浙江大学

本科实验报告

课程名称：　计算机网络基础

实验名称：　基于 LoRa 协议实现可靠点对点传输

姓　　名：

学　　院：　计算机学院

系：　计算机科学与技术

专　　业：

学　　号：

指导教师：　董玮

2018 年　06 月　11 日

浙江大学实验报告

实验名称：基于 LoRa 协议实现可靠点对点传输　实验类型：编程实验

同组学生：　　实验地点：计算机网络实验室

一、 实验目的

- 了解 **LoRa** 协议；
- 熟悉**停止-等待协议**流程。

二、 实验内容和原理

- LoRa 是一种**基于扩频技术的远距离无线传输技术**，最早由美国 Semtech 公司采用和推广。LoRa 使用 433MHz，868MHz，915MHz 等免许可的无线电频段，可以实现超低功耗的远距离传输。LoRa 协议包括 **LoRa 物理层协议**和 **LoRaWAN 协议(MAC 层)**。LoRa 物理层协议**主要负责在不同设备之间建立长距离的通信链路，确保上层数据可在物理链路上进行传输**。LoRaWAN 协议基于 **LoRa 物理层**，定义了网络的通讯协议和系统架构；
- 基于 LoRa 物理层，采用**差错检测及差错控制技术**，将不可靠的物理链路变为可靠的数据链路，实现两个 LoRa 节点之间**可靠的数据传输**；
- 基于上述内容，对 **send 函数**和 **recv 函数**进行封装，封装后的接口描述如下：

  bool reliable_send_to(uint8_t* buf, uint8_t len, uint8_t address, uint8_t max_retry)
  描述：发送一个数据包并等待 ACK。收到 ACK，返回 true；当重传次数用尽，仍未收到 ACK，返回 false。
  buf: 要发送的数据地址；
  len: 要发送的数据长度(单位为字节)；
  address：数据发送目的地址；
  max_retry: 最大重传次数；
  返回值：数据发送成功并收到 ACK，返回 true；当重传次数用尽，仍未收到 ACK，返回 false。

  bool reliable_recv_from(uint8_t* buf, uint8_t* len, uint8_t* from, uint8_t* to, uint8_t* id, uint8_t* flags)
  描述： 接收数据包并发送 ACK。
  buf: 存储接收数据空间地址；
  len: 最大接收数据长度，接收完成，将其置为实际接收长度；
  from: 接收到的数据帧原地址；
  to: 接收到的数据帧目的地址；
  id: 接收到的数据帧序号；
  flags: 接收到的数据帧类型；
  返回值：成功接收到数据包，返回 true；否则，返回 false。

三、 主要仪器设备

- Arduino Uno 开发板；
- LoRa Shield 模块；
- TinyLink 客户端。

四、 操作方法与实验步骤

- 下载 TinyLink 客户端；
- 安装 Arduino 驱动；
- 阅读 TinyLink API Reference 中的 TL_LoRa 部分，熟悉 TinyLink 提供的 LoRa 相关 API;
- 两个节点分别运行客户端示例程序和服务端示例程序（example 文件夹下），改变两个节点之间的距离，观察记录数据帧丢失率以及接收信号强度（RSSI）变化；
- 在示例程序的基础上，通过设置包头，实现两个 LoRa 节点之间点对点的数据传输（client 节点地址为组号+1，server 节点地址为组号+2。例如第一组 client 端节点地址为 11， server 端节点地址为 12）；
- 改变发送数据帧大小，记录不同大小的数据帧发送完成所需要的时间。（数据帧发送时间定义为数据帧开始发送到数据帧结束发送所需要的时间。数据帧大小从 10-200Byte 变化，步长为 20Byte）
- 在上一步的基础上，根据实验内容部分的接口定义，封装 reliable_send_to()函数和 reliable_recv_from()函数，利用停止等待协议，实现两个 LoRa 节点之间可靠的数据传输。对于每一帧，在 client 端测量从数据帧发送完成到接收到 server 端发送的 ACK 所需要的时间并记录。
- 测试 reliable_send_to()函数和 reliable_recv_from()函数，客户端发送 500 个数字，判断接收端是否完全接收正确。（传输数据生成可参考示例代码，判断接收端接收数据是否正确可参考附件测试脚本（script 文件夹下））

