**实践报告**

**Task 2: UDP socket programming**

**Part 1:**

1. client 和 server都是基于UDP的命令行程序。client运行在host os,server运行在 guest os。Server端启动后给出一个提示表示正在监听，如图1.1所示。 client端启动时在命令行方式下指定serverlP 和serverPort，若未输入，则给出提示，输入正确的serverlP 和serverPort如图1.2所示：

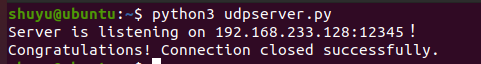


图1.1

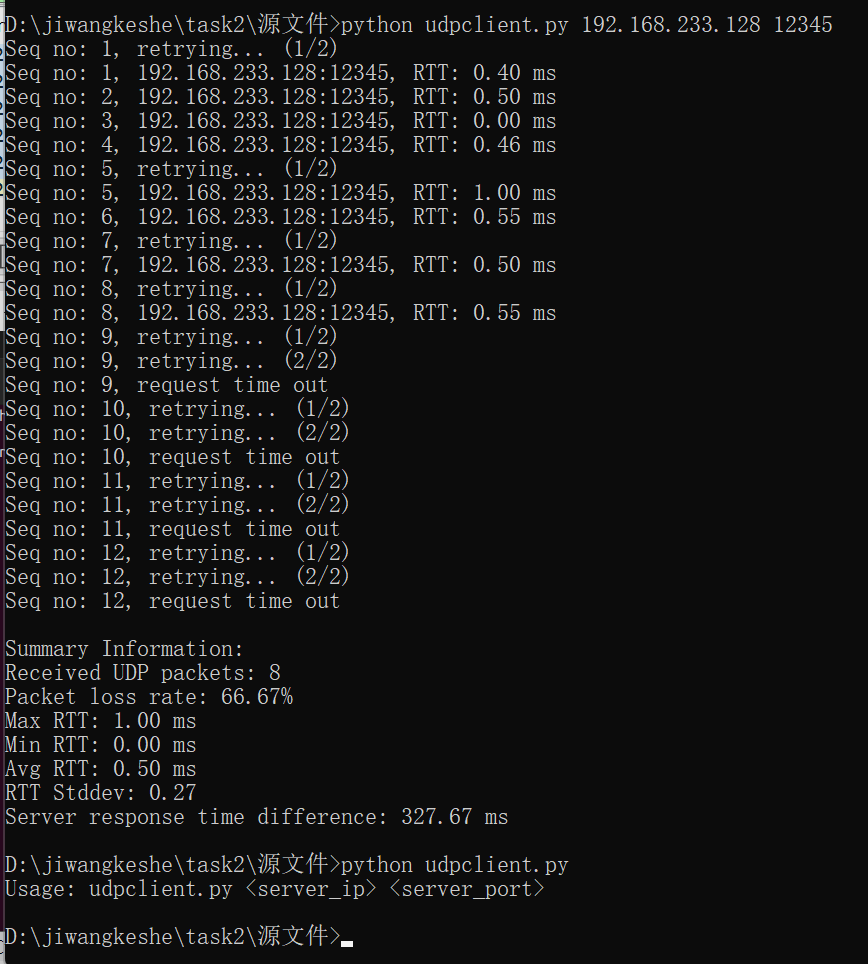


图1.2

1. 模拟TCP连接建立过程，即建立完毕连接以后，才进入到数据传输阶段。
2. client 发送12个request数据包，对每个reqeust 数据包，server端采用随机不响应client请求的方式，来模拟丢包场景，丢包率定义为0.3，如图1.3所示：

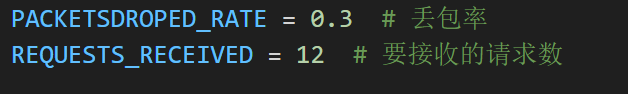


图1.3

1. 给client设定一个超时时间，100ms。在超时时间内:
2. 若能接收到来自于server的响应报文，则计算本次交互的RTT(单位:ms)以及 sever的系统时间(hh-mm-ss)，如图1.4所示：

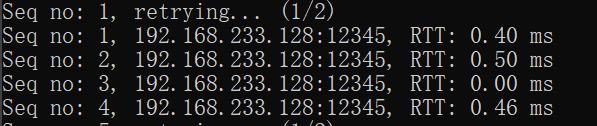


图1.4

1. 若没有收到来自于server的回复，则认为发了丢包。若发生丢包，则重传;两次重传失败，则放弃本次重传，如图1.5所示：

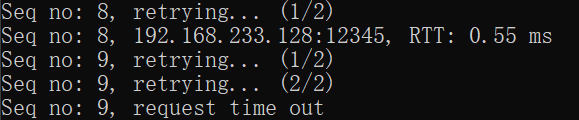


图1.5

1. client端和 server端的交互及本应用层的报文格式，如图1.6，图1.7所示：

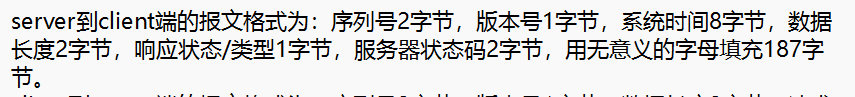


图1.6

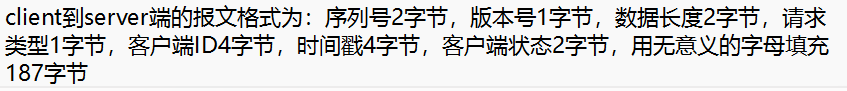


图1.7

1. client端每发送一个报文，若收到了来自于server端的response，则print出“sequence no, serverIP:Port、RTT”。若发生超时，即在100ms内没有收到来自于server 的response，则print出“sequence no,request time out”，如图1.8所示：

