第四章 数据链路层进阶

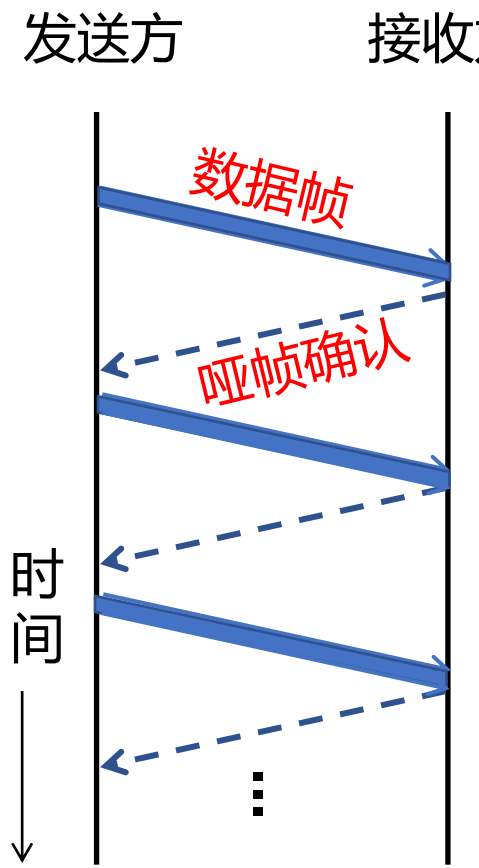
授课教师：崔勇

清华大学

计算机网络
教案社区

致谢社区成员	
安徽大学 王贵竹	华北科技学院 陈振国
华东交通大学 王艳	福州大学 张浩
中国传媒大学 林卫国	枣庄学院 徐涛
华南理工大学 王昊翔	南开大学 徐敬东

思考：提高信道利用率



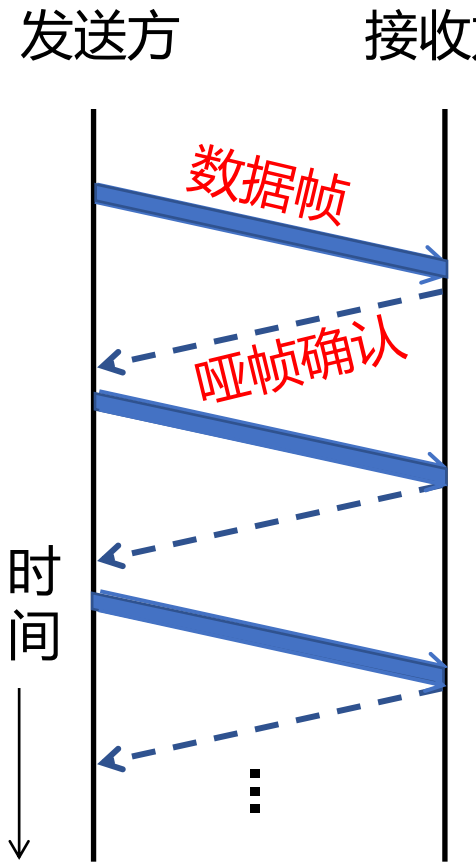
停等协议
在途数据太少

- 停止等待协议的效率问题
  - 停止等待协议的问题是只能有一个没有被确认的帧在发送中
- 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，即停止等待协议的发送工作时间
  - 发送空闲时间：D =2I
  - 信道利用率 (line utilization)=F/(F+R·D)
  - 当 F<DR 时，信道利用率 < 50%

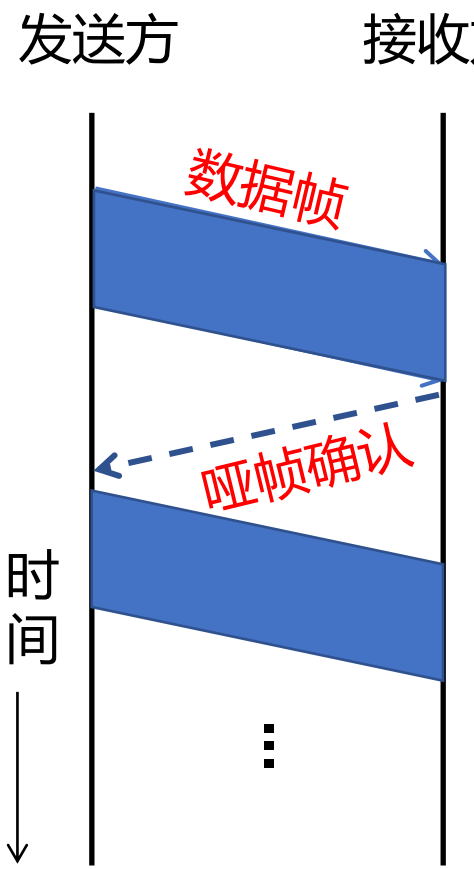
摘自4.3节

发明新协议，以提高效率?

思考：提高信道利用率



停等协议
在途数据太少

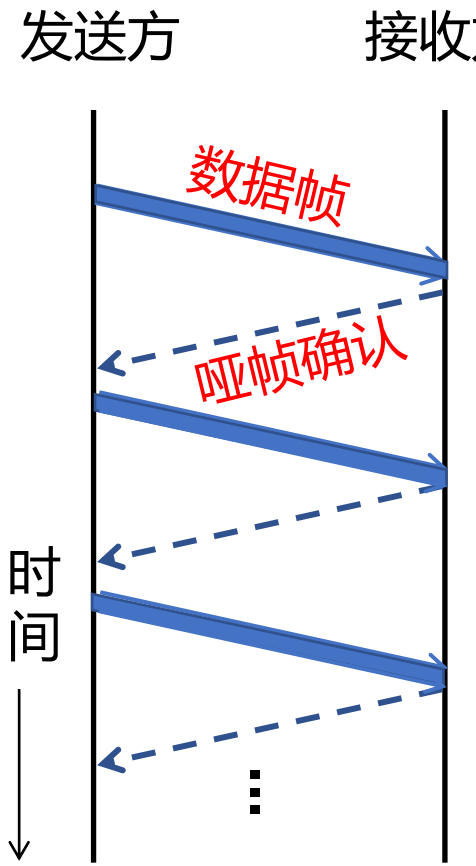


增加帧大小?

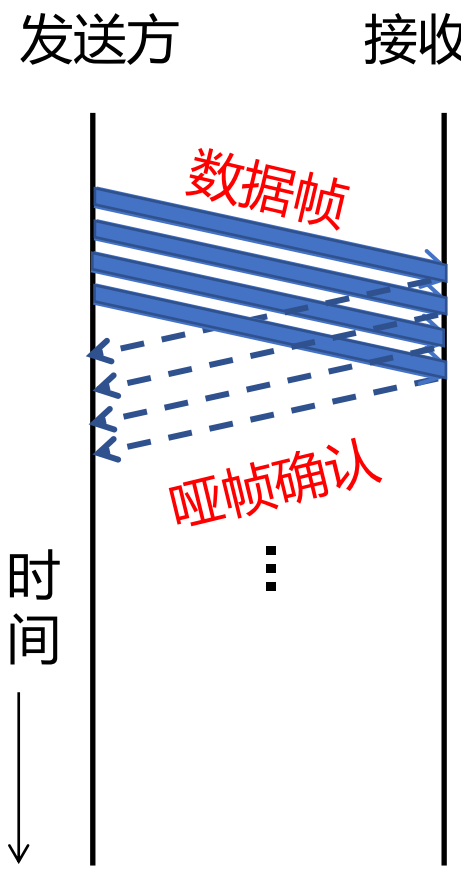
- 但是帧的最大长度受到信道比特错误率（BER，Bit Error Ratio）的限制
- 帧越大，在传输中出错的概率越高，将导致更多的重传

摘自4.3节

思考：提高信道利用率



停等协议
在途数据太少



帧大小不变，但允许多个没有被确认的帧在发送中？

- 帧丢失？
- 重传哪个帧？
- 流控：未确认的帧数？
- 哑帧确认？

本节目标

- 理解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）
- 了解典型链路层协议的工作机制
- 理解由状态与变迁构成的状态转换图（状态机）

本章内容

清華大學 Tsinghua University

计算机网络教案社区

4.4 滑动窗口协议

4.5 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

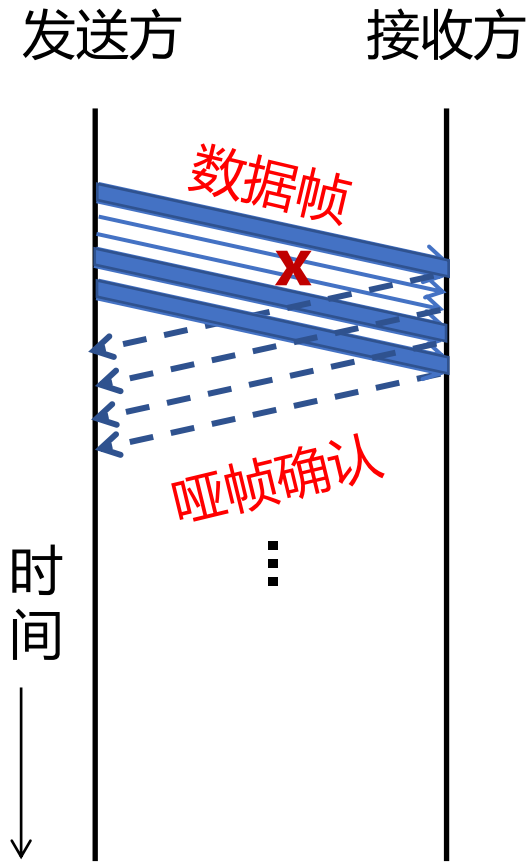
滑动窗口协议—协议基本思想

- 流水线技术的需求

  - 帧丢失？ACK丢失？
  - 重传哪个帧？
  - 流控：未确认的帧数？

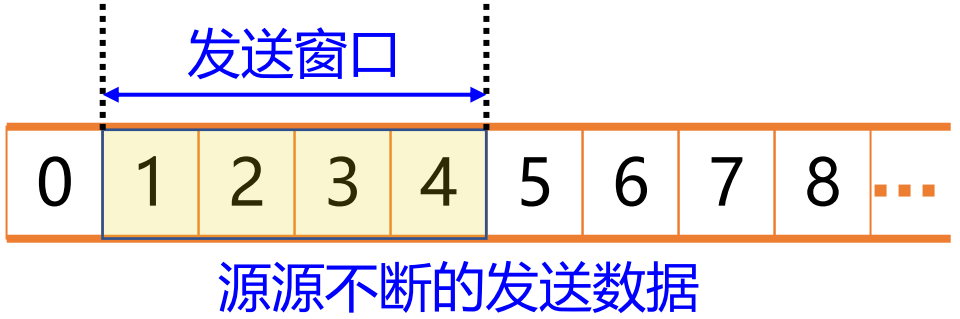
  - 发送方：要暂存哪些帧以便可能的重传
  - 接收方：如何能向网络层按序提交数据
  - 双方：允许发送方发多少帧以不淹没接收方

- 流水线技术的实现方式
  - 滑动窗口协议（本节的三个协议P4/P5/P6）
  - 都能在实际（非理想）环境下正常工作
  - 区别仅在于效率、复杂性和对缓冲区的要求



滑动窗口协议—协议基本思想

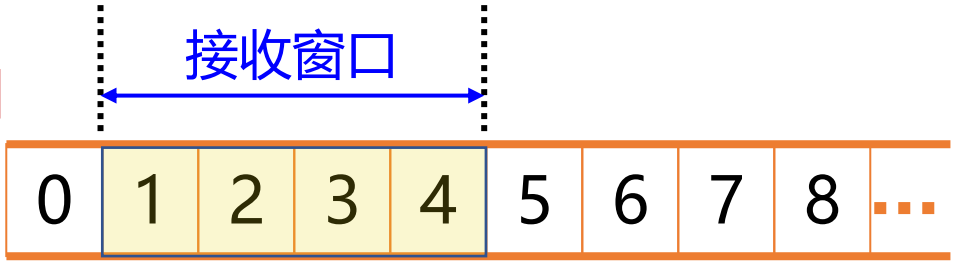
- 发送者面临的问题
  - 发了哪些帧，需要重传时怎么办？
- 发送窗口
  - 发送端始终保持一个已发送但尚未确认的帧的序号表，称为发送窗口
  - 发送窗口的上界表示要发送的下一个帧的序号，下界表示未得到确认的帧的最小序号
  - 发送窗口大小 = 上界 - 下界，大小可变
  - 发送端每发送一个新帧，帧序号取上界值，上界加1
  - 每接收到一个确认序号 = 发送窗口下界的正确响应帧，下界加1



滑动窗口

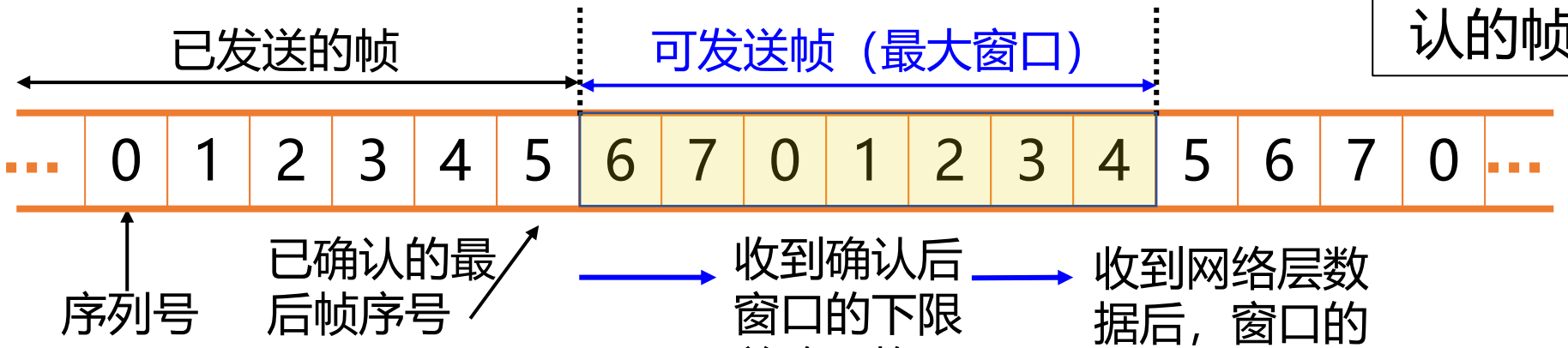
滑动窗口协议—协议基本思想

- 接收者面临的问题
  - 保序：接收方需按照发送方网络层发送的顺序，将数据提交给上层
  - 避免被发送方淹没：需要告诉发送方，能够发送多少数据
- 接收窗口
  - 接收端有一个接收窗口，但不一定与发送窗口相同
  - 接收窗口容纳允许接收的信息帧，落在窗口外的帧均被丢弃
  - 接收窗口的上界表示允许接收的最大序号，下界表示希望接收的最小序号
  - 序号等于下界的帧被正确接收，并产生一个确认帧，上界、下界都加1
  - 接收窗口大小通常保持不变

