数据链路层之可靠数据传输

教材目录

3.3 数据链路协议：可靠数据传输

- **停等协议**
- **滑动窗口协议：GBN和选择重传**

6.2.1 TCP概述

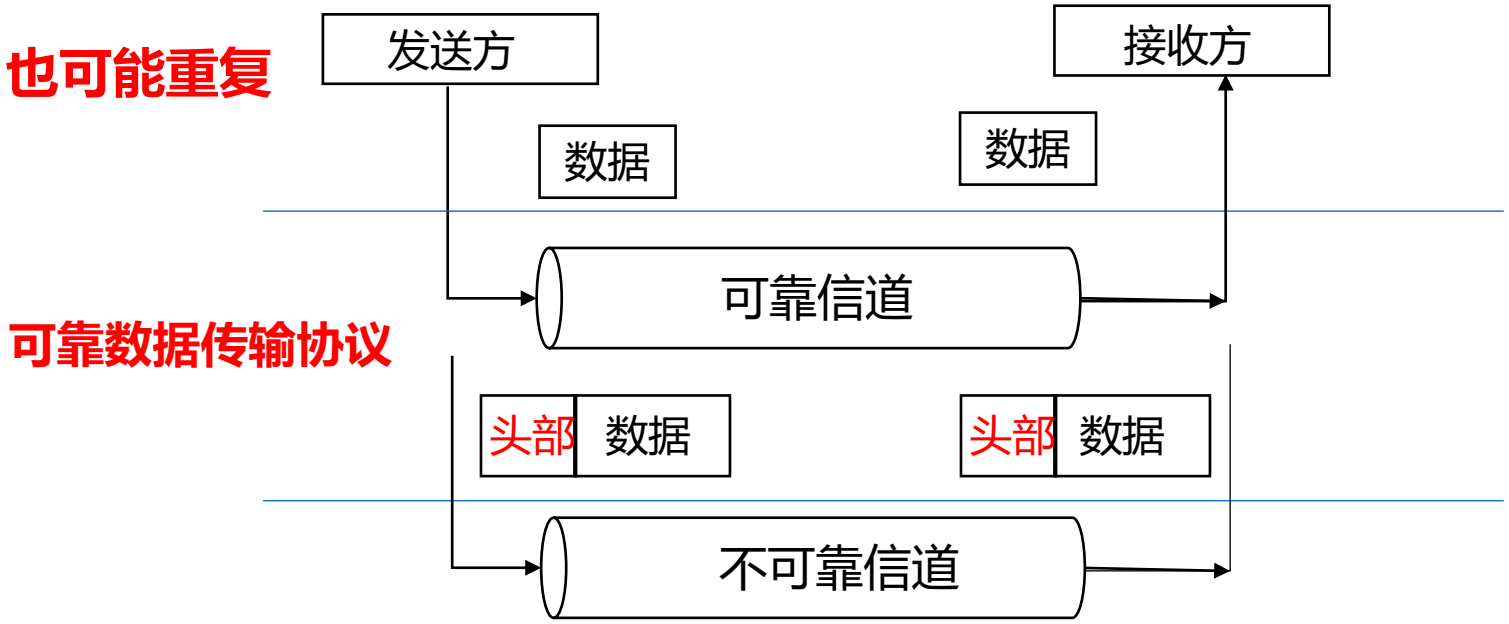
6.2.4 TCP可靠传输

6.2.5 TCP流量控制

6.2.2 TCP段格式

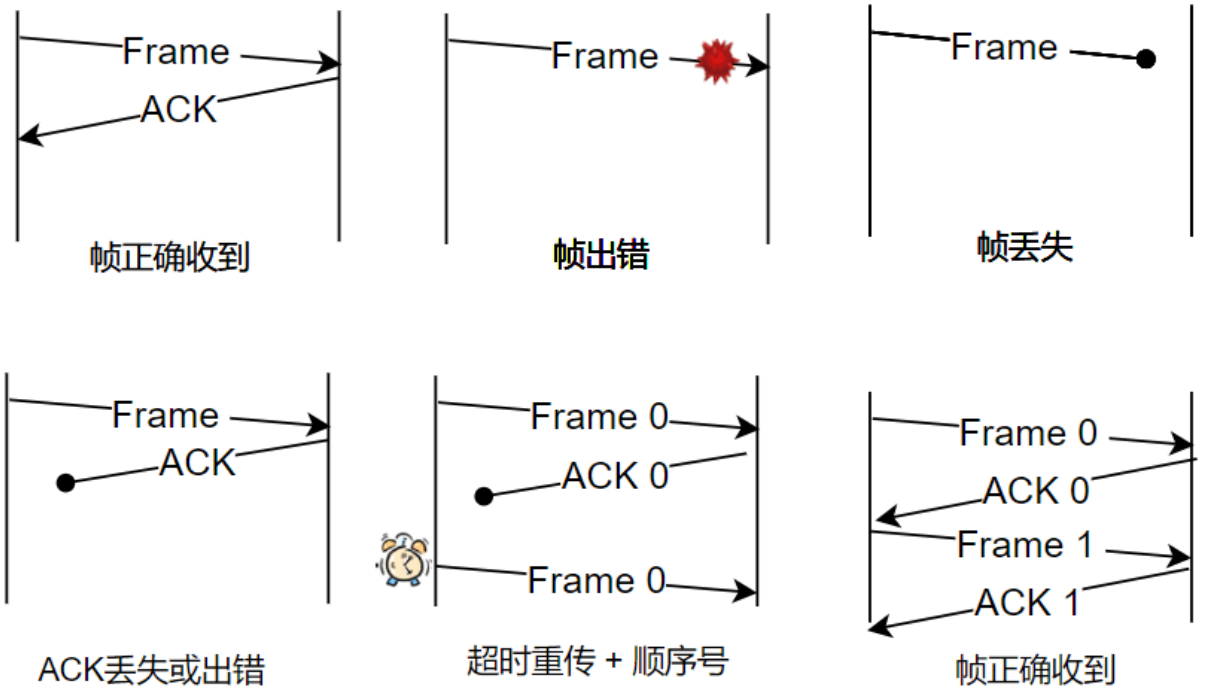
可靠数据传输（3.3 数据链路协议）

- 计算机网络领域最重要的问题之一
- 在运输层和数据链路层都包括了可靠的数据传输功能，应用层也有可能要实现可靠数据传输
- 在运输层(或数据链路层)为高层提供可靠的数据传输服务
- 利用底层所提供的不可靠的数据递交服务。假设底层的数据传输服务：
  - 分组（数据链路层的帧或运输层的TCP段）在传输过程中可能出错，可能完全丢失
  - 链路层一般可假设其传输的帧可能会丢失，但不会失序递交。比如发送1/2/3，假设2号帧丢失，可能收到1/3，但不会出现3/1
  - **TCP假设IP网络可能失序递交，也可能重复**