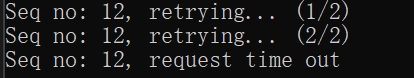


图1.8

1. 当发送完毕12个request 报文后，client端，打印出如下[汇总]信息:

接收到的udp packets数目

丢包率(百分比:1-接收到的udppacket/发送的udppacket)

最大RTT、最小RTT、平均RTT、RTT的标准差

server的整体响应时间(server 最后一次response的系统时间与第一次response的系统时间之差)，如图1.9所示：

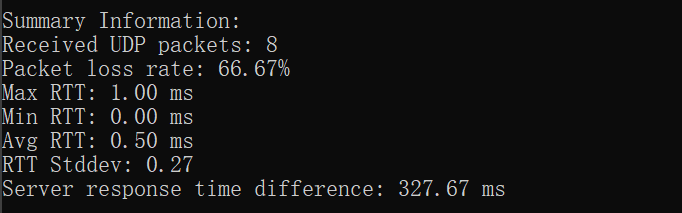


图1.9

1. 模拟TCP连接释放过程，关闭client-server 之间的交互。
2. 在 client-server 交互过程中，使用wireshark抓包，找到有两次重传的报文。wireshark包捕获的截图如图1.10所示：

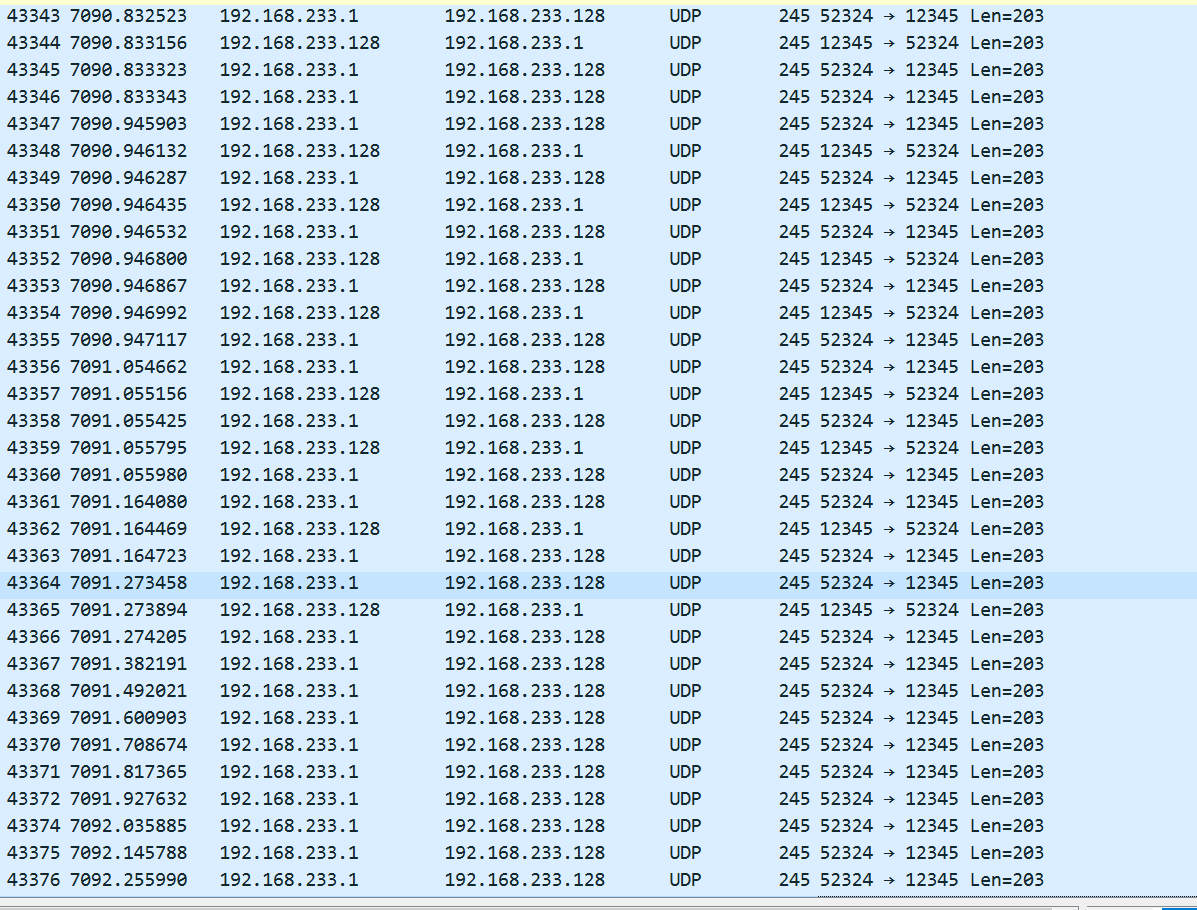


图1.10

43343-43345行为模拟TCP连接建立的过程，设置其Seq.no为0，即前四位0000，其内部的如图1.11，图1.12，图1.13所示：