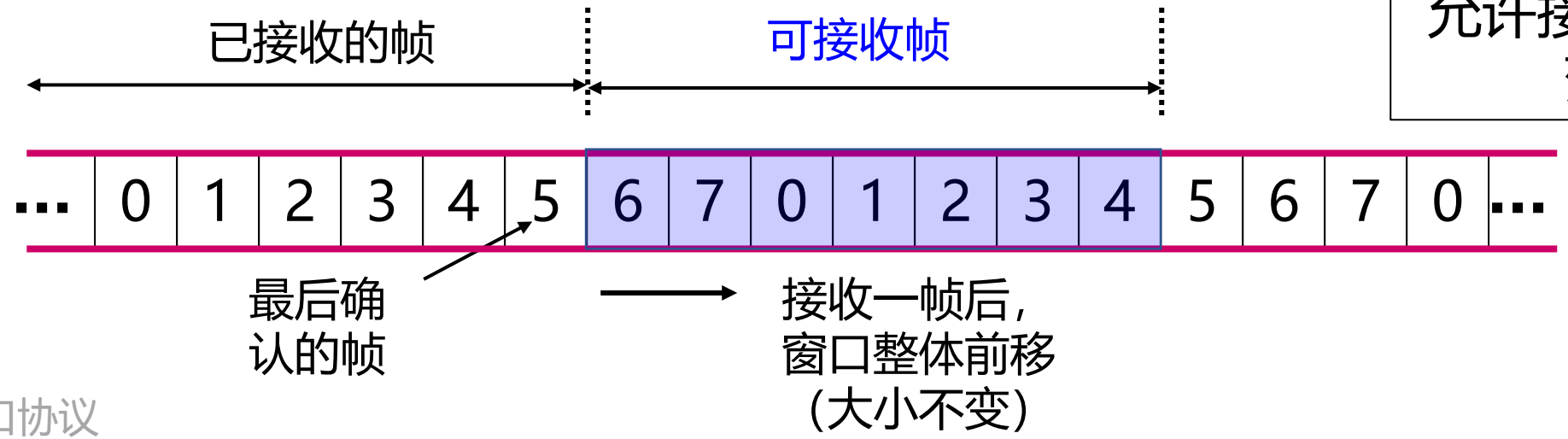


滑动窗口协议—协议基本思想

- 发送窗口 (例如大小7)



- 接收窗口(如大小7)



一比特滑动窗口协议P4

- 设计目标
  - 通信双方互发数据
  - 信道条件：全双工
  - 最最简单的窗口机制（即窗口大小为1比特）
- 哑帧确认->捎带确认（piggybacking）
  - 将确认帧与反向数据帧合并，可以暂时延迟待发确认，以便附加到下一个待发数据帧
  - 优点：充分利用信道带宽，减少帧数目，减少“帧到达”中断

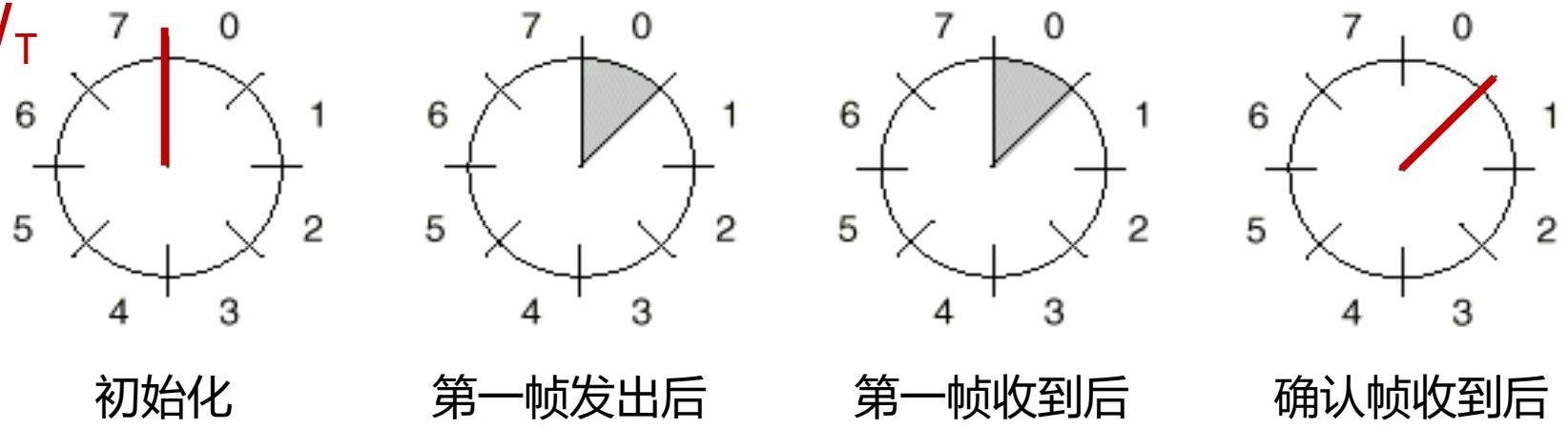
呼唤
循序渐进的
设计协议

发送方和接收方
代码二合一

一比特滑动窗口协议P4

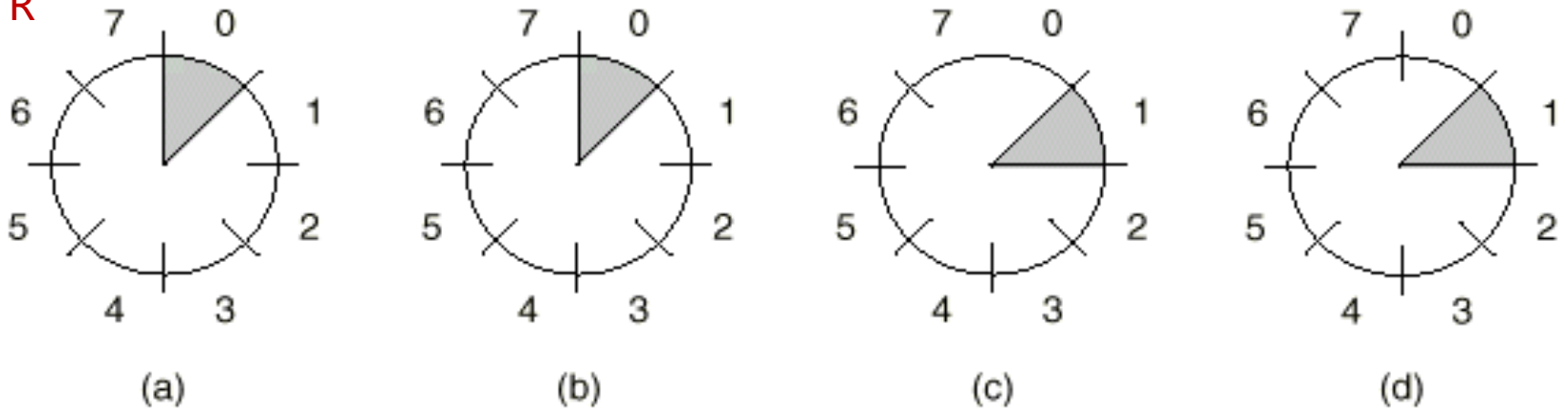
- 发送窗口与接收窗口(窗口大小1)

发送窗口W_T



初始化　　第一帧发出后　　第一帧收到后　　确认帧收到后

接收窗口W_R



(a)　　(b)　　(c)　　(d)

一比特滑动窗口协议P4

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

捎带确认

- 两个独立的序号序列
  - r.seq为对端序号、本地确认
  - s.seq为本地序号、对端确认

```
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);
         }
      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);
}
```

一比特滑动窗口协议P4

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。

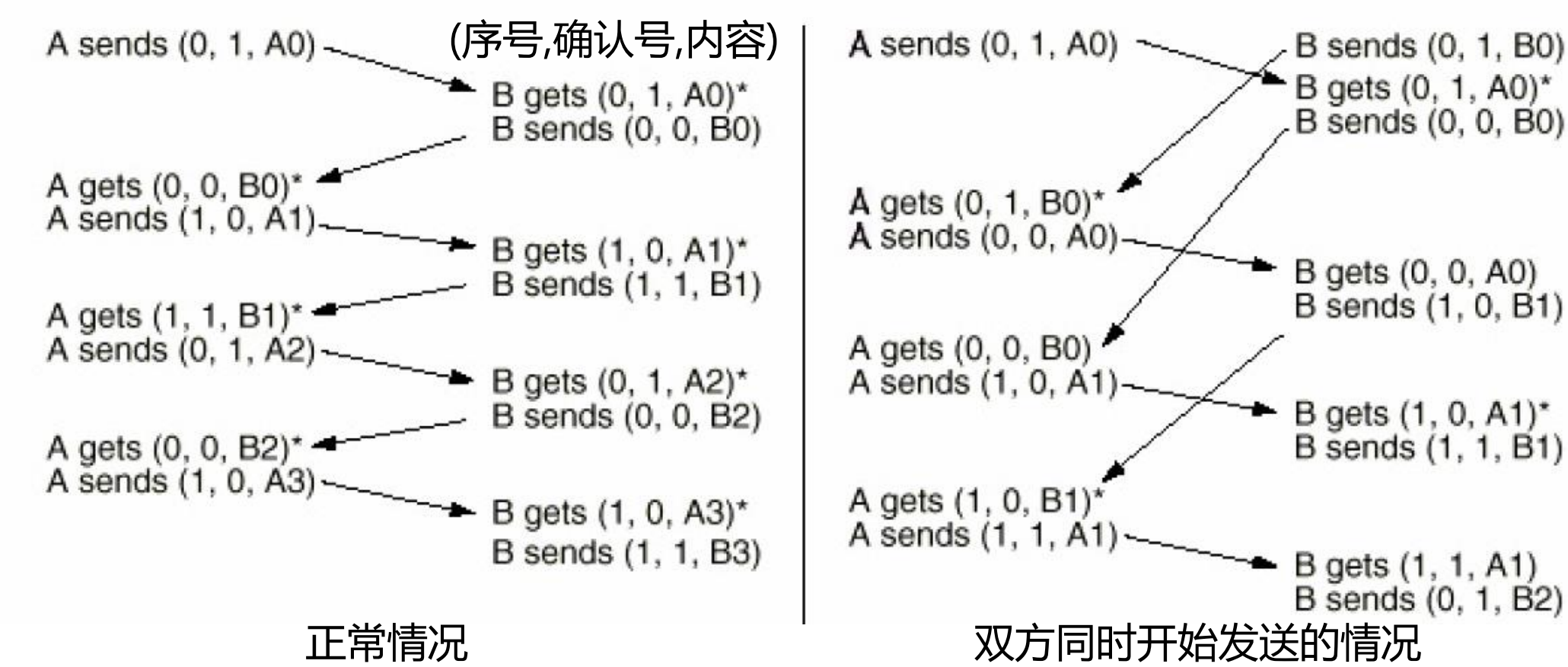
- 窗口大小：N = 1，序号取值范围：0，1
- 可进行数据双向传输，信息帧中可含有确认信息（piggybacking技术）
- 信息帧中包括两个序号域：发送序号和确认序号（已经正确收到的帧的序号）

```
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);
         }
      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);
}
```

r.seq为对端序号、本地确认
s.seq为本地序号、对端确认

一比特滑动窗口协议P4

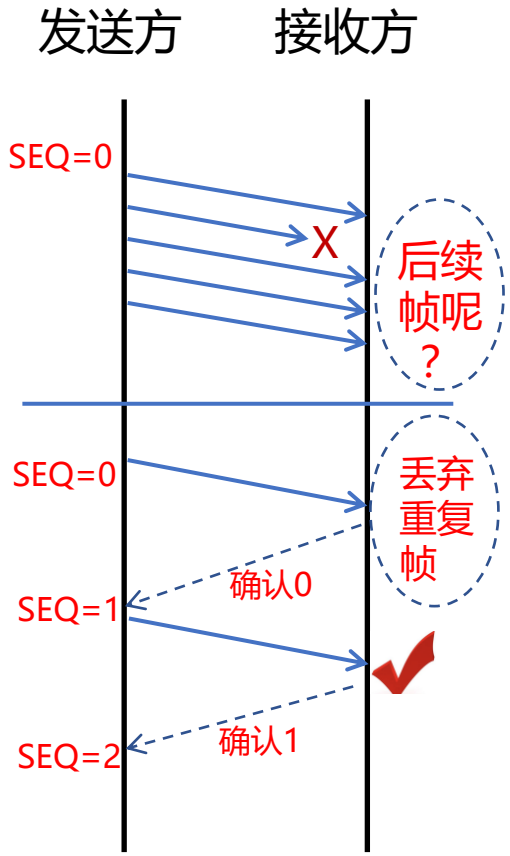
- 存在的问题
  - 若双方同时开始发送，则会有一半重复帧



正常情况　　双方同时开始发送的情况

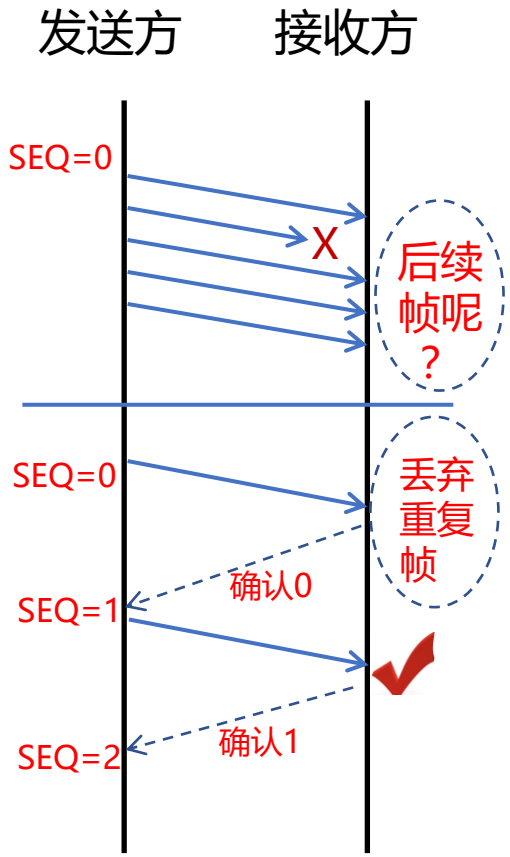
一比特滑动窗口协议P4

- 停止-等待机制降低了信道利用率
  - 假如将链路看成是一根管道，数据是管道中流动的水
  - 能否将水充满管道？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
- 发明新协议？
  - 流水线协议：连续发送多帧后再等待确认
  - 窗口机制：允许发送方在没收到确认前连续发送多个帧？2倍时延带宽积的数据量？
  - 出错怎么办：接收方 or 发送方？



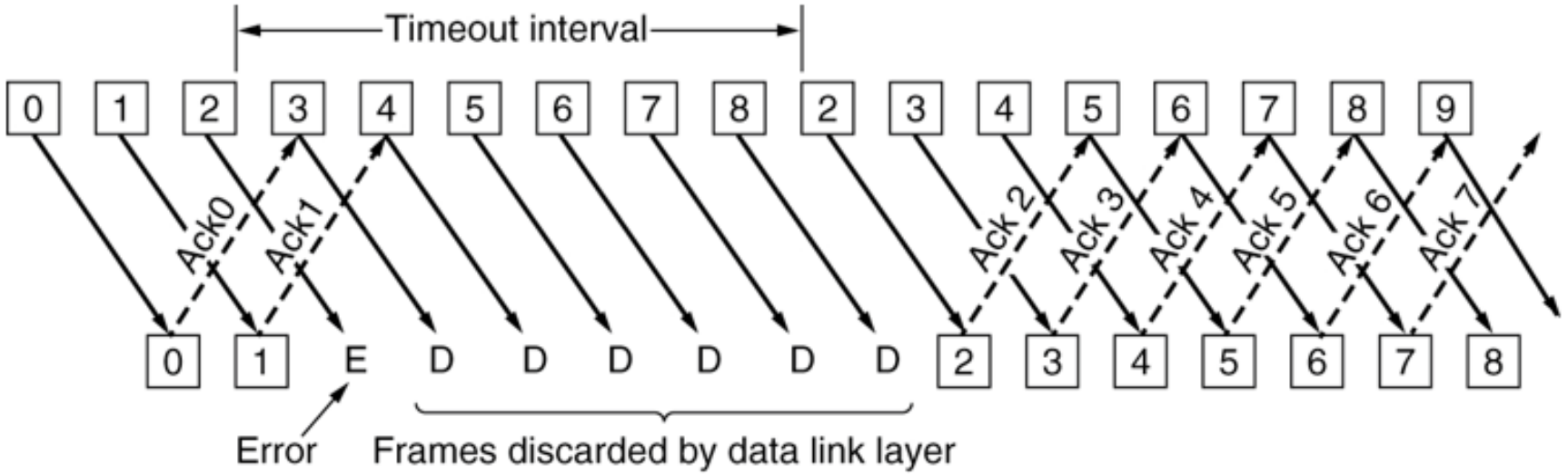
回退N协议P5—设计目标与基本思路

- 设计目标
  - 目标1：向上层按序提交（不能提交乱序或错误内容）
  - 目标2：实现流水线的发送机制，提高信道利用率
  - 简化设计：接收窗口为1
- 出错全部重发
  - 由于接收窗口为1，只能按顺序接收帧
  - 当接收端收到出错帧或乱序帧时，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
- 优缺点
  - 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P5—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
- 滑动窗口长度
  - 出错全部重发时，若帧序号为n位，接收窗口W_R=1，发送窗口 $W_T \leq 2^n - 1$