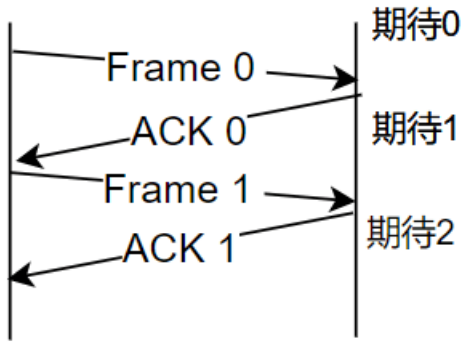
可靠数据传输

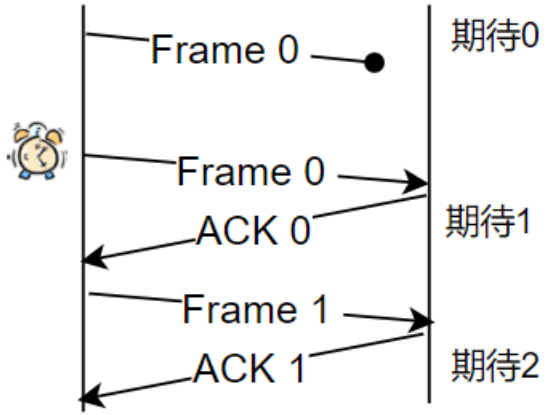
- 数据帧(分组)在传输过程中可能出错，通过**检验和(checksum)**来检测差错
- 发送方传输时附加足够多的冗余信息，接收方能够从错误中恢复，即采用前向纠错FEC(Forward Error Correction)
- 接收方发送**确认(ACK)**通知发送方帧的接收情况
  - (positive)ACK： 确认数据帧正确收到
  - (negative)NAK: 通知数据帧接收时出错，这样发送者可以尽快重传
    - 出错时怎么知道出错的是哪个数据帧呢?
    - NAK较少使用
- 数据帧可能出错或完全丢失；ACK帧也可能会出错或完全丢失
- **超时机制**：发送方一直没有收到确认时认为数据帧丢失而**重传数据帧**
- **顺序号**： （由于ACK帧可能丢失）数据帧应该包括顺序号，以便区分是否为重传帧，类似地ACK帧也应该包含顺序号，以明确哪个或哪些帧已经正确收到
- **ARQ(Automatic Repeat reQuest)**，自动请求重发：采用超时和确认实现可靠数据传输的协议

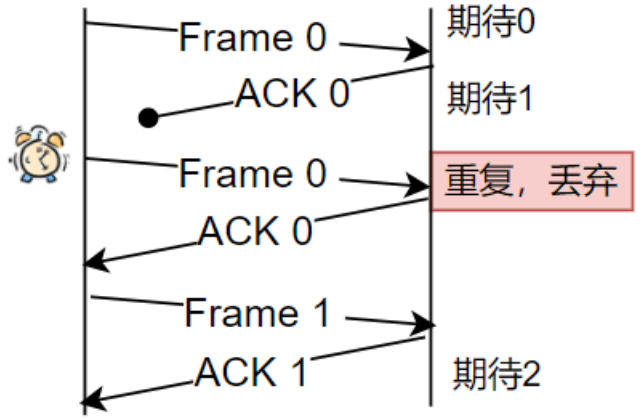


可靠数据传输：停等协议（Stop-and-Wait）

- 发送者在发送一个帧之后停下来，等待对方确认后继续发送下一帧。如果一段时间后没有收到确认而导致发送者超时，重传帧
- 接收者维护期待接收的帧的序号frame_expected（初始期待0号帧）。收到数据帧时：
  - 如果接收正确，发送ACK，更新frame_expected，即期待接收下一个帧
  - 如果收到重复的数据帧，也发送ACK（why?可能是之前的ACK丢失导致的重传)
- 发送者维护下一个发送的新的数据帧的顺序号next_frame_to_send。在收到ACK帧时：
  - 如果是新的ACK，更新next_frame_to_send，发送下一个数据帧（如果有）
  - 有没有可能出现重复的ACK?
    - 有些底层信道传输时可能失序，可能重复
    - 还有没有其他可能的情形？ACK并没有丢失，虚假超时导致数据帧提早重传，从而发送重复ACK（参见后面的slide)

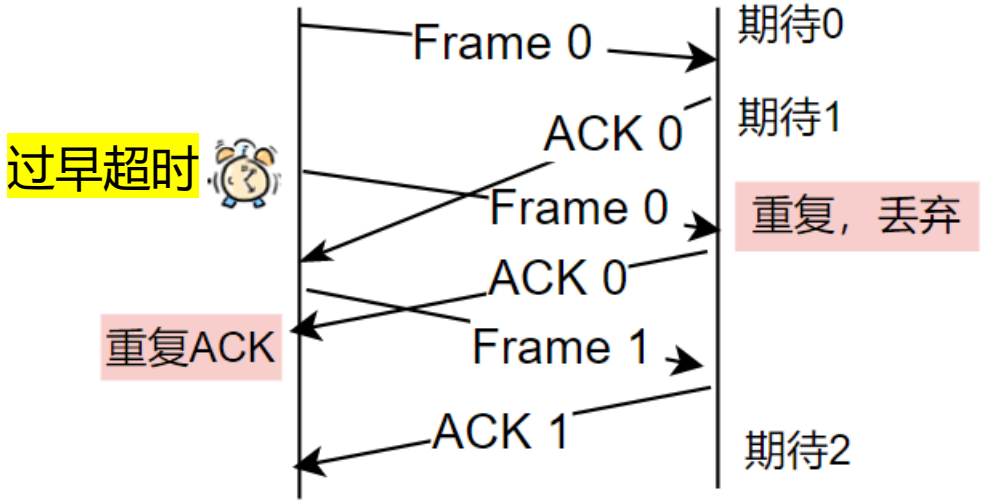






可靠数据传输：停等协议（Stop-and-Wait）

- 虚假超时(spurious timeout): ACK迟来导致的超时，可能出现重复的ACK，而且这些ACK都到达发送方，因此ACK帧也需要顺序号



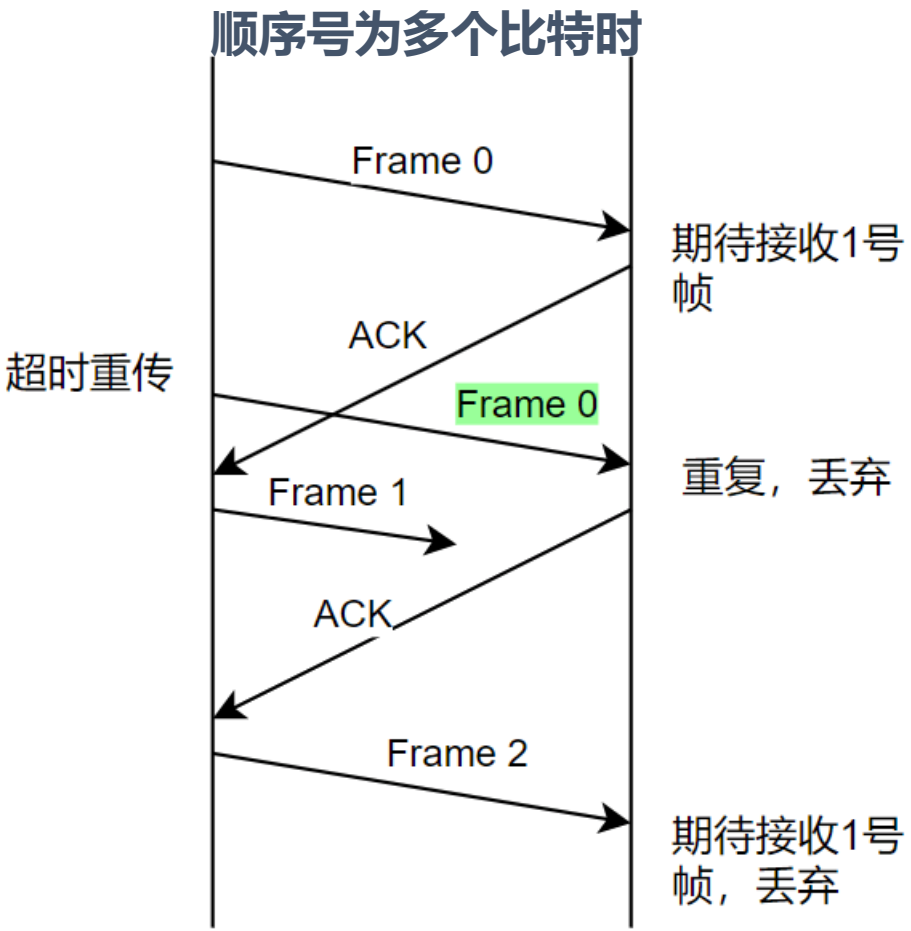
- 对于数据链路层，可假设在链路上的传输可能出错、完全丢失，但是**不会出现失序和重复**的情况
  - 问题：采用停等协议，数据帧和ACK帧头部的顺序号字段需要多少个比特呢?
    - 发送方只有在收到正确的ACK帧之后才会发送新的帧，要能够区分是新帧还是重传帧
    - 接收方收到的是新帧或重传帧，发送的ACK可能是新的ACK或重复的ACK
    - 顺序号字段只要1个比特即可，采用模2运算，顺序号为0、1、0、1...

