第三章 数据链路层基础

授课教师：张圣林

南开大学

本章目标

- 了解数据链路层在网络体系结构中的位置及基本功能和服务
- 掌握差错检测和纠正的基本原理和典型的编码方法（核心内容）
- 掌握无错信道和有错信道上停等协议的设计和实现方法（核心内容）
- 理解停等协议的性能问题及滑动窗口协议的基本思想
- 掌握回退N和选择重传两种典型滑动窗口协议的工作机制（核心内容）
- 了解点到点链路层协议PPP与PPPoE

本章内容

3.1 数据链路层的设计问题

3.2 差错检测和纠正

3.3 基本的数据链路层协议

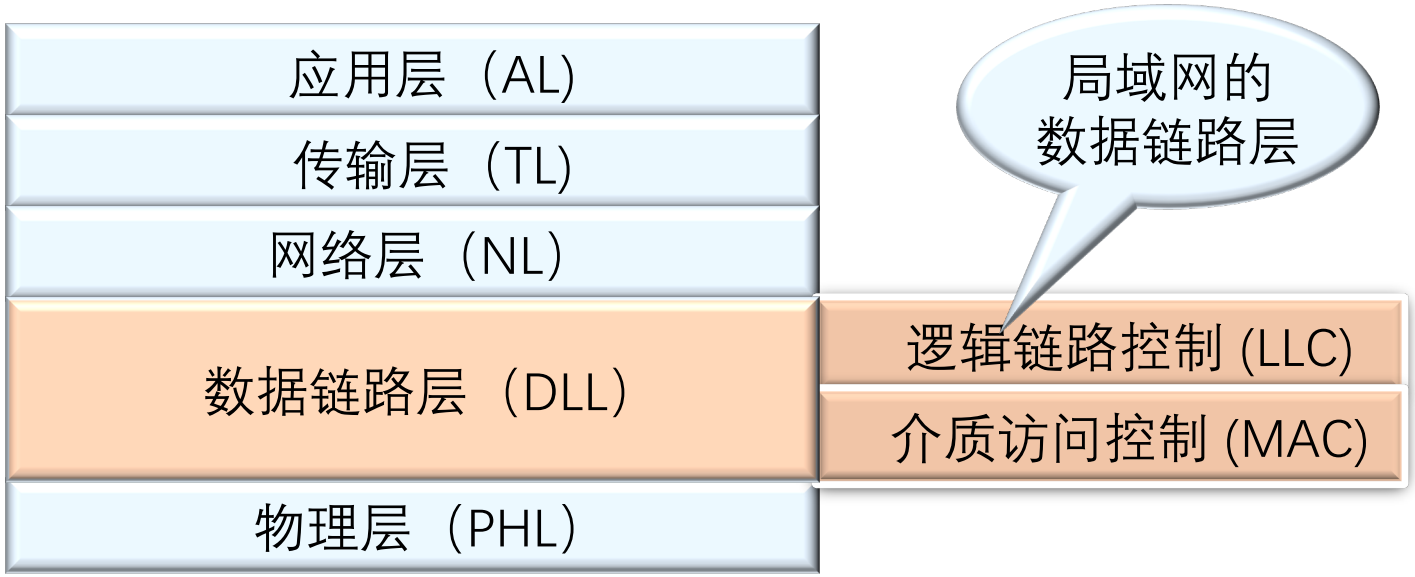
3.4 滑动窗口协议

3.5 数据链路协议实例

1. 数据链路层在协议栈中的位置
2. 数据链路层的功能
3. 数据链路层提供的服务
4. 成帧
5. 差错控制
6. 流量控制

数据链路层在协议栈中的位置

- 向下：利用物理层提供的位流服务
- 向上：向网络层提供明确的 (well-defined) 服务接口



参考协议栈

数据链路层的功能

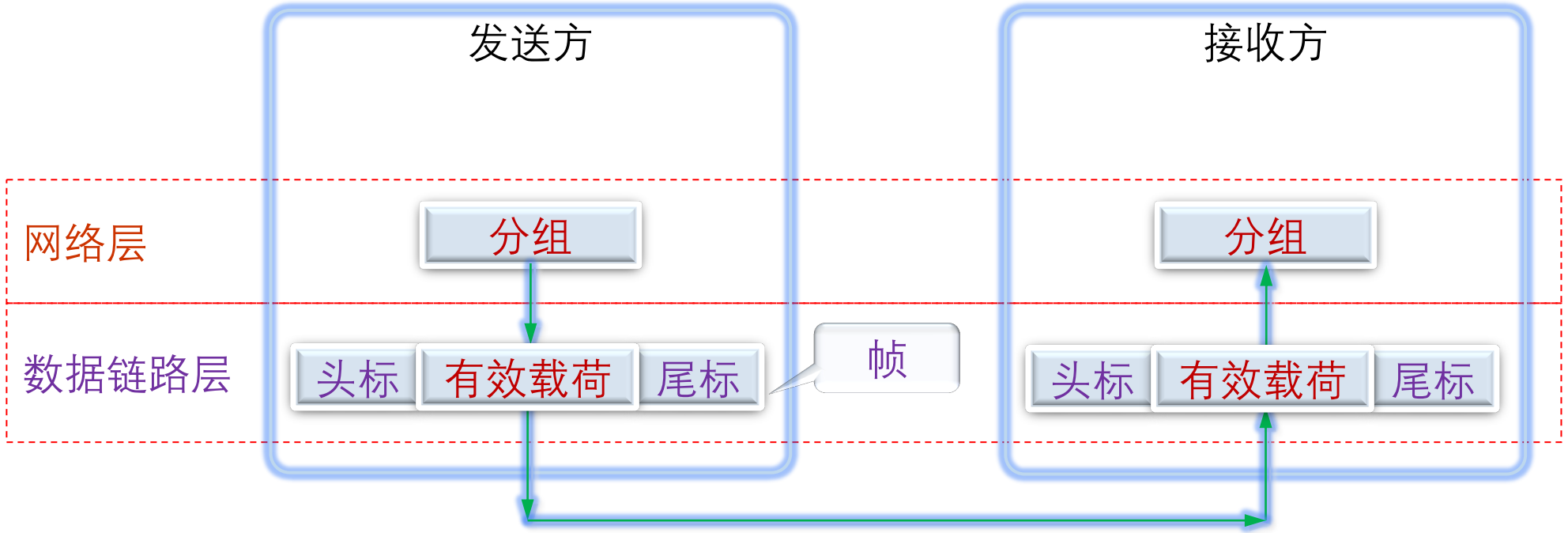
- 成帧 （Framing）
  - 将比特流划分成 “帧” 的主要目的是为了检测和纠正物理层在比特传输中可能出现的错误，数据链路层功能需借助 “帧” 的各个域来实现
- 差错控制 （Error Control）
  - 处理传输中出现的差错，如位错误、丢失等
- 流量控制 （Flow Control）
  - 确保发送方的发送速率，不大于接收方的处理速率
    - 避免接收缓冲区溢出

数据链路层提供的服务

- 无确认 无连接 服务（Unacknowledged connectionless）
  - 接收方不对收到的帧进行确认
  - 适用场景：误码率低的可靠信道；实时通信；
  - 网络实例：以太网
- 有确认 无连接 服务（Acknowledged connectionless）
  - 每一帧都得到单独的确认
  - 适用场景：不可靠的信道（无线信道）
  - 网络实例：802.11
- 有确认 有连接 服务（Acknowledged connection-oriented）
  - 适用场景：长延迟的不可靠信道