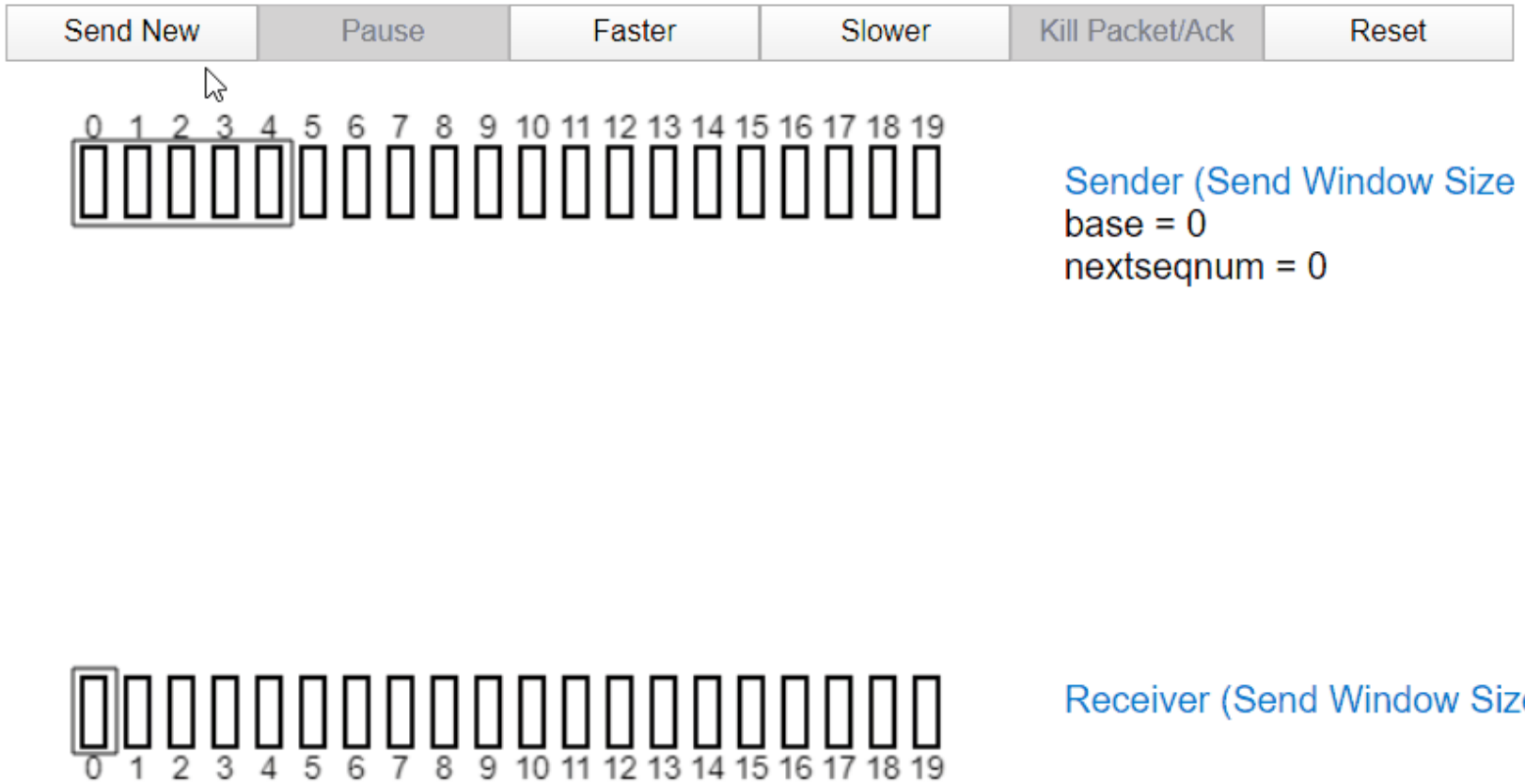


回退N协议P5—协议原理分析

- 正常情况下的GBN

在回退N步协议中，允许发送方发送多个分组而不用等待确认

正常情况下，发送、接收窗口会随着帧的确认状态随包右移

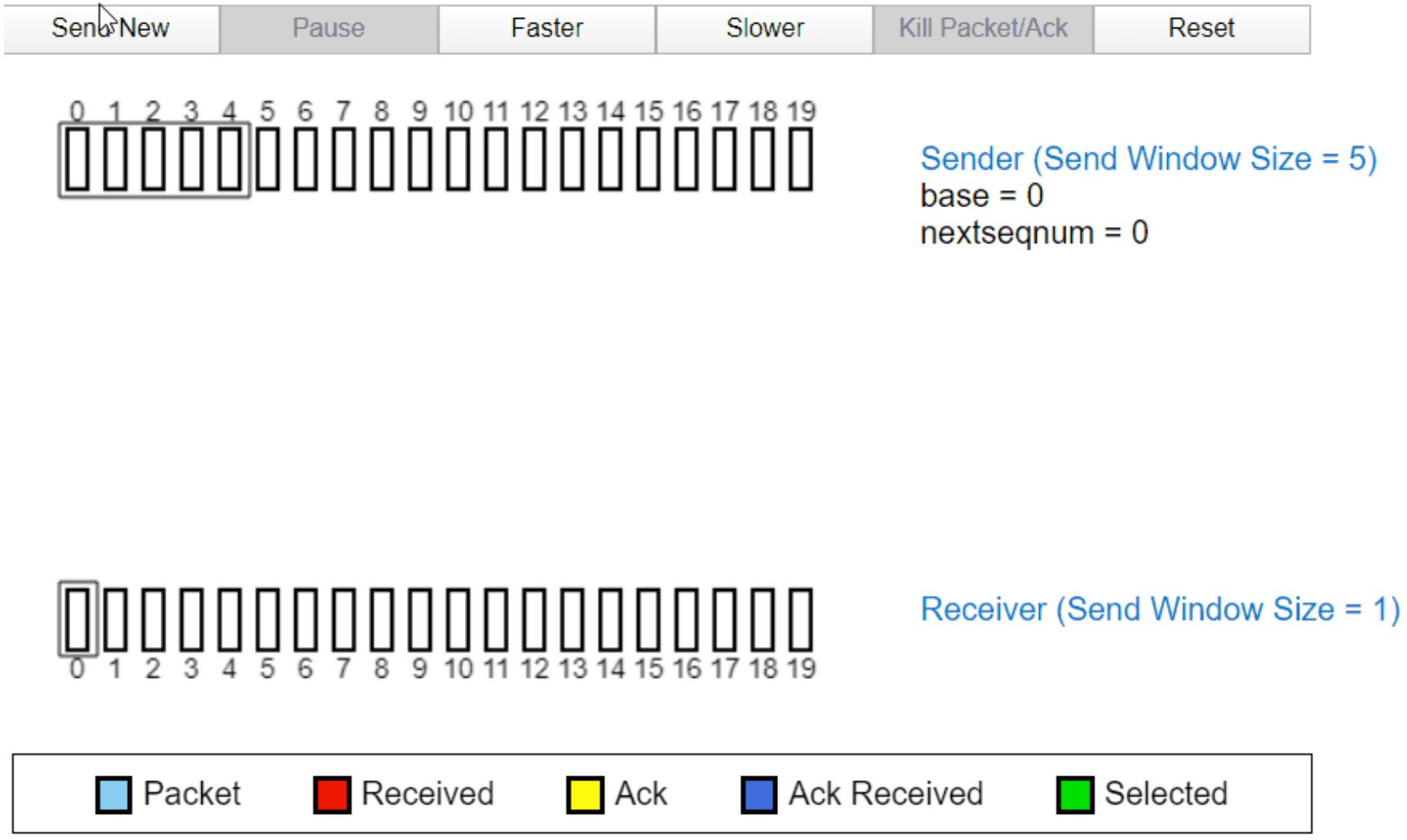


回退N协议P5—协议原理分析

- 发送过程中数据包丢失

当发送过程中的2号包发生丢失时：

- 发送方没有收到接收方的ACK2，于是后面发送的ACK3，ACK4全部变成了ACK1，代表接收方因为丢失了分组2，所以分组3和分组4都被丢弃，全部返回ACK1
- 经过一段时间后，定时器确认超时没有收到ACK2/3/4，所以发送方将重新发送



回退N协议P5—协议原理分析

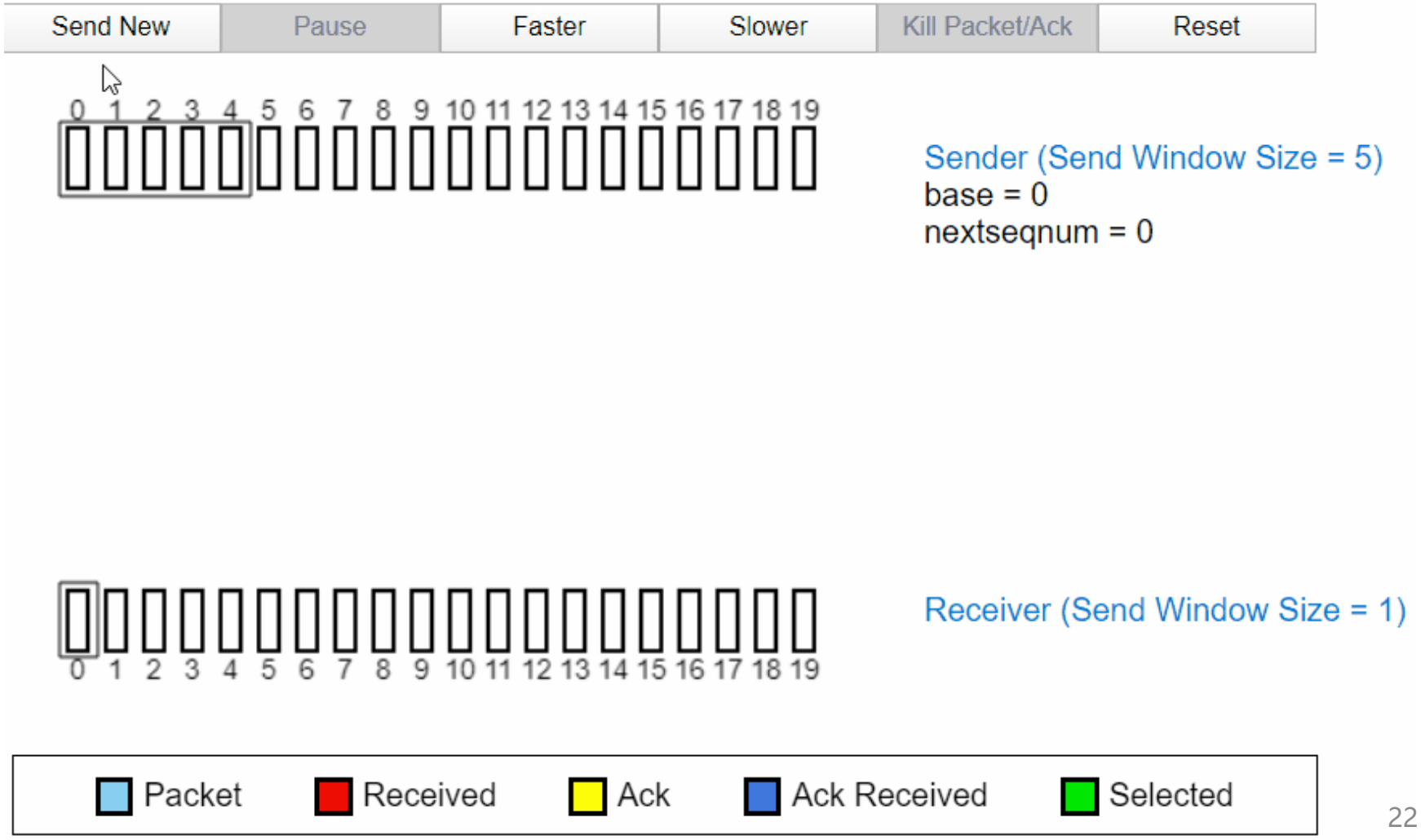
- 接收方返回的ACK丢失

累计确认：

- 对收到的分组不必逐个发送确认，而对按序到达的最后一个分组发送确认

假设ACK2发生丢失：

- 如果接收方没有收到分组2，则后面返回的都是ACK1
- 因返回ACK3和ACK4，所以发送方可判断接收方已收到分组2



回退N协议P5—协议的实现分析

- 实现要点
  - 发送方为支持重传，发送方需要缓存多个分组，即增加序号范围
- 两个窗口
  - 发送窗口：发送方维持一组连续允许发送的帧序号，不断向前滑动
  - 接收窗口：接收方维持一个允许接收的帧序号，不断向前滑动
- 发送方的三个功能
  - 上层发送数据：检测有没有可以使用的序号，如果有就发送
  - 收到ACK：对n号帧的确认，采用累积确认的方式
  - 超时事件：如果出现超时，就重传已发送未确认的所有分组

回退N协议P5—协议的实现分析

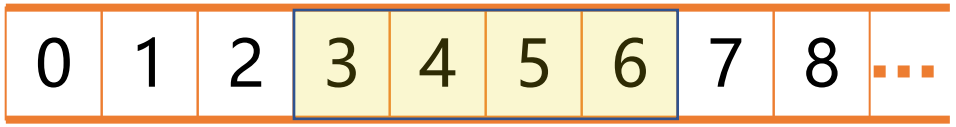
发送方：

1. 窗口尺寸：$1<W_T\leq 2^n-1$，最多连续发送窗口中的W_T个PDU
2. 窗口滑动：收到期望的ACK(k)：窗口底部移到PDU(k)，最大窗口顶部向前移动
3. 窗口滑动后，发送新进入窗口的PDU，始终保持窗口里最多有W_T个PDU未确认
4. 超时重发： 超过T未收到期望的ACK，重发窗口中的PDU（回退整个窗口）
5. 超次数失败：超过最大重发次数N_{max}仍无正确应答

回退N协议P5—协议的实现分析

接收方：

1. 窗口尺寸：$W_R=1$
2. 按序接收：按照PDU编号依序接收，出错、乱序PDU一律丢弃
3. 确认含义：ACK(k)表示对k-1及以前各编号的PDU的确认，同时期望接收第k号PDU
4. 确认策略：按序到达的PDU可立即确认，也可延迟确认(收到多帧后一起确认)
5. 但出错或乱序的PDU，只能反向确认NACK(k)（期望接收k号PDU）或不应答，在正确接收到PDU(k)前不能发送ACK(k+1)等



回退N协议P5—协议的实现分析

实现基本过程如下：

1. 初始化。ack_expected = 0 (此时处于发送窗口的下沿)；next_frame_to_send = 0，frame_expected = 0 (初始化正在发送的帧和期待的帧序号)；nbuffered = 0 (进行发送大小初始化)
2. 等待事件发生（网络层准备好，帧到达，收到坏帧，超时）
3. 如果事件为网络层准备好，则执行以下步骤。从网络层接收一个分组，放入相应的缓冲区；发送窗口大小加1；使用缓冲区中的数据分组、next_frame_to_send和frame_expected构造帧，继续发送；next_frame_to_send加1；跳转（7）

```
while(1) {
   wait_for_event(&event);
   switch(event){
       case network_layer_ready:
           from_network_layer(&buffer[next_frame_to_send]);
          nbuffered=nbuffered+1;
          send_data(next_frame_to_send, frame_expected,
buffer);
          inc(next_frame_to_send);
          break;
```