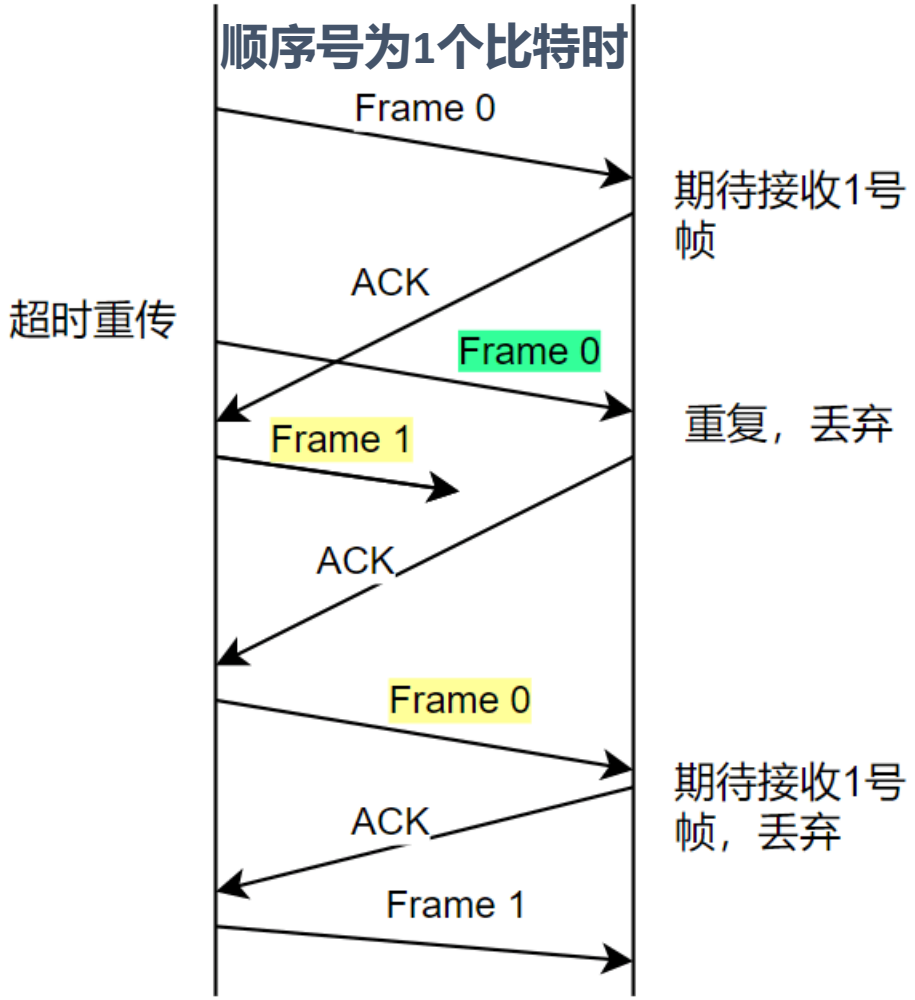
可靠数据传输：停等协议（Stop-and-Wait）

- 简单版本：一方发送而另一方接收，且**确认不包含序号**（协议有问题!）

虚假超时(**spurious timeout**): ACK迟来导致的超时

- 如果顺序号不为1bit,则接收者一直等待Frame 1，停滞





- 如果顺序号为1个比特，即模2运算。注意到发送方发送了老Frame 0、重传的老Frame 0，新的Frame 1(丢失)和新的Frame 0。接收者接受了老Frame 0、重传的老Frame 0(丢弃)、新的Frame 0(丢弃）。重传协议可继续，但是2个帧（新Frame 1和新Frame 0)丢失!

可靠数据传输：停等协议（Stop-and-Wait）

发送方

```
next_frame_to_send= 0;
while (1) {
    wait (frame ready);
    while (1) {
        transmit (frame next_frame_to_send);
        try {
            while (1) {
                receive (ack n);
                if (n == next_frame_to_send) break;
            }
        } catch (timeout) { continue; }
        break;
    }
    next_frame_to_send ++;
}
```

接收方

```
frame_expected = 0;
while (1) {
    receive (frame n);
    ack (frame n);
    if (n != frame_expected)
        continue;
     frame_expected ++;
}
```

addr	seq	ack	length	payload	checksum

- **捎带确认**(piggyback)：大部分的通信是双向的
  - 既充当发送方也充当接收方
  - ACK帧可由反向发送的数据帧一起捎带回来
  - 或者单独的ACK帧

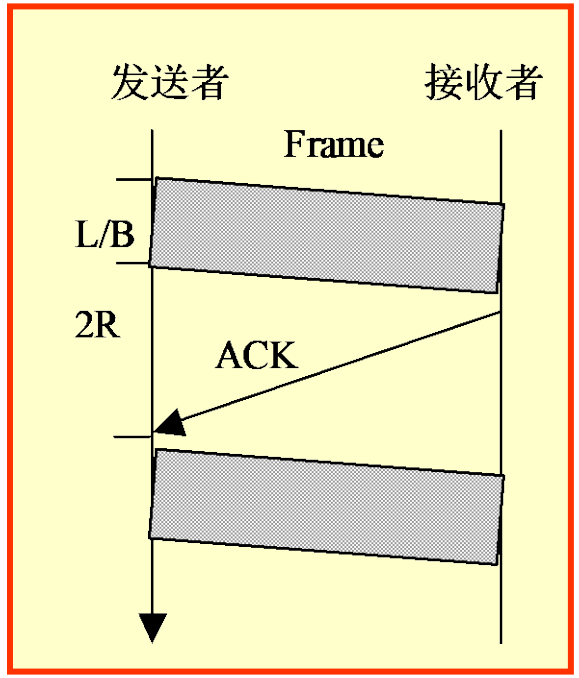
注意：**教材P102的图3.10**假设不支持单独ACK导致大量重复发送，实践中会采用单独ACK，不大会出现这种情形

停等协议：性能分析

- 考虑停等因素的**信道效率或有效利用率(effective utilization)**：信道上的传输中实际用于传输有效数据的比率，也称为**归一化的吞吐率**，即吞吐率与信道带宽的比率
  - 信道容量B bps
  - 帧长L bit
  - 往返传播延迟2R秒
  - 忽略ACK的开销

$$U = \frac{L/B}{L/B + 2R} = \frac{L}{L + 2RB} = \frac{1}{1 + \frac{2RB}{L}}$$

$\frac{2RB}{L}$ ： 往返传播延迟乘积（以帧为单位）



- 以卫星信道为例： B=50kbps,2R=0.5s, L=1000bit

  U = $\frac{L/B}{L/B+2R}$ = 20/(20 + 500) = 1/26

  或者 U = $\frac{L}{L+2RB}$ = 1000/(1000+25000)=1/26

- **吞吐率(throughput)**： 单位时间内通过的(有效)数据量

  throughput = 1/26*50kbps = 1.92kbps

$$efficiency = \frac{\frac{Data}{Link\ Rate}}{Time} = \frac{\frac{Data}{Time}}{Link\ Rate} = \frac{Throughput}{Link\ Rate}$$

