网络技术基础

高智刚

M.P. & WeChat：13572460159

E-mail: gaozhigang@nwpu.edu.cn

第三章：数据链路层

内容回顾

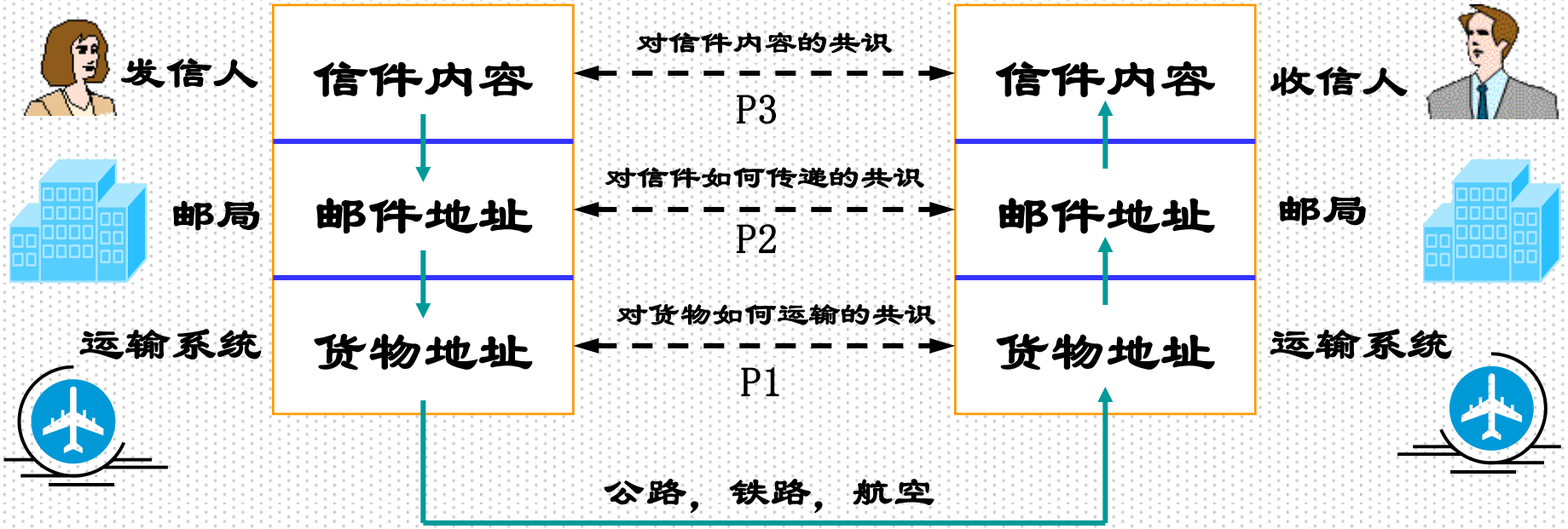
- 物理层主要功能
  - 为数据端设备提供传送数据的通路
  - 物理层要形成适合数据传输需要的实体，为数据传送服务。一是要保证数据能在其上正确通过，二是要提供足够的带宽，以减少信道上的拥塞。因此，我们讲了调制解调和编码解码技术，以及信道复用技术

内容回顾

- 网络对等层通信的实质
  - 网络分层体系结构原理禁止不同主机的对等层之间进行直接通信
  - 每一层必须依靠下层提供的服务来与另一台主机的对等层通信
    - 上层使用下层提供的服务——Service user；
    - 下层向上层提供服务——Service provider

内容回顾

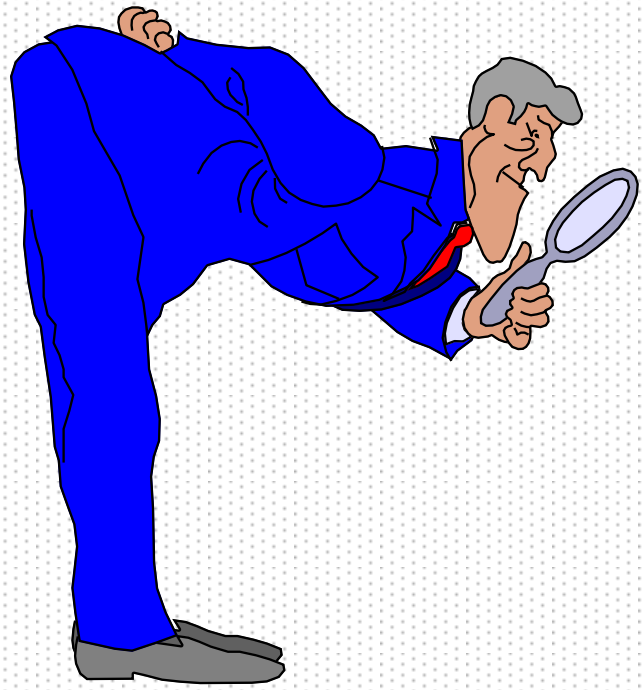
- 对等通信实例：两个人收发信件



- **问题：**
  - 收信人与发信人之间、邮局之间，他们是在直接通信吗？
  - 邮局、运输系统各向谁提供什么样的服务？
  - 邮局、收发信人各使用谁提供的什么服务？

内容回顾

- 网络对等层通信的实质



- 对等层实体之间实现的是虚拟的逻辑通信；
- 下层向上层提供服务；
- 上层依赖下层提供的服务来与其它主机上的对等层通信；
- 实际通信在最底层完成。

本章主要内容

- 数据链路层概述
- 差错检验
- 数据链路控制
- 自动请求重传（ARQ）
- 数据链路层协议实例

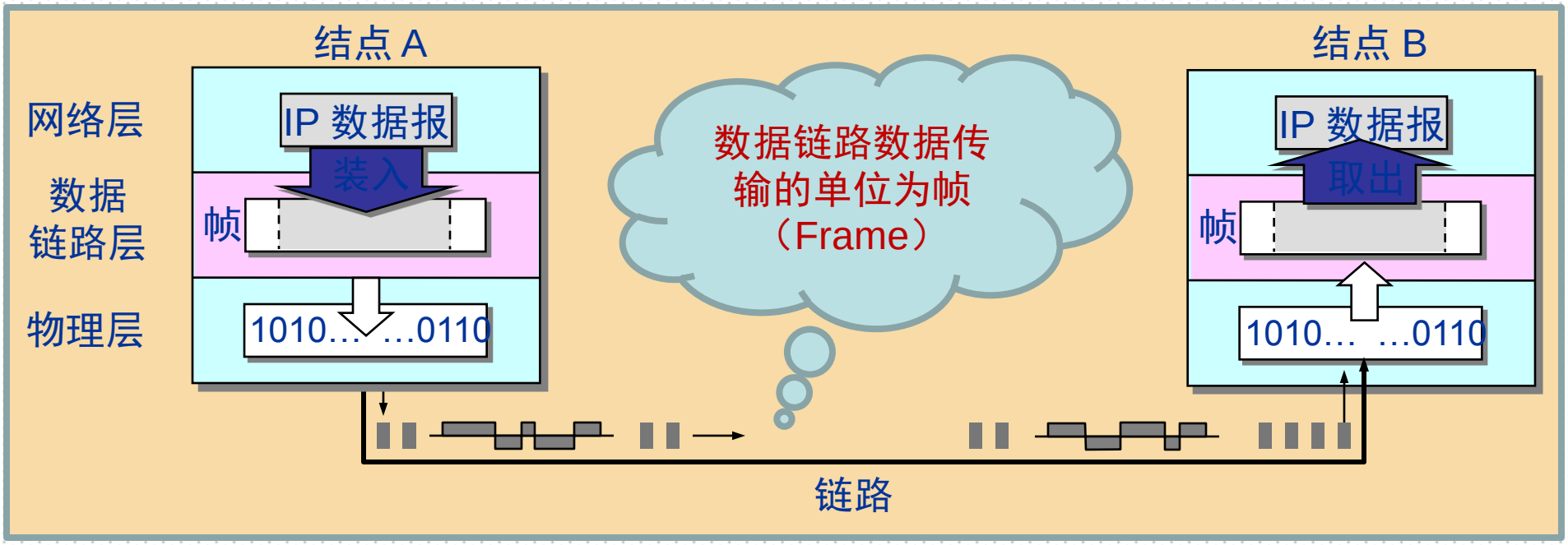
3.1 数据链路层概述

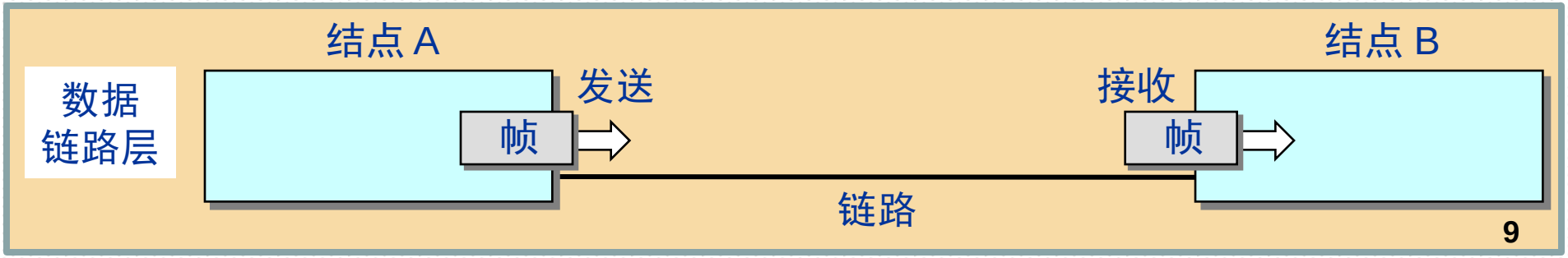
- 数据链路层任务
  - 数据链路：通信对等实体之间的数据传输通道。它是一个逻辑概念，包括物理线路和必要的传输控制协议
  - 数据链路层：在两台相邻的机器之间实现可靠、有效的通信，并在物理层提供比特流服务的基础上，建立相邻结点之间的数据链路，通过差错控制提供数据帧在信道上无差错的传输。相邻指的是两台机器通过一条通信信道连接起来

3.1 数据链路层概述

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- 数据链路层信息传输





3.1 数据链路层概述

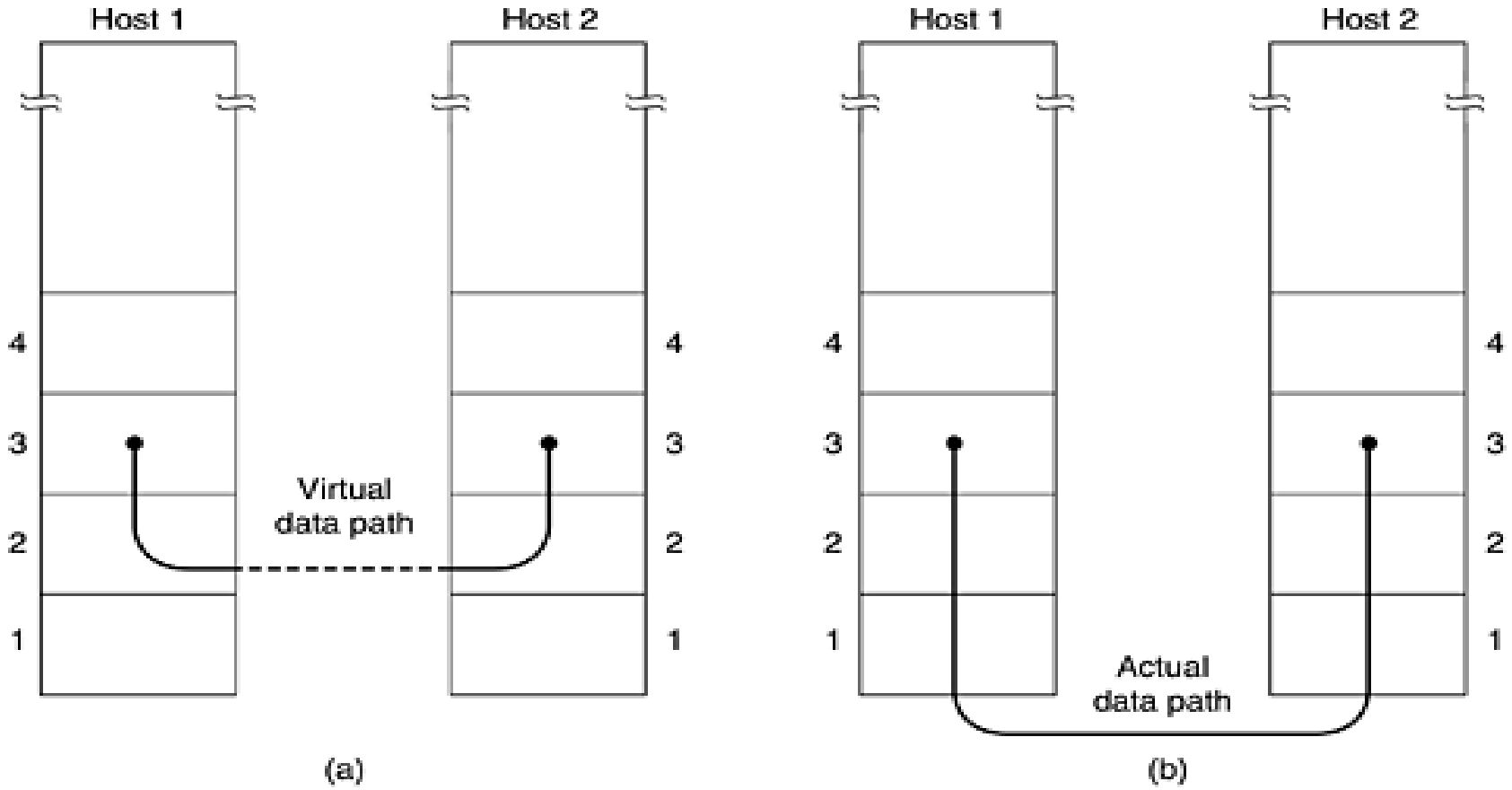
- 数据链路层设计要点
  - 数据链路层从网络层获得到分组，然后将分组(packet)封装到帧(frame)以便传输，每一帧包含一个帧头、一个有效载荷(用于存放分组)，以及一个帧尾。帧管理构成数据链路层工作的核心
  - 数据链路层的设计要点
    1）为网络层提供服务接口
    2）封装成帧
    3）错误控制

3.1 数据链路层概述

- 为网络层提供服务
  - 将数据从源机器的网络层传输到目标机器的网络层
  - 一般来说，在源机器的网络层中有一个实体，称为进程，它将一些数据位交给数据链路层，要求传输到目标机器。数据链路层的任务是将这些位传输给目标机器，然后再将这些数据进一步交给目标机器的网络层

3.1 数据链路层概述

- 为网络层提供服务



- 为网络层提供服务
  - 数据链路层的设计目标是向网络层提供服务，

    一般情况下，通常会提供三种可能的服务：

    （1）无确认的无连接服务

    （2）有确认的无连接服务

    （3）有确认的面向连接服务

- 无确认的无连接服务
  - 源机器向目标机器发送独立的帧，目标机器并不对这些帧进行确认。事先不建立逻辑连接，事后也不释放逻辑连接。若由于线路噪声而造成了某一帧丢失，则数据链路层不会检测到这样的丢帧现象，也不会恢复
  - 适合于线路错误率很低的时候，恢复过程可交由上层来完成，绝大多数LAN在数据链路层上都使用无确认的无连接服务

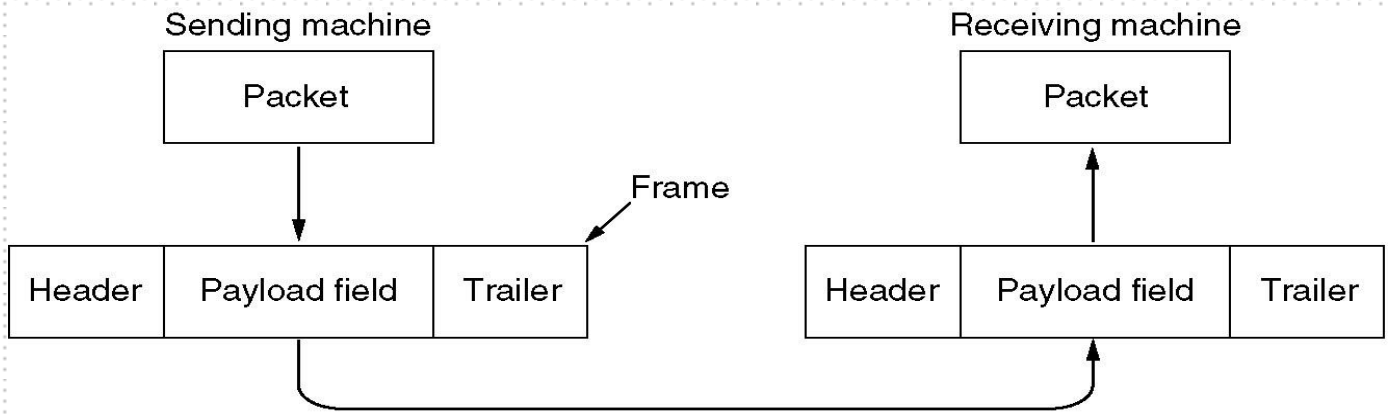
- 有确认的无连接服务
  - 为了提供可靠性，引入了有确认的无连接服务。当提供这种服务的时候，仍然没有使用逻辑连接，但是所发送的每一帧都需要单独确认。由此，发送方知道每一帧是否已经正确到达，如果有一帧在指定时间间隔内还没有到达，则发送方将再次发送该帧
  - 适合不可靠的信道，比如无线系统

- 有确认的面向连接服务
  - 利用这种服务，源机器和目标机器在传输数据之前首先建立一个逻辑连接，该连接上发送的每一帧都被编号，数据链路层保证每一帧都会真正被接收到。保证每一帧只被接受一次，且所有的帧都按照正确的顺序接受
  - 在无连接服务中，如果确认报文丢失了，则一个分组可能被发送多次，因而被接收多次。与无连接服务相比，面向连接服务相当于为网络层提供了一个可靠的位流

- 有确认的面向连接服务
  - 当使用面向连接的服务时，数据传输要经过三个不同的阶段：
    - 第一个阶段：建立连接，双方初始化各种变量和计数器，这些变量和计数器记录了哪些帧已经接收到，哪些帧还没有
    - 第二个阶段：一个或者多个数据帧被真正传输出去
    - 第三个阶段：连接被释放，所有的变量，缓冲区，以及其它用于维护该连接的资源也随之被释放

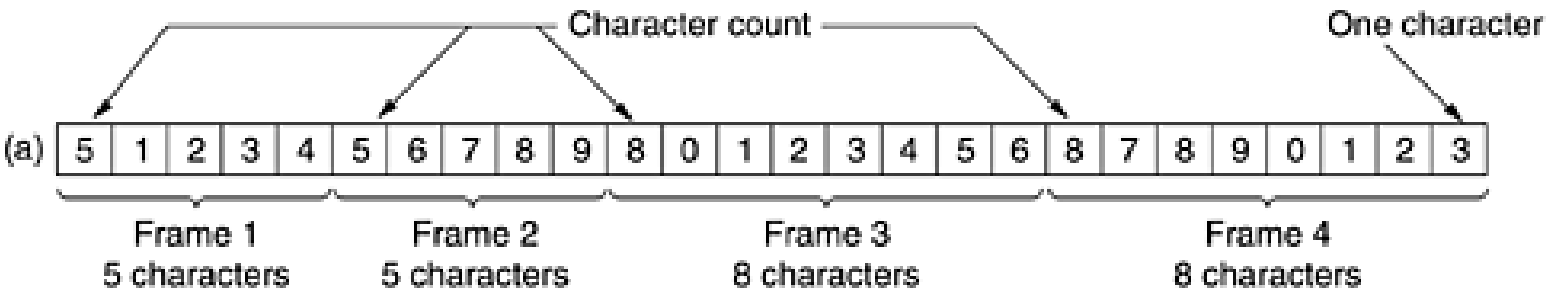
- 封装成帧
  - 为了向网络层提供服务，数据链路层必须使用物理层提供的服务。物理层的任务是接收一个原始的位流，并试图将它递交给目标机器。这个位流并不能保证没有错误，接收到的位的数量可能少于、等于或多于发送的位的数量，且它们可能有不同的值。检测错误（纠正错误）的工作由数据链路层完成

- 封装成帧
  - 数据链路层一般的做法是将位流分解成离散的帧，并计算每一帧的校验和。当一帧到达目标机器的时候，重新计算校验和从而发现错误，然后采取措施来处理错误

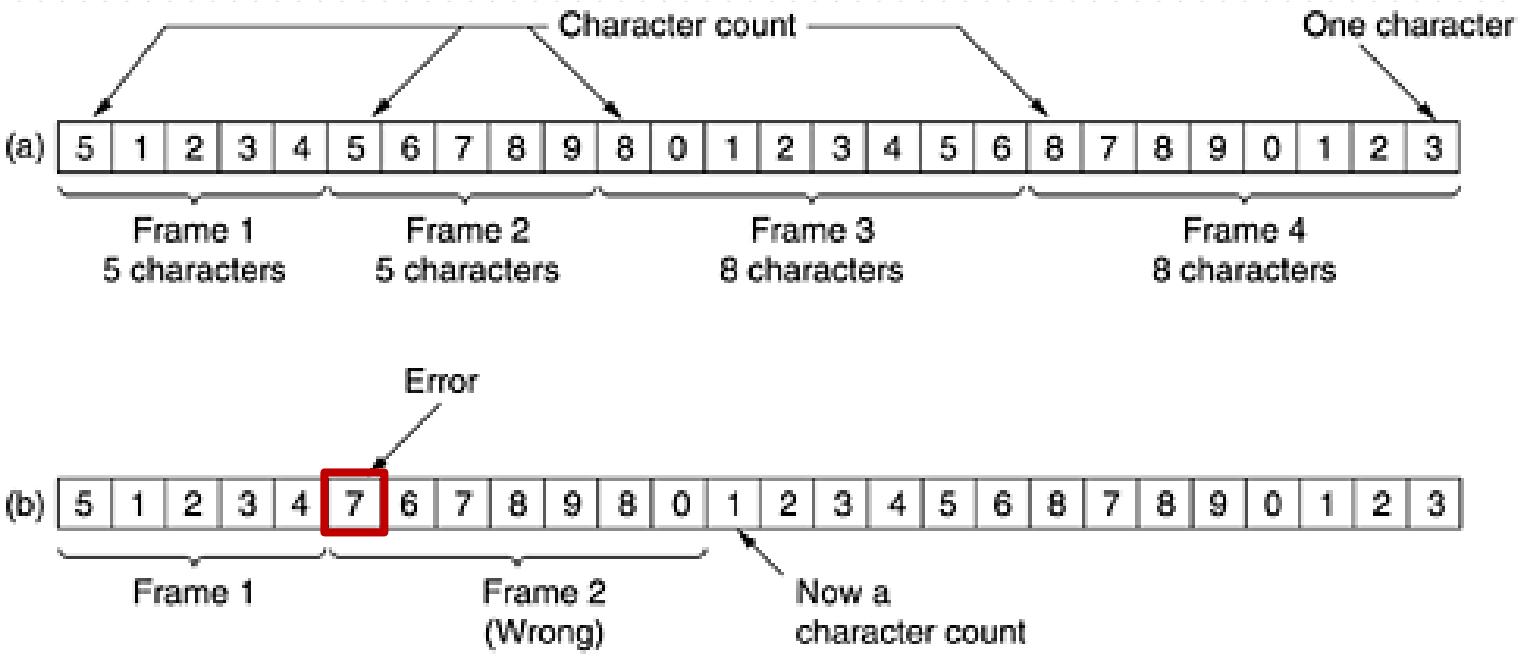


- 封装成帧
  - 成帧最简单的方法是在帧之间插入时隙（time gap），如同在普通正文的英文单词之间插入空格。但依靠时间来标识每一帧的起始和结束位置风险太大，有必要设计其他的成帧方法
  - 成帧四种方法
    1）字符计数法
    2）含字节填充的分界符法
    3）含位填充的分界标志法
    4）物理层编码违例法

