第三章 数据据链路层基础

数据链路层的设计问题

数据链路层提供的服务：无确认 无连接、有确认 无连接、有确认 有连接

功能：

成帧 （Framing）将比特流划分成“帧”的主要目的是为了检测和纠正物理层在比特传输中可能出现的错误，数据链路层功能需借助“帧”的各个域来实现。接收方必须能从物理层接收的比特流中明确区分出一帧的开始和结束，这个问题被称为帧同步或帧定界，方法有字节计数法、带字节填充的定界符法、带比特填充的定界符法、物理层编码违例

差错控制 （Error Control）处理传输中出现的差错，包括：

- 错误（incorrect）：数据发生错误——差错校验
- 丢失（lost）：接收方未收到——计时器
- 乱序（out of order）：先发后到，后发先到——序列号
- 重复（repeatedly delivery）：一次发送，多次接收——序列号

流量控制 （Flow Control）确保发送方的发送速率不大于接收方的处理速率，避免接收缓冲区溢出

差错检测和纠正

> 最初的方法：通常采用增加冗余信息（或称校验信息）的策略。每个比特传三份，如果每比特的三份中有一位出错，可以纠正。但是这样冗余信息太多，效率低

概念

码字 (code word)：一个包含m个数据位和r个校验位的n位单元 描述为 (n, m) 码，n=m+r

码率 (code rate)：码字中不含冗余部分所占的比例，可以用m/n表示

海明距离 (Hamming distance)：两个码字之间不同对应比特的数目

如果两个码字的海明距离为d，则可以检测d-1个比特错，纠正d/2-1个比特错

典型检错码

奇偶校验 (Parity Check)：1位奇偶校验是最简单、最基础的检错码

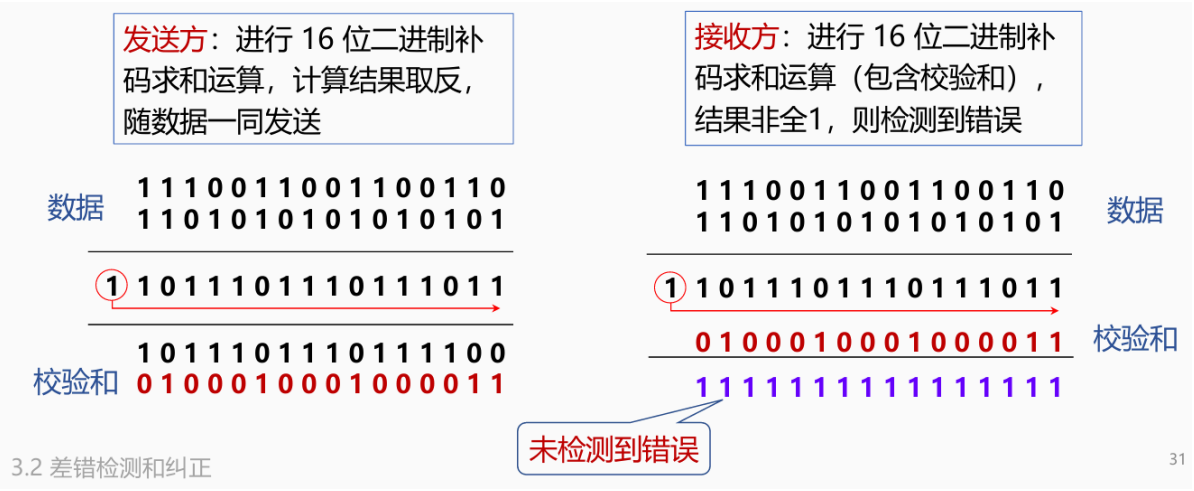
➢ 1位奇偶校验：增加1位校验位，可以检查奇数位错误

- 偶校验：保证1的个数为偶数个，例如：

- 奇校验：保证1的个数为奇数个，例如：

1	0	1	1	0	1	0	0
0	0	0	1	1	1	0	1
1	1	1	1	1	1	1	0
0	0	1	0	0	0	1	0
0	0	1	1	1	1	0	0
0	1	0	1	1	0	0	