  数据包结构如下：

Header From	Header To	Header ID	Header Flags
Payload			

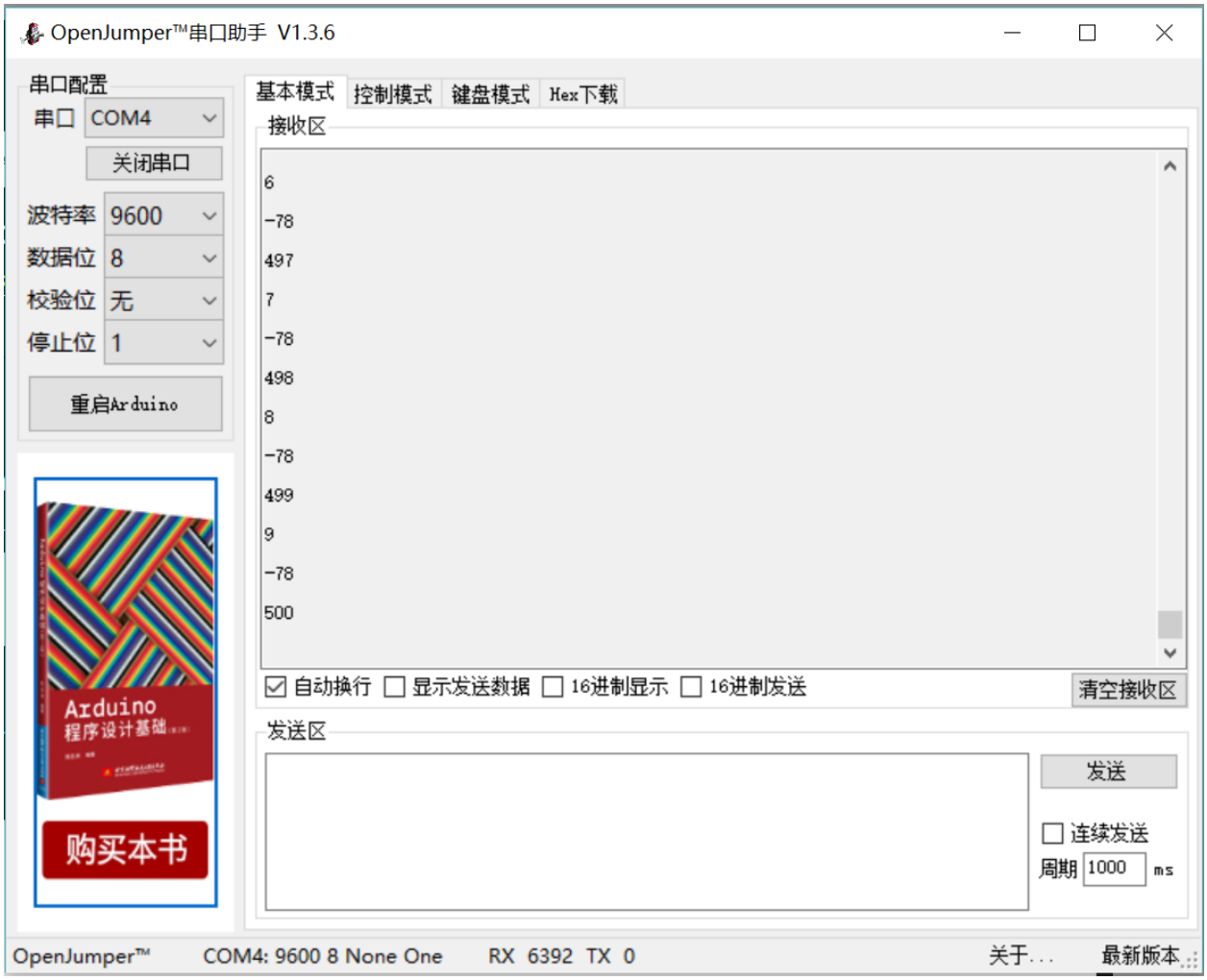
Header From：一个字节，数据包的原地址；
Header To：一个字节，数据包的目的地址；
Header ID： 一个字节，数据包 ID；
Header Flags：一个字节，数据包类型，可自行定义；
Payload：最大长度为（255-4）个字节，传输的数据内容；

