第三章 数据链路层

三个单工协议

基本数据链路协议

- 由浅入深地学习六个协议，了解数据链路层的主要工作，有些工作原理和概念是网络通信的基础，其运行机理在网络层和传输层也会采用。
- 首先介绍最简单的三个协议：
  - 无限制的单工协议
  - 单工停—等协议
  - 有噪声信道的单工协议

数据的传输在某时是单向的

几个假设

- 物理层、数据链路层和网络层各自是独立的处理进程
- 机器A希望向B发送的是一个可靠的、面向连接的长数据流
- 假设机器不会崩溃
- 从网络层拿到的数据是纯数据

几点说明（1/2）

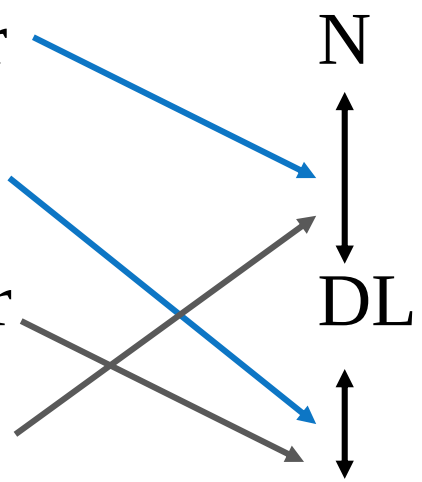
- 六个协议共同使用的数据类型、调用的函数过程定义见图3.9（protocol.h）。
- 与网络层、物理层之间的数据传送接口

发送：from_network_layer

to_physical_layer

接收：from_physical_layer

to_network_layer



几点说明（2/2）

- Wait_for_event 等待某个事件发生event：frame_arrival，cksum_err，timeout
- Timer计时器的作用
  - start_timer，stop_timer，
  - start_ack_timer，stop_ack_timer

重传定时器

捎带确认定时器

帧结构

```
typedef struct{
    frame_kind kind;
    seq_nr seq;
    seq_nr ack;
    packet info;
}frame;
```

无限制的单工协议(协议1)

- 数据单向传送
- 收发双方的网络层都处于就绪状态（随时待命）
- 处理时间忽略不计（瞬间完成）
- 可用的缓存空间无穷大（无限空间）
- 假设DLL之间的信道永远不会损坏或者丢失帧（完美通道）
- “乌托邦”

