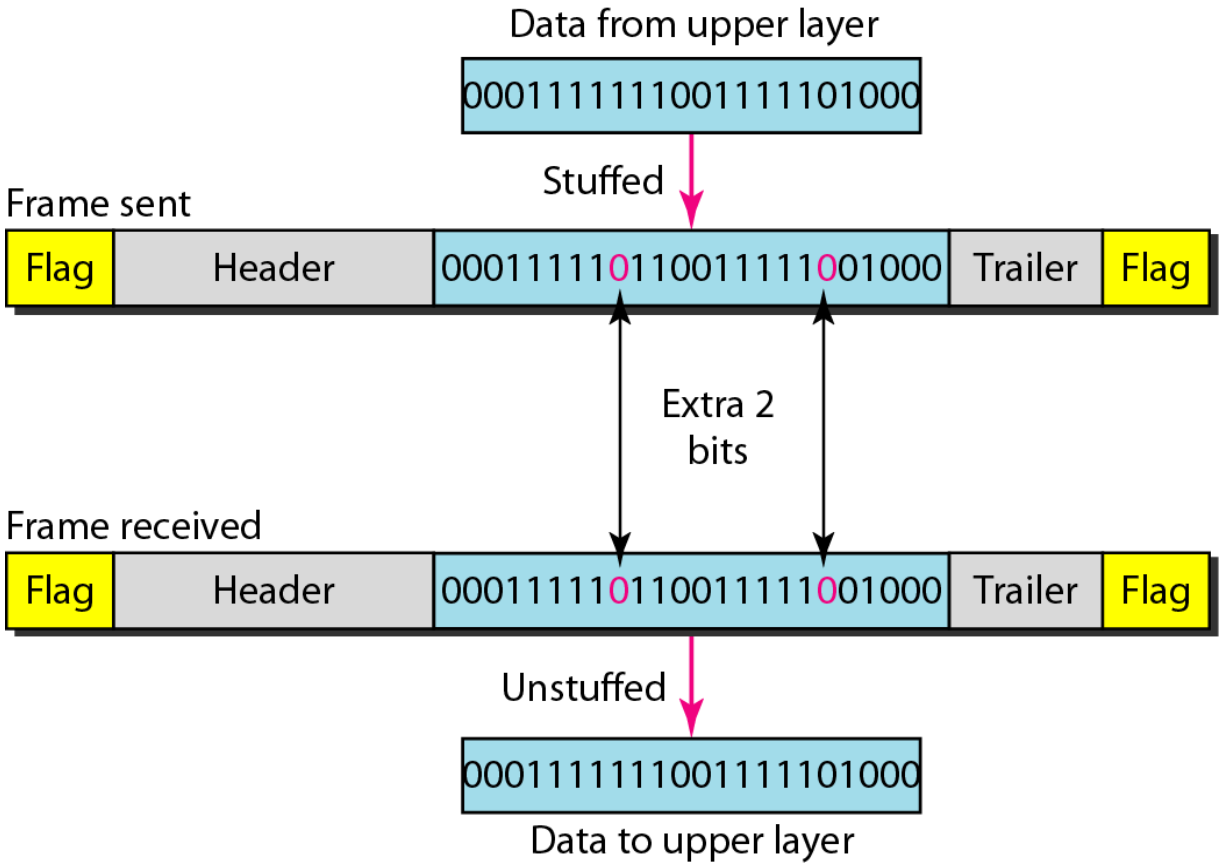
- 控制报文与数据报文的格式不一致。
- 协议规定通信双方采用停止等待方式，收发双方只能交替工作，协议效率低，通信线路的利用率低。
- 协议只对数据部分进行差错控制，如果控制字符出错无法控制，系统可靠性较差。
- 系统每增加一种新的功能，需要设定一个新的控制字符，因此功能扩展困难。

Figure 11.3 面向比特位的帧



位填充：遇到 1 个 0 后面连续 5 个 1 时，插入一个比特 0 的过程，使得接收方不会误认为 0111110 是帧的标记。

Figure 11.4 位填充和移除



11-2 FLOW AND ERROR CONTROL
流量控制和差错控制

数据链路层最重要的职能就是流量控制和差错控制。通常一起称为数据链路控制。

Topics discussed in this section:

Flow Control 流量控制
Error Control 差错控制

流量控制就是一系列程序，用来限制发送方在等到确认之前发送的数据数量。

数据链路层的差错控制基于自动重复请求（Automatic Repeat Request, ARQ）即重传数据。

11-3 PROTOCOLS 几种协议

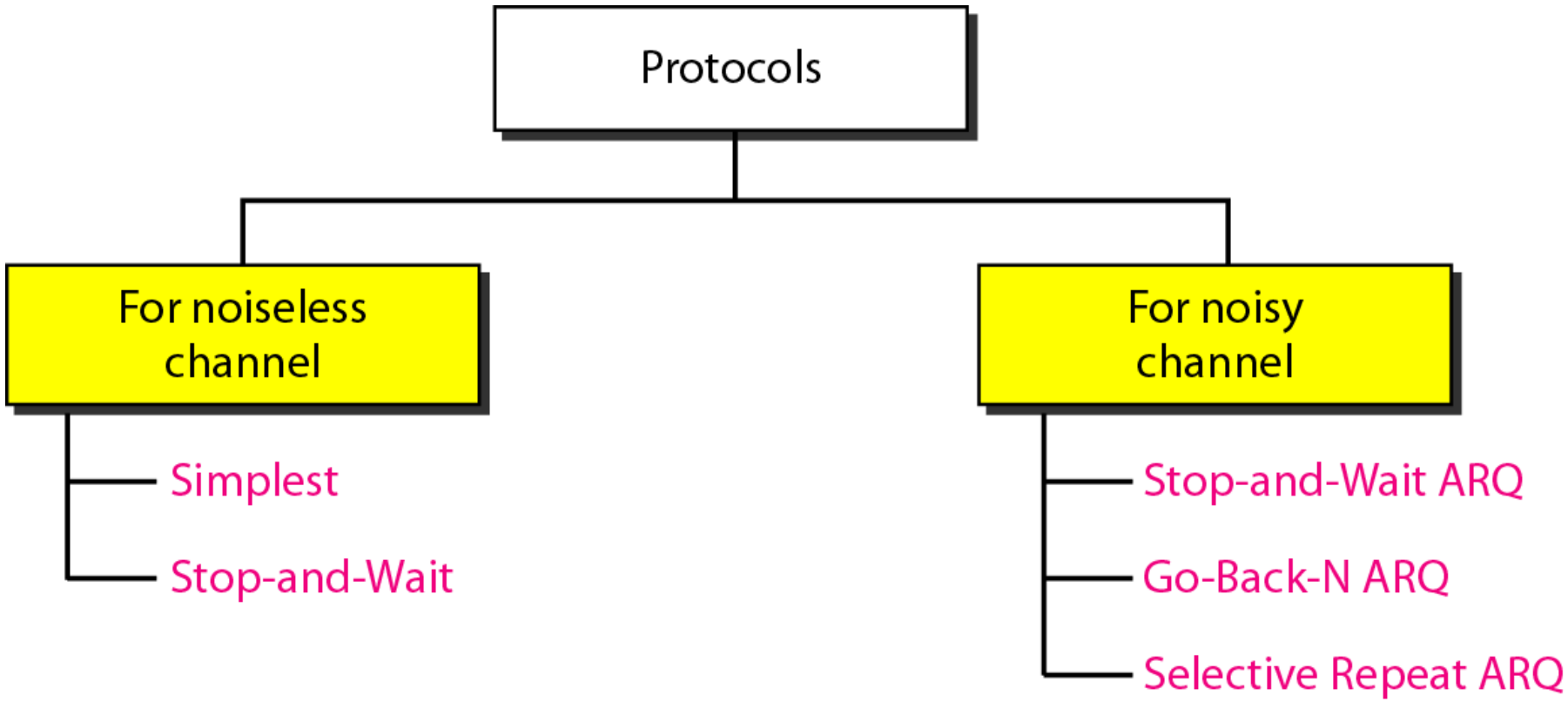
数据链路层如何将成帧、流量控制和差错控制结合起来，实现节点到节点间的数据传输，通常用某种通用的程序语言实现相关协议。这里采用伪代码方式，主要关注数据链路实现的过程，而非程序语言实现的细节。

确认帧： ACK （ Acknowledgement ）

否定确认帧： NAK （ Negative Acknowledgement ）

（也称为否定应答或非应答，表明确认数据收到但有小错误存在，请求对方重新发送。当正确接收后返回的信号应该是 ACK 。）

Figure 11.5 几种协议分类



11-4 NOISELESS CHANNELS 无噪声信道

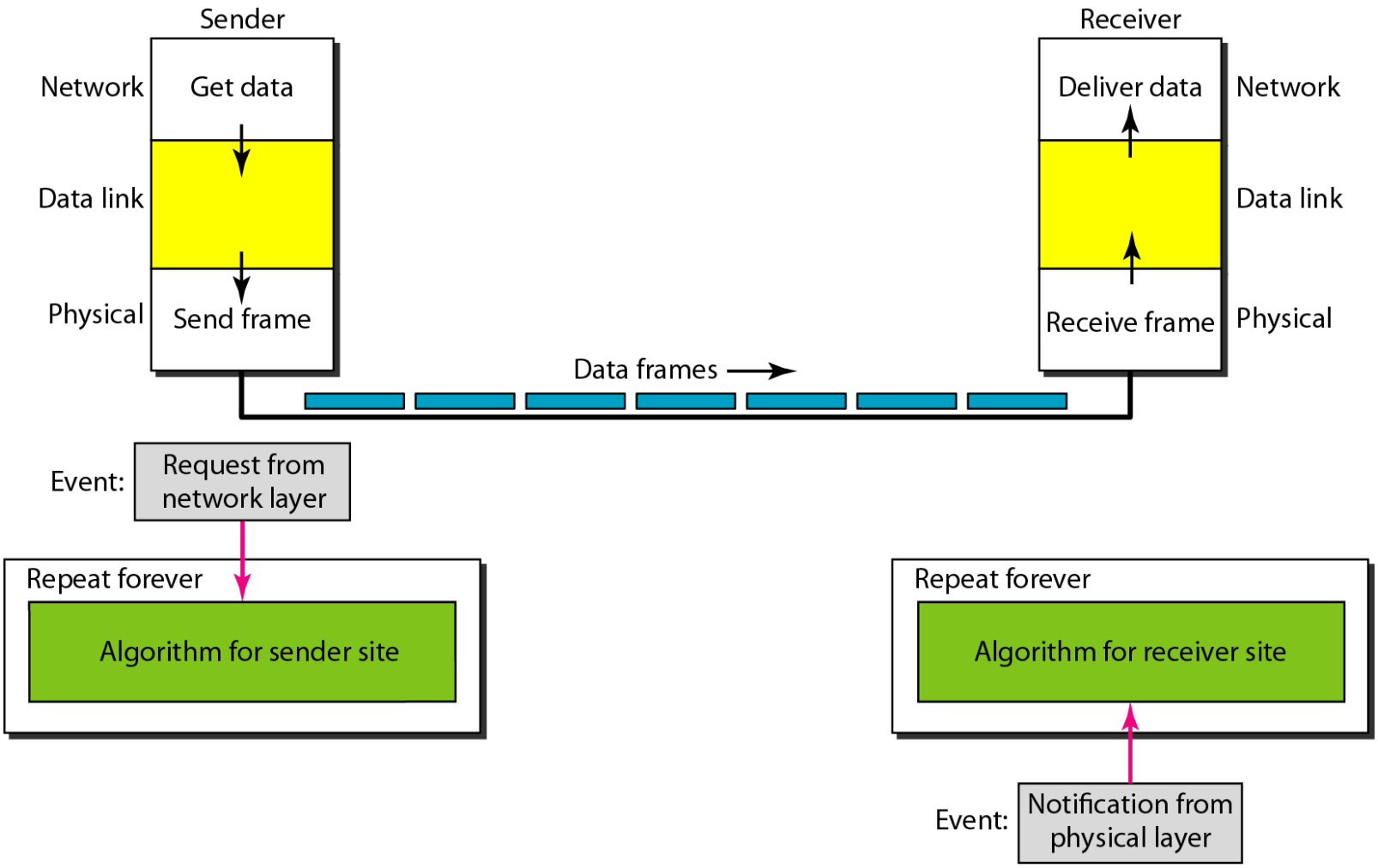
假定有一种不会丢失帧、复制帧或损坏帧的理想信道。包括两种协议：不使用流量控制和使用流量控制的协议，但两种协议均不使用差错控制。

Topics discussed in this section:

Simplest Protocol 最简单的协议

Stop-and-Wait Protocol 停等协议

Figure 11.6 没有流量控制和差错控制的最简单协议



Algorithm 11.1 最简单协议中发送方的算法

```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(RequestToSend))           //There is a packet to send
5    {
6       GetData();
7       MakeFrame();
8       SendFrame();                    //Send the frame
9    }
10 }
```

Algorithm 11.2 最简单协议中接收方的算法

```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrived
5    {
6       ReceiveFrame();
7       ExtractData();
8       DeliverData();                  //Deliver data to network layer
9    }
10 }
```

Example 11.1

使用最简单协议的例子。发送方发送一个帧序列而不用考虑接收方。发了三帧，在发送方和接收方各发生了三个事件。

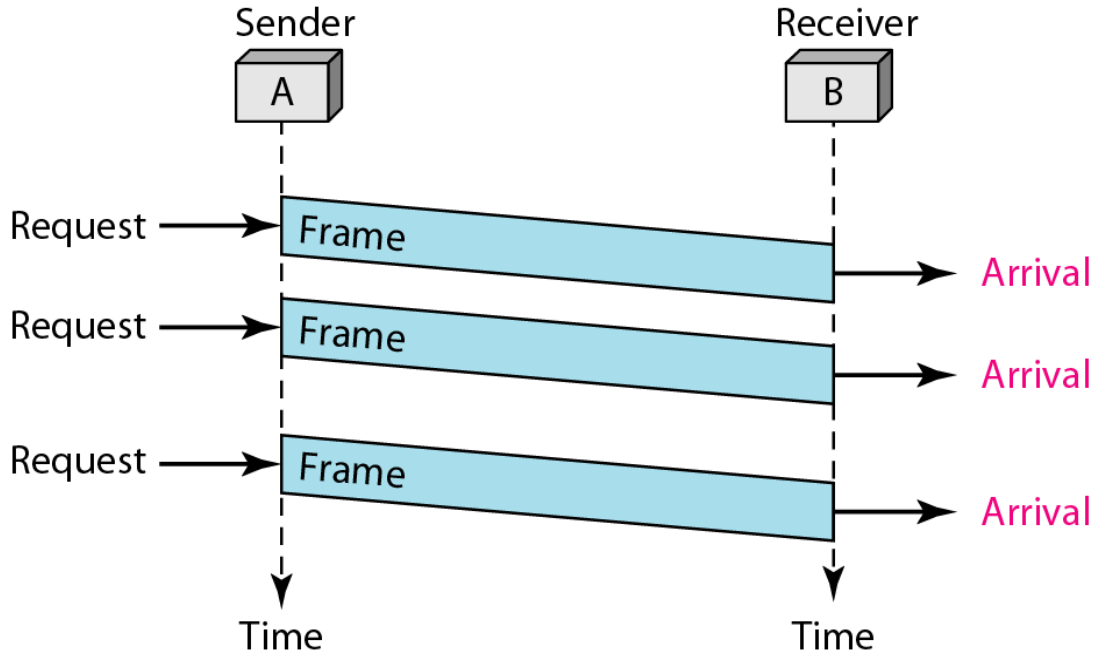
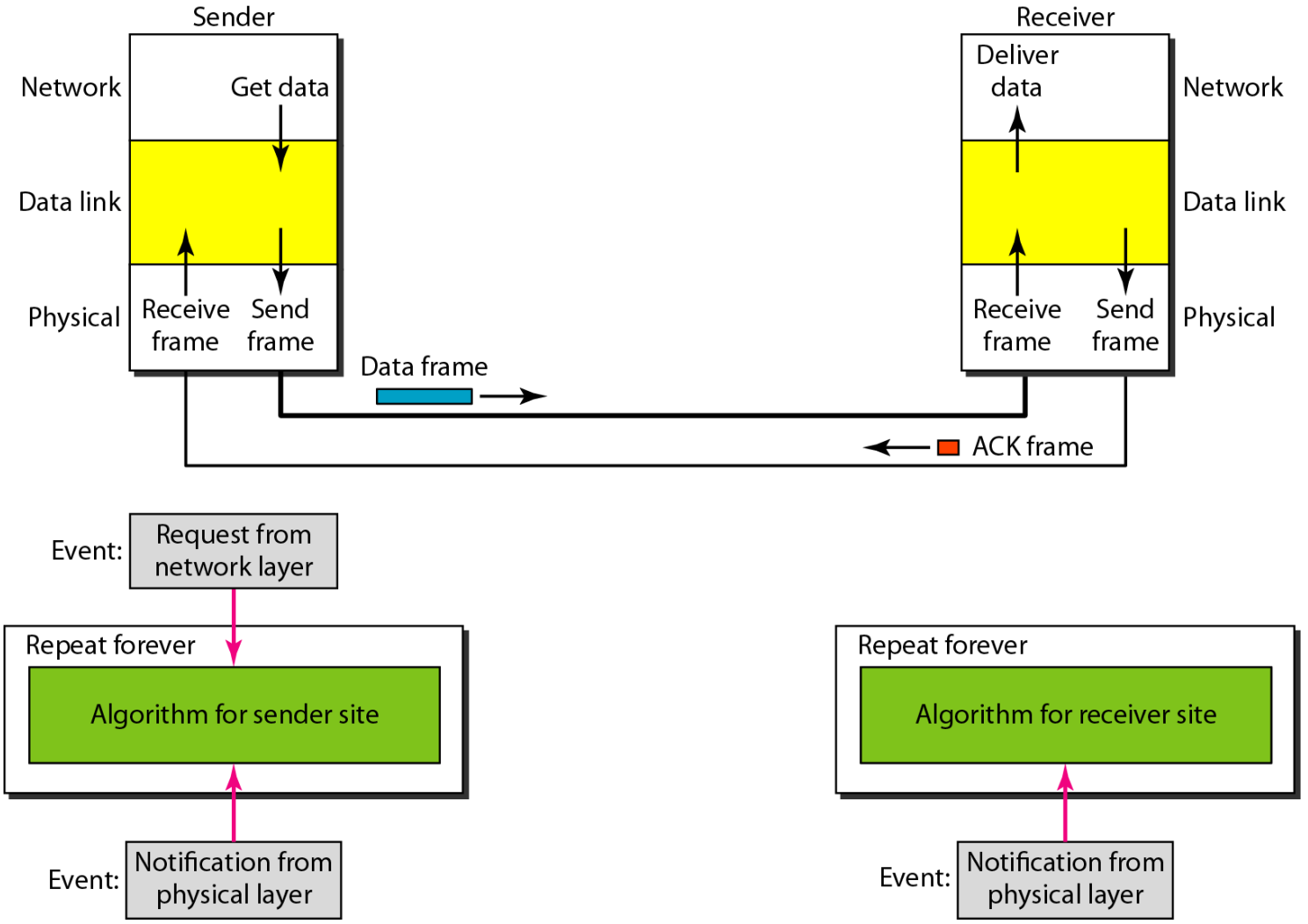