五、 实验数据记录和处理

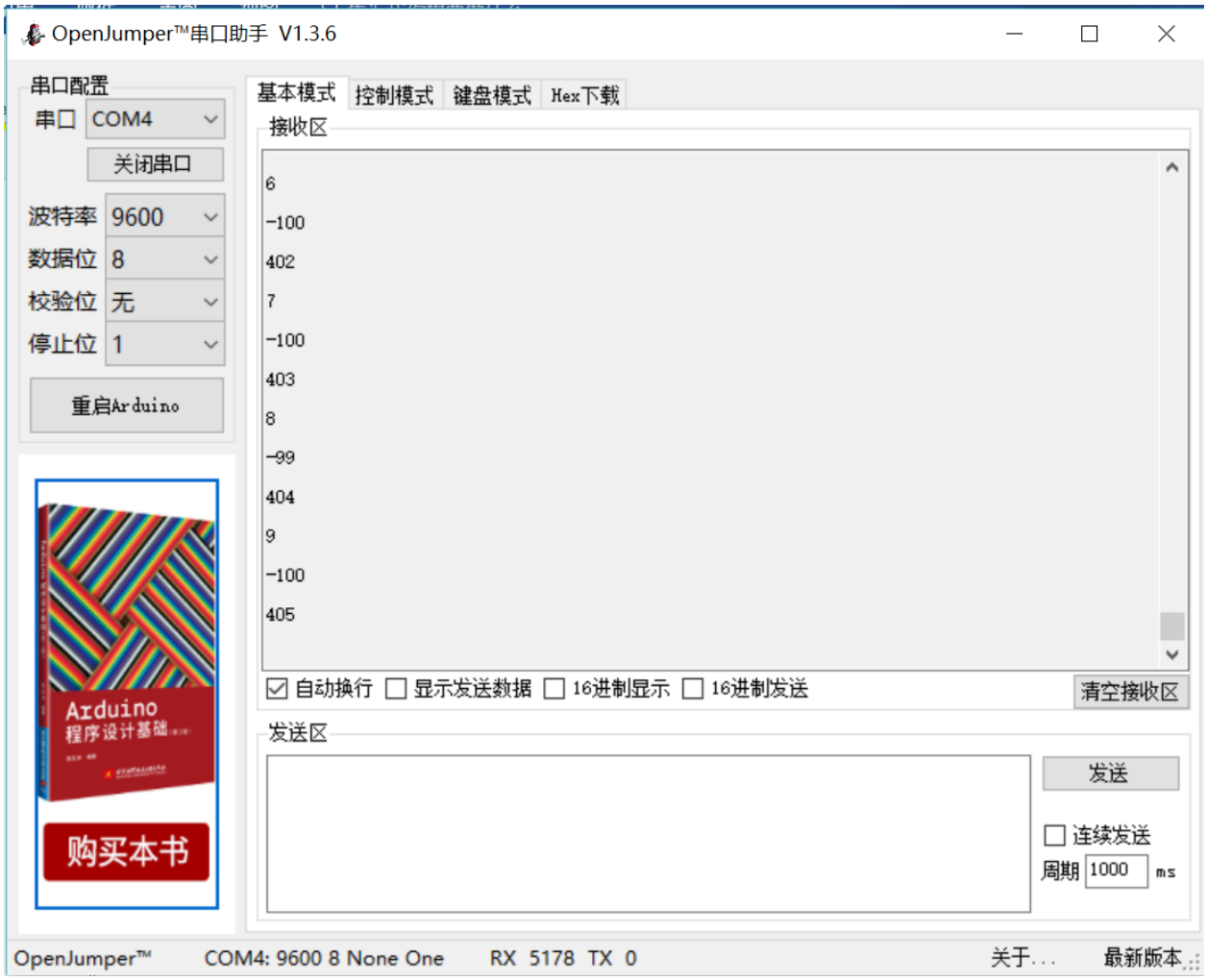
请将以下内容和本实验报告一起打包成一个压缩文件上传：

- **源代码：客户端和服务端的代码分别在一个目录**
- 两个节点分别运行客户端示例程序和服务端示例程序（example 文件夹下），改变两个节点之间的距离，观察记录数据帧丢失率以及接收信号强度（RSSI）变化：
  （1） 以下三张截图，两个节点之间的距离由远至近；
  （2） 以下截图为 server 收到 client 发出的数据后，每收到一次数据，server 都会显示“收到的数据，计数（第几个数据）， 接收信号强度（RSSI）”；