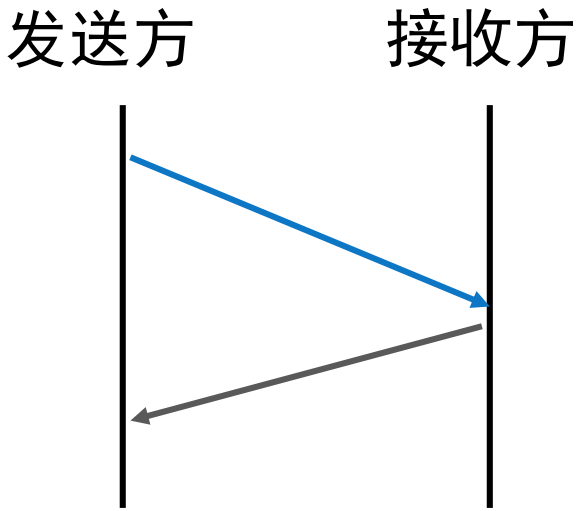
封装
发送

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender1(void)
{
  frame s;                          / * buffer for an outbound frame * /
  packet buffer;                    / * buffer for an outbound packet * /
  while (true) {
      from_network_layer(&buffer);          / * go get something to send * /
      s.info = buffer;              / * copy it into s for transmission * /
      to_physical_layer(&s);         / * send it on its way * /
  }                                 / * Tomorrow, and tomorrow, and tomorrow,
                                        Creeps in this petty pace from day to day
                                         To the last syllable of recorded time.
                                          – Macbeth, V, v * /
}
void receiver1(void)
{
  frame r;
  event_type event;                  / * filled in by wait, but not used here * /
  while (true) {
      wait_for_event(&event);       / * only possibility is frame arrival * /
      from_physical_layer(&r);      / * go get the inbound frame * /
      to_network_layer(&r.info);    / * pass the data to the network layer * /
  }
}
```

解封装
发送

单工的停—等协议（协议2）

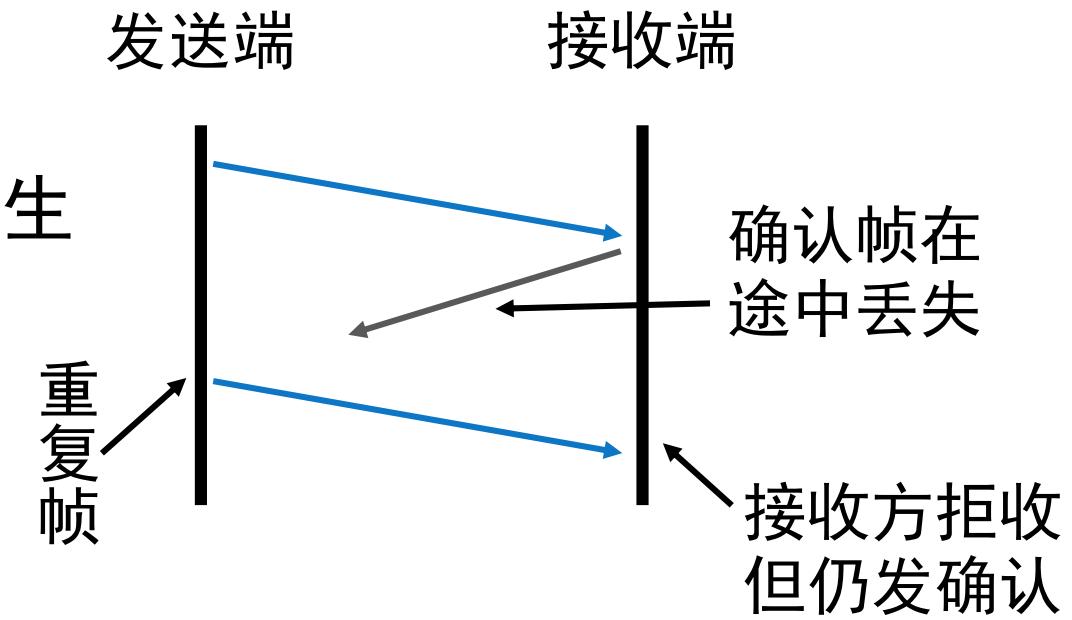
- 解决如何避免收方被涌入的数据淹没，即取消“接收方允许无限量接收”的假设
- 解决方法：收方回发一个哑帧，接收方收到哑帧，表明收方允许接收数据，此时再次发送下一帧数据。
- 实际上是半双工协议



