第3章 数据链路层
——ARQ

刘志敏
liuzm@pku.edu.cn

数据链路层的功能

- 相邻结点间的数据传输存在哪些问题？

(1) 链路管理：建立链路、拆除链路

(2) **同步**：组帧、解帧

(3) 流量控制

(4) **差错控制**

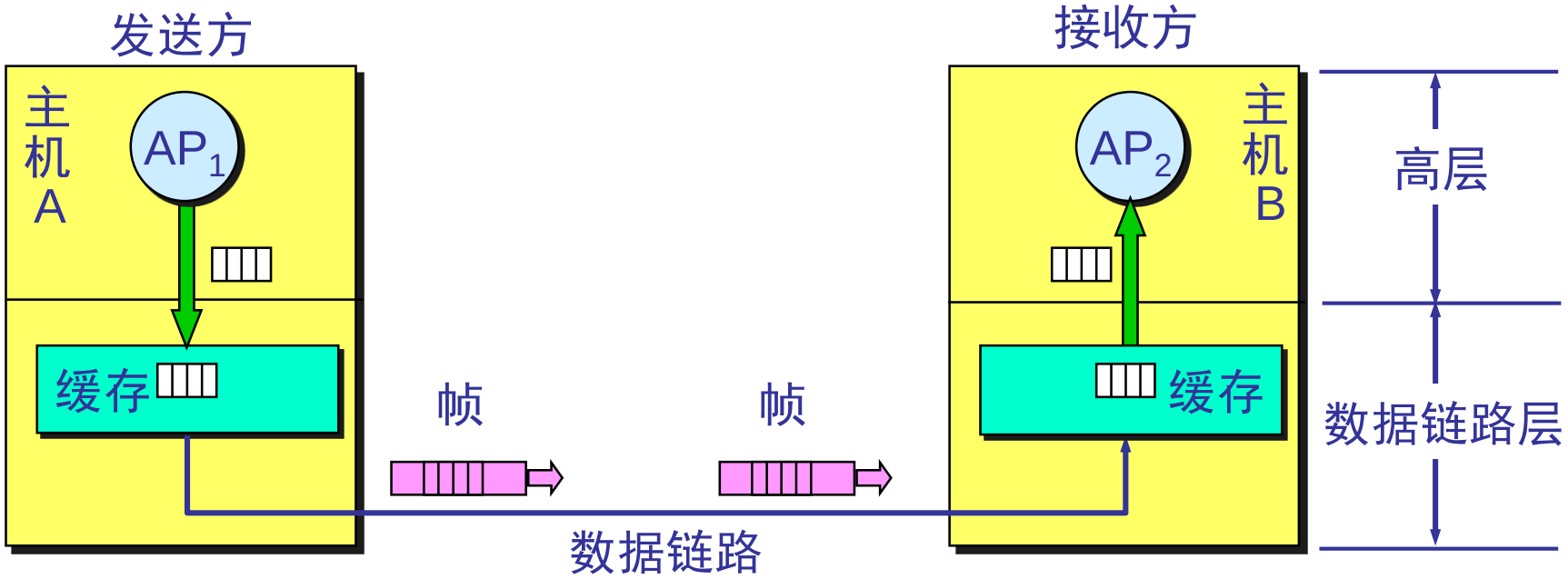
(5) 区分数据信息和控制信息

(6) **透明传输**

(7) 寻址

基本的数据链路层协议

- 检错及纠错：在帧头中增加控制信息，在帧尾增加帧校验
- 定义一系列的函数，如
  - wait_for_event(&event)；event，表示发生的事件
- 发送方：发送帧给另一主机
- 接收方：收到数据后计算帧校验，设置event；
  若有错，则event=chksum_err；否则event=frame_arrival



数据链路层协议程序

数据类型定义*protocol.h*

```
#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;
```

数据链路层协议程序

函数定义 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);
```

数据链路层协议程序V0.1

```
typedef enum {frame, arrive} event type;
#include "protocol.h"
void sender1(void)
{
   frame s;
   packet buffer;
   while(true){
     from_network_layer(&buffer);
     s.info = buffer;
     to_physical_layer(&s);
   }
}
```

```
#include "protocol.h"
void receiver1(void)
{
  frame r;
  packet buffer;
  event type event;
  while(true){
     wait_for_event(&event);
     physical_layer(&r)
     to_network_layer(&buffer);
   }
}
```

- 这一对程序运行之后有何问题？
- 若发送得太快来不及接收或接收未准备好，则导致数据丢失。要根据收端的情况控制发端的速率——流量控制！

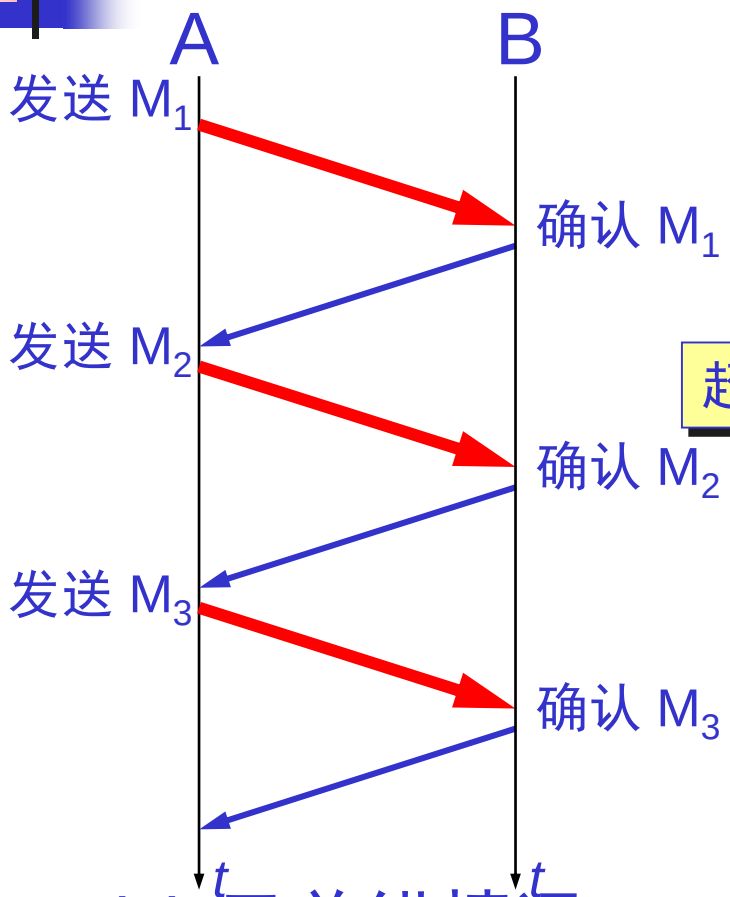
数据链路层协议程序V0.2

```
typedef enum {frame, arrive} event type;
#include "protocol.h"
void sender2(void)
{
  frame s;
  packet buffer;
  while(true){
     from_network_layer(&buffer);
     s.info = buffer;
     to_physical_layer(&s);
     wait_for_event(&event);
   }
}
```