回退N协议P5—协议的实现分析

4. 如果事件为帧到达，则从物理层接收一个帧，则执行以下步骤。首先检查帧的seq域，若正是期待接收的帧（seq = frame_expected），将帧中携带的分组交给网络层，frame_expected加1；然后检查帧的ack域，若ack落于发送窗口内，表明该序号及其之前所有序号的帧均已正确收到，因此终止这些帧的计时器，修改发送窗口大小及发送窗口下沿值将这些帧去掉，继续执行步骤（7）

```
case frame_arrival:
    from_physical_layer(&r);
    if(r.seq==frame_expected) {
        to_network_layer(&r.info);
        inc(frame_expected);
        }
    while(between(ack_expected, r.ack, net_frame_to_send)){
        nbuffered=nbuffered-1;
        stop_timer(ack_expected);
        inc(ack_expected);
        }
    break;
```

回退N协议P5—协议的实现分析

5. 如果事件是收到坏帧，继续执行步骤（7）
6. 如果事件是超时。使next_frame_to_send = ack_expected，从发生超时的帧开始重发发送窗口内的所有帧，然后继续执行步骤（7）
7. 若发送窗口大小小于所允许的最大值（MAX-SEQ），则可继续允许网络层发送，否则则暂停网络层发送

```
  case cksum_err: ;
      break;
  case timeout:
      next_frame_to_send=ack_expected;
      for(i=1; i<=nbuffered; i++){
          send_data(next_frame_to_send, frame_expected, buffer);
          inc(next_frame_to_send);
          }
  }
  if(nbuffered<MAX_SEQ)
      enable_network_layer();
  else
      disable_network_layer();
}
```

回退N协议的缺点：
重传所有已发送未确认的分组？

选择重传协议P6—协议设计思想

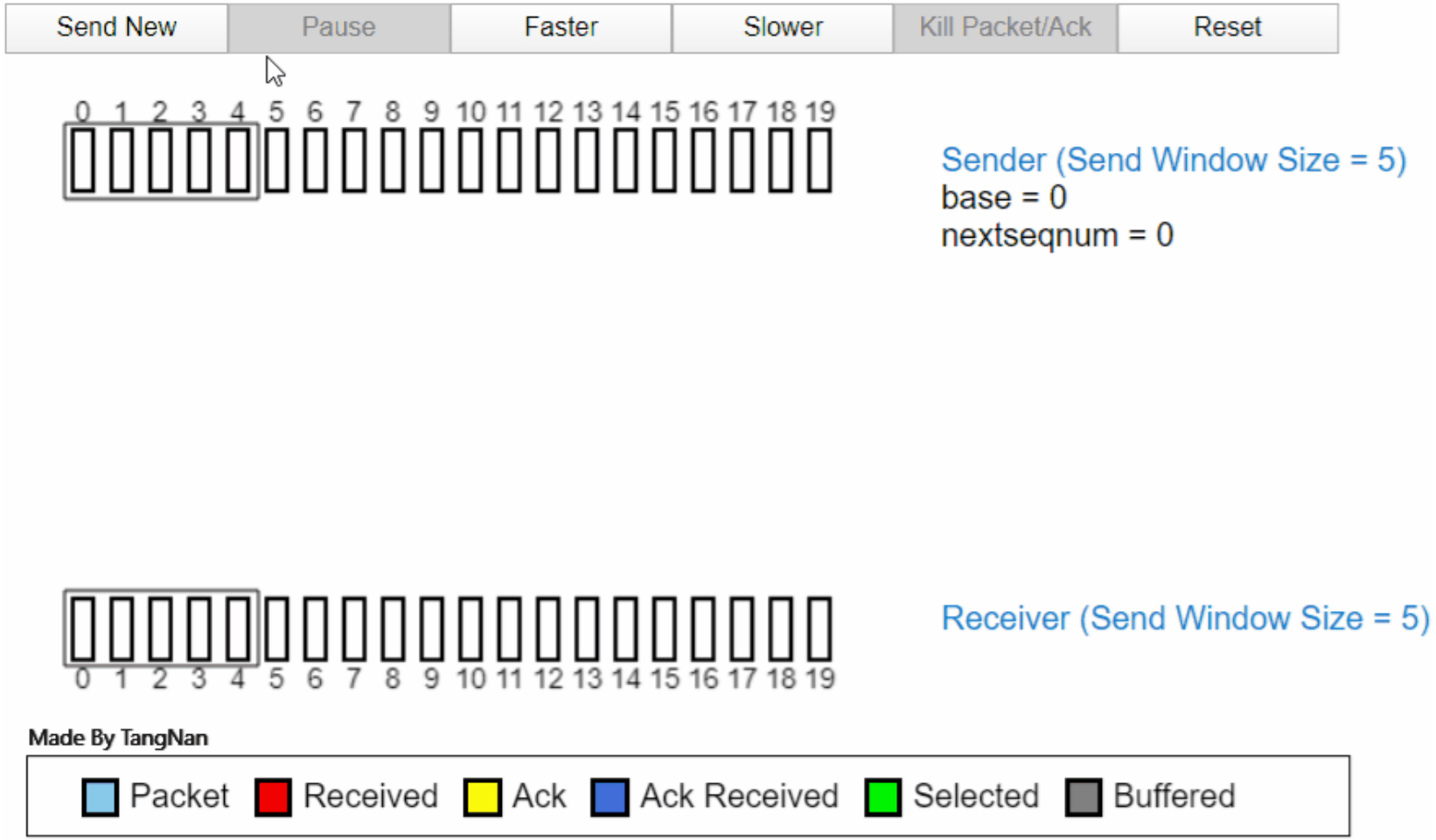
- 设计目标
  - 目标：能否仅重传出错的帧，不重传后续可能正确的帧
- 设计思想
  - 接收方需要暂存出错帧之后的数据帧，即接收窗口需要大于1
  - 按序交付：如果落在接收窗口内的帧从未接收过，那么存储起来，等比它序列号小的所有帧都正确接收后，按次序交付给网络层
  - 乱序接收：接收端收到的数据包的顺序可能和发送的数据包顺序不一样，因此在数据包里必须含有顺序号来帮助接收端进行排序
- 优缺点
  - 缺点：在接收端需要占用一定容量的缓存

选择重传协议P6——协议原理分析

- 正常情况下SR

同样，在选择重传协议中，也允许发送方发送多个分组而不用等待确认
与回退N步不同，接收窗口大于1

正常情况下，随着帧的确认状态，发送、接收两个窗口都随包右移

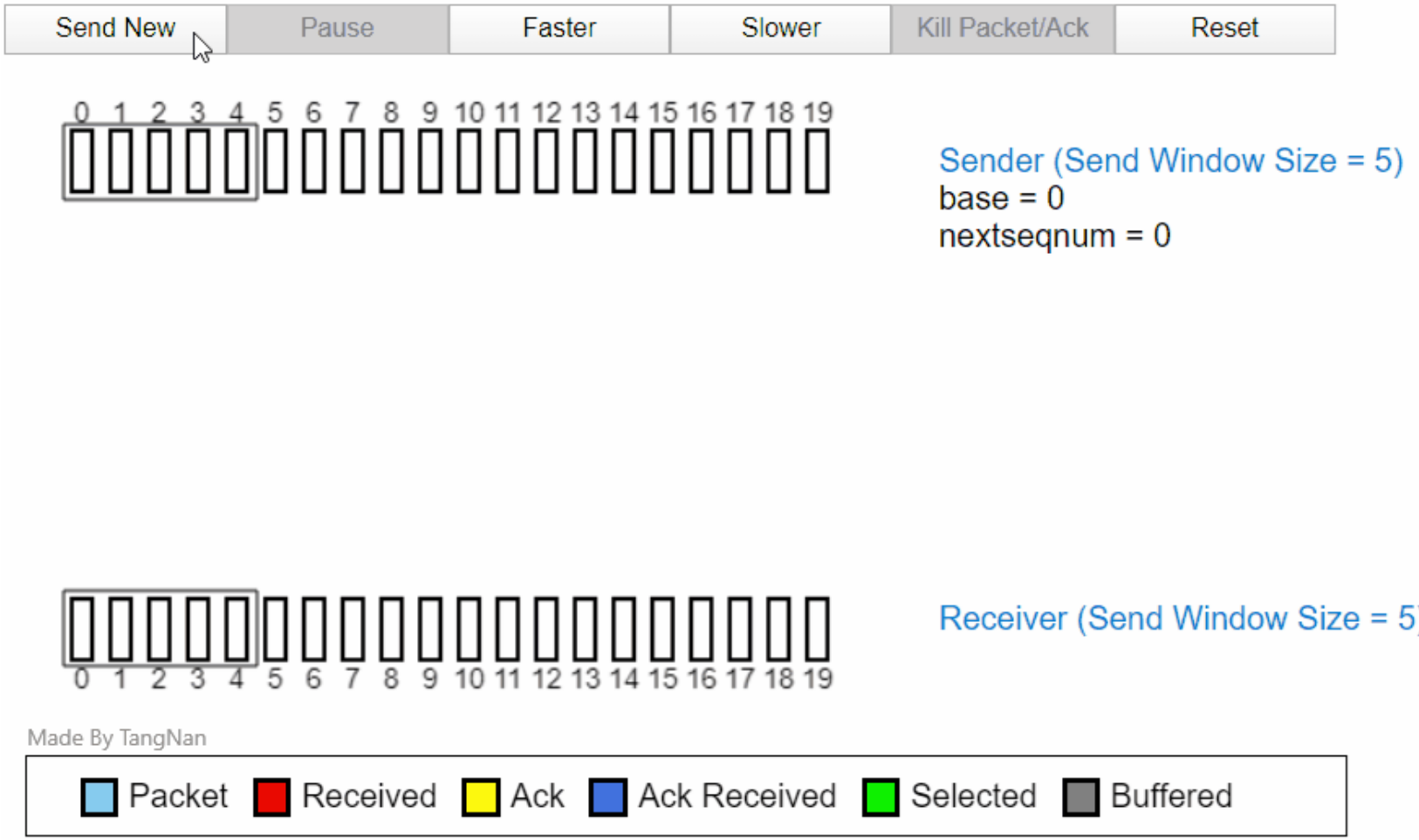


选择重传协议P6——协议原理分析

- 发送过程中发生丢包

当发送过程中的2号包发生丢失时：

- 与GNB不同的是，接收方在没有收到分组2的情况下，依然返回了ACK3, ACK4
- 当ACK1返回以后，分组5，分组6就已经可以发送
- 接收方缓存了分组3456，等待分组2的计时器超时后，分组2将重新发送

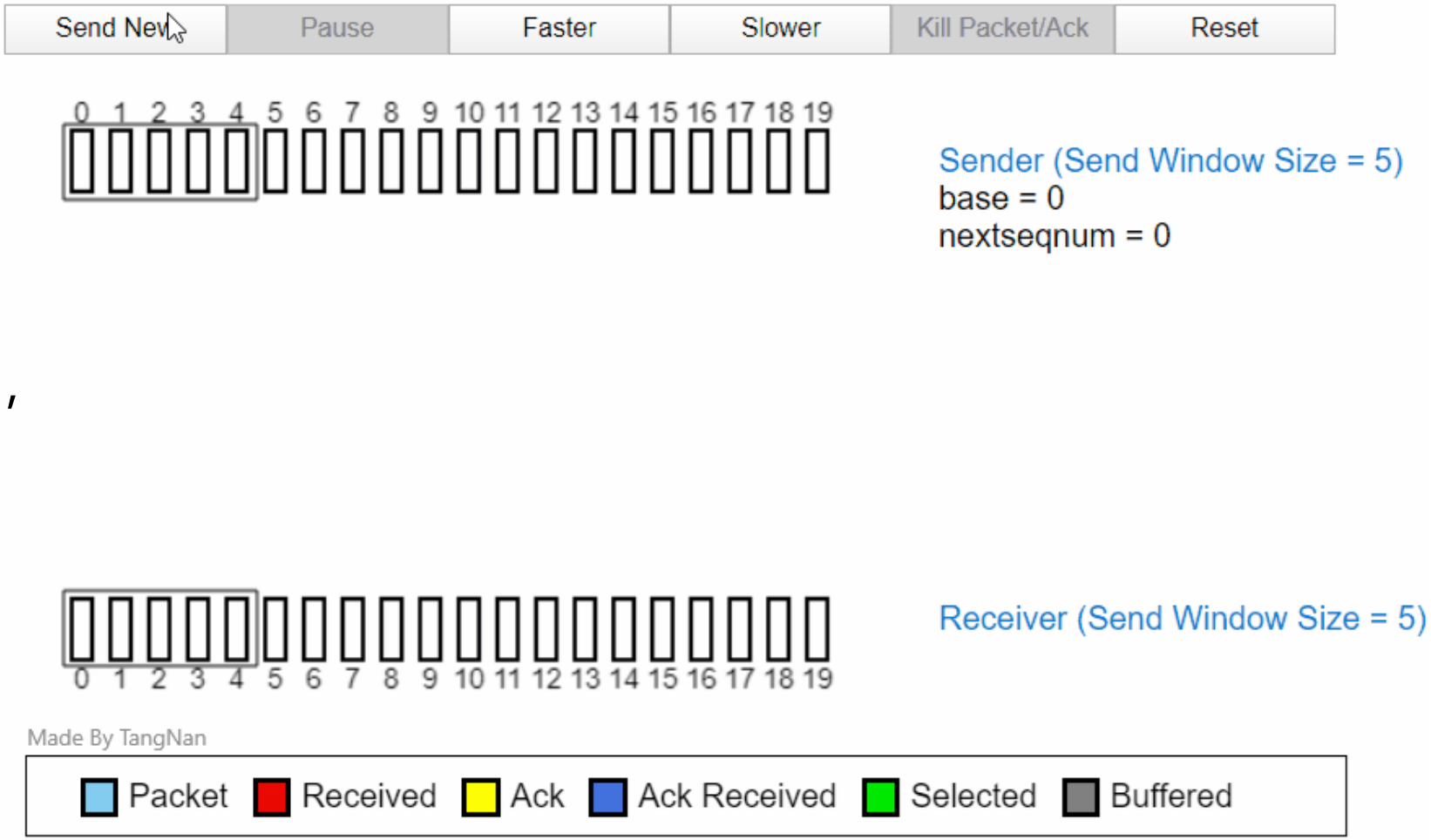


选择重传协议P6——协议原理分析

➢ 确认过程发生丢包

当ACK2丢失时：

- 发送方的分组已经由0-4到了2-6,在最后2-6的窗口中
- 分组2因为ACK2被丢失，然后在发送计时器超时后会被重新发送
- 如果在接收过程中有丢失发生，选择重传SR的效率是不如回退N步GBN的



选择重传协议P6—协议实现分析

- 实现要点
  - 与GBN不同，P6是给每一个PDU设置定时器，发送端只重传出错帧PDU
  - 接收端缓存所收到的乱序PDU，当前面PDU到达后一起按序提交上层
- 两个窗口
  - 发送窗口：发送方维持一组连续帧序号（以便重传）
  - 接收窗口：接收方维持一组连续的允许接收帧序号（以不被淹没）
- 发送方必须响应的三件事
  - 上层的调用：检测有没有可以使用的序号，如果有就发送
  - 收到ACK：如果收到的是最小序号的ACK，窗口滑动。如果收到其他序号的ACK，进行标记
  - 超时事件：每个PDU都有定时器，哪个超时重传哪个

选择重传协议P6——协议原理分析

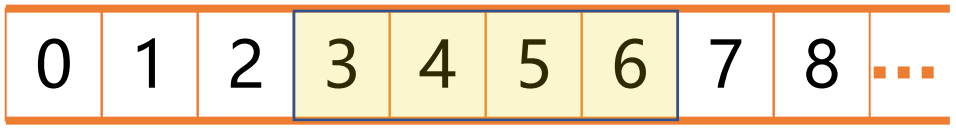
- 快速重传优化：否认确认帧NAK
  - 在发送过程中，如果一个数据帧计时器超时，就认为该帧丢失或者被破坏
  - 若发送方发出连续的若干帧后，收到对其中某一帧的否认确认帧NAK，或某一帧的定时器超时，则只重传该出错帧或计时器超时的数据帧
  - 接收端只把出错的帧丢弃，其后面的数据帧保存在缓存中，并向发送端回复NAK；发送端接收到NAK时，只重传出错的帧