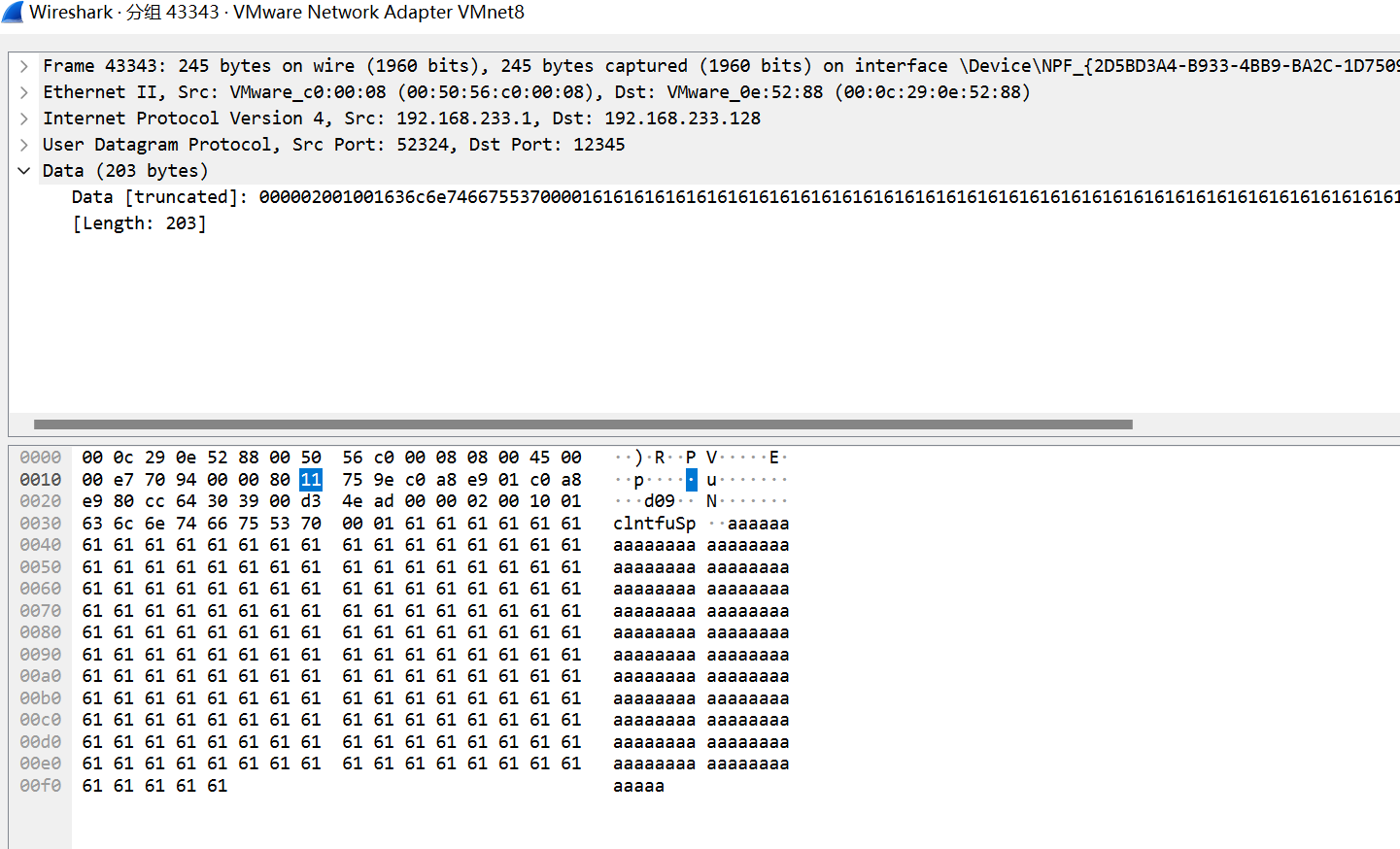


图1.11

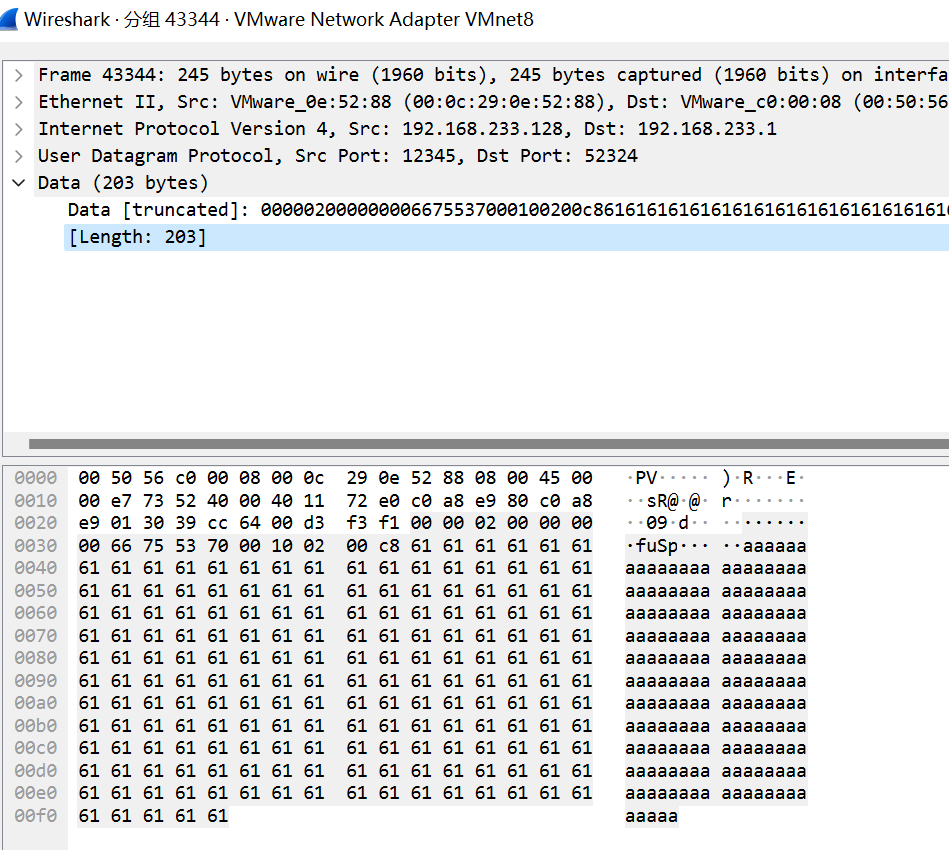


图1.12

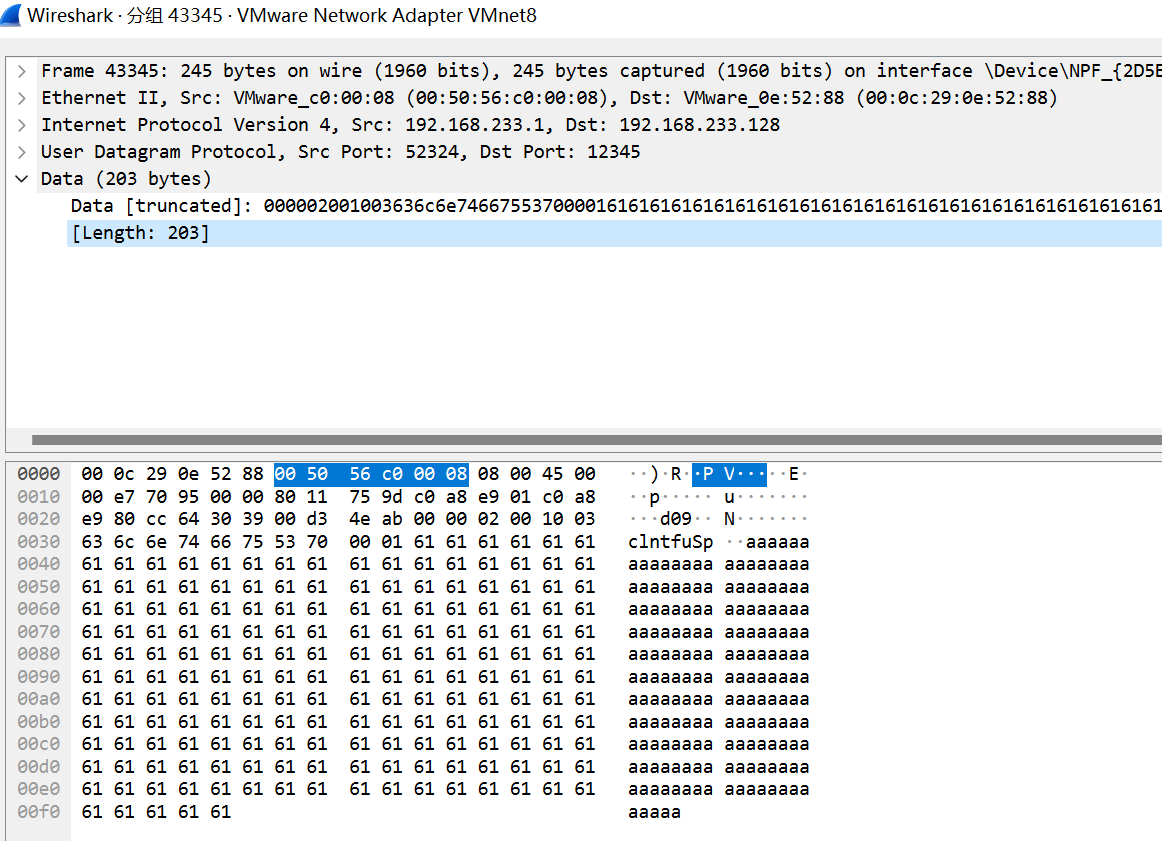


图1.13

43347-43348行为发送Seq.no1的报文以及响应报文，其Seq.no为1，即前四位0001，版本号为2，即第五位和第六位02，发送和响应的内部的报文分别如图1.14，图1.15所示：