校验和(Checksum)：主要用于TCP/IP体系中的网络层和传输层

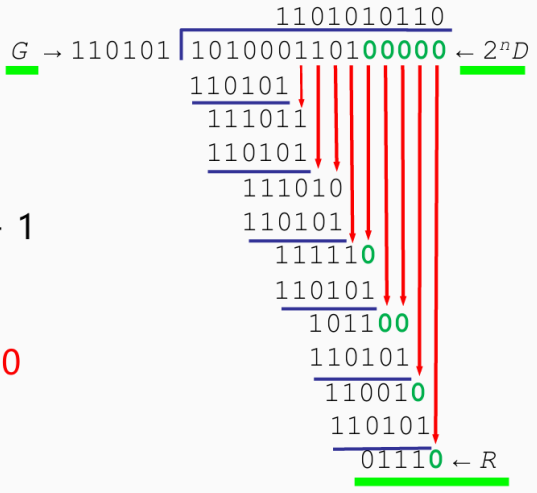


循环冗余校验 (Cyclic Redundancy Check，CRC)：数据链路层广泛使用的校验方法

设原始数据D为k位二进制位模式，如果要产生n位CRC校验码，事先选定一个n+1位二进制位模式G (称为生成多项式，收发双方提前商定)，G的最高位为1，将原始数据D乘以2n （相当于在D后面添加 n 个0），产生k+n位二进制位模式，用G对该位模式做模2除，得到余数R（n位，不足n位前面用0补齐）即为CRC校验码

➢ CRC校验码计算示例

- D = 1010001101
- n = 5
- G = 110101 或 G = $x^5 + x^4 + x^2 + 1$
- R = 01110
- 实际传输数据：1010001101**01110**



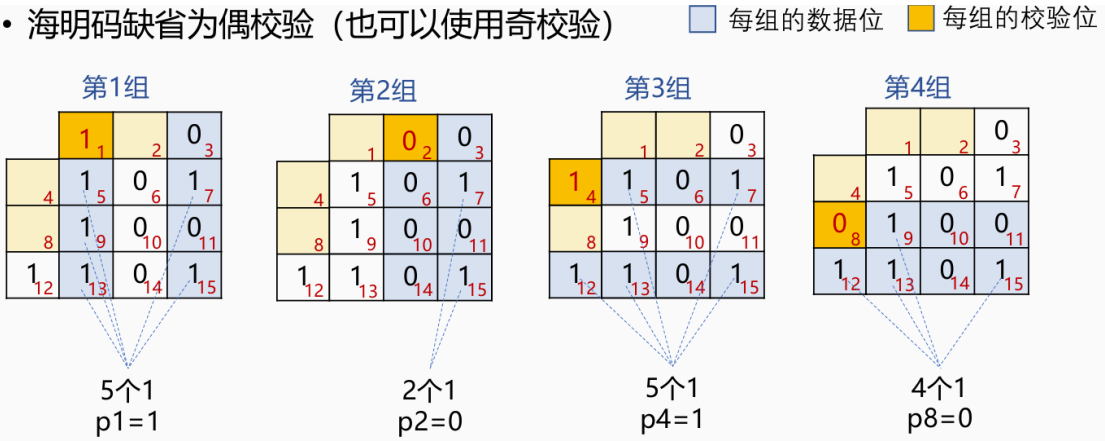
典型纠错码

➢ 设计纠错码

- 要求：m个信息位，r个校验位，纠正单比特错
- 对2^m个有效信息中任何一个，有n个与其距离为1的无效码字：
  - 可以考虑将该信息对应的合法码字的n位逐个取反，得到n个距离为1的非法码字，需要n+1个位模式来标识
- 因此有：$(n + 1)\ 2^m \le 2^n$
- 利用 $n = m + r$，得到 $(m + r + 1) \le 2^r$
- 在给定m的情况下，利用该式可以得出纠正单比特错误校验位数的下界