```
#include "protocol.h"
void receiver2(void)
{ frame r,s;
  packet buffer;
  event type event;
  while(true){
     wait_for_event(&event);
     physical_layer(&r)
     to_network_layer(&buffer);
     to_physical_layer(&s);
   }
}
```

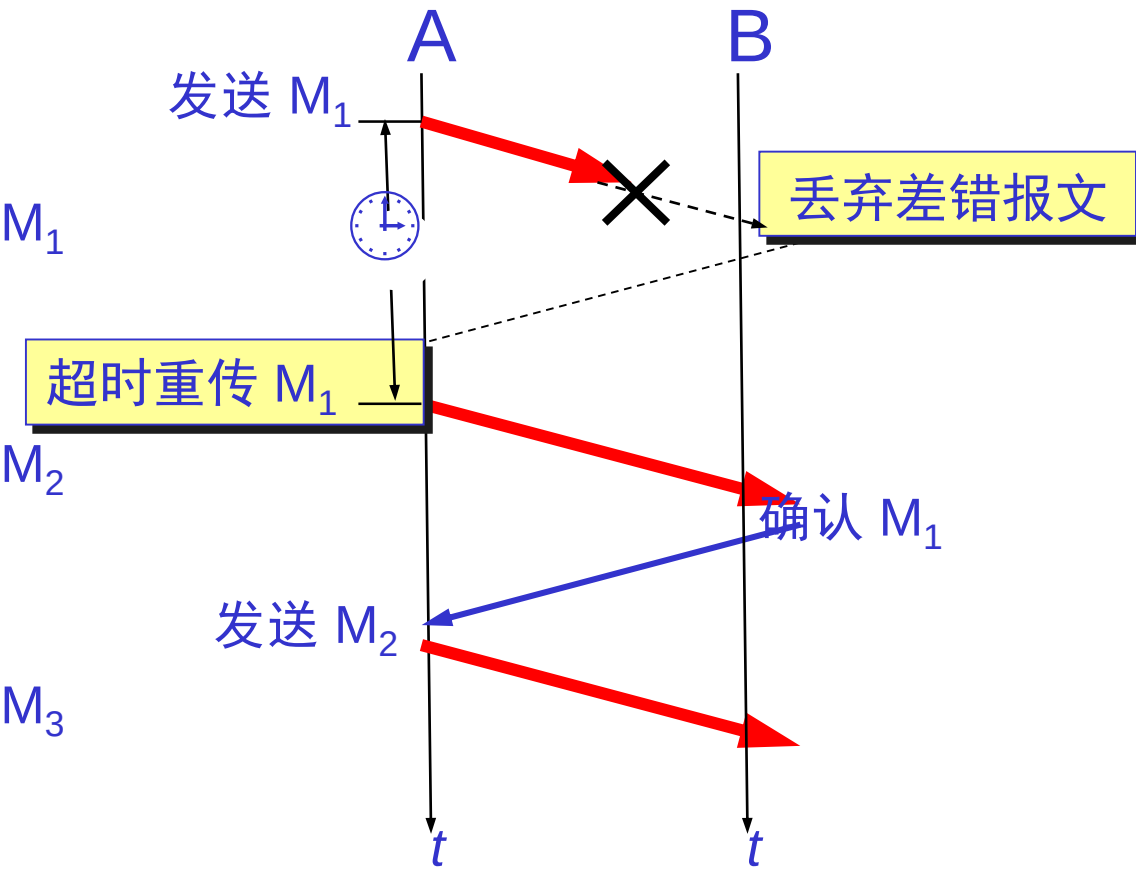
- 这一对程序运行之后还有何问题？
- 发送帧可能会错，需要接收端确认；
- 若发端没有收到确认，原因是什么？怎么办？

停等协议的原理



(a) 无差错情况

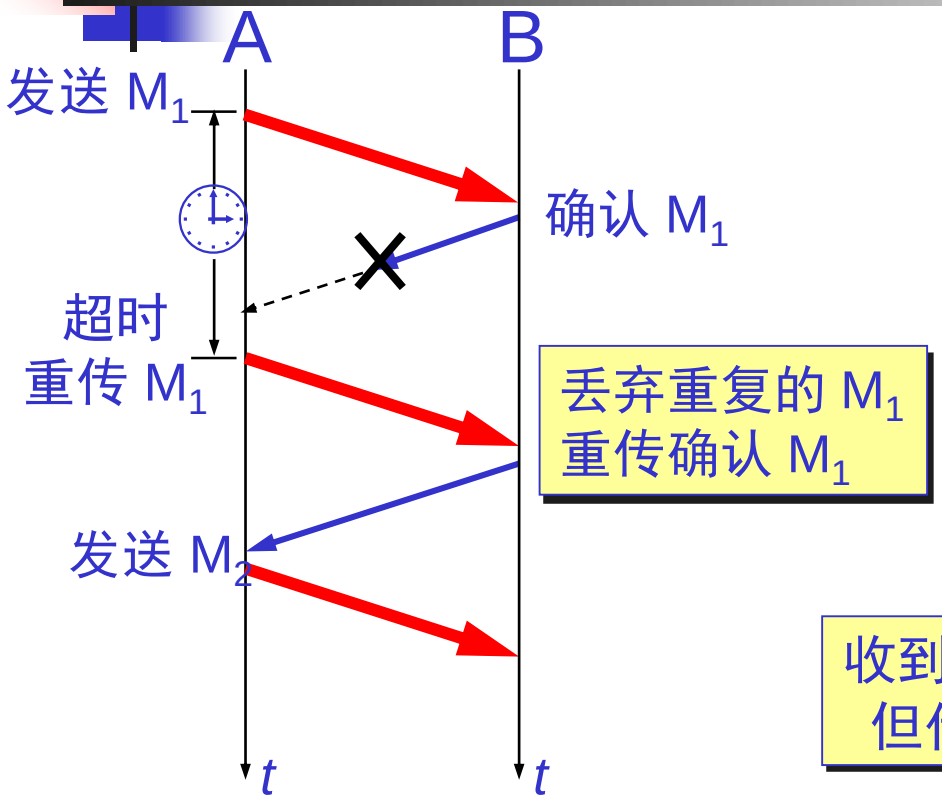
收端对发端确认，
实施流量控制



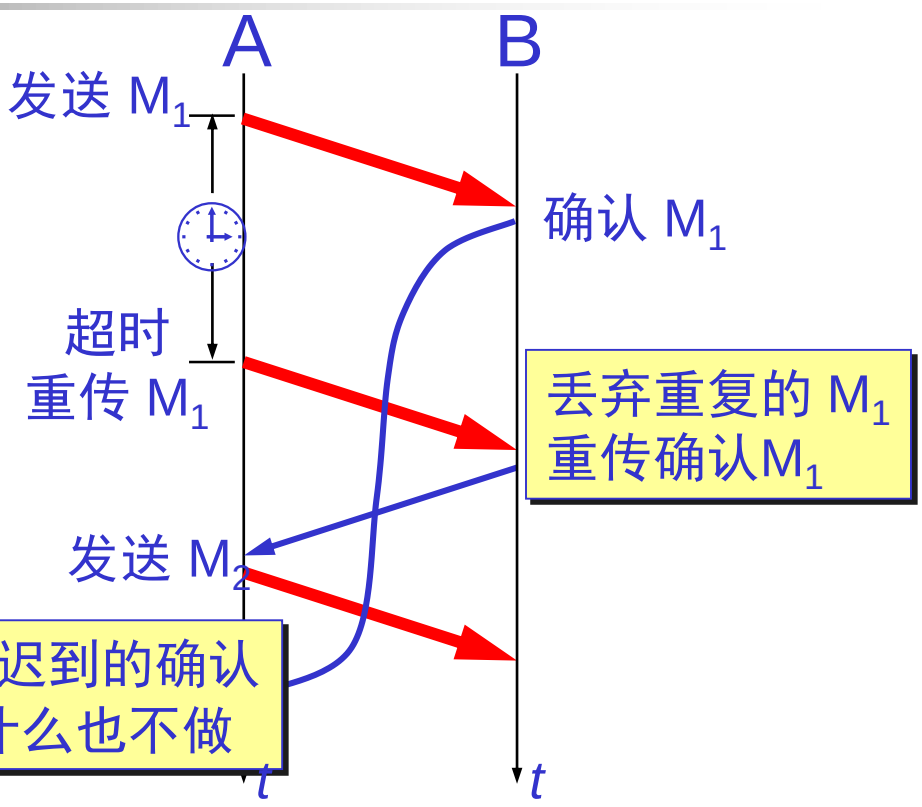
(b) 数据有差错，超时重传

增加帧校验及超时定时器，
发端对出错的帧自动重传

停等协议的原理



(c) 确认帧丢失



(d) 确认迟到

避免重复帧：数据帧带序号，
每发一个新数据帧，序号加1；

确认帧：带序号

数据链路层协议程序V0.3

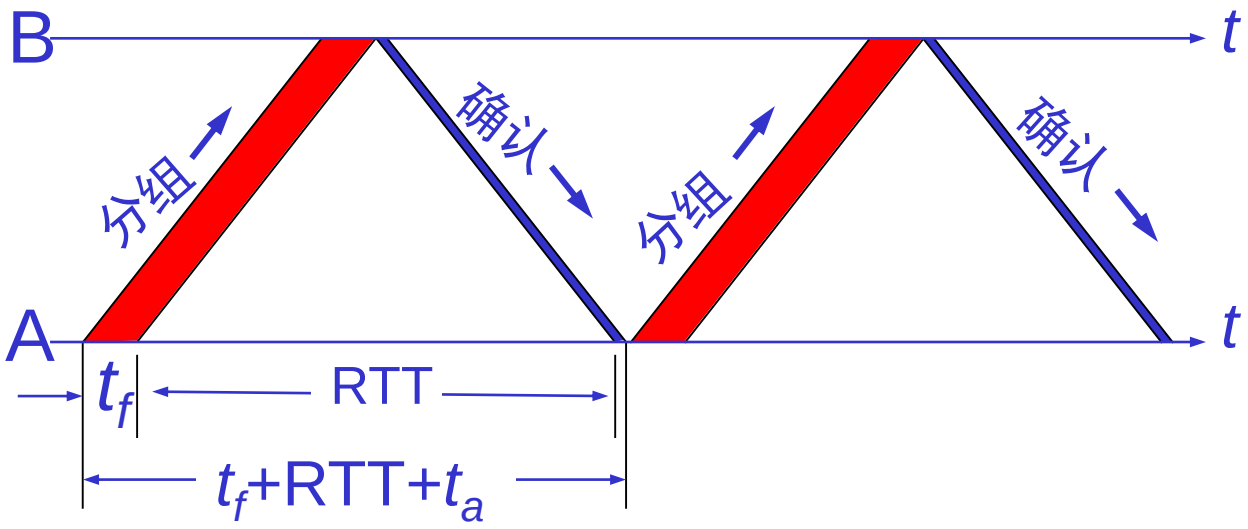
```
typedef enum {frame_arrive} event type;
#define MAX_SEQ1
#include "protocol.h"
void sender3(void)
{ seq_nr next_frame_to_send=0;
  frame s;
  packet buffer;
  from_network_layer(&buffer);
  while(true){
     s.info = buffer;
     s.seq = next_frame_to_send;
     to_physical_layer(&s);
     start_timer(s.seq);
     wait_for_event(&event);
     if(event== arrive)
         from_physical_layer(&s);
     if(s.ack==next_frame_to_send){
        stop_timer(s.ack):
        from_network_layer(&buffer);
        inc(next_frame_to_send);     }
    }
}
```