成帧（Framing）

- 分组 (packet) 与 帧(frame)的关系



差错控制

- 链路层存在的一个问题：信道的噪声导致数据传输问题
  - 差错（incorrect）：数据发生错误
  - 丢失（lost）：接收方未收到
  - 乱序（out of order）：先发后到，后发先到
  - 重复（repeatedly delivery）：一次发送，多次接收
- 解决方案：差错检测与纠正、确认重传
  - 确认：接收方校验数据（差错校验），并给发送方应答，防止差错
  - 定时器：发送方启动定时器，防止丢失
  - 顺序号：接收方检查序号，防止乱序递交、重复递交

流量控制

- 链路层存在的另一个问题：接收方的处理速率
  - 接收方的接收缓冲区溢出
- 解决方案
  - 基于反馈 (feedback-based) 的流量控制
    - 接收方反馈，发送方调整发送速率
  - 基于速率 (rate-based) 的流量控制
    - 发送方根据内建机制，自行限速

本章内容

3.1 数据链路层的设计问题

3.2 差错检测和纠正

3.3 基本的数据链路层协议

3.4 滑动窗口协议

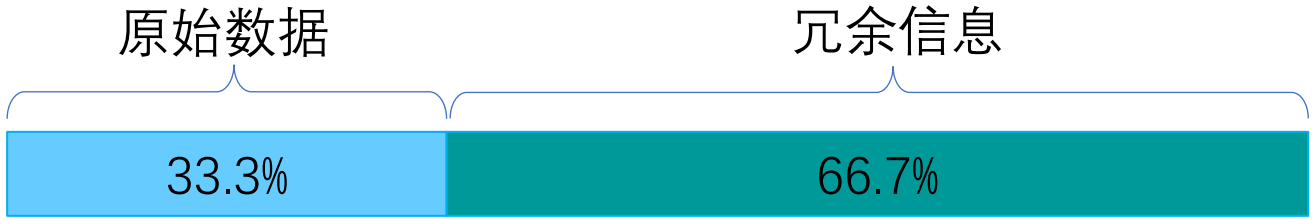
3.5 数据链路协议实例

1. 差错检测与纠正概述
2. 典型的纠错码
3. 典型的检错码

差错检测和纠正概述

- 如何解决信道传输差错问题
  - 通常采用增加冗余信息（或称校验信息）的策略
  - 简单示例：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

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



携带2/3的冗余信息！

冗余信息量大！

差错检测和纠正概述

- 目标：保证一定差错检测和纠错能力的前提下，如何减少冗余信息量？
- 考虑的问题
  - 传输需求
  - 冗余信息的计算方法、携带的冗余信息量
  - 计算的复杂度等
- 两种主要策略
  - 检错码（error-detecting code）
  - 纠错码（error-correcting code）

差错检测和纠正概述

- 检错码（error-detecting code）
  - 在被发送的数据块中，包含一些冗余信息，但这些信息只能使接收方推断是否发生错误，但不能推断哪位发生错误，接收方可以请求发送方重传数据
  - 主要用在高可靠、误码率较低的信道上，例如光纤链路
  - 偶尔发生的差错，可以通过重传解决差错问题

差错检测和纠正概述

- 纠错码（error-correcting code)
  - 发送方在每个数据块中加入足够的冗余信息，使得接收方能够判断接收到的数据是否有错，并能纠正错误（定位出错的位置）
  - 主要用于错误发生比较频繁的信道上，如无线链路
  - 也经常用于物理层，以及更高层（例如，实时流媒体应用和内容分发）
  - 使用纠错码的技术通常称为前向纠错（FEC，Forward Error Correction)

差错检测和纠正概述

- 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
  - 例：0000000000 与0000011111的海明距离为5
  - 如果两个码字的海明距离为d，则需要d个单比特错就可以把一个码字转换成另一个码字
  - 为了检查出d个错（比特错），可以使用海明距离为 **d+1** 的编码
  - 为了纠正d个错，可以使用海明距离为 **2d+1** 的编码

差错检测和纠正概述

- 例如：
  - 一个只有4个有效码字的编码方案：0000000000, 0000011111, 1111100000, 1111111111
  - 海明距离为5，可以检测4位错，纠正2位错
  - 如果已知只有1位或2位错误，接收方接收0000000111
    - 则可知原码字为：0000011111
  - 如果发生至多3位错误，例如0000000000变成0000000111，接收方无法纠正错误，但可以检测出错误

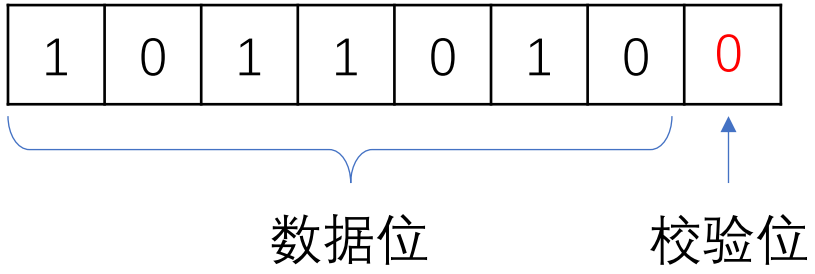
典型检错码

- 常用的检错码包括：
  - 奇偶检验 (Parity Check)：1位奇偶校验是最简单、最基础的检错码
  - 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

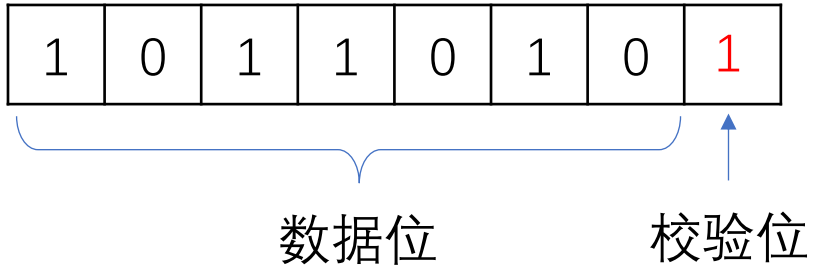
典型检错码—奇偶校验

➢ 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：



典型检错码——校验和

- TCP/IP体系中主要采用的校验方法

发送方：进行 16 位二进制补码求和运算，计算结果取反，随数据一同发送

接收方：进行 16 位二进制补码求和运算（包含校验和），结果非全1，则检测到错误

数据

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

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

校验和

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

数据

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

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

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

校验和

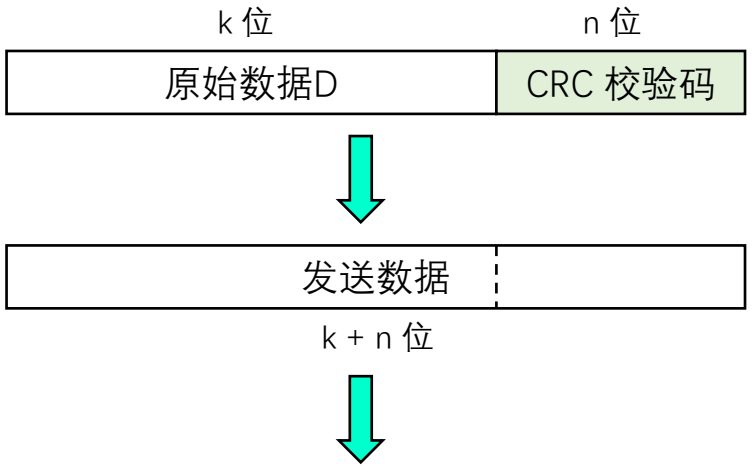
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