海明码 以奇偶校验为基础，如何找到出错位置，提供1位纠错能力

- 海明码缺省为偶校验（也可以使用奇校验）



- 总体的定位错误列表

出错位	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
组1	×	√	×	√	×	√	×	√	×	√	×	√	×	√	×
组2	√	×	×	√	√	×	×	√	√	×	×	√	√	×	×
组3	√	√	√	×	×	×	×	√	√	√	√	×	×	×	×
组4	√	√	√	√	√	√	√	×	×	×	×	×	×	×	×

➢ 海明码纠正的实现过程

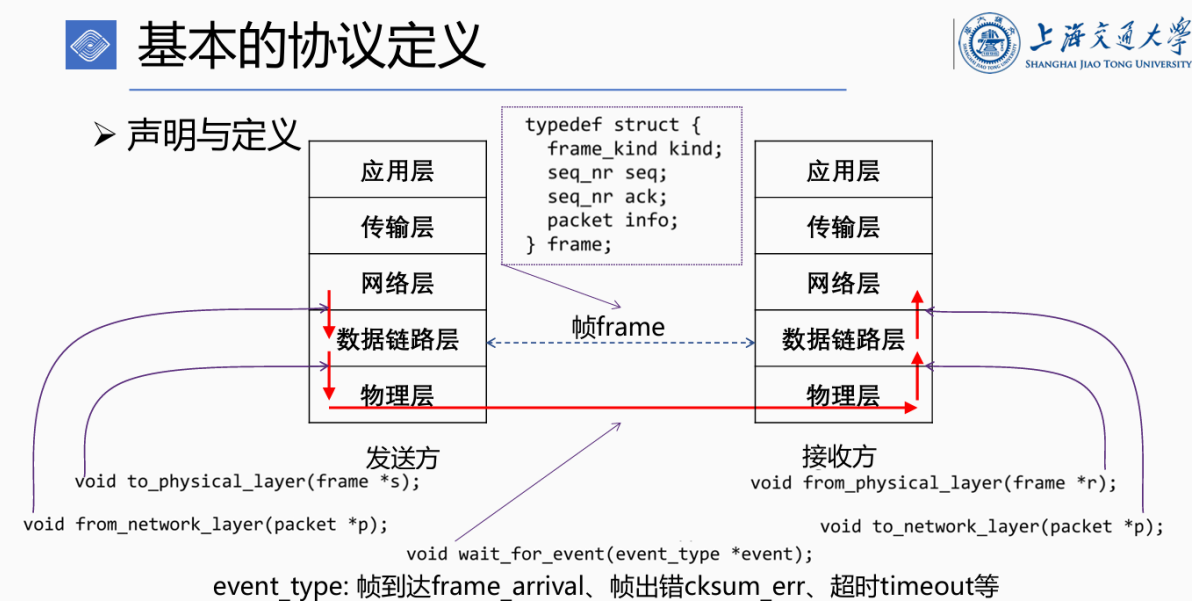
- 每个码字到来前，接收方**计数器**清零
- 接收方检查每个校验位k (k = 1, 2, 4 ...)的奇偶值是否正确（每组运算）
- 若 P_k 奇偶值不对，计数器加 k
- 所有校验位检查完后，若计数器值为0，则码字有效；若计数器值为 j，则第 j 位出错。例：若校验位p1、p2、p8出错，则第11位变反

使用海明码纠正突发错误：可采用k个码字（n = m + r）组成 k * n 矩阵，按列发送，接收方恢复成 k * n 矩阵。kr个校验位，km个数据位，可纠正最多为k个的突发性连续比特错

例如一组(5,3)码字1 2 3 4 5 6 7 8 9 a b c d e f若出现连续三位的比特错想要纠错(假设是456)，需要将三个码字按列排序，即1 6 b 2 7 c...这样能将原本连续的错误分散到不同码字里

基本的数据链路层协议

基本定义与假设



- 分层进程独立假设
  - 网络层、数据链路层、物理层为独立的进程
  - 进程间通过传递消息实现通信
- 提供可靠服务假设
  - 提供可靠的、面向连接的服务
  - 数据链路层发送的数据随时可向网络层获得
- 只处理通信错误假设
  - 仅处理通信错误
  - 假设机器不会崩溃，不考虑断电、重启等引起的问题