```
#include "protocol.h"
void receiver3(void)
{ seq_nr frame_expected=0;
  frame r,s;
  packet buffer;
  event type event;
  while(true){
     wait_for_event(&event);
     if(event == frame_arrive){
        from_physical_layer(&r);
        if(r.seq==frame_expected){
            to_network_layer(&r.info);
            inc(frame_expected);        }
        s.ack=1-frame_expected;
        s.info = buffer;
        to_physical_layer(&s);    }
}
```

停等协议的要点

- 发端发送数据帧时同时缓存；当收到确认帧后，才清除。
- 只有收到确认帧ACK后，才发送新的数据帧。
- 接收端收到数据帧时，发送确认；若是新数据帧，则保存；若为重复帧，则丢弃。
- 发送端自动对错帧重传，因此称为自动请求重传ARQ (Automatic Repeat reQuest)。

信道利用率



- 信道利用率U = $\frac{t_f}{t_f+RTT+t_a}$
- 停等协议的优点是简单，但在长时延信道上，信道利用率变低

举例

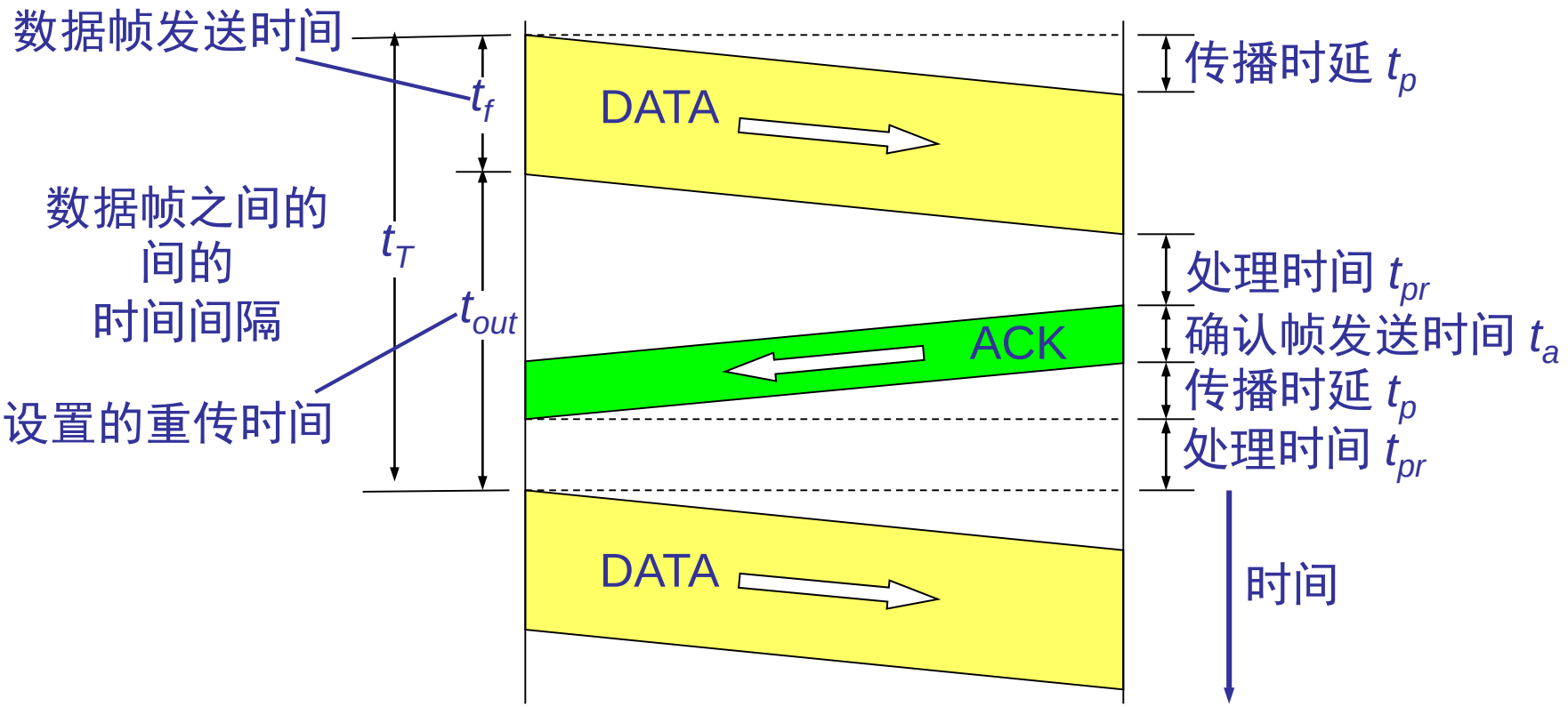
- 一条链路，速率为2Mbps，往返时延为40ms
- 延迟带宽积=2Mbps×40ms=80Kb=10KB
- 设帧长为1KB，则最大发送速率为1000×8/0.04=200Kbps，为链路速率的1/10，远小于2Mbps

- 如何提高链路利用率？
- 如何分析停等协议的性能？

停等协议的定量分析

- 设数据帧长度 l_f (bit)，数据发送速率 R (bit/s)，则一个数据帧的发送时间t_f

$$t_f = l_f / R \quad \text{(s)} \qquad (3\text{-}1)$$



计算重传时间及信道利用率