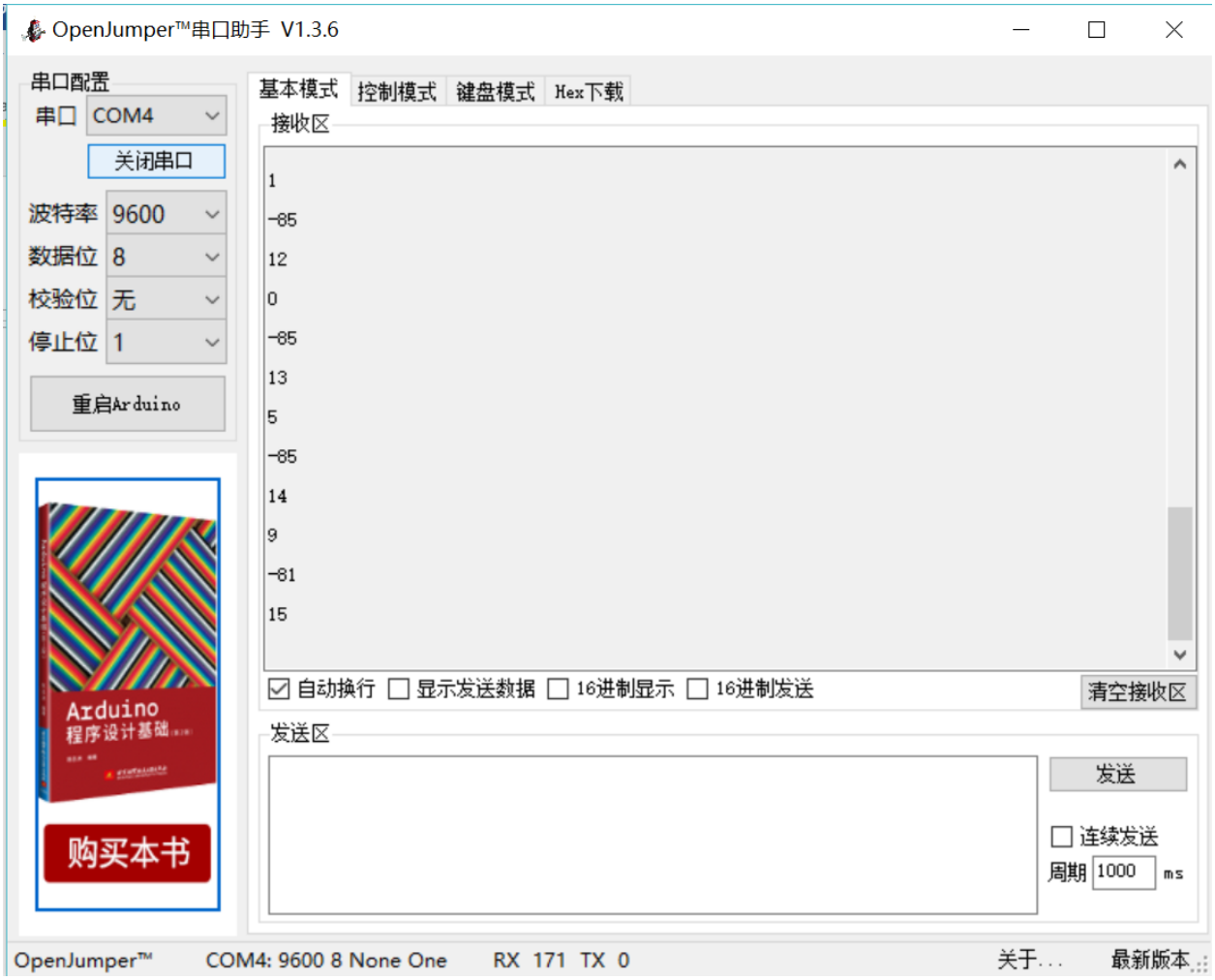
（3） 数据帧丢失率：1 - （500 / 500） = 0%



（4） 数据帧丢失率：1 - （405 / 500） = 19%



（5） 数据帧丢失率：1 - （15 / 500） = 77%




```
#define RH_RF95_MAX_MESSAGE_LEN 251
int count = 0;

void setup()
{
   TL_Serial.begin(9600);
   if (!TL_LoRa.init())
   {
      TL_Serial.println("init failed");
   }
   TL_LoRa.setThisAddress(12);

   TL_LoRa.setTxPower(-1, true);
}

void loop()
{
   if (TL_LoRa.available())
```

```
  {
    // Should be a message for us now
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);