乌托邦式单工协议

假设：

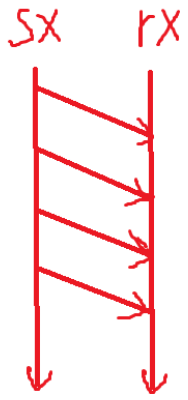
单工（Simplex）协议：数据单向传输

完美信道：帧不会丢失或受损

始终就绪：发送方/接收方的网络层始终处于就绪状态

瞬间完成：发送方/接收方能够生成/处理无穷多的数据

> 乌托邦：完美但不现实的协议：不处理任何流量控制或纠错工作；接近于无确认的无连接服务，必须依赖更高层次解决上述问题



➢发送方

- 在循环中不停发送
- 从网络层获得数据
- 封装成帧
- 交给物理层
- 完成一次发送

```
frame s;
packet buffer;

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

➢ 接收方

- 在循环中持续接收
- 等待帧到达（frame_arrival）
- 从物理层获得帧
- 解封装，将帧中的数据传递给网络层
- 完成一次接收

```
frame r;
event_type event;

while (true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
}
```

无错信道上的停等式协议

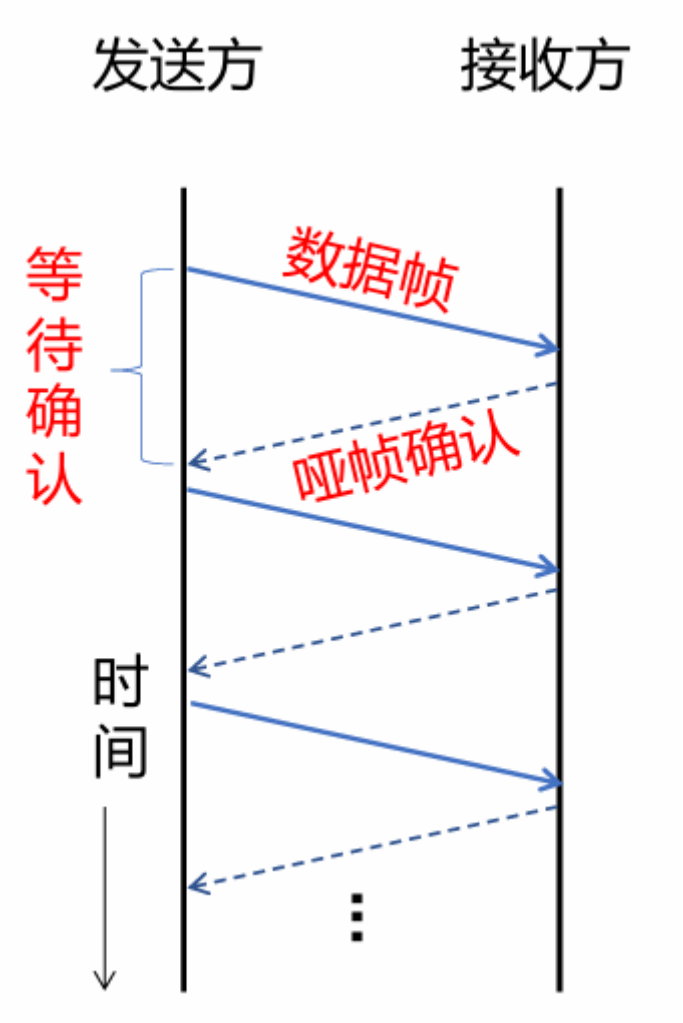
仍然假设：

通信信道不会出错（Error-Free）

数据传输保持单向，但是需要双向传输链路（半双工物理信道）

停-等式协议（stop-and-wait）