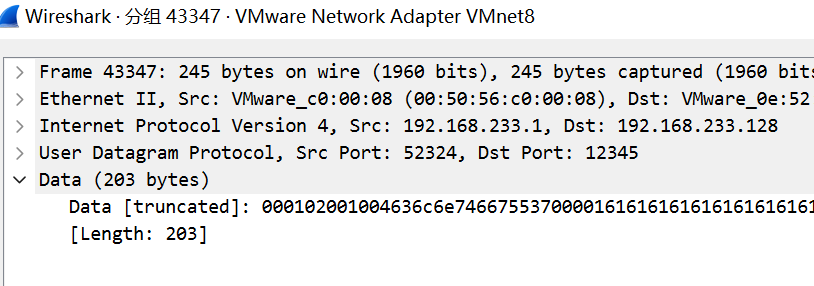


图1.14

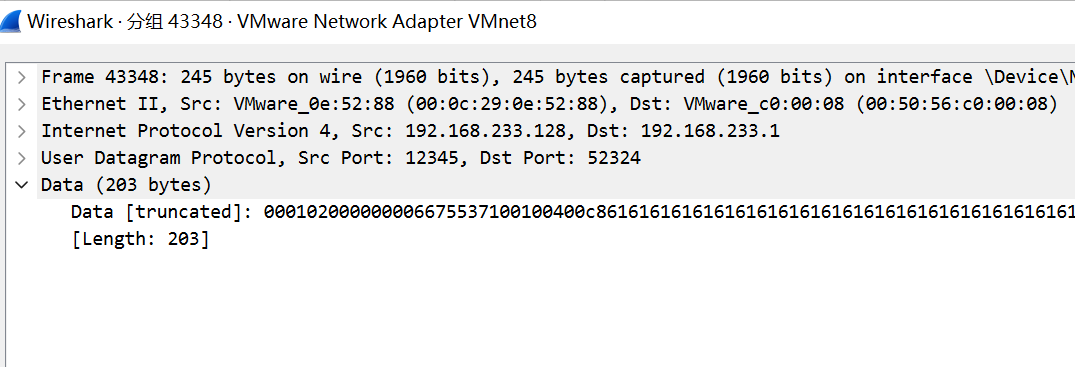


图1.15

43363到43365的报文是重传Seq.no8的过程，重传成功，43364和43365是成功发送和接收Seq.no8的过程；43366-43368的报文为重传Seq.no9的过程，两次重传均失败，如图1.16所示：