    if (TL_LoRa.recv(buf, &len))
    {
      count = count + 1;
          for (int i = 0; i < len; i++)
          {
              TL_Serial.println((char)buf[i]);
              TL_Serial.println(TL_LoRa.lastRssi());
              TL_Serial.println(count);
      }
    }
    else
    {
          TL_Serial.println("recv failed");
    }
  }
}
```

（6） 由上述实验可知，RSSI 和节点之间的距离有着密切的关系。

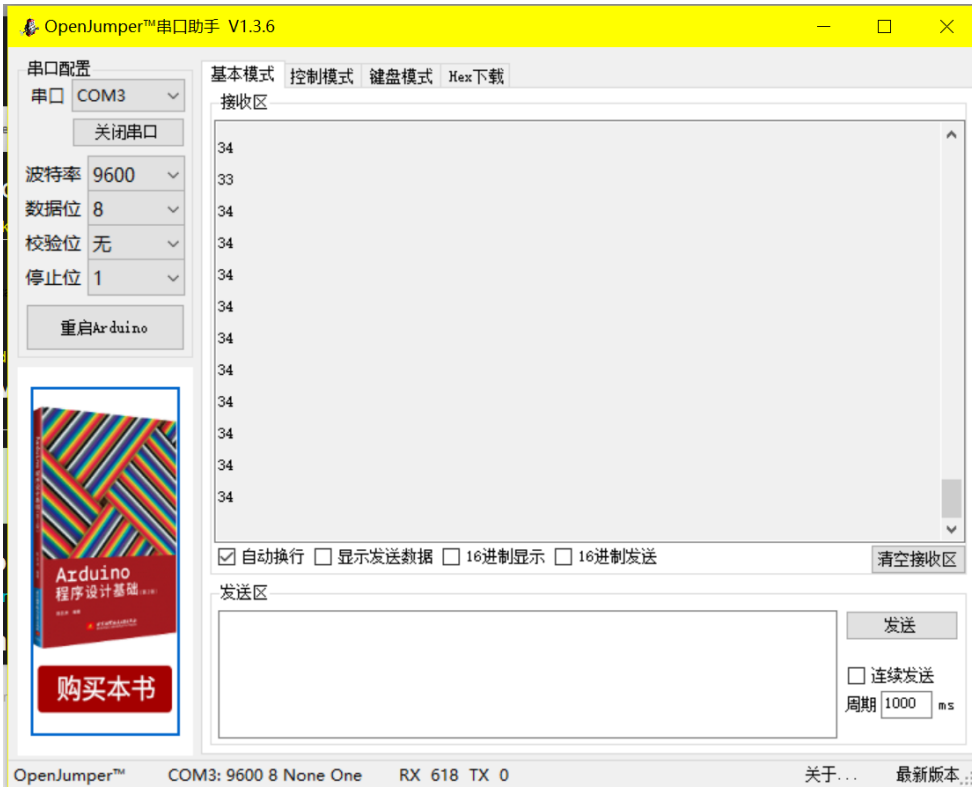
- 改变发送数据帧大小，记录不同大小的数据帧发送完成所需要的时间。（数据帧发送时间定义为数据帧开始发送到数据帧结束发送所需要的时间。数据帧大小从 10-200Byte 变化，步长为 20Byte）