- 字符计数法
  - 利用头部的一个域来指明该帧中的字符数。当目标端看到这个字符计数值的时候，可以得知后面跟着多少个字符，因此也就知道了该帧的结束在哪里



- 字符计数法
  - 缺点：计数值可能因为传输错误而被打乱

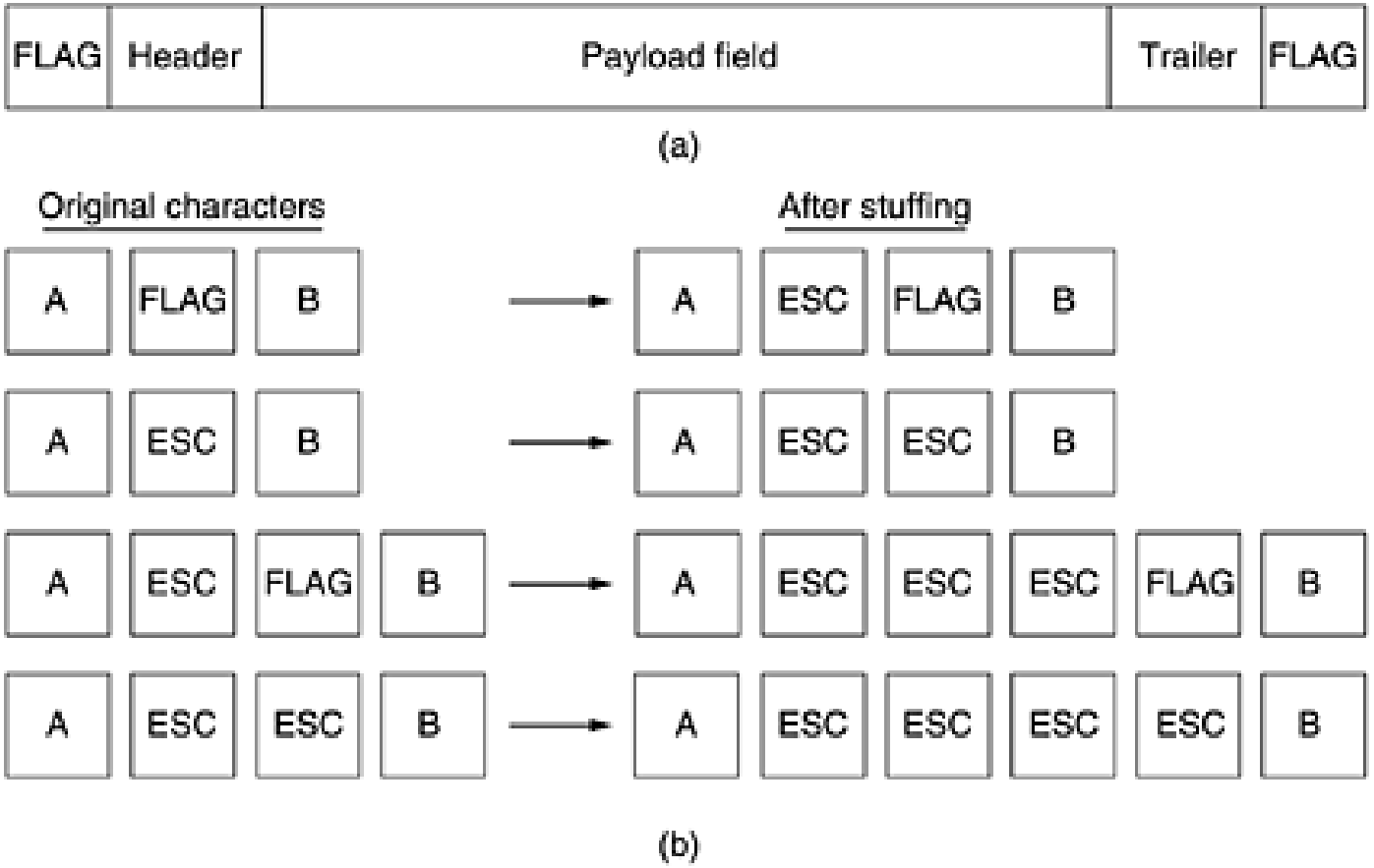


- 字符计数法
  - 接收方会失去同步，从而不可能找到下一帧的起始位置。同时，由于校验和不正确，目标方虽然知道了该帧已被损坏，但是无法知道下一帧从哪里开始。在这种情况下，给源方发送一个“请求重传”也很难恢复错误，因为目标方并不知道该跳过多少个字符才能到达重传的开始处。由于这个原因，字符计数法已很少使用

- 含字节填充的分界符法
  - 考虑到错误之后重新同步的问题，做法是让每一帧都用一些特殊的字符作为开始和结束。绝大多数协议使用一个标志字节（Flag Byte），作为起始和结束分界符。按照这种做法，如果接收方丢失了同步，只需搜索标志字节就能够找到当前帧的结束位置。两个连续的标志字节代表了当前帧的结束和下一帧的开始

- 含字节填充的分界符法
  - 当二进制数据被传输时（例如程序或者浮点数据），标志字节的位模式可能会出现在传输过程中，这种位模式往往会干扰帧的分界。解决这个问题的一种方法是，发送方的数据链路层在这种“偶尔出现的字节”前面插入一个特殊的转义字符（ESC）。接收端的数据链路层在将数据送给网络层之前删掉标志转义字符。这种计数被称为字节填充或者字符填充

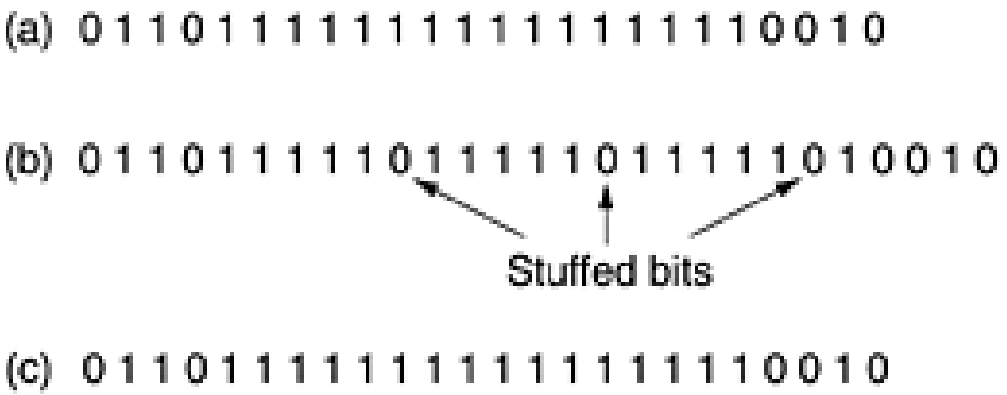
- 含字节填充的分界符法


FLAG
Header
Payload field
Trailer
FLAG
(a)
Original characters
After stuffing
A FLAG B → A ESC FLAG B
A ESC B → A ESC ESC B
A ESC FLAG B → A ESC ESC ESC FLAG B
A ESC ESC B → A ESC ESC ESC ESC B
(b)

- 含字节填充的分界符法
  - 字节填充成帧方法的主要缺点是，它依赖于8位字符的模式，但是并不是所有的字符码都使用8位字符（UNICODE使用16位字符）。并且随着网络的发展，在成帧机制中内含字符码长度的缺点变得越来越明显，所以有必要开发一种新的技术以便允许任意长度的字符

- 含位填充的分界符法
  - 新的技术允许数据帧包含任意长度的位，也允许每个字符有任意长度的位。工作方式如下：每一帧的开始和结束都有一个特殊的位模式01111110。当发送方的数据链路层碰到数据中5个连续的位1的时候，自动在输出位流中填充一个位0。这种位填充的方法与字节填充机制非常相似。当接收方碰到5个连续的1时，如果后面是0则自动去掉，如果是1的话则代表着数据的结束，转入结束处理

- 含位填充的分界符法
  - 位填充机制中，通过标志模式可以明确的识别出两帧之间的边界。因此如果接收方失去了帧同步，只需在输入流中扫描标志序列即可，因为标志序列只可能出现在帧边界上，永远不可能出现在数据中



图a：原始数据

图b：线路上的数据

图c：删除填充之后存储在接收方存储器中的数据。

- 封装成帧补充说明
  - 许多实际的数据链路层联合使用字符计数法和其他某种方法，以保证额外的安全性。当一帧到达时，先利用计数域定位到该帧的结束处，只有当这个位置上确实出现了正确的分界符，并且帧的校验和也正确的情况下，该帧才能认为是有效的。否则接受方在输入流中扫描下一个分界符

3.2 差错检验

- 在物理层中丢失信息、干扰信息及顺序不正确等情况都可能发生，在数据链路层中必须用纠错码来检错与纠错。数据链路层是对物理层传输原始比特流功能的加强，将物理层提供的可能出错的物理连接改造成为逻辑上无差错的数据链路，使之对网络层表现为一个无差错的线路

3.2 差错检验

- 一个实用的通信系统必须具备发现（即检测）差错的能力，并采取某种措施纠正，使差错被控制在所能允许的尽可能小的范围内，这就是差错控制过程，也是数据链路层的主要功能之一。对差错编码（如奇偶校验码检查或CRC）的检查，可以判定一帧在传输过程中是否发生了错误。一旦发现错误，一般可以采用反馈重发的方法来纠正。

因此，差错控制的前提是进行差错检验

3.2 差错检验

- 检验差错的常用方法是对被传送的信息进行适当的编码，给信息码加上冗余码
- 冗余码一般是固定的且比信息码的长度短，因为过长会增加额外的负担
- 冗余码通过一定的运算得出，它与信息码之间具备某种特定关系。由信息码求冗余码的运算是由通信双方的数据链路层协议约定

3.2 差错检验

- **发送方**
  - 将信息码和冗余码一起封装在帧里，通过信道发出
- **接收方**
  - 接收到帧后，检验它们之间的关系是否符合双方约定，符合就认为没有传输差错，不符合就认为发现了差错
- **差错检验方法包括奇偶校验码、海明码、循环冗余码等**

3.2 差错检验

- 奇偶校验码基本原理
  - 在ASCII代码后增加一位校验位，使码中“1”的个数成奇数（奇校验）或偶数（偶校验）。经过传输后，如果其中一位（甚至校验位）出错，则接收端按同样的规则就能发现错误

3.2 差错检验

- 奇偶校验码基本原理
  - 奇校验：确保整个被传输的数据中“1”的个数是奇数个，即载荷数据中“1”的个数是奇数个时校验位填“0”，否则填“1”
  - 偶校验：确保整个被传输的数据中“1”的个数是偶数个，即载荷数据中“1”的个数是奇数个时校验位填“1”，否则填“0”

3.2 差错检验

- 奇偶校验码基本原理
  - 这种方法简单实用，但如果数据中发生多位数据错误就可能检测不出来，更检测不到错误发生在哪一位
  - 在实际使用时又可分为垂直奇偶校验、水平奇偶校验和水平垂直奇偶校验等几种

3.2 差错检验

- 海明码
  - 1950年海明（Hamming）研究了用冗余数据位来检测和纠正代码差错的理论和方法。他指出可以在数据代码上添加若干冗余位组成码值
  - 海明码的编码方案

    n位纠错码=m位数据+k位冗余位

3.2 差错检验

- 海明码基本原理
  - 首先把码字的位从1到n编号，并把这个编号表示成二进制数，即2的幂之和。然后对2的每一个幂设置一个奇偶位。其余各位放置数据并参加对应每一二进制数位的校验
  - 例如，对于6位号，由于6=110，所以6号位参加第2位和第4位的奇偶校验，而不参加第1位奇偶校验。类似的，9位号参加第1位和第8位的校验而不参加第2位或第4位的校验。
  - 把奇偶校验分配在第1、2、4、8等2的幂次位上，其他位放置数据并参与对应幂次位校验

	8	4	2	1
3	0	0	1	1
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1

- 海明码校验举例
  - 假设传送的信息为“1001011”
  - 我们把各个数据放在3、5、6、7、9、10、11等位置上，1、2、4、8位留做校验位

		1		0	0	1		0	1	1
1	2	3	4	5	6	7	8	9	10	11

- 海明码校验举例 “1001011”
  - 根据图，3、5、7、9、11的二进制编码的第一位为1，所以3、5、7、9、11号位参加第一位校验，若按偶校验计算，1号位应为1

1	X	1	X	0	0	1	X	0	1	1
1	2	3	4	5	6	7	8	9	10	11

- 海明码校验举例 “1001011”
  - 类似地，3、6、7、10、11号位参加2号位校验，5、6、7号位参加4号位校验，9、10、11号位参加8号位校验，全部按偶校验计算，最终得到

1	0	1	1	0	0	1	0	0	1	1
1	2	3	4	5	6	7	8	9	10	11

- 海明码校验举例 “1001011”
  - 如果码字传输中出错，如6号位出错，即变成

1	0	1	1	0	1	1	0	0	1	1
1	2	3	4	5	6	7	8	9	10	11

  - 当接收方按照同样规则计算奇偶位时，发现1和8号位的奇偶性正确，而2和4号位的奇偶性不对，于是2+4=6，立即可确认错在6号位

- 循环冗余码
  - 码多项式：任何一个二进制编码的位串都可以用一个多项式来表示，多项式的系数由该位串的码元表示，只有0和1
  - 对于n位长度的位串，有：

    $C(x)=C_{n-1}x^{n-1}+C_{n-2}x^{n-2}+\cdots+C_1x+C_0$
  - 举例：

    1 0 1 0 0 0 1 为 x^6+x^4+1

- 循环冗余码基本思想
  - 收发双方约定一个生成多项式G(x)，发送方根据发送的数据K(x)，与G(x)计算出CRC校验和R(x)并把它加在数据的末尾，使这个带校验和的数据多项式能被G(x)除尽。接收方则用G(x)去除接收到的数据，若有余数，则传输有错

3.2 差错检验

- 循环冗余码计算方法

（1）设生成多项式G(x)为r阶，在k位原始帧K(x)的末尾附加r个零，使帧长为k+r位，则相应的多项式是$x^r \cdot K(x)$

（2）按模2除法用对应于G(x)的位串去除对应于$x^r \cdot K(x)$的位串，所得的余数为R(x)

（3）按模2减（加）法从对应于$x^r \cdot K(x)$的位串中减去（加上）余数R(x)。结果就是要传送的带校验和的帧，叫多项式T(x)

- 循环冗余码计算方法
  - 多项式的运算法则是模2运算。按照它的运算法则，加法不进位，减法不借位。加法和减法两者都与异或运算相同。举例：

```
  00110011          11110000
+ 11001101        - 10100110
----------        ----------
  11111110          01010110
```

3.2 差错检验

- 循环冗余码计算方法

帧：1101011011

生成多项式：

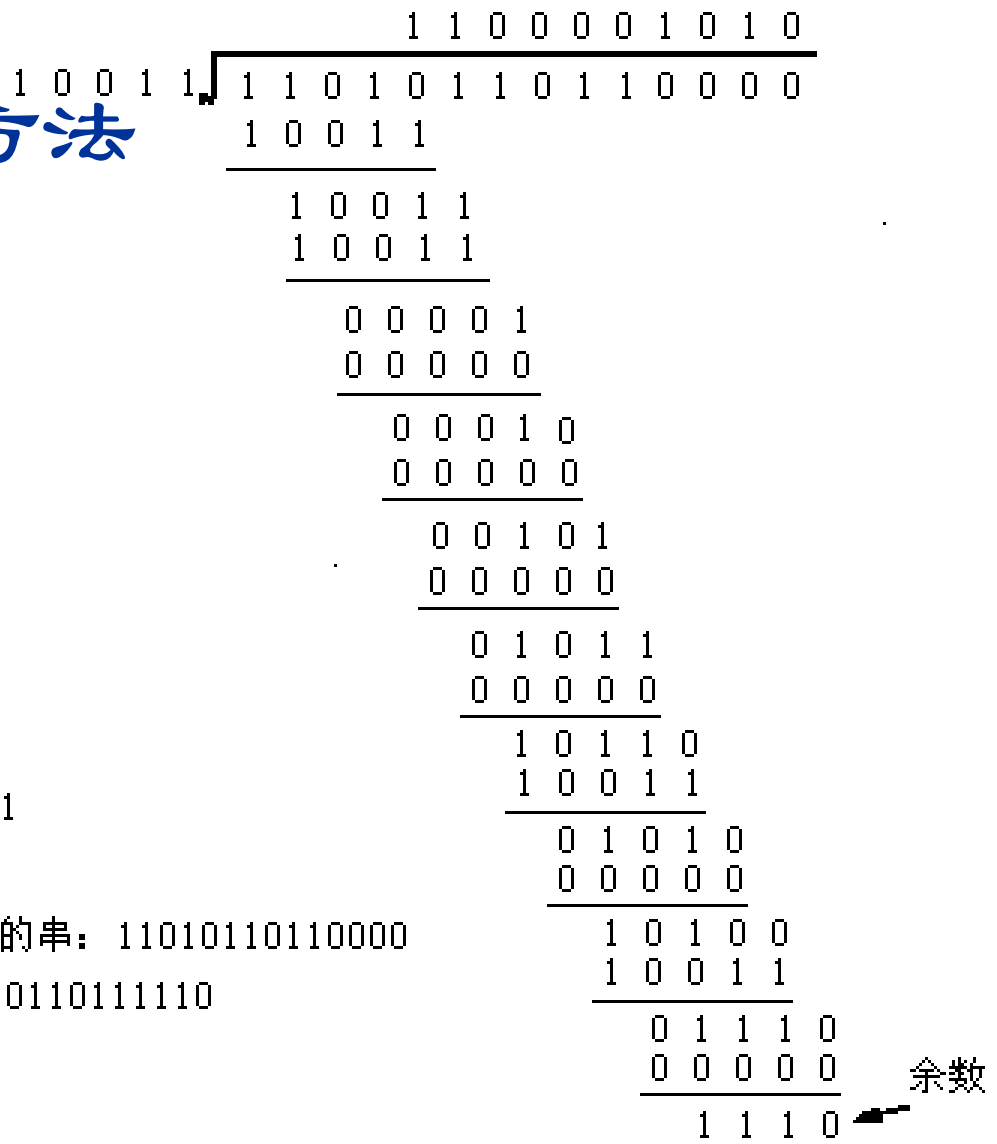
$G(x)=x^4+x+1$

帧 ： 1101011011

除数 ： 10011

附加4个零后形成的串：11010110110000

传输的帧 ： 11010110111110



3.2 差错检验

- 常用的生成多项式
  - CRC-8=x^8+x^2+x+1
  - CRC-16=$x^{16}+x^{15}+x^2+1$
  - CRC-32=$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

CRC-8用于ATM信元头差错检验，CRC-16是HDLC规程中使用的CRC检验生成多项式，CRC-32是IEEE802.3以太网采用的CRC检验生成多项式。这些多项式都是经过数学上的精心设计和实际验证的。

3.3 数据链路控制

- 数据链路控制的基本思想
  - 一条可靠的数据链路应该满足以下两个条件：

    (1) 传输的任何数据，既不会出现差错也不会丢失数据

    (2) 不管发送方以多快的速率发送数据，接收方总能够来得及接收、处理并上交主机。也就是接收方有足够的接收缓存和处理速度

3.3 数据链路控制

- 实际应用的数据链路并不能满足上述的条件
- 条件一不满足：出错和丢帧

  —〉差错控制
- 条件二不满足：接收缓存和处理速度

  —〉流量控制

3.3 数据链路控制

- **差错控制：使得链路传输出现差错时得到补救**
- **主要有两种差错发生：**
  - （1）帧丢失：一个数据帧未能到达接收端；
  - （2）帧损坏：例如其中有几位数据出错。
- **差错控制 → 反馈重传机制**

3.3 数据链路控制

- **流量控制：**
  - 保证发送数据在任何情况下都不会“淹没”接收方的接收缓存（接收缓存溢出），从而不会丢失数据，且应使传输达到理想吞吐率
- **基本思想：**
  - 接收方根据其缓存状况控制发送方数据流量
- **流量控制 → 滑动窗口机制**

- 反馈重传机制（确认-重传机制）
  - 基本思想：接收方对接收到的数据进行差错检验后，以某种方式向发送方反馈差错状况，称为确认，发送方根据确认信息对出现传输差错的帧进行重传
  - 差错控制的常用方法，也是数据链路控制的一个基本机制

3.3 数据链路控制

- 反馈重传机制包括两步：

  （1）接收方反馈确认信息

  （2）发送方重传差错帧

(1) 接收方反馈确认信息方法

- 正确认或肯定确认：
  - 接收方收到一个经过检验正确无错的帧后，返回确认。记为ACK（ACKowledgement）
- 累计确认：
  - 接收方收到多个连续且正确的数据帧以后，只对最后一个帧发回一个确认
  - 累计确认表明该帧及其以前所有帧均正确收到

(1) 接收方反馈确认信息方法

- 捎带确认：
  - 在双向数据传输情况下，将确认信息放在自己的数据帧的首部字段中捎带过去
- 负确认：
  - 接收方收到一个有差错的帧后，返回对此帧的负确认（NAK，Negative ACKnowledgement）

累计确认和捎带确认都可以提高传输效率

(2) 发送方重传差错帧

- 超时重传:
  - 发送方在发送完一帧时即启动一个重传定时器，若由它设定的重传时间到且未收到反馈的确认信息，则重传此帧
  - 经常采用的重传方式
- 负确认重传:
  - 发送方收到接收方对一个帧的负确认，重传此帧

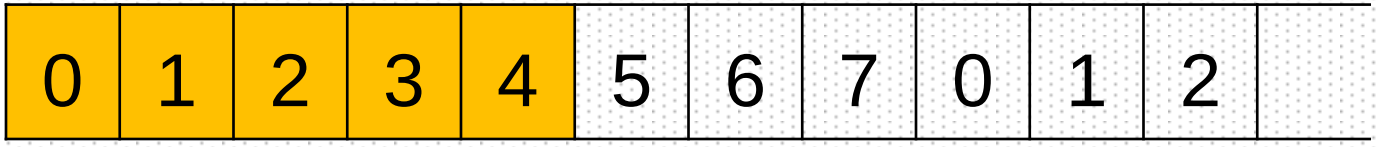
- 滑动窗口机制
  - 滑动窗口是数据链路控制的一个基本机制，发送方和接收方分别设置发送窗口和接收窗口，数据传输过程中在接收方的控制下向前滑动，从而对数据传输流量进行控制

