udp\_packet\_capture

1. wireshark包捕获截图

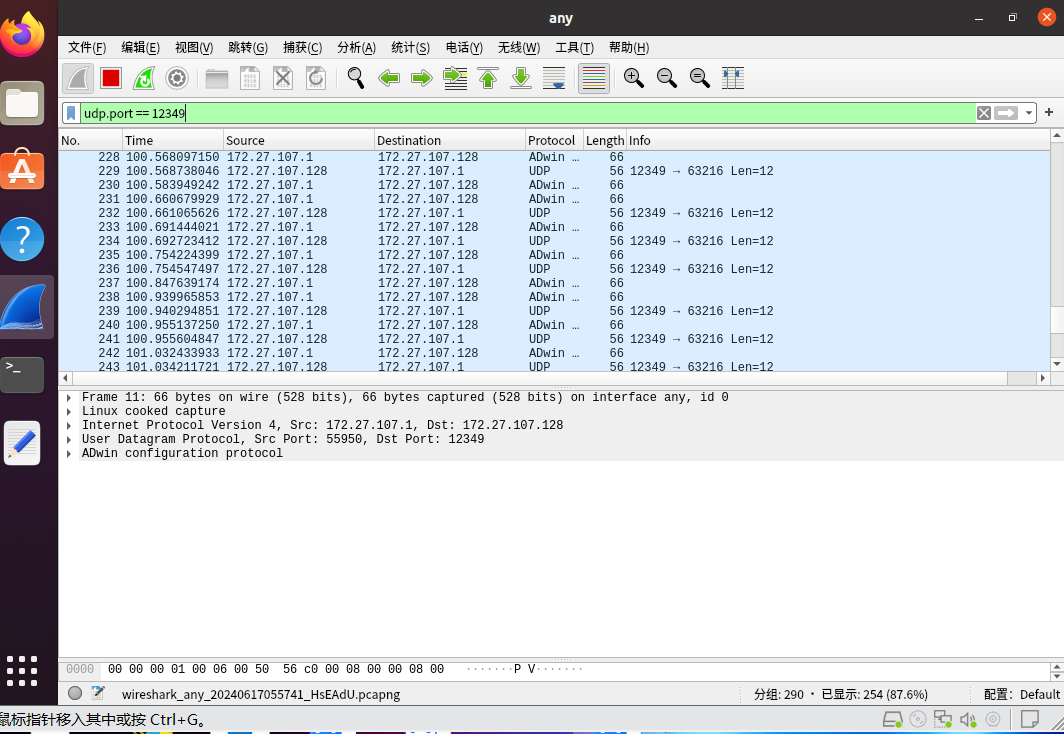


图1 wireshark抓取传输包

为了抓取两次重传的报文，将两次报文传输间隔改为400ms，使重传报文相邻，方便识别

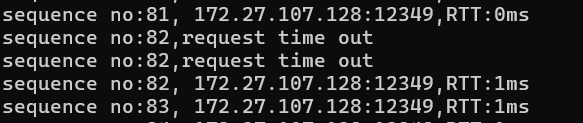


图2 两次重传的报文

图3中，客户端向服务端发送三个报文才收到回复，因为前两次包丢失，第2次重传才收到回复。

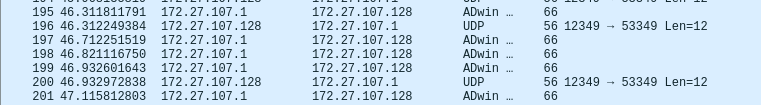


图3 wireshark抓取两次重传的报文

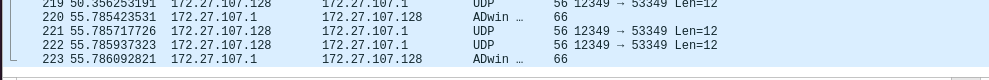


图4 模拟四次挥手连接释放报文抓取

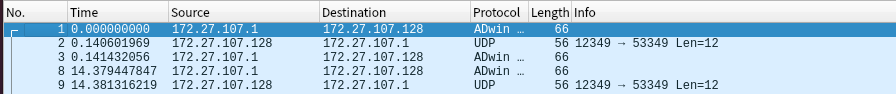


图5 模拟三次握手建立连接报文抓取

1. 实现的关键点与对应代码

客户端：

每发送一个报文，若收到回复，需要打印出序列号、IP和端口号、RTT，若没有，提示超时并打印。

若使用一对一的模式，即发送报文后必须等待收到回复或超时才发送下一个报文，实现简单，但效率低下，且不符合现实要求。

因此需要采取多线程，即为每一个发送的报文单独启动一个Sender进程，通过Thread.Sleep(100)实现以100ms为超时限制。若100ms后收到回复，则打印出序列号、IP和端口号、RTT，进程结束；若没有，通过while循环重新发送报文并开启新一轮的计时。通过time实现记录重传次数，若重传两次都失败了，跳出循环不再重传。

而Receiver进程随着连接建立启动，收到报文后，根据前两个字节的序列号可以知道是对哪个报文的回复。通过flag数组完成数据在Receiver和Sender进程之间的共享，初始全部置为0，若收到回复，则flag[seq]置为1，表明序号为seq的报文收到了回复。

计算RTT的实现，通过client.startTime[seq]=System.currentTimeMillis();记录报文发送的时间，在Receiver进程收到服务端的报文时，endtime= System.currentTimeMillis();记录收到回复报文的时间，即可计算RTT。

服务端：

需要实现与多个客户端同时通讯的情况，因此需要记录每个连接的状态，创建linkip和linkport数组来对应唯一一个客户端程序，同时beginflag、connectflag、endflag三个数组来记录该客户端当前处于何种状态。

收到报文时，首先根据报文的源IP和源端口号确认该客户端是否在记录中，若不在，则增添一条新的记录，将beginflag置为1，connectflag和endflag置为0，表明该客户端处于连接建立阶段；若在记录中，根据三个数组的值确认客户端所处阶段，然后分为以下情况：

1. 收到SYN标志位值为1的包，说明是请求建立连接的包，若当前不处于连接建立阶段，提示连接错误；否则正常建立连接，实现第二次握手回复。
2. 收到FIN标志位值为1的包，说明是请求释放连接的包，回复第二次挥手与第三次挥手。
3. 收到ACK标志位值为1的包，可能是第三次握手也可能是第四次挥手，则根据所处阶段做不同操作，若为连接建立阶段，提示连接成功，将beginflag置为0，connectflag置为1，表明进入了数据传输阶段；若为连接释放阶段，提示释放成功，将beginflag置为1，endflag置为0，表明重新进入了连接建立阶段。
4. 若以上三种情况都不符合，说明是一个携带数据传输的包，若当前不处于数据传输阶段，提示传输错误；否则回复一个序列号与该包相同的包。
5. 掌握到的知识点
6. UDP传输与接收数据的代码实现方式
7. 应用层自定义报文的方法
8. 各种RTT的计算方式
9. 三次握手建立连接和四次挥手释放连接的流程
10. 超时重传的实现原理
11. git的url

https://github.com/mtreeshan/udp-socket-programming