client 发送不同大小的（大小从 10 到 200 变化，步长是 20）的数据包，上述截图是 client 端对数据包从开始发送到发送完所经历的时间。可以看出数据包越大，发送数据包需要的时间越长。

```
#define DATA_LENGTH 500
uint8_t data[DATA_LENGTH];
unsigned long beginTime,endTime;
void setup() {
    TL_Serial.begin(9600);
    if (!TL_LoRa.init()){
        TL_Serial.println("init failed");
    }
    TL_LoRa.setThisAddress(11);
    TL_LoRa.setHeaderFrom(11);
    TL_LoRa.setHeaderTo(12);
    TL_Serial.println("Sending to server");
    TL_LoRa.setTxPower(-1, true);
    for (int i = 0; i < DATA_LENGTH; i++) {
        data[i] = i%10 + '0';
    }
    //dataLength 表示数据的长度
    for (uint8_t dataLength = 10; dataLength <= 200; dataLength+=20) {
        beginTime= TL_Time.millisFromStart();
        TL_LoRa.send(&data[0],dataLength);
        TL_LoRa.waitPacketSent();
```

```
            endTime=TL_Time.millisFromStart();
            unsigned long du=endTime-beginTime;
            TL_Serial.println(du);
        }
        TL_Serial.println("send over!");
}
void loop() {
}
```

- 在上一步的基础上，根据实验内容部分的接口定义，封装 reliable_send_to()函数和 reliable_recv_from()函数，利用停止等待协议，实现两个 LoRa 节点之间可靠的数据传输。对于每一帧，在 client 端测量从数据帧发送完成到接收到 server 端发送的 ACK 所需要的时间并记录。



（1） 服务器实现 reliable_recv_from();

```
#define RH_RF95_MAX_MESSAGE_LEN 251
int expectID=0;

bool reliable_recv_from(){
  uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);

  if(TL_LoRa.recv(buf, &len))
```

```
    {
      TL_LoRa.setHeaderFlags(2);
      TL_LoRa.send(&buf[0], sizeof(buf[0]));
      if(expectID%256 != TL_LoRa.headerId()){
        return false;
      }
      expectID++;
      TL_Serial.println((char)buf[0]);
      return true;
    }
    else
    {
      return false;
    }
}

void setup()
{
    TL_Serial.begin(9600);
    if (!TL_LoRa.init())
      TL_Serial.println("init failed");

    TL_LoRa.setThisAddress(12);
    TL_LoRa.setHeaderFrom(12);
    TL_LoRa.setHeaderTo(11);

    TL_LoRa.setTxPower(-1, true);
}

void loop()
{
    if (TL_LoRa.available())
    {
      //Should be a message for us now
      if(reliable_recv_from())
      {
        TL_Serial.println("recv success");
      }
      else
      {
        TL_Serial.println("recv failed");
      }
    }
}
```