- 滑动窗口机制
  - 发送窗口：用来对发送方进行流量控制
    - 落在窗口内的帧可连续发送，其大小W_T指明在收到对方ACK之前发送方最多可以发送多少帧
  - 接收窗口：控制哪些帧可以接收
    - 只有到达帧的序号落在接收窗口W_R之内时才可以被接收，否则将被丢弃

- 滑动窗口机制举例

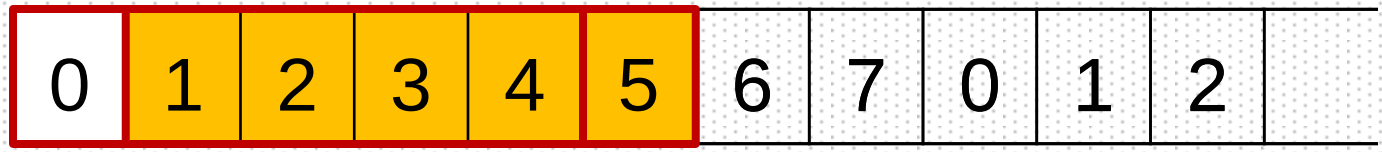
发送序号使用3bit来编码，即发送序号可以有从0~7的8个不同的序号，发送窗口W_T=5

- （1）发送窗口内共有从0~4的5个序号，这些帧现在可以连续发送，而5号及以后的帧是当前不能发送的。当发送方发送完了窗口内的全部5个帧，若没有收到接收方的ACK，就必须停止发送



（a）初始状态，可发送0~4号帧

- （2）收到了接收方对0号帧的ACK1，发送窗口向前滑动1个序号。则5号帧进入发送窗口之内，可以发送



（b）收到0号帧的确认，向前滑动一个号，可发送1~5号帧

- （3）假设又收到了对3号帧的累积确认ACK4，说明接收方又正确地收到了1~3号帧，于是发送窗再向前移动3个序号，那么6号、7号和0号帧又进入发送窗口

（c）又收到3号帧的累积确认，向前滑动3个号，可发送4~7及0号帧

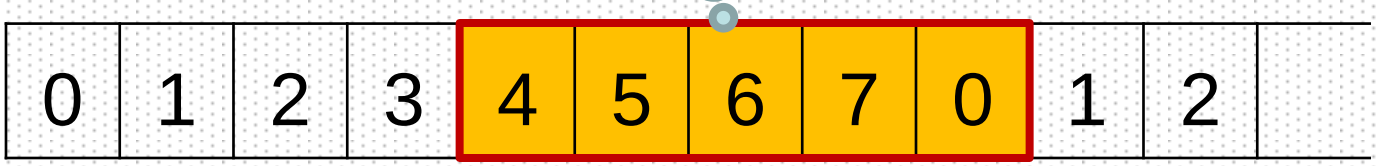
- （3）假设又收到了对3号帧的累积确认ACK4，说明接收方又正[illegible]于是发送窗再向前移动3[illegible]0号帧又进入发送窗口

窗左边：得到ACK确认的帧

（c）又收到3号帧的累积确认，向前滑动3个号，可发送4~7及0号帧

- （3）假设又收到了对3号帧的累积确认ACK4，说明接收方又正[illegible]于是发送窗再向前移动3个[illegible]0号帧又进入发送窗口

窗口内：可以发送的帧（包括已发送但未确认的帧、尚未发送的帧）



（c）又收到3号帧的累积确认，向前滑动3个号，可发送4~7及0号帧

- （3）假设又收到了对3号帧的累积确认ACK4，说明接收方又正[illegible]于是发送窗再向前移动3个[illegible]0号帧又进入发送窗口

窗右边：不可以发送的数据帧

（c）又收到3号帧的累积确认，向前滑动3个号，可发送4~7及0号帧

- 滑动窗口机制的具体实施方式
  - 当接收方收到一个有序且无差错的帧后—>接收窗口向前滑动—>准备接收下一帧，并向发送

由此可见，接收方的ACK作为授权发送方发送数据的凭证，接收方可以根据自己的接收能力来控制确认的发送时机，从而实现对传输流量的控制

能向前滑动，滑动的长度取决于接收方确认的序号。向前滑动后，又有新的待发帧落入发送窗口，可以被发送

3.3 数据链路控制

- 在滑动窗口机制中，为控制传输流量可以设置合适大小的W_T，一般不超过接收方接收缓存大小，这样发送的数据就不容易淹没接收缓存造成数据丢失
- 可采用可变滑动窗口，由接收方根据目前可用接收缓存的大小动态改变W_T，如在TCP（传输层的传输控制协议）流量控制中就采用可变滑动窗口

3.4 自动请求重传

- 反馈重传机制对出差错的数据帧的重传是自动进行的，因此这种控制机制称为自动请求重传(ARQ, Automatic Repeat Request)
- 根据反馈重传方式的不同可分为：
  - 停等ARQ
  - 回退-N ARQ
  - 选择重传ARQ

3.4 自动请求重传

- 自动请求重传ARQ既使用了反馈重传机制对传输过程进行差错控制，同时也使用了滑动窗口机制进行流量控制，从而保证了数据链路层的可靠的数据帧传输

3.4 自动请求重传

- 停等ARQ基本思想
  - 在发送方发出一个数据帧后停下来不再发送，等待接收方的ACK到达后才发送下一帧数据
  - 停等ARQ实际上也使用了滑动窗口技术，它的发送窗口大小是W_T=1，接收窗口W_R大小=1。因此，在发送出去一个数据帧后，停止发送，等待接收方的ACK

- 停等ARQ三种传输差错

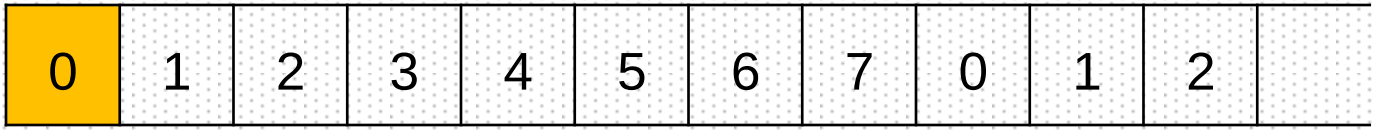
（1）发送方数据丢失，接收方收不到，发送方也不可能收到ACK

（2）接收方收到数据帧，检测出帧数据差错

（3）接收方正确收到数据帧，但发出的ACK丢失，发送方收不到ACK

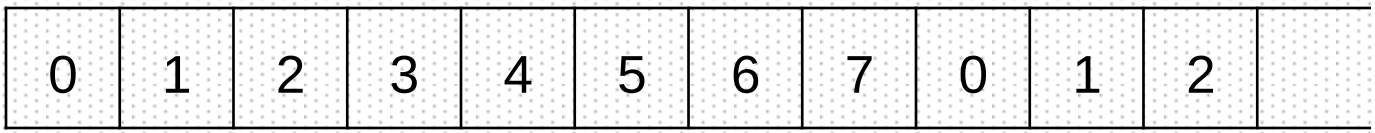
- 停等ARQ示例

发送方



（a）初始状态，可发送0号帧

接收方



（a）初始状态，准备接收0号帧

- 停等ARQ示例


发送方
0 1 2 3 4 5 6 7 0 1 2
ACK1
（b）向前滑动1个号，可发送1号帧始状态
接收方
0 1 2 3 4 5 6 7 0 1 2
（b）可接收1号帧始状态

- 停等ARQ示例

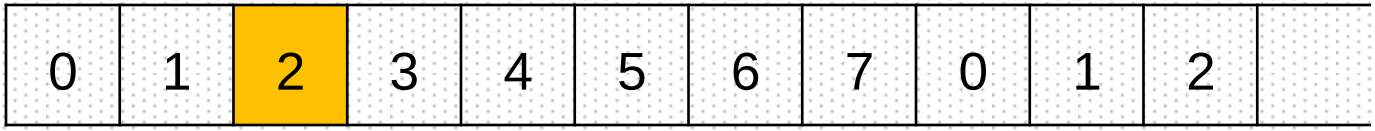

发送方
0
1
2
3
4
5
6
7
0
1
2
ACK1
（b）向前滑动1个号，可发送1号帧始状态
接收方
0
1
2
3
4
5
6
7
0
1
2
（b）可接收1号帧始状态

- 停等ARQ示例

（c）可接收2号帧始状态

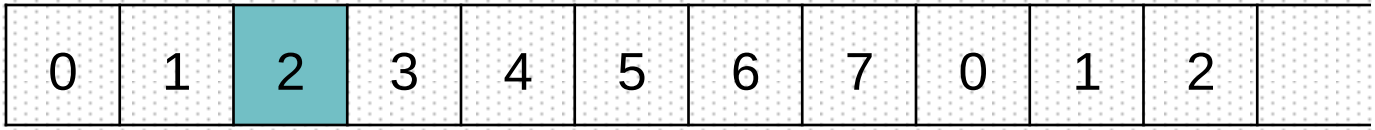
- 停等ARQ示例

发送方



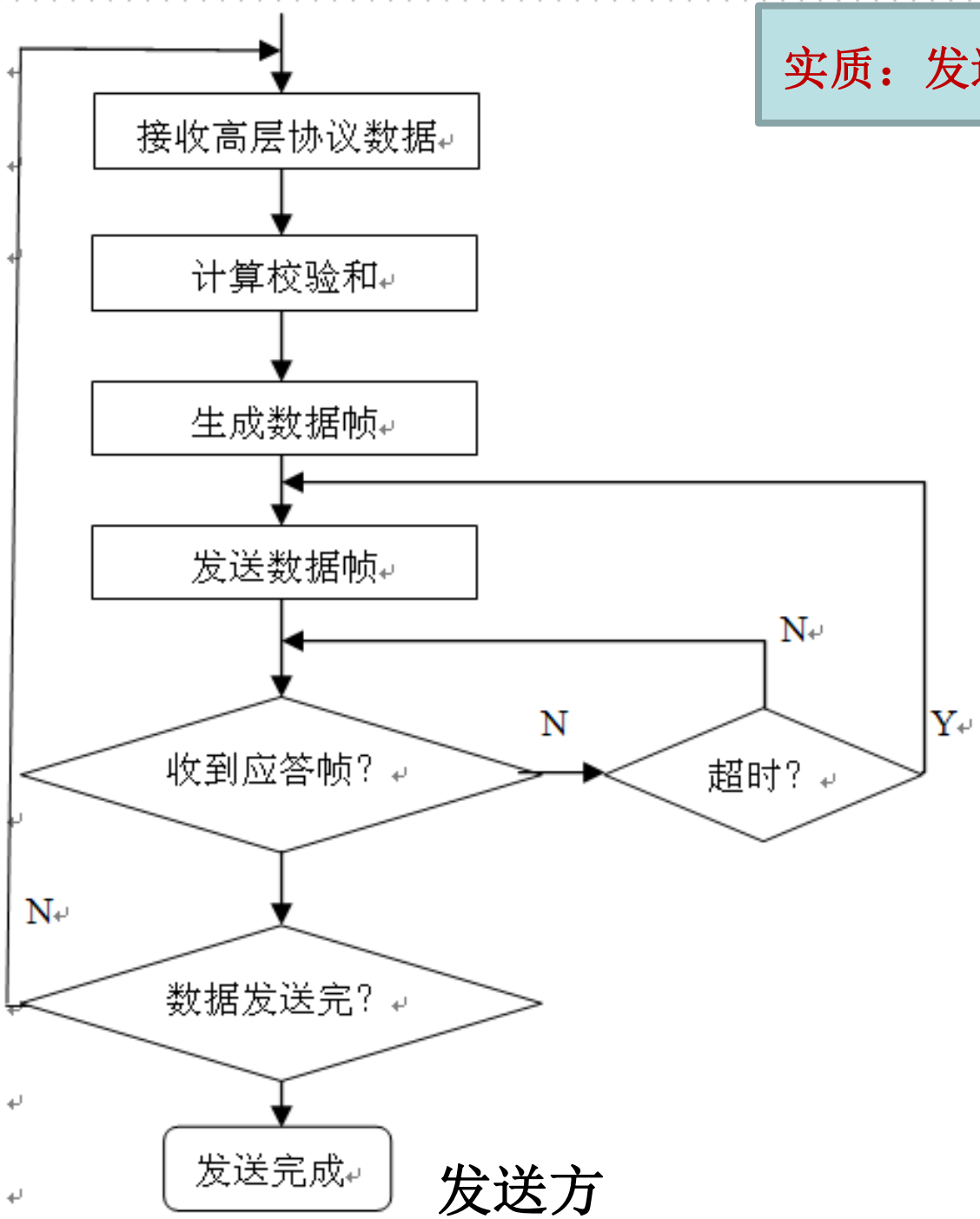
（c）向前滑动1个号，可发送3号帧始状态

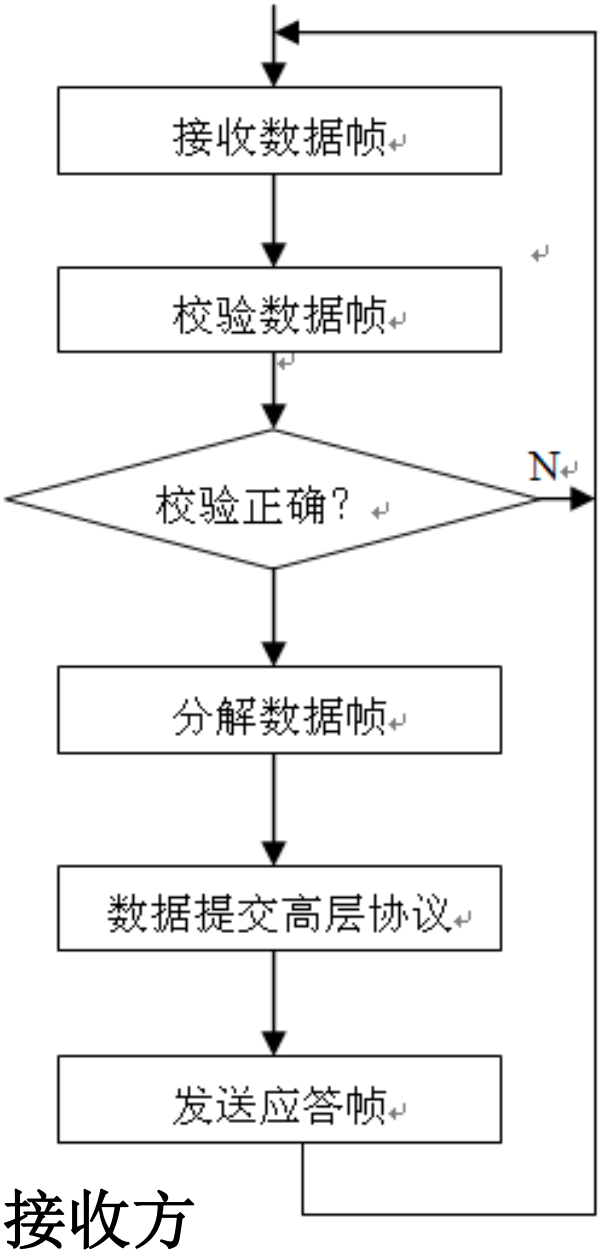
接收方



（c）可接收3号帧始状态

实质：发送窗口W_T=1，接收窗口W_R=1





3.4 自动请求重传

- 停等ARQ三种传输差错解决方法

(1) 发送方数据丢失，接收方收不到，发送方也不可能收到ACK

- 超时重传

需要注意：$T_{OUT} > T_{DATA}+T_{ACK}+2\tau+T_{PRO}$

(2) 接收方收到数据帧，检测出帧数据差错

- 负确认重传
- 超时重传

(3) 接收方正确收到数据帧，但发出的ACK丢失，发送方收不到ACK

- 超时重传

存在问题：接收方将收到两个同样的数据帧，接收到无法判断是相同的新帧还是重传的旧帧

解决办法：为数据帧和确认帧编上序号。

3.4 自动请求重传

- 停等ARQ链路的利用率
  - 可能产生严重的低效率
  - 如果帧的长度T_{DATA}很大而传播延时τ又小，停等ARQ可以有很高的链路利用率
  - 如果帧的长度T_{DATA}较小，而链路的传播延时τ又长（如卫星链路），链路利用率就变得很低
  - 因此，下面两种回退-N ARQ和选择重传ARQ对停等ARQ做了改进

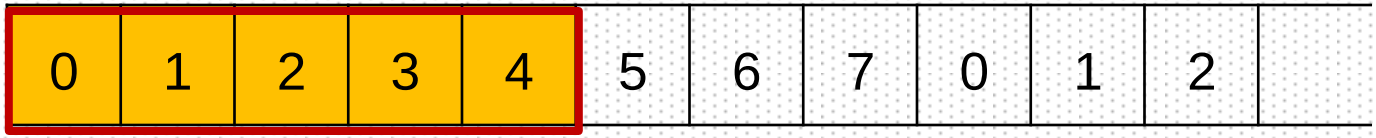
- 回退-N帧 ARQ（对停等ARQ的改进）
  - 基本思想：回退-N ARQ也使用滑动窗口机制，但$W_T>1$，发送方在每收到一个ACK之前不必等待，可以连续地发送窗口内的多个帧，如果这时收到接收方发回的ACK，还可以继续发送后续的帧，因此这种方式也称为连续ARQ

3.4 自动请求重传

- 回退-N帧 ARQ示例

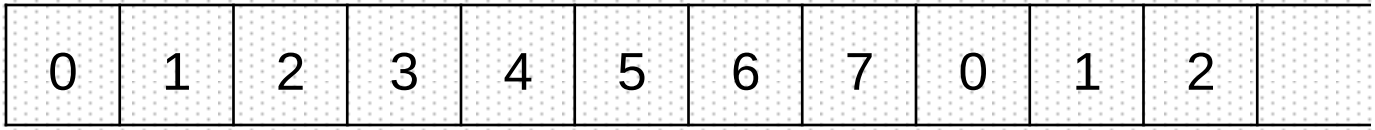
发送窗口W_T>1，接收窗口W_R=1

发送方



（a）初始状态，可发送0~4号帧

接收方



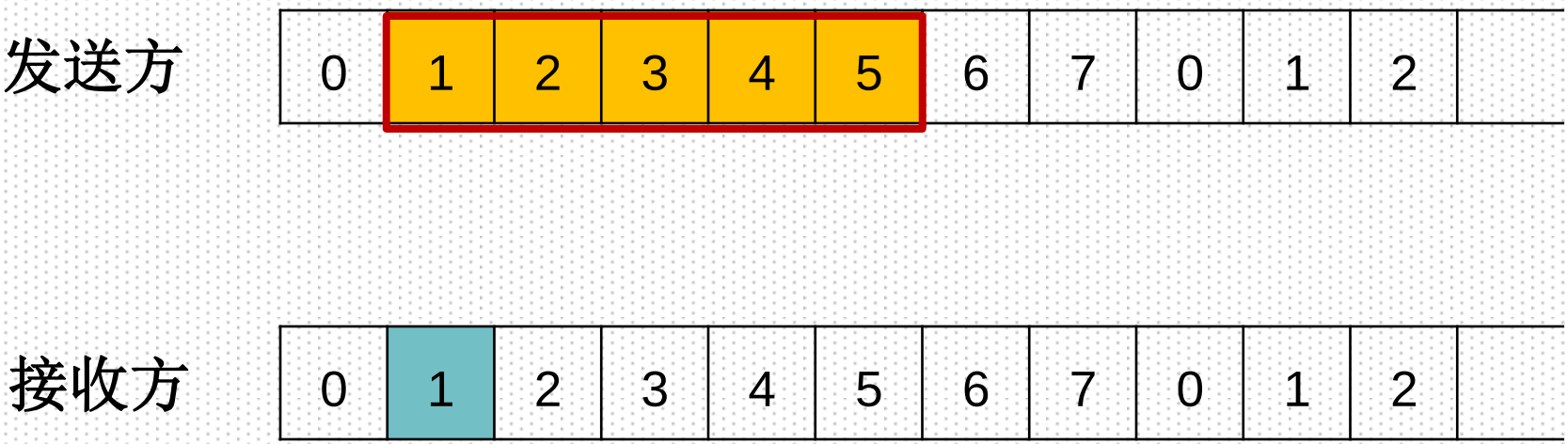
（a）初始状态，准备接收0号帧

3.4 自动请求重传

- 回退-N帧 ARQ示例

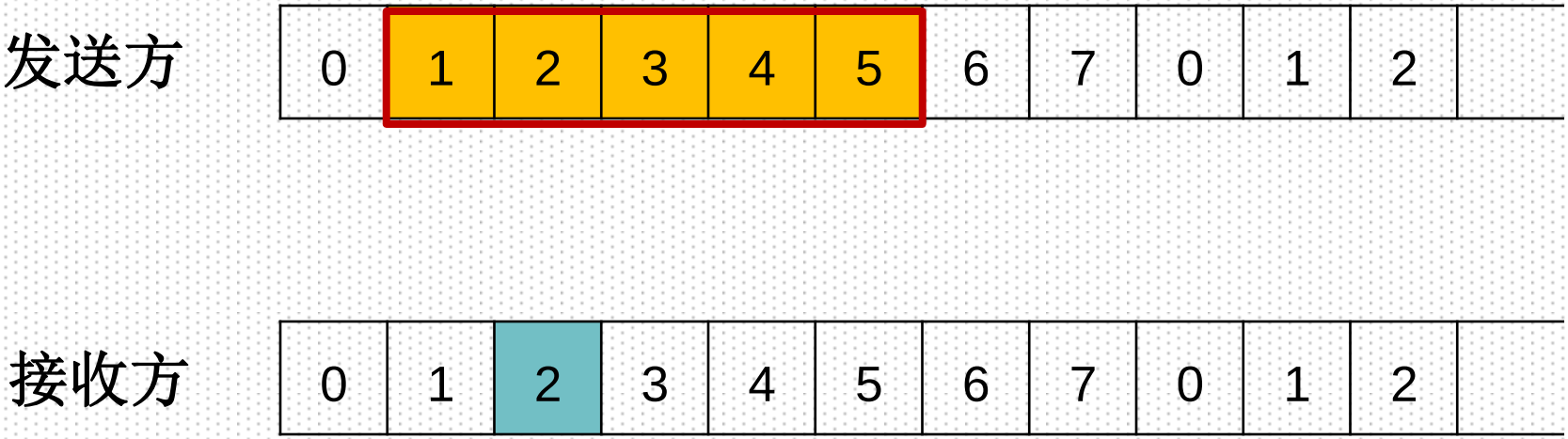
3.4 自动请求重传

- 回退-N帧 ARQ示例



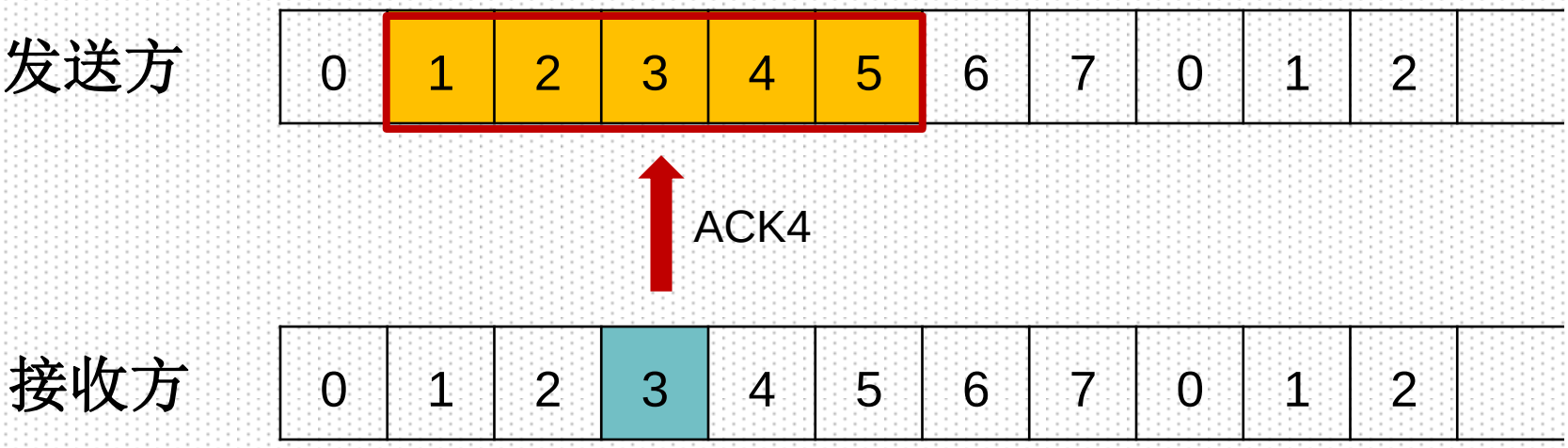
3.4 自动请求重传

- 回退-N帧 ARQ示例



3.4 自动请求重传

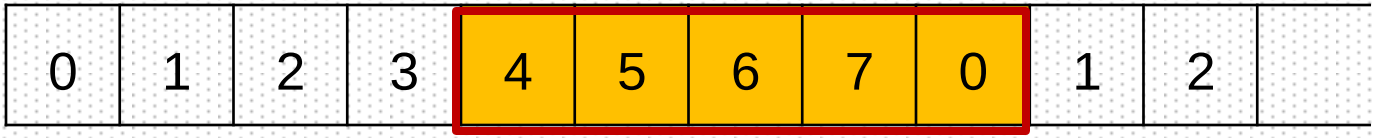
- 回退-N帧 ARQ示例



3.4 自动请求重传

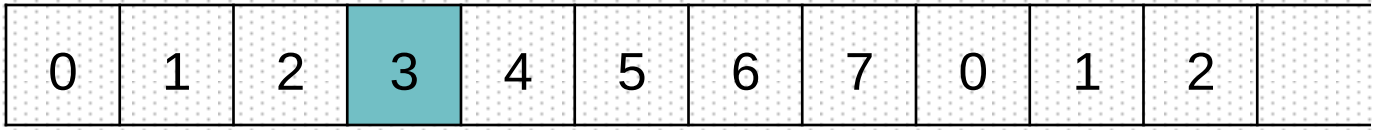
- 回退-N帧 ARQ示例

发送方



（c）向前滑动3个号，可发送4~7及0号帧

接收方



（c）可接收4号帧

- 回退-N帧 ARQ（对停等ARQ的改进）
  - （1）发送方的W_T>=1
    - 发送方按照发送窗口尺寸连续发送各个编号帧，每发送一个帧，窗口上限向前滑动一格，直至达到最大的发送窗口尺寸，然后停下来等待接收方的应答帧
    - 每当接收到一个应答帧，窗口下限向前滑动一格，发送方再按发送窗口尺寸发送后续的帧
    - 如果窗口下限指示的帧超时，没有接收到应答帧，则发送方需要重发窗口内自超时起的后续各个编号帧