未检测到错误

典型检错码—循环冗余校验CRC

➢ CRC校验码计算方法

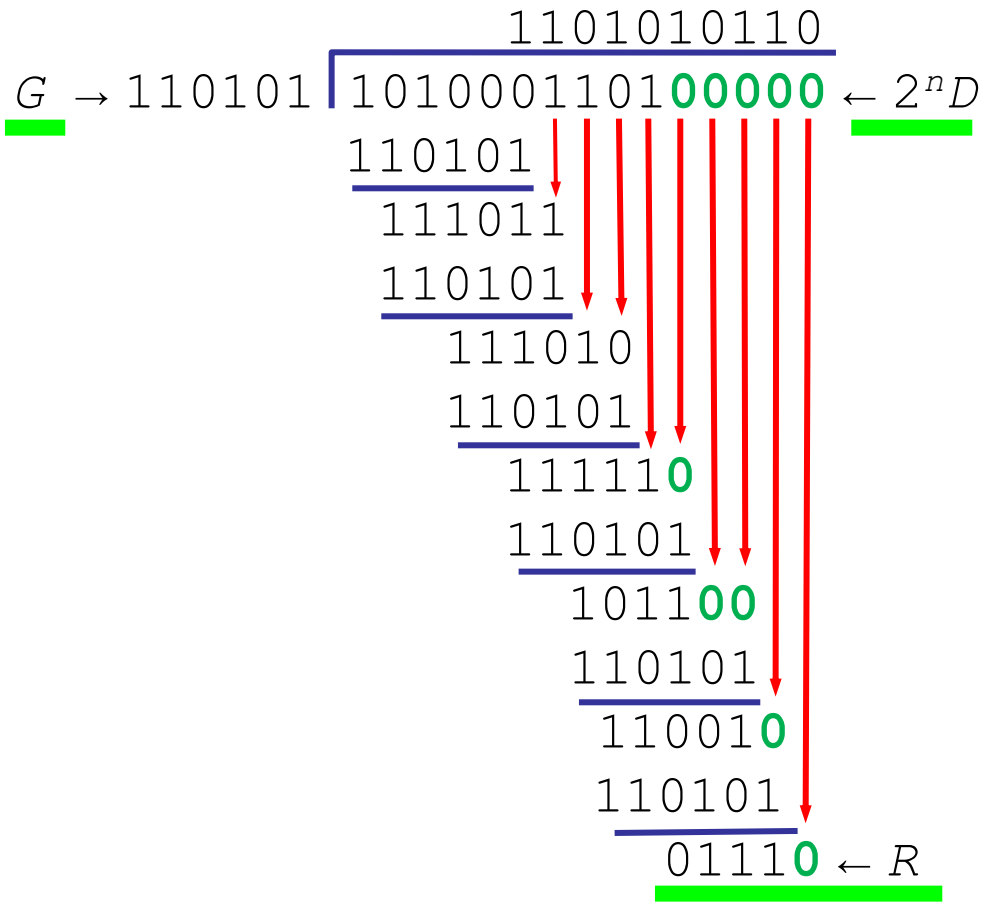
- 设原始数据D为k位二进制位模式
- 如果要产生n位CRC校验码，事先选定一个n+1位二进制位模式G (称为生成多项式，收发双方提前商定)，G的最高位为1
- 将原始数据D乘以2^n （相当于在D后面添加 n 个0），产生k+n位二进制位模式，用G对该位模式做模2除，得到余数R（n位，不足n位前面用0补齐）即为CRC校验码



CRC校验能力：能检测出少于n+1位的突发错误

检错码—循环冗余校验CRC

- 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**



检错码—循环冗余校验CRC

- 四个国际标准生成多项式
  - CRC-12 = $x^{12}+x^{11}+x^3+x^2+x+1$
  - CRC-16 = $x^{16}+x^{15}+x^2+1$
  - CRC-CCITT = $x^{16}+x^{12}+x^5+1$
  - CRC-32 = $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

例如，以太网、无线局域网使用CRC-32生成多项式

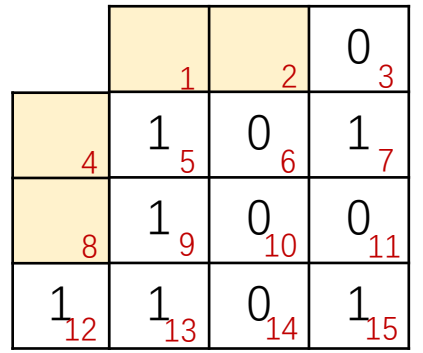
典型纠错码

➢ 设计纠错码

- 要求：m个信息位，r个校验位，纠正单比特错
- 对于任何一个有效信息，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的非法码字和1个合法码字，需要r个校验位来标识
- 因此有：$n + 1 \leq 2^r$,即$m + r + 1 \leq 2^r$
- 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

典型纠错码—海明码

- 目标：以奇偶校验为基础，如何找到出错位置，提供1位纠错能力
- 理解海明码编码过程，以 (15, 11)海明码为例
  - 例如：11比特的数据01011001101
  - 11比特数据按顺序放入数据位
  - 校验位：2的幂次方位（记为p1, p2, p4, p8）
  - 每个校验位对数据位的子集做校验，缩小定位错误的范围
  - 问题：每个校验位如何计算？