- 停止等待协议关键代码截图，并根据代码，简述停止等待协议流程

```
//要发送的数据包的 id
int id=0;
bool reliable_send_to(uint8_t* data, uint8_t len) {
    //设置包头的 id
    TL_LoRa.setHeaderId(id);
    TL_LoRa.send(data, len);
    TL_LoRa.waitPacketSent();
    //发送完数据包开始计时
    beginTime= TL_Time.millisFromStart();
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
    uint8_t rev_len = sizeof(buf);
    while (true){
        if(TL_LoRa.available()){
            if(TL_LoRa.recv(buf, &rev_len)){
                if(TL_LoRa.headerFlags()==2){
                    //若收到的数据包 flag 是 2（约定好的），就认为是 ACK，跳出循环
                    endTime=TL_Time.millisFromStart();
                    unsigned long du=endTime-beginTime;
                    TL_Serial.println(du);
                    break;
                }
            }
        }
        //超时返回 false，表示发送失败
        if(TL_Time.millisFromStart() - beginTime >timeout){
            return false;
        }
    }
    //跳出循环表示发送成功，id 增 1，返回 true
    id++;
    return true;
}
```

```
int expectID=0; //期望收到的数据包 id
bool reliable_recv_from(){
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if(TL_LoRa.recv(buf, &len)) {
        //每当收到一个数据报就发送一个 ACK
        TL_LoRa.setHeaderFlags(2);
```

```
            TL_LoRa.send(&buf[0], sizeof(buf[0]));
            //如果收到的数据报 id 不是期望的 id，表示收到的包已经收过了，返回 false，发送失败
            if(expectID%256 != TL_LoRa.headerId()){
                //headerId 是 uint_8 的，最大 255，所以判断时要对 256 取模
                return false;
            }
            expectID++;
            TL_Serial.println((char)buf[0]);
            return true;
        } else {
            return false;
        }
}
```

六、 实验结果与分析

- 在停止等待协议实现过程中，客户端发送数据包之后，需要等待服务端返回 ACK，等待的最长时间设置为多大？如何确定？

```
int timeout = 3000;
```

在本次实验中，我们采用了 3s 作为等待的最长时间。应该要大于协议的 TTL。

- 比较停止等待、后退 N 帧以及选择重传协议，讨论在本实验场景下，适合使用哪种协议并分析原因。

（1） 停止等待： 发送端给接收端发送数据，等待接收端确认回复 ACK，并停止发送新的数据包，开启计时器。数据包在计时器超时之前得到确认，那么计时器就会关闭，并发送下一个数据包。如果计时器超时，发送端就认为数据包丢失或被破坏，需要重新发送之前的数据包，说明数据包在得到确认之前，发送端需要存储数据包的副本。 停止等待协议是发出一个帧后得到确认才发下一个，降低了信道的利用率。

（2） 后退 N 帧： 在发送完一个帧后，不用停下来等待确认，而是可以连续发送多个数据帧。收到确认帧时，任可发送数据，这样就减少了等待时间，整个通信的通吞吐量提高。 如果前一个帧在超时时间内未得到确认，就认为丢失或被破坏，需要重发出错帧及其后面的所有数据帧。这样有可能有把正确的数据帧重传一遍，降低了传送效率。

线路很差时，使用退后 N 帧的协议会浪费大量的带宽重传帧。

（3） 重传协议：在发送过程中，如果一个数据帧计时器超时，就认为该帧丢失或者被破坏，接收端只把出错的的帧丢弃，其后面的数据帧保存在缓存中，并向发送端回复 NAK。发送端接收到 NAK 时，只发送出错的帧。 如果落在窗口的帧从未接受过，那么存储起来，等比它序列号小的所有帧都按次序交给网络层，那么此帧才提交给网络层。 接收端收到的数据包的顺序可能和发送的数据包顺序不一样。因此在数据包里必须含有顺序字符来帮助接受端来排序。 选择重传协议可以避免重复传送那些正确到达接收端的数据帧。但是接收端要设置具有相当容量的缓存空间，这在许多情况下是不够经济的。

我认为在实验场景中使用停止等待协议最好，因为 LoRa 实现超低功耗的远距离传输，因此后退 N 帧非常消耗带宽，重传协议则需要相当容量的缓存空间，都不够经济。而停止等待是更符合实验需求的。

七、 讨论、心得

本次实验总体来说比较顺利，前期都较快完成。但是最后的可信传输在逻辑上相对复杂一些，需要注意对 Header_id 的处理,//如果收到的数据报 id 不是期望的 id,表示收到的包已经收过了,返回 false，发送失败。其次要注意 headerId 是 uint_8 的，最大 255，所以判断时要对 256 取模。