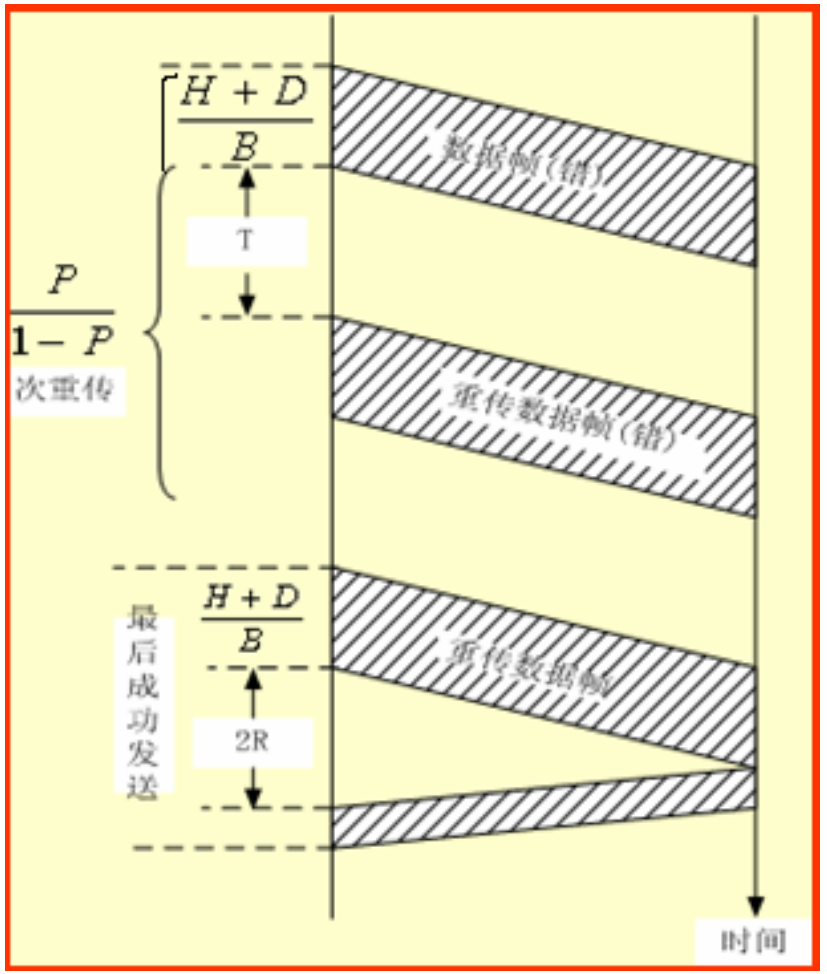
停等协议：性能分析

- 如果考虑ACK、帧头以及**重传的**开销
  - D为帧中有效数据长度，H为帧头长
  - 数据帧长L=H+D，ACK帧长为H
  - T表示发送方等待ACK的超时间隔
  - P1和P2分别表示数据帧和ACK**丢失的概率**

$$N = \sum_{k=1}^{\infty} kp^{k-1}(1-p) = (1-p)\frac{d(\sum_{k=1}^{\infty} P^k)}{dp}$$

$$\because \sum_{k=1}^{\infty} P^k = \frac{1}{1-p} - 1$$

$$N = (1-p)\frac{1}{(1-p)^2} = \frac{1}{1-p}$$



考虑一个数据帧最终传输成功：

- 前面多次传输失败：
  - 传输数据帧(H+D)
  - 超时间隔T
- 最后一次传输成功：
  - 传输数据帧
  - 收到ACK帧

$$N - 1 = \frac{1}{1-P} - 1 = \frac{P}{1-P}$$

$$U = \frac{\frac{D}{B}}{\frac{P}{1-P}(\frac{H+D}{B} + T) + (\frac{H+D}{B} + 2R + \frac{H}{B})}$$

需要首先计算出平均传输次数$N = \sum_{k=1}^{\infty} kp_k$

$$\text{平均传输次数} N = \sum_{k=1}^{\infty} kP^{k-1}(1-P) = \frac{1}{1-P}$$

- P_k=正好传输k次成功的概率，即前面k-1次失败，第k次成功
- **某次传输失败的概率**P=1-Data/ACK都成功的概率，即$P = 1 - (1-P_1)(1-P_2)$

几何分布的随机变量的期望是单次实验成功的概率的倒数

概率论知识：几何分布以及其数学期望

- 进行多次独立的伯努利实验，每次实验中成功的概率为p
- 随机变量X为多次伯努利实验中第一次成功的实验次数，X的可能取值为1, 2, ….
- 我们称X服从几何分布，$P\{\mathrm{X}=k\}=(1-p)^{k-1}p$
- 服从几何分布的随机变量X的数学期望(均值) $\mathrm{E(X)}=\sum_{k=1}^{\infty}kP\{X=k\}=\sum_{k=1}^{\infty}k(1-p)^{k-1}p=\frac{1`}{p}$

几何分布的随机变量的期望是单次实验成功的概率的倒数

停等协议：性能分析

- 假设信道上每比特的误码率为E，且各比特相互独立
- P1和P2分别表示数据帧和ACK**丢失的概率**
  - 帧不丢失的概率=每个比特都成功传输的概率

$$1-P_1=(1-E)^{H+D}$$
$$1-P_2=(1-E)^{H}$$

$$P=1-(1-P_1)(1-P_2)$$

- 为避免出现误重传，超时间隔T必须取得足够大，即T≥H/B +2R，U要达到最大值（T越小越好），应该 **T=H/B +2R，即BT = H + 2RB**

$$U=\frac{\frac{D}{B}}{\frac{P}{1-P}(\frac{H+D}{B}+T)+(\frac{H+D}{B}+2R+\frac{H}{B})}$$

T=H/B +2R

$$U=\frac{D(1-E)^{2H+D}}{(H+D+BT)}$$

$$U_{max}=\frac{\frac{D}{B}}{(\frac{P}{1-P}+1)(\frac{H+D}{B}+2R+\frac{H}{B})}$$
$$=\frac{H+D}{H+D+2RB+H}\bullet(1-P_1)(1-P_2)\bullet\frac{D}{H+D}$$

停等的因素　　出错重传因素　　帧头的开销

最佳帧长度Dopt ：假设H、B、T(=H/B+2R)、E不变

$$U_D'=0$$

$$\text{Dopt}=\frac{H+BT}{2}(\sqrt{1-4/[(H+BT)\ln(1-E)]}-1)$$

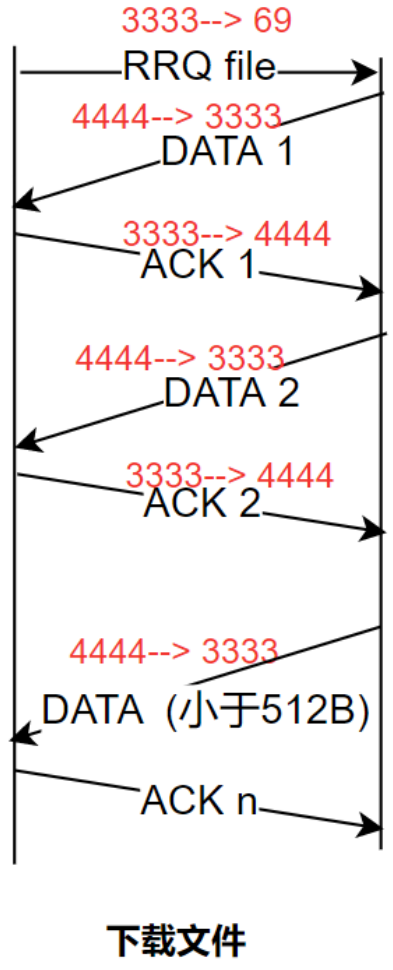
头部越大
带宽越大
误码率越低
} 数据部分越大

$E\to 0$时 $\sqrt{1-4/[(H+BT)\ln(1-E)]}-1\approx\sqrt{4/[E(H+BT)]}$

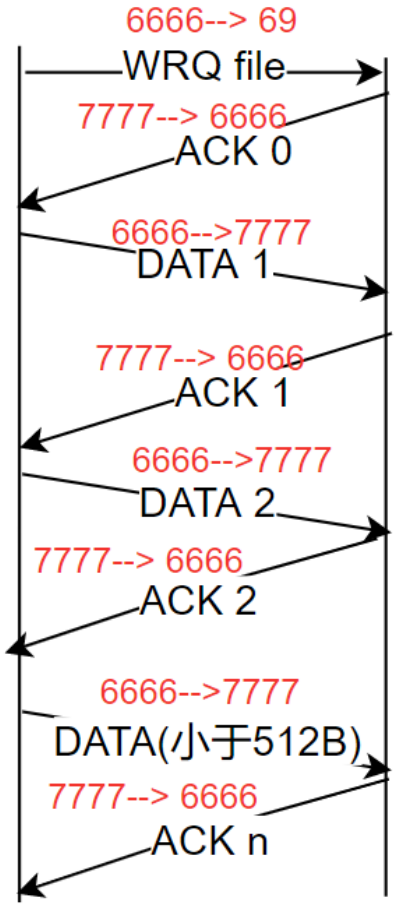
$$Dopt\approx\sqrt{\frac{H+BT}{E}}$$

拓展：TFTP(Trivial File Transfer Protocol)