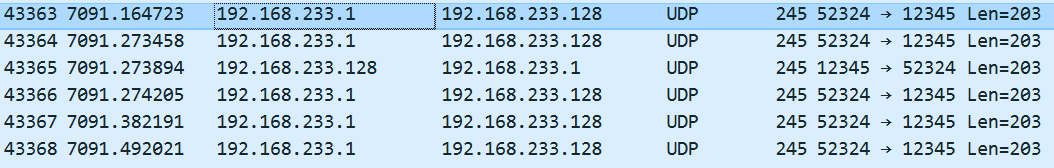


图1.16

43364行的client端发送的请求报文内部如图1.17所示：

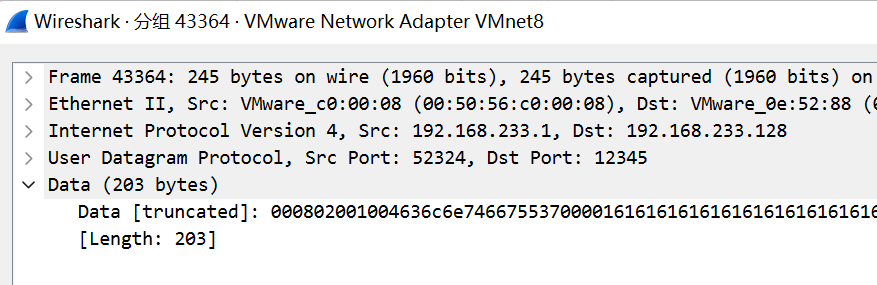


图1.17

43365行的server端发送的回应报文内部如图1.18所示：

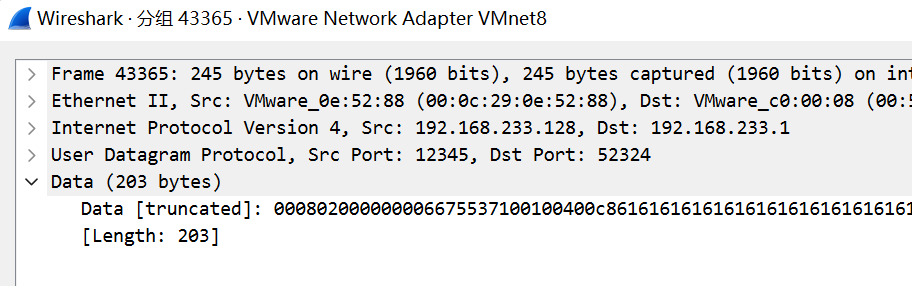


图1.18

43368行的client端发送的请求报文内部如图1.19所示：

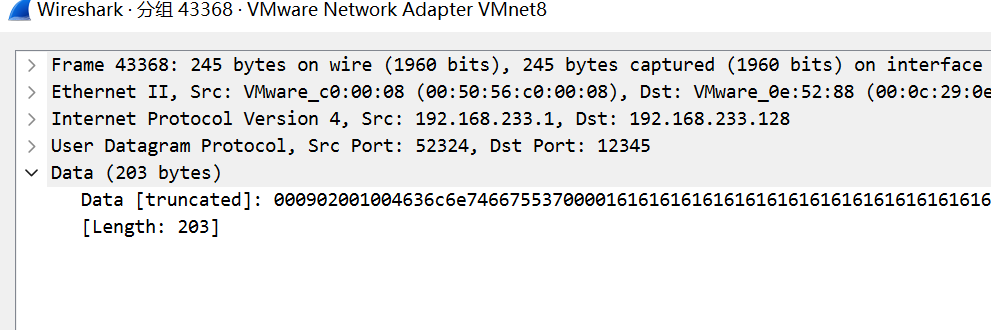


图1.19

**Part 2:**

**实现的关键点和对应的（代码）解决方案**

1. 关键点一：建立连接，模拟TCP的连接建立过程

解决方案：

让客户端发送一个SYN包到服务器，服务器接收到SYN包后，回应一个SYN-ACK包，客户端接收到SYN-ACK包后，发送一个ACK包作为确认。客户端使用 packet\_creat 创建一个SYN包，并通过 client\_socket.sendto 发送，然后，服务器使用 server\_socket.recvfrom 接收SYN包，然后使用 resppacket\_creat 创建并返回一个SYN-ACK包，最后，客户端接收到SYN-ACK包后，使用 packet\_creat 创建一个ACK包，并通过 client\_socket.sendto 发送，如图2.1，图2.2所示：

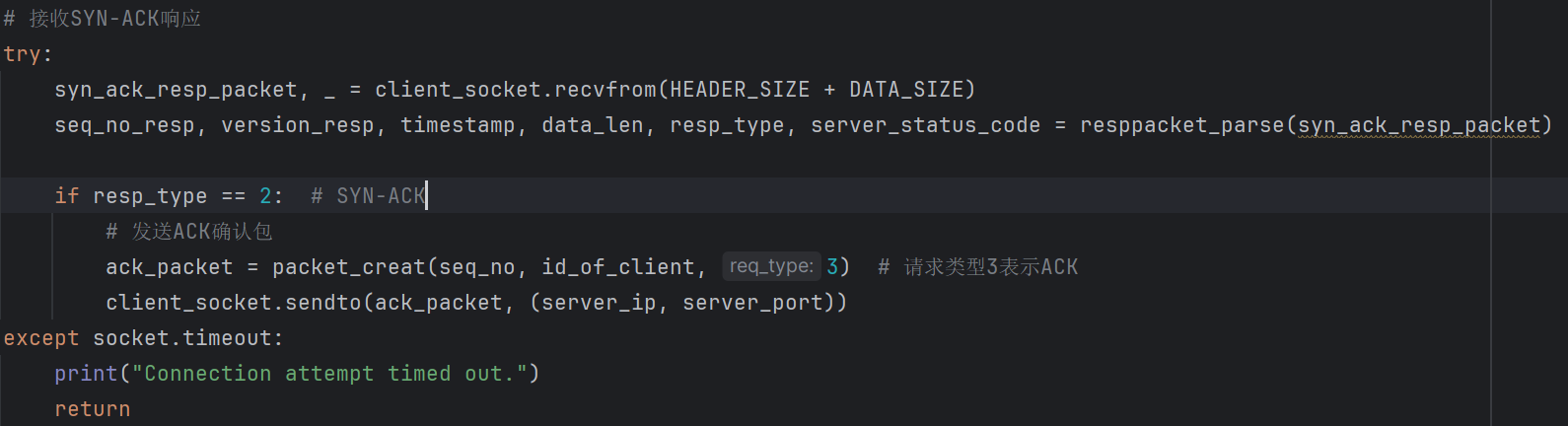


图2.1



图2.2

1. 关键点二，数据传输。

解决方案：

客户端发送12个请求数据包到服务器，服务器接收到数据包后，可能模拟丢包，然后回应一个响应数据包，同时，客户端记录RTT (往返时间)。客户端使用 packet\_creat 创建请求数据包，并通过 client\_socket.sendto 发送，服务器使用 server\_socket.recvfrom 接收数据包，可能模拟丢包，然后使用 resppacket\_creat 创建并返回一个响应数据包，客户端使用 client\_socket.recvfrom 接收响应数据包，并记录RTT，如图2.3所示：



图2.3

1. 关键点三，关闭连接，模拟TCP连接释放的过程。

解决方案：

客户端发送一个FIN包到服务器，请求关闭连接，服务器接收到FIN包后，回应一个FIN-ACK包。服务器发送一个FIN包到客户端，请求关闭连接，客户端接收到FIN包后，回应一个ACK包，然后关闭连接。客户端使用 packet\_creat 创建一个FIN包，并通过 client\_socket.sendto 发送，服务器接收到FIN包后，使用 resppacket\_creat 创建并返回一个FIN-ACK包，然后创建并发送一个FIN包，客户端接收到FIN-ACK和FIN包后，使用 packet\_creat 创建一个ACK包，并通过 client\_socket.sendto 发送，如图2.4，图2.5所示：