- 发送方发送一帧后暂停，等待确认(Acknowldgement)到达后发送下一帧
- 接收方完成接收后，回复确认接收
- 确认帧的内容是不重要的：哑帧（dummy frame）



➢ 发送方

- 完成一帧发送后
- 等待确认到达
- 确认到达后，发送下一帧

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

➢接收方

- 完成一帧接收后
- 交给物理层一个哑帧
- 作为成功接收上一帧的确认

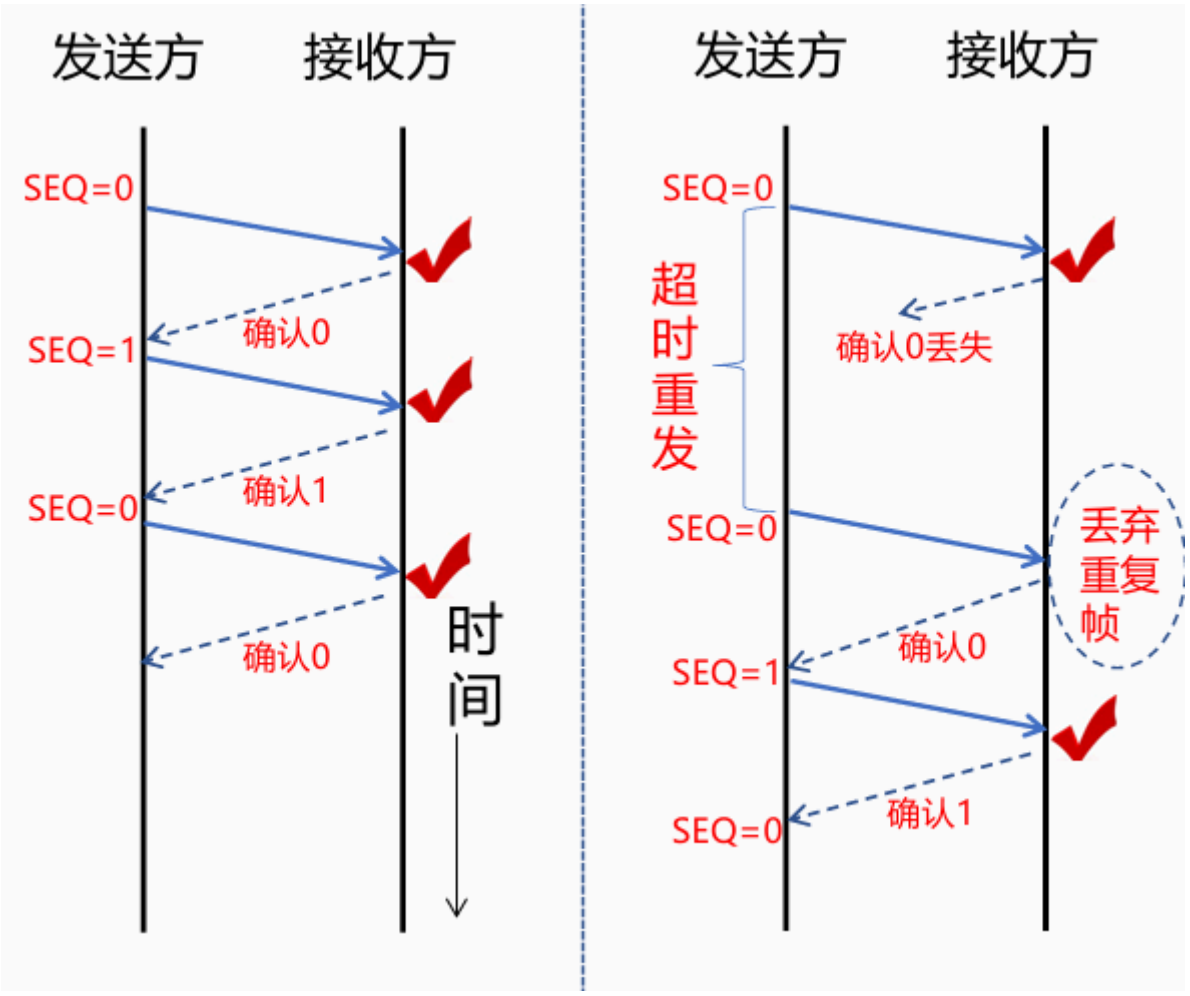
```
while (true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
        to_physical_layer(&s);
}
```

有错信道上的单工停等式协议

假设 通信信道可能会出错，导致：帧在传输过程中可能会被损坏，接收方能够检测出来；帧在传输过程中可能会丢失，永远不可能到达接收方

一个简单的解决方案：发送方增加一个计时器(timer)，如果经过一段时间没有收到确认，发送方将超时，于是再次发送该帧

序号（SEQ：sequence number）：让发送方在发送的帧的头部放一个序号，接收方可以检查它所收到的帧序号，由此判断这是个新帧还是应该被丢弃的重复帧。 1bit就足够



➢发送方

- 初始化帧序号0，发送帧
- 等待：正确的确认/错误的确认/超时
- 正确确认：发送下一帧
- 超时/错误确认：重发