- RFC 1350: The TFTP Protocol (Revision 2)
- 采用UDP，缺省端口号为69，仅仅提供了下载文件和上传文件的功能
- TFTP client选择一个端口，然后发送RRQ(Read Request，下载文件)或WRQ(Write Request，上传文件)给TFTP服务方，TFTP服务方选择某个端口，以后的数据传输就在client和server之间选择的这两个端口之间展开
- 采用停等协议，顺序号从1开始递增
- 前面的数据分组携带的数据长度都为512字节，以一个长度小于512字节（0-511）的数据表示传输结束



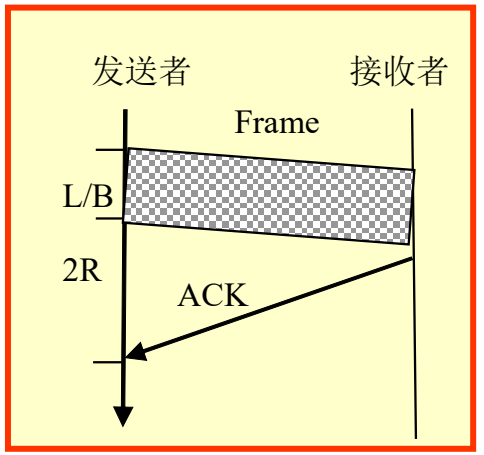
下载文件



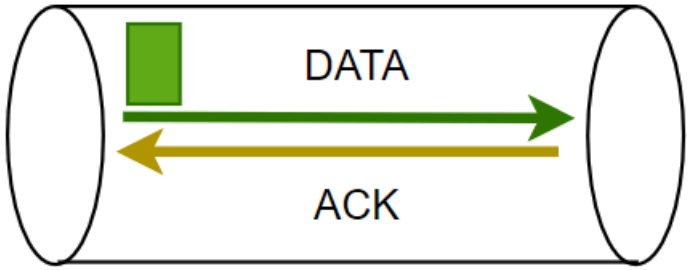
上传文件

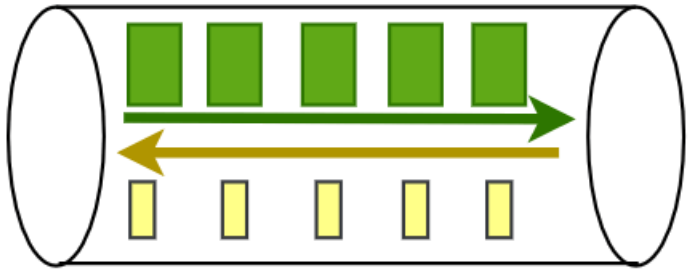
管道化(pipelining)

- 停等协议在开始传输一个数据帧到确认回来这一段时间里必须等待，当往返传播时间(2R)远大于传输时间(L/B)的时候会带来很大的浪费
  - 覆盖范围100m的传播时间大致为0.5个微秒；覆盖范围1000m大约为5个微秒；覆盖范围1000km(上海到北京?)大约为5ms；覆盖范围15000km(上海到纽约?)大约为75ms
  - 1500字节的帧在100kbps链路上的传输时间120毫秒；在10Mbps链路上为1.2ms；在100Mbps链路上为0.12ms



- 管道Pipelining：
  - 发送方在等待确认帧回来之前可以连续发送多个数据帧
  - 要求更大的顺序号空间
  - 发送者要求能够**缓存那些已经发送的但是没有确认的帧**，接收者也可以缓存失序到达的帧
  - **发送窗口**：发送方允许发送的数据帧的顺序号范围，在收到ACK后滑动发送窗口





管道化(pipelining)

- **发送窗口**：发送方允许发送的数据帧的顺序号范围
  - 停等协议的发送窗口=1
- 流量控制：控制发送方的发送速度，避免接收者来不及进行处理

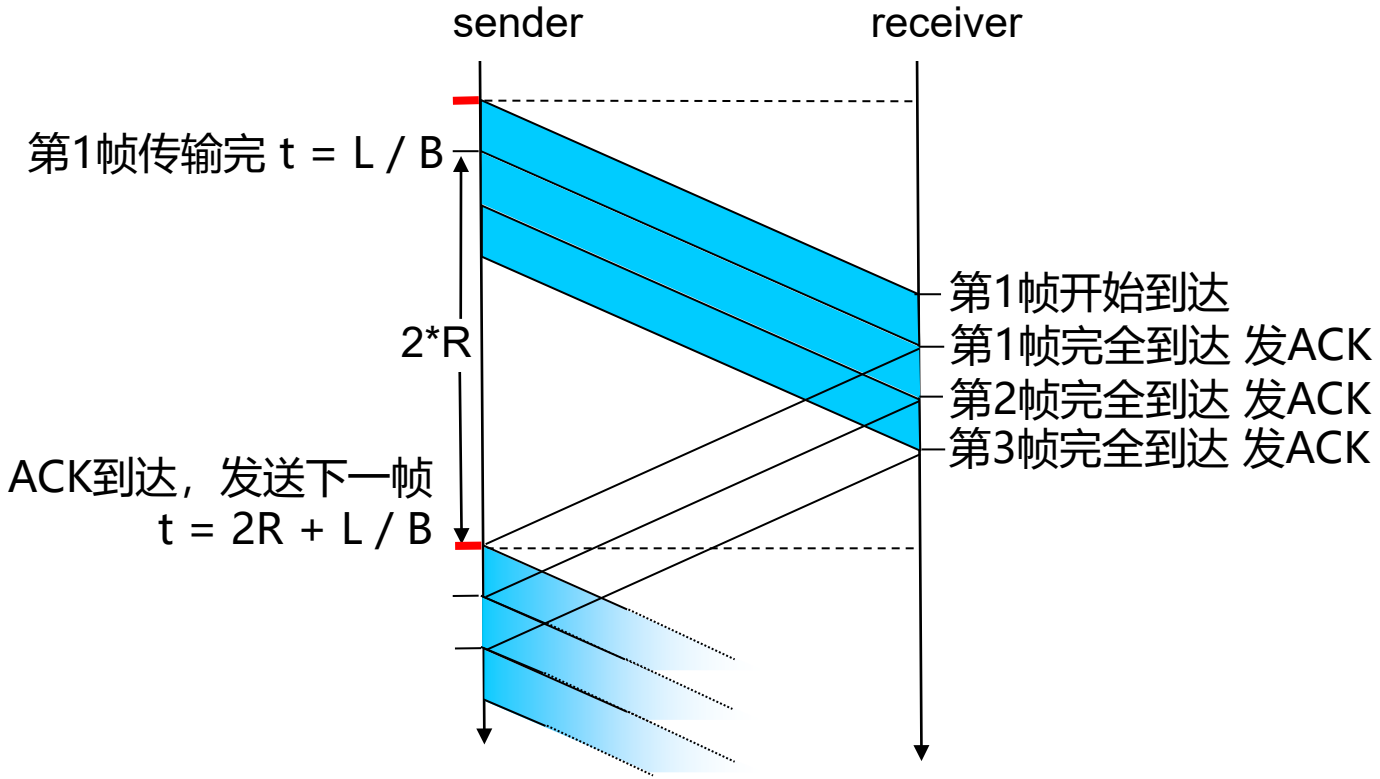
$$U_{pipeline} = \frac{W * L/B}{2R + L/B}$$

