南开大学

Nankai University

南　　开　　大　　学

计 算 机 学 院

计算机网络实验报告

基于 UDP 服务设计可靠传输协议 Part3

2011763 黄天昊

年级：2020 级

专业：计算机科学与技术

指导教师：徐敬东、张建忠

2022 年 12 月 25 日

摘要

在基于 UDP 实现可靠数据传输协议 Part3 的设计中，主要工作为实现了双线程机制以及 New Reno 拥塞控制算法。其中，对于双线程的实现，加锁和解锁的位置需要仔细考虑，而对于 New Reno 算法，由于我们实现的通讯服务毕竟与 TCP 服务有所区别，因此我对 TCP 中使用的 New Reno 算法快速重传部分的实现细节也做了改进。

关键字：UDP 可靠数据传输; New Reno 拥塞控制算法; 多线程

目录

一、 双线程机制的实现

要实现更高的性能、更快的传输速度，不管是客户端还是服务器端，将收与发作为不同的线程同步处理必不可少。因此，基于上次的代码框架，我增添了双线程机制，实现了收发同步。

代码如下：

main 函数中开辟线程的代码

```
// 开辟收发线程
hThread[0] = CreateThread(NULL, 0, send, NULL, 0, NULL);
hThread[1] = CreateThread(NULL, 0, recv, NULL, 0, NULL);
WaitForMultipleObjects(2, hThread, TRUE, INFINITE); // 等待收发线程全部结束才进行主线程
CloseHandle(hThread[0]);
CloseHandle(hThread[1]);
```

负责发送数据包的线程代码

```
DWORD WINAPI send(LPVOID lparam) {
        sendBase = 0;
        nextSeqNum = 0;
        selectiveRepeatBuffer = new char* [WINDOW_SIZE];
        for (int i = 0; i < WINDOW_SIZE; i++)selectiveRepeatBuffer[i] = new char[sizeof(Packet)]; // char[32][1048]
        for (int i = 0; i < WINDOW_SIZE; i++)timerID[i] = -1; // 双线程启动的时候会有一些同步的问题，这里先设为-1区别一下
        clock_t start = clock();

        // 先发一个记录文件名的数据包，并设置HEAD标志位为1，表示开始文件传输
        Packet* headPkt = new Packet;
        printTime();
        cout << "发送文件头数据包..." << endl;
        headPkt->setHEAD(0, fileSize, fileName);
        headPkt->checksum = checksum((uint32_t*)headPkt);
        printSendPacketMessage(headPkt); // 检查一下文件的各个内容
        cout << endl;
        err = sendto(socketClient, (char*)headPkt, BUFFER_SIZE, 0, (SOCKADDR*)&socketAddr, sizeof(SOCKADDR));
        if (err == SOCKET_ERROR) { // 这里没有实现文件头的缺失重传机制
                cout << "Send Packet failed with ERROR: " << WSAGetLastError() << endl;
        }

        // 开始发送装载文件的数据包
        printTime();
        cout << endl << "开始发送文件数据包..." << endl;
        Packet* sendPkt = new Packet;
        while (sendBase < packetNum) {
                mutexLock.lock();
```

```
28                while (nextSeqNum < sendBase + min(cwnd, WINDOW_SIZE) &&
                      nextSeqNum < packetNum) {  // 只要下一个要发送的分组序号
                      还在窗口内，就继续发
29                    if (nextSeqNum == packetNum - 1) {  // 如果是最后一个
                          包
30                        sendPkt->setTAIL();