- 数据帧发送后若经过重传时间还没有收到确认帧，就重传数据帧。 设重传时间为

$$t_{out} = t_p + t_{pr} + t_a + t_p + t_{pr} \qquad (3\text{-}2)$$

- 处理时间 t_{pr} 和确认帧的发送时间 t_a 远小于传播时延 t_p，予以忽略，因此，$t_{out} = 2t_p$ (3-3)

- 定义$\alpha=\frac{t_p}{t_f}$，表示传播时间（即延迟）与帧发送时间（为带宽的倒数）之比

- 信道利用率$U=\frac{t_f}{t_f+2t_p}=1/(1+2\alpha)$

例：计算停等协议信道利用率

- 例：信道速率为2Mbps，单向传播延迟为20ms，问帧长在什么范围内，才使停等协议的效率至少为50%？若信道误码率为10^{-6}，求误帧率。

$$u = \frac{1}{1 + 2t_p R / l_f} \geq 50\%$$

$$l_f \geq 2t_p R$$
$$= 2 \times 2000K \times 0.02$$
$$= 80Kb = 10KB$$

- 误帧率=1-(1-p)^L= 80K×10^{-6}=0.08

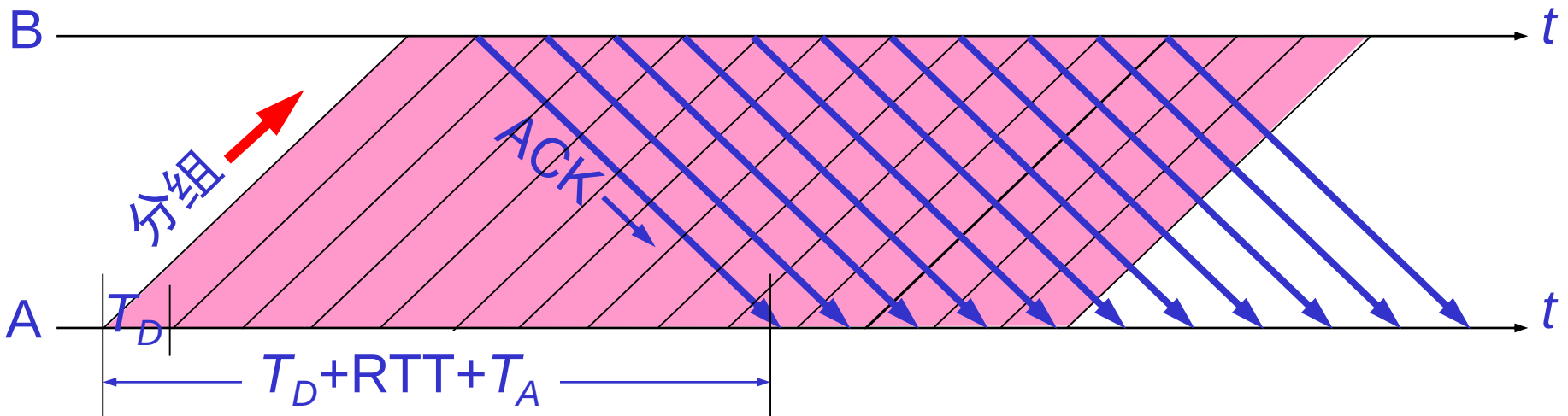
若传输信道有误码，仅增加帧长是无法有效提高链路传输效率的！

讨论

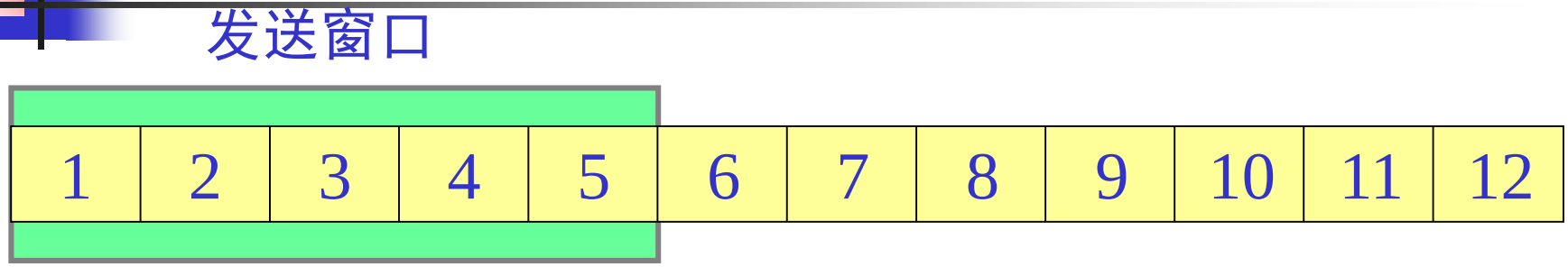
- 停等协议的优缺点：
  - 优点：简单
  - 缺点：信道利用率不高，即信道远没有被数据充满，尤其不适于在长时延的信到上采用停等协议
- 如何克服缺点？
  - 连续 ARQ
  - 选择重传ARQ
  - 多通道（或多进程）的停等协议（在LTE系统中采用）

连续发送多个帧

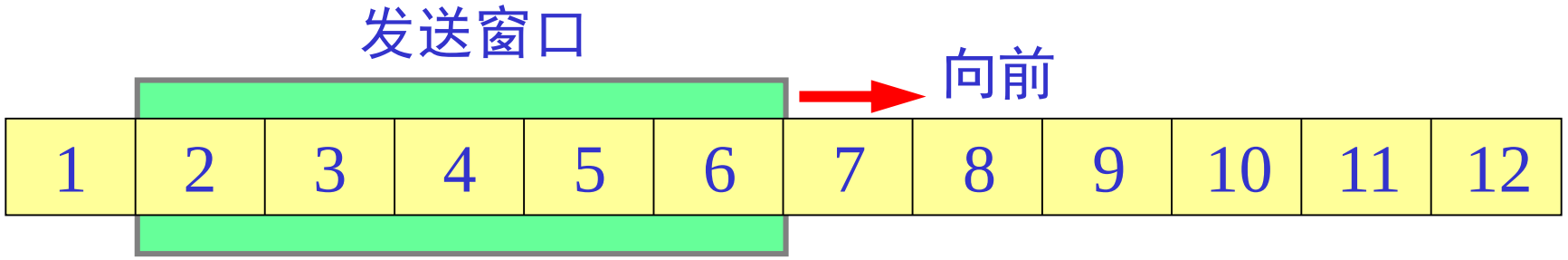
- 发送方连续发送多个分组，不必每发完一个分组就停下来等待对方的确认。
- 因信道上有多帧在传送，提高了信道利用率
- 若延迟为t_p，带宽为帧率$\frac{1}{t_f}$（即单位时间内的发送的帧数），则延迟带宽积为$\alpha=\frac{t_p}{t_f}$
- 若可连续发送的帧数为w，则利用率$\leq \frac{w}{1+2\alpha}$