- 回退-N帧 ARQ（对停等ARQ的改进）
  - （2）接收方的$W_R=1$
    - 接收方依次处理和校验接收到的各个数据帧
    - 如果帧的编号与接口窗口的序号一致，并且帧校验正确，则发送确认，同时接收窗口向前滑动一个窗口
    - 如果帧的编号与接收窗口的需要不一致，或者帧校验错误，则不发送确认或发送负确认，并且丢弃自出错帧起的后续各个编号帧

3.4 自动请求重传

- 回退-N帧 ARQ（对停等ARQ的改进）
  - 回退-N帧 ARQ也使用超时重传机制。对于发送的每一帧设置重传定时器，发送方发出一个帧之后启动该定时器。若因发送帧丢失、出现传输差错或ACK丢失使定时器超时仍未收到ACK，则要重传此帧，而且还必须重传此帧后面所有已发帧（不管这些帧是否有传输差错），这正是这种机制称为回退-N帧 ARQ的原因
  - 与停等ARQ相比，连续ARQ减少了等待时间，提高了传输的吞吐量和传输效率

- 回退-N帧 ARQ链路利用率
  - 可连续发送窗口内的多个数据帧，回退-N ARQ比停等ARQ提高了链路利用率
  - 如果在已发送的数据帧中，有一个前面的数据帧出错，那么其后的数据帧必须重传，浪费了信道资源，降低了链路利用率
  - 信道传输质量好、误码率很小时，回退-N ARQ协议链路利用率高，反之利用率降低

3.4 自动请求重传

- **回退-N帧 ARQ链路利用率**
  - 为了提高链路的利用率，对于比特长度大的链路，应采用大的W_T，而且W_T应该使连续发送的比特长度大于链路的往返比特长度，使得传输链路处于忙碌状态
  - 由于确认帧在传输过程中可能发生丢失，发送方超时后会重发帧，但会产生重复帧问题。因此，接收方必须通过接收窗口来验证编号帧编号的一致性，以排除重复帧

回退-N帧 ARQ的W_T最大是多少？

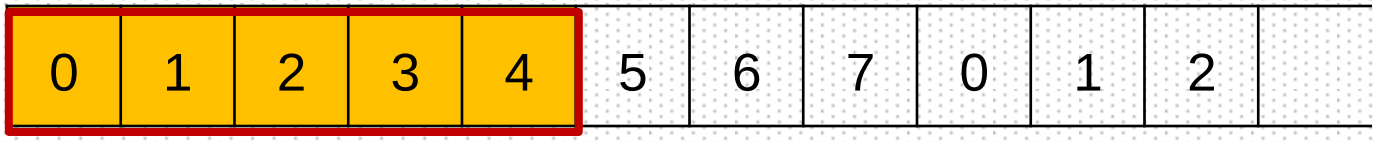
3.4 自动请求重传

- 选择重传ARQ（对回退-N ARQ的改进）
  - 在回退-N ARQ的基础上作了两点改进：
    - 接收窗口W_R>1，这样可保证接收和保存正确到达的失序的帧数据
    - 出现差错时只重传出错的帧，后续正确到达的帧不再需要重新传输，提高了信道的利用率

- 选择重传ARQ示例

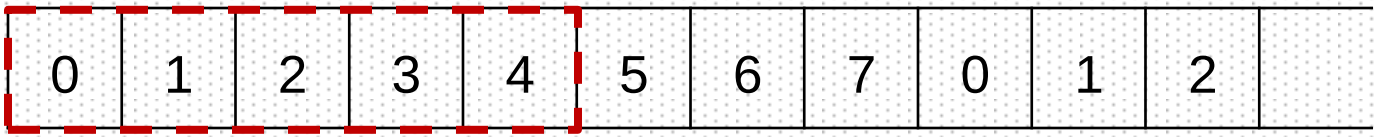
发送窗口W_T=5，接收窗口W_R=5

发送方



（a）初始状态，可发送0~4号帧

接收方



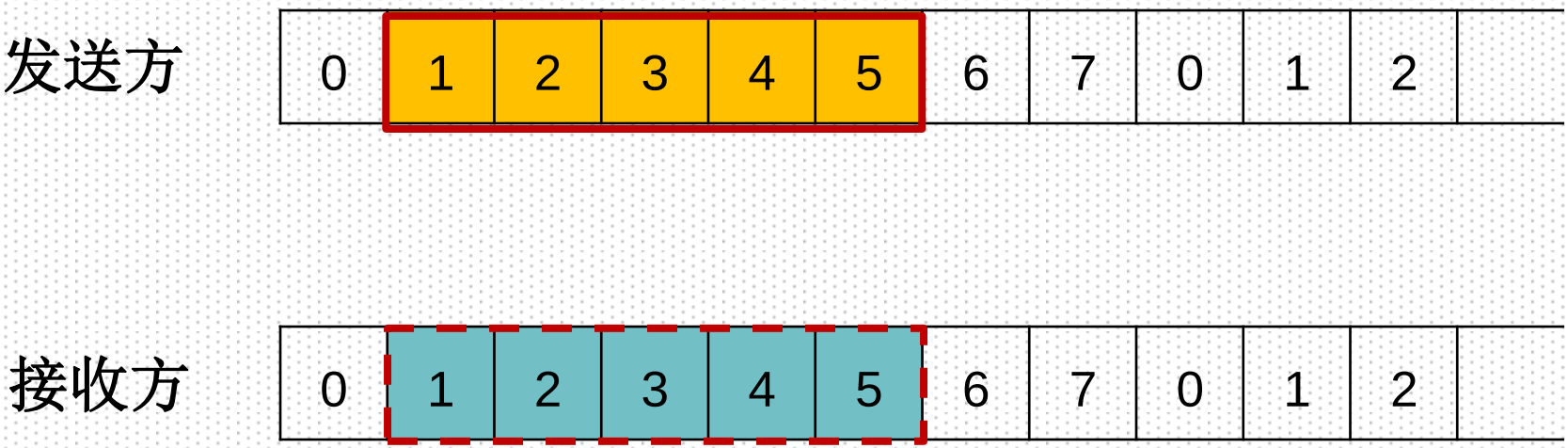
（a）初始状态，准备接收0~4号帧

- 选择重传ARQ示例

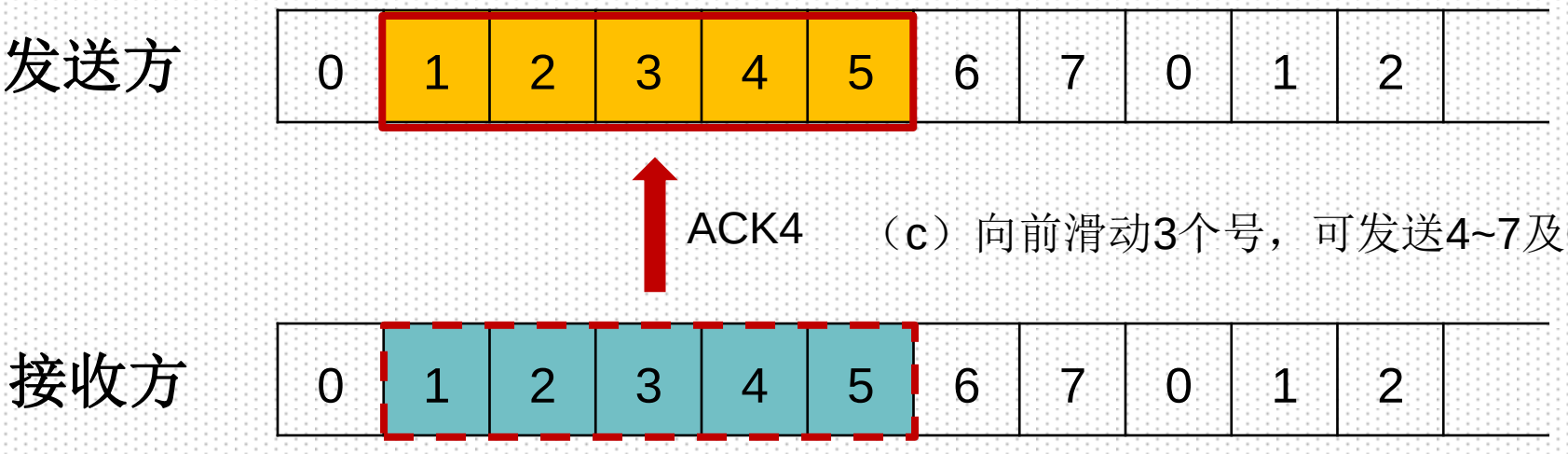

发送方
0
1
2
3
4
5
6
7
0
1
2
ACK1
（b）向前滑动一个号，可发送1~5号帧状态
接收方
0
1
2
3
4
5
6
7
0
1
2
（b）向前滑动一个号，可接收1~5号帧

3.4 自动请求重传

- 选择重传ARQ示例

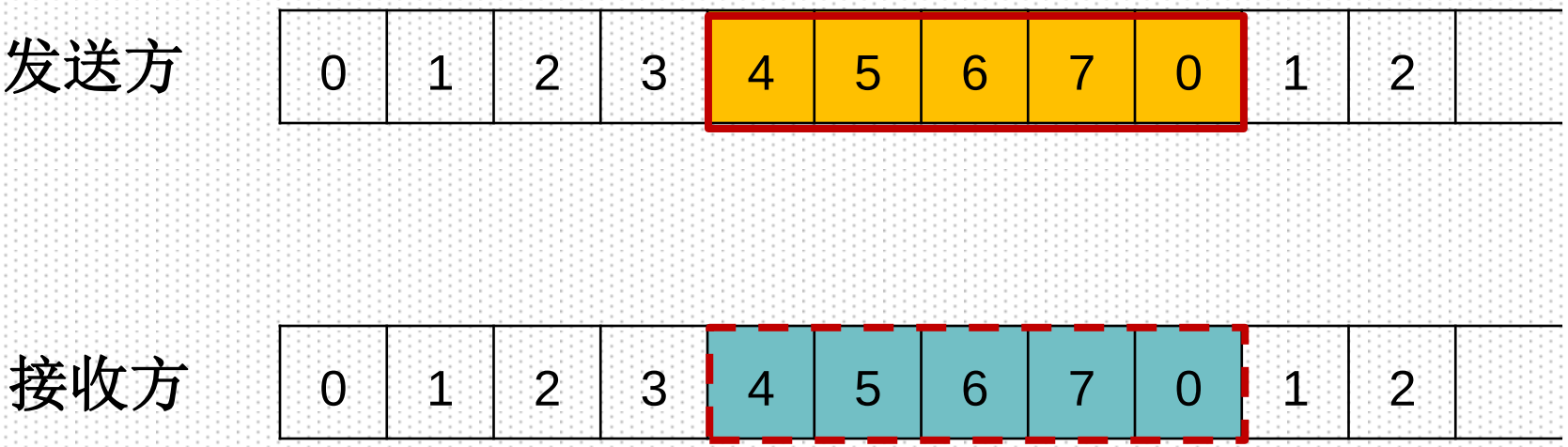


- 选择重传ARQ示例


发送方
0
1
2
3
4
5
6
7
0
1
2
ACK4
（c）向前滑动3个号，可发送4~7及0号帧
接收方
0
1
2
3
4
5
6
7
0
1
2
（c）向前滑动3个号，可接收4~7及0号帧

3.4 自动请求重传

- 选择重传ARQ示例



选择重传ARQ需要接收方设置一定容量的缓存空间

一般选择重传ARQ的$W_T=W_R$，最大是多少？

3.5 数据链路层协议实例

- 数据链路层协议实例
  - HDLC（High-level Data Link Control）
    - 高级数据链路控制
    - 曾有重要的影响和广泛的应用
  - PPP（Point-to-Point Protocol）
    - 点对点协议
    - 目前使用得最广泛的数据链路层协议

3.5 数据链路层协议实例

- HDLC协议（高级数据链路控制协议）
  - 最初来自70年代初，IBM大型机领域中使用的数据链路层协议SDLC（Synchronous Data Link Control，同步数据链路控制），IBM将SDLC提交给ANSI和ISO，希望成为美国标准和国际标准
  - ANSI对它修改，变成Advanced Data Communication Control Procedure，高级数据通信控制规程，ADCCP
  - ISO将它改为High-level Data Link Control，高级数据链路控制，HDLC
  - 面向比特的链路层规程

3.5 数据链路层协议实例

- HDLC基本概念
  - 为了能够适应不同配置、不同操作方式和不同传输距离的数据通信链路，HDLC定义了：
    - 三种类型的站
    - 两种链路配置
    - 三种数据传输方式

3.5 数据链路层协议实例

- HDLC中三种类型的站
  - 主站：发送命令帧、数据信息帧和应答帧，并负责控制链路
  - 从站（次站）：接收命令帧，向主站发送响应帧，并配合主站进行链路控制
  - 复合站（组合站）：同时具有主站和从站功能

3.5 数据链路层协议实例

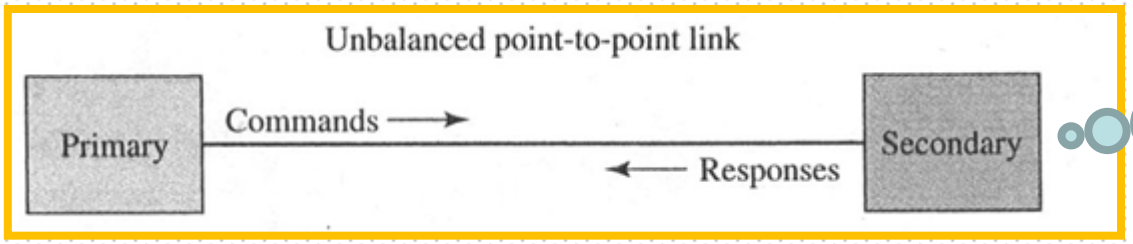
- HDLC中两种链路配置
  - 不平衡控制：适用于点对点、多点链路。由一个主站和多个从站组成，支持全双工或半双工传输
  - 平衡控制：适用于点对点链路。由两个复合站组成，支持全双工或半双工传输

3.5 数据链路层协议实例

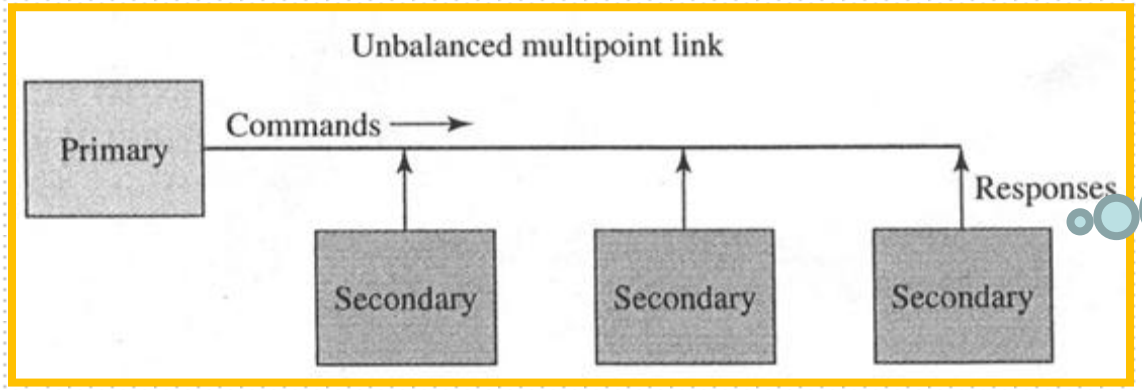
- HDLC中三种数据传输方式
  - 正常响应方式NRM：适用于不平衡配置链路。数据传输由主站发起，从站只能响应主站的轮询后才能发送数据
  - 异步响应方式ARM：适用于不平衡配置链路。从站主动启动数据传输，主站只负责链路管理
  - 异步平衡方式ABM：适用于平衡配置链路。任何一个复合站都可以发起数据传输

3.5 数据链路层协议实例

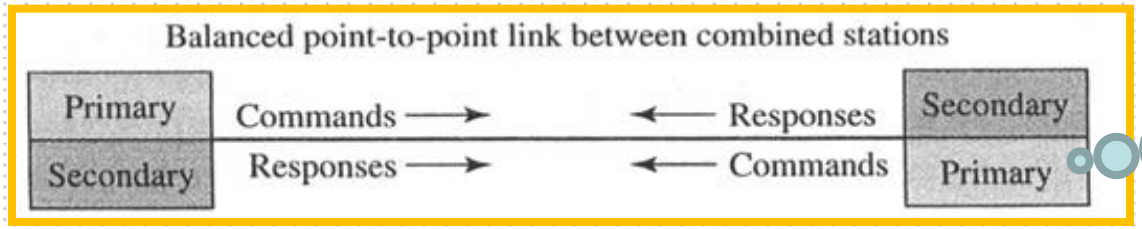
- HDLC工作方式



工作方式1：
不平衡配置电路
正常响应方式



工作方式2：
不平衡配置电路
异步响应方式

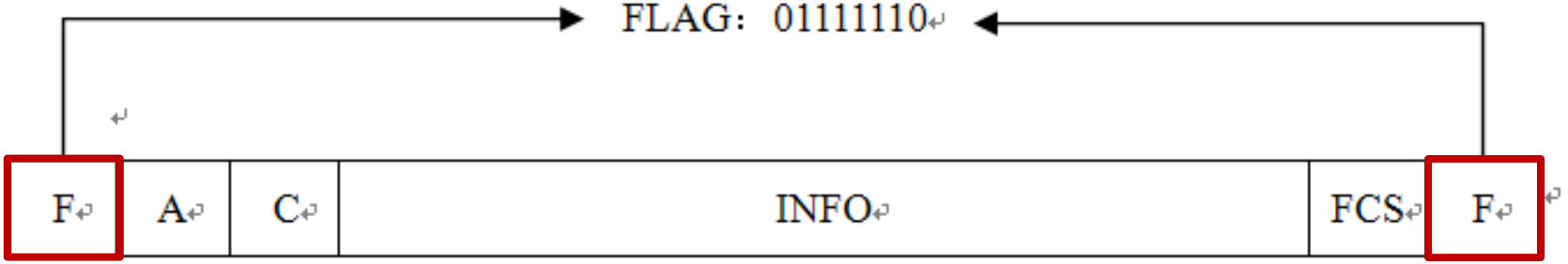


工作方式3：
平衡配置电路
异步平衡方式

3.5 数据链路层协议实例

- HDLC帧结构

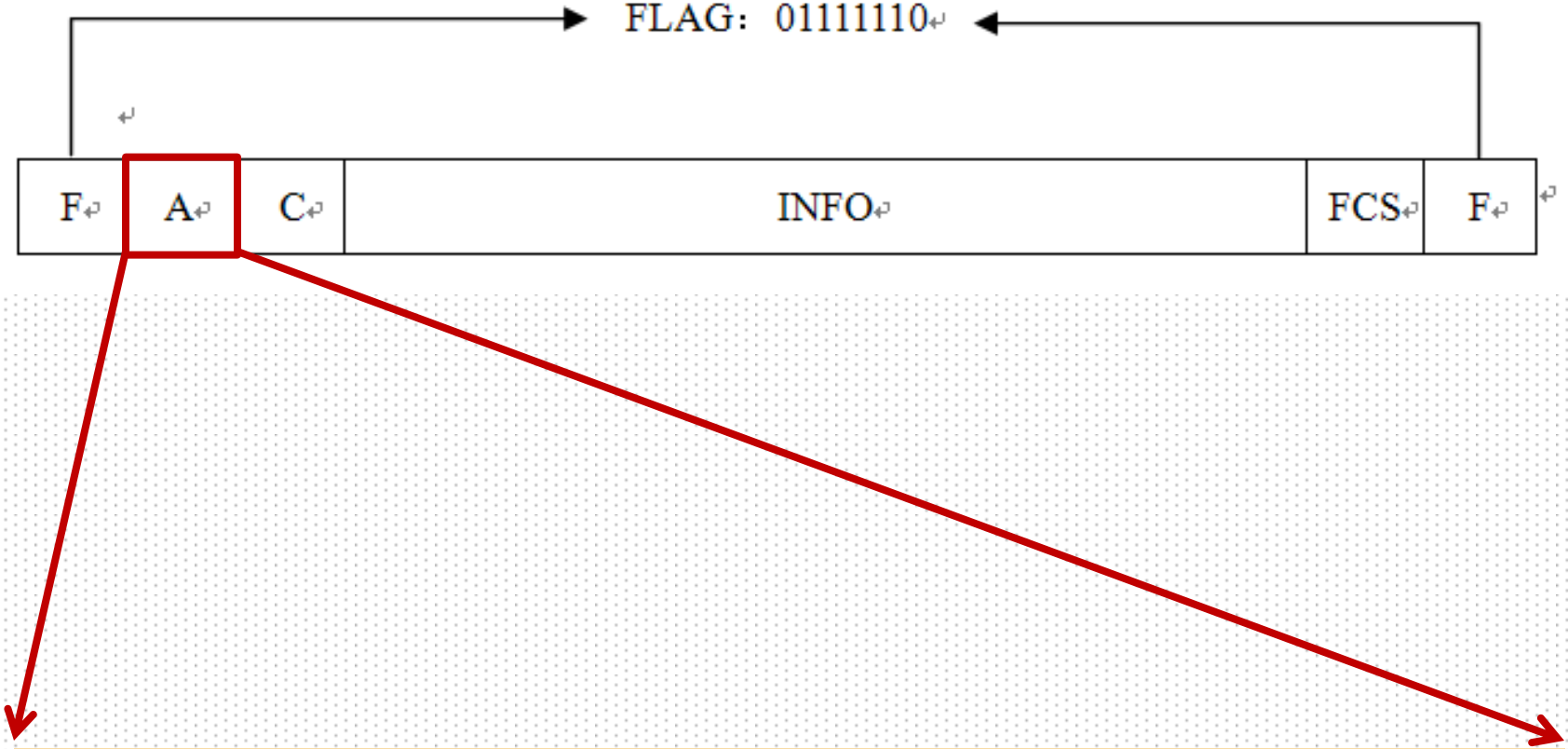
FLAG：帧标志
（用以确定帧的边界。既可以作前一帧的结束，也可以做后一帧的开始）