31                        sendPkt->fillData(nextSeqNum, fileSize -
                              nextSeqNum * DATA_AREA_SIZE, fileBuffer +
                               nextSeqNum * DATA_AREA_SIZE);
32                        sendPkt->checksum = checksum((uint32_t*)
                              sendPkt);
33                    }
34                    else {  // 正常的1024Bytes数据包
35                        sendPkt->fillData(nextSeqNum, DATA_AREA_SIZE,
                               fileBuffer + nextSeqNum * DATA_AREA_SIZE
                              );
36                        sendPkt->checksum = checksum((uint32_t*)
                              sendPkt);
37                    }
38                    memcpy(selectiveRepeatBuffer[nextSeqNum - sendBase],
                          sendPkt, sizeof(Packet));  // 存入缓冲区
39                    sendPacket(sendPkt);
40
41                    timerID[nextSeqNum - sendBase] = SetTimer(NULL, 0,
                          TIME_OUT, (TIMERPROC)resendPacket);  // 启动计时
                          器，这里nIDEvent=0，是因为窗口句柄设为NULL的情况
                          下这里填什么都无所谓，反正它会重新分配，返回值才
                          是真正的nIDEvent
42                    cout << "第 " << nextSeqNum << " 对应的timerID为 " <<
                           timerID[nextSeqNum - sendBase] << endl;
43                    nextSeqNum++;
44
45                    Sleep(10);  // 睡眠10ms，包与包之间有一定的时间间隔
46                }
47                mutexLock.unlock();
48
49                while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {  // 以查看
                      的方式从系统中获取消息，可以不将消息从系统中移除，是非阻
                      塞函数；当系统无消息时，返回FALSE，继续执行后续代码。
50                    if (msg.message == WM_TIMER) {  // 定时器消息
51                        DispatchMessage(&msg);
52                    }
53                }
54            }
55
56            clock_t end = clock();
57            printTime();
58            cout << "文件发送完毕，传输时间为: " << (end - start) /
```

```
            CLOCKS_PER_SEC << "s" << endl;
59          cout << "吞吐率为: " << ((float)fileSize) / ((end - start) /
                CLOCKS_PER_SEC) << " Bytes/s " << endl << endl;
60          cout << endl << "********************************" << endl << endl;
61
62          return 0;
63  }
```