Figure 11.7 三个帧的流量图

Figure 11.8 停等协议的设计



Algorithm 11.3 停等协议的发送方的算法

```
1  while(true)                          //Repeat forever
2  canSend = true                       //Allow the first frame to go
3  {
4    WaitForEvent();                    // Sleep until an event occurs
5    if(Event(RequestToSend) AND canSend)
6    {
7       GetData();
8       MakeFrame();
9       SendFrame();                    //Send the data frame
10      canSend = false;                //Cannot send until ACK arrives
11   }
12   WaitForEvent();                    // Sleep until an event occurs
13   if(Event(ArrivalNotification)      // An ACK has arrived
14    {
15      ReceiveFrame();                 //Receive the ACK frame
16      canSend = true;
17    }
18 }
```

Algorithm 11.4 停等协议的接收方算法

```
1  while(true)                          //Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrives
5    {
6       ReceiveFrame();
7       ExtractData();
8       Deliver(data);                  //Deliver data to network layer
9       SendFrame();                    //Send an ACK frame
10   }
11 }
```

Example 11.2

使用停等协议的例子。发送方发送一帧后要等待接收方 ACK 帧的反馈确认，才能发送下一帧。

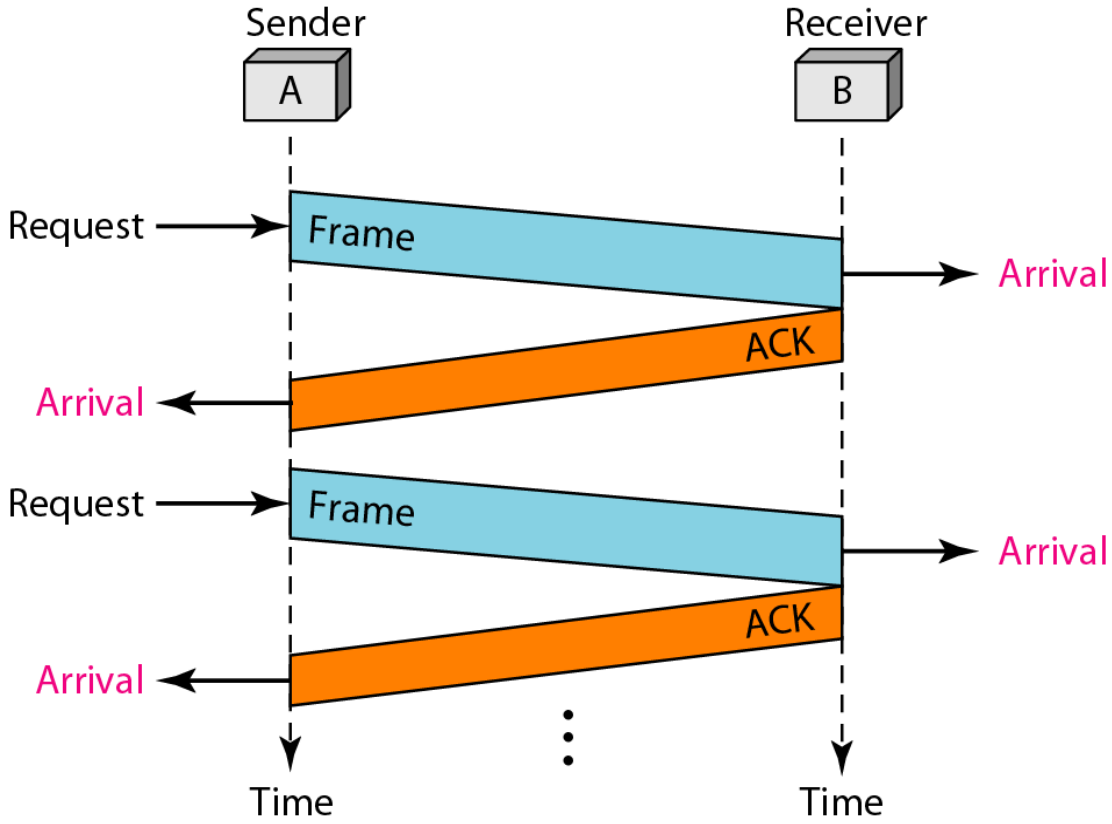


Figure 11.9 停等协议的帧流量图

11-5 NOISY CHANNELS 有噪声信道

尽管停等协议增加了流量控制的概念，但无噪声信道是理想的，不存在的。需要讨论三个使用差错控制的链路协议。

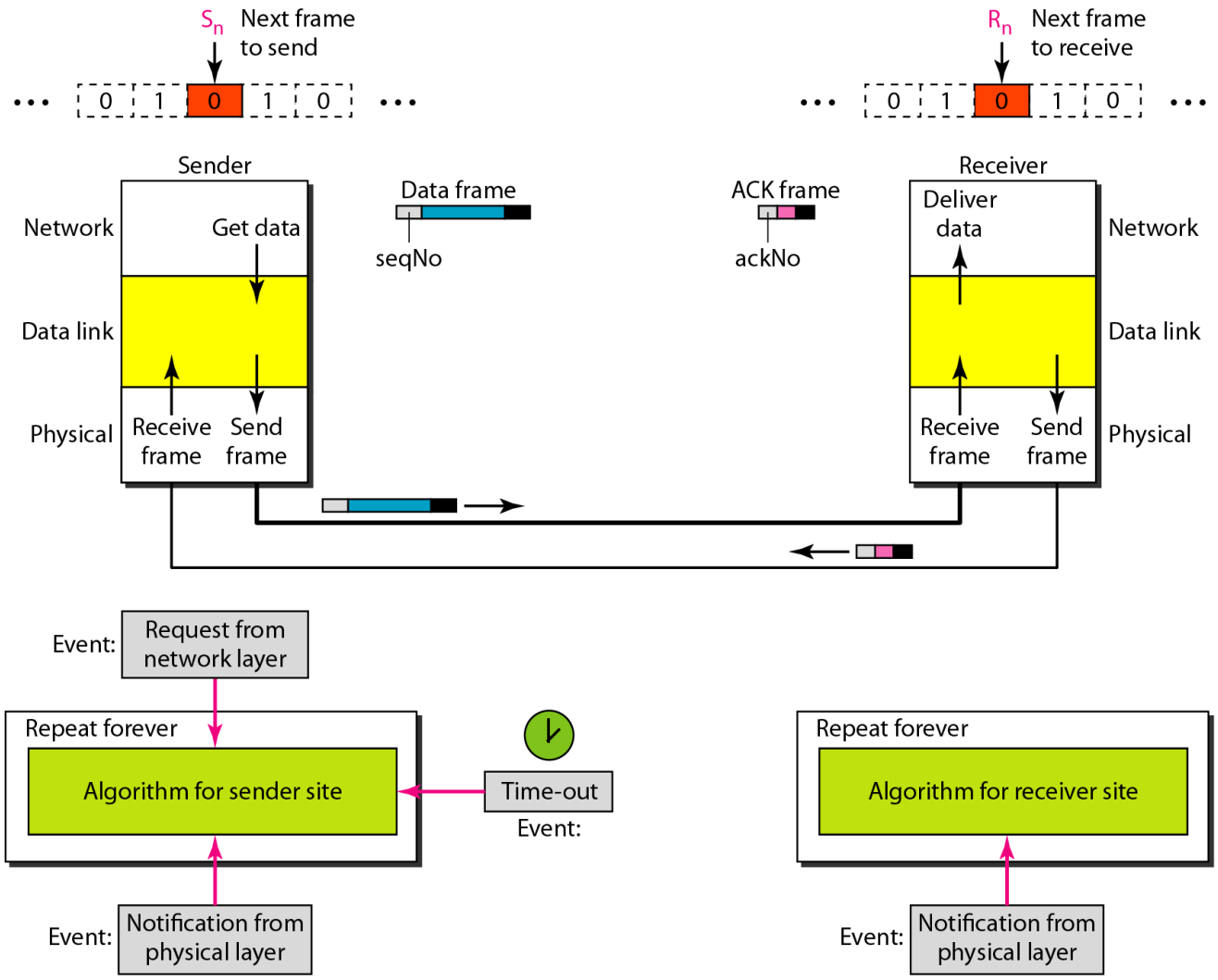
Topics discussed in this section:

Stop-and-Wait Automatic Repeat Request 停等 ARQ
Go-Back-N Automatic Repeat Request 后退 N 帧 ARQ
Selective Repeat Automatic Repeat Request 选择性重传 ARQ

停等 ARQ （Stop-and-Wait Automatic Repeat Request）

- 停等 ARQ 协议中，保留已发送帧的副本，当重传定时器到期后，重传这个帧来实现差错检测。
- 停等 ARQ 使用序列号为每一个帧编号。序列号基于模 2 运算。
- ACK 确认帧也会损坏或丢失，因此也需要一个序列号。
- 停等 ARQ 中， ACK 确认编号是期望收到下一帧的编号（模 2 计算）。

Figure 11.10 停等 ARQ 协议设计



Algorithm 11.5 停等 ARQ 发送方的算法

```
1  Sn = 0;                              // Frame 0 should be sent first
2  canSend = true;                      // Allow the first request to go
3  while(true)                          // Repeat forever
4  {
5    WaitForEvent();                    // Sleep until an event occurs
6    if(Event(RequestToSend) AND canSend)
7    {
8       GetData();
9       MakeFrame(Sn);                        //The seqNo is Sn
10      StoreFrame(Sn);                       //Keep copy
11      SendFrame(Sn);
12      StartTimer();
13      Sn = Sn + 1;
14      canSend = false;
15   }
16   WaitForEvent();                          // Sleep
```

(continued)

Algorithm 11.5 停等 ARQ 发送方的算法 (continued)

```
17      if(Event(ArrivalNotification)          // An ACK has arrived
18      {
19        ReceiveFrame(ackNo);                 //Receive the ACK frame
20        if(not corrupted AND ackNo == Sn)    //Valid ACK
21          {
22            Stoptimer();
23            PurgeFrame(Sn-1);                //Copy is not needed
24            canSend = true;
25          }
26       }
27
28      if(Event(TimeOut)                      // The timer expired
29      {
30       StartTimer();
31       ResendFrame(Sn-1);                    //Resend a copy check
32      }
33  }
```

Algorithm 11.6 停等 ARQ 接收方算法

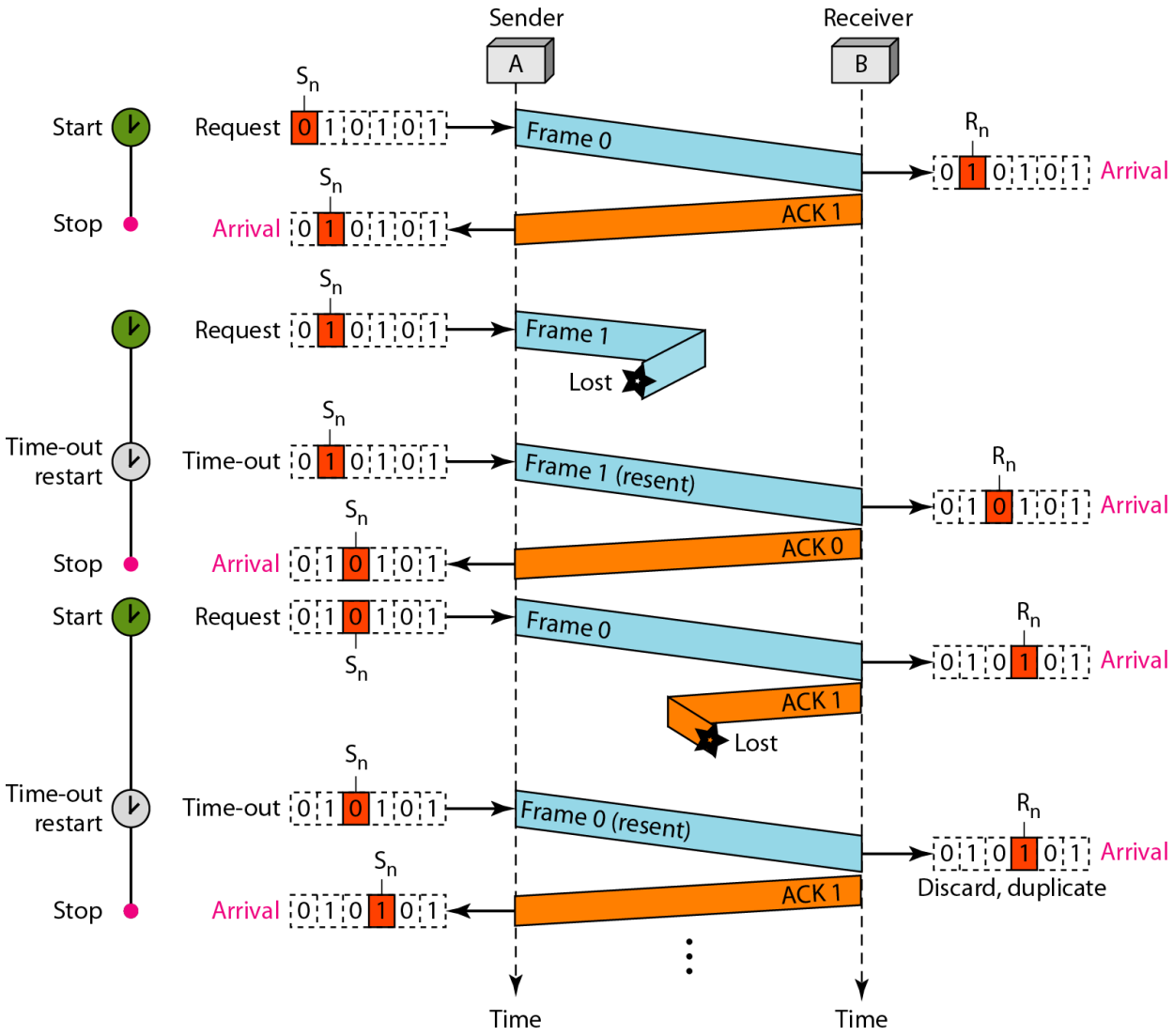
```
1  Rn = 0;                          // Frame 0 expected to arrive first
2  while(true)
3  {
4    WaitForEvent();                // Sleep until an event occurs
5    if(Event(ArrivalNotification))  //Data frame arrives
6    {
7       ReceiveFrame();
8       if(corrupted(frame));
9          sleep();
10      if(seqNo == Rn)              //Valid data frame
11      {
12       ExtractData();
13        DeliverData();             //Deliver data
14        Rn = Rn + 1;
15      }
16       SendFrame(Rn);             //Send an ACK
17   }
18 }
```

序列号不匹配时也会发送 ACK

Example 11.3

停等 ARQ 的例子。第 0 帧发送并确认。第 1 帧丢失，超时后重传。重传的第 1 帧确认，定时器停止计时。第 0 帧发送，但确认帧丢失。超时后，再次重传第 0 帧，并被确认。

Figure 11.11 例 11.3 的帧流量图



Example 11.4

在一个停等 ARQ 系统中，带宽为 1Mbps，往返传播时间是 20ms，求带宽时延积。如果帧长度为 1000bit，则链路的利用率是多少？

Solution

带宽时延积为

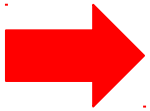
$$1\times10^{6}\times20\times10^{-3}=20000\text{bit}$$

∵ 发送一帧的时间为 1000bit/1M = 1ms，

∴ 链路利用率 = 1ms/(1ms+20ms) = 4.8%

或者利用带宽时延积近似计算，一次往返只能发送 1000bit，

∴链路利用率 = 1000 / 20000 = 5%



大量带宽资源被浪费！

Example 11.5

例 11.4 中，如果在停止和确认之前能发送 15 帧，则链路的利用率是多少？

Solution

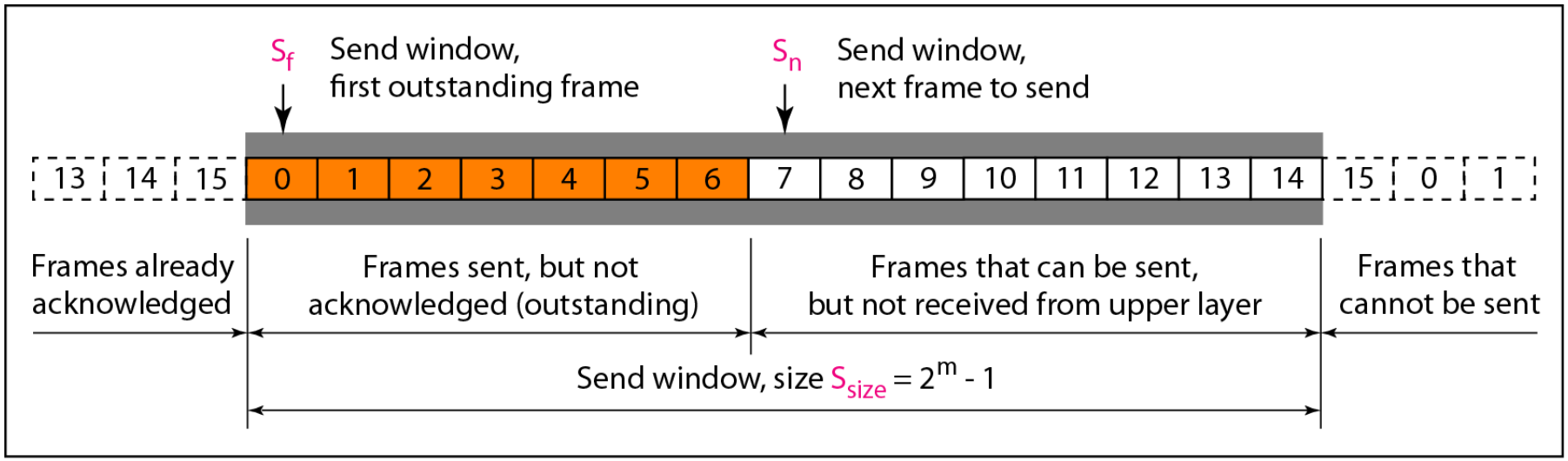
带宽延迟积仍为 20000 bit，在一个往返过程中，系统能发送 15 帧即 15000 bit，因此链路利用率为

$$15000 / 20000 = 75\%$$

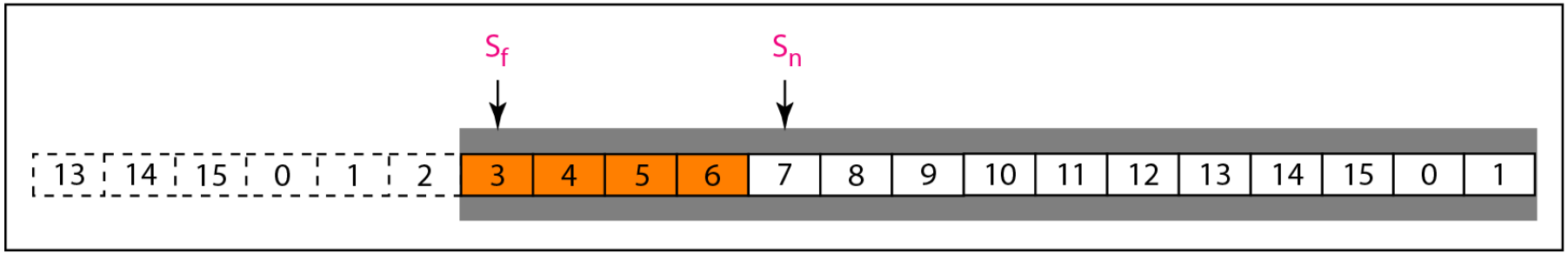
回退 N 帧自动重发请求（Go-Back-N Automatic Repeat Request）