要达到100%的信道利用率：

$$W_{\max} = \frac{L + 2R * B}{L}$$

顺序号的bit数要求至少：

$$\left\lceil \log_2 \frac{L + 2BR}{L} \right\rceil$$



✓真正的顺序号空间可能还要更大，要考虑到ACK丢失时窗口的变化

滑动窗口协议

- 流量控制：控制发送方的发送速度，避免接收者来不及进行处理
- **发送窗口**：发送方在收到新的ACK之前允许发送的数据分组的顺序号范围（**只有序号在窗口内的分组才可以发送**），也称为outstanding或in-flight
- **接收窗口**：接收方目前期待接收的数据分组序号（**只有序号在窗口内的分组才可以接收**，否则丢弃）
- 停等协议：发送窗口和接收窗口都是1
- 差错控制：如果数据分组和ACK在传输过程中出错时如何恢复

GBN(Go back N)

- 发送者可以发送N个未确认的分组(发送窗口=N)
- 发送者为所有未确认分组维护一个计时器
  - 计时器超时时，重传**所有未确认分组**
- 接收者只允许顺序接收，即接收窗口=1
- 接收者发送**累加确认**：不会单独确认那些失序到达的分组
  - 由于只允许顺序接收，发送ACK n，表示n之前的都已正确收到)

选择重传(Selective Repeat) （第1种实现策略）

- 发送者可以发送N个未确认的分组(发送窗口=N)
- 发送者为每个未确认分组各维护一个计时器
  - 计时器超时时，仅仅重传**该超时的分组**
- 接收方允许失序接收，即接收窗口>1
- 接收者对每个到达(包括**失序到达**)分组发送**单独ACK**

滑动窗口协议:GBN(Go back N)

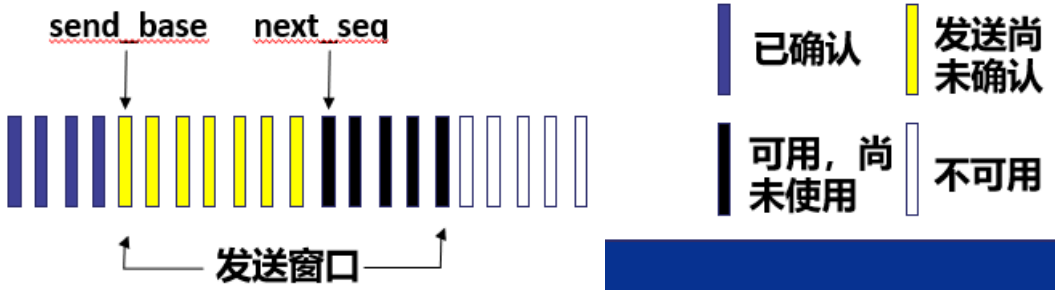
之前的停等协议和教材中的描述：ack n表示n已经正确收到，采用累加确认

发送者

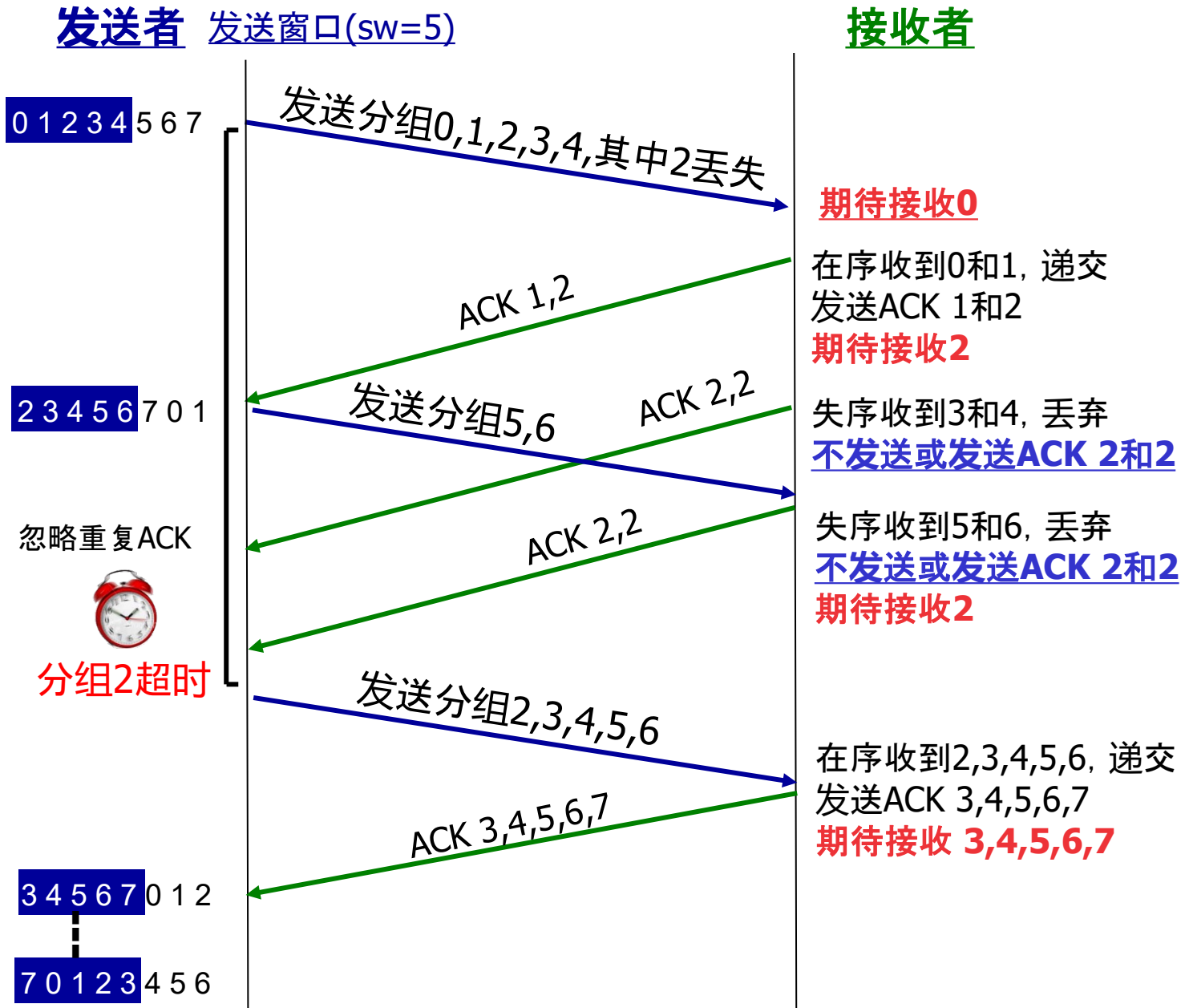
- 发送窗口：允许发送的尚未确认的分组顺序号空间 [send_base, send_base + SW - 1]
  - send_base: 发送窗口前沿，之前的分组都已经确认
  - next_seq: 下一个待使用的顺序号
- 有分组要发送时，如果发送窗口允许，则发送并缓存，更新next_seq
- 在收到ACK时：
  - 如果是**新的ACK**，即确认了outstanding分组
    - 更新send_base = ack
    - 发送窗口允许，选择下一个分组发送
  - 收到了**其他ACK**，丢弃
- 计时器，超时时**重传所有outstanding分组**

接收者

- 采用**累加确认**(Cumulative ACK)
  - ACK n：n之前分组已经到达接收者，**期待n**
  - 可以比较好地应对可能的ACK丢失：后续的ACK确认的信息包含了之前ACK确认的信息
- 接收窗口=1，只允许顺序接收，维护expected
  - 如果**收到的分组顺序号等于expected**，发送新的ACK expected+1并更新expected
  - 如果**收到的分组顺序号<expected**，发送（重复的）ACK expected
  - 如果收到的分组顺序号>expected，失序到达，丢弃(也可以发送ACK expected)