校验位　数据位

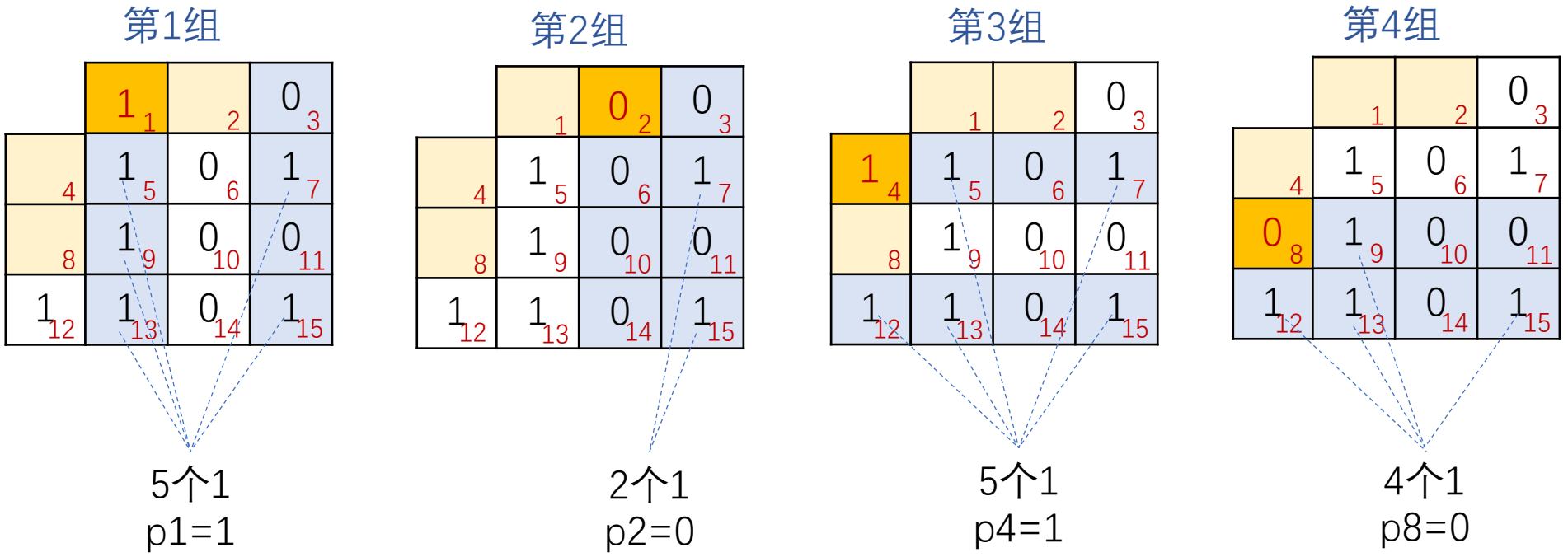
右下角数字为码字中位序号

典型纠错码—海明码

➢ 子集的选择与校验位计算

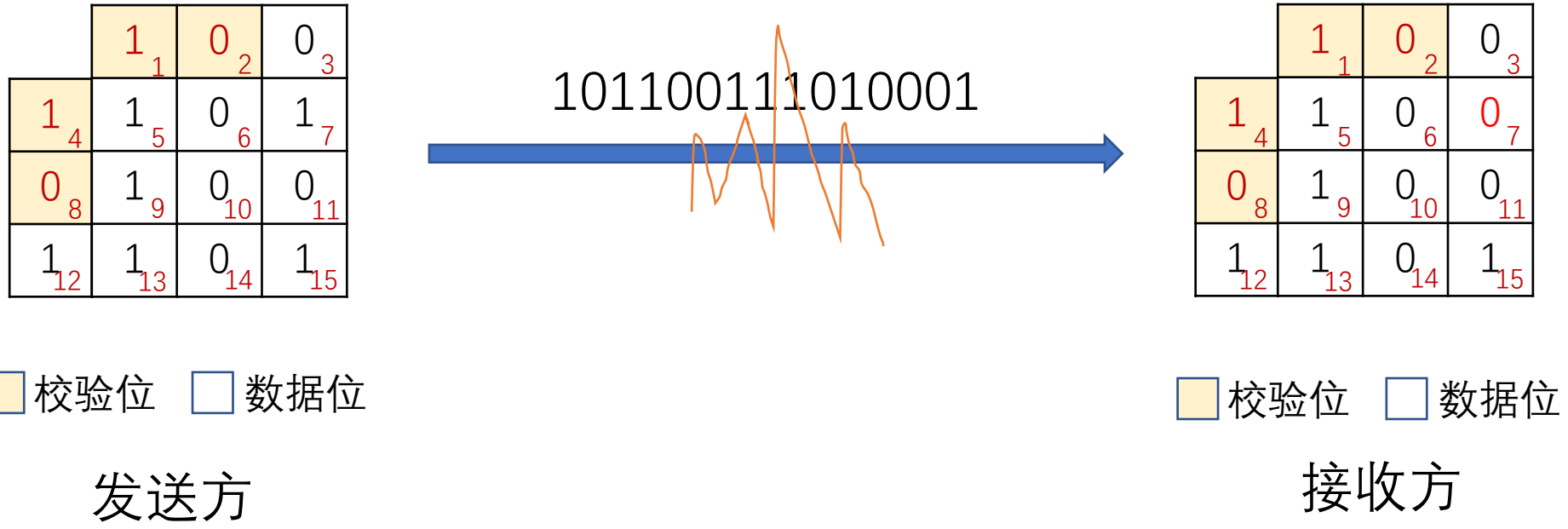
- 海明码缺省为偶校验（也可以使用奇校验）

每组的数据位　每组的校验位



典型纠错码—海明码

- 码字的发送与接收
  - 如果发送过程中第7位出现差错，如何定位错误？

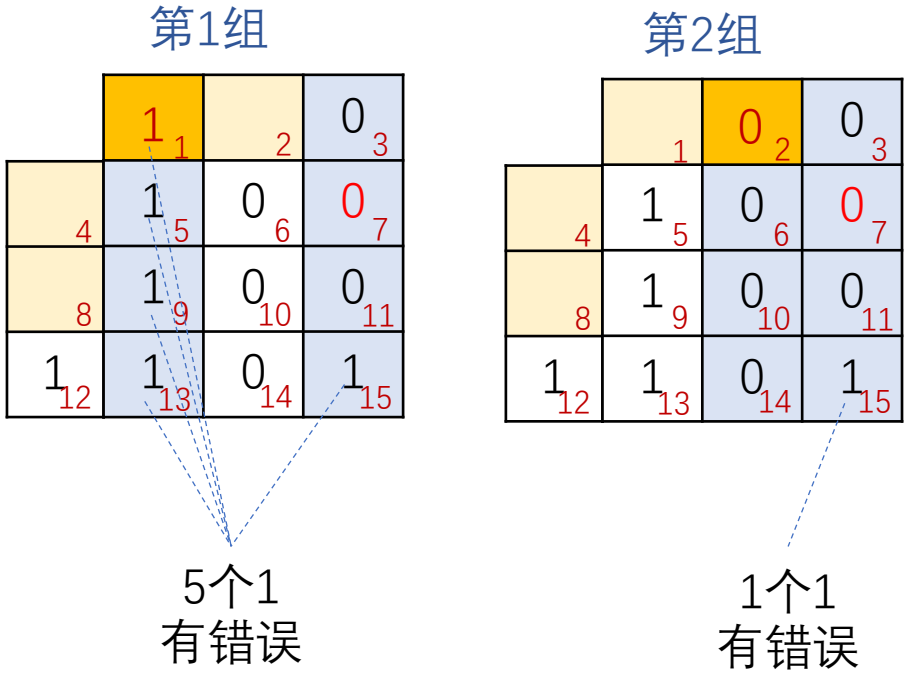


典型纠错码—海明码

➢ 定位错误与纠正

- 组1和组2的校验结果可以定位错误所在的列
- 例如：组1和组2校验结果都指明存在错误，
  可定位错误位于第4列
- 其他导致组1和组2出错的情况判断

出错列	2	3	4
组1	✗	√	✗
组2	√	✗	✗



典型纠错码—海明码

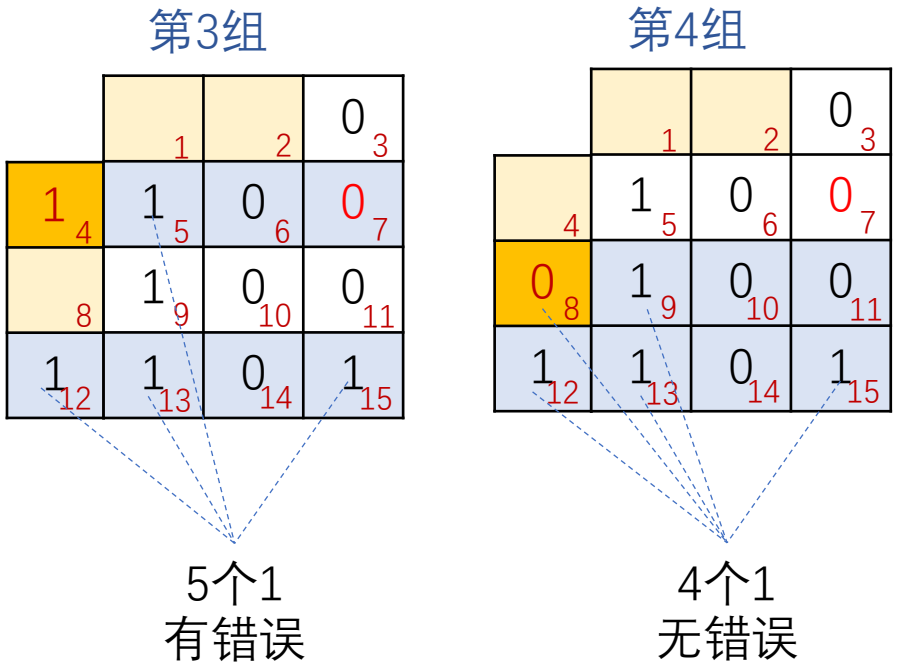
➢ 定位错误与纠正（续）

- 组3和组4的校验结果可以定位错误所在的行
- 例如：组3校验结果指明存在错误，组4校验结果指明无错误，可定位错误位于第2行

  ➡ 结论：错误位于第2行、第4列（位置7）

- 其他导致组3和组4出错情况判断

出错行	2	3	4
组3	✗	√	✗
组4	√	✗	✗



典型纠错码—海明码

➢ 定位错误与纠正（续）

- 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	√	√	√	√	√	√	√	×	×	×	×	×	×	×	×

典型纠错码—海明码

- 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 pk 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

本章内容

3.1 数据链路层的设计问题

3.2 差错检测和纠正

3.3 基本的数据链路层协议

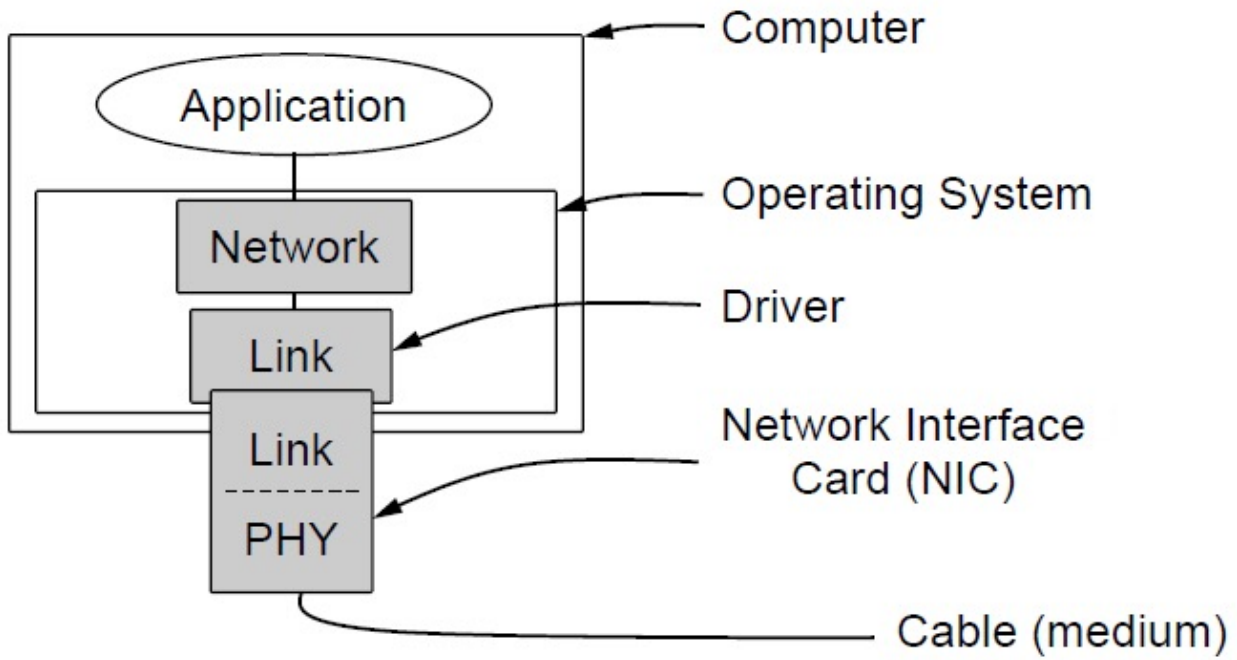
3.4 滑动窗口协议

3.5 数据链路协议实例

1. 定义与假设
2. 乌托邦式单工协议
3. 无错信道单工停止-等待协议