连续 ARQ 协议



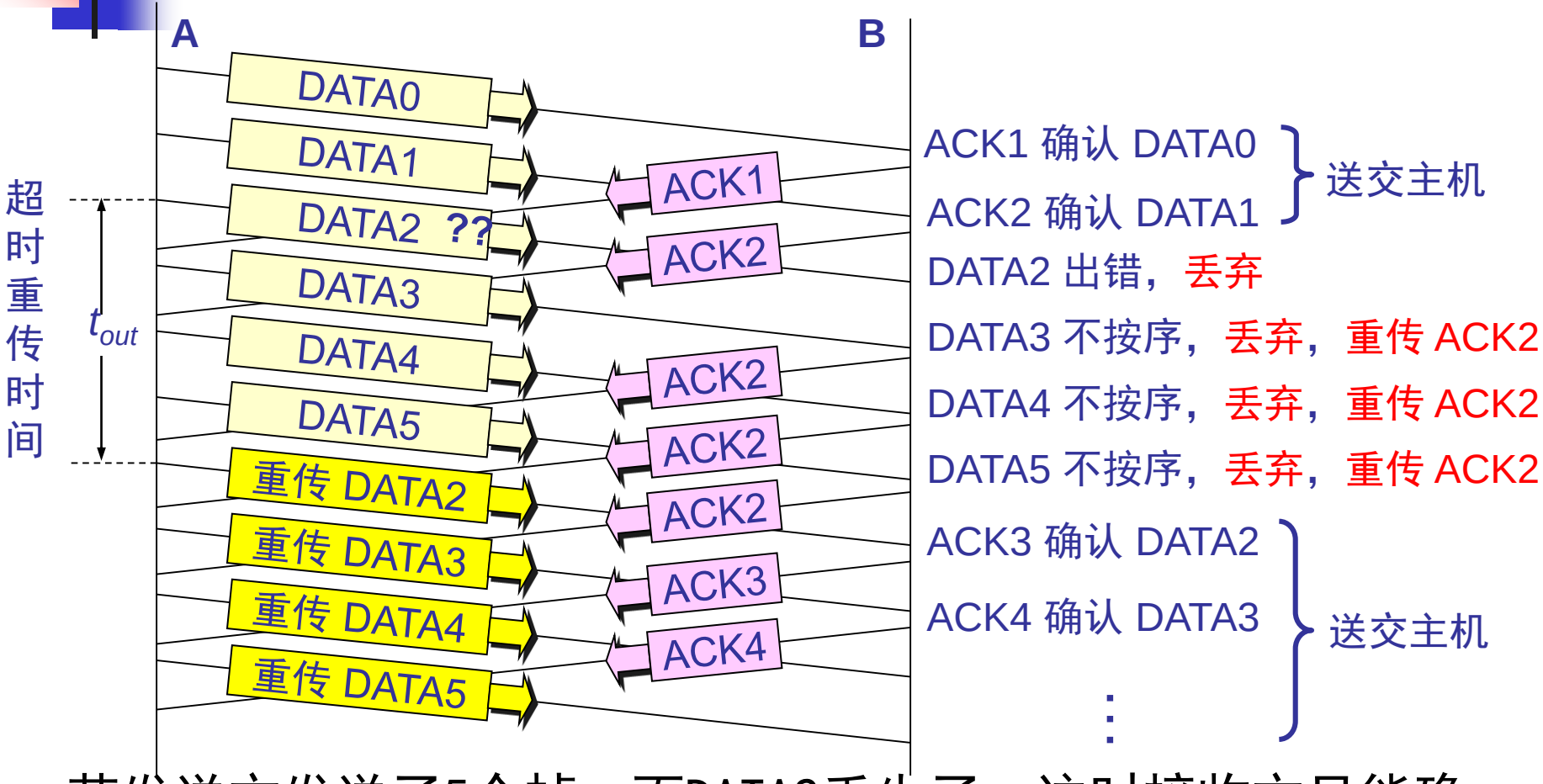
(a) 发送方维持发送窗口（发送窗口是 5）



(b) 收到一个确认后发送窗口向前滑动

- 发送方：只允许发送在发送窗口W个帧；每发送一个帧，启动一个定时器，等待确认
- 接收方：仅接收序号连续的帧，如收到0号帧则响应ACK1，收到1号帧则响应ACK2；ACKi表示期望收到的下一个帧的序号
- 发送方：收到ACKi则取消定时器，调整发送窗口为[i, i+W-1]
- 发送方：若第j个帧定时器到，则调整发送窗口为[j, j+W-1]

连续 ARQ 协议



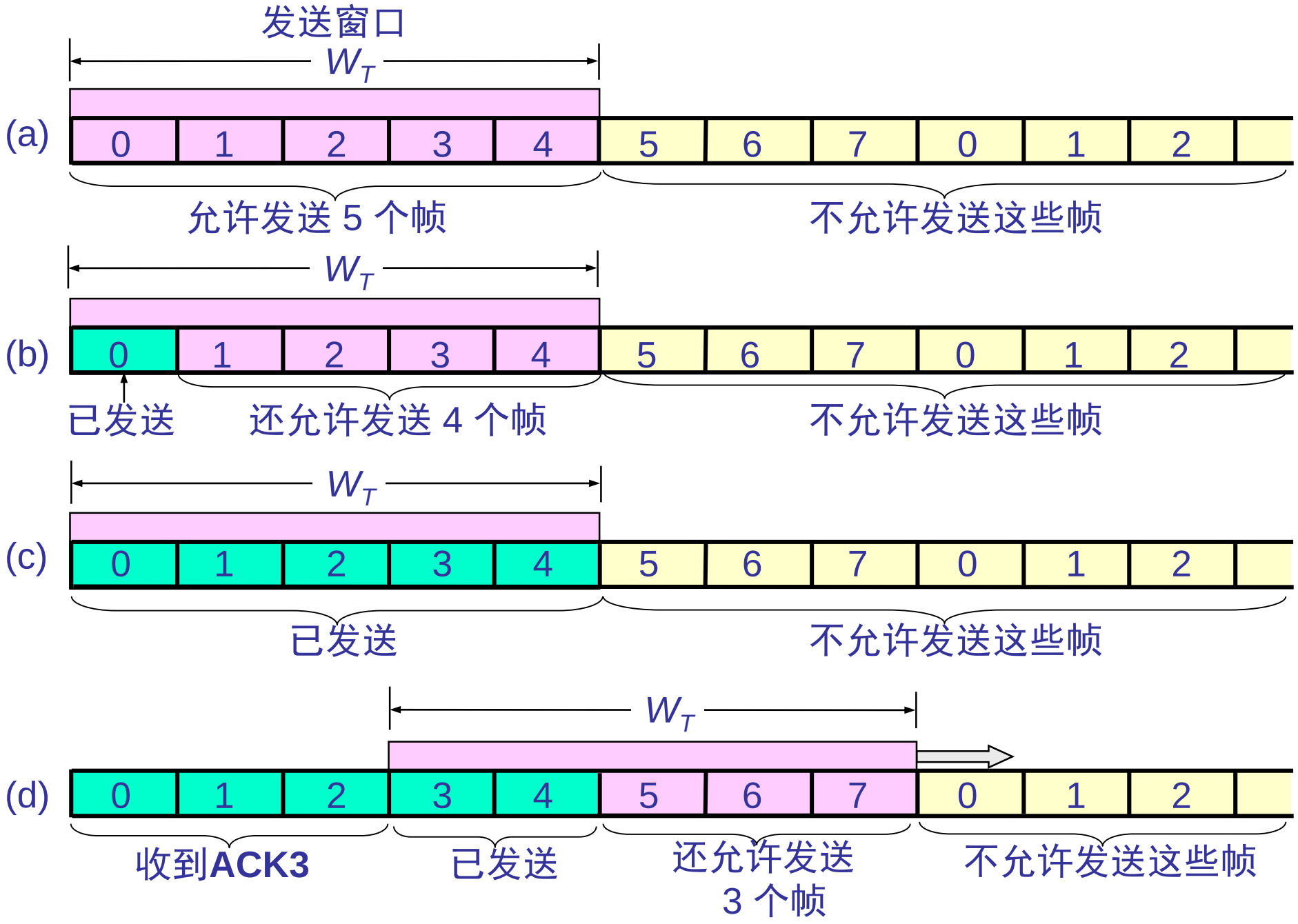
若发送方发送了5个帧，而DATA2丢失了。这时接收方只能确认前2个数据帧。发送方只能从DATA2帧开始重传，因此称之为Go-back-N（回退 N），表示退回来重传已发送的N个分组。可见在质量不好的链路上，不适于采用连续ARQ

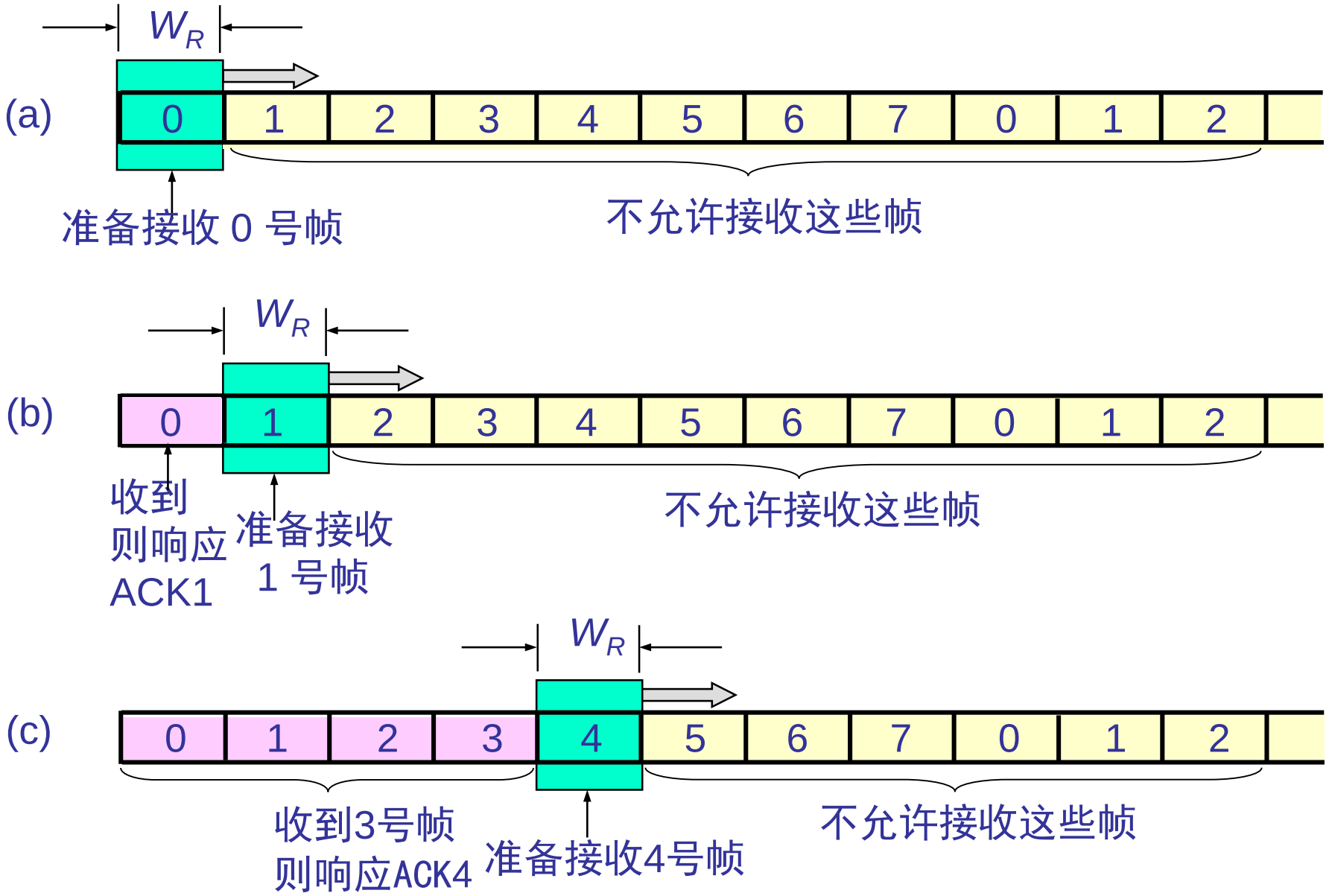
连续 ARQ 的工作原理

- 发送端：发送一个数据帧后，连续再发送若干个数据帧。如果收到了确认帧，则接着发送数据帧。
- 接收端：按序接收数据帧。
- 发送端：每发送一个数据帧都设置超时计时器。收到ACKn时，则取消第（n-1）数据帧的定时器。当第n数据帧定时器超时时，自ACKn开始重传数据帧。
- 连续 ARQ 又称为Go-back-N ARQ。

滑动窗口的概念

- 发送端和接收端分别设置发送窗口和接收窗口
- 发送窗口W_T用来对发送端进行流量控制，表示发送端最多可发送的数据帧数量
- 接收端只接收发送序号落入接收窗口的数据帧


发送窗口
W_T
(a)
0 1 2 3 4 5 6 7 0 1 2
允许发送 5 个帧
不允许发送这些帧
W_T
(b)
0 1 2 3 4 5 6 7 0 1 2
已发送
还允许发送 4 个帧
不允许发送这些帧
W_T
(c)
0 1 2 3 4 5 6 7 0 1 2
已发送
不允许发送这些帧
W_T
(d)
0 1 2 3 4 5 6 7 0 1 2
收到ACK3
已发送
还允许发送
3 个帧
不允许发送这些帧


W_R
(a)