3.5 数据链路层协议实例

- HDLC帧结构

FLAG：帧标志
（用以确定帧的边界。既可以作前一帧的结束，也可以做后一帧的开始）

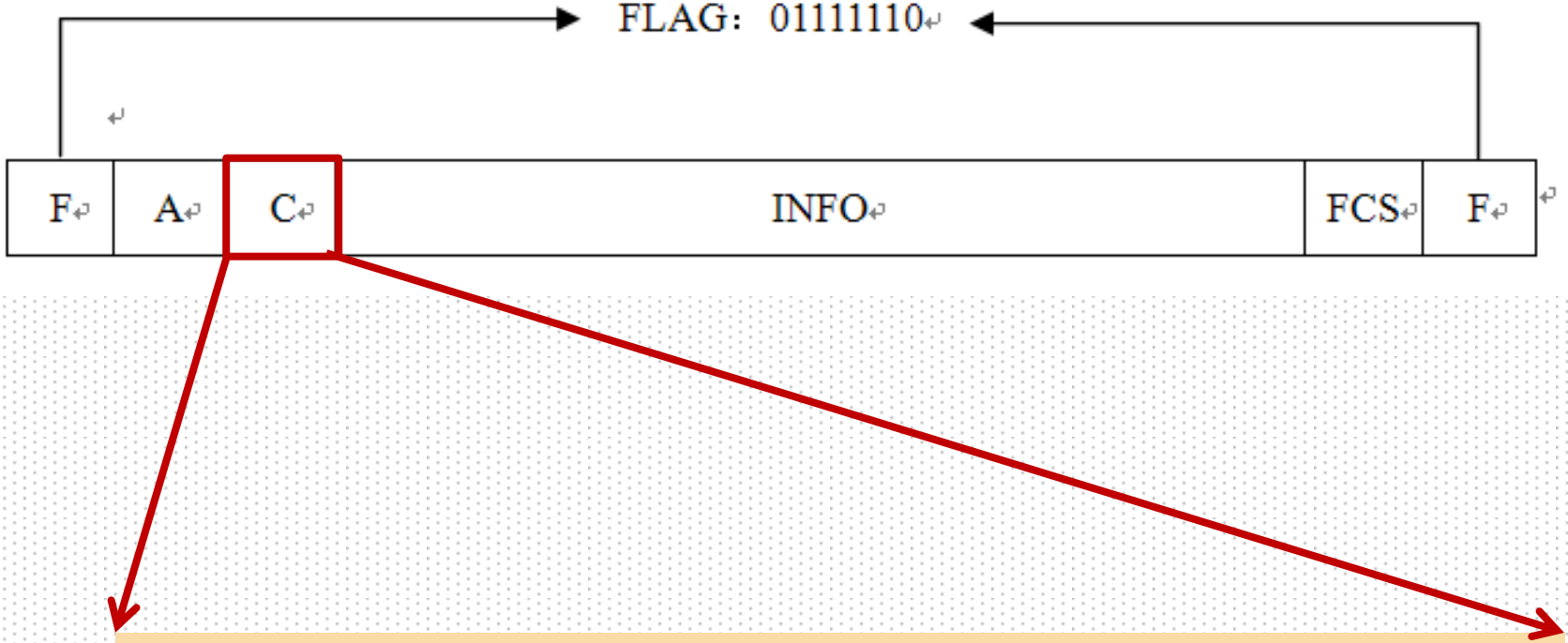


Address：8位，用来在存在多个终端的情况下标识一个终端

3.5 数据链路层协议实例

- HDLC帧结构

FLAG：帧标志
（用以确定帧的边界。既可以作前一帧的结束，也可以做后一帧的开始）



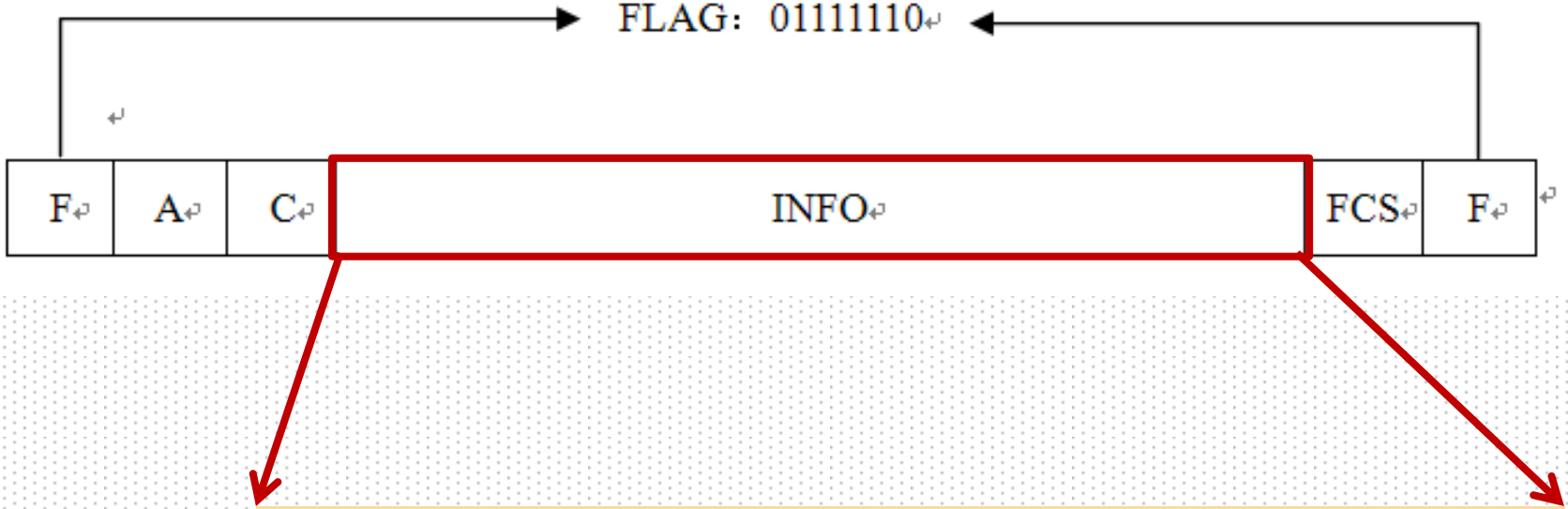
Control：8位，用来包含帧序列号以及协议信息等控制信息

Address：8位，用来在存在多个终端的情况下标识一个终端

3.5 数据链路层协议实例

- HDLC帧结构

FLAG：帧标志
（用以确定帧的边界。既可以作前一帧的结束，也可以做后一帧的开始）



INFOrmation：数据，0或者多位需要传输的信息

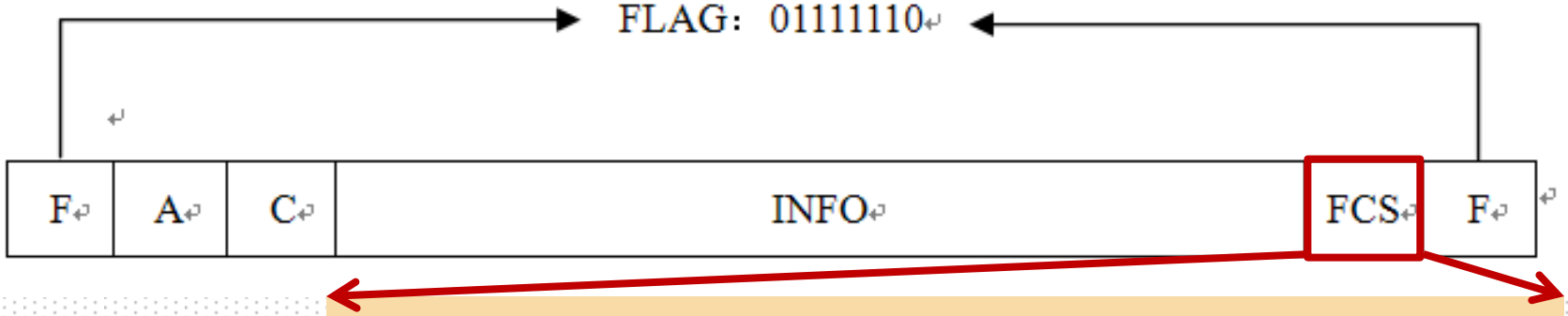
Control：8位，用来包含帧序列号以及协议信息等控制信息

Address：8位，用来在存在多个终端的情况下标识一个终端

3.5 数据链路层协议实例

- HDLC帧结构

FLAG：帧标志
（用以确定帧的边界。既可以作前一帧的结束，也可以做后一帧的开始）



FCS：Frame Check，16位CRC校验码，包括对两个Flag之间所有数据的校验

INFOrmation：数据，0或者多位需要传输的信息

Control：8位，用来包含帧序列号以及协议信息等控制信息

Address：8位，用来在存在多个终端的情况下标识一个终端

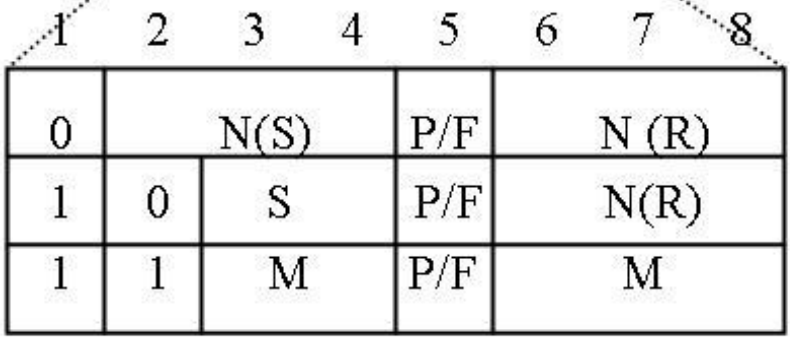
- HDLC帧结构
  - 地址字段A

  - 用来在存在多个终端的情况下标识一个终端
  - 若全为1的8位组（11111111）表示广播地址，表示所有从站都要接收

- HDLC帧结构
  - 控制字段C
  - HDLC定义了三种帧，根据控制字段格式区分
    - 信息帧（I帧）：承载要传送的数据，捎带流量控制和差错控制信号
    - 管理帧（S帧）：提供实现ARQ的控制信息，不使用捎带机制时控制传输过程
    - 无编号帧（U帧）：提供链路控制功能

3.5 数据链路层协议实例

- HDLC帧结构
  - 控制字段C
  - 前1-2位区分三种不通过格式的帧



信息帧：I 帧：

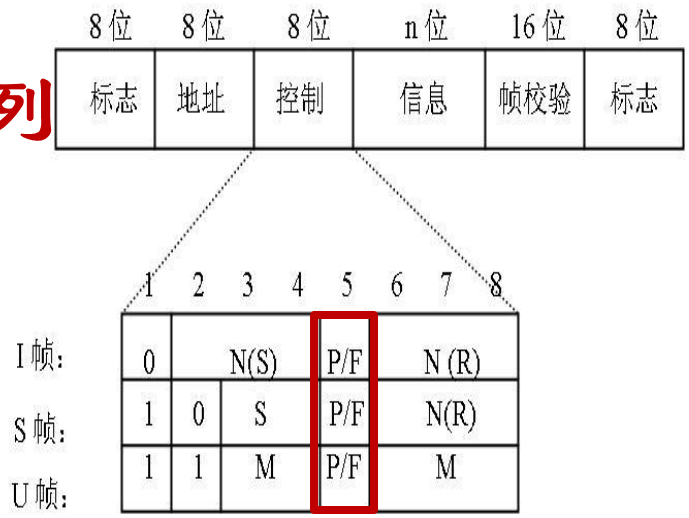
管理帧：S 帧：

无编号帧：U 帧：

N(S)：发送顺序号

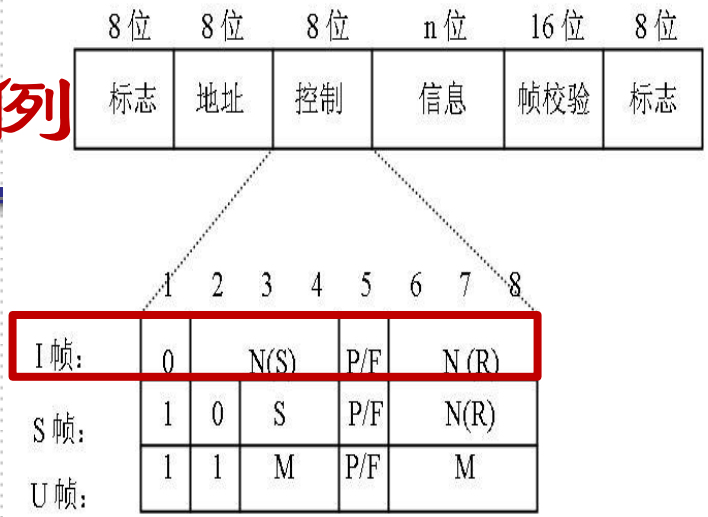
N(R)：接收顺序号

3.5 数据链路层协议实例



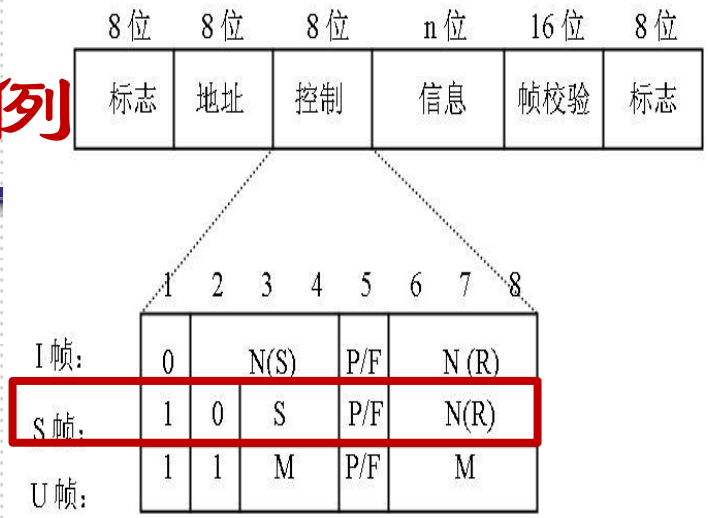
- HDLC帧结构-P/F位
  - P/F代表Poll/Final（查询/结束），当计算机询问一组终端时需要用到该位，用作P的时候，计算机请求终端发送数据。终端发送的所有帧，结束时最后一帧设置F
  - 有些协议中P/F位被用于强迫要求其他机器立即发送一个管理帧，而不是使用捎带确认技术

3.5 数据链路层协议实例



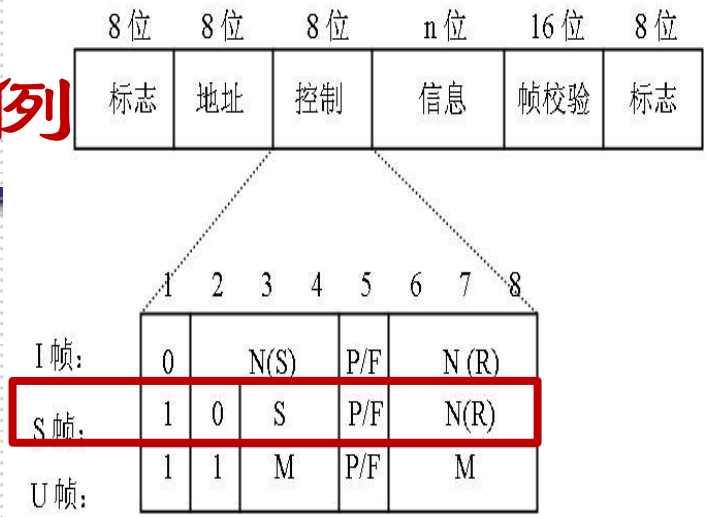
- HDLC帧结构-信息帧I帧
  - 用于提供面向连接的数据传输
  - 协议使用了滑动窗口协议，序列号的长度是3位，在任何时刻允许7个（2^3-1）未被确认的帧处于等待状态
    - **N(S)**：帧的编号
    - **N(R)**：捎带确认应答帧的编号
    - **P/F**位：**P/F**代表**Poll/Final**（查询/结束），当计算机询问一组终端的时候需要用到该位，用作**P**的时候，计算机请求终端发送数据。终端发送的所有帧，结束时最后一帧设置**F**

3.5 数据链路层协议实例



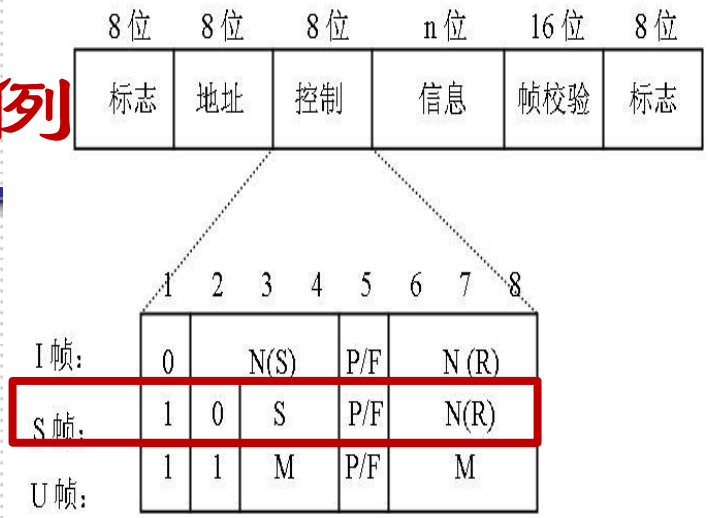
- HDLC帧结构-管理帧S帧
  - 用于双方数据链路层的管理和协商，即进行差错控制和流量控制（支持多少种？）
  - Type = 0 0（接收就绪，RR，Receive Ready）：肯定应答，接收第i帧。对N(R)-1帧的确认，对接收N(R)帧的应答。N(R)：期望收到的下一帧

3.5 数据链路层协议实例



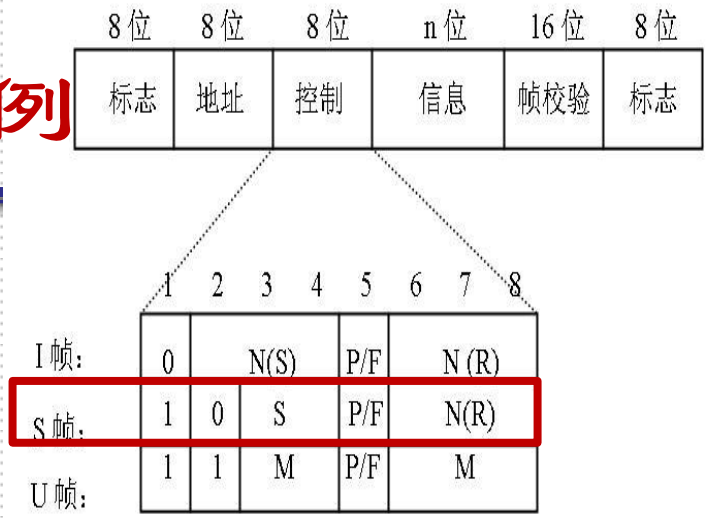
- HDLC帧结构-管理帧S帧
  - 用于双方数据链路层的管理和协商，即进行差错控制和流量控制（支持多少种？）
  - Type = 0 0（接收就绪，RR，Receive Ready）
  - Type = 0 1（接收未就绪，RNR，Receive Not Ready）：肯定应答，不能继续接收。对N(R)帧之前的帧确认，拒绝进一步接收后续帧

3.5 数据链路层协议实例



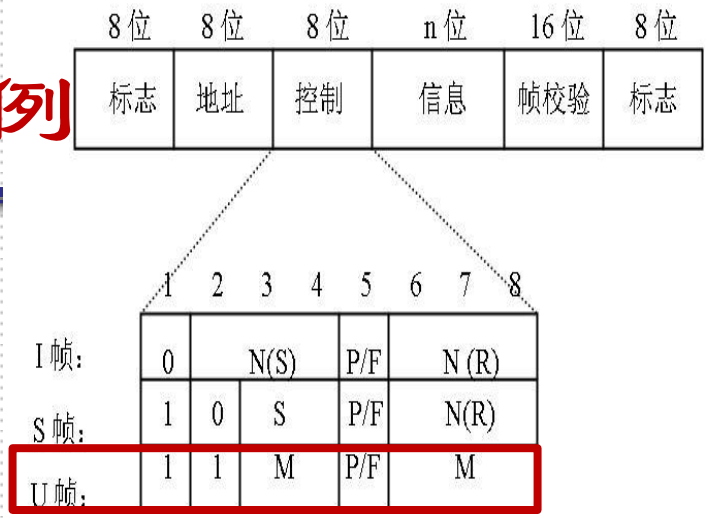
- HDLC帧结构-管理帧S帧
  - 用于双方数据链路层的管理和协商，即进行差错控制和流量控制（支持多少种？）
  - Type = 0 0（接收就绪，RR，Receive Ready）
  - Type = 0 1（接收未就绪，RNR，Receive Not Ready）
  - Type = 1 0（拒绝，REJ，Reject）：否定应答，后退N帧重发。拒绝接收，要求重发N(R)帧及后续帧。N(R)：指明没有被接收到的帧序号，需从该帧向后重发