4. 有错信道单工停止-等待协议

物理层、数据链路层和网络层的实现

- 物理层进程和某些数据链路层进程运行在专用硬件上（网络接口卡）
- 数据链路层进程的其他部分和网络层进程作为操作系统的一部分运行在CPU上，数据链路层进程的软件通常以设备驱动的形式存在



关键假设

- 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
  - 网络层可随时获得数据链路层发送的数据
- 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（1）

- 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

乌托邦式单工协议（2）

- 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);
}
```

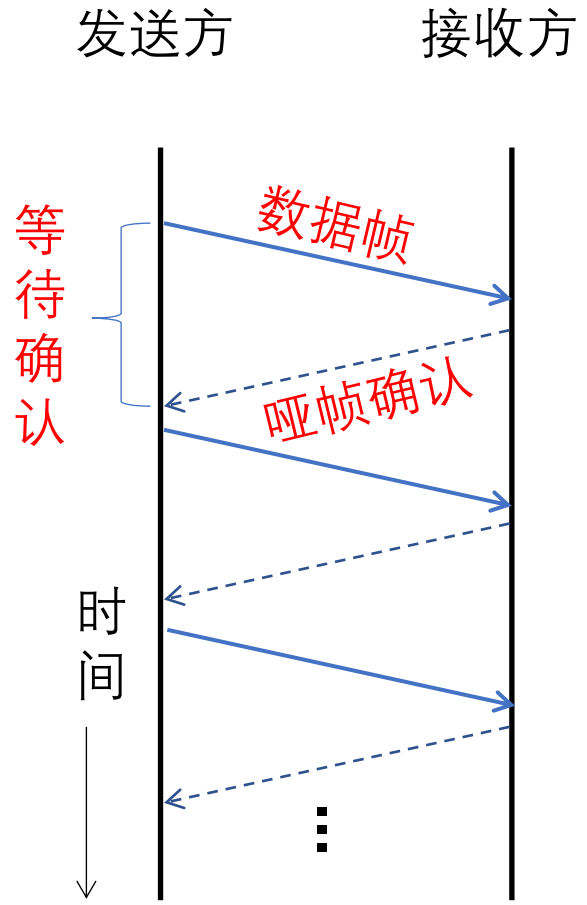
无错信道上的停等式协议（1）

- 不再假设
  - 接收方能够处理以无限高速进来的数据
  - 发送方以高于接收方能处理到达帧的速度发送帧，不会导致接收方被“淹没”(overwhelming)
- 仍然假设
  - 通信信道不会出错（Error-Free）
  - 数据传输保持单向，需要双向传输链路（半双工物理信道）

无错信道上的停等式协议（2）

➢ 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）



无错信道上的停等式协议（3）

- 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

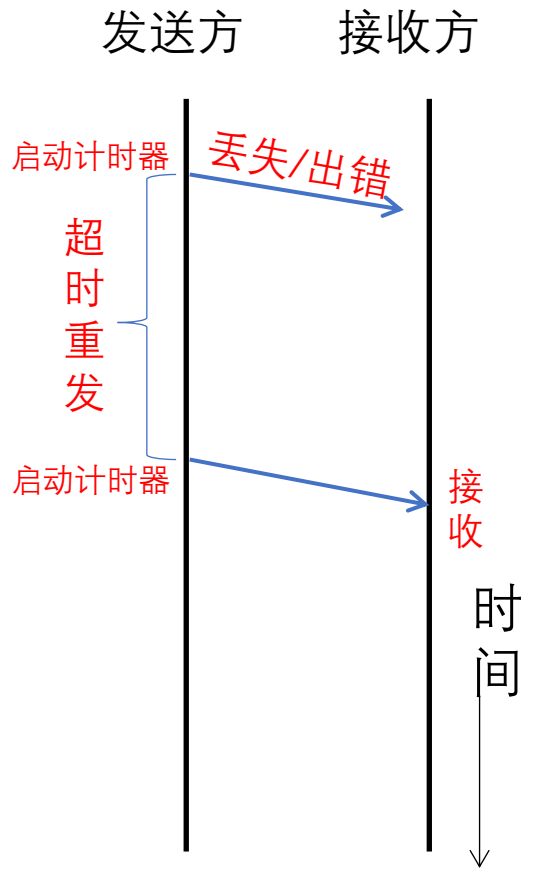
```
while (true) {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
        wait_for_event(&event);
}
```

- 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);
}
```