```
next_frame_to_send = 0;
from_network_layer(&buffer);
while (true) {
        s.info = buffer;
        to_physical_layer(&s);
        start_timer(s.seq);
        wait_for_event(&event);
        if (event == frame_arrival){
                from_physical_layer(&s);
                if (s.ack == next_frame_to_send){
                        from_network_layer(&buffer);
                        inc(next_frame_to_send);
                }
        }
}
```

➢接收方

- 初始化期待0号帧
- 等待帧达到
- 正确帧：交给网络层，并发送该帧确认
- 错误帧：发送上一个成功接收帧的确认

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

➢ 效率的评估

- F = frame size (bits)
- R = channel capacity (Bandwidth in bits/second)
- I = propagation delay + processor service time (second)
- 每帧发送时间 (Time to transmit a single frame) = F/R
- 总延迟 (Total Delay) = D =2I
- 停止等待协议的发送工作时间是F/R，空闲时间是D
- 当 F<R·D 时：信道利用率 (line utilization)=F/(F+R·D) < 50%

滑动窗口协议

停等协议的性能问题

停止-等待机制降低了信道利用率

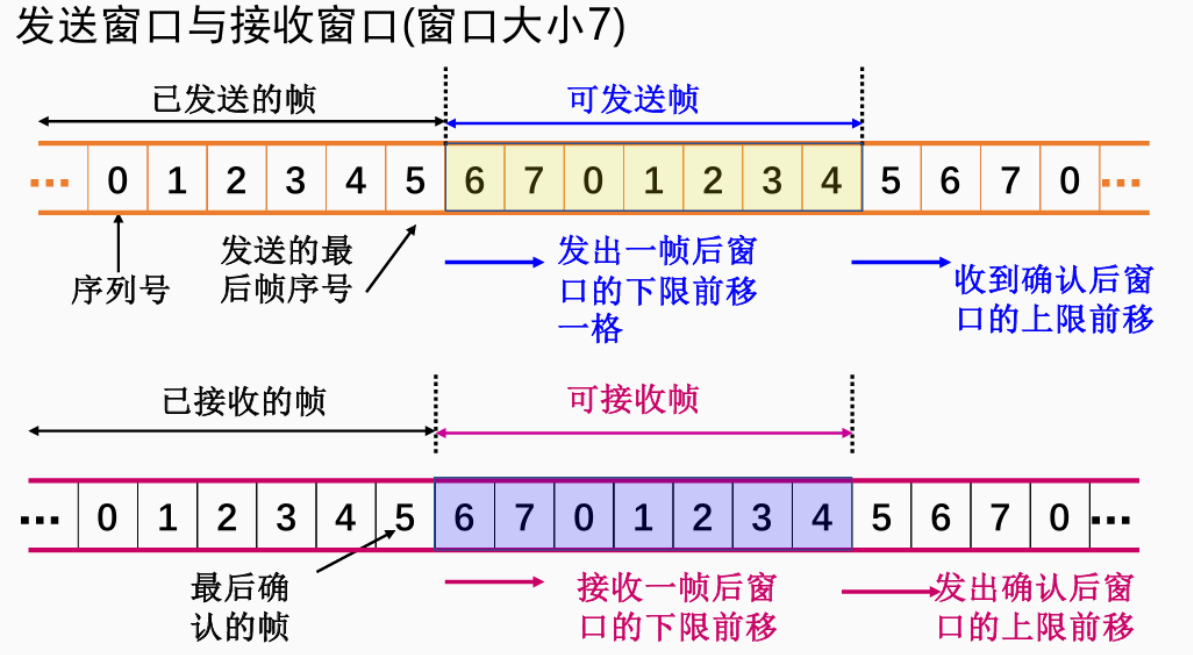
- 设数据速率记为R，帧长度记为F，往返延迟记为D，则采用停-等协议的线路效率为：F/(F+R·D)
- 假如将链路看成是一根管道，数据是管道中流动的水，那么在传输延迟较长的信道上，停-等协议无法使数据充满管道，因而信道利用率很低

解决办法 流水线协议或管道协议：允许发送方在没收到确认前连续发送多个帧

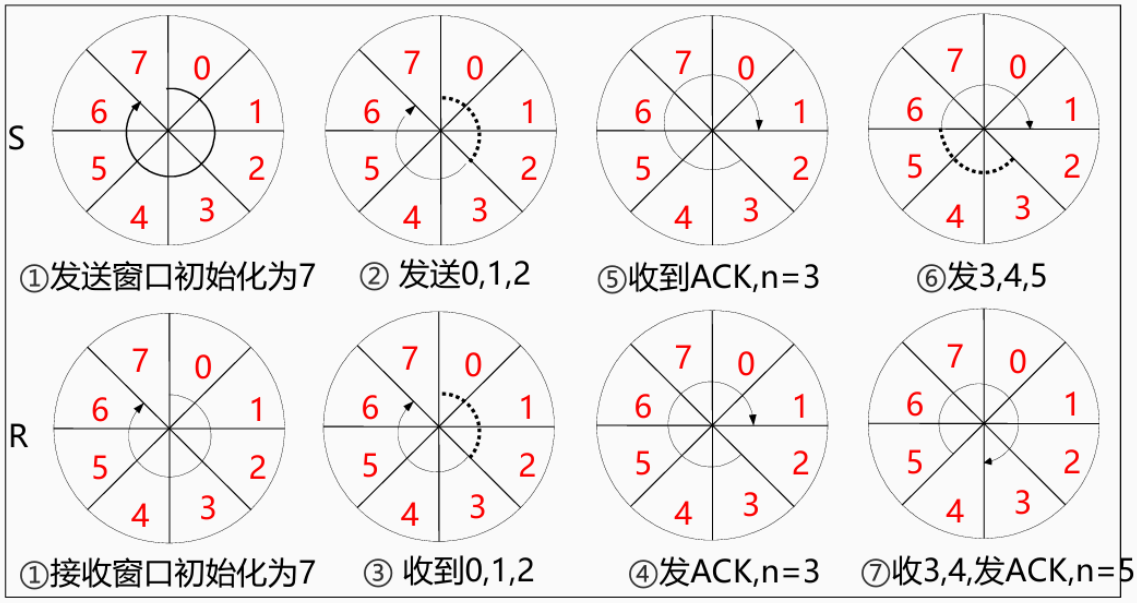
滑动窗口协议

协议基本思想

- 窗口机制：发送方和接收方都具有一定容量的缓冲区（即窗口），发送端在收到确认之前可以发送多个帧
- 流量控制：接收窗口驱动发送窗口的转动
- 累计确认：不必对收到的分组逐个发送确认，而是对按序到达的最后一个分组发送确认
- 对可以连续发出的最多帧数（已发出但未确认的帧）作限制



➢ 发送窗口与接收窗口(窗口大小7)



实现：

发送方（$W_T=W_R=1$，序号空间0和1）

1. 初始化：ack_expected = frame_expected = next_frame_to_send = 0
2. 从网络层接收分组，放入相应的缓冲区，构造帧，物理层发送，开启计时。
3. 等待确认帧到达，从物理层接收一个帧，判断确认号是否正确，正确则停止计时器，并从网络层接收新分组。
4. 发送新的帧，跳转至3