负责接收数据包的线程代码

```
1   DWORD WINAPI recv(LPVOID lparam) {
2           Packet* recvPkt = new Packet;
3           while (sendBase < packetNum) {
4                   err = recvfrom(socketClient, (char*)recvPkt, BUFFER_SIZE, 0,
                        (SOCKADDR*)&(socketAddr), &socketAddrLen);
5                   if (err > 0) {
6                           int ack = recvPkt->ack;
7                           printReceivePacketMessage(recvPkt);
8
9                           mutexLock.lock();
10                          switch (state) {
11                          case SLOW_START:
12                                  cout << "======================== 慢启动阶段
                                        ========================" << endl;
13                                  cout << "Send Window Size: " << min(cwnd,
                                        WINDOW_SIZE);
14                                  cout << " sendbase: " << sendBase;
15                                  cout << " ack; " << ack;
16                                  cout << " cwnd: " << cwnd << " ssthresh: " <<
                                        ssthresh << endl;
17                                  if (ack >= sendBase && (ack - sendBase) < min
                                        (cwnd, WINDOW_SIZE)) {  // 在窗口内并且是
                                        一个还没收到的ACK
18                                          if (cwnd <= ssthresh) {
19                                                  cwnd++;
20                                                  cout << "************ 慢启动
                                                        阶段接收到新的ACK, cwnd++
                                                        ************" << endl;
21                                          }
22                                          else {
23                                                  state = CONGESTION_AVOIDANCE;
24                                                  cout << "************ 慢启动
                                                        阶段结束, 进入拥塞避免阶
                                                        段 ************" << endl;
25                                          }
26
27                                          if (ack == sendBase)
                                                duplicateACKCount = 0;  // 收到
                                                sendBase的ACK, 置零
```

```
28                              else duplicateACKCount++;
29
30                              if (duplicateACKCount == 3) {  // 如果有三个ack都不是sendBase，开始提前重传sendBase
31                                  cout << "********** 连续三个ack都不是sendBase，开始提前重传 **********" << endl;
32                                  fastRetransmission(sendBase);
33                                  ssthresh = cwnd / 2 == 0 ? 1 : cwnd / 2;
34                                  cwnd = ssthresh + 3;
35                                  state = FAST_RECOVERY;
36                                  duplicateACKCount = 0;
37                              }
38                          }
39                          break;
40
41                      case CONGESTION_AVOIDANCE:
42                          cout << "======================== 拥塞避免阶段 ========================" << endl;
43                          cout << "Send Window Size: " << min(cwnd, WINDOW_SIZE);
44                          cout << " sendbase: " << sendBase;
45                          cout << " ack: " << ack;
46                          cout << " cwnd: " << cwnd << " ssthresh: " << ssthresh << " cwndFlag: " << cwndFlag << endl;
47                          if (ack >= sendBase && (ack - sendBase) < min(cwnd, WINDOW_SIZE)) {  // 在窗口内并且是一个还没收到的ACK
48                              cwndFlag++;
49                              // 这里设置cwnd或者给定值应该都可以，但是cwnd快
50                              if (cwndFlag == cwnd) {
51                                  cwnd++;
52                                  cwndFlag = 0;
53                                  cout << "********** 拥塞避免阶段，cwnd线性递增1 **********" << endl;
54                              }
55
56                              if (ack == sendBase)
                                    duplicateACKCount = 0;  // 收到sendBase的ACK，置零
57                              else duplicateACKCount++;
58
```

```
59                                  if (duplicateACKCount == 3) {  // 如果有三个ack都不是sendBase，开始提前重传sendBase
60                                      cout << "************ 连续三个ack都不是sendBase，开始提前重传 ************" << endl;
61                                      fastRetransmission(sendBase);
62                                      ssthresh = cwnd / 2 == 0 ? 1 : cwnd / 2;
63                                      cwnd = ssthresh + 3;
64                                      state = FAST_RECOVERY;
65                                      duplicateACKCount = 0;
66                                  }
67                              }
68                              break;
69
70                          case FAST_RECOVERY:
71                              cout << "======================== 快速恢复阶段 ========================" << endl;
72                              cout << "Send Window Size: " << min(cwnd, WINDOW_SIZE);
73                              cout << " sendbase: " << sendBase;
74                              cout << " ack: " << ack;
75                              cout << " cwnd: " << cwnd << " ssthresh: " << ssthresh << endl;
76                              if (ack >= sendBase && (ack - sendBase) < min(cwnd, WINDOW_SIZE)) {  // 在窗口内并且是一个还没收到的ACK
77                                  if (ack == sendBase) {
78                                      cwnd = ssthresh;
79                                      duplicateACKCount = 0;
80                                      state = CONGESTION_AVOIDANCE;
81                                  }
82                                  else {
83                                      cwnd++;
84                                  }
85                              }
86                              break;
87
88                          default:
89                              cout << "ERROR STATE!" << endl;
90                              break;
91                          }
92
93                          ackHandler(ack);  //  处理ACK
94                          mutexLock.unlock();
95                      }
```

```
96          }
97          return 0;
98  }
```