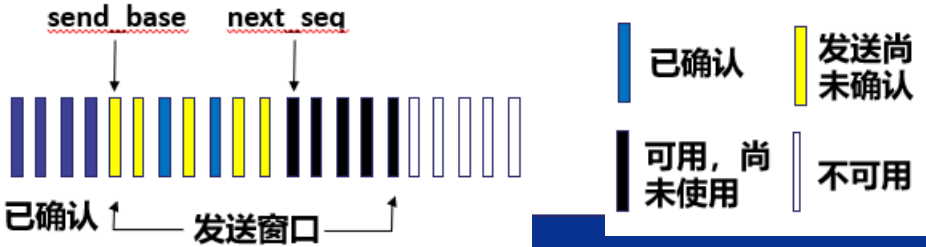
滑动窗口协议:GBN(Go back N)实例



滑动窗口协议:选择重传(Selective Repeat)

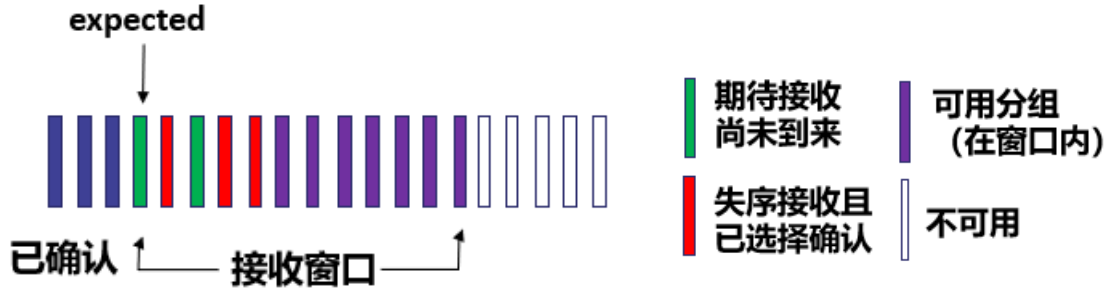
发送者

- 发送窗口：允许发送的尚未确认的分组顺序号空间 [sendbase, sendbase + SW - 1]
  - send_base: 发送窗口前沿，之前分组都已确认
  - next_seq: 下一个待使用的顺序号
- 有分组要发送时，如果发送窗口允许，则发送并缓存，更新next_seq
- 在收到ACK seq时：
  - 如果**seq在发送窗口内**，标记分组已收到；检查是否seq之前的都已收到，如果收到，则滑动send_base到下一个未确认顺序号
  - 收到了其他ACK，丢弃
- 每个分组一个计时器，**超时时重传该分组**

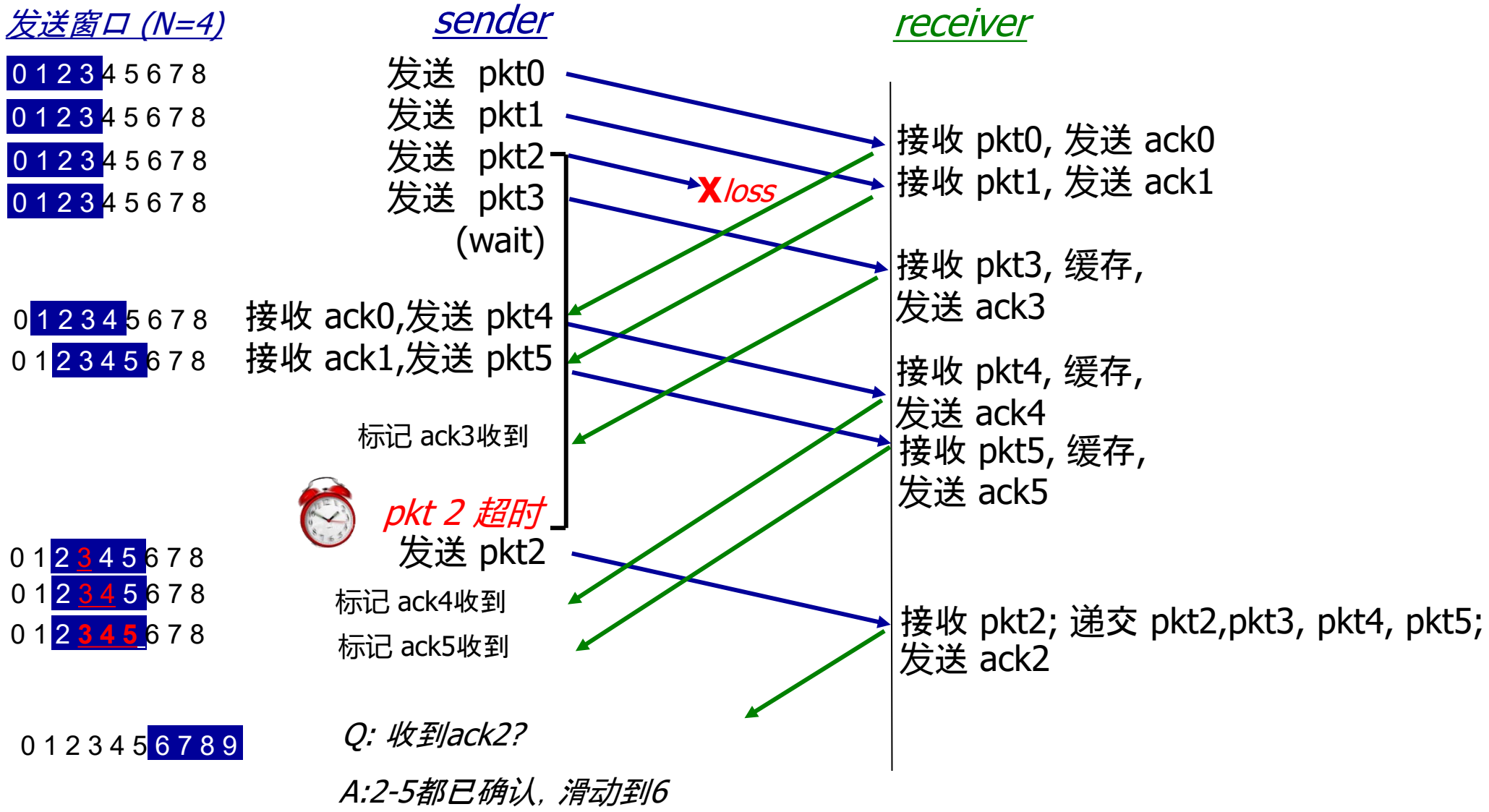


接收者

- **允许失序接收**，接收窗口 > 1， [expected, expected + RW - 1]
  - expected: 接收窗口前沿，之前分组都已确认
- **单独(individual)ACK**：表示已经收到指定顺序号的分组（并缓存或递交）
- 在收到顺序号为seq分组时：
  - 如果seq等于expected，发送ACK seq，滑动接收窗口前沿expected到下一个尚未收到的顺序号
  - 如果seq在接收窗口内，则缓存，并发送ACK seq
  - 如果seq不在接收窗口内，丢弃，**并发送ACK seq**



滑动窗口协议:选择重传(Selective Repeat)示例



滑动窗口协议:选择重传(Selective Repeat)顺序号回绕

- 接收窗口≤ 发送窗口，一般接收窗口=发送窗口

习题3.23 选择重传协议，顺序号m比特，发送窗口最大为2^{m-1}

假设顺序号m=2个比特

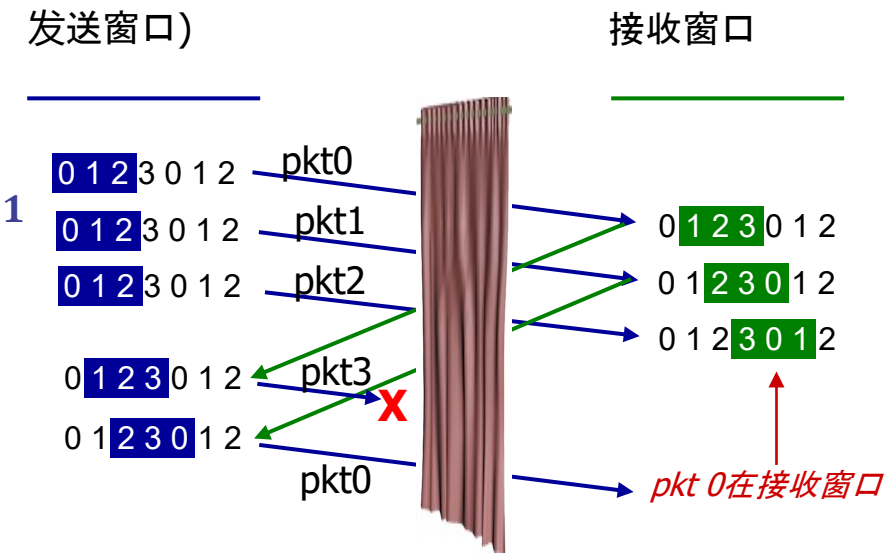
- 顺序号空间: 0, 1, 2, 3
- 发送窗口=接收窗口，最大为$2^{m-1} = 2$