选择重传协议P6—协议实现分析

- 实现基本过程与回退N协议基本类似，其中

发送方：

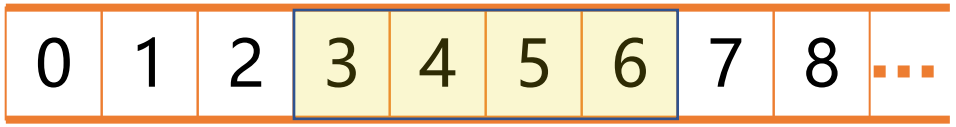
1. 窗口尺寸：$1<W_T\leq 2^{n-1}$，最多连续发送窗口中的W_T个PDU
2. 窗口滑动：与回退N帧协议相同
3. 选择重发：收到NAK (k)，重发PDU (k)
4. 超时重发：超过T未收到期望的ACK，重发当前超时未应答的PDU
5. 超次数失败：超过最大重发次数N_{max}仍无正确应答，报告上层失败



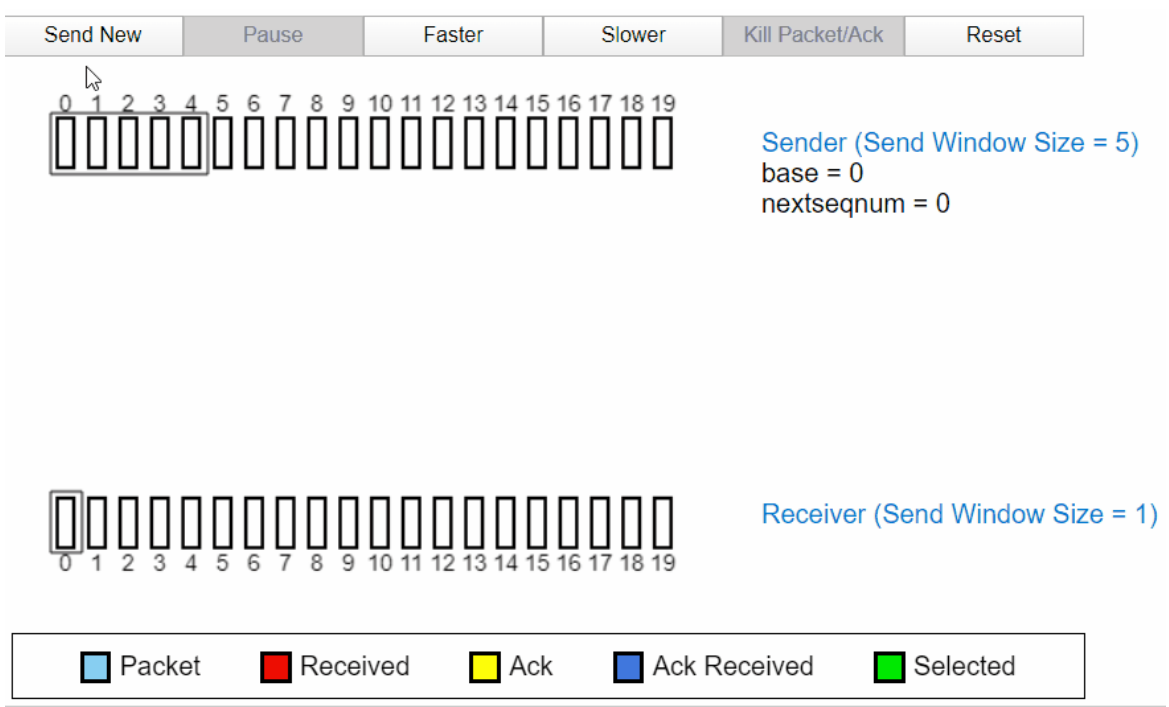
选择重传协议P6—协议实现分析

接收方：

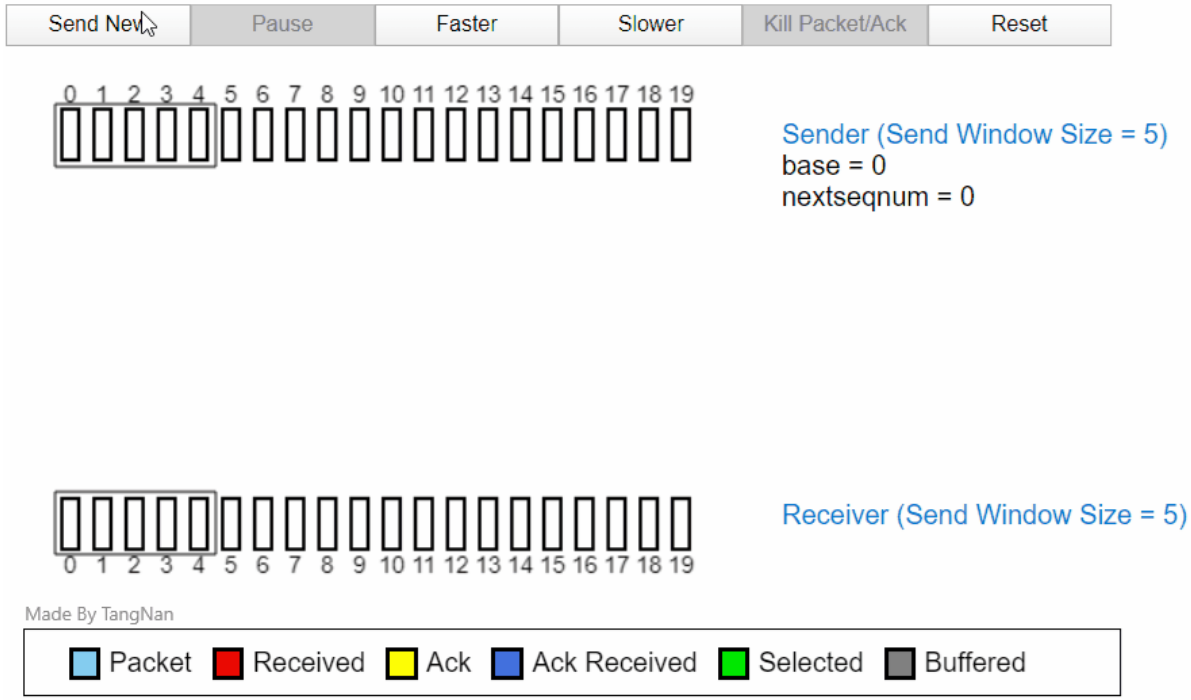
1. 窗口尺寸：$1<W_R\leq 2^{n-1}$
2. 窗口内接收：窗口内的PDU全部接收，存储出错的后续PDU；窗口外的PDU一律丢弃
3. 窗口滑动：窗口底部数据按序上交，窗口向前滑动
4. 确认策略：按序到达的PDU可立即确认，也可延迟确认(收到多帧后一起确认)ACK (k)；出错用否定性确认NAK(k)（期望重发k号PDU)



累计确认和选择重传的比较



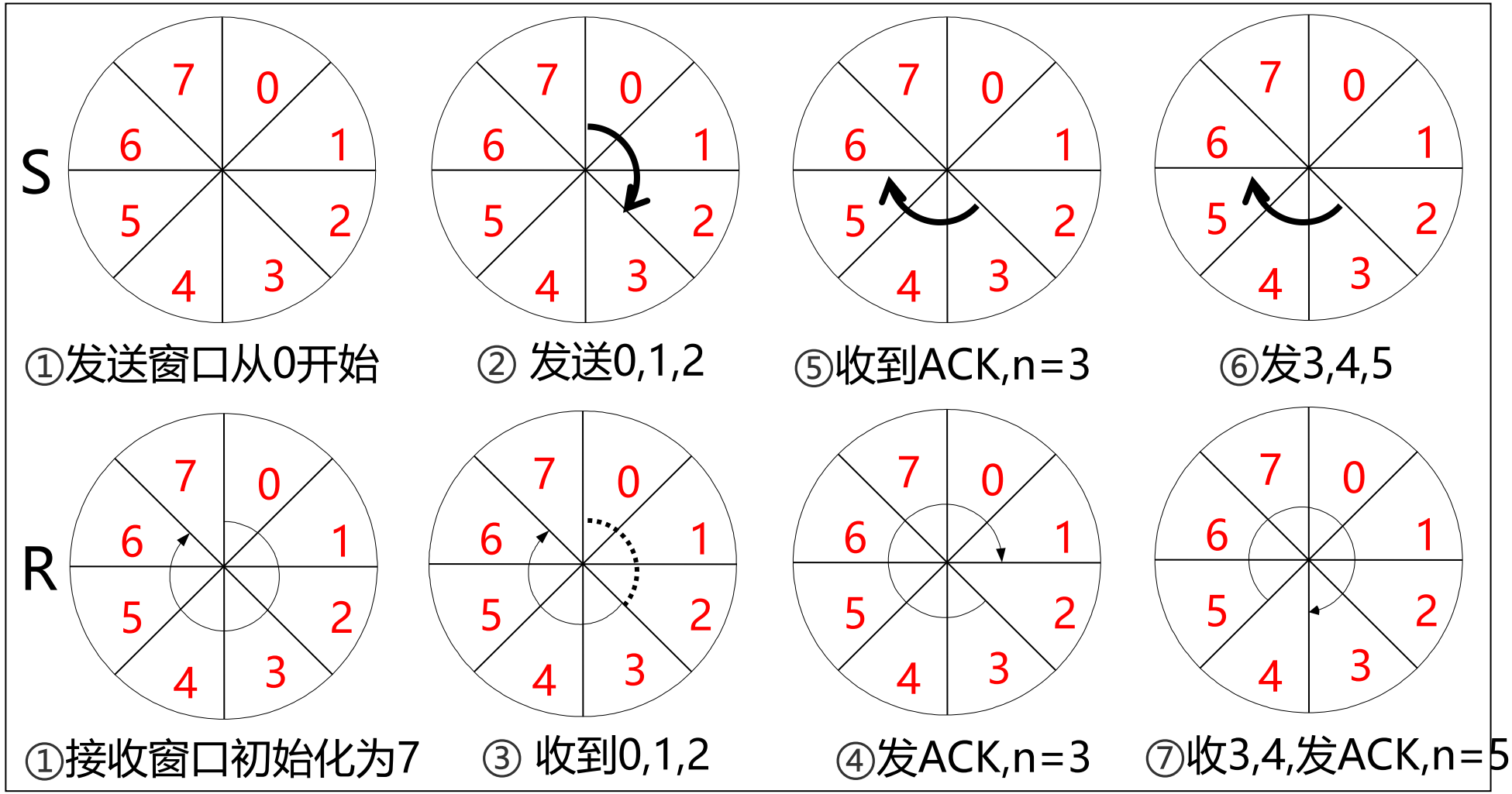
累计确认在ack丢失时效率高



选择重传在ack丢失时效率低

选择重传协议P6—协议的实现分析

- 发送窗口(窗口大小3)与接收窗口(窗口大小7)



选择重传协议P6—协议的实现分析

- 事件驱动
  - Network_layer_ready（内部事件）
    - 发送帧（帧类型，帧序号，确认序号，数据）
  - Timeout （内部事件）：选择重传
  - Ack_timeout （内部事件）：发送确认帧ACK
  - Frame_arrival （外部事件）
    - 若是数据帧，则检查帧序号，落在接收窗口内则接收，否则丢弃；不等于接收窗口下界还要发NAK
    - 若是NAK，则选择重传
    - 检查确认序号，落在发送窗口内则移动发送窗口，否则不做处理
  - Cksum_err （外部事件）：发送NAK

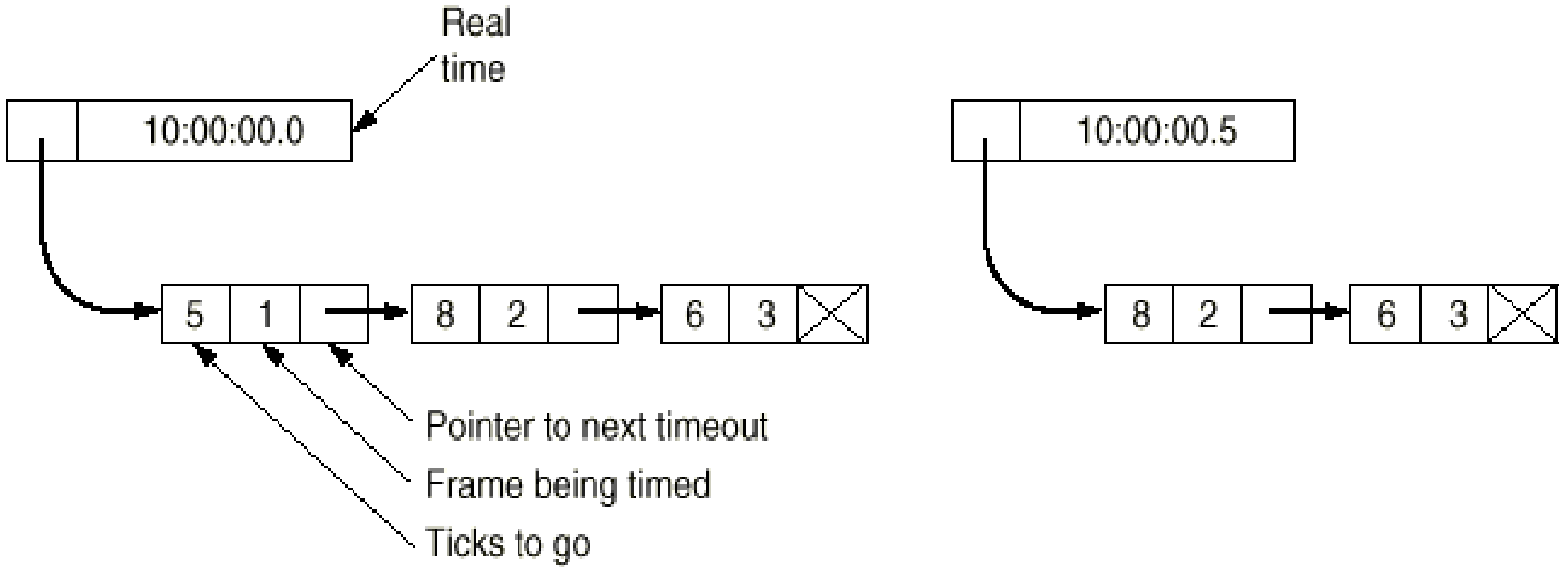
选择重传协议P6—协议的实现分析

- 计时器处理
  - 启动：发送数据帧时启动
  - 停止：收到正确确认时停止
  - 超时则产生timeout事件
- Ack计时器处理
  - 启动：收到帧的序号等于接收窗口下界或已经发过NAK时启动
  - 停止：发送帧时停止
  - 超时则产生ack_timeout事件

大量计时器
谁来管理?