- 在回退 N 帧协议中，序列号是模 2^m ，其中 m 为序列号字段长度。
- 滑动窗口定义发送方和接收方关心的序列号范围。

Figure 11.12 回退 N 帧 ARQ 发送窗口



a. Send window before sliding



b. Send window after sliding

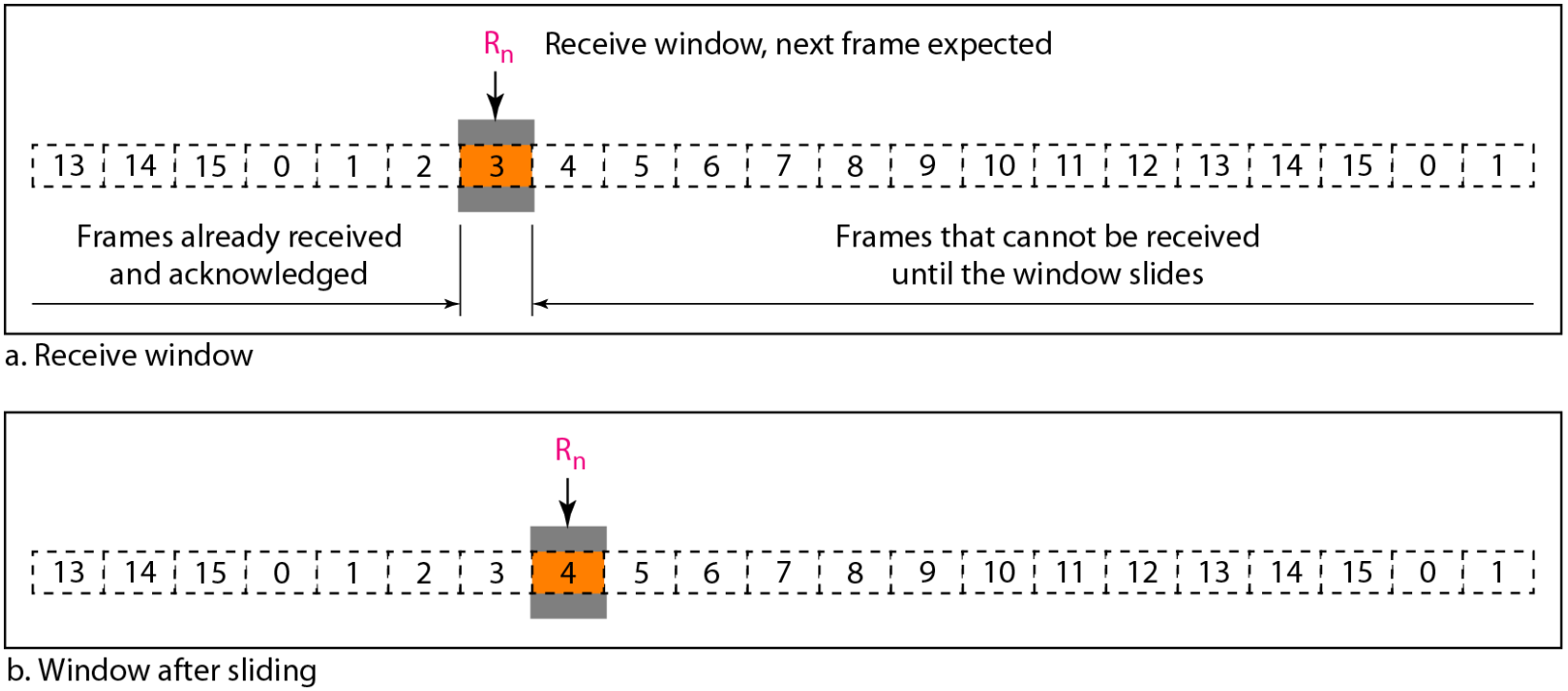
Note

发送窗口是一个抽象概念，通过三个变量 S_f 、 S_n 和 S_{size} 来定义它的大小。

S_{size} 小于 2^m ，一般取 $2^m - 1$ 。

当有效确认到达时，发送窗口滑动一个或者多个帧时隙。

Figure 11.13 回退 N 帧 ARQ 接收窗口



Note

接收窗口是一个抽象概念，用变量 R_n 定义了大小为 1 的接收窗口。正确的帧到达时，接收窗口滑动到下一个时隙。

确认

- 当一个帧安全有序到达时，接收方发送肯定的确认；
- 如果一个帧被损坏或收到时次序颠倒了，接收方不响应，丢弃所有后来的帧直到收到一个期待帧；
- 接收方不响应使得不被确认帧的定时器过期，发送方**从此帧开始重发所有帧**；
- 接收方发送一个累积确认。

定时器过期时，发送方会重发所有待处理的帧，这是其称为回退 N 帧自动重发请求协议的由来。

Figure 11.14 回退 N 帧的 ARQ 设计

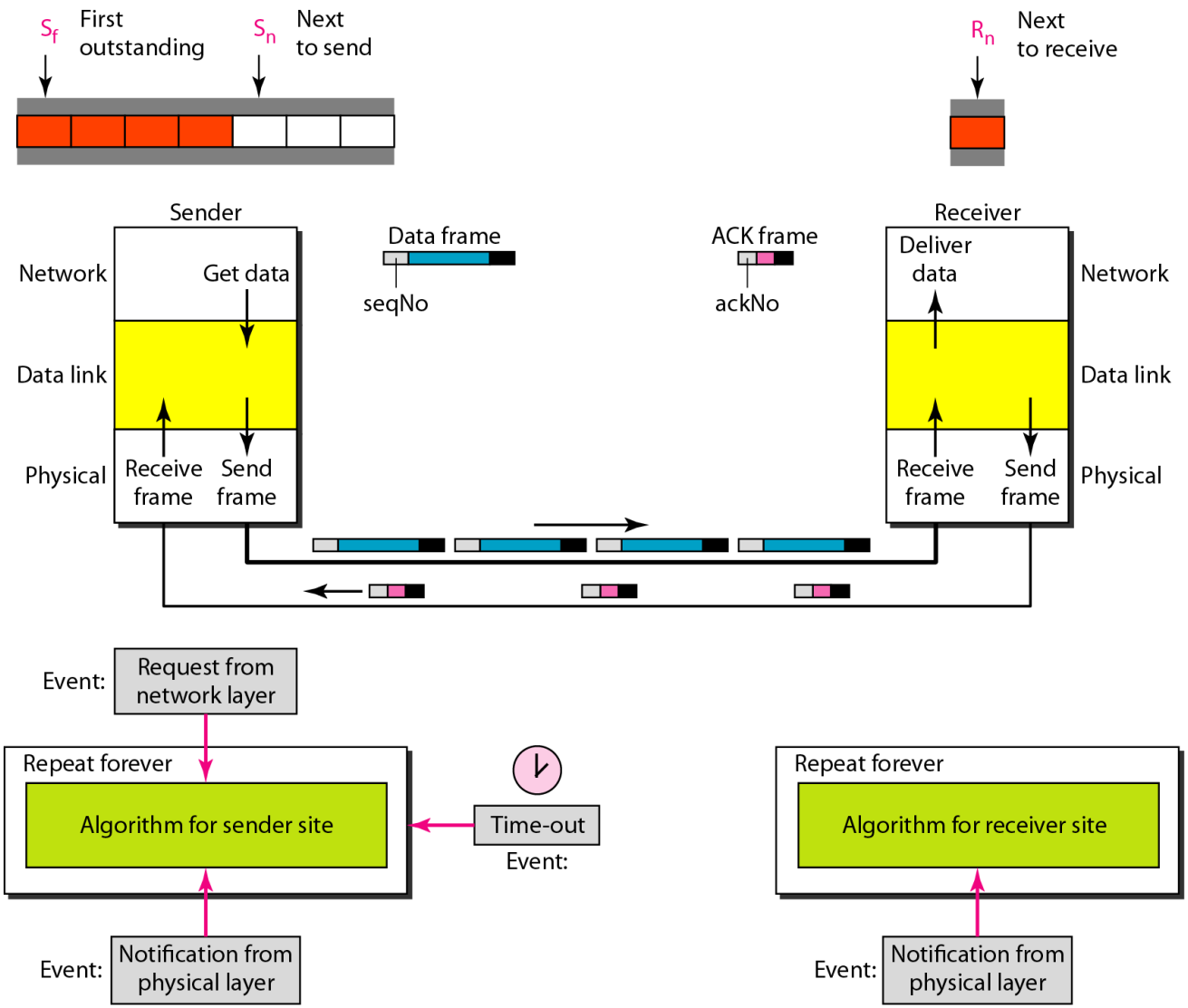
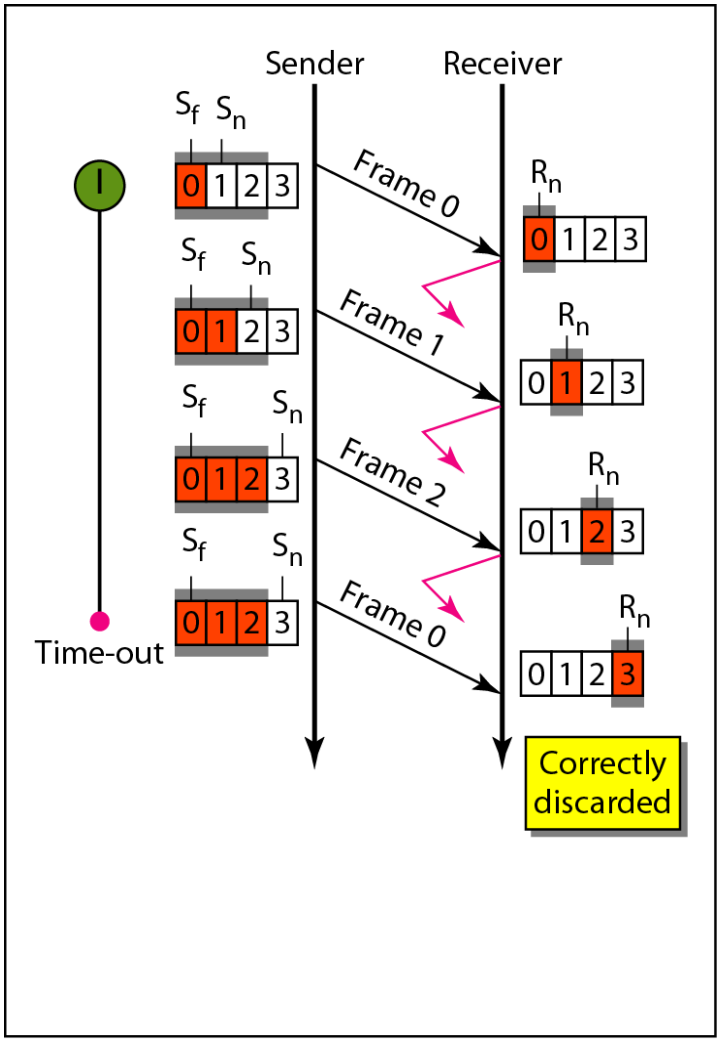
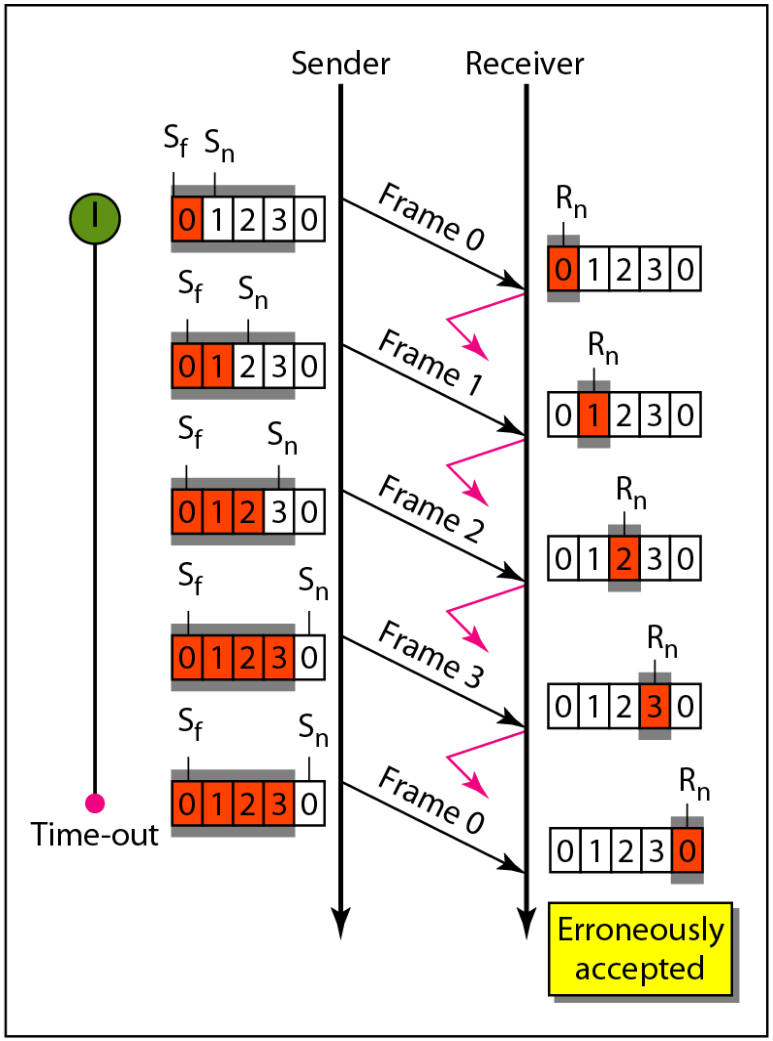


Figure 11.15 回退 N 帧 ARQ 窗口的大小



a. Window size < 2^m



b. Window size = 2^m

Note

回退 *N* 帧 ARQ 中，发送窗口必须小于 2^m ；
接收窗口始终为 1 。

Algorithm 11.7 回退 N 帧发送方算法

```
1  Sw = 2^m - 1;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7   WaitForEvent();
8     if(Event(RequestToSend))           //A packet to send
9     {
10       if(Sn-Sf >= Sw)                //If window is full
11             Sleep();
12       GetData();
13       MakeFrame(Sn);
14       StoreFrame(Sn);
15       SendFrame(Sn);
16       Sn = Sn + 1;
17       if(timer not running)
18            StartTimer();
19    }
20
```

(continued)

Algorithm 11.7 回退 N 帧发送方算法

(continued)

```
21    if(Event(ArrivalNotification))  //ACK arrives
22    {
23       Receive(ACK);
24       if(corrupted(ACK))
25            Sleep();
26       if((ackNo>Sf)&&(ackNo<=Sn))  //If a valid ACK
27       While(Sf <= ackNo)
28        {
29         PurgeFrame(Sf);
30         Sf = Sf + 1;
31        }
32        StopTimer();
33    }
34
35    if(Event(TimeOut))              //The timer expires
36    {
37     StartTimer();
38     Temp = Sf;
39     while(Temp < Sn);
40       {
41        SendFrame(Temp);
42        Temp = Temp + 1;
43       }
44    }
45 }
```

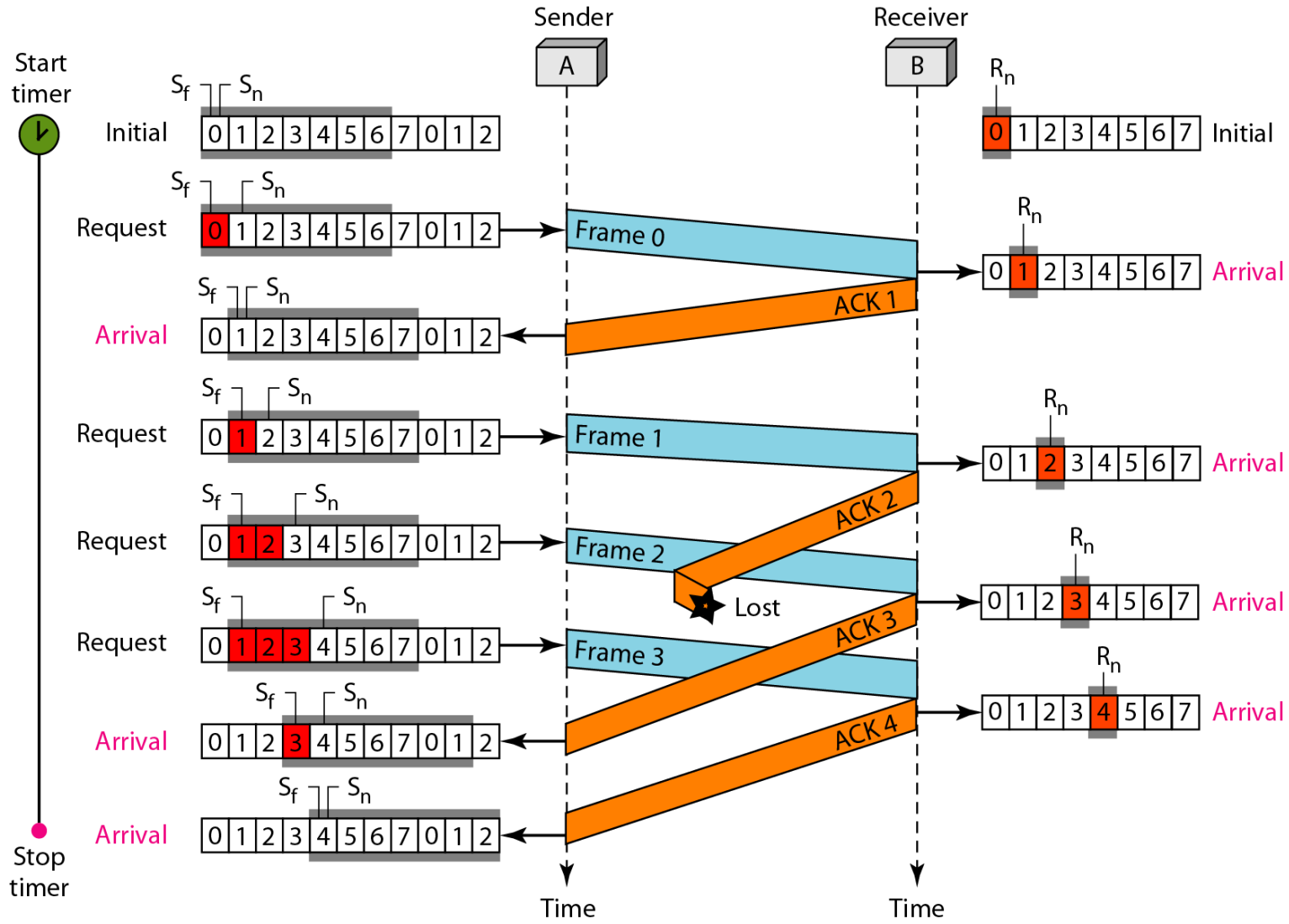
Algorithm 11.8 回退 N 帧接收方算法

```
1  Rn = 0;
2
3  while (true)                          //Repeat forever
4  {
5    WaitForEvent();
6
7    if(Event(ArrivalNotification)) /Data frame arrives
8    {
9       Receive(Frame);
10      if(corrupted(Frame))
11           Sleep();
12      if(seqNo == Rn)                  //If expected frame
13      {
14        DeliverData();                 //Deliver data
15        Rn = Rn + 1;                   //Slide window
16        SendACK(Rn);
17      }
18   }
19 }
```