0 1 2 3 4 5 6 7 0 1 2
准备接收 0 号帧
不允许接收这些帧
W_R
(b)
0 1 2 3 4 5 6 7 0 1 2
收到
则响应
ACK1
准备接收
1 号帧
不允许接收这些帧
W_R
(c)
0 1 2 3 4 5 6 7 0 1 2
收到3号帧
则响应ACK4
准备接收4号帧
不允许接收这些帧

有关滑动窗口协议

- 在连续ARQ中，接收窗口$W_R = 1$
- 收发两端的窗口不断向前滑动，协议又称为滑动窗口协议
- 若W_T=1，则为停止等待协议
- 发送窗口的最大值：当帧序号用 n 个比特编号时，若接收窗口为1，则发送窗口为

$$W_T \leq 2^n - 1$$

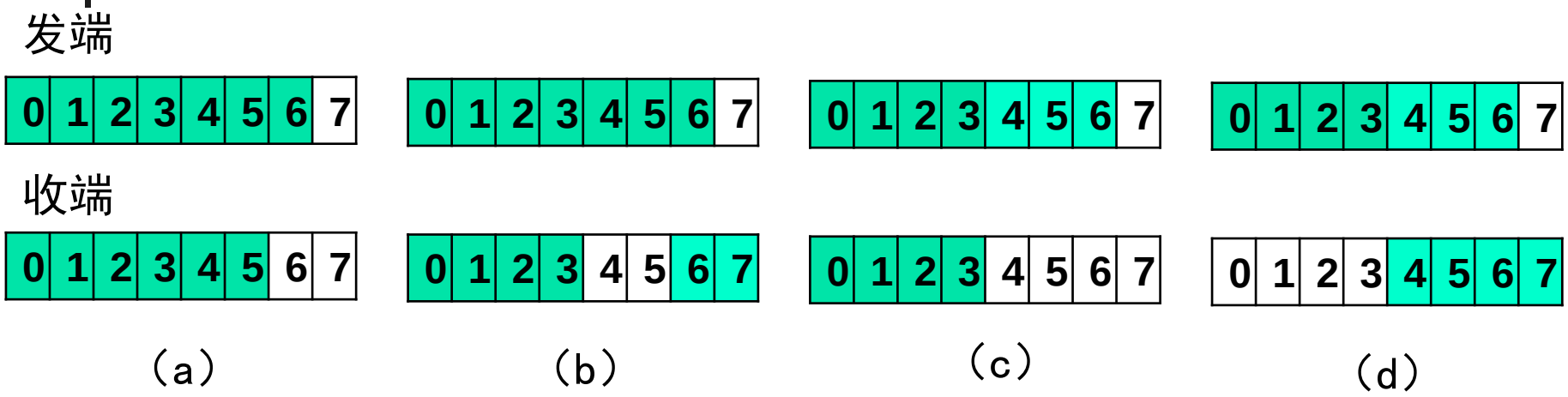
时，连续ARQ才能正确运行。（试证明）

选择重传 ARQ

- 增大接收窗口，接收序号不连续但仍在接收窗口中的数据帧。等收到所缺序号的数据帧后再送交主机
- 选择重传 ARQ 避免重传那些正确到达接收端的数据帧。但代价是在接收端要设置一定的缓存
- 对于选择重传 ARQ 协议，若用n比特进行编号，则接收窗口的最大值

$$W_R \leq 2^n/2$$

选择重传协议



- 设序号用3位表示；发送窗口为7，接收窗口为6

(a)发送0~6帧后收端确认0~5，但确认帧全部丢失，发端重发0~6
(b)收端接收窗口调整为6~3，但发端窗口为0~6；

- 问题：当收端移动窗口时，新序号范围6~7与旧序号范围0~3有重叠，后续帧可能是重复的或新的，导致接收端无法判定
- 解决方法：避免新旧接收窗口重叠！故接收窗口极大值为$2^n/2$。例如（c）发送0-6帧 （d）收端发送确认，接收窗口为4-7，确认帧全部丢失，则发送端重发0-6，则4-6帧被接收

捎带确认与累计确认

- 发送数据帧是有效的数据传输，而发送确认帧为协议开销；如何降低开销？例如降低发送确认帧的频度？采用累计确认和捎带确认！
- 累计确认：接收端不是每收到一个数据帧就发送一个确认帧，若确认序号为i，则表示 (i-1)及之前的所有数据帧均被确认，且期望收到的下一个帧的序号为i。
- 捎带确认：当接收端向发送端发送确认信息时，如果同时存在需要发送的数据，那么可以将确认信息和数据合并在一个帧中发送

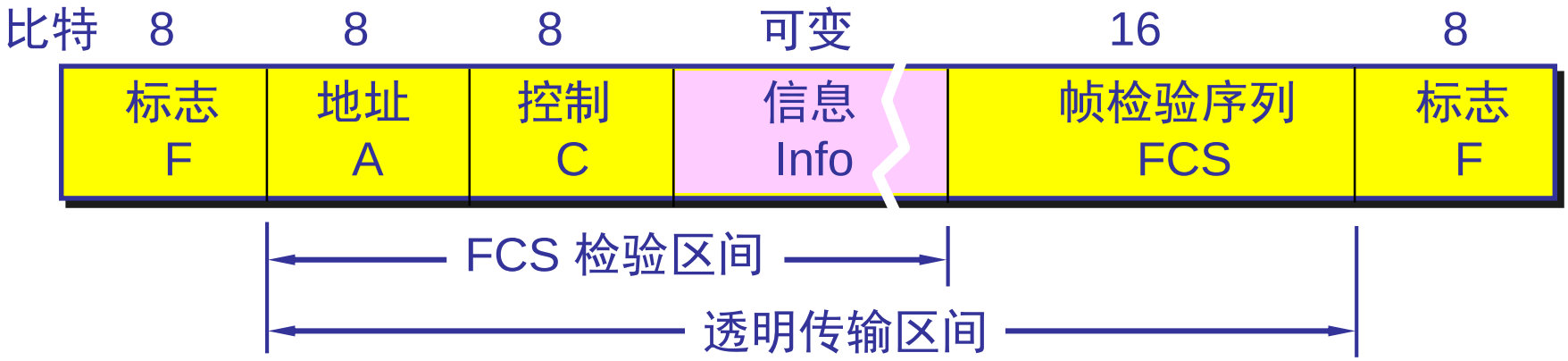