选择重传协议P6—协议的实现分析

- 计时器链表
  - 由于有多个未确认帧，需要设多个计时器



选择重传协议P6—协议实现分析

实现基本过程如下：

1、初始化。ack_expected = 0 (此时处于发送窗口的下沿)；next_frame_to_send = 0，frame_expected = 0（初始化正在发送的帧和期待的帧序号）；nbuffered = 0（进行发送窗口大小初始化）；

2、等待事件发生（网络层准备好，帧到达，收到坏帧，超时，确认超时）。

3、如果事件为网络层准备好，则执行以下步骤。从网络层接收一个分组，放入相应的缓冲区；发送窗口大小加1；使用缓冲区中的数据分组、next_frame_to_send和frame_expected构造帧，继续发送；next_frame_to_send加1；跳转（8）；

4、如果事件为帧到达，则从物理层接收一个帧，则执行以下步骤。首先检查帧的kind域，若是数据包，再检查seq域，若不是期待接收的帧（seq ！ = frame_expected）并且不是nak，则发送nak，否则开启定时器；如果seq落入接收窗口之内并且没有被接收，则接收帧，将帧中携带的分组交给网络层，frame_expected、too_far加1,开启确认定时器；若kind为nak则重新发送数据。最后检查帧的ack域，若ack落于发送窗口内，表明该序号及其之前所有序号的帧均已正确收到，因此终止这些帧的计时器，修改发送窗口大小及发送窗口下沿值将这些帧去掉，继续执行步骤（8）；

5、如果事件是收到坏帧，如果no_nak为真，则发送nak帧，然后继续执行步骤（8）。

6、如果事件是发送超时，即：next_frame_to_send = oldest_frame，则重发超时帧，然后继续执行步骤（8）。

7、如果事件是确认超时，则重发超时的确认帧，然后继续执行步骤（8）。

8、若发送窗口大小小于所允许的最大值（MAX_SEQ），则可继续向网络层发送，否则则暂停继续向网络层发送，同时返回互步骤（2）等待。

选择重传协议P6—协议的实现分析

核心代码

```
while(1) {
    wait_for_event(&event);   /* 包括5种情况 */
    switch(event) {
       case network_layer_ready:  /* 从网络层接收数据，传输新帧 */
          from_network_layer(&out_buffer[next_frame_to_send%NR_BUFS]);
          nbuffered=nbuffered+1;
          send_frame(data,next_frame_to_send, frame_expected, out_buffer);
          inc(next_frame_to_send);
          break;
      case frame_arrival:  /* 数据帧或控制帧到达 */
          from_physical_layer(&r);
          if (r.kind==data) {
             if ((r.seq!=frame_expected) && no_nak)
                send_frame(nak, 0, frame_expected, out_buf);  else  start_ack_timer();
             if (between(frame_expected, r.seq,too_far) &&arrived[r.seq%NR_BUFS]==false)) {
                arrived[r.seq%NR_BUFS]=true;
                in_buf[r.seq%NR_BUFS]=r.info;
                while(arrived[frame_expected%NR_BUFS]){
                   to_network_layer(&in_buf[frame_expected%NR_BUFS]);
                   no_nak=true;
                   arrived[frame_expected%NR_BUFS]=false;
```

选择重传协议P6—协议的实现分析

核心代码

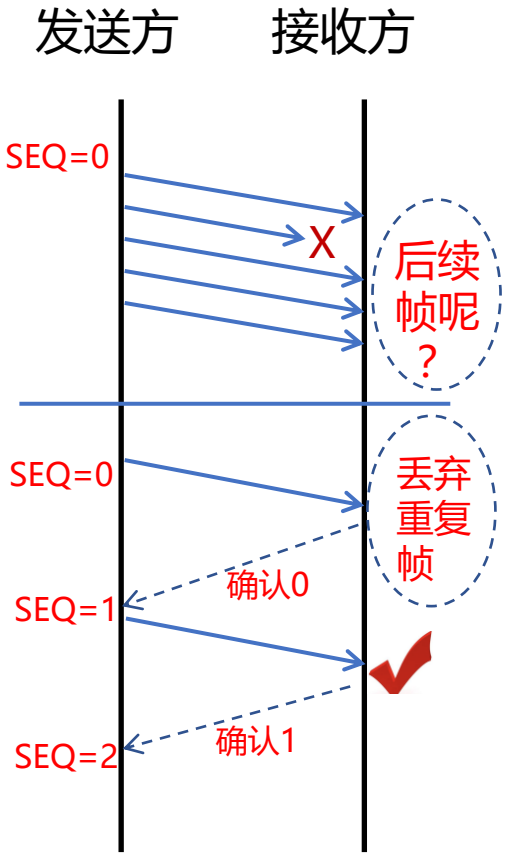
```
                inc(frame_expected);
                inc(too_far);
                start_ack_timer();
                }
            }
        }
    if((r.kind==nak) && between(ack_expected ,(r.ack+1)%(MAX_SEQ+1), next_frame_to_send))
        send_frame(data,(r.ack+1)%(MAX_SEQ+1),frame_expected, out_buf);
    while(between(ack_expected, r.ack, next_frame_to_send)) {
        nbuffered=nbuffered-1;
        stop_timer(ack_expected%NR_BUFS);
        inc(ack_expected);
        }
    break;
  case cksum_err: if(no_nak) send_frame(nak, 0, frame_expected, out_buf); break;  /* 帧出错 */
  case timeout: send_frame(data, oldest_frame,frame_expected,out_buf); break;  /* 超时 */
  case ack_timeout: send_frame(ack, 0, frame_expected, out_buf); break;  /* ACK定时器超时，发送ACK */
  }
if (nbuffered<MAX_BUFS)  enable_network_layer();  else  disable_network_layer();
}
```

滑动窗口协议—小结

- 基本的数据链路层协议
  - 乌托邦式单工协议P1
  - 交互：无错信道上的停等协议P2
  - 差错：有错信道上的停等协议P3
- 一比特滑动窗口协议P4
  - 捎带确认：发送方与接收方二合一
  - 仍为停等：发送窗口等于1，接收窗口等于1
- 回退N协议P5
  - 流水线：可连续发多帧增加在途数据
  - 发送窗口大于1，接收窗口等于1
  - 接收方从坏帧起丢弃所有后继帧，发送方从坏帧开始重传
- 选择重传协议P6
  - 减少重传：接收方可暂存坏帧的后继帧，发送方只重传坏帧
  - 发送窗口大于1，接收窗口大于1
  - 接收窗口较大时，需较大缓冲区

学到大招了吗？
从最简单开始
逐步深入
越来越实际

窗口大小、变化是核心
分析窗口大小优缺点？



思考

- 如何实现可靠传输？
  - 纠错编码；检错码、确认和重传机制
  - 传送层TCP也提供可靠传输服务
  - 链路层的可靠传输服务通常用于高误码率的连路上，如无线链路
  - 对于误码率低的链路，链路层协议可以不实现可靠传输功能

优化与折衷
链路利用率、复杂程度和协议开销
（与网络环境和上层需求相关）

本章内容

4.4 滑动窗口协议

4.5 数据链路协议实例：PPP协议

成 帧	差错控制	流量控制
• 字节计数法 • 带字节填充的定界符法（借助转义字节 ESC 和定界符 FLAG） • 带比特填充的定界符法（借助连续出现的 1 的个数） • 其它	• 海明距离 • 检错码：奇偶校验，校验和，循环冗余校验 • 纠错码：海明码	• 乌托邦式单工协议P1 • 无错信道上的停等式协议P2 • 有错信道上的单工停等式协议P3 • 一比特滑动窗口协议P4 • 回退N协议P5 • 选择重传协议P6

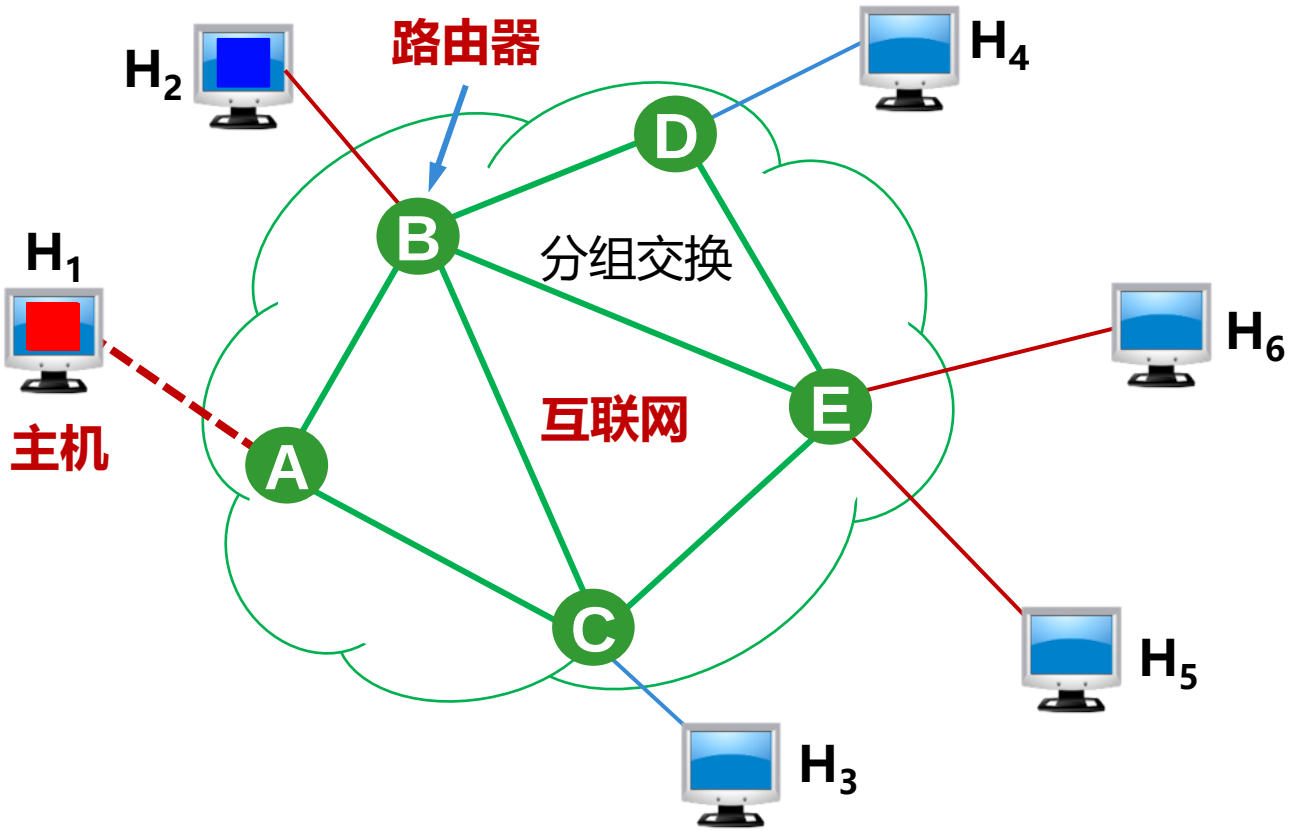
从DHLC到更为简单的PPP

- HDLC协议介绍
  - 高级数据链路控制HDLC (High-level Data Link Control)协议
  - 帧头和帧尾都是特定的二进制序列，即帧标志：01111110作为帧的边界
  - 可以采用多种编码方式实现高效、可靠的透明传输
  - 校验字段：使用16bit的CRC-CCITT标准，或32bit的CRC-32校验
  - 超时断连，递增序号，流量控制和差错控制
- 链路层的简化
  - 随着通信技术进步，信道可靠性大幅提升（路由器带宽不断增大）
  - 没有必要（难以）在链路层使用复杂协议（序号、检错、重传等）来实现数据的可靠传输
  - 不可靠传输协议PPP已成为数据链路层主流协议
  - 可靠传输责任落到传输层TCP协议上

核心路由器的需求

核心路由器
高速网口100G
16~64个
核心诉求是路由转发
重传会有什么问题?

互联网设计原则
核心路由器
只做不得不做的任务
避免了重传怎么办?



世界上最为复杂的人造系统之一

PPP协议简介

- PPP协议
  - PPP(Point-to-Point Protocol)协议由IETF制定，1994年成为RFC1661
  - PPP协议是目前使用最多的数据链路层协议之一
  - 能在不同链路上运行，能承载不同的网络层
- 主要功能特点
  - 利用帧定界符封装成帧：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是点到点，
不是点到多点，
更不是端到端。

简单、灵活

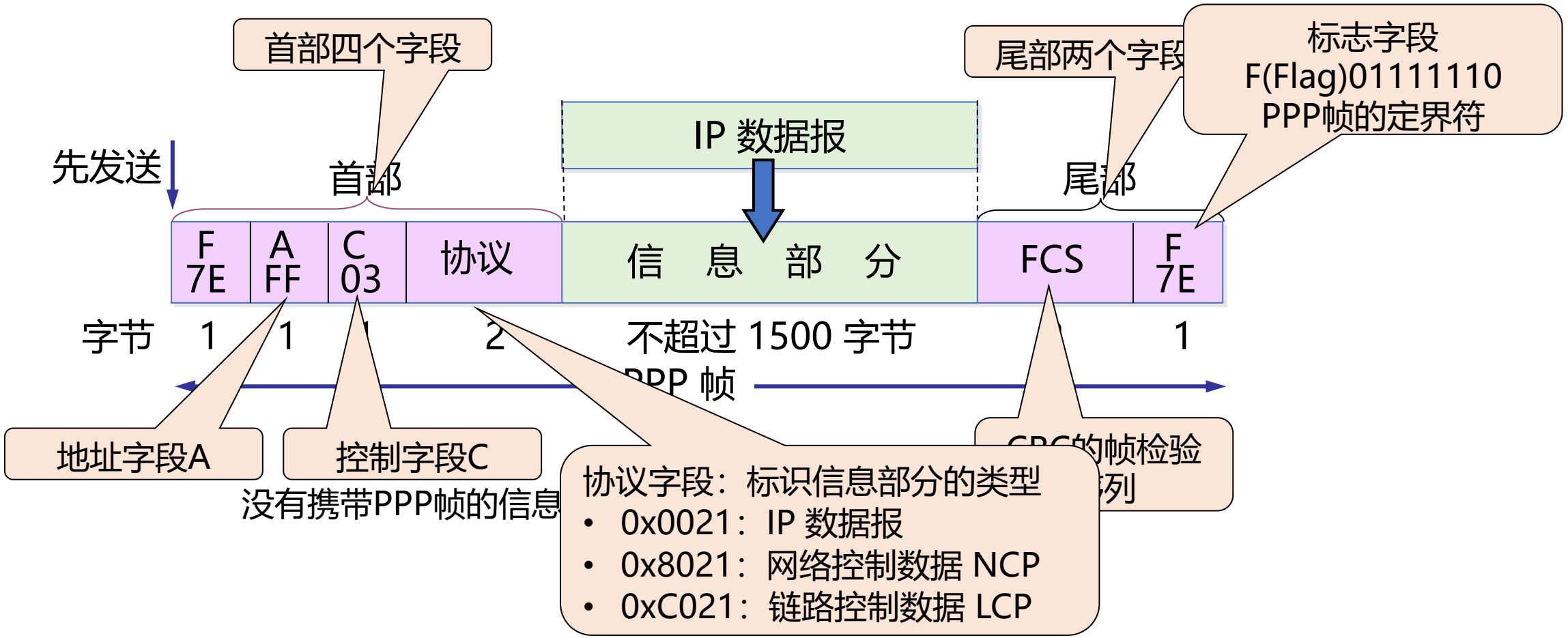
PPP协议未实现的功能

- 帧数据的纠错功能
  - 数据链路层的PPP协议只进行检错，PPP协议是不可靠传输协议
- 流量控制功能
  - PPP协议未实现点到点的流量控制
- 可靠传输功能
  - PPP为不可靠协议
  - 不使用帧的序号（不可靠网络中可能使用有序号的工作方式）
- 多点连接功能
  - PPP协议不支持多点线路，只支持点对点的链路通信
- 单工和半双工链路
  - PPP协议支持全双工链路

PPP协议的构成

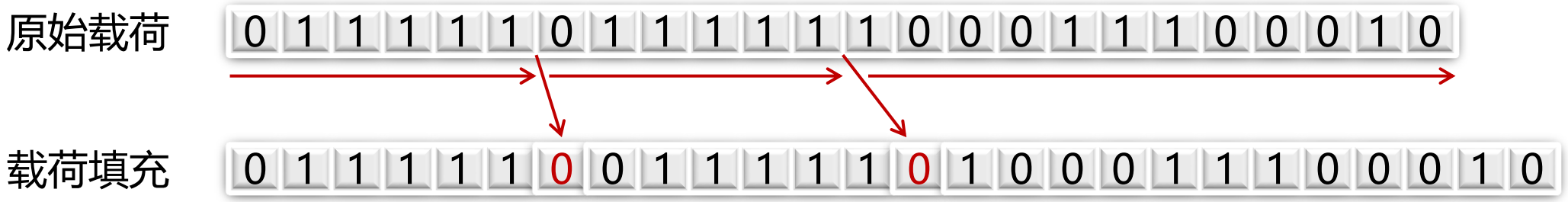
- 封装 (Encapsulation)
  - 提供在同一链路上支持不同的网络层协议
  - PPP既支持异步链路（无奇偶检验的8比特数据），也支持面向比特的同步链路
  - IP数据包在PPP帧中是其信息部分，其长度受到MTU的限制
- 链路控制协议 LCP (Link Control Protocol)
  - 用来建立、配置和测试数据链路的链路控制协议，通信双方可协商一些选项
- 网络控制协议 NCP (Network Control Protocol)
  - 其中每个协议支持一种不同的网络层协议，如IP、OSI的网络层、DECnet、AppleTalk等