3.5 数据链路层协议实例



- HDLC帧结构-管理帧S帧
  - 用于双方数据链路层的管理和协商，即进行差错控制和流量控制（支持多少种？）
  - Type = 0 0（接收就绪，RR，Receive Ready）
  - Type = 0 1（接收未就绪，RNR，Receive Not Ready）
  - Type = 1 0（拒绝，REJ，Reject）
  - Type = 1 1（选择性拒绝，SREJ，Selective Reject）：否定应答，选择重发。N(R)帧必须重发

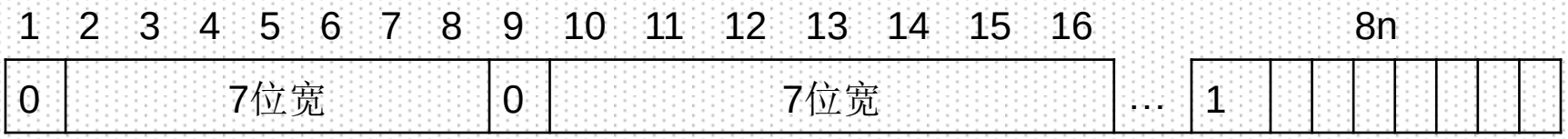
3.5 数据链路层协议实例



- HDLC帧结构－无编号帧U帧
  - 用于链路控制
  - 其控制功能可划分为以下几个子类：
    （1）设置数据传输方式的命令和响应帧
    （2）传输信息的命令和响应帧
    （3）用于链路恢复的命令和响应帧
    （4）其他命令和响应帧

- HDLC帧结构
  - U帧类型
    - SNRM：置正常响应方式
    - SARM：置异步响应方式
    - SABM：置异步平衡方式
    - UA：无编号应答——>对置数据传输方式的肯定应答
    - SNRME：置扩展的正常响应方式
    - SARME：置扩展的异步响应方式
    - SABME：置扩展的异步平衡方式
    - SIM：置初始化命令，使接收该命令的从接收站启动建立链路的过程
    - UI：交换控制信息
    - DISC：拆除逻辑连接
    - 。。。。。。。

- HDLC地址和控制字段的扩展
  - 地址字段和控制字段可按8位位组倍数进行扩展
  - 扩展地址：如果双方事先约定地址字段是可以扩展的，则地址字节的第1位为0时，表示它后面的字节仍为地址字节，依次类推，直到出现一个第1位为1的字节为止，该字节就是地址字段的最后一个字节



3.5 数据链路层协议实例

- HDLC地址和控制字段的扩展
  - 控制字段扩展：控制字段可以扩展为两个字节，扩展后的控制字段主要增加了N(S)和N(R)的长度，即由原来的3位增加到7位，序号的模数由原来的8增加到128。控制字段的扩展是通过相应的U帧来设置的

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
I帧	0	N(S)							P /F	N(R)						
S帧	1	0	SS		0	0	0	0	P /F	N(R)						

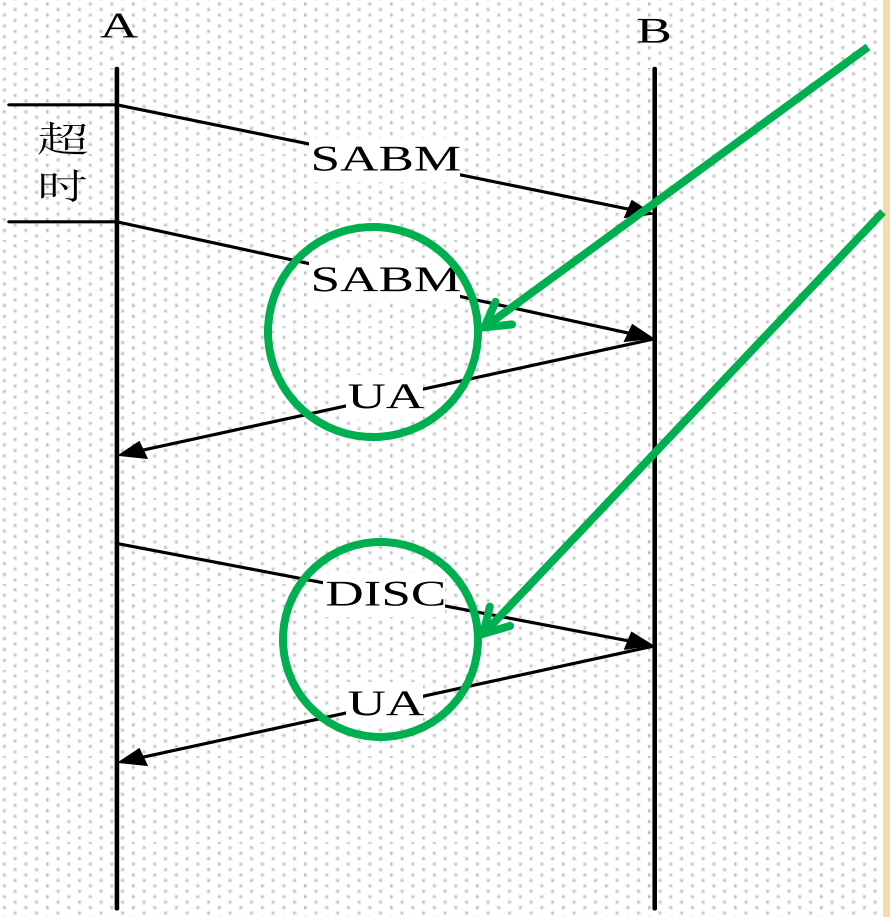
- HDLC帧结构
  - 信息字段INFO
    - 这个字段可含有表示用户数据的任何比特序列，其长度可变，往往具体实现时限定了最大帧长
  - 帧校验字段FCS
    - 含有除了标志字段外的所有其他字段的校验序列。通常使用16bit的CRC-CCITT标准产生校验序列，有时也使用CRC-32产生32位的校验序列

3.5 数据链路层协议实例

- HDLC的操作过程实例
  - I表示信息帧。I后面的两个数字分别表示信息帧中的N(S)和N(R)值
  - I21表示信息帧的发送顺序号是N(S)=2，接收顺序号N(R)=1，意味着该帧是发送站发出的第2帧，并捎带应答已接收了对方站的第0帧，期望接收的下一帧是第1帧
  - 管理帧和无编号帧都直接给出帧名字，管理帧中的数字表示帧中的N(R)值
  - P和F表示该帧中的P/F位置1，没有P和F表示这一位置0

3.5 数据链路层协议实例

- 实例1—数据链路的建立和拆除



双方应答建立逻辑链路

双方应答拆除逻辑链路

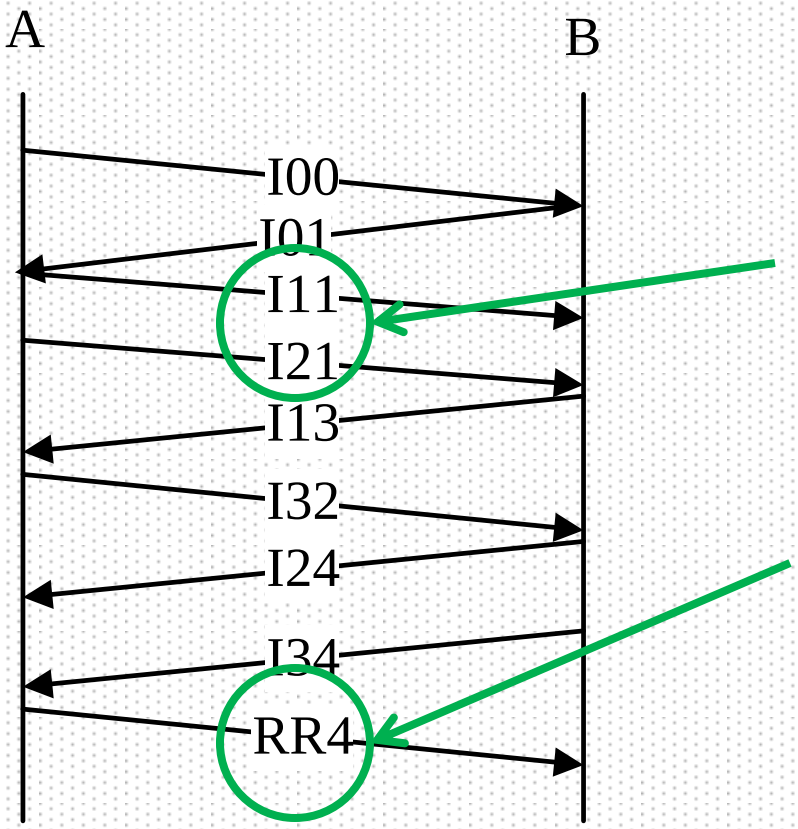
SABM：置异步平衡方式

UA：无编号应答

DISC：拆除逻辑连接

实际使用中，可能出现链路不能建立的情况，B站以DM（非连接方式，从站处于逻辑上断开的状态）响应A站，A站放弃建立连接

- 实例2-数据交换（全双工传输方式）

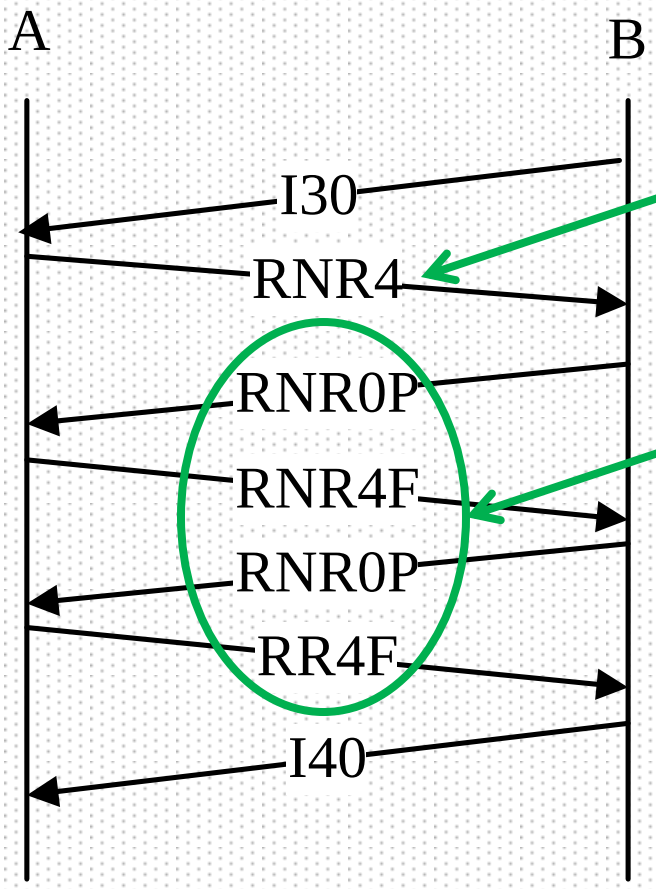


当一个站连续发送了若干帧而没有收到对方发来的信息帧时，N(R)字段只能简单地重复

可通过累计应答全部进行应答响应

RR：接收就绪

- 实例3—接收站忙



可能是接收站数据链路层缓冲区发生溢出；上层实体来不及处理接收的数据

发送站B每隔一段时间以P位置1的RNR命令询问接收站A

RR：接收就绪

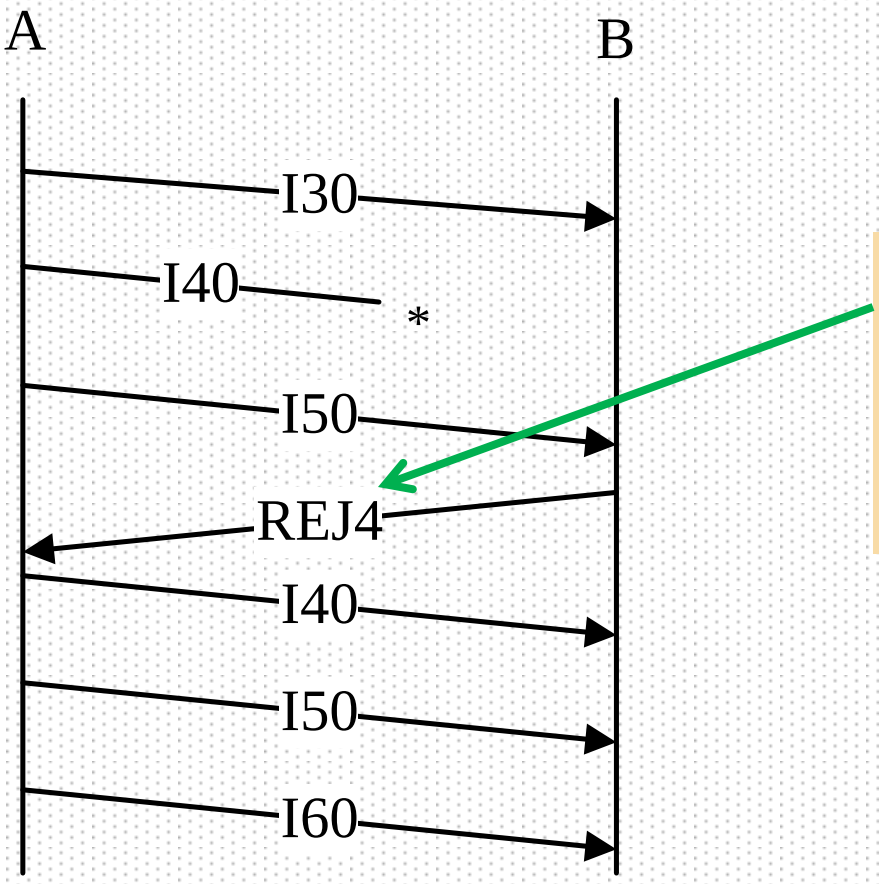
RNR：接收未就绪

P：请求终端发送数据

F：终端发送的所有帧结束

3.5 数据链路层协议实例

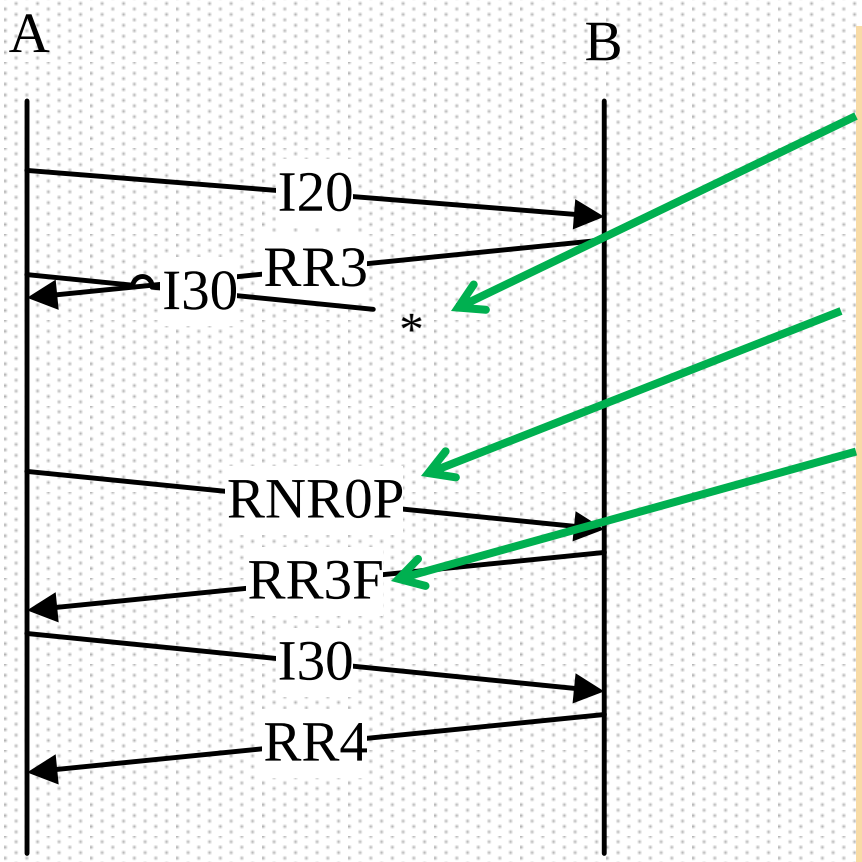
- 实例4–后退重发



I40发送出错，B站检出后要求A站回退重发

REJ：拒绝

- 实例5-超时重发



I30出错，B站检测到后直接丢弃

A站超时后询问B站状态

B站响应希望从3号帧重发

RR：接收就绪

RNR：接收未就绪

P：请求终端发送数据

F：终端发送的所有帧结束

3.5 数据链路层协议实例

西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- PPP协议
  - 对于点对点的链路，点对点协议PPP协议是目前使用得最广泛的数据链路层协议。用户接入因特网有多种途径，如通过电话线拨号入网或各种宽带入网，但不管怎样，总是要通过某个因特网服务提供者ISP才能接入到因特网。从用户计算机到ISP的链路所使用的数据链路层协议就是PPP协议

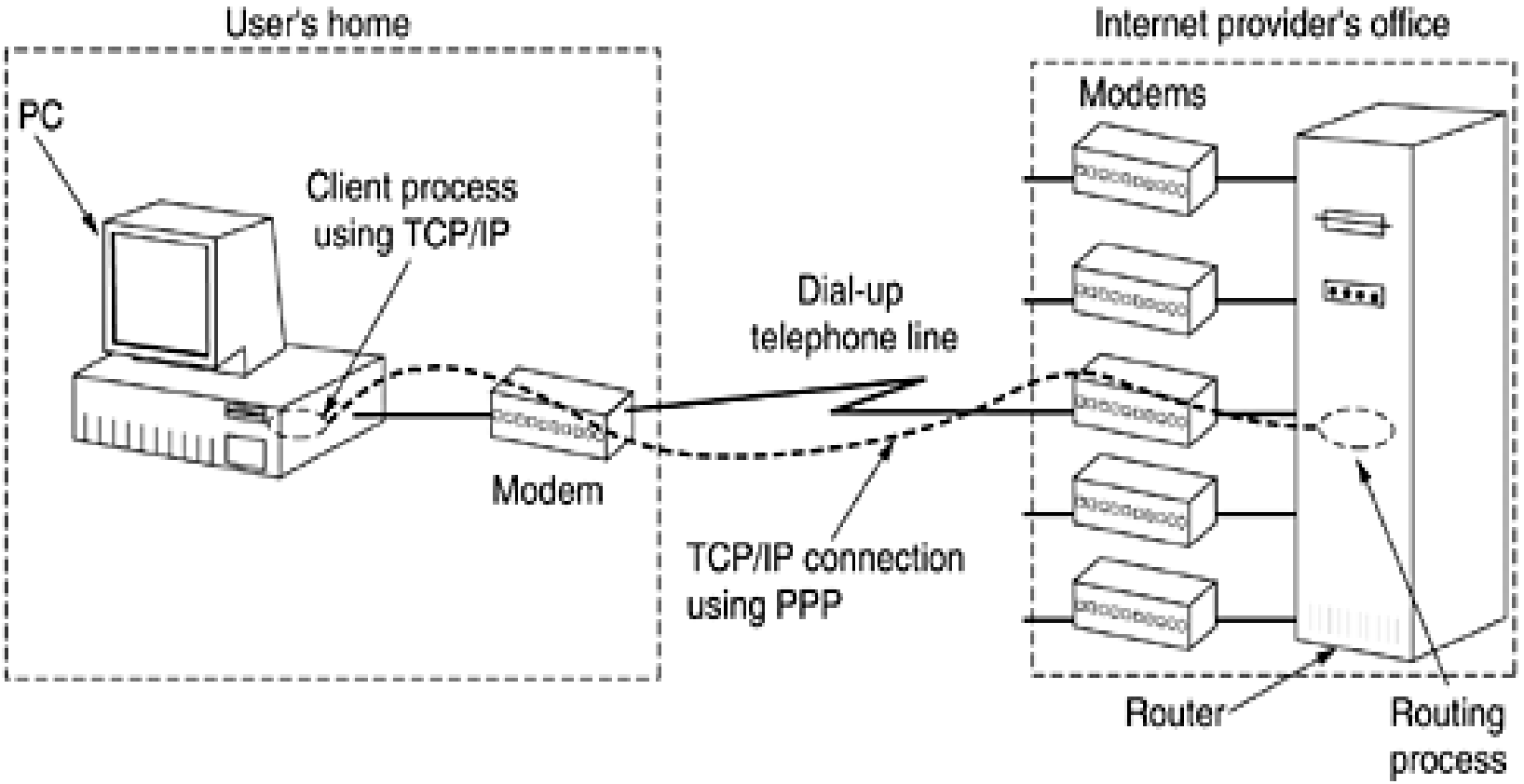
3.5 数据链路层协议实例

西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- PPP协议
  - 为在同等单元之间传输数据包这样的简单链路设计的链路层协议。设计的目的主要是用来通过拨号或专线方式建立点对点连接发送数据
  - 在Internet中使用的最普遍的数据链路层协议，称为PPP（Point-to-Point Protocol）

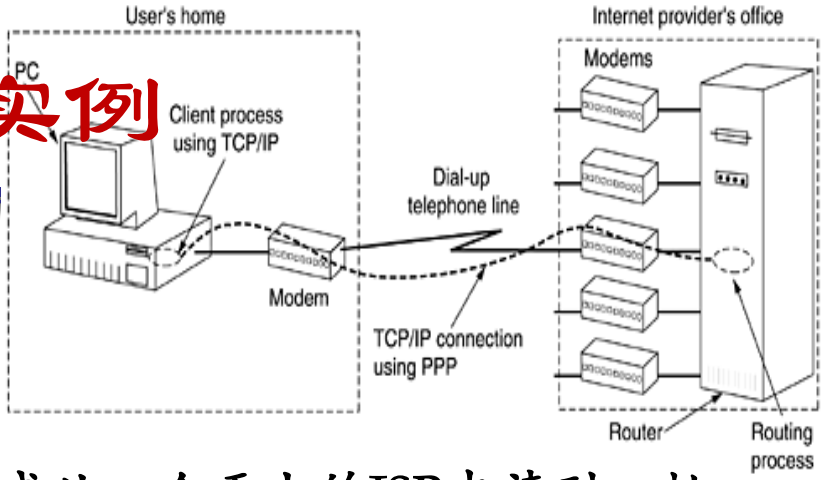
3.5 数据链路层协议实例

- PPP协议



家用计算机使用调制解调器连接到ISP示意图

3.5 数据链路层协议实例



- PPP协议
  - 每一个ISP都已从因特网的管理机构或从一个更大的ISP申请到一批IP地址。ISP还有与因特网通过高速通信专线相连的路由器。大的ISP拥有属于自己通信线路，小的ISP则向电信公司租用通信线路
  - 用户在某一个ISP缴费登记后(有的ISP是出售上网卡)，就可用自己的计算机通过调制解调器由电话线接入到该ISP。用户接通ISP后，ISP就分配给该用户一个临时的IP地址(在网络层中详细讨论)
  - 用户计算机获得临时的IP地址后，就成为连接在因特网上的主机，因而就可使用因特网所提供的各种服务
  - 当用户结束通信并断开连接后，ISP就把刚才分配给该用户的IP地址收回，以便再分配给后面拨号入网的其他用户使用