```
void sender2(void)
{
  frame s;                          / * buffer for an outbound frame * /
  packet buffer;                    / * buffer for an outbound packet * /
  event_type event;                  / * frame arrival is the only possibility * /
  while (true) {
        from_network_layer(&buffer);              / * go get something to send * /
        s.info = buffer;              / * copy it into s for transmission * /
        to_physical_layer(&s);       / * bye-bye little frame * /
        wait_for_event(&event);     / * do not proceed until given the go ahead * /
  }
}
void receiver2(void)
{
  frame r, s;                       / * buffers for frames * /
  event_type event;                  / * frame arrival is the only possibility * /
  while (true) {
  wait_for_event(&event);           / * only possibility is frame arrival * /
  from_physical_layer(&r);          / * go get the inbound frame * /
  to_network_layer(&r.info);        / * pass the data to the network layer * /
  to_physical_layer(&s);            / * send a dummy frame to awaken sender * /
  }
}
```

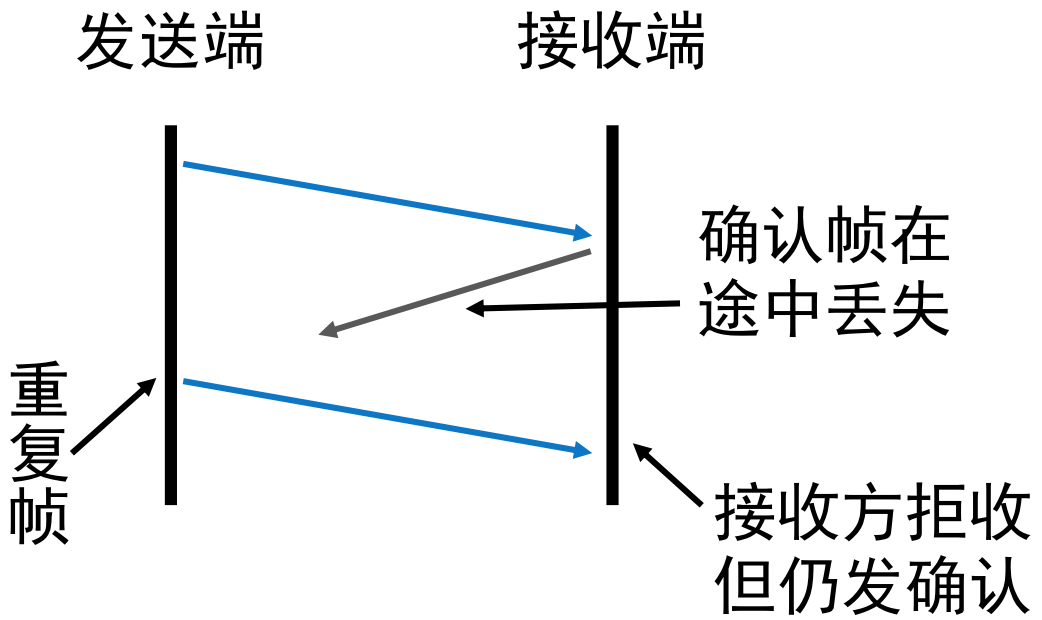
有错误信道的单工协议（协议3）

- 有噪声就会产生差错，有差错就可能会引起以下这些问题：
  - 接收方检测到错误帧，如何通知发送方？如何恢复正确帧？
    - 对正确帧的确认
    - 定时器超期：重传
    - 重传定时器可防止死锁的产生



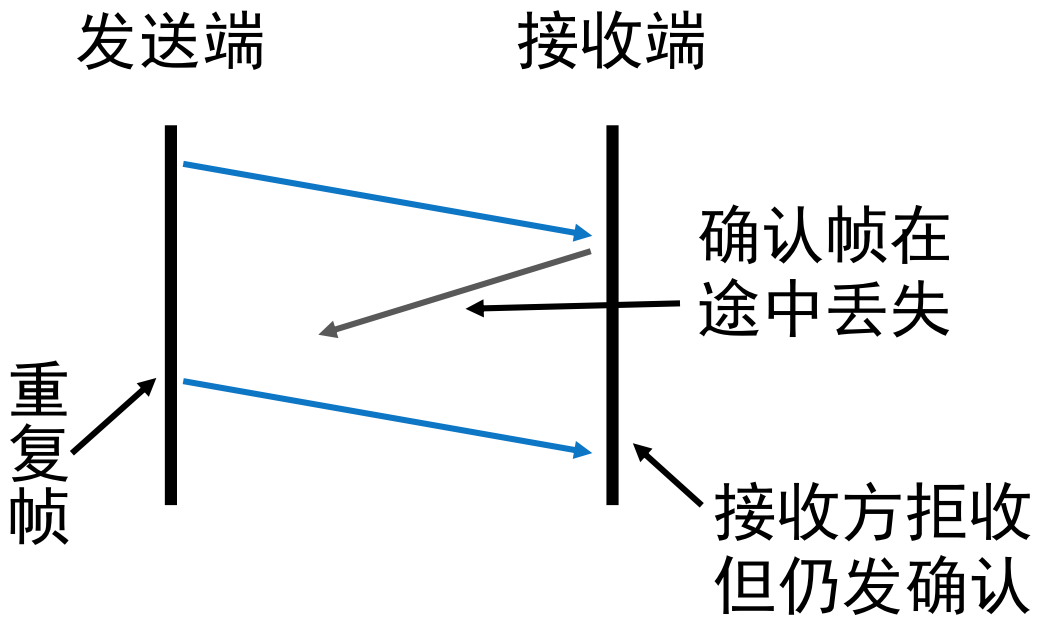
有错误信道的单工协议（协议3）

- 有噪声就会产生差错，有差错就可能会引起以下这些问题：
  - 数据帧或确认帧在途中丢失将如何解决？
    - 定时器超期：重传



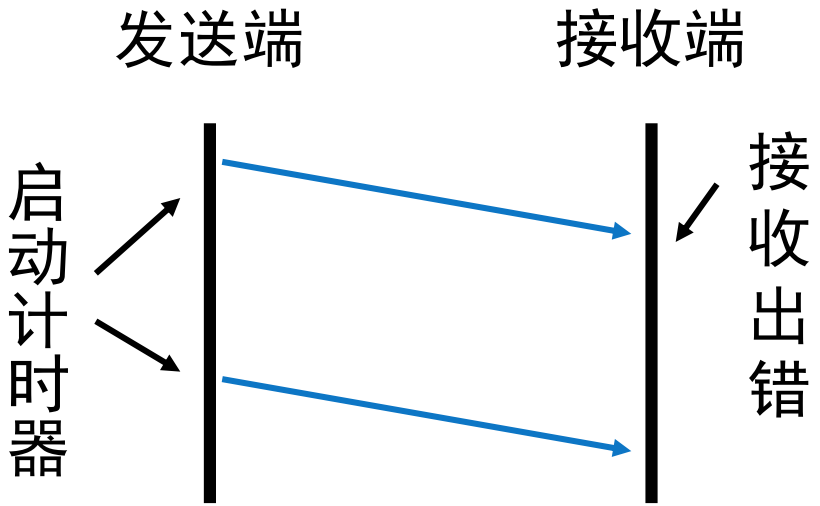
有错误信道的单工协议（协议3）

- 有噪声就会产生差错，有差错就可能会引起以下这些问题：
  - 有可能收到重复帧，如何解决？
    - 给每个帧一个独一无二的序列号
    - 序号也用来重组排序



主动确认重传

- 接收方无误，回发确认帧。
- ARQ:automatic repeat request
- PAR:positive acknowl-edgement with retransmission



PAR/ARQ的基本工作机制

- 发送方每发出一个帧，启动一个重传定时器
  - 超时前，如果收到收方的确认，拆除定时器
  - 超时还未收到确认，重传，重置定时器
    - 数据帧丢失或错误
    - 确认帧丢失