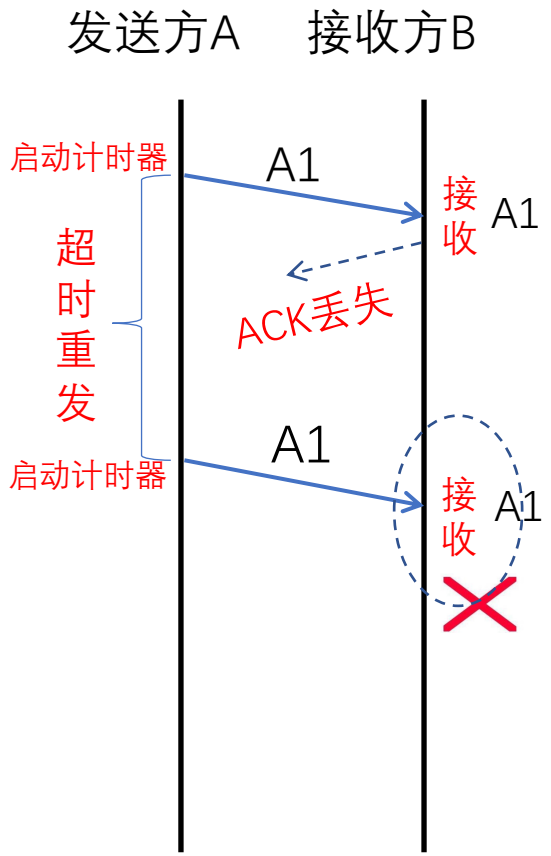
有错信道上的单工停等式协议（1）

- 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



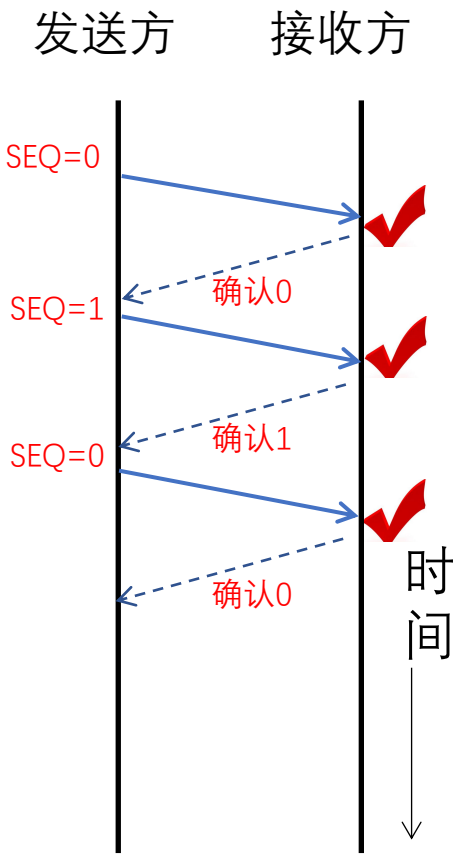
有错信道上的单工停等式协议（2）

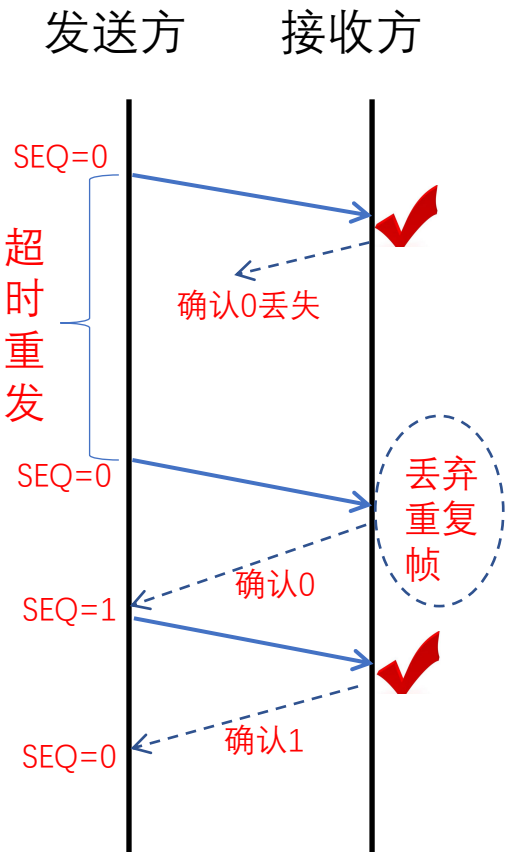
- 考虑一个特别场景
  - A发送帧A1
  - B收到了A1
  - B生成确认ACK
  - ACK在传输中丢失
  - A超时，重发A1
  - B收到A1的另一个副本（并把它交给网络层）
- 其他的场景
  - 另一个导致副本产生的场景是过长的延时 (long delay)



有错信道上的单工停等式协议（3）

- 序号（SEQ：sequence number）
  - 接收方需要确认到达的帧是否第一次发来的新帧
  - 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。
- 序号所需要的最小位数（bits）
  - 序号需要区分当前帧（序号m）和它的直接后续帧（序号m+1）
  - 1 bit序号(0或1)就足以满足要求。