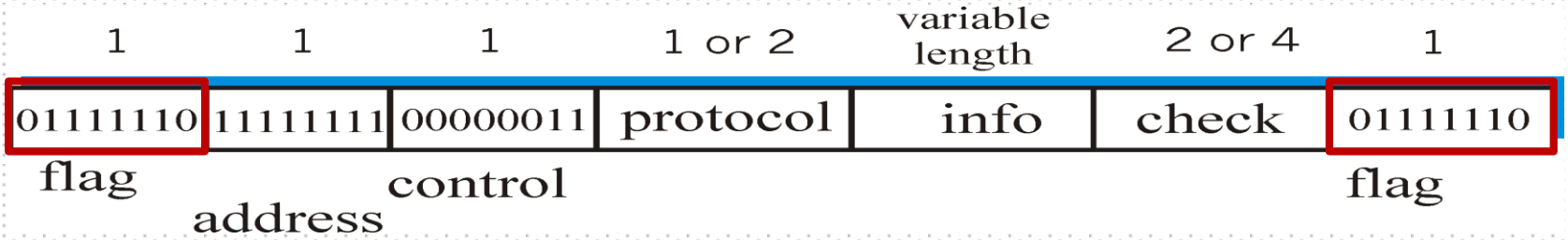
3.5 数据链路层协议实例

- PPP协议组成主要包括两部分
  - 链路控制协议LCP（Link Control Protocol）
    - 可使用多种物理层服务：**Modem**，**HDLC**串线，**SDH/SONET**（光纤网传输标准）等
  - 网络控制协议NCP（Network Control Protocol）
    - 可支持多种网络层协议，如**IP**、**OSI**网络层和**Netware**的网络层**IPX**等
  - 帧格式与HDLC相似，区别在于PPP是面向字符的，采用字符填充技术（HDLC使用比特填充技术）

3.5 数据链路层协议实例

- PPP帧结构

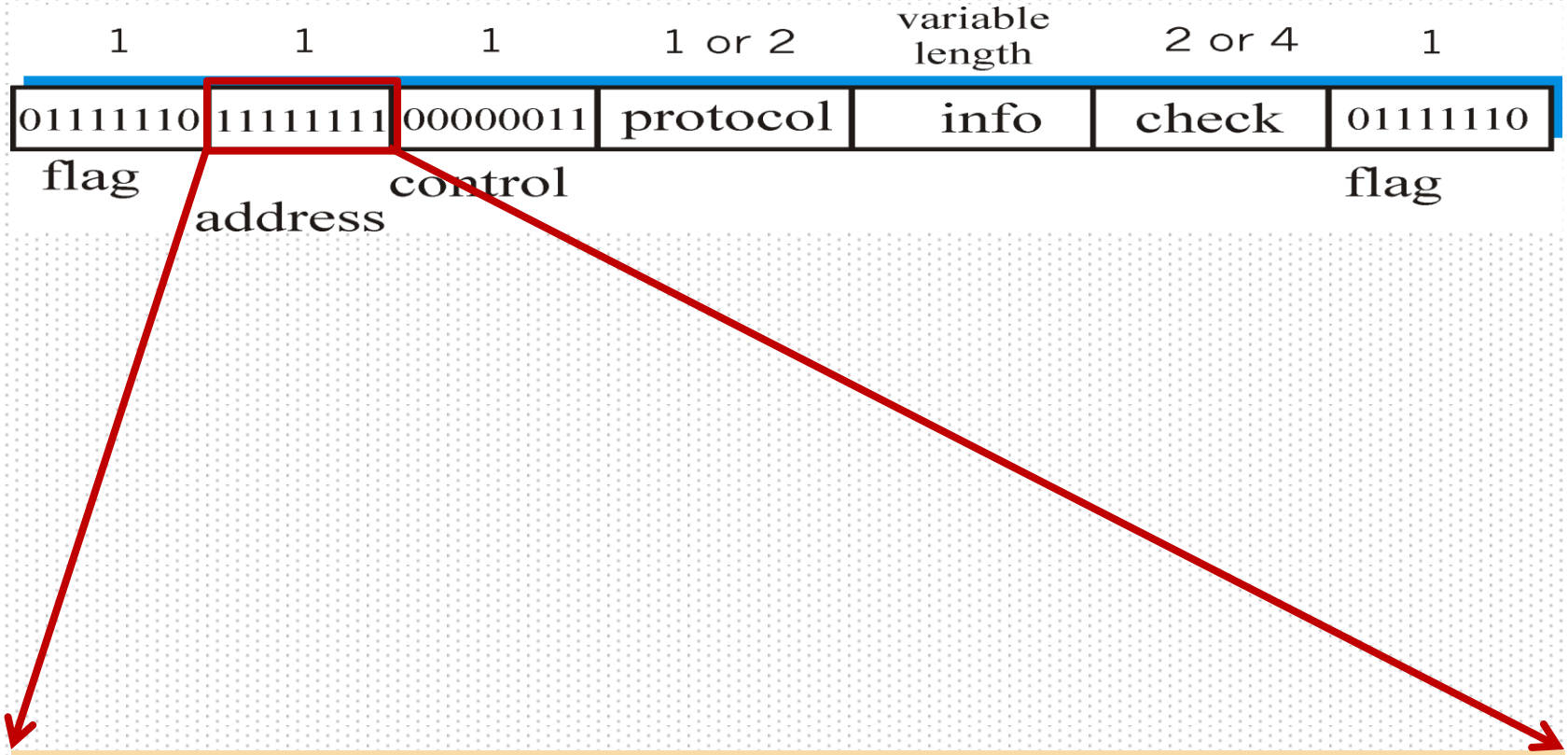
帧界标识符F：01111110，与HDLC相同



3.5 数据链路层协议实例

- PPP帧结构

帧界标识符F：01111110，与HDLC相同

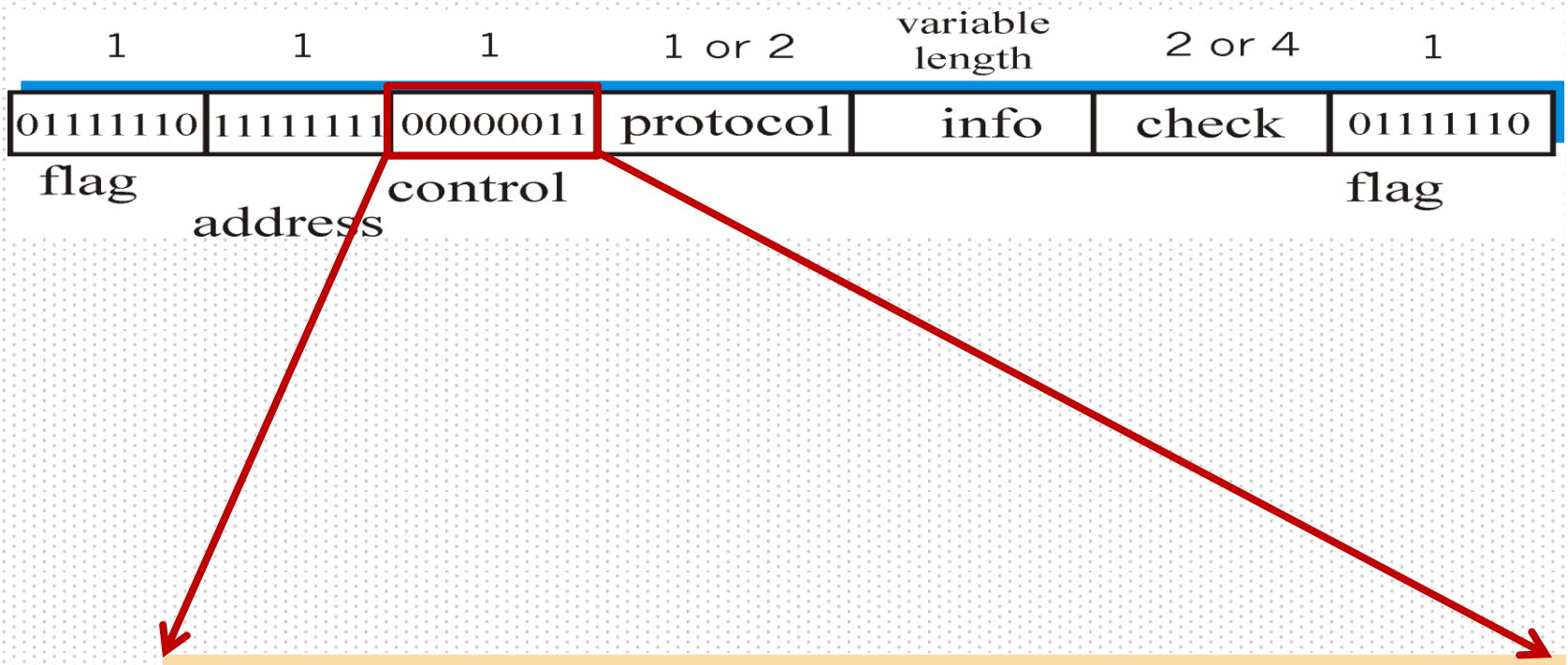


地址字段A：11111111表示广播字段，表示所有的站都可以接收该帧。用于点对点链路，实际上不需要数据链路地址字段。默认情况下，PPP并不采用序列号和确认来实现可靠传输。

3.5 数据链路层协议实例

- PPP帧结构

帧界标识符F：01111110，与HDLC相同



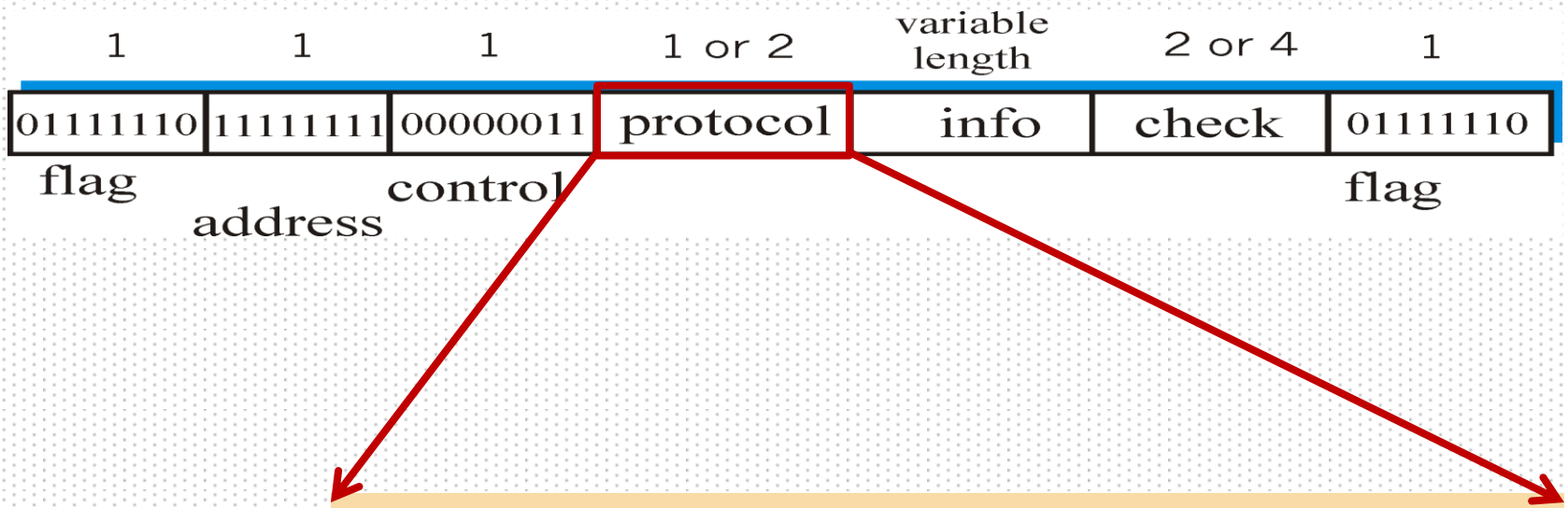
控制字段C：控制域默认00000011。没有实际的意义

地址字段A：11111111表示广播字段，表示所有的站都可以接收该帧。用于点对点链路，实际上不需要数据链路地址字段。默认情况下，PPP并不采用序列号和确认来实现可靠传输。

3.5 数据链路层协议实例

- PPP帧结构

帧界标识符F：01111110，与HDLC相同



协议字段P：HDLC协议中所没有的。是用来说明数据部分封装的是哪类协议的分组。如LCP、NCP、IP或IPX等

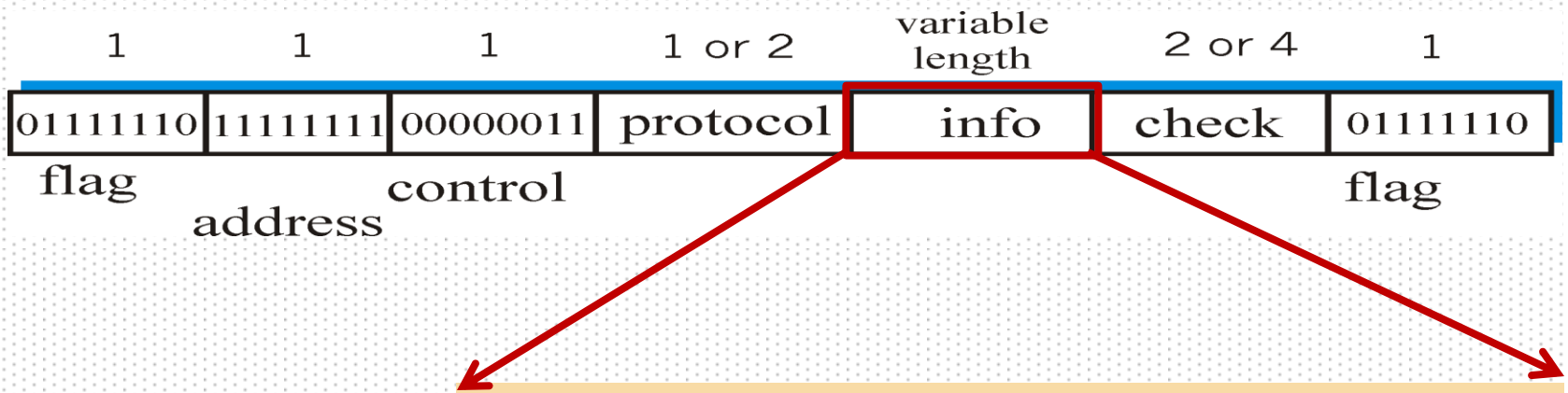
控制字段C：控制域默认00000011。没有实际的意义

地址字段A：11111111表示广播字段，表示所有的站都可以接收该帧。用于点对点链路，实际上不需要数据链路地址字段。默认情况下，PPP并不采用序列号和确认来实现可靠传输。

3.5 数据链路层协议实例

- PPP帧结构

帧界标识符F：01111110，与HDLC相同



信息字段INFO：数据长度可变

协议字段P：HDLC协议中所没有的。是用来说明数据部分封装的是哪类协议的分组。如LCP、NCP、IP或IPX等

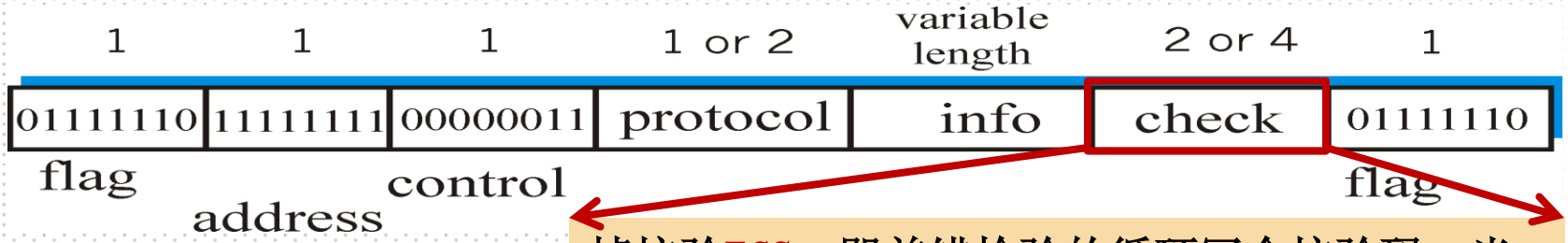
控制字段C：控制域默认00000011。没有实际的意义

地址字段A：11111111表示广播字段，表示所有的站都可以接收该帧。用于点对点链路，实际上不需要数据链路地址字段。默认情况下，PPP并不采用序列号和确认来实现可靠传输

3.5 数据链路层协议实例

- PPP帧结构

帧界标识符F：01111110，与HDLC相同



帧校验FCS：即差错检验的循环冗余校验码。当FCS字段检测到有差错时，便丢弃该帧

信息字段INFO：数据长度可变

协议字段P：HDLC协议中所没有的。是用来说明数据部分封装的是哪类协议的分组。如LCP、NCP、IP或IPX等

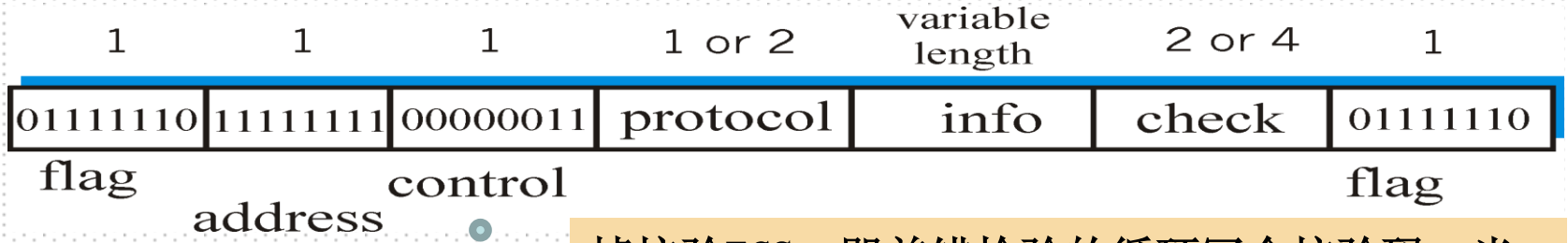
控制字段C：控制域默认00000011。没有实际的意义

地址字段A：11111111表示广播字段，表示所有的站都可以接收该帧。用于点对点链路，实际上不需要数据链路地址字段。默认情况下，PPP并不采用序列号和确认来实现可靠传输。

3.5 数据链路层协议实例

- PPP帧结构

帧界标识符F：01111110，与HDLC相同



帧校验FCS：即差错检验的循环冗余校验码。当FCS字段检测到有差错时，变丢弃该帧

PPP不进行差错控制，因此提供的是不可靠的传输服务

信息字段INFO：数据长度可变

协议字段P：HDLC协议中所没有的。是用来说明数据部分封装的是哪类协议的分组。如LCP、NCP、IP或IPX等

控制字段C：控制域默认00000011。没有实际的意义

地址字段A：11111111表示广播字段，表示所有的站都可以接收该帧。用于点对点链路，实际上不需要数据链路地址字段。默认情况下，PPP并不采用序列号和确认来实现可靠传输。

3.5 数据链路层协议实例

- PPP身份认证
  - 是一个重要的安全措施。包括两种机制：

    （1）口令认证协议（PAP）：简单的明文认证方式

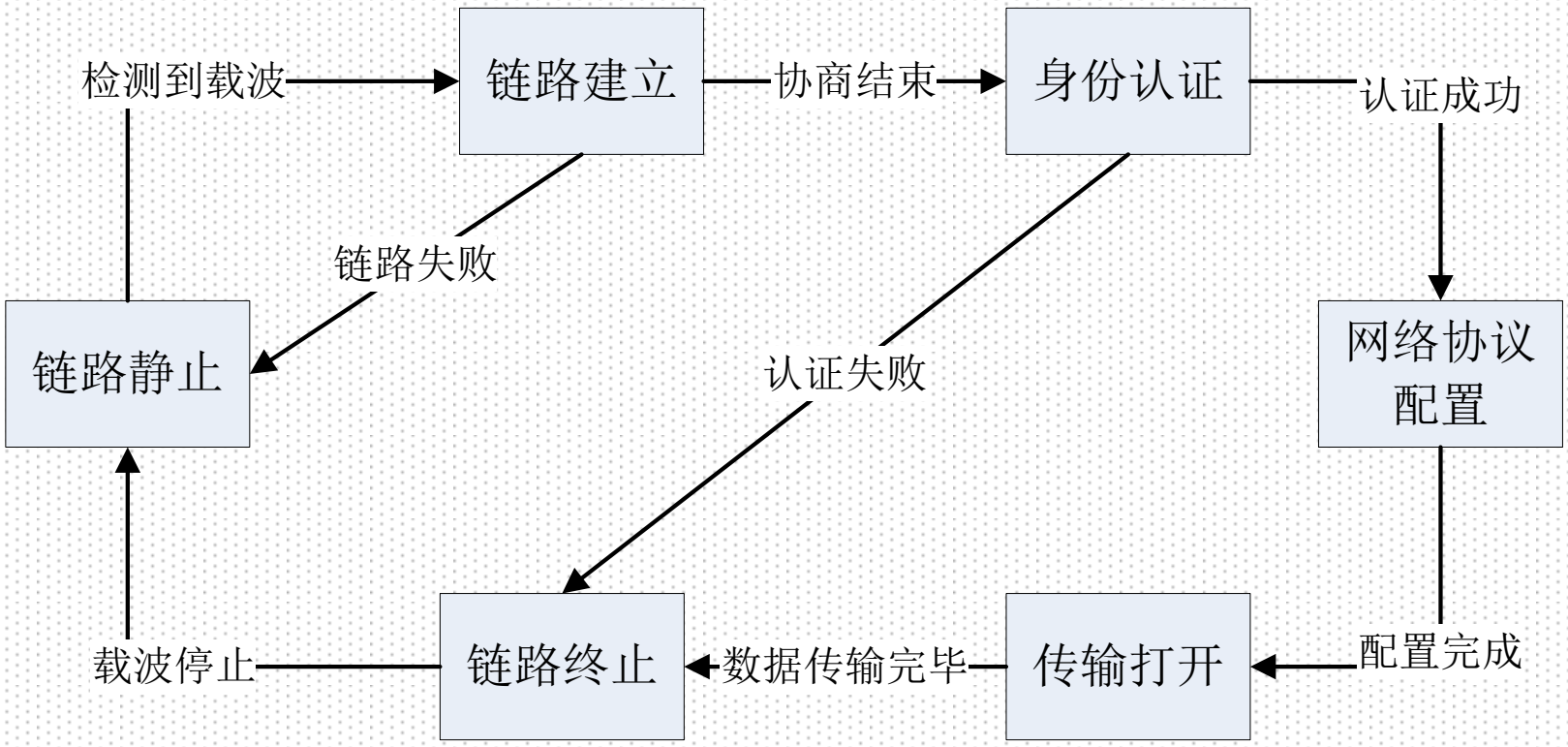
    （2）质询-握手认证协议（CHAP）：加密认证（PAP的改进）

- 口令认证协议（PAP）
  - 两次握手交互方式
    （1）发起通信的一方提供用户名和口令
    （2）对方以认证成功或不成功进行消息响应
  - 缺点：用户名和口令容易被第三方窃取，网络安全性差