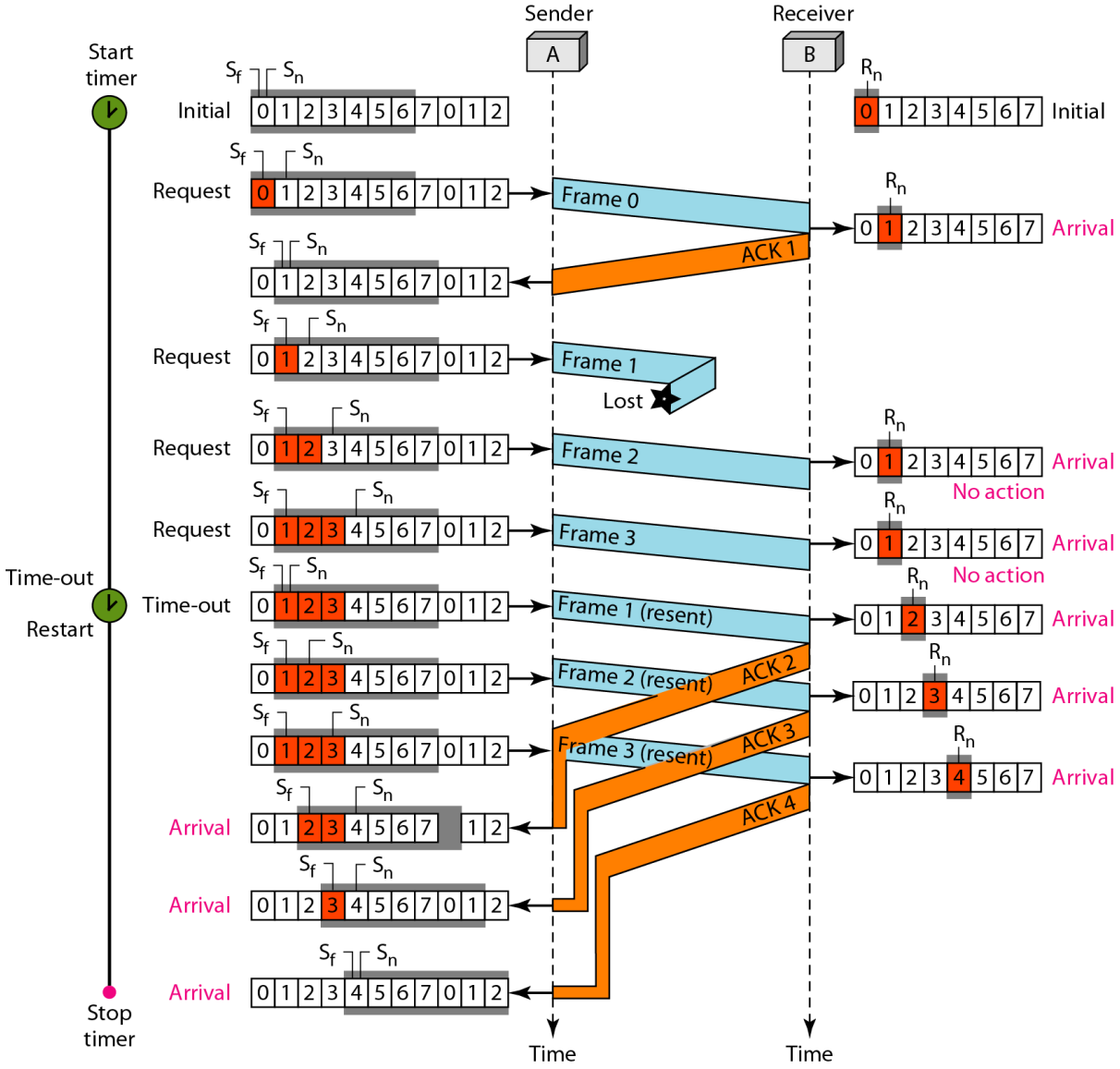
Example 11.6

以下回退 N 帧的例子中，前向信道可靠，反向信道不可靠。造成一些 ACK 帧迟延或丢失时，链路协议发生累积确认的情况。

Figure 11.16 ACK 丢失时累积确认的情况



Example 11.7 Figure 11.17 发送帧丢失时的情况



Note

停等 ARQ 是窗口大小为 1 的回退 N 帧 ARQ 的特殊情况。

选择性重复 ARQ （ Selective Repeat ARQ ）

回退 N 帧 ARQ 的问题：

- 接收方只有一个变量，不能解决帧失序问题；
- 在有噪声信道效率低下，因为需要重发多个帧。

选择性重复 ARQ ：

- 窗口尺寸更小，为 2^{m-1} ；
- 接收窗口与发送窗口大小相同。

Figure 11.18 选择性重传 ARQ 发送窗口

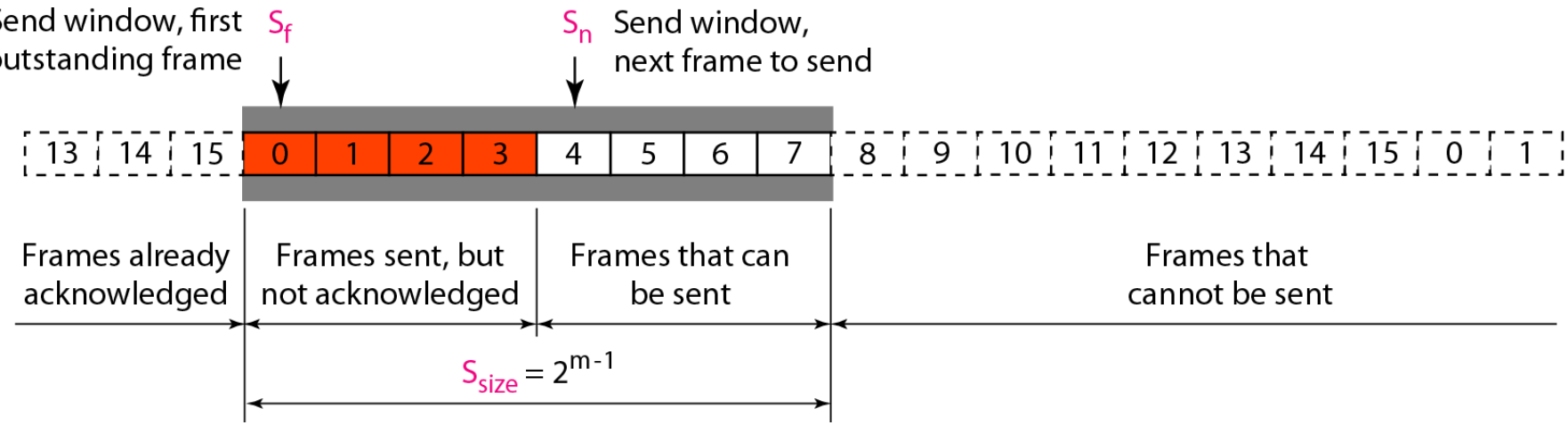


Figure 11.19 选择性重传 ARQ 接收窗口

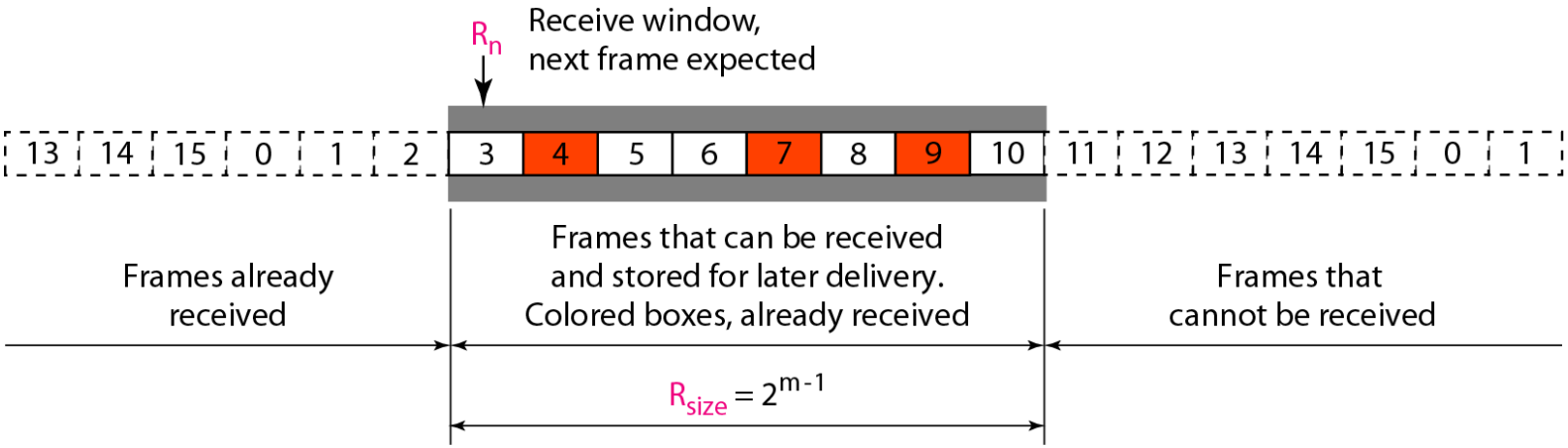


Figure 11.20 选择性重传 ARQ 设计

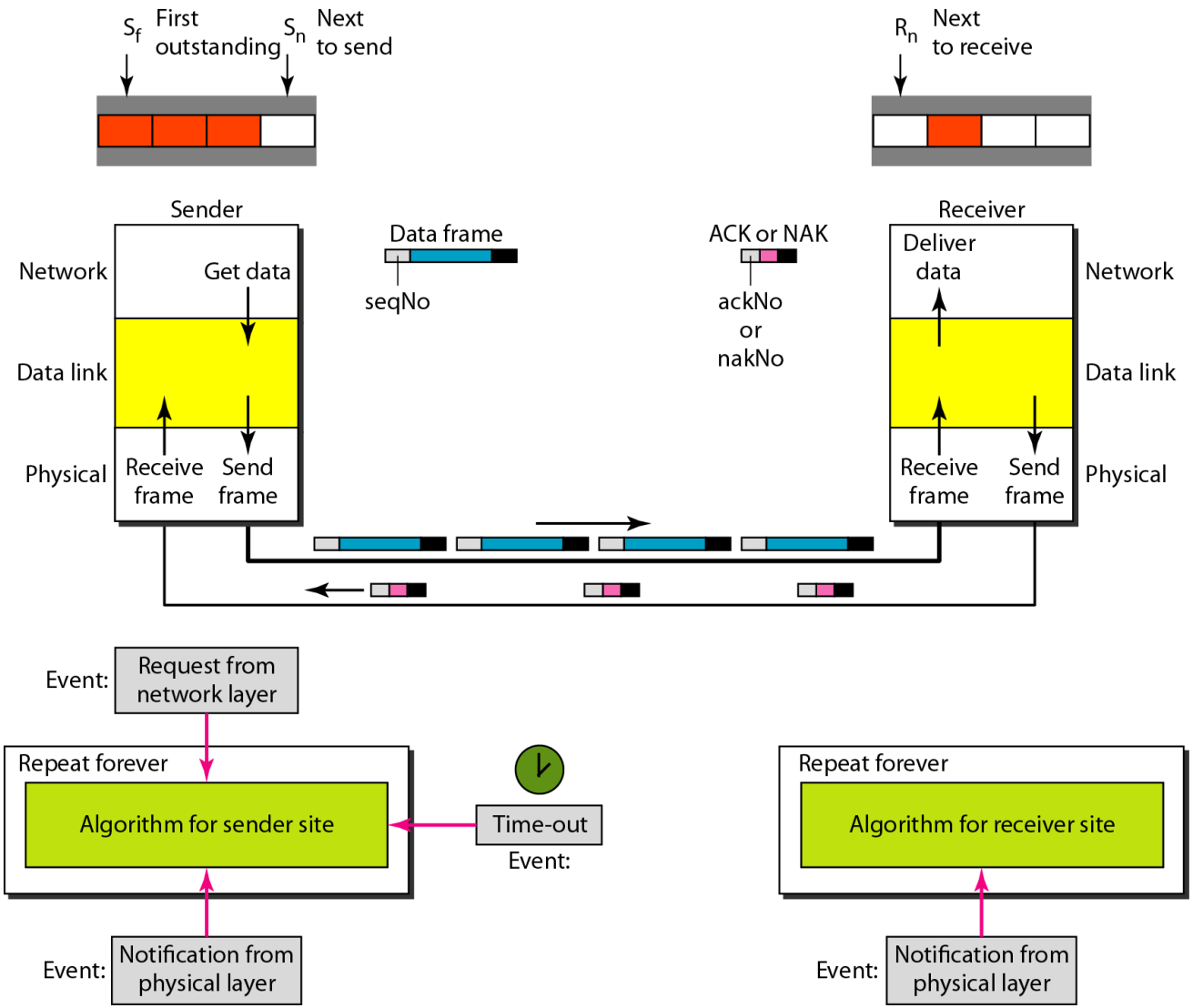
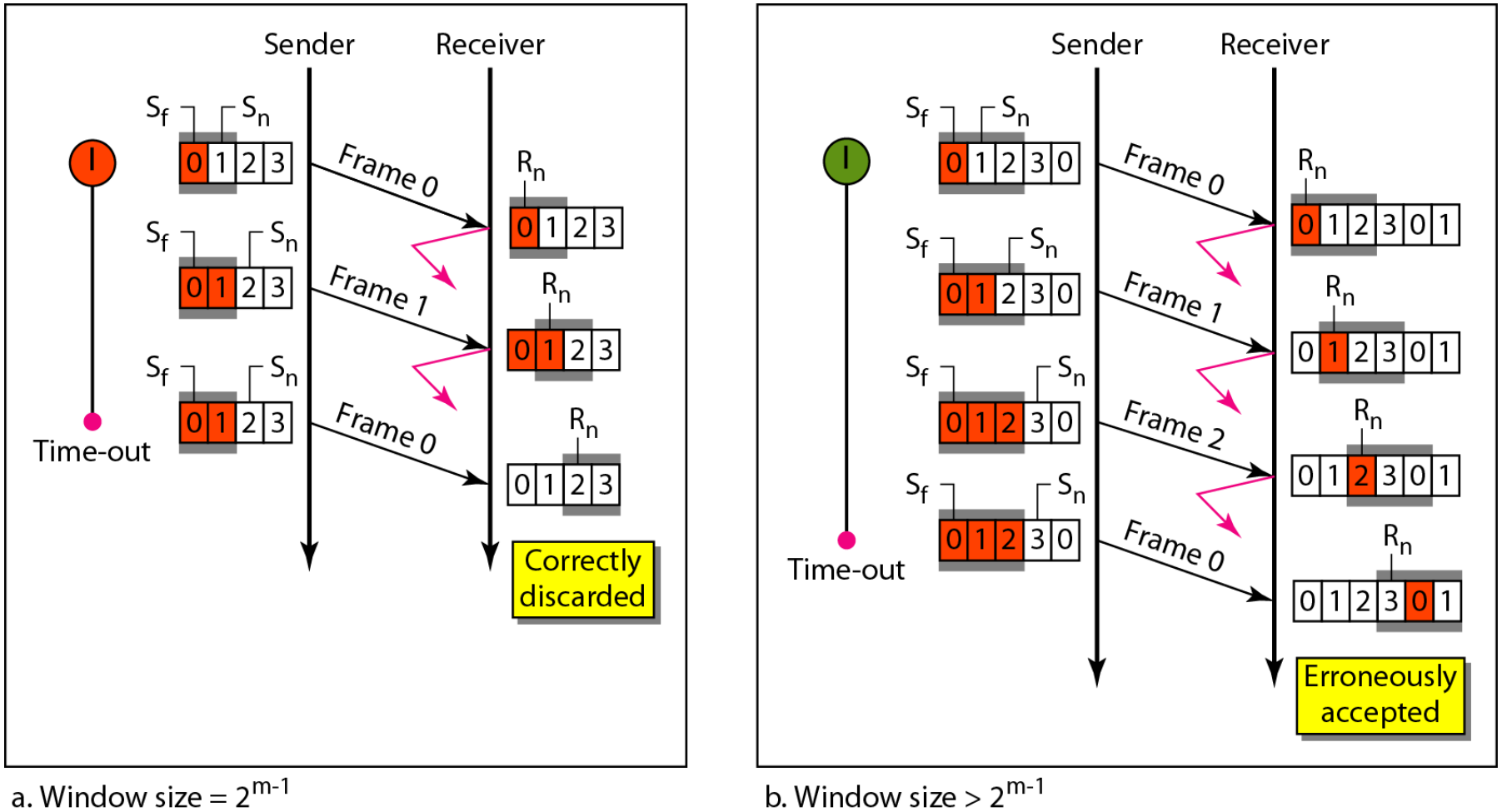


Figure 11.21 选择重传 ARQ 窗口大小



a. Window size = 2^{m-1}

b. Window size > 2^{m-1}

Note

选择性重传 ARQ，发送和接收窗口必须小于等于 2^{m-1} 。

Algorithm 11.9 选择重传发送方算法

```
1  Sw = 2^(m-1) ;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7    WaitForEvent();
8    if(Event(RequestToSend))            //There is a packet to send
9    {
10     if(Sn-Sf >= Sw)                   //If window is full
11           Sleep();
12     GetData();
13     MakeFrame(Sn);
14     StoreFrame(Sn);
15     SendFrame(Sn);
16     Sn = Sn + 1;
17     StartTimer(Sn);
18   }
19
```

(continued)

Algorithm 11.9 选择重传发送方算法

(continued)

```
20  if(Event(ArrivalNotification)) //ACK arrives
21  {
22     Receive(frame);              //Receive ACK or NAK
23     if(corrupted(frame))
24          Sleep();
25     if (FrameType == NAK)
26        if (nakNo between Sf and Sn)
27        {
28         resend(nakNo);
29         StartTimer(nakNo);
30        }
31     if (FrameType == ACK)
32        if (ackNo between Sf and Sn)
33        {
34          while(sf < ackNo)
35          {
36           Purge(sf);
37           StopTimer(sf);
38           Sf = Sf + 1;
39          }
40        }
41  }
```


(continued)

