项目2 –可靠的传输——总览

本文件夹中包含接收端(receiver)程序的例子，计算校验和(checksums)的代码，也包含发送端(sender)的例子。

你要做什么?
==========
简单来说你需要扩充 Sender.py 以实现可靠传输。完成此项目，你不必包含其他文件。Sender.py 中提供了所有用于处理命令行参数的基本材料，在给你评分的时候，我们会使用这些命令行参数。

接收端（Receiver）
==========
Receiver.py 是接收端的示例程序。这里提供的版本很大程度上与我们用来给你的项目评分的版本一致(其不同点主要在于，评分的版本中包括了辅助评分的一些信息，但在功能没有大的区别).如果你想修改接收端，可随意修改，但请记住，老师会用自己的版本来测试你的发送端，而不是你改过的版本。

BasicSender and Friends(友元)
==========
文件 BasicSender.py 中的 BasicSender 类提供了一个框架,你可以基于此框架来建立可靠的发送端(sender).其提供了下列方法:

__init__(self,dest,port,filename): 创建 BasicSender. 指定目标主机名，接收端监听的端口，及要传输的文件名（filename）. 如果不提供文件名，将从键盘(STDIN)读入数据.

receive(self, timeout): 接收一个数据包。并等待包的到达，包到达后结束该函数。可指定最长等待时间（timeout），即如果此时间后，包还未到达，函数也结束。该函数将以字符串的形式返回接收到的包，而当超时时返回值为 None.

send(self,message): 发送消息给接收端，该接收端是你在创建发送端时指定的。

make_packet(self,msg_type,seqno,message): 创建一个 BEARS-TP 包，并指定该包的消息类型（msg_type）,序列号(seqno),和内容（message）。该函数将产生合适的校验和（checksum），并返回已经添加了校验和(checksum)的 BEARS-TP 包。

split_packet(self,packet): Given a BEARS-TP packet, splits a packet into a tuple of the form (msg_type, seqno, data, checksum). For packets without a data field, the data element will be the empty string.

另外, 它定义了一个你必须实现的方法:

start(self): 启动发送端。

我们还提供了三个基于 BasicSender 的发送端的示例;你也可以通过修改它们来实现你的工作:

UnreliableSender.py: 尽管该程序不提供可靠性，但却用我们规定的协议实现了从文件或者键盘中读数据，并发送给接收端。

InteractiveSender.py: 一个简单的发送端. 它将发送你键入的任何信息给指定的接收端。然后等待接收端的响应，之后提示你输入一个新的消息。

StanfurdSender.py: 和 InteractiveSender.py 几乎完全一样,在包计数上做得不是很好，有时会失去序列号的踪迹。你可以用此来考察，当出现包计数错误（或者失去包计数的踪迹）时，接收端的表现。

Checksums
=========
Checksum.py 文件包括用于对包进行验证和生成校验和的二个函数:

validate_checksum(message): 当消息的校验和正确时，返回 true,否则返回 false. 此函数假定消息的最后域是校验和.

generate_checksum(message): 返回消息（message）的校验和字符串. 此函数假定消息包含有尾定界符对此校验和(checksum)你应该简单的附加在消息的最后。

Testing
=======
你需要替你自己的代码来编写测试用例，以保证与项目规格说明书中的描述一致。为了帮助大家,我们提供 了一个最简单的测试工具(TestHarness.py). 该测试工具将截获发送端和接收端之间的包。它可以修改包流并检验以确保流满足特定条件。这和我们用来评估大家项目的脚本非常相似。

为了告诉大家如何使用此测试工具，我们提供了二个测试用例(BasicTest() and DropRandomPackets)。这些测试用例使用指定的发送端将 README 文件发送到指定的接收端,其中一个测试用例是将所有包不做修改的转发出去，另一个是随机的丢弃包。最后检验接收到的文件与输入的文件是否匹配。

为了运行此测试平台进行测试，输入下面的命令行指令:

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

其中"YourSender.py"是你的发送端的文件名，"Receiver.py"是你的接收端的文件名。在 TestHarness.py 中,你需要修改位于脚本头部的函数"tests_to_run",来包含你增加的所有测试用例。

通过我们所提供的基本测试用例是必须的，但对完成好此项目，并不足够。还有很多边缘条件用例未覆盖。 另外，除了简单的产生与输入文件相同的拷贝之外，还有其他正确性条件(例如,不应进行不必要的重传,也不应对已经应答的进行重传)。你需要想出各种可能的边界条件，并编写合适的测试用例来覆盖这些条件。

附：关于测试的进一步的一些说明，这些说明可以以后再学习。

如果想对项目 2 进行测试，只需要运行 TestHarness.py 文件即可。两种方法运行 TestHarness.py：

- 在 main 函数里面，sender 变量换成自己的 sender 文件名，同理，receiver 换成自己的 receiver 文件，直接运行 python TestHarness.py。
- 通过命令行，python TestHarness.py -s MySender.py -r Receiver.py 运行， -s 后面添加自己的 sender 文件，-r 后面添加 receiver 文件。

TestHarness.py 的 main 函数中，主要是调用了 tests_to_run（）这个函数对你的 sender 测试。tests_to_run 里面包含了几个测试模块，其中，BasicTest 里面模拟了理想网络环境，而 RandomDropTest.py 模拟了网络中丢包，RandomDropTest 继承了 BasicTest，但是并没有实现随机丢包这个功能。好在 RadnomDropTest.py 提示你如何编写：This tests random packet drops. We randomly decide to drop about half of the packets that go through the forwarder in either direction. Note that to implement this we just needed to override the handle_packet() method -- this gives you an example of how to extend the basic test case to create your own。你只需要覆盖 handle_packet（）这个函数，并在这个函数里面模拟丢包的环境即可。

当然，你也可以编写自己的 RandomTest 来模拟整个网络环境，只需要这个 RandomTest 继承 BasicTest 即可。并且你需要重写 handle_packet（）函数。你不仅可以模拟丢包率，还可以模拟包的重复，包出错，延时的存在等等情况。当然，你需要在 tests_to_run（）这个函数中，调用你自己的模拟环境。

提示：如何覆盖 handle_packet（）函数？

在 BasicTest 类中，handle_packet（）函数是这样实现的：

```
def handle_packet(self):
        """
        This method is called whenever the forwarder receives a packet,
        immediately after the packet has been added to the forwarder's input
        queue.

        The default behavior of the base class is to simply copy whatever is in
        the input queue to the output queue, in the order it was received.
        Most tests will want to override this, since this doesn't give you the
        opportunity to do anything tricksy with the packets.

        Note that you should NEVER make any assumptions about how many packets
        are in the in_queue when this method is called -- there could be zero,
        one, or many!
        """

        for p in self.forwarder.in_queue:
            self.forwarder.out_queue.append(p)
        # empty out the in_queue
        self.forwarder.in_queue = []
```

标红色的两句话是关键。所有的包都存放在 self.forwarder.out_queue 这个队列中，你只需要对这个队列里面的数据，进行随机的 remove，就可以模拟丢包。。。剩下的，你可以自由发挥。