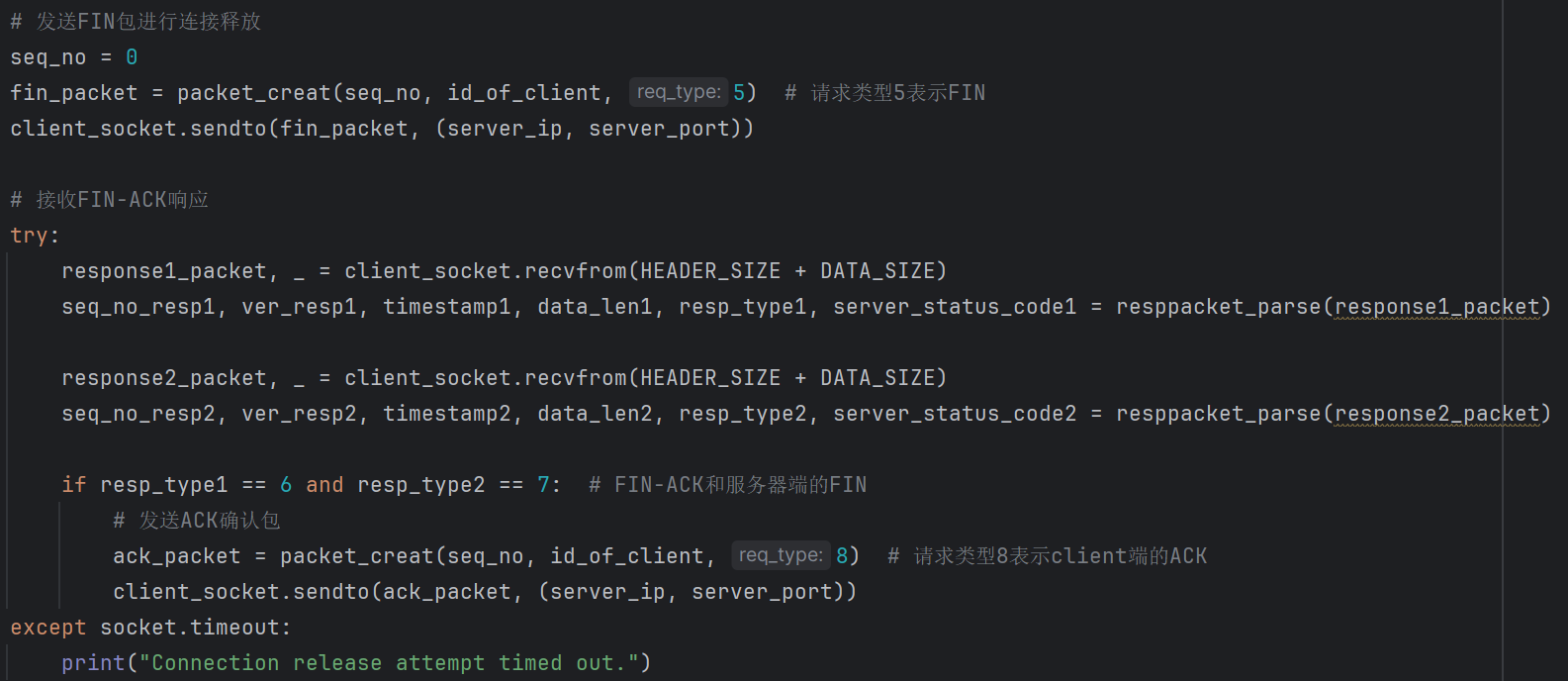


图2.4

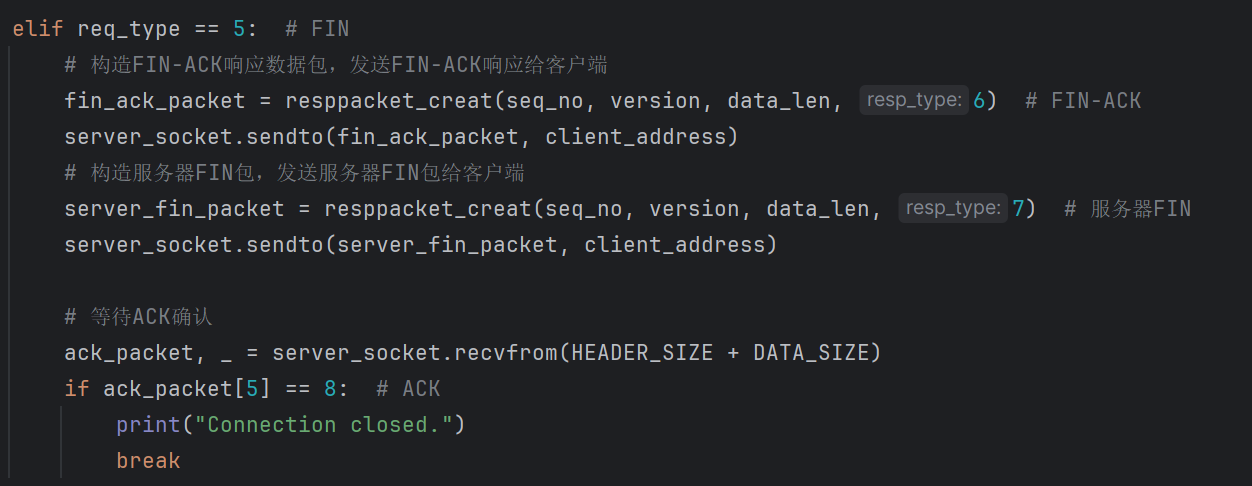


图2.5

1. 关键点四，超时和重传。

解决方案：

客户端在发送数据包后设置超时时间，如果在该时间内没有接收到服务器的响应，则重传数据包。客户端使用 client\_socket.settimeout 设置超时时间，如果在超时时间内没有接收到响应，客户端在一个循环中重新发送数据包，最多重传 RETRIES 次，即2次，如图2.6所示：