PPP协议的帧格式


首部四个字段
尾部两个字段
标志字段
F(Flag)01111110
PPP帧的定界符
IP 数据报
先发送
首部
尾部
F
7E
A
FF
C
03
协议
信 息 部 分
FCS
F
7E
字节 1 1 1 2 不超过 1500 字节 2 1
PPP 帧
地址字段A
控制字段C
没有携带PPP帧的信息
协议字段：标识信息部分的类型
• 0x0021：IP 数据报
• 0x8021：网络控制数据 NCP
• 0xC021：链路控制数据 LCP

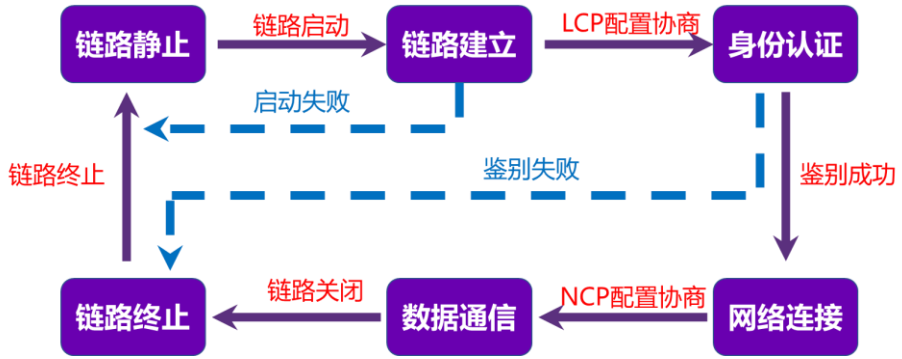
PPP协议的成帧方式

- PPP通常使用的字符填充法
  - 避免在信息字段中出现和标志字段一样的比特组合（0X7E）
  - 当PPP使用异步传输时，定义转义字符0X7D，并使用字节填充
  - 发送端进行字节填充，链路上的字节数超过上层发送的字节数
- PPP支持带**比特**填充的定界符法
  - PPP协议用在SONET/SDH链路时，采用标志字段0x7E
  - 若在有效载荷中出现连续5个1比特，则直接插入1个0比特



PPP协议的工作状态及转换

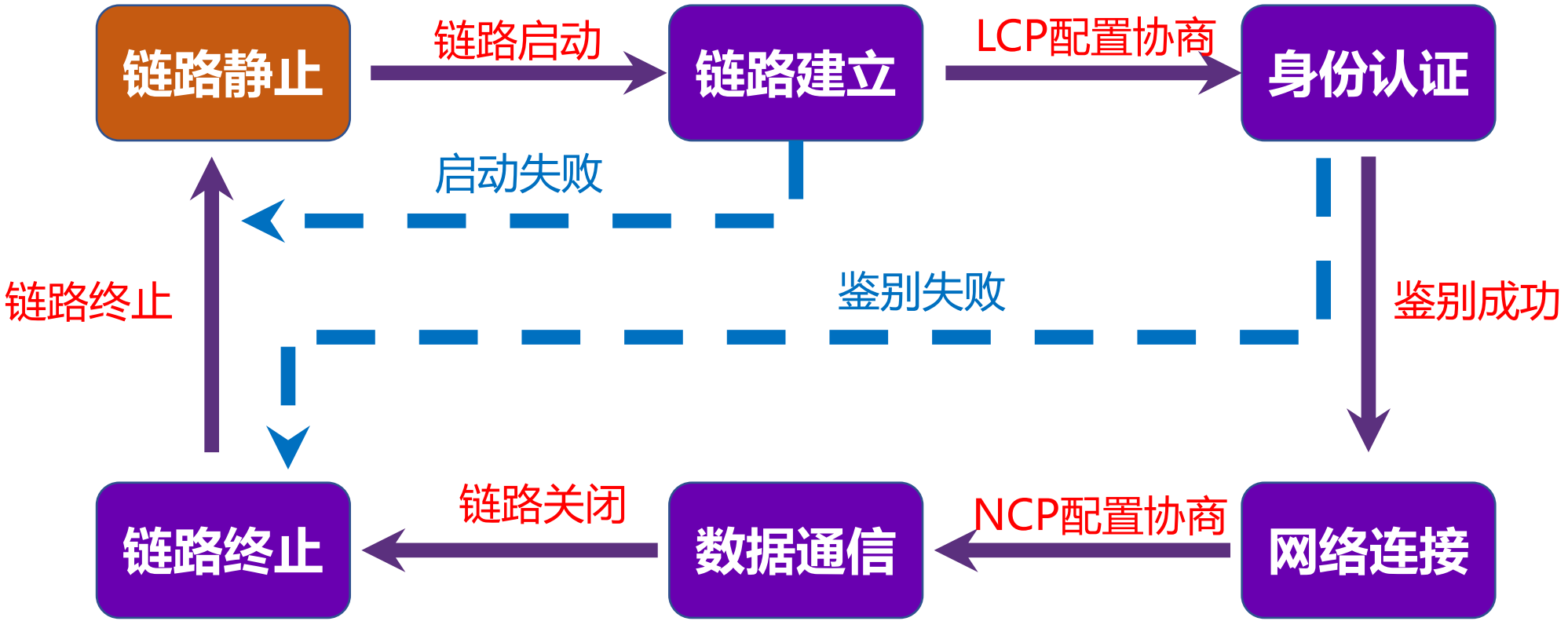
- 状态转换图(State Transform Diagram)：状态机
  - 通过描述系统的状态和引起系统状态转换的事件，指出作为特定事件的结果将执行哪些动作，从而描述系统的行为
  - 状态转换图中以节点表示状态，有向边来表示内外部事件和响应（变迁）
- 状态
  - PPP协议的状态转换图中共设置了六个状态



- 状态的变迁
  - PPP协议中通过一些外部事件来触发状态的变迁，如链路启动、LCP配置协商、鉴别成功、NCP配置协商、链路关闭、链路终止等

PPP协议的工作状态及转换

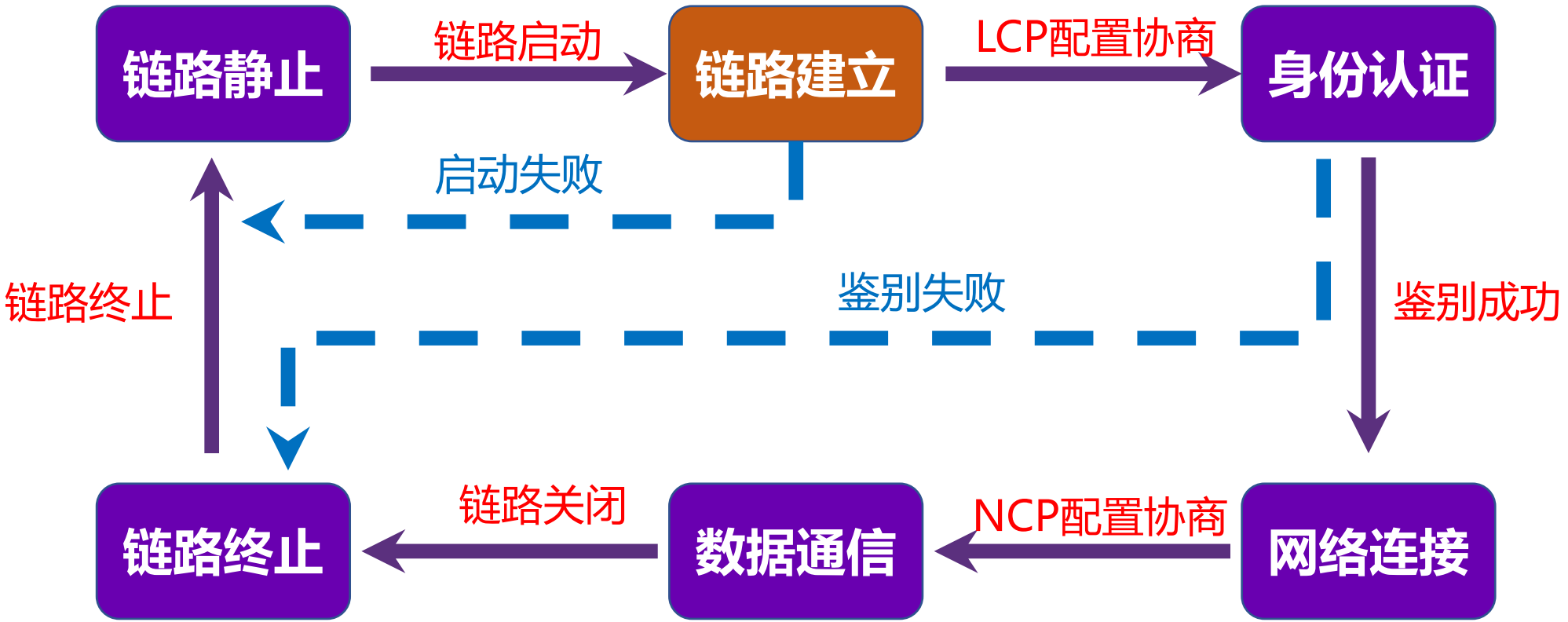
- 采用状态转换图来表示PPP协议工作的行为模型



PPP协议的工作状态及转换

- 链路静止状态
  - PPP链路的起始和终止状态，通信双方尚无链路
  - 当外部事件表明物理层可以使用时，PPP将进入链路建立阶段
    - 如载波检测、网络管理员配置
    - PC机通过调制解调器呼叫路由器时，路由器能检测到调制解调器发出的载波信号，并作出应答

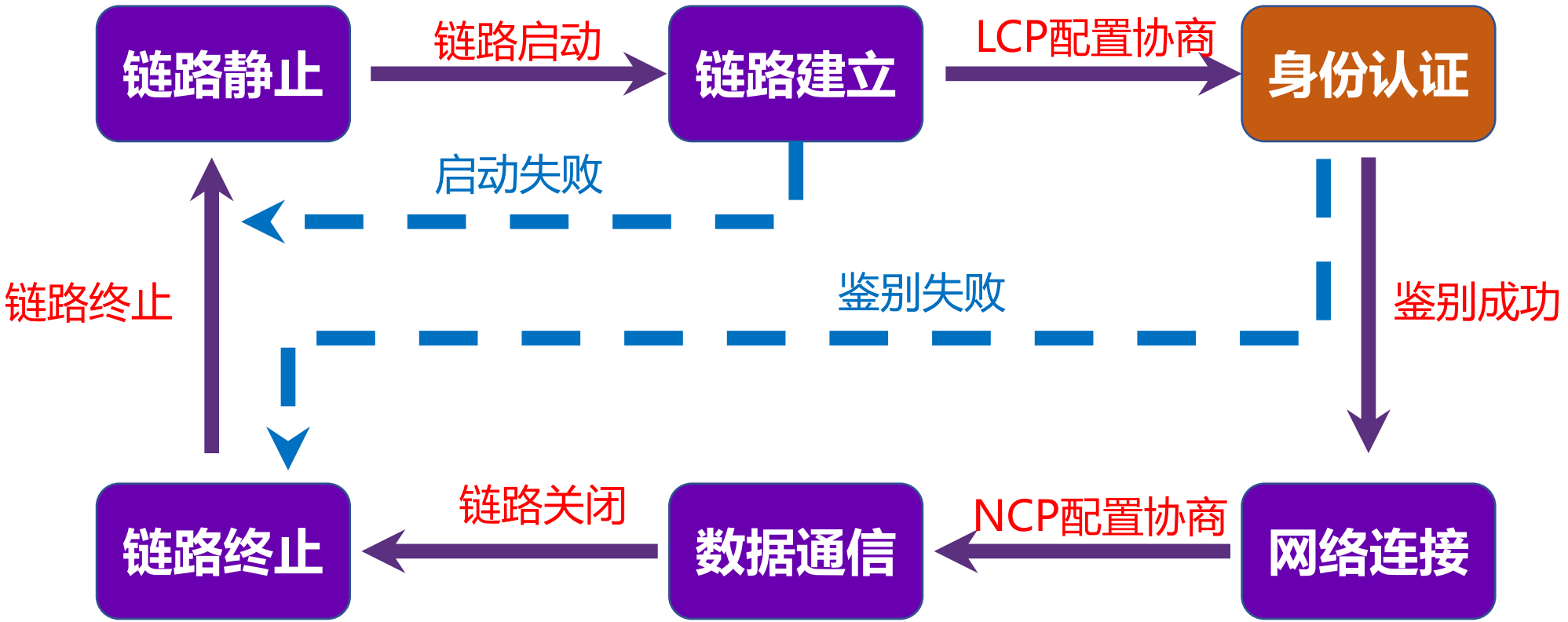
PPP协议的工作状态及转换



PPP协议的工作状态及转换

- 链路建立状态
  - LCP通过交换配置包来建立连接
  - LCP配置选项包括：链路上的最大帧长MTU、是否要进行身份验证，不使用PPP帧中的地址和控制字段
  - LCP开始协商一些配置选项，发送LCP的配置请求帧（Configure-Request），协议字段为LCP对应的代码，信息字段包含特定的配置请求
  - 链路的另一端可以发送几种响应的一种
    - 配置确认帧（Configure-Ack）：所有选项都接受
    - 配置否认帧（Configure-Nak）：所有选项都理解但是不能接受
    - 配置拒绝帧（Configure-Reject）：选项有的无法识别或不能接受，需要协商

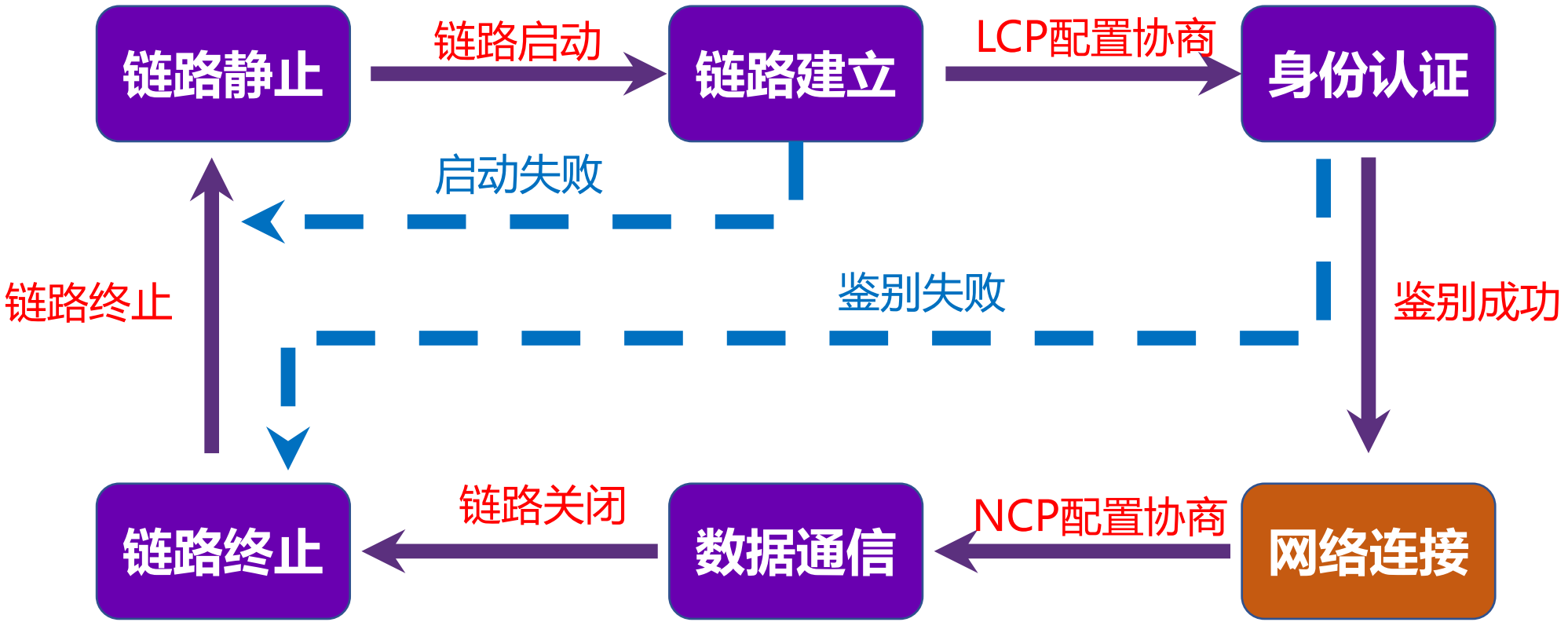
PPP协议的工作状态及转换



PPP协议的工作状态及转换

- 身份认证状态
  - 在允许交换网络层协议包之前，对等点可选身份验证
  - 默认情况下，不需要身份验证
  - 如果选择身份验证协议，需要在链接建立阶段请求使用该身份验证协议
  - 在身份验证完成之前，不能从身份验证阶段推进到网络层协议阶段
  - 如果身份验证失败，则身份验证器应该继续到链接终止阶段
  - 此阶段只允许链路控制协议、认证协议和链路质量监控包
  - 此阶段接收到的所有其他数据包必须丢弃