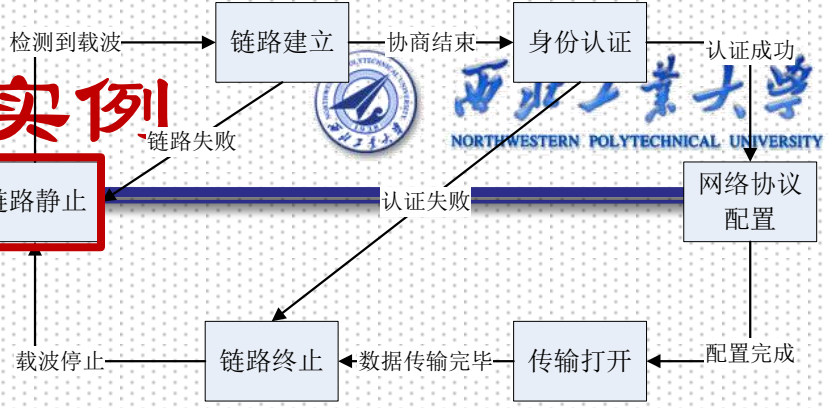
- 质询-握手认证协议（CHAP）
  - 三次握手交互方式
    - （1）验证方向被验证方发送一些随机产生的报文，并附带上主机名一起发送给被验证方
    - （2） 被验证方接收到验证请求时，根据本报文中的用户名和本端的用户表查找用户口令字。如找到用户表中与验证方主机名相同的用户，便利用接收到的随机报文、此用户的密钥生成应答，随后将应答和自己的主机名送回
    - （3）验证方接到此应答后，利用用户表中查找本方保留的口令字，用本方保留的口令字（密钥）和随机报文得出结果，与被验证方应答进行比较，根据比较结果放回相应的结果（ACK 或 NAK）
  - 特点：只在网络上传输用户名，而不传输用户口令，因此安全性比PAP高

3.5 数据链路层协议实例

- PPP运行状态
  - 通讯双方在建立点对点链路后，在协议控制下进行运行状态转换，从而完成数据传输过程



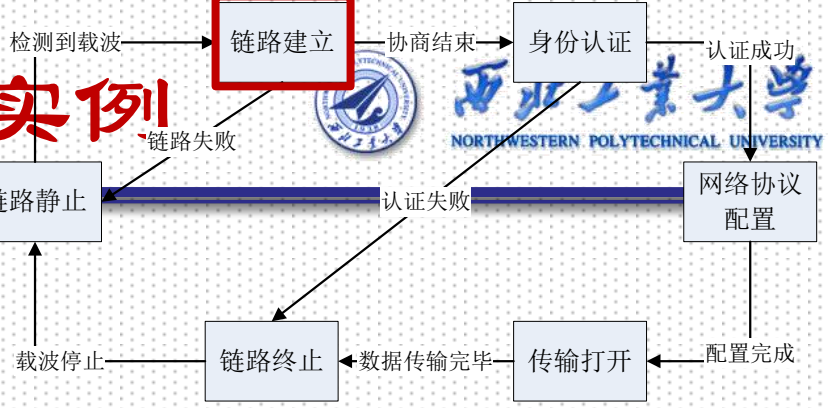
3.5 数据链路层协议实例



- PPP运行状态

(1) 链路静止状态：PPP的起始和终止状态是链路静止状态，不存在物理层的连接，链路不能活动

3.5 数据链路层协议实例

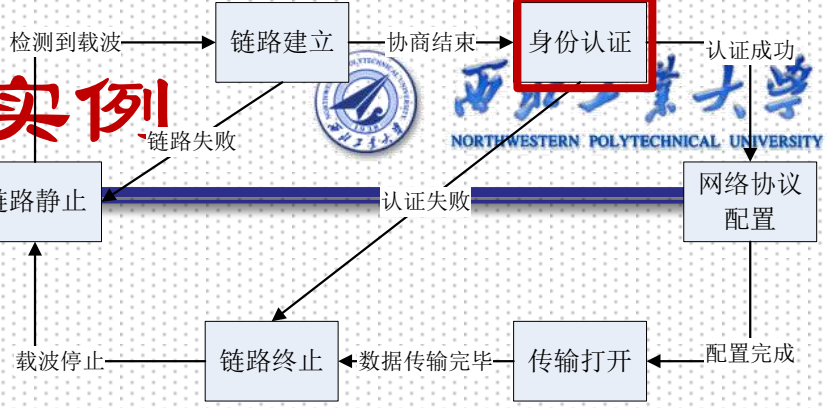


- PPP运行状态

(1) 链路静止状态：PPP的起始和终止状态是链路静止状态，不存在物理层的连接，链路不能活动

(2) 链路建立状态：当物理层检测到载波，则进入链路建立状态。通过链路控制协议LCP协商、配置和测试数据链路

3.5 数据链路层协议实例



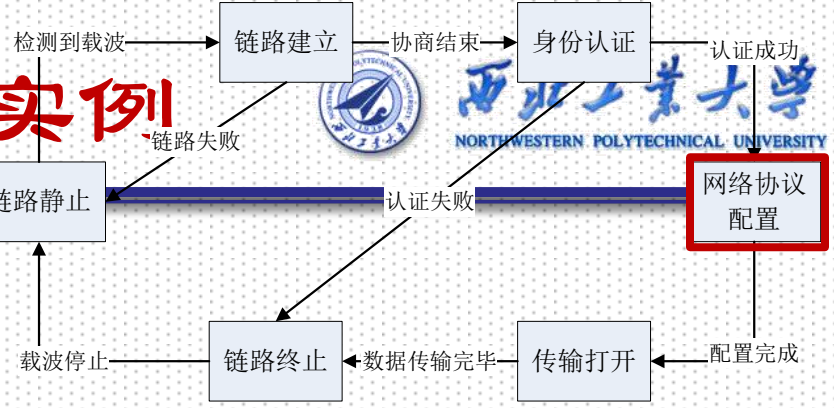
- PPP运行状态

(1) 链路静止状态：PPP的起始和终止状态是链路静止状态，不存在物理层的连接，链路不能活动

(2) 链路建立状态：当物理层检测到载波，则进入链路建立状态。通过链路控制协议LCP协商、配置和测试数据链路

(3) 身份认证状态：双方协商后建立了LCP连接，进入身份认证状态

3.5 数据链路层协议实例



- PPP运行状态

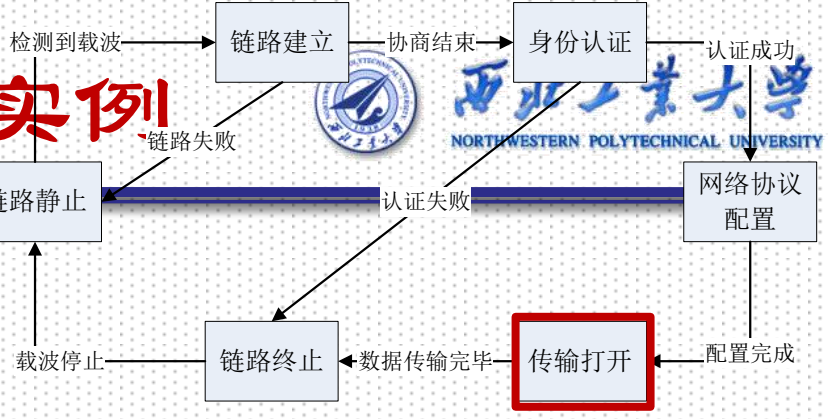
(1) 链路静止状态：PPP的起始和终止状态是链路静止状态，不存在物理层的连接，链路不能活动

(2) 链路建立状态：当物理层检测到载波，则进入链路建立状态。通过链路控制协议LCP协商、配置和测试数据链路

(3) 身份认证状态：双方协商后建立了LCP连接，进入身份认证状态

(4) 网络协议配置状态：若身份认证成功，进入网络协议配置状态。通过发送NCP分组来选择和配置网络层协议

3.5 数据链路层协议实例



- PPP运行状态

(1) 链路静止状态：PPP的起始和终止状态是链路静止状态，不存在物理层的连接，链路不能活动

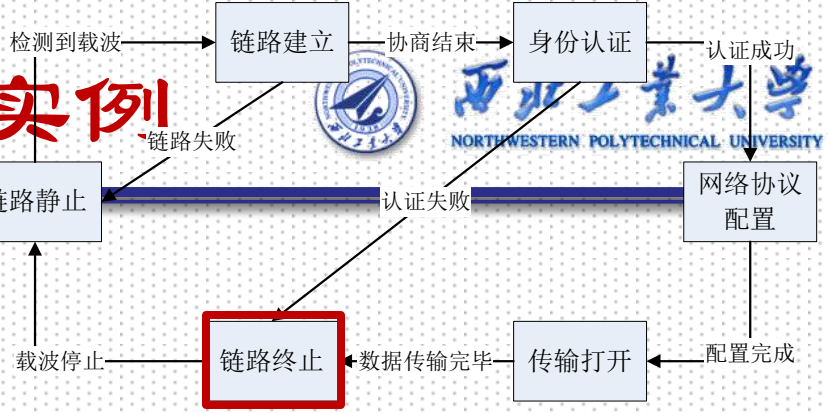
(2) 链路建立状态：当物理层检测到载波，则进入链路建立状态。通过链路控制协议LCP协商、配置和测试数据链路

(3) 身份认证状态：双方协商后建立了LCP连接，进入身份认证状态

(4) 网络协议配置状态：若身份认证成功，进入网络协议配置状态。通过发送NCP分组来选择和配置网络层协议

(5) 传输打开状态：完成网络层协议进入传输打开状态，双方进行通信

3.5 数据链路层协议实例



- PPP运行状态

(1) 链路静止状态：PPP的起始和终止状态是链路静止状态，不存在物理层的连接，链路不能活动

(2) 链路建立状态：当物理层检测到载波，则进入链路建立状态。通过链路控制协议LCP协商、配置和测试数据链路

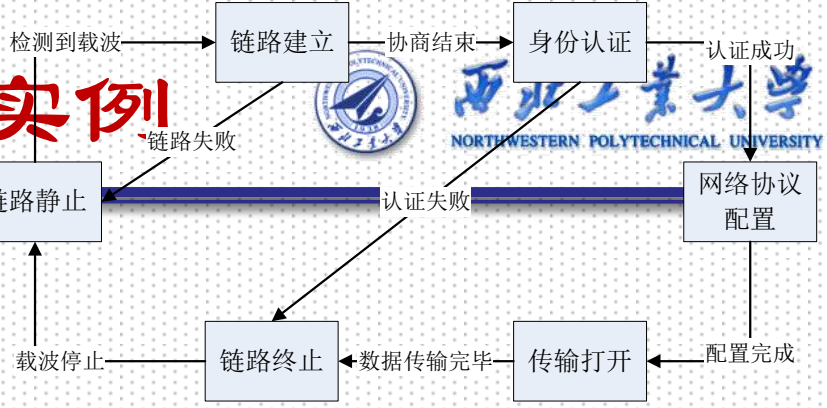
(3) 身份认证状态：双方协商后建立了LCP连接，进入身份认证状态

(4) 网络协议配置状态：若身份认证成功，进入网络协议配置状态。通过发送NCP分组来选择和配置网络层协议

(5) 传输打开状态：完成网络层协议进入传输打开状态，双方进行通信

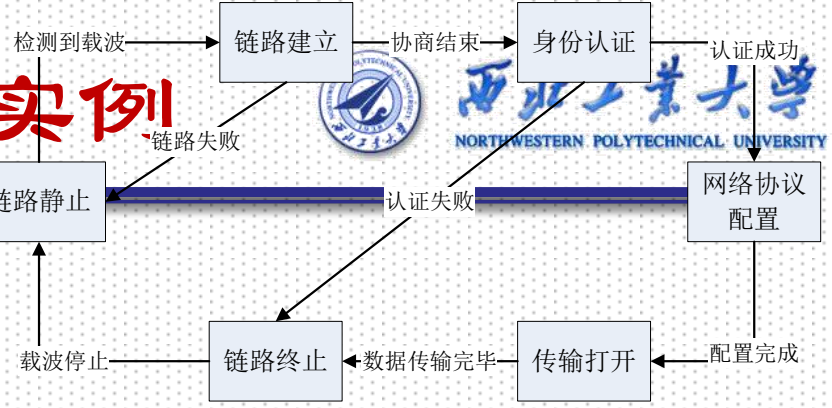
(6) 链路终止状态：数据传输完成、认证失败等，都进入终止状态，关闭连接

3.5 数据链路层协议实例



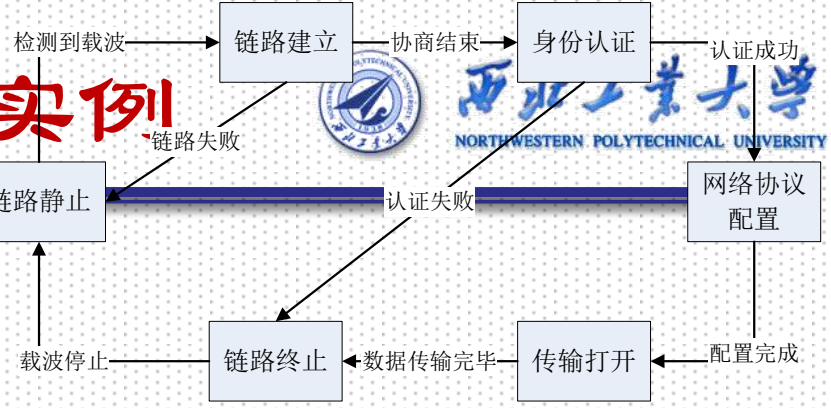
- PPP运行状态
  - PPP链路的起始和终止状态永远是图的“静止状态”，这时并不存在物理层的连接。当检测到调制解调器的载波信号，并建立物理层连接后，PPP就进入链路的“建立状态”

3.5 数据链路层协议实例



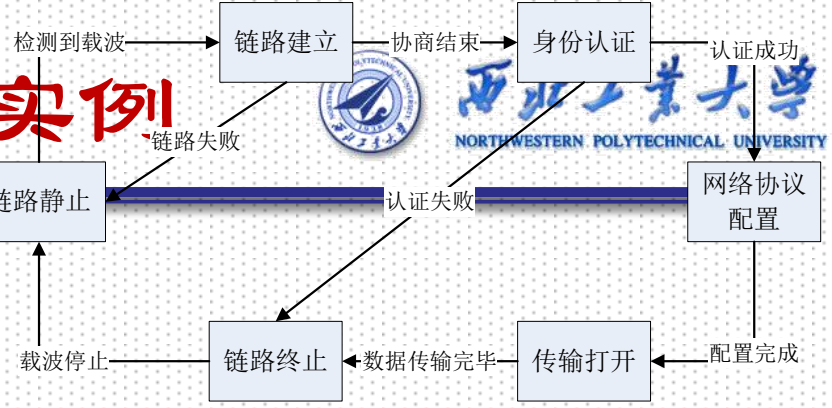
- PPP运行状态
  - 这时LCP开始协商配置选项，即发送LCP的配置请求帧(configure-request)。这是个PPP帧，其协议字段配置为LCP对应的代码，而信息字段包含特定的配置请求。链路的另一端可以发送以下几种响应：
    - (1)配置确认帧(configure-ack)：所有选项都接受
    - (2)配置否认帧(configure-nac)：所有选项都理解但不能接受
    - (3)配置拒绝帧(configure-reject)：选项有的无法识别或不能接受，需要协商

3.5 数据链路层协议实例



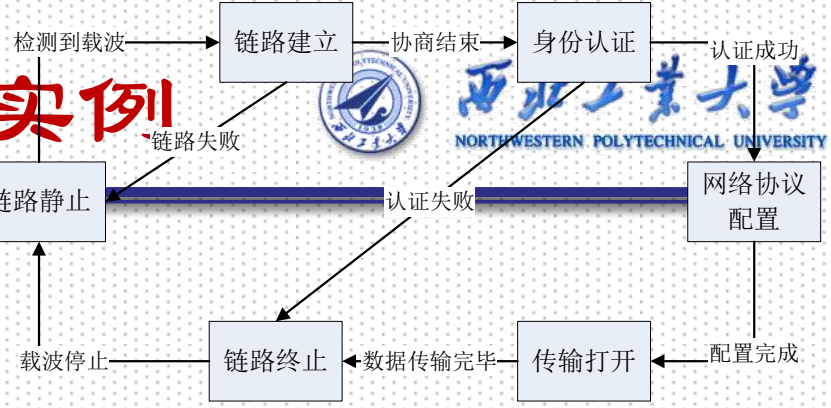
- PPP运行状态
  - LCP配置选项包括链路上的最大帧长、所使用的鉴别协议(authentication protocol)的规约(如果有的话)，以及不使用PPP帧中的地址和控制字段(因为这两个字段的值是固定的，没有任何信息量，可以在PPP帧的首部中省略这两个字节)

3.5 数据链路层协议实例



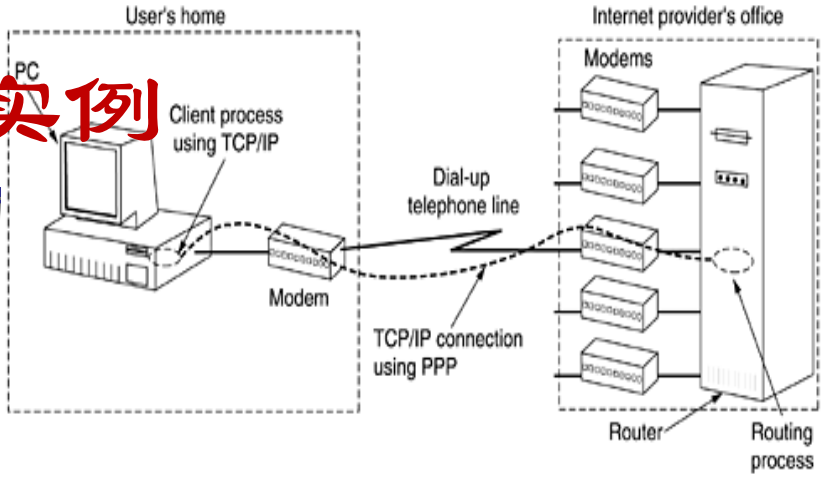
- PPP运行状态
  - 协商结束后就进入“鉴别状态”。若通信的双方鉴别身份成功，则进入“网络状态”。这就是PPP链路的两端互相交换网络层特定的网络控制分组。如果在PPP链路上运行的是IP，则使用IP控制协议IPCP(IP Control Protocol)来对PPP链路的每一端配置IP模块(如分配IP地址)。和LCP分组封装成PPP一样，IPCP分组也封装成PPP帧(其中的协议字段为0x8201)在PPP链路上传送。当网络层配置完毕后，链路就进入可进行数据通信的“打开状态”

3.5 数据链路层协议实例



- PPP运行状态
  - 两个PPP端点还可发送回送请求LCP分组(echo－request)和回送回答LCP分组（echo－reply）以检查链路的状态。数据传输结束后，链路的一端发出终止请求LCP分组(terminate-request)请求终止链路连接，而当收到对方发来的终止确认LCP分组(terminate-ack)后，就转到“终止状态”。当载波停止后则回到“静止状态”

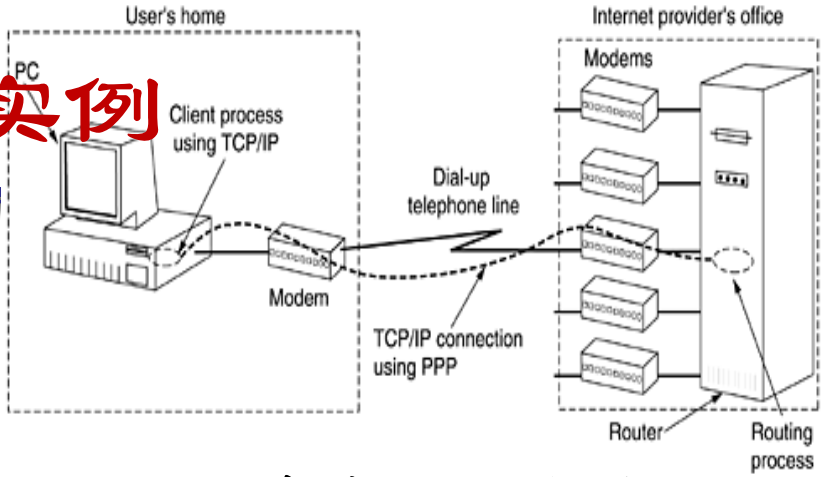
3.5 数据链路层协议实例



- PPP协议典型情况
  - 家庭用户通过调制解调器呼叫ISP，以便个人计算机可以成为一台临时的Internet主机

    (1) PC首先通过调制解调器呼叫供应商的路由器，当路由器的调制解调器回答了用户电话呼叫，并建立起一个物理连接之后，PC给路由器发送一系列的LCP分组，它们被包含在一个或者多个PPP帧的有效载荷中。这些分组以及他们的应答信息将决定所使用的PPP参数

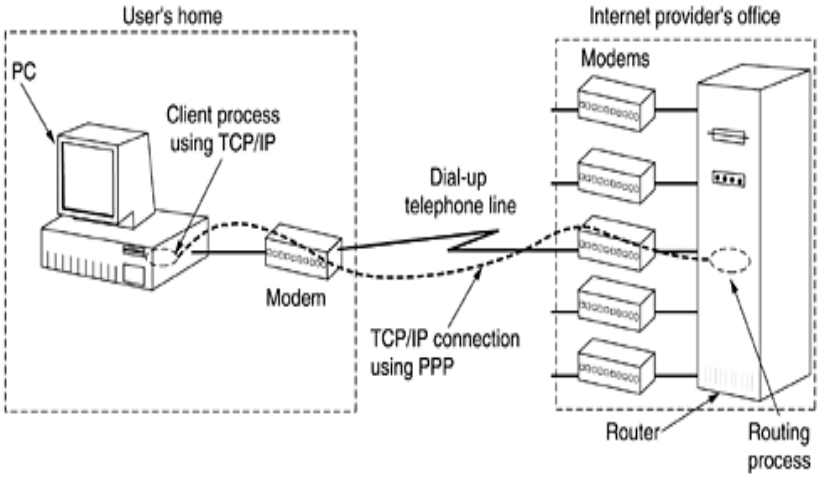
3.5 数据链路层协议实例



- PPP协议典型情况

(2) 一旦对PPP参数达成一致之后，PC会发送一系列的NCP分组，这些NCP分组用于配置网络层。通常情况PC机希望运行TCP/IP协议栈，所以需要有一个IP地址。由于没有足够的IP地址可供使用，所以每个ISP会使用一段IP地址范围，然后动态的分配一个地址给每台新近登陆的PC机，保证在会话过程中使用该地址。如果一个ISP拥有N个IP地址，则可以允许同时有N个用户登陆进来。针对IP协议的NCP负责分配IP地址

3.5 数据链路层协议与



- PPP协议典型情况

(3) 在NCP协商好之后，PC已经成为Internet中的一台主机，可以发送和接收IP分组。当完成工作之后，NCP断掉网络层连接，并释放IP地址。然后NCP停掉数据链路层连接。最后，计算机通知调制解调器挂断电话，释放物理层连接

本章知识点小结

- 数据链路层需要解决的问题？
- 数据链路层的设计要点？
- 为网络层提供的服务有哪些？
- 差错检验方法？
- 数据链路控制基本机制？
- 自动请求重传方法？
- 思考题
  - P69：3.2、3.3、3.8、3.9、3.11、3.12、3.15、3.20、3.21