Algorithm 11.9 选择重传发送方算法

(continued)

```
42
43    if(Event(TimeOut(t)))              //The timer expires
44    {
45     StartTimer(t);
46     SendFrame(t);
47    }
48  }
```

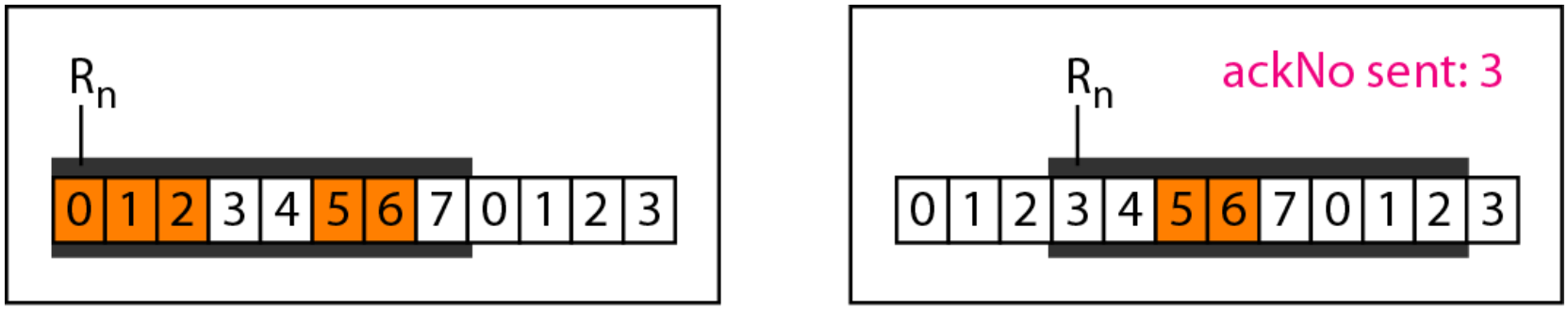
Algorithm 11.10 选择重传接收方算法

```
1  Rn = 0;
2  NakSent = false;
3  AckNeeded = false;
4  Repeat(for all slots)
5      Marked(slot) = false;
6
7  while (true)                                  //Repeat forever
8  {
9    WaitForEvent();
10
11   if(Event(ArrivalNotification))              /Data frame arrives
12   {
13      Receive(Frame);
14      if(corrupted(Frame))&& (NOT NakSent)
15      {
16       SendNAK(Rn);
17       NakSent = true;
18       Sleep();
19      }
20      if(seqNo <> Rn)&& (NOT NakSent)
21      {
22       SendNAK(Rn);
```

Algorithm 11.10 选择重传接收方算法

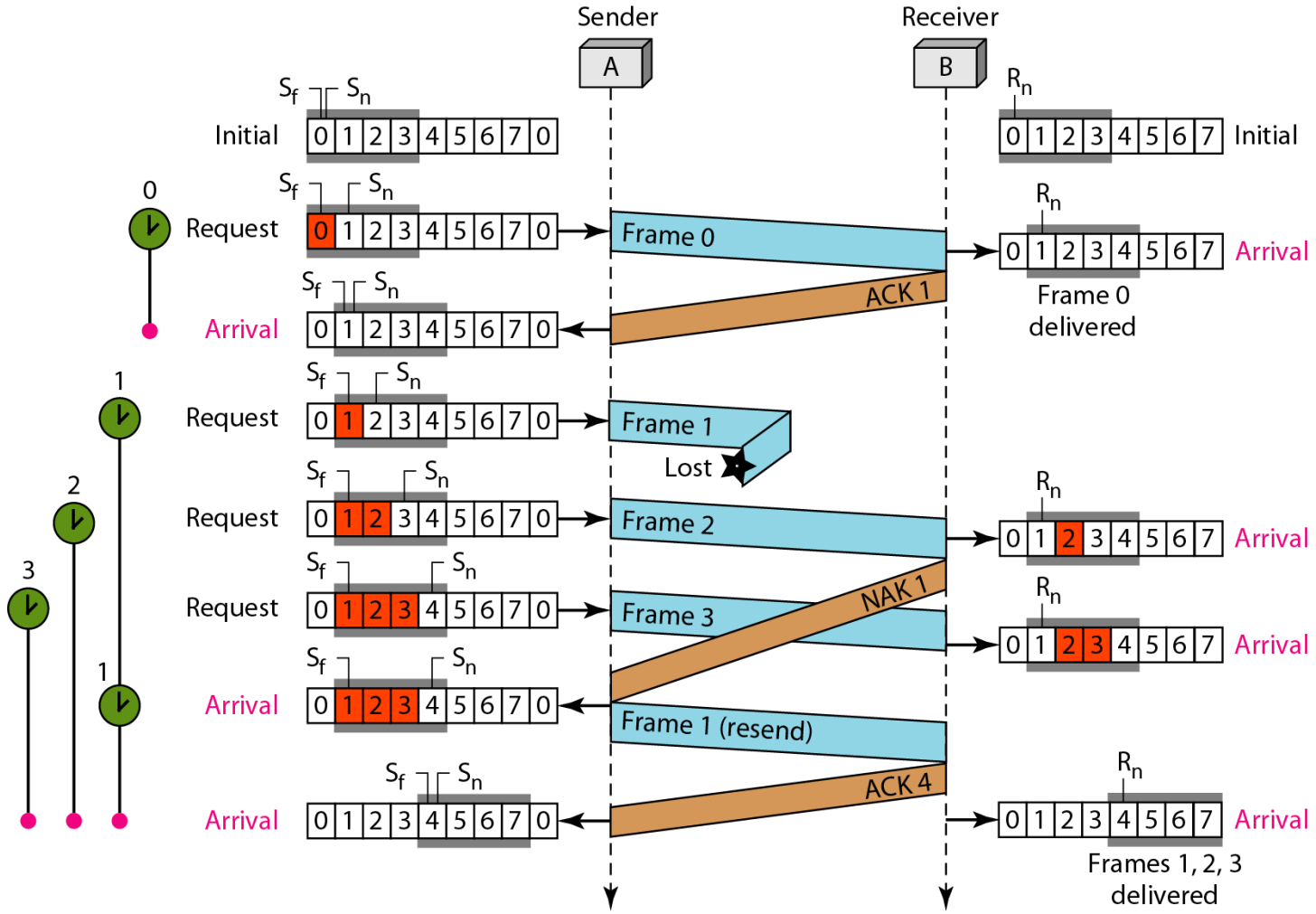
```
23          NakSent = true;
24          if ((seqNo in window)&&(!Marked(seqNo))
25          {
26           StoreFrame(seqNo)
27           Marked(seqNo)= true;
28           while(Marked(Rn))
29           {
30            DeliverData(Rn);
31            Purge(Rn);
32            Rn = Rn + 1;
33            AckNeeded = true;
34           }
35            if(AckNeeded);
36            {
37            SendAck(Rn);
38            AckNeeded = false;
39            NakSent = false;
40            }
41          }
42        }
43    }
44  }
```

Figure 11.22 选择性 ARQ 中传输数据



有颜色代表 ACK 已确认，白色代表无确认

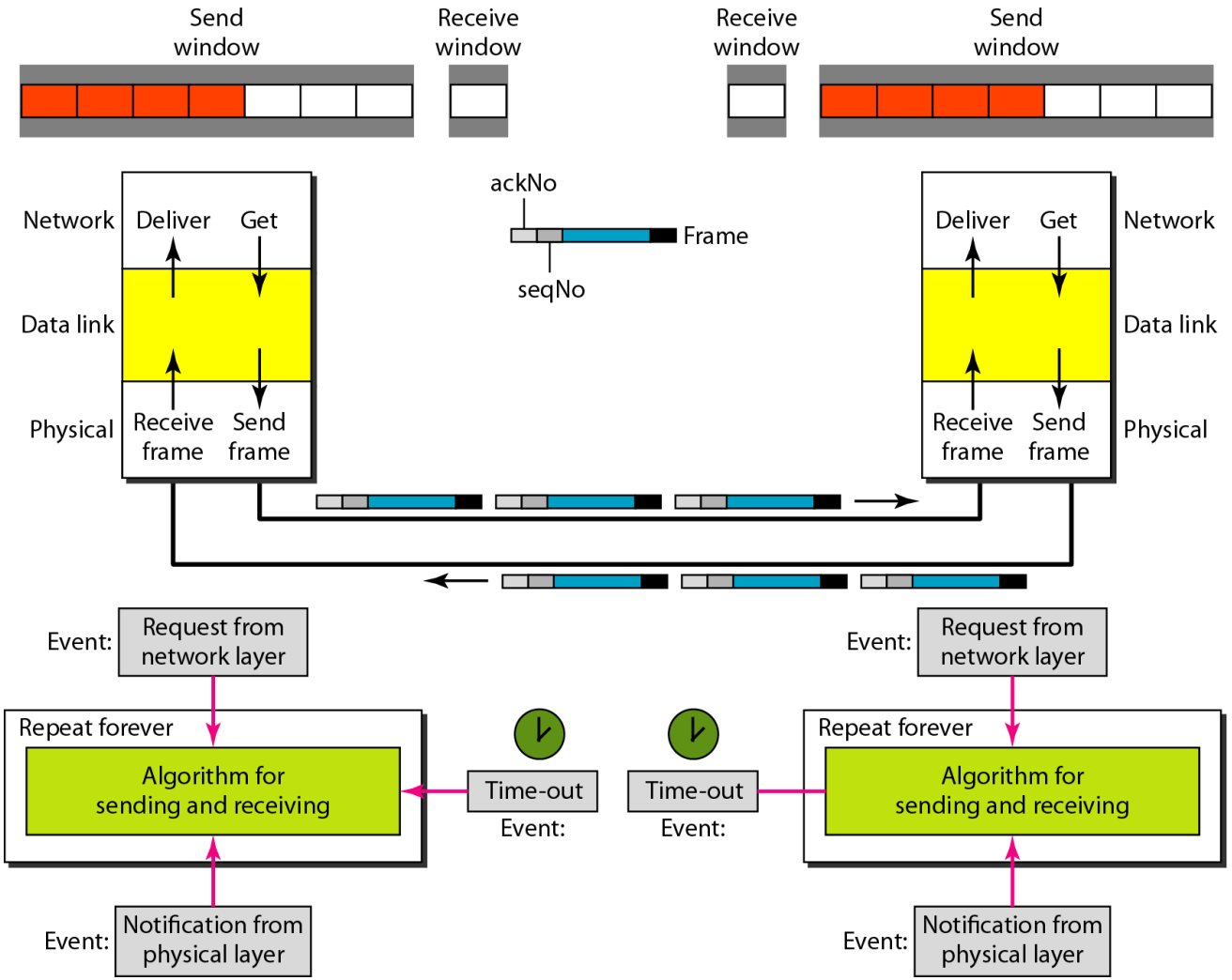
Figure 11.23 11.8 帧丢失情况和 NAK 帧的情况



需注意的问题

- 定时器启动、重启和停止；
- 一组连续帧到达且从窗口起点开始则可向网络层交付；
- 每个窗口位置只发送一个 NAK 并指明窗口中的第一个时隙；
- 当数据被交付到网络层时才发送 ACK 。

Figure 11.24 带捎带的 N 步返回 ARQ



例题

用户 A 与用户 B 通过卫星链路通信时，传播延迟为 270ms，假设数据速率为 64kb/s，帧长为 4000bit。

（1）若采用停止等待 ARQ 协议通信，则最大链路利用率是多少？

（2）若采用返回 N 帧 ARQ 协议通信，发送窗口为 8，则最大链路利用率可以达到多少？

例题

最大链路利用率，即一个周期中（传输时延 + 传播时延），按最大窗口发送帧的时间（传输时延） / 总时延。

（1）传输时延 = 4000bit / (64kb/s) = 62.5ms

传播时延 = 270ms * 2 = 540ms

利用率 = 62.5ms / (62.5ms + 540ms) = 0.104

例题

（2）8帧的传输时延 = 8 * 4000bit / (64kb/s) = 500ms

由于500ms < 540ms（往返的传播时延），一个周期内可以全部发送，所以一个周期仍按1帧的传输时延+传播时延计算。

利用率 = 500ms / (62.5ms + 540ms) = 0.832 (即 0.104 * 8)

11-6 HDLC 高级数据链路控制协议

HDLC （*High-level Data Link Control*）是一个实际应用的面向比特的数据链路协议，支持点到点链路和多点链路。具体实现了本章讨论的各种 ARQ 协议。具有两种通用传输模式：

- 正常响应方式（ Normal Response Mode， NRM ）
- 异步平衡方式（ Asynchronous Balanced Mode， ABM ）

Topics discussed in this section:

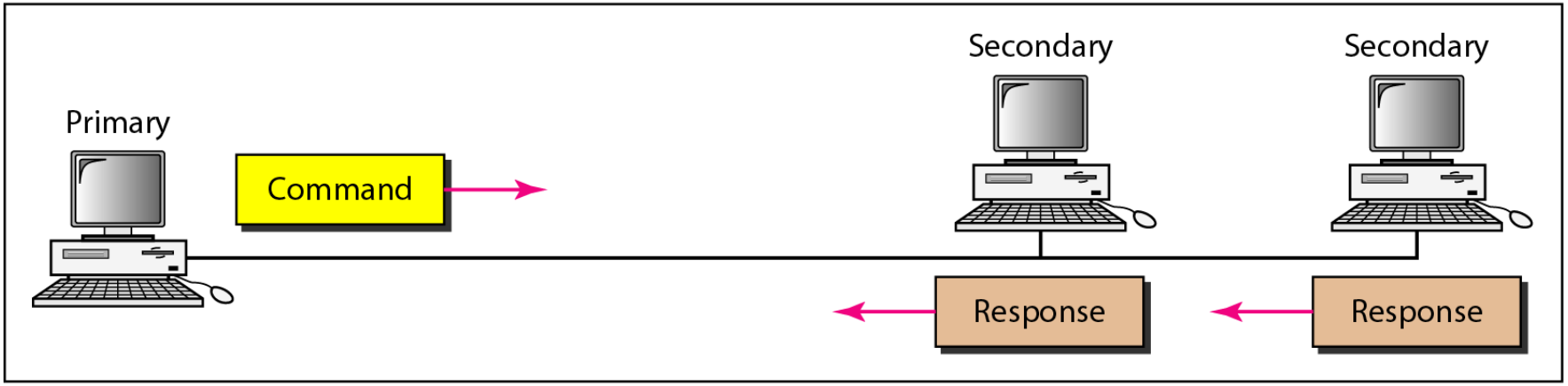
Configurations and Transfer Modes	配置和传输方式
Frames	HDLC 的帧格式
Control Field	HDLC 的帧控制字段

数据链路配置方式

- **非平衡配置方式**
  - 主站与从站：一组结点根据在通信过程中的地位分为主站与从站，由主站来控制数据链路的工作过程。主站发出命令，从站接受命令，发出响应，配合主站工作。
  - 点对点方式与多点方式：分为点对点方式与多点方式两种类型，在多点方式的链路中，主站与每个从站之间分别建立数据链路。
  - 正常响应模式与异步响应模式：分为正常响应模式与异步响应模式两种数据传输方式。在正常响应模式中，主站可随时向从站传输数据帧。只有在主站向从站发送命令帧探询，从站响应后才可以向主站发送数据帧。在异步响应模式中，主站和从站可以随时相互传输数据帧，从站不需要等待主站发出探询就可以发送数据帧，但是主站仍然负责数据链路的初始化、建立、释放与差错恢复等功能。
- **平衡配置方式**
  - 链路两端的两个站都是复合站，复合站同时具有主站与从站的功能，每个复合站都可以发出命令与响应。平衡配置方式只有异步平衡模式一种工作模式，每个复合站都可以发起数据传输，而不需要得到对方的许可。

Figure 11.25 正常响应方式

a. Point-to-point



b. Multipoint

Figure 11.26 异步平衡方式 （普遍应用的方式）

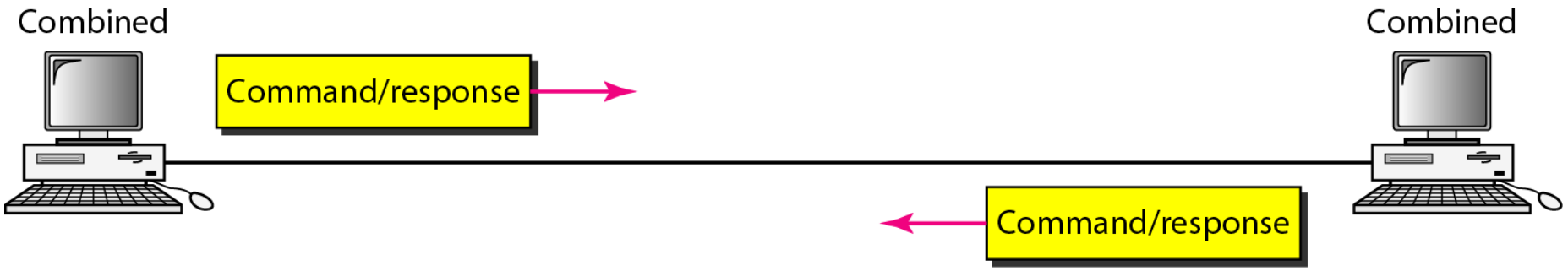


Figure 11.27 HDLC 帧结构 (信息帧，管理帧，无编号帧)