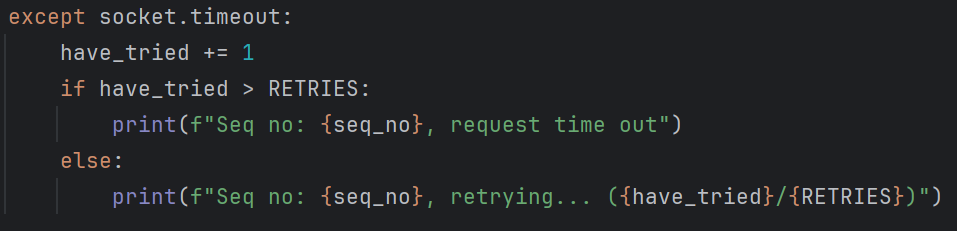


图2.6

1. 关键点五，统计所需要的信息。

解决方案：

客户端计算并显示接收到的数据包数量、丢包率、RTT的最大值、最小值、平均值和标准差。客户端使用 rtt\_sets 存储所有RTT值，并使用Python的 statistics 模块计算最大值、最小值、平均值和标准差，计算丢包率，并显示所有统计信息，如图2.7所示：

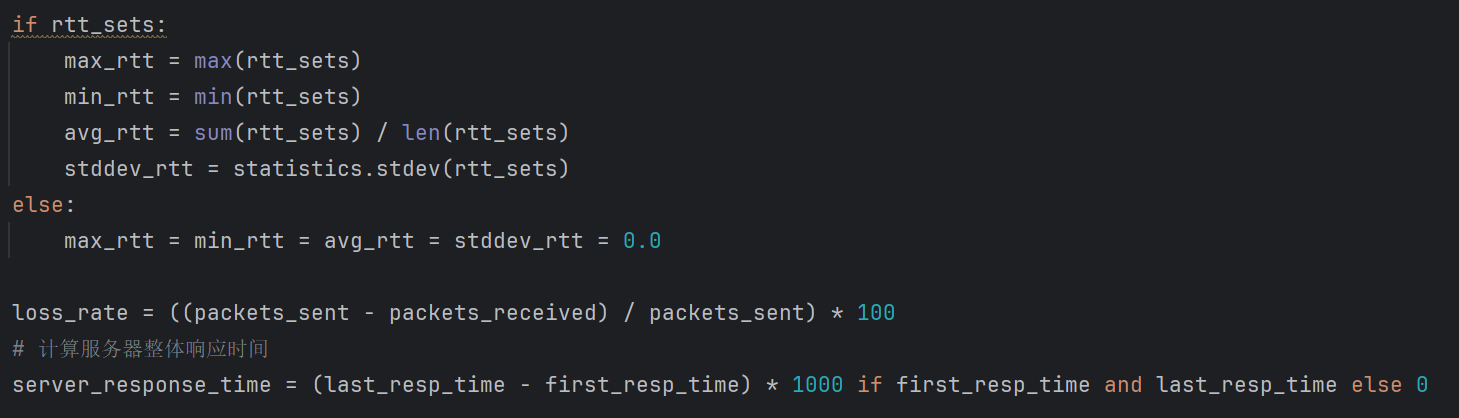


图2.7

**Part 3:**

**运用并掌握的知识点**

在本次的课设实践中，我不仅重新巩固复习了TCP的连接建立和连接释放过程，更能理解其中的本质原理，将其运用到UDP上，模拟了TCP的连接建立三次握手和连接释放四次挥手机制，通过这一过程，我对数据传输的细节有了更清晰的认识。

同时，我通过自主设计数据包的报文格式，深入理解了数据传输过程中的报文格式。通过掌握struct.pack和struct.unpack的使用，我学会了如何确保数据在网络传输中的字节顺序一致性，以及如何通过设置特定的字段来识别和处理传输的数据信息和类型。

此外，我进一步了解和学习了时间管理方面的应用，包括通过获取当前时间创建时间戳和计算RTT，使用time.sleep（）模拟网络延迟和数据包处理时间等以及套接字编程，对利用python的socket库创建和操作套接字，发送和接受UDP数据包。在此基础之上，我对超时和重传机制有了更深刻的理解，通过循环重传机制处理可能会丢失或延迟的数据包。同时，通过设置丢包率，模拟随机丢包，对网络数据传输的复杂性和层层确保数据传输的机制有了更全面的理解。

总之，通过本次的课设实践，我对以往在课堂上所学习到的UDP相关的知识理解更加透彻，对如何构成一个简单的基于UDP的客户端-服务器通信协议加入了自己的理解，对知识点的学习和理解进一步加深，以往的模糊不清的概念和过程通过本次课设实践，上网查询相关的资料以及翻阅课本、笔记，重新学习了这些概念和过程，并加以运用，从理论到实践，让我对网络编程的各个方面都有了更加全面和深入的认识，成就感非常大！

**Part 4:**

**git的url如下：**