有错信道上的单工停等式协议（4）

- 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);
        }
}
```

本章内容

3.1 数据链路层的设计问题

3.2 差错检测和纠正

3.3 基本的数据链路层协议

3.4 滑动窗口协议

3.5 数据链路协议实例

1. 停等协议的性能问题
2. 滑动窗口协议
3. 回退N协议
4. 选择重传协议

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

➢ 协议基本思想

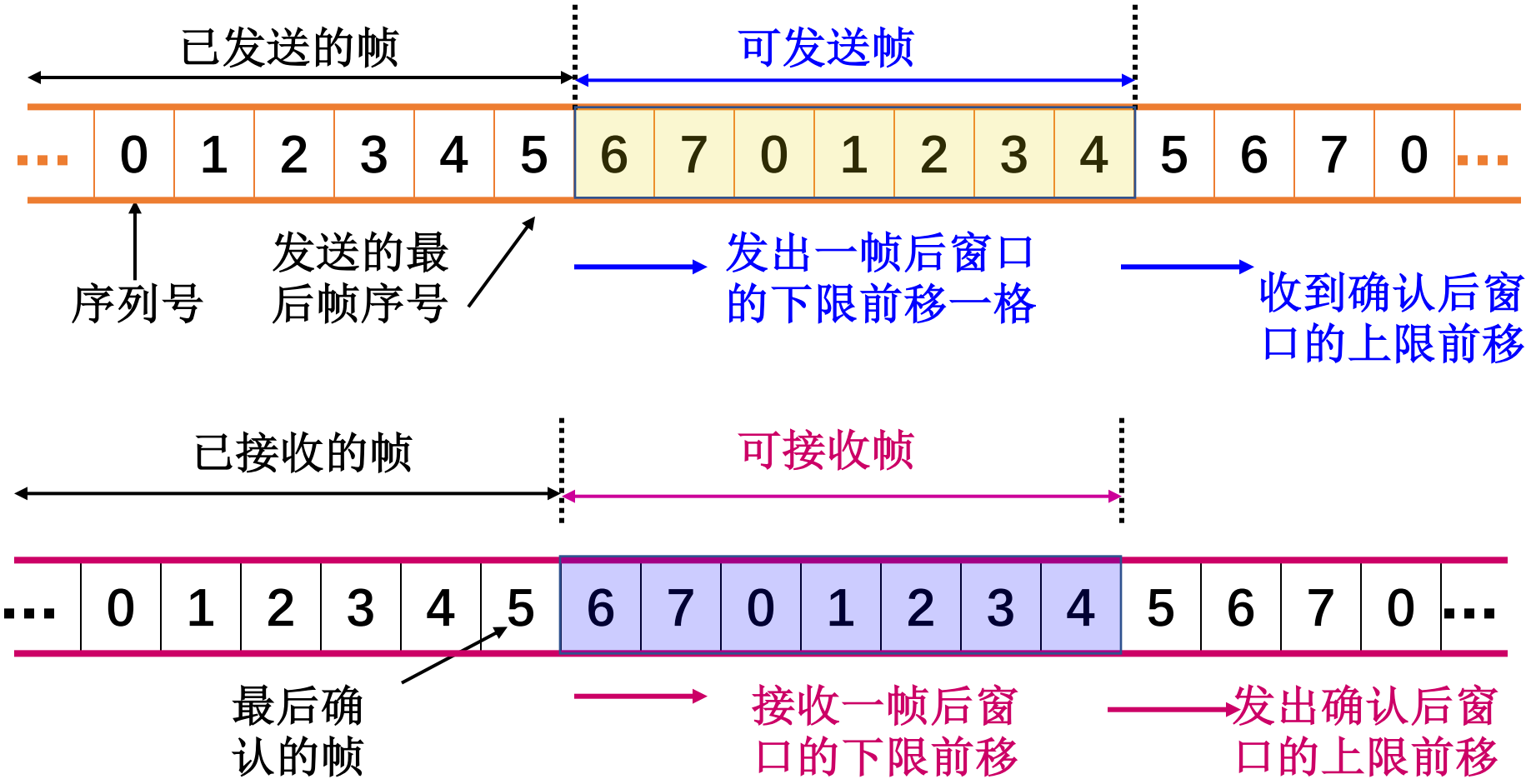
- 窗口机制
  - 发送方和接收方都具有一定容量的缓冲区（即窗口），发送端在收到确认之前可以发送多个帧
- 问题
  - 发送端一次可以发送多少个帧?
  - 某个帧出错或丢失怎么处理？

滑动窗口协议—协议基本思想

- 目的
  - 对可以连续发出的最多帧数（已发出但未确认的帧）作限制
- 序号使用
  - 循环重复使用有限的帧序号
- 流量控制：接收窗口驱动发送窗口的转动
  - 发送窗口：其大小记作W_T，表示在收到对方确认的信息之前，可以连续发出的最多数据帧数
  - 接收窗口：其大小记作W_R，为可以连续接收的最多数据帧数
- 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

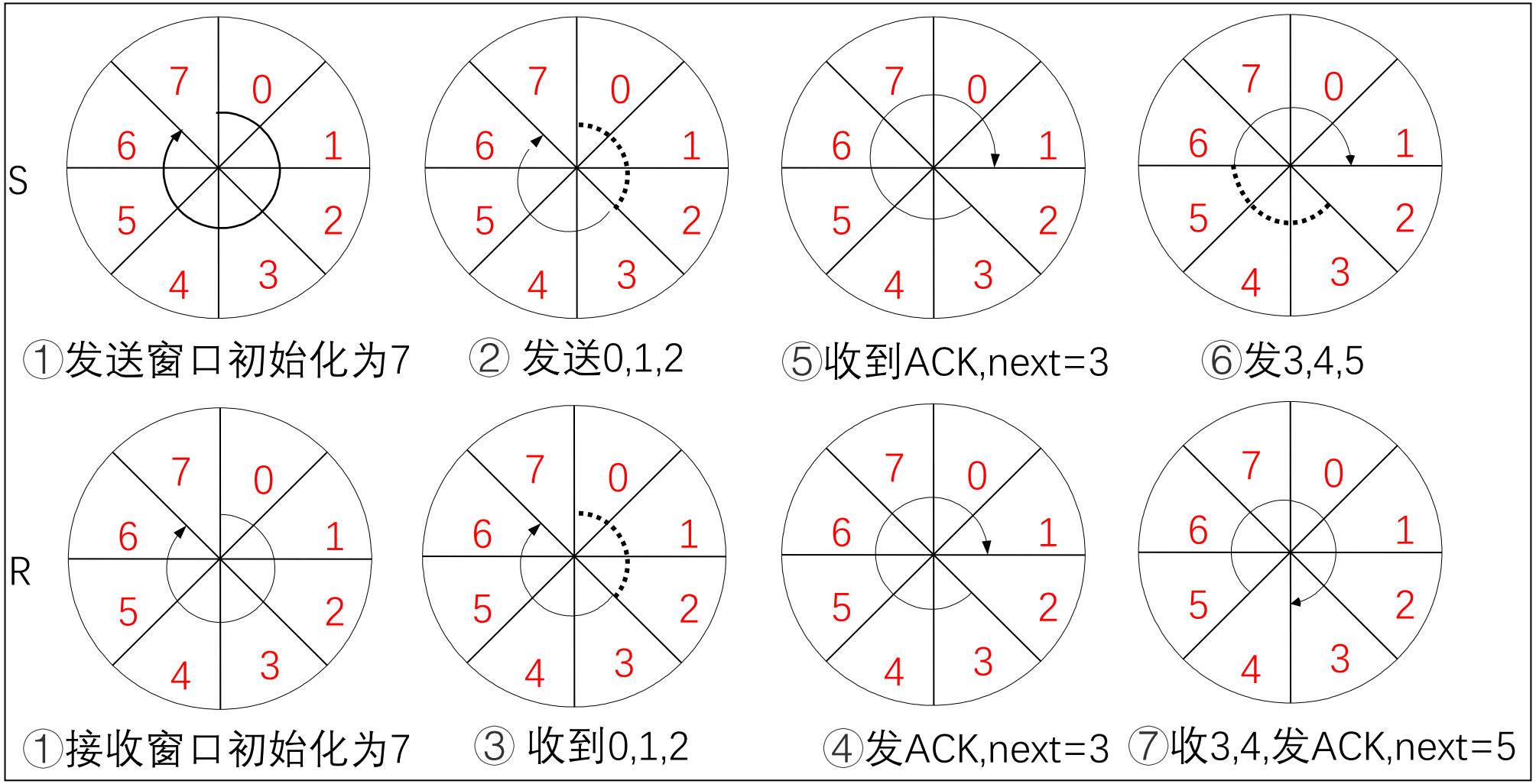
滑动窗口协议—协议基本思想

- 发送窗口与接收窗口(窗口大小7)



滑动窗口协议—协议基本思想

- 发送窗口与接收窗口(窗口大小7)



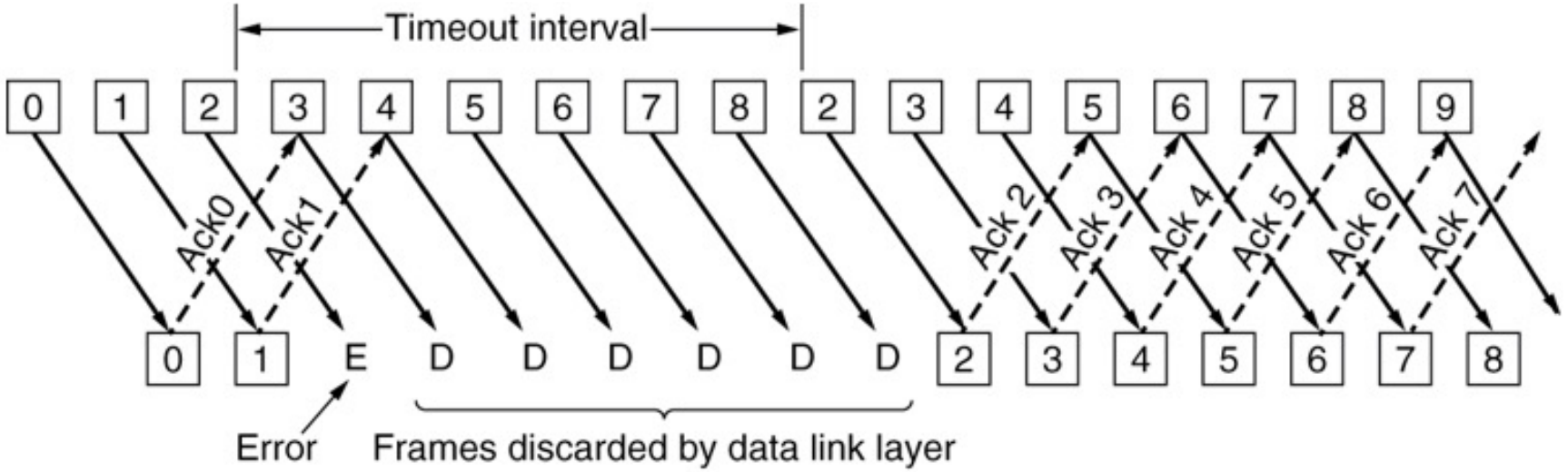
回退N协议—协议设计思想

- 出错全部重发
  - 当接收端收到一个出错帧或乱序帧时，丢弃所有的后继帧，并且不为这些帧发送确认
  - 发送端超时后，重传所有未被确认的帧
- 适用场景
  - 该策略对应接收窗口为1的情况，即只能按顺序接收帧
- 优缺点
  - 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

回退N协议—协议原理分析

➢ 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。



选择重传协议—协议设计思想

- 设计思想
  - 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
- 适用场景
  - 该策略对应接收窗口大于1的情况，即暂存接收窗口中序号在出错帧之后的数据帧
- 优缺点
  - 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

选择重传协议—协议原理分析

➢ 基本原理