```
while(1){
  wait_for_event(&event);
  if(event==frame_arrival) {
    from_physical_layer(&r);
    if(r.ack==next_frame_to_send) {
      from_network_layer(&buffer);
      inc(next_frame_to_send);
      }
    }
  s.info=buffer;
  s.seq=next_frame_to_send;
  s.ack=1-frame_expected;
  to_physical_layer(&s);
  start_timer(s.seq);
}
```

接收方（$W_T=W_R=1$，序号空间0和1）

1. 初始化：ack_expected = frame_expected = next_frame_to_send = 0
2. 等待帧到达，从物理层接收一个帧，校验和计算，并判断收到的帧序号是否正确，正确则交给网络层处理，期待帧号增加。
3. 返回确认帧，跳转至2。

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

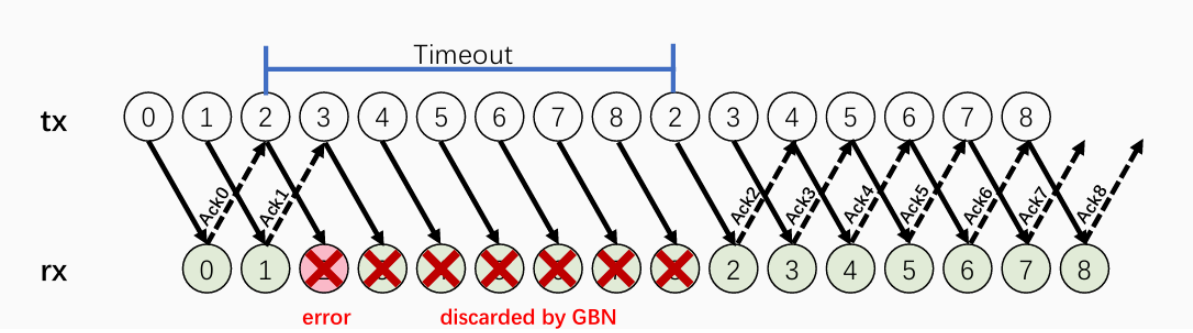

考虑若中间的帧未接收到怎么办，两种方法：回退N 和 选择重传

回退N协议

基本原理：当发送方发送了N个帧后，若发现该N帧的前一个帧在计时器超时后仍未返回其确认信息，则该帧被判为出错或丢失，此时发送方就重新发送出错帧及其后的N帧。

优点：连续发送提高了信道利用率

缺点：按序接收，出错后即便有正确帧到达也丢弃重传



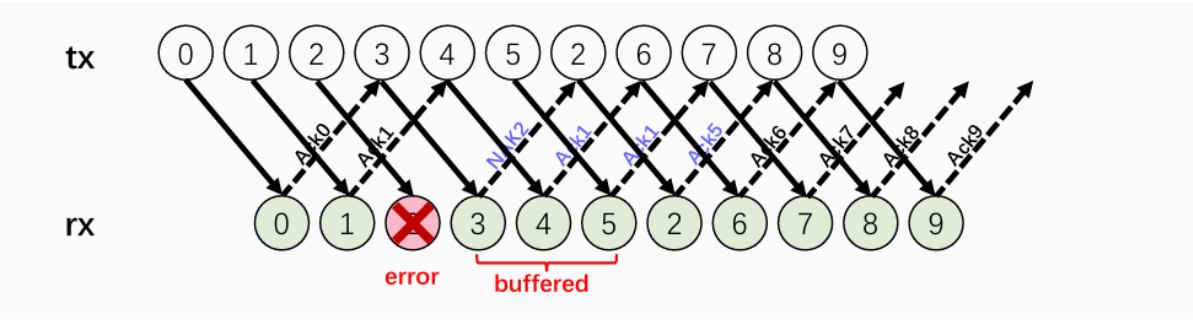
如上图例子，接收方一旦没收到就停止发ack；发送方每一次发送都有一个计时器，一旦超时就重传

发送方窗口尺寸：$1< W_T ≤2^n-1$ ，n是序列号的位数

选择重传协议

若发送方发出连续的若干帧后，收到对其中某一帧的否认帧，或某一帧的定时器超时， 则只重传该出错帧或计时器超时的数据帧

优点：避免重传已正确传送的帧 缺点：在接收端需要占用一定容量的缓存



如上图，2没收到时，接收方不法任何东西；3收到时，缓存下来，发送NAK2；4和5收到时缓存下来，发送ack1，代表发送方可以继续传后面的帧；重传的2收到时，发ack5，代表最新收到的是5