问题：

- 实现滑动窗口协议的程序？
- 设置发送窗口大小需要考虑哪些因素？
- 设置数据帧超时定时器的长度又与哪些因素有关？
- 实际系统的链路速率、传输时延、缓存能力差异很大，如何选择发送窗口以及超时定时器？

- TCP协议给出了最佳的解决方案

链路控制规程 HDLC

HDLC 的帧结构

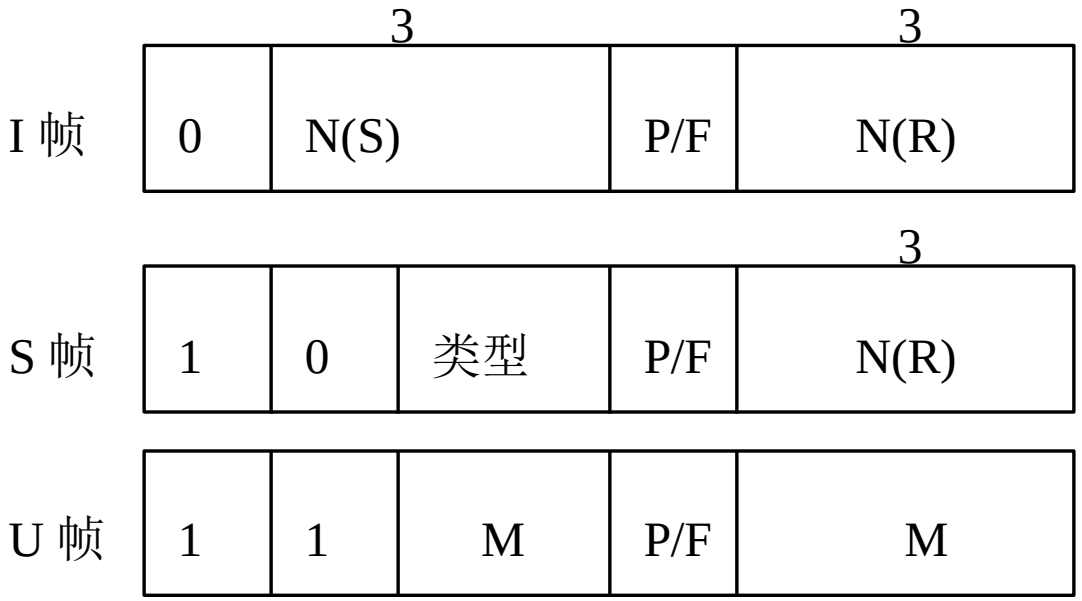


- 标志字段 F (Flag)：01111110（0x7E）。
- 地址字段 A： 占8 bit。
- 控制字段 C： 占8 bit，根据其判断帧的类型。
- 帧检验序列 FCS：占16 bit。

HDLC支持三种类型的帧

- 信息 I 帧：携带用户数据，帧的序号在N(S)中，支持捎带确认：N(R)表示之前的所有帧都正确
- 监控 S 帧：单独确认机制
- 无编号 U 帧：其它链路控制功能

其中，控制字段C的内容：



N(S)：发送顺序号

N(R)：接收顺序号

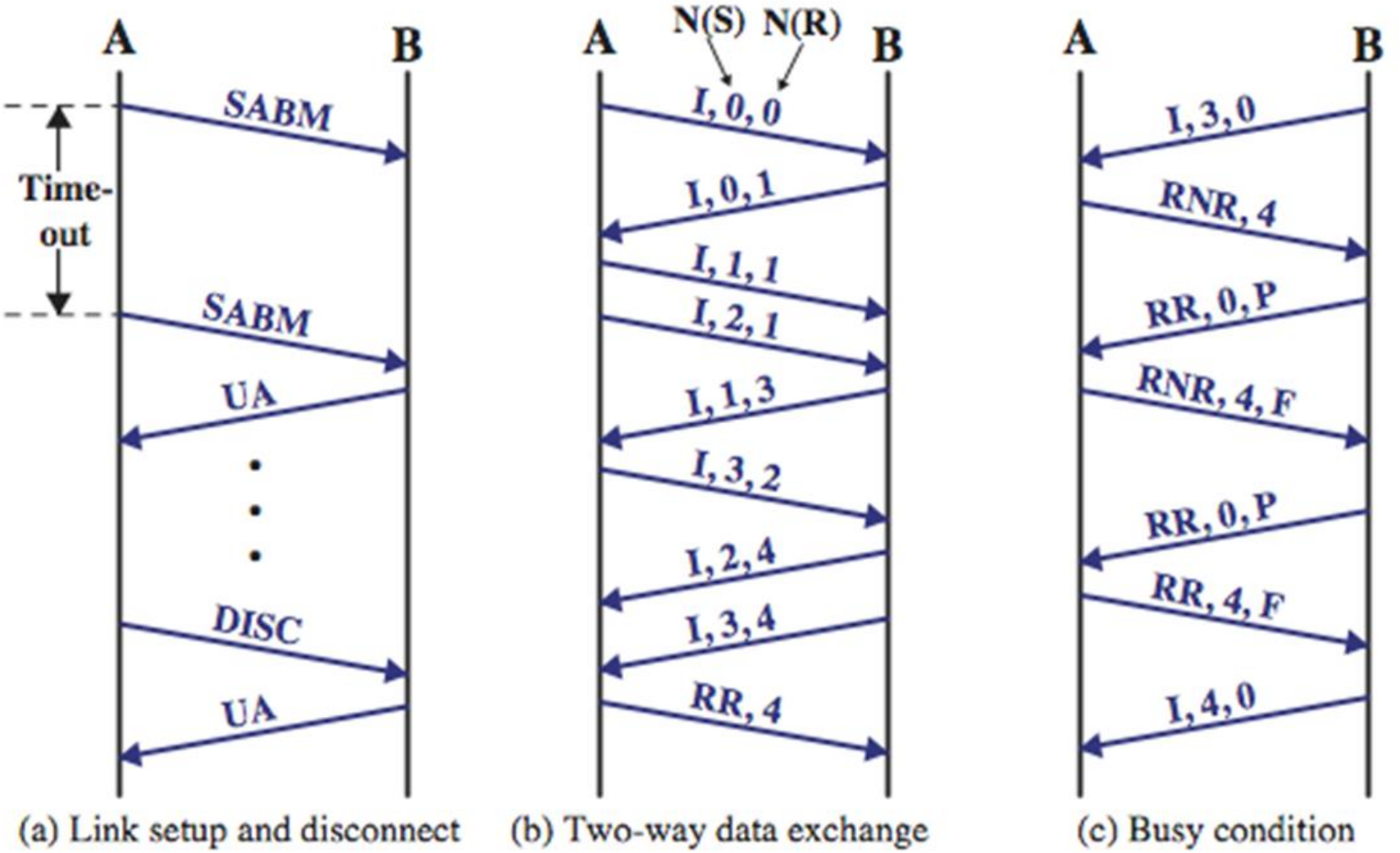
M：U 帧功能编码

P/F：Poll/Final

HDLC 的帧结构（续）

- S帧：
  - RR：positive ACK，准备接收I帧
  - RNR：positive ACK，尚未准备好接收
  - REJ：Negative ACK，回退N
  - SACK：Negative ACK, 选择重传
- U帧：典型的U帧
  - SABM：设置ABM模式
  - DM：断开模式
  - DISC：终止链路连接
  - UA：确认U帧
  - RSET：重置位，reset N(R),N(S)

HDLC工作过程举例

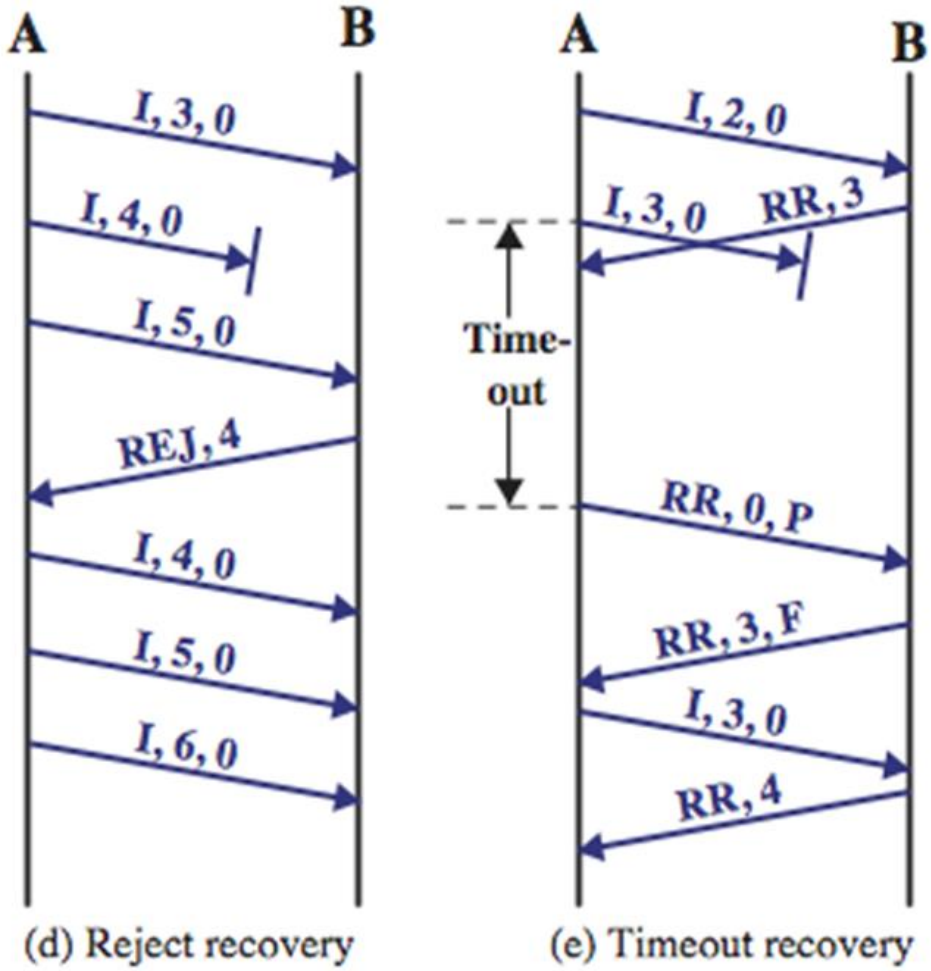


(a) Link setup and disconnect (b) Two-way data exchange (c) Busy condition

图(B)：全双工 I 帧交换。注意N(S) N(R)，采用捎带确认

图(C)：接收忙的情形。接收方发送RNR令发送方停止发送，需要继续发送时，发送RR

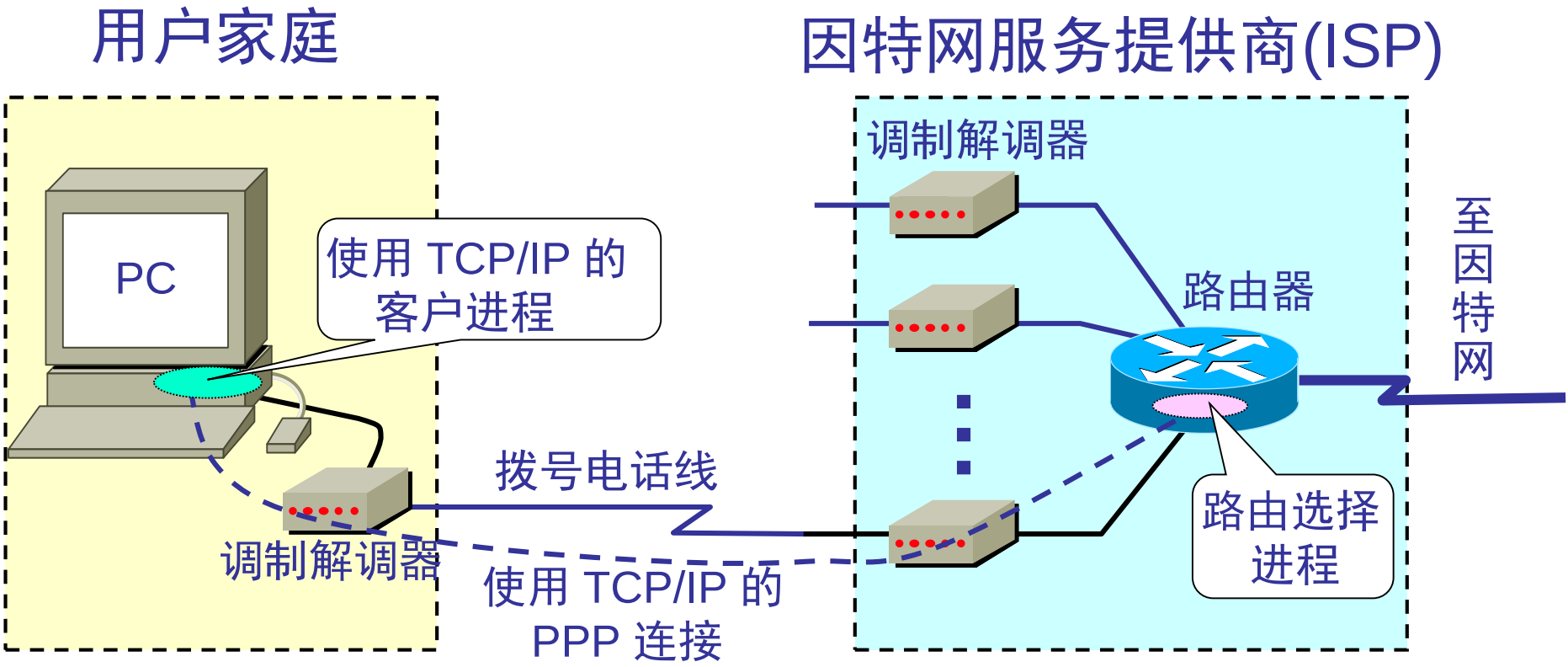