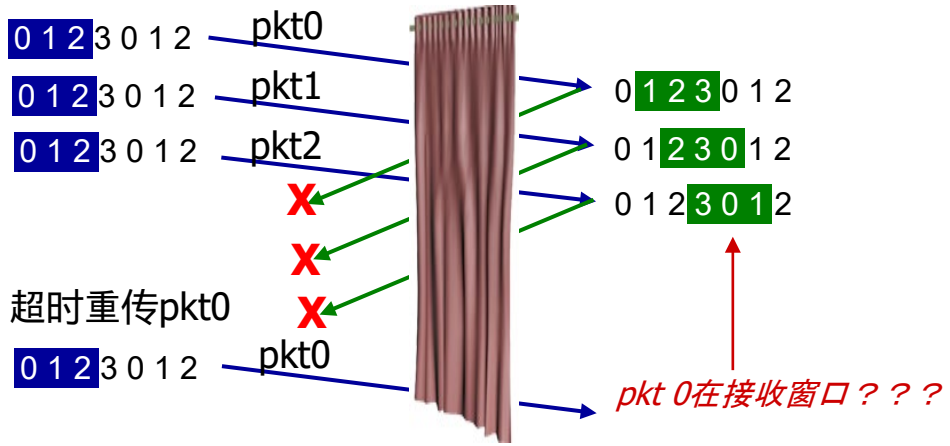
反证法：假设窗口大小=3

发送者和接收者状态不同步

- 接收者同样的行为
- 发送者不同的行为

考虑ACK帧没有丢失或者ACK帧全部丢失的情形





滑动窗口协议：教材的版本

- GBN：与之前介绍的版本基本一致
  - 采用累加确认
  - 只允许在序接收：失序到达的分组丢弃
  - 超时时：重传所有发送窗口中的分组
- 选择重传：（第2种实现策略）
  - 采用累加确认
  - 引入NAK n，表示请求**重传分组n**
    - 出错时发送NAK，但是帧出错，其顺序号可能也出错
    - 假设：可能会丢失，但不**改变顺序**
      - 分组出错或失序到达时发送NAK expected
      - 最近发送的反馈为NAK时，不发送第2个NAK
  - 每个分组对应一个计时器，超时时**重传该分组**

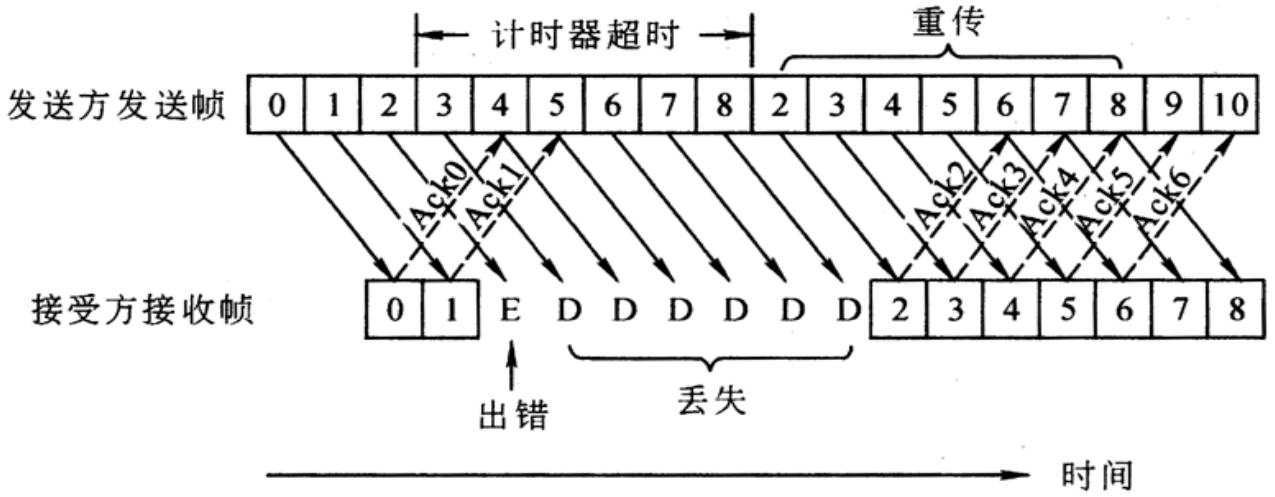


图3.11 回退n

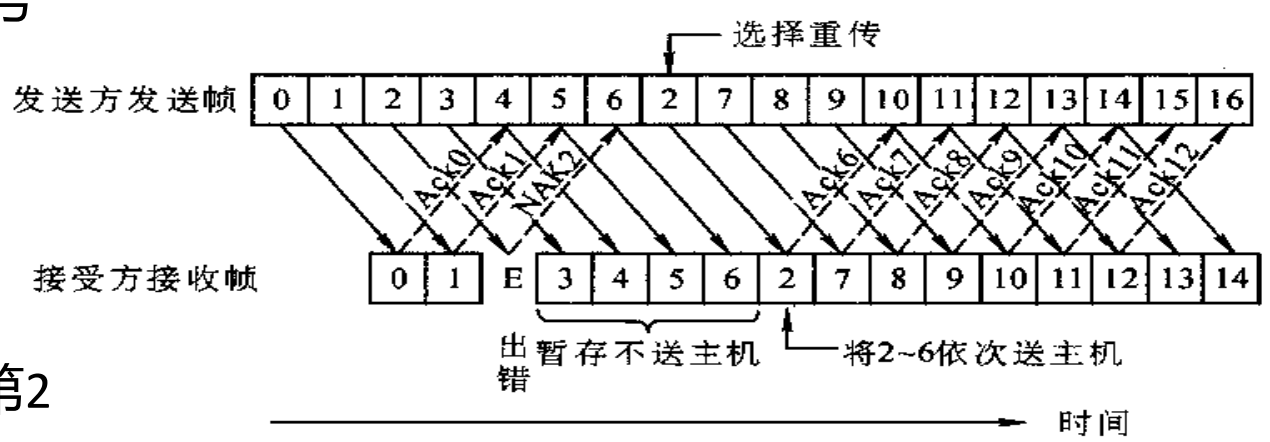


图3.12 选择重传

数据链路协议总结

- **发送窗口：**表示在收到对方确认的信息之前，可以连续发出的数据帧（**只有序号在窗口内的帧才可以发送**）
- **接收窗口：**可以连续接收的数据帧（**只有序号在窗口内的帧才可以接收**，否则丢弃）
- 接收窗口（ACK）驱动发送窗口的转动
- 假设**接收窗口的大小不变**，接收者在序收到分组后立即递交给更高层，不再占用缓冲区
- 停等协议：发送窗口=接收窗口=1
- GBN：接收窗口=1，发送窗口最大为$2^m - 1$
- 选择重传：1 < 接收窗口≤ 发送窗口
  - 一般接收窗口=发送窗口，最大为2^{m-1}
  - 失序到达的分组被缓存
  - 两种实现策略：
    - 单独ACK+ 超时重传该分组
    - 累加ACK+ NAK（或根据收到的重复ACK推断）重传尚未收到的某个分组 + 超时重传该分组