实验七 基于UDP实现可靠传输

一 总述

本实验中，我们需要基于UDP实现一个可靠传输协议，该协议需保证UDP报文段能够在包丢失、重复、失序等特殊情况下按序、可靠地交付，我们暂且称该协议为RUDP(Reliable UDP)。

保证报文在发送者和接收者之间可靠交付的方法有多种，在本实验中，我们给大家提供协议接收端的代码，该接收端收到数据包后会返回一个累积确认ACK，在之后的例子中，我们会对此过程作进一步的阐述。大家要做的，是实现发送端的代码，保证该发送端发出的数据包能够可靠地交付给接收端。

我们定义RUDP协议有四种消息类型，分别是start，end，data和ack。start，end，data的消息格式如下：

start|<sequence number>|<data>|<checksum>

data|<sequence number>|<data>|<checksum>

end|<sequence number>|<data>|<checksum>

* start消息用来初始化连接，该消息中的sequence number是收发双方使用的初始包序号。发送完start消息后，连接得以建立，接下来便会发送data消息传递数据。
* end消息用来断开连接，并且，end消息中携带了发送方发给接收方的最后一段数据。
* 大家要实现的发送端应该能够接收并处理来自接收端的ack消息，ack消息的格式如下：

ack|<sequence number>|<checksum>

需要注意，我们的RUDP报文的最大长度是有限制的。因为我们模拟以太网下的数据传输，以太网帧最大长度是1518字节，帧头部占18字节，IP头部占20字节，UDP头部占8字节，而我们的RUDP是实现在UDP之上的，因此RUDP报文最大长度是1472字节（包括类型、序列号、数据、校验和）。在实现发送端的代码时，如果待发送的文件过大，则需要将待发送的文件进行分割以“装进”RUDP报文中。

上面所示的协议消息格式中，尖括号（“<”和“>”）不属于消息的一部分，并且注意在“｜”两侧没有空格，在实现发送端代码时需要严格遵循我们规定的这种格式。

二 接收端说明

我们已经为大家写好了接收端，接收端在接收到数据包后，会发送ack消息给发送端，并且采用累积确认。一旦收到类型为start，end或data的消息，接收端就发送一个ack消息，该ack消息的sequence number表示接收端期望接下来收到的数据包的序号。现在，如果接收端期望收到sequence number为N的数据包，那么接下来会有两种情况：

1. 如果接收端收到了一个sequence number不为N的数据包，它会发送“ack|N”
2. 如果接收端收到了一个sequence number为N的数据包，它会检查自己已按序收到的数据包中序号最大的数据包，假设该数据包的sequence number为M，那么接收端会发送“ack|M+1”

我们再通过两个例子具体看一下。假如发送端已经发送了数据包0，1和2，但是1在到达接收端之前丢了，那么，接收端收到0后，会发送“ack|1”，收到2后，仍然会发送“ack|1”，之后，由于发送端重传了1，接收端收到了1，于是会发送“ack|3”。

注意，如果某一时刻接收端期望收到的序号是N，那么对于大于N+4的数据包，它都会丢掉，这是因为接收端维护了一个容纳5个数据包的接收窗口，对于收到的但是未落入接收窗口的数据包，接收端会直接丢掉。接收窗口是可以滑动的，此时接收窗口为[N, N+4]，如果接收端收到了N，那么接收窗口变为[N+1, N+5]。

如果接收端在10s内没有收到任何数据包，那么它会自动断掉连接。

三 发送端说明

发送端主要实现以下几点功能：

* 读取一个文件并将该文件通过UDP套接字发给指定的接收端
* 如果文件过大，需将文件划分为大小合适的段后再装入RUDP报文
* 通过start消息指定初始序号
* 在每个报文后附加校验和字段（生成和验证校验和的函数已经在Checksum.py中提供）
* 对于收到的校验和无效的ack消息，直接丢掉
* 通过Go Back N算法实现可靠传输（记得接收端窗口大小为5）

发送端应当在以下几种情况下仍然保证可靠传输：

* 丢包：任意等级的丢包，包括100%的丢包率，也包括数据包或确认包的丢失
* 失序：数据包到达接收端的顺序是任意的
* 重复：数据包可能会重复发送给接收端

发送端通过以下命令调用：

Python Sender.py -f <input file>

关于发送端的其他注意事项：

* 发送端应当设置一个500ms的重传计时器，对于超过500ms但是仍未收到确认消息的数据包，应当进行重传
* 发送端应当设置一个大小为5的发送窗口
* 发送端应当可以支持任意类型的文件（即可以发送图片/视频文件等等）

仅实现Go-Back-N算法最高可获得85分，如果额外实现Selective Acknowledgements（选择重传），最高可获得100分，我们提供的Receiver.py代码已经支持选择重传，在运行的时候加上-k或--sack选项即可以选择重传模式运行，在选择重传模式下，接收端除了发送累积确认，还会发送选择确认，具体格式如下：

sack|<cum\_ack;sack1,sack2,sack3,...>|<checksum>

cum\_ack是累积确认，这跟ack消息中的sequence number含义相同，cum\_ack之后，是一系列的选择确认。

比如，如果发送端发送了数据包0，1，2，3，4，但是1和2丢了，下面的表格展示了当接收端依次接收到数据包的时候发送的sack消息：

|  |  |
| --- | --- |
| Packet Received | sack packet sent by the receiver |
| 0 | sack|1;|<checksum> |
| 3 | sack|1;3|<checksum> |
| 4 | sack|1;3,4|<checksum> |

发送端和接收端都应当通过添加-k或--sack选项来开启选择确认模式：

发送端：python Sender.py ­f <filename>­k

接收端：python Receiver.py ­k

我们给出了一个UnreliableSender作为样例Sender，该Sender没有实现可靠传输，大家可以参考该Sender是如何继承BasicSender，如何实现最基本的发包功能。

四 测试

TestHarness.py为大家提供了一个测试框架（通过python TestHarness.py -s YourSender.py -r Receiver.py调用），tests目录下是具体的测试用例。目前的测试用例只包含了模拟丢包的测试，大家需要自己编写模拟失序和重复的测试用例（以及其他大家可以想到的一些边界条件的测试用例）。

如果实现了选择重传，每种测试条件（丢包、失序、重复）需要写两个测试类，分别用来测Go-Back-N和选择重传；如果没有实现选择重传，每种测试条件只需写一个测试类；

编写完测试用例后，需要在TestHarness.py中的tests\_to\_run函数中加入该测试用例，然后才能通过调用TestHarness运行该用例。

五 评分

* 实现了Go-Back-N算法，测试用例编写正确无误，并且能通过丢失、失序、重复三个测试用例，最高得85分；
* 在以上基础上正确实现了选择重传算法，最高可得100分；
* 代码以及测试都将被实际执行进行验证，请务必按如下格式保证代码和测试可执行：

1. 代码执行：

Go-Back-N接收端：python Receiver.py

Go-Back-N发送端：python Sender.py -f <file name>

选择重传接收端：python Receiver.py -k

选择重传发送端：python Sender.py -f <file name> -k

1. 测试执行

python TestHarness.py -s Sender.py -r Receiver.py