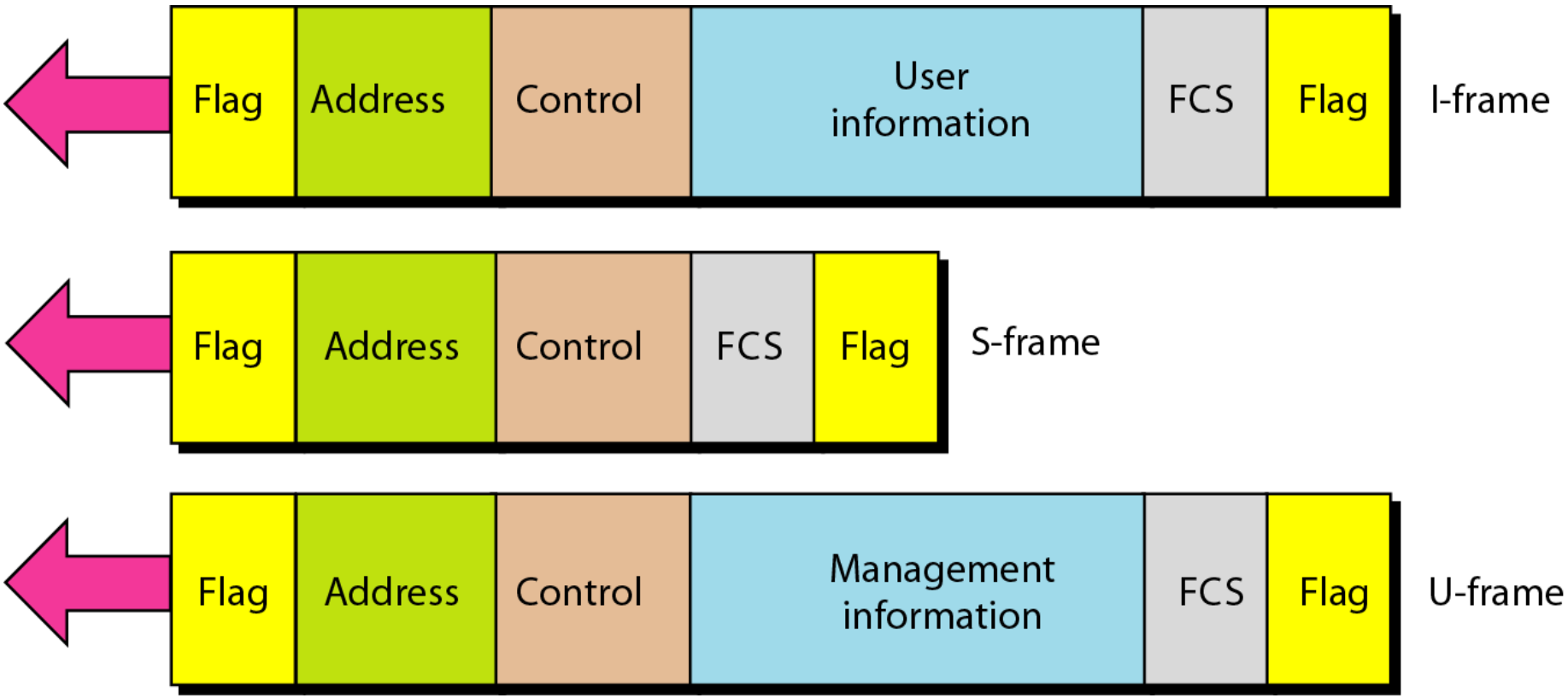
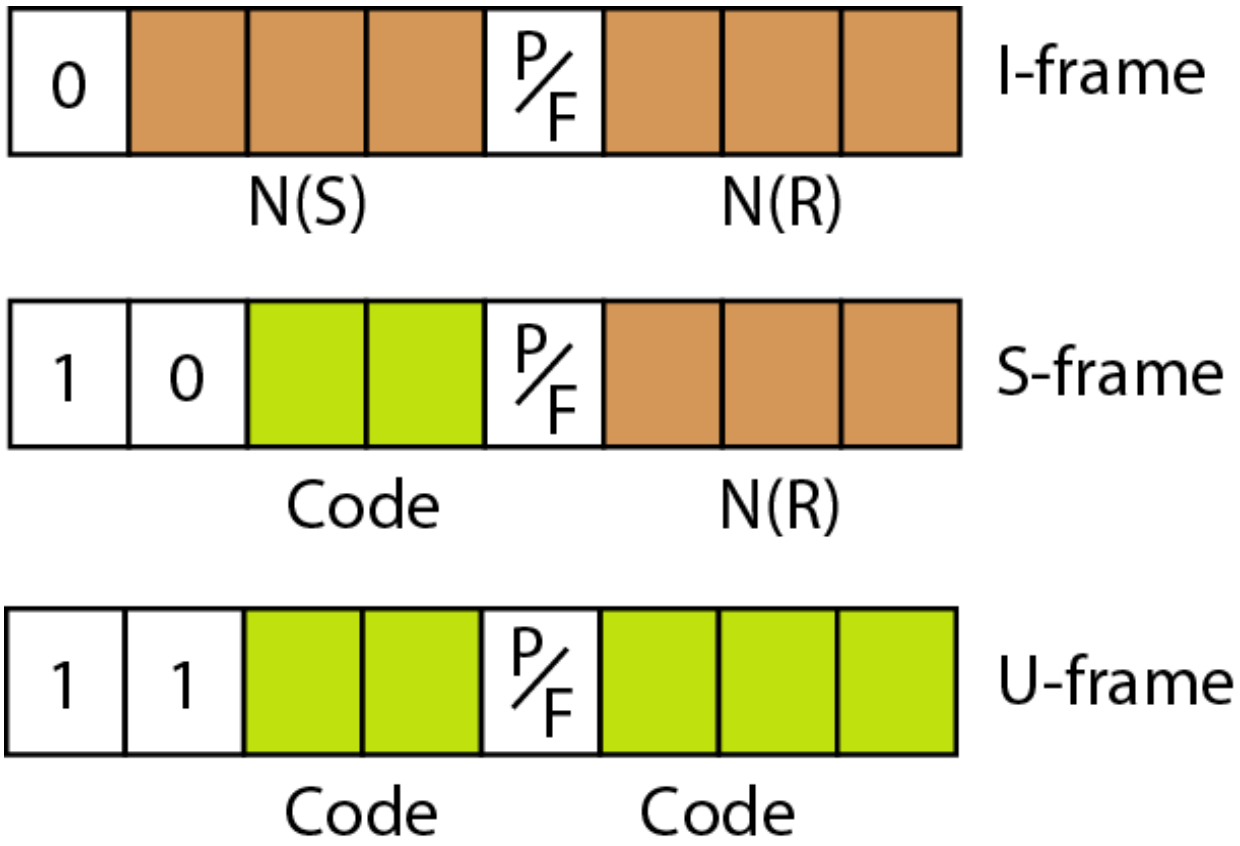


Figure 11.28 控制字段



管理帧的控制字段

- 准备接收 RR，字段标识是 00
- 不准备接收 RNR，字段标识是 10
- 拒绝接收 REJ，字段标识是 01
- 选择性拒收 SREJ，字段标识是 11

Table 11.1 无编号帧的指令和响应

Code	*Command*	*Response*	*Meaning*
00 001	SNRM		Set normal response mode
11 011	SNRME		Set normal response mode, extended
11 100	SABM	**DM**	Set asynchronous balanced mode or **disconnect mode**
11 110	SABME		Set asynchronous balanced mode, extended
00 000	UI	**UI**	Unnumbered information
00 110		**UA**	**Unnumbered acknowledgment**
00 010	DISC	**RD**	Disconnect or **request disconnect**
10 000	SIM	**RIM**	Set initialization mode or **request information mode**
00 100	UP		Unnumbered poll
11 001	RSET		Reset
11 101	XID	**XID**	Exchange ID
10 001	FRMR	**FRMR**	Frame reject

Example 11.9

异步平衡方式下，使用无编号帧建立链路连接和断开链接的情况。

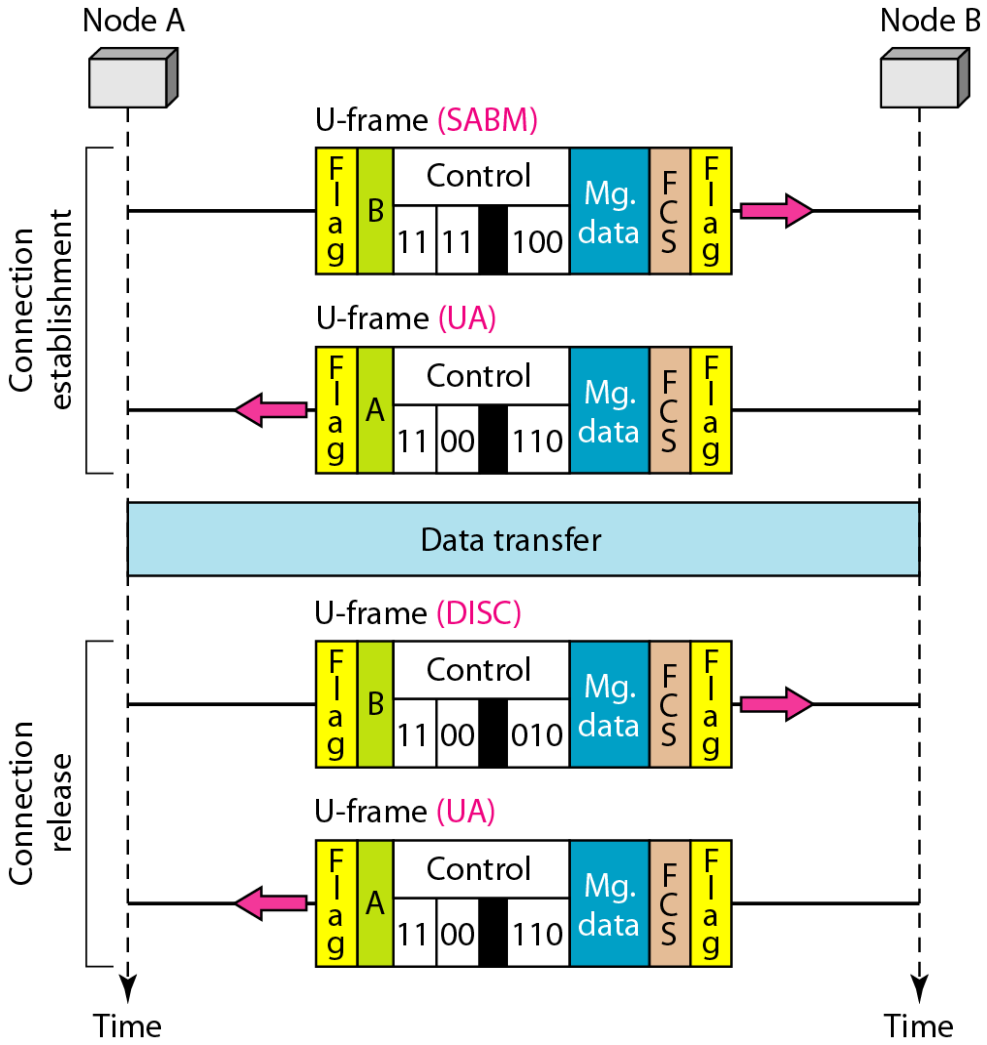
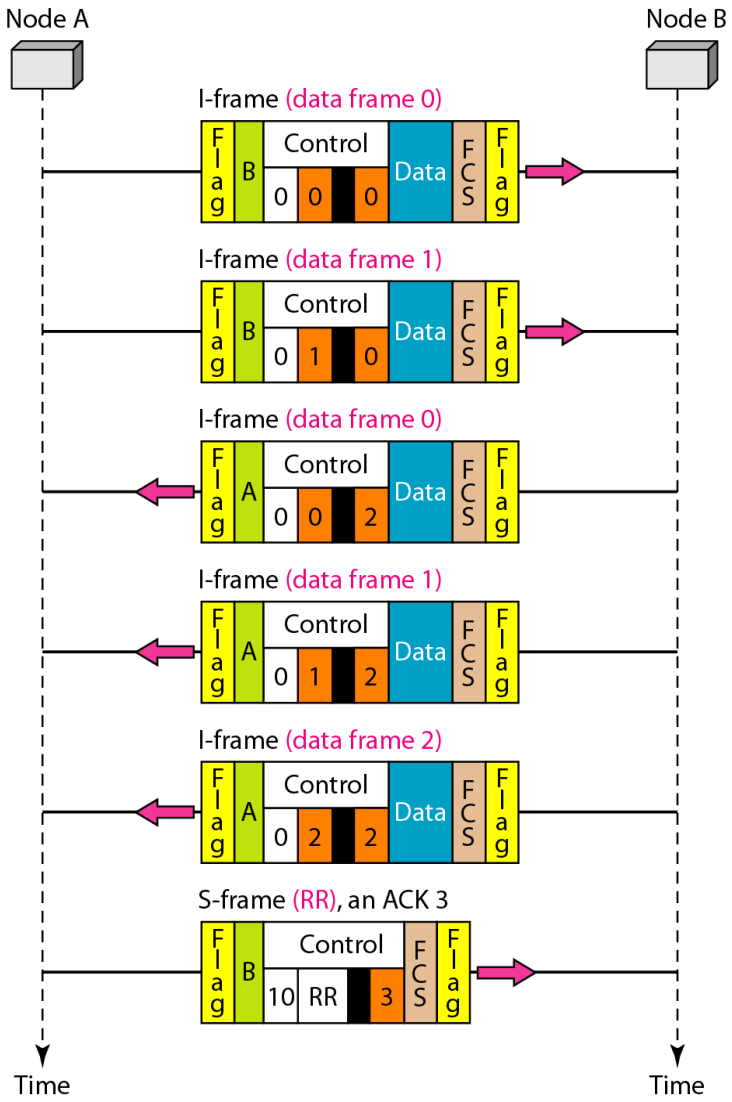
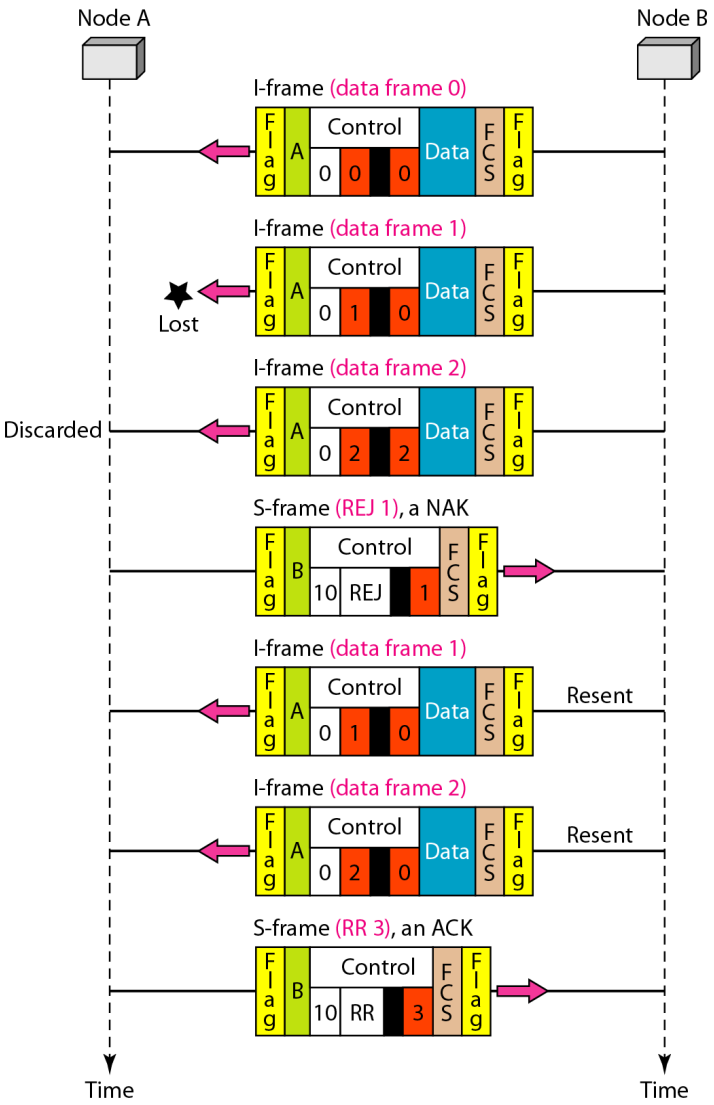


Figure 11.29 建立连接和拆除连接

Example 11.10 Figure 11.30 无差错捎带情况



Example 11.11 Figure 11.31 有差错捎带情况



11-7 POINT-TO-POINT PROTOCOL
点到点协议

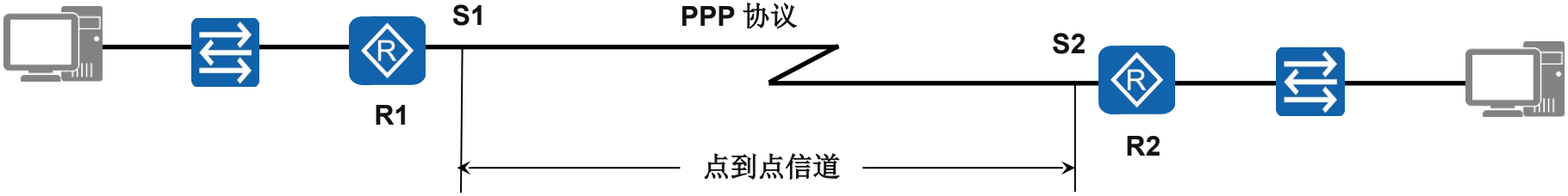
高级数据链路控制协议是点到点和点到多点都能使用的一个通用协议，但最通用的协议还是点到点协议（ Point-to-Point Protocol， PPP ），使用面向字节的方式。

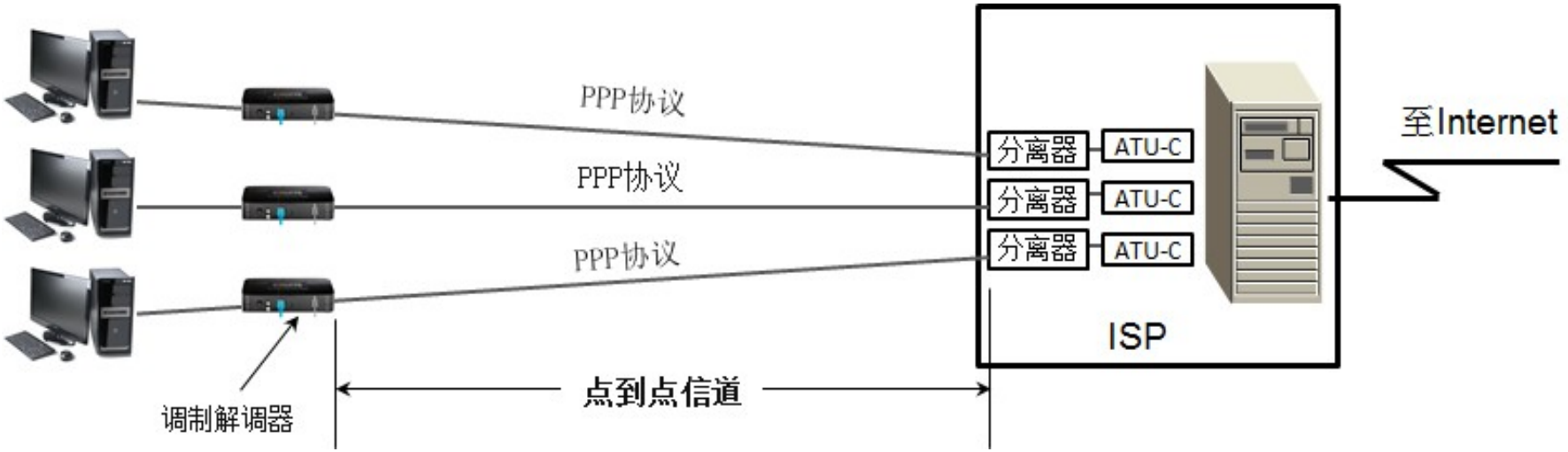
Topics discussed in this section:

Framing	帧格式
Transition Phases	传输阶段
Multiplexing	多路复用
Multilink PPP	多链路 PPP

点到点信道的数据链路

点到点信道是指的一条链路上就一个发送端和接收端的信道，通常用在广域网链路。

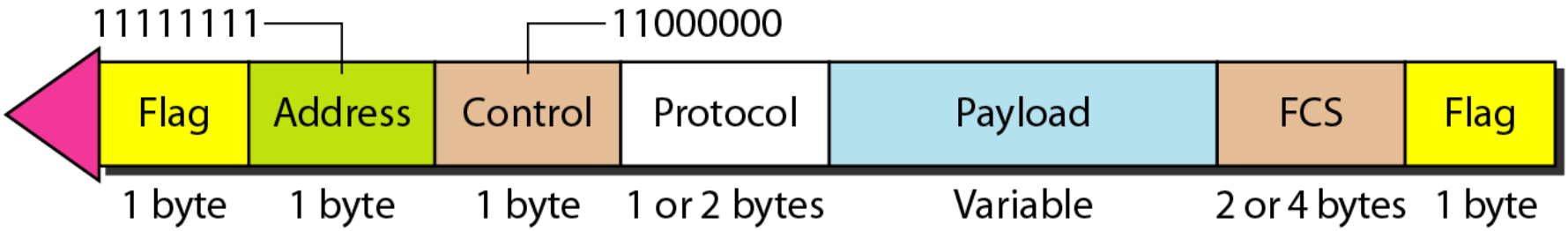




PPP 协议的特点

- 简单：不提供可靠传输，无流量控制，无重传机制，网络开销小，速度快
- 封装成帧：首部和尾部，帧开始符，帧结束符
- 透明传输：可传输任意比特组合的数据，加转义字符，收到后去掉转义字符
- 差错检测： CRC 计算帧校验序列 FCS
- 支持多种网络层协议： IPv4 和 IPv6 网络层协议都可以封装到 PPP 帧中
- 多种类型链路：光纤、铜线，同步传输、异步传输，串行、并行链路均可
- 最大传送单元： 1500 字节
- 网络层地址协商：能够为拨号的一端分配 IP 地址、子网掩码、网关和 DNS
- 数据压缩协商

Figure 11.32 PPP 帧格式



Note

PPP 是面向字节的协议，通过转义字节 01111101 进行透明插入和删除。

Figure 11.33 状态迁移图

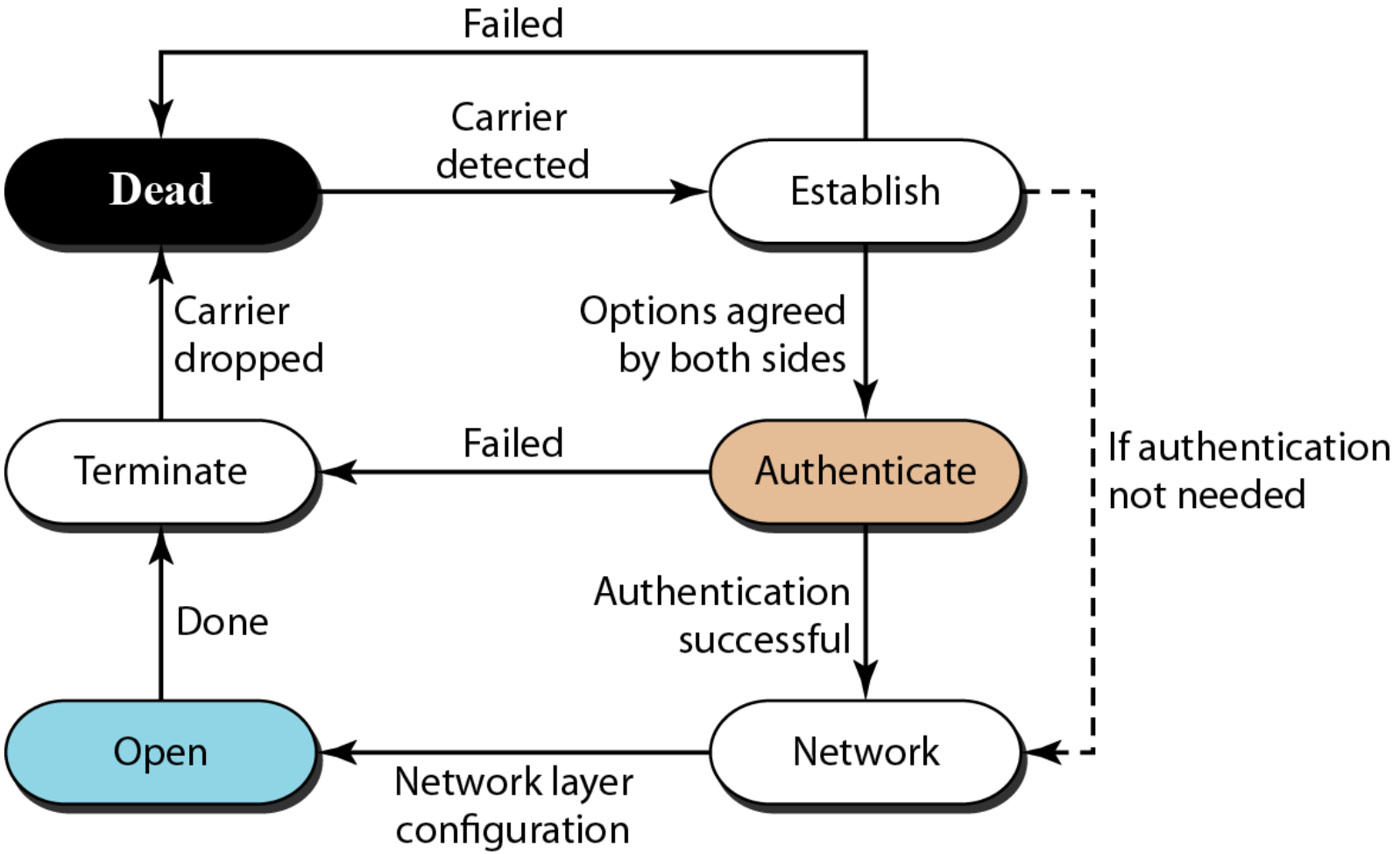


Figure 11.34 PPP 多路复用

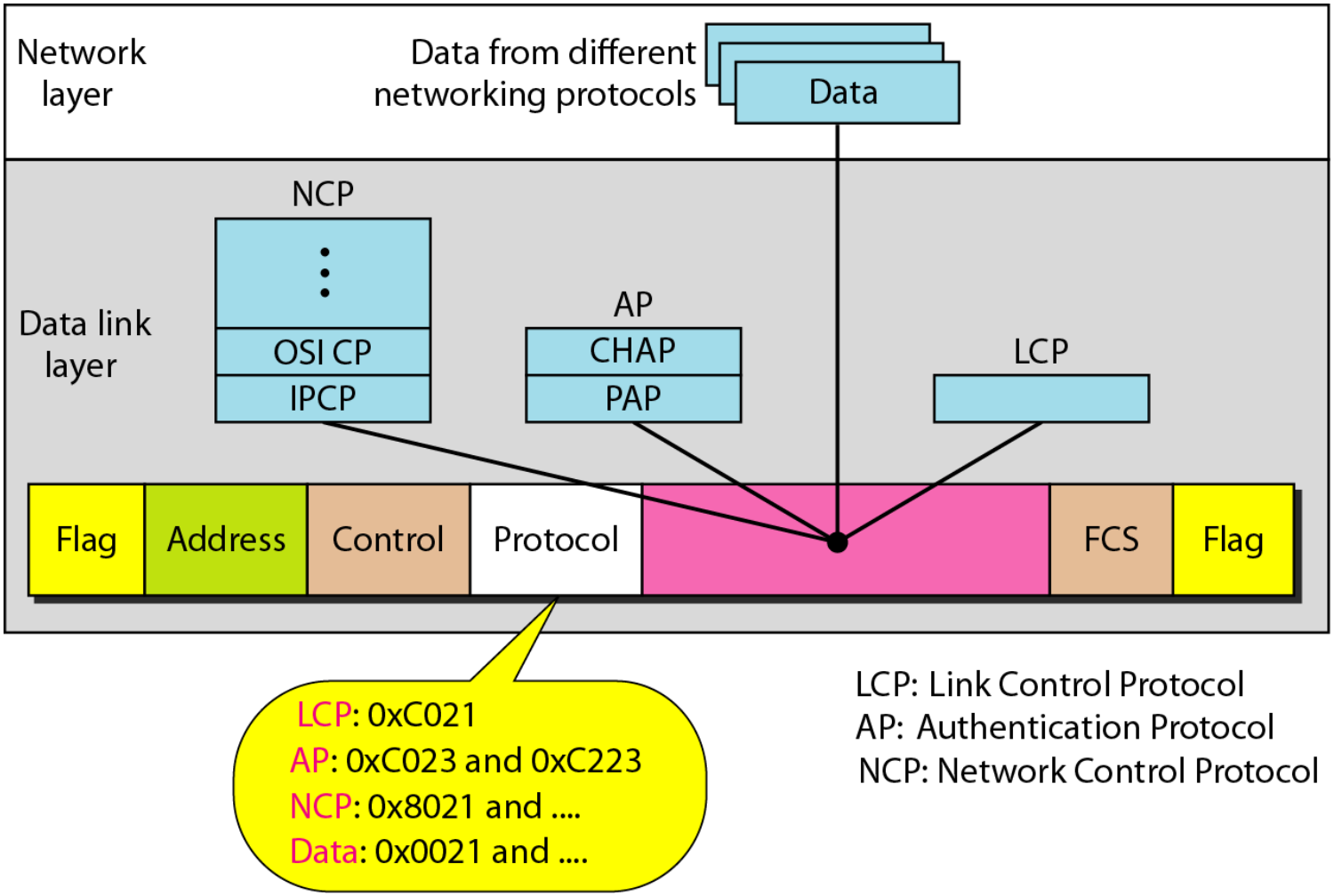


Figure 11.35 LCP 分组封装到帧中

LCP 负责建立、维护、确认并终止链路。

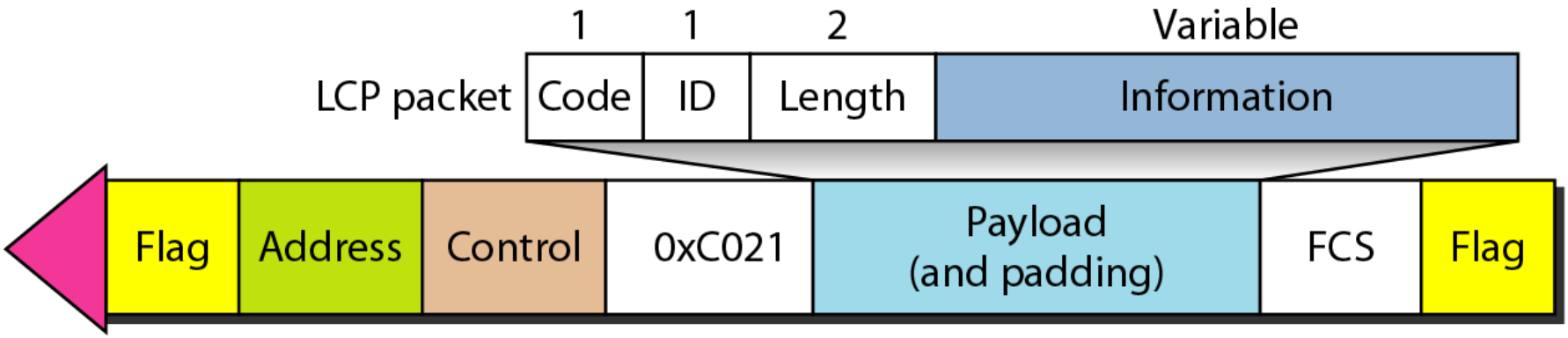


Table 11.2 LCP 分组

Code	*Packet Type*	*Description*
0x01	Configure-request	Contains the list of proposed options and their values
0x02	Configure-ack	Accepts all options proposed
0x03	Configure-nak	Announces that some options are not acceptable
0x04	Configure-reject	Announces that some options are not recognized
0x05	Terminate-request	Request to shut down the line
0x06	Terminate-ack	Accept the shutdown request
0x07	Code-reject	Announces an unknown code
0x08	Protocol-reject	Announces an unknown protocol
0x09	Echo-request	A type of hello message to check if the other end is alive
0x0A	Echo-reply	The response to the echo-request message
0x0B	Discard-request	A request to discard the packet

Table 11.3 常用选项

Option	*Default*
Maximum receive unit (payload field size)	1500
Authentication protocol	None
Protocol field compression	Off
Address and control field compression	Off

认证（鉴别）协议

- 分为两种：
  - 口令认证协议 PAP(Password Authentication Protocol)
  - 查询握手认证协议 CHAP(Challenge Handshake Authentication Protocol)

Figure 11.36 PAP 分组封装入 PPP 帧中

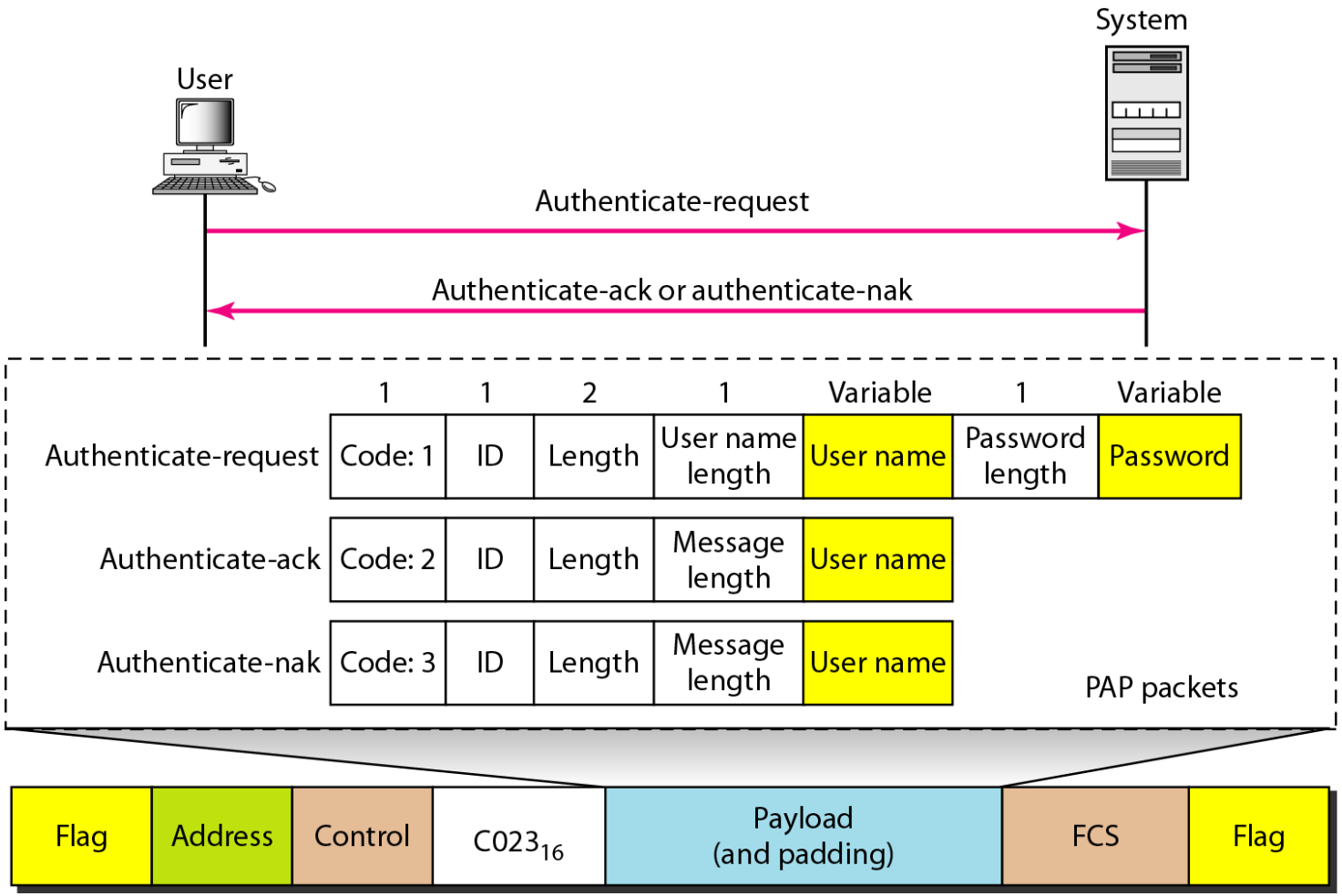


Figure 11.37 CHAP 分组封装入 PPP 帧中

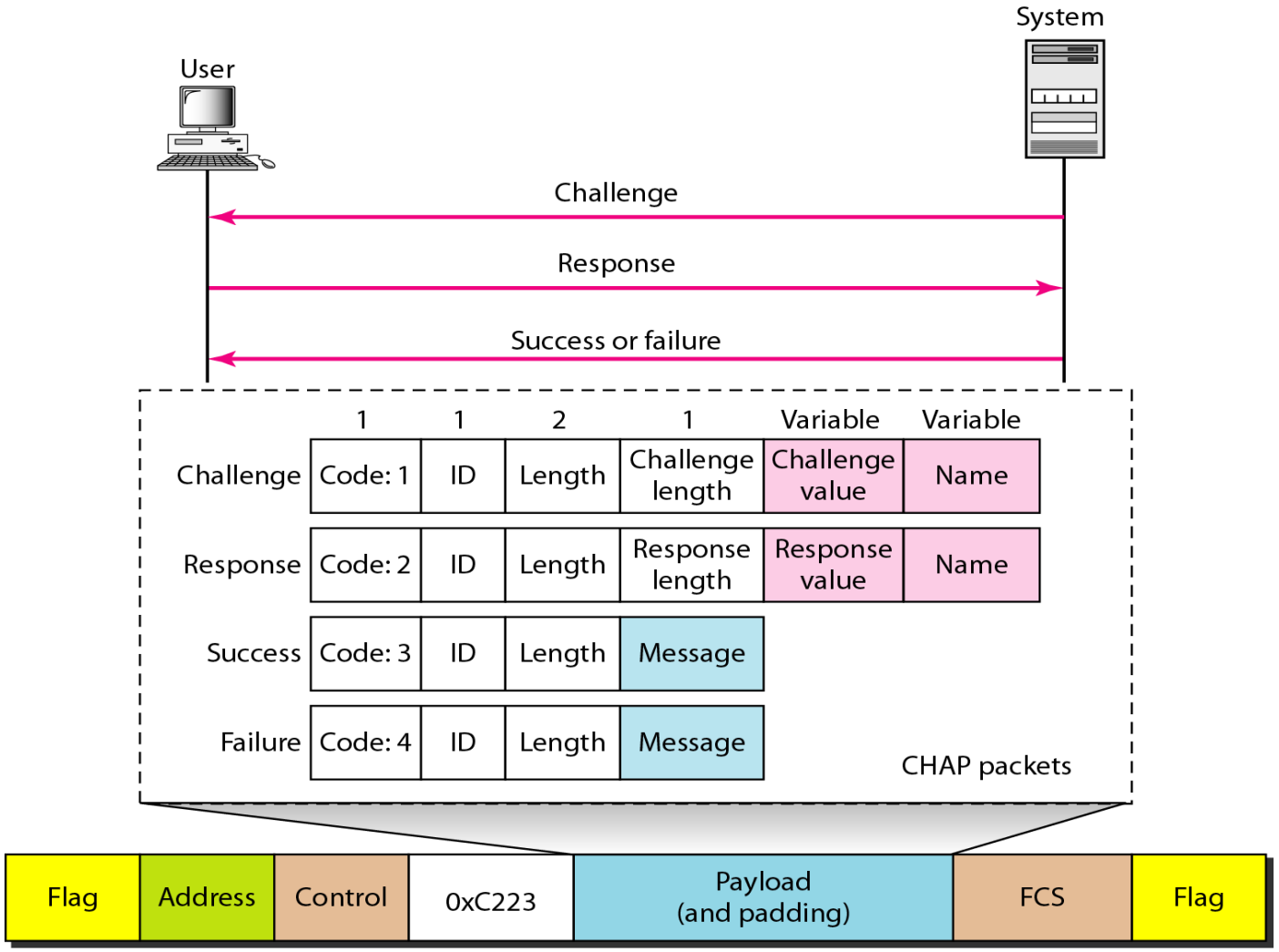


Figure 11.38 IPCP 分组封装入 PPP 帧中

网络控制协议， IPCP(Internet Protocol Control Protocol)