- 在发送过程中，如果一个数据帧计时器超时，就认为该帧丢失或者被破坏；接收端只把出错的的帧丢弃，其后面的数据帧保存在缓存中，并向发送端回复NAK；发送端接收到NAK时，只重传出错的帧
- 如果落在窗口内的帧从未接受过，那么存储起来，等比它序列号小的所有帧都正确接收后，按次序交付给网络层
- 接收端收到的数据包的顺序可能和发送的数据包顺序不一样，因此在数据包里必须含有顺序号来帮助接收端进行排序。

本章内容

3.1 数据链路层的设计问题

3.2 差错检测和纠正

3.3 基本的数据链路层协议

3.4 滑动窗口协议

3.5 数据链路协议实例

1. 点到点链路层协议PPP
2. PPPoE

PPP协议简介

- PPP (Point-to-Point Protocol)协议由IETF制定，1994年成为正式标准（RFC1661）
- PPP协议是目前使用最多的数据链路层协议之一
- 能够在不同的链路上运行
- 能够承载不同的网络层分组
- 特点：简单、灵活

PPP协议实现的功能

- 利用帧定界符封装成帧
- 填充技术实现透明数据传输：字节填充、零比特填充
- 帧的差错检测
- 实时监测链路工作状态
- 设置链路最大传输单元（MTU）
- 网络层地址协商机制
- 数据压缩协商机制

PPP协议的构成

- 封装 (Encapsulation)
  - 提供在同一链路上支持不同的网络层协议
  - PPP既支持异步链路（无奇偶检验的8比特数据），也支持面向比特的同步链路
  - IP数据包在PPP帧中是其信息部分，其长度受到MTU的限制
- 链路控制协议 LCP (Link Control Protocol)。
  - 用来建立、配置和测试数据链路的链路控制协议，通信双方可协商一些选项
- 网络控制协议 NCP (Network Control Protocol)。
  - 其中每个协议支持一种不同的网络层协议，如IP、OSI的网络层、DECnet、AppleTalk等

PPPoE概述

- Ethernet优点
  - 原理简单，应用非常广，设备成本低
- Ethernet缺点
  - 安全性较低、不宜管理：使用广播信道，造成了安全性较低，无认证功能
- PPP优点
  - 原理简单
  - 安全性高：点对点信道，提供认证机制
  - 提供良好的访问控制和计费功能

PPPoE概述

- PPPoE（Point-to-Point Protocol over Ethernet）
  - 提供在以太网链路上的PPP连接
  - 实现了传统以太网不能提供的身份验证、加密，以及压缩等功能
  - 实现基于用户的访问控制、计费、业务类型分类等，运营商广泛支持
  - PPPoE使用Client/Server模型，服务器通常是接入服务器

本章总结

- 通过本章学习，重点掌握如下知识点：
  - 理解数据链路层在五层网络体系结构中的位置及解决的问题，了解数据链路层提供的服务类型
  - 掌握差错检测和纠正的基本原理和典型的编码方法
  - 掌握无错信道和有错信道上停等协议的设计和实现方法
  - 掌握回退N和选择重传两种典型滑动窗口协议的工作机制