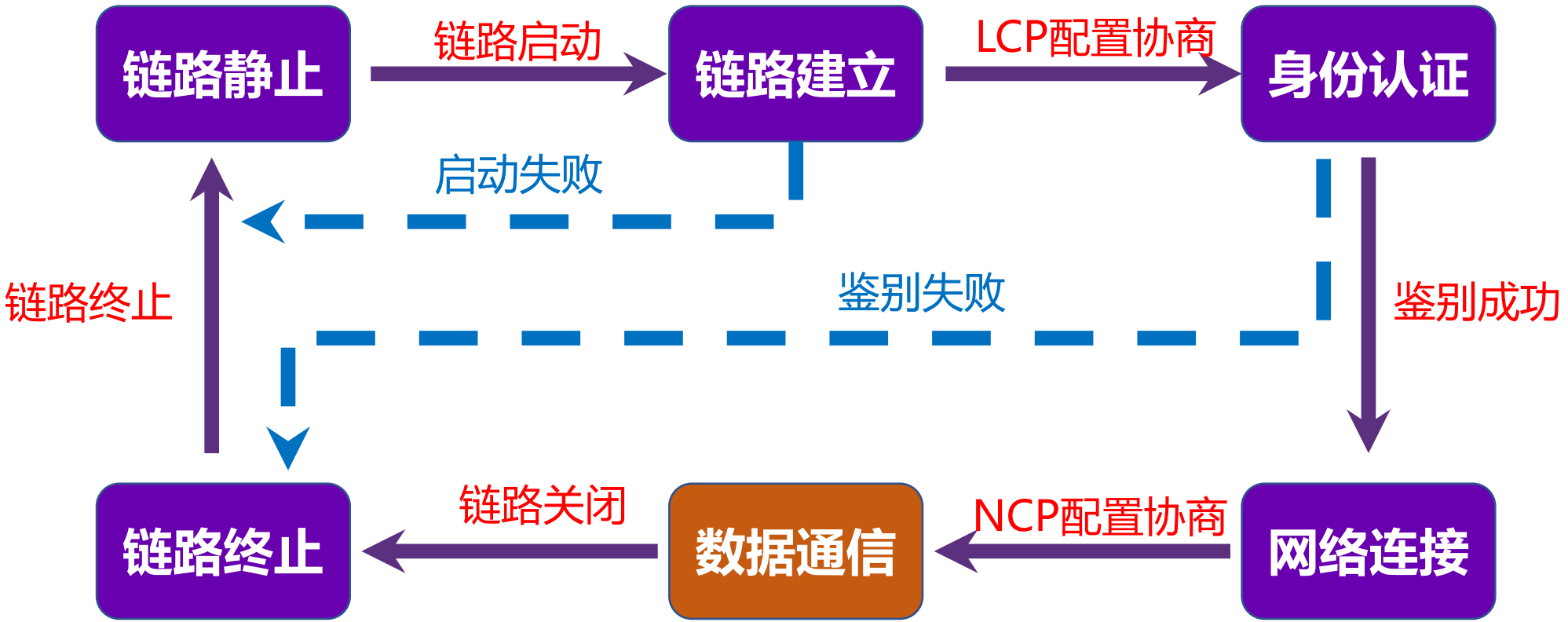
PPP协议的工作状态及转换



PPP协议的工作状态及转换

- 网络连接状态
  - 完成了前面的阶段，每个网络层协议(如IP、IPX或AppleTalk)必须由适当的网络控制协议(NCP)单独配置
  - 每个NCP可以在任何时候打开和关闭
  - NCP到达打开状态后，PPP将携带相应的网络层协议包
  - 如果相应的NCP不处于打开状态时，接收到的任何受支持的网络层协议包都必须丢弃
  - 在此阶段，链路流量由LCP、NCP和网络层协议包的任何可能组合组成

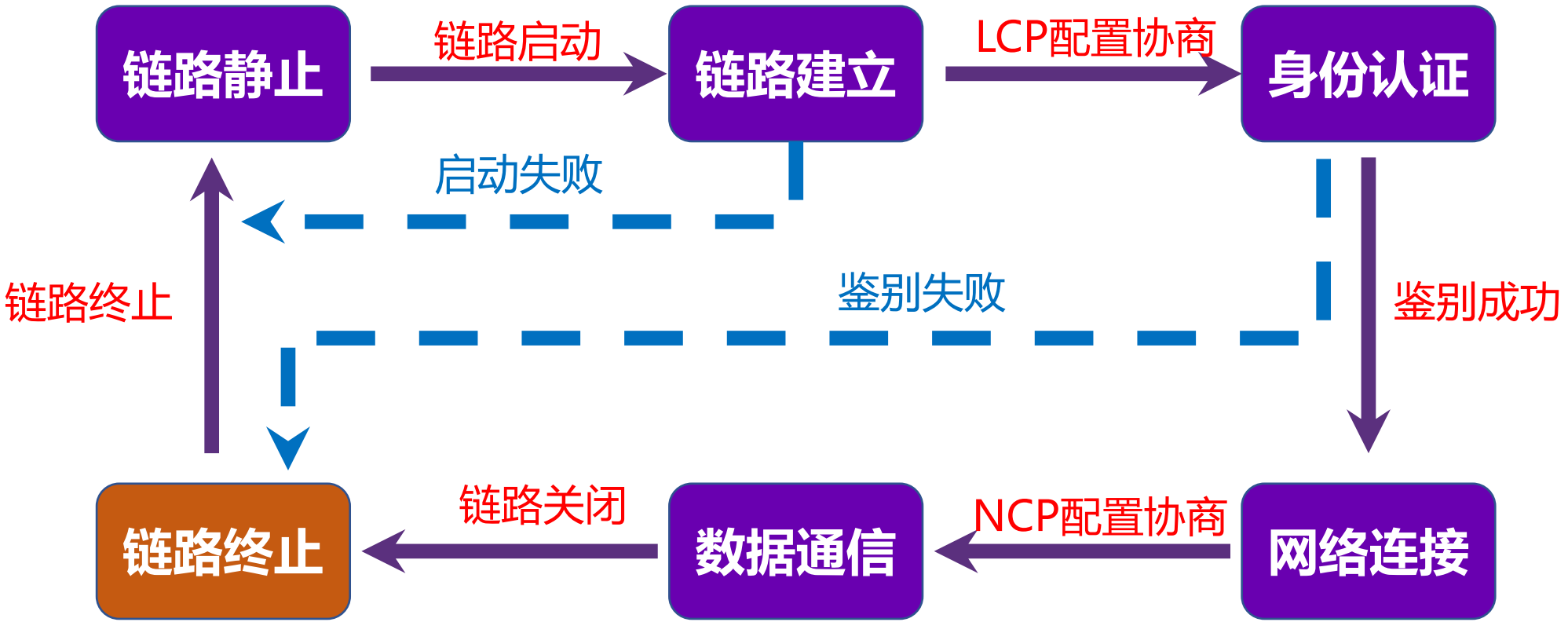
PPP协议的工作状态及转换



PPP协议的工作状态及转换

- 数据通信状态（网络打开状态）
  - 链路的两个PPP端点可以彼此向对方发送分组
  - 或者发送检查链路状态的Echo-*帧

PPP协议的工作状态及转换



PPP协议的工作状态及转换

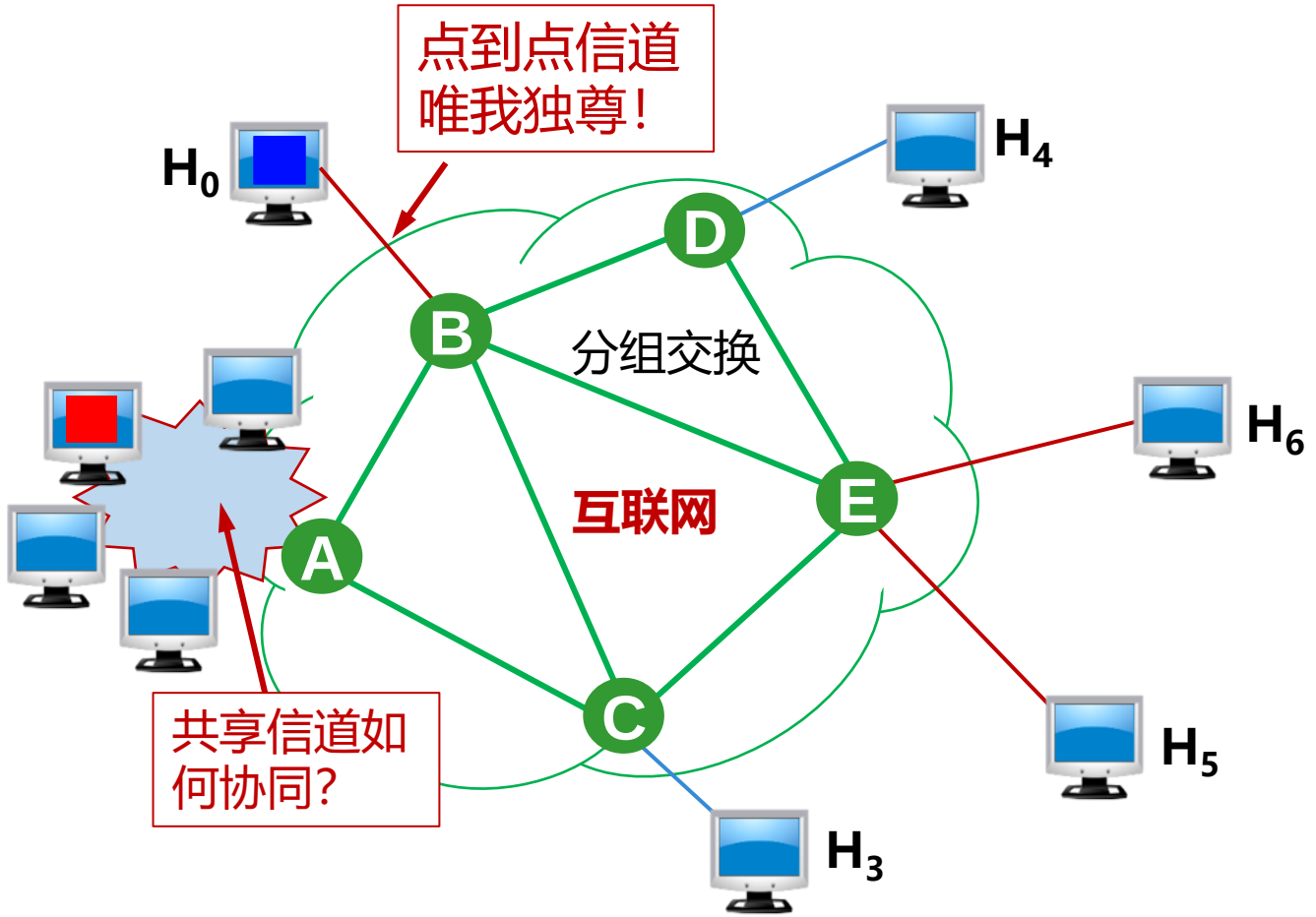
- PPP可以在任何时候终止链接
  - 运营商丢失、身份验证失败、链接质量失败、空闲期计时器过期或链接的管理关闭造成的
- LCP用于通过交换终止包来关闭链接
  - 当连接关闭时，PPP通知网络层协议
  - 在交换终止包之后，实现应该向物理层发出断开的信号，以强制终止链接，特别是在身份验证失败的情况下
- Terminate-Request的发送方在收到Terminate-Ack或重启计数器过期后断开连接
- Terminate-Request的接收方等待对等方断开连接，并且在发送Terminate-Ack之后，在至少一次重新启动时间过去之前，绝不能断开连接
- PPP随后进入链路静止阶段
- 在此阶段接收到的任何非LCP数据包都必须丢弃

本章总结

- 成帧的方式
  - 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
- 差错检测和纠正
  - 海明距离
  - 检错码：奇偶校验，校验和，循环冗余校验
  - 纠错码：海明码
- 基本的数据链路层协议
  - 乌托邦式单工协议P1
  - 无错信道上的停等协议P2
  - 有错信道上的停等协议P3
- 提升效率的滑动窗口协议
  - 一比特滑动窗口协议P4
  - 回退N协议P5
  - 选择重传协议P6
- 数据链路协议实例（了解）
  - PPP协议
  - 协议状态机的描述方式

思考与展望

- 数据链路层
  - 终于搞定了点到点信道的传输难题
  - 接下来呢?
- 共享信道访问?
  - 局限在一跳范围内
  - 多个设备难以协同
  - 互相冲突怎么办?
  - 集中控制v.s.分布式?
  - 发明：以太网、交换机



作业

27. 地球到一个遥远行星的距离大约是 9×10^{10} 米。如果采用停-等式协议在一条 64 Mbps 的点到点链路上传输帧，试问信道的利用率是多少？假设帧的大小为 32 KB，光的速度是 3×10^{8} m/s。
28. 在前面的问题中，假设用滑动窗口协议来代替停-等式协议。试问多大的发送窗口才能使得链路利用率为 100%？发送方和接收方的协议处理时间可以忽略不计。

32. 利用地球同步卫星在一个 1 Mbps 的信道上发送长度为 1000 位的帧，该信道的传播延迟为 270 毫秒。确认总是被捎带在数据帧中。帧头非常短，序号使用了 3 位。试问，在下面的协议中，可获得的最大信道利用率是多少？
    （a）停等式?
    （b）协议 5?
    （c）协议 6?

作业

34. 考虑在一个无错的 64kbps 卫星信道上单向发送 512 字节长的数据帧，来自另一个方向反馈的确认帧非常短。对于窗口大小为 1、7、15 和 127 的情形，试问最大的吞吐量分别是多少？从地球到卫星的传播时间为 270 毫秒。

37. 试问，使用 PPP 发送一个 IP 数据包的最低开销是多少？如果只计算 PPP 自身引入的开销，而不计 IP 头开销，试问最大开销又是多少？

致谢社区本章贡献者

王贵竹

王艳

林卫国

王昊翔

陈振国

张浩

徐涛

徐敬东

研究生面试的启示

- 如何拿到上研究生的资格
  - 大四9月份推免（硕士、直博）
  - 1月份考研（应届和社招）
  - 竞争者：舍友、同学？计算机类，跨专业，外校？自己？
- 面试小误区
  - 清华真好！清华CS世界第一！一心一意三进宫？
  - 我很牛！兴趣驱动型？没发挥好？
  - 想起来了，老师真的没讲☹
- 老师们关心什么
  - 数学、英语还是专业课？学分绩？
  - 不同的高校、专业、排名？
  - 编程能力（机考），研究能力亦或兴趣？责任使命感？

读研不是上课学习
而是跟导师创新

1. 能力
2. 努力
3. 踏实