```
void sender3(void)
{
 seq_nr next_frame_to_send;                    / * seq number of next outgoing frame * /
 frame s;                                      / * scratch variable * /
 packet buffer;                                / * buffer for an outbound packet * /
 event_type event;
 next_frame_to_send = 0;                       / * initialize outbound sequence numbers * /
 from_network_layer(&buffer);                  / * fetch first packet * /
 while (true) {
        s.info = buffer;                       / * construct a frame for transmission * /
        s.seq = next_frame_to_send;            / * insert sequence number in frame * /
        to_physical_layer(&s);                 / * send it on its way * /
        start_timer(s.seq);                    / * if answer takes too long, time out * /
        wait_for_event(&event);                / * frame arrival, cksum err, timeout * /
        if (event == frame_arrival) {
              from_physical_layer(&s);         / * get the acknowledgement * /
              if (s.ack == next_frame_to_send) {
                 stop_timer(s.ack);            / * turn the timer off * /
                 from_network_layer(&buffer);  / * get the next one to send * /
                 inc(next_frame_to_send);      / * invert next frame to send * /
                 }
               }
        }
}
```

```
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;
  frame_expected = 0;
  while (true) {
        wait_for_event(&event);                  / * possibilities: frame arrival, cksum err * /
        if (event == frame_arrival) {            / * a valid frame has arrived * /
            from_physical_layer(&r);             / * go get the newly arrived frame * /
            if (r.seq == frame_expected) {       / * this is what we have been waiting for * /
                  to_network_layer(&r.info);     / * pass the data to the network layer * /
                  inc(frame_expected);           / * next time expect the other sequence nr * /
                  }
            s.ack = 1 − frame_expected;          / * tell which frame is being acked * /
            to_physical_layer(&s);               / * send acknowledgement * /
            }
          }
}
```

数据传输的效率很低，怎么办？

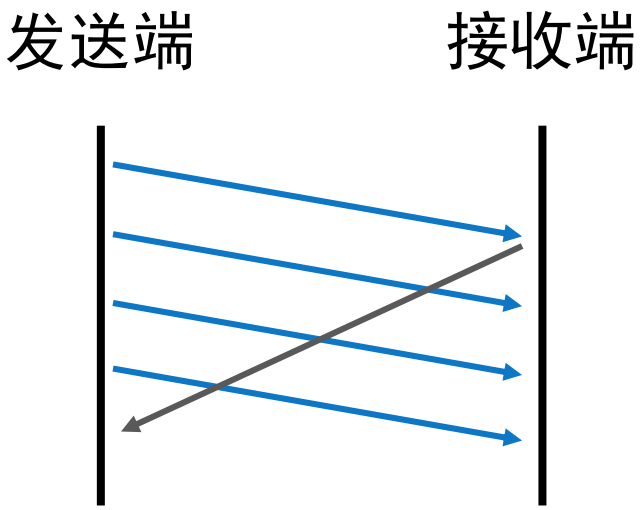
- 全双工
- 捎带确认
- 批发数据
- ？

捎带确认（piggyBacking）

- 捎带确认的作用
  - 进一步减少确认帧
- 捎带确认：将确认暂时延迟以便可以钩到一个外发的数据帧（s. ack）
- 如无法“捎带”，当一个控制捎带确认的计时器超时后，单独发确认帧

批量发送数据

- 停—等协议是每收到一个确认才能发送下一帧，发送端等待时间太长，网络通信效率不高。为了提高效率，可以在等待的时间发送数据帧，这样大大减少了浪费的时间。



小结

- 协议1处理的是“完美”的传输环境，所以叫“乌托邦”协议。
- 协议2做了简单的流控，防止接收方被数据所淹没。
- 协议3解决了噪声信道带来的错误，引出了肯定确认技术。
- 三个协议都是模拟协议，关注其中的技术。

思考题

- 怎么解决接收方被大量数据所淹没？
- 肯定确认重传技术是怎样工作的？
- 重传定时器什么时候启动，什么时候拆除？
- 接收方为什么会收到重复帧？
- 如果发送的帧丢失了，意味着发方永远收不到这帧的确认，发方会死等确认帧（永远不可能到来）的到来吗？
- 什么是捎带确认？

谢谢观看

致谢

本课程课件中的部分素材来自于：（1）清华大学出版社出版的翻译教材《计算机网络》（原著作者：Andrew S. Tanenbaum，David J. Wetherall）；（2）思科网络技术学院教程；（3）网络上搜到的其他资料。在此，对清华大学出版社、思科网络技术学院、人民邮电出版社、以及其它提供本课程引用资料的个人表示衷心的感谢！

对于本课程引用的素材，仅用于课程学习，如有任何问题，请与我们联系！