HDLC 工作过程举例



图(d)：丢帧情况，通过发送REJ帧来恢复

图(e)：超时情况，通过交换RR来恢复

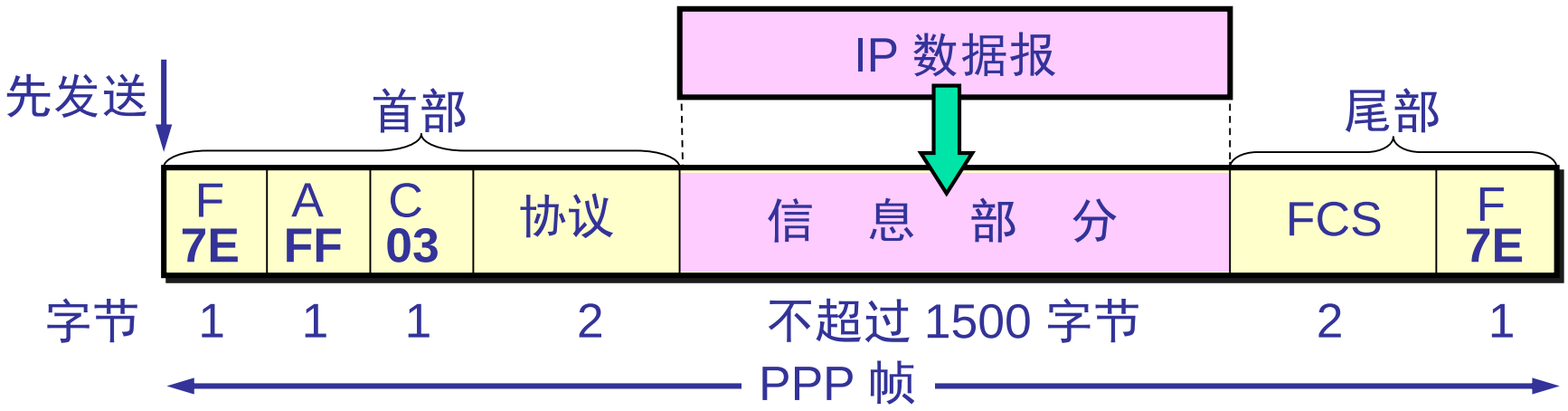
互联网的点对点协议 PPP



PPP 协议

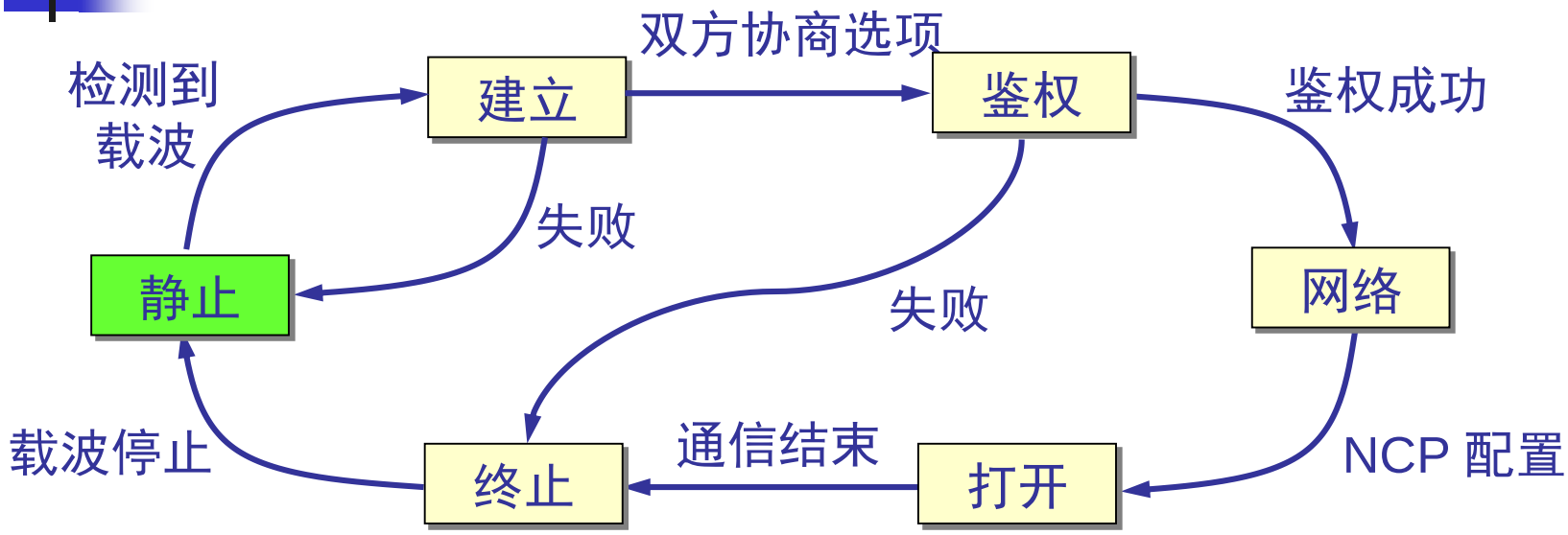
- PPP协议有三个组成部分
  - 将 IP 数据报封装到串行链路的方法。
  - 链路控制协议 LCP (Link Control Protocol)。
  - 网络控制协议 NCP (Network Control Protocol)。
- PPP 的帧格式和 HDLC 的相似。
- PPP 协议不采用链路层可靠传输协议，（序号和确认）
- 传输的帧为字节的整数倍，采用字节填充的透明传输

PPP 协议的帧格式



- PPP的协议字段为 2 个字节，其值为
  - 0x0021，信息字段是IP 数据报。
  - 0xC021，信息字段是 PPP 链路控制数据。
  - 0x8021，表示网络控制数据。

PPP 协议的状态转移图



当用户拨号接入ISP时，路由器做出确认，并建立连接。
PC机向路由器发送LCP分组，选择PPP参数并配置网络；
双方认证，检查对方的身份；由NCP配置网络参数，包括分配临时的IP地址。
通信完毕，NCP释放网络连接，收回IP地址。
LCP释放数据链路连接，最后释放物理连接。

小结

- 数据链路层，基于比特流组成帧流
- 组帧方法：帧格式及其功能（帧头、类型、数据、校验）
- 差错编码：检错及纠错
- 差错控制、流量控制
- 典型的链路层控制协议及信道利用率
  - 停等
  - 回退N
  - 选择重传

练习题

- 名词解释：捎带确认，滑动窗口协议
- 试证明连续ARQ协议，发送窗口的极大值为2^n-1，其中，n为帧编号所用比特数。
- 采用滑动窗口协议在3000Km的T1链路上传输64B的帧，若信号传播速度为6μs/km，问序号应为多少位。（参考答案：7位）
- 利用地球同步轨道的卫星，在1Mbps的信道上发送1000b的帧。信号传播时间为270ms，采用捎带确认，假设帧头很短，序号占3位，信道上传输无错帧，分别采用（1）停等协议（2）滑动窗口协议，求信道利用率。（参考答案：0.18%，1.29％）