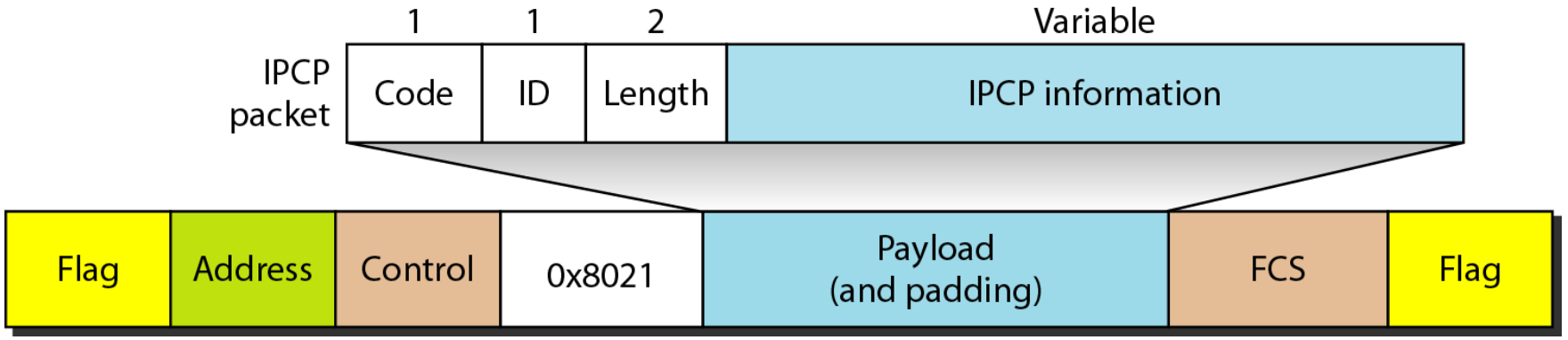


Table 11.4 IPCP 分组的编码值

Code	*IPCP Packet*
0x01	Configure-request
0x02	Configure-ack
0x03	Configure-nak
0x04	Configure-reject
0x05	Terminate-request
0x06	Terminate-ack
0x07	Code-reject

Figure 11.39 IP 分组封装入 PPP 帧中

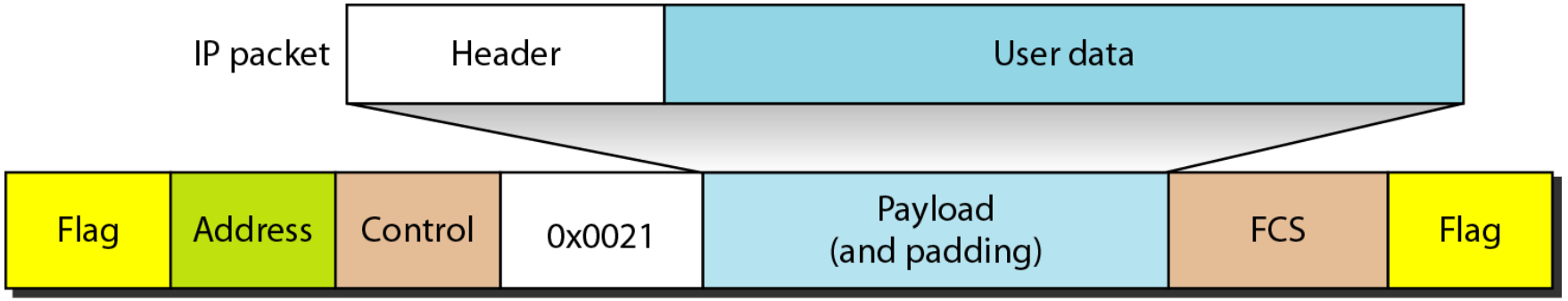


Figure 11.40 多链路 PPP

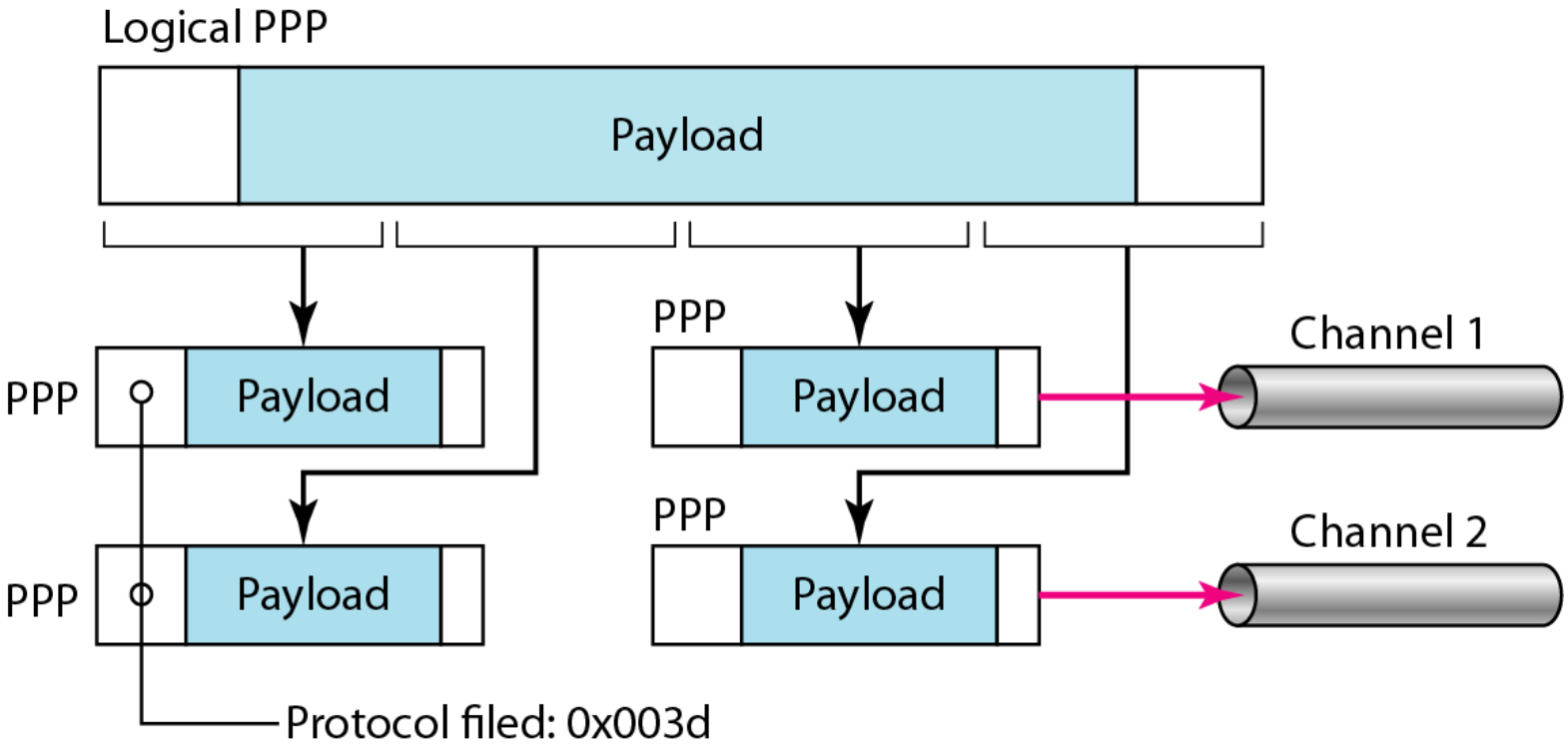


Figure 11.41 *An example*

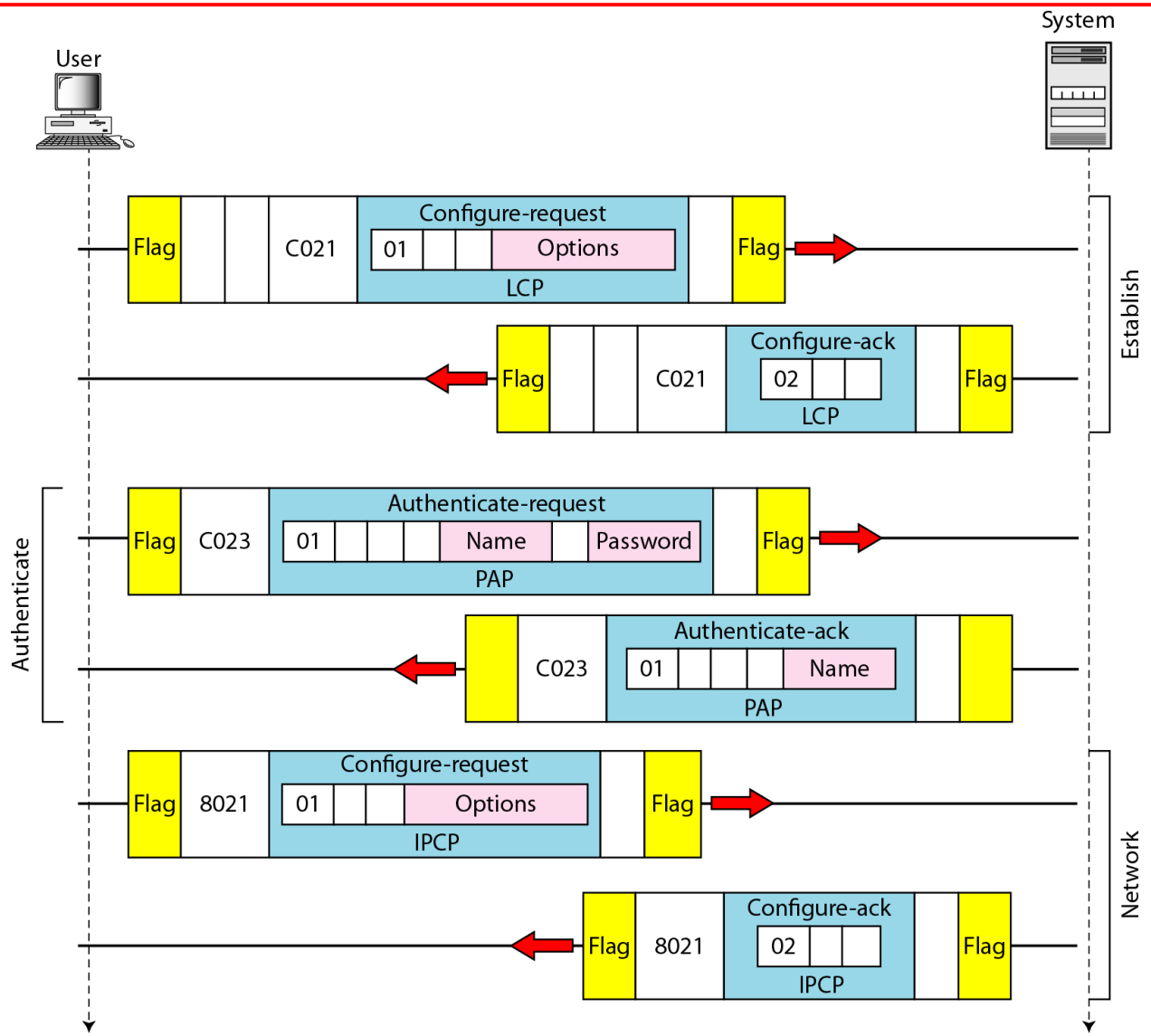
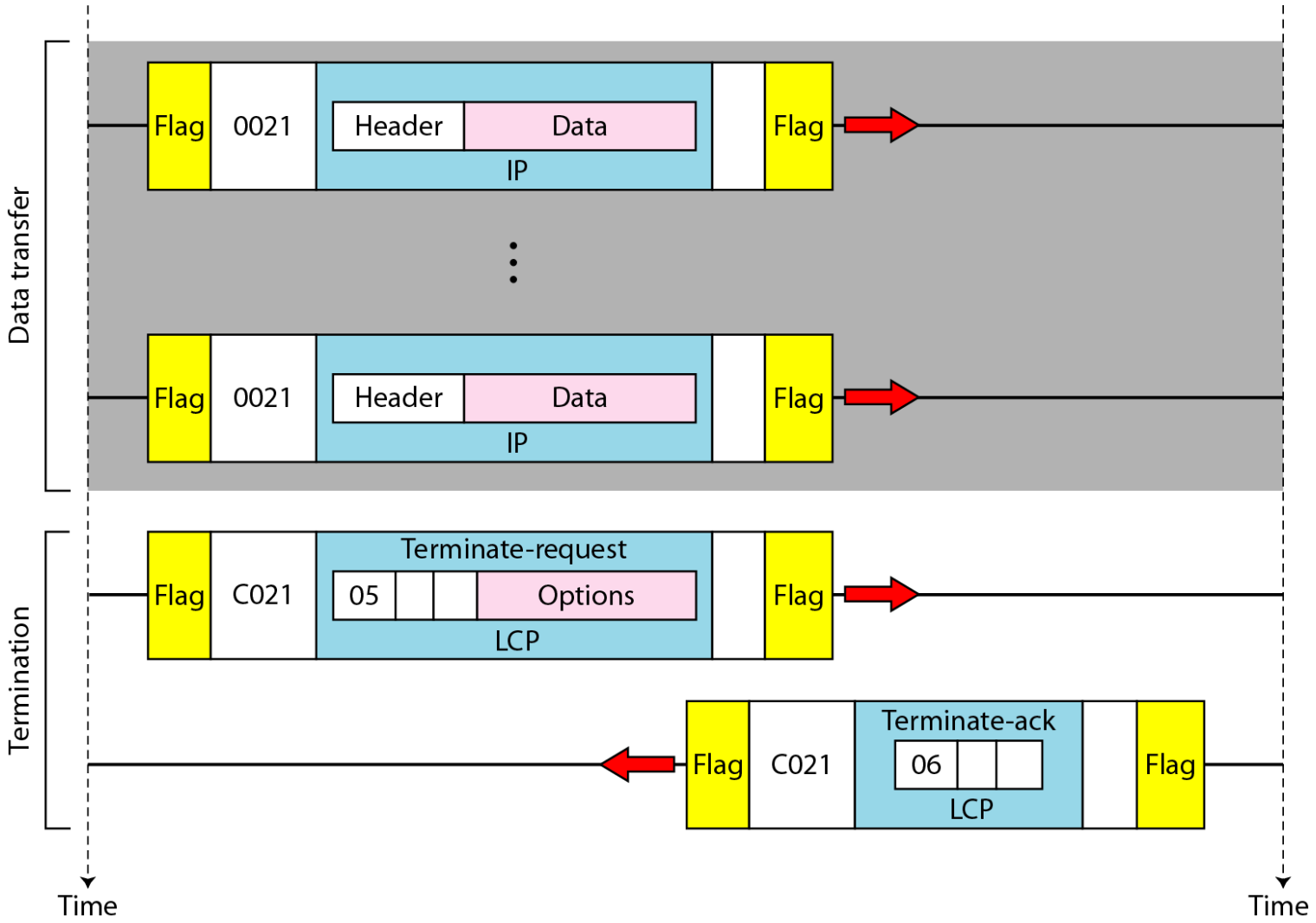


Figure 11.41 *An example (continued)*



11-8 PPPoE 协议（补充，一般了解）

PPPoE = PPP Over Ethernet

•用于实现 PPP 在以太网上的传输。

•是为了满足越来越多的宽带上网设备（如 ADSL--- 最初是静态 IP 、无线、有线电视等）和越来越快的网络之间的通信而指定开发的标准，它给出了两个广泛的接受的标准：以太网和 PPP 拨号协议。

•PPPoE 就是将 PPP 数据承载到以太网上，实质是在共享介质的网络中提供一条逻辑上的点到点链路（ Session ID ）。

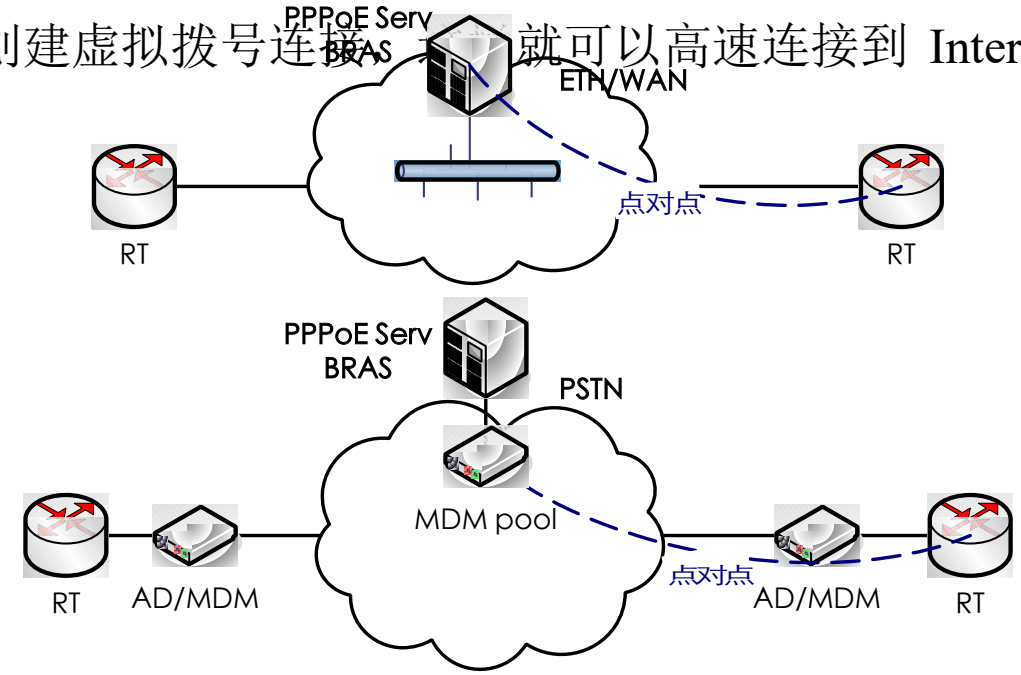
•PPPoE 主要协议标准： RFC2516

PPPoE 的由来

PPPoE 广泛应用在 ADSL 接入方式中。通过 PPPoE 技术和宽带调制解

调器（比如 ADSL Modem ）可以实现高速宽带网的个人身份验证访问

，为每个用户创建虚拟拨号连接，就可以高速连接到 Internet 。



作业

13， 17， 29， 31（改：传播速度 2×10^8m/s，传输速度 1Mbit/s）