需要注意的是，由于两个线程共享一些全局变量，不可避免的会发生读写冲突，因此我们需要给相应的代码加锁和解锁，以防同一变量在某一时刻发生数据冒险。这里可以看到，在 recv 函数和 send 函数中，代码的相应位置使用 C++ 的 mutex 库中的 lock(), unlock() 函数进行了加锁和解锁。

同时，还有一个实现的细节问题，如果在某个函数中进行了加锁，在加锁的代码段是不能再出现加锁命令的，比如在这段代码调用了另一个函数，而被调用的函数也有加锁命令，这便会产生死锁问题，程序会崩溃。

二、 New Reno 拥塞控制算法实现

对于拥塞控制算法，我选择的是 New Reno 算法进行实现，其状态机如图1所示：

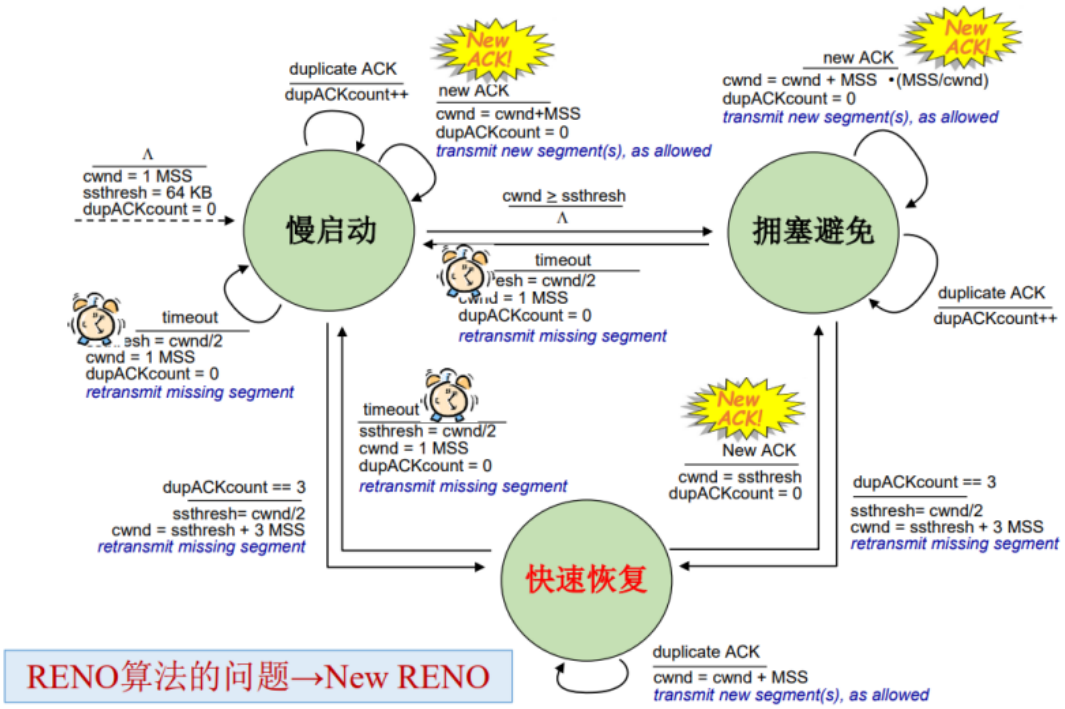

图 1: New Reno 算法有限状态机

我们可以看到具体的 Reno 算法的有限状态机转换的过程，首先是慢启动阶段，这时的拥塞窗口为 1，我们需要注意的是每次的发送过程之中，发送端的窗口大小除了要考虑拥塞窗口的大小之外，还要考虑接收端的消息接收窗口，在慢启动阶段，每次接收到新的 ACK 响应消息，那么拥塞窗口 cwnd 就会递增 +1，也就是说在一个 RTT 后会翻倍，当慢启动阶段的 cwnd 大小增大到大于 ssthresh 阈值时，就会进入拥塞避免阶段（但是也有例外，例如在慢启动阶段接收到三次 ACK 消息，或者更加极端的是在慢启动阶段窗口尚未达到 3 的大小时的情形），特殊情况将在第三部分进行分析。

在拥塞避免阶段，每隔一个 RTT 时间 cwnd 递增 +1，线性增长的方式如图2展示：

拥塞避免算法

```
/* slowstart is over       */
/* cwnd >= ssthresh     */
Until (loss event) {
  every w segments ACKed:
     cwnd ++
  }
ssthresh = cwnd /2
If (loss detected by timeout) {
   cwnd = 1
   perform slowstart }
If (loss detected by triple
            duplicate ACK)
   cwnd = ssthresh + 3
```

图 2: 实现 cwnd 线性增长伪代码

对于 TCP 来说，如果处于拥塞避免阶段，若出现连续的 3 个重复的 ACK 消息就会进入快速恢复阶段，这时阈值减半，cwnd 重设，并在退出快速恢复阶段后再重新设置为阈值，但是我实现的选择重传机制明显不适合这个算法逻辑，因为选择重传算法允许序号靠后的分组先被接收，接收端的 ACK 序号一般不会重复，因此不能机械地采用 TCP 的方法去实现快速重传。在这里，我采用如果“连续三个 ACK 序号都不等于目前已发送但未受到 ACK 的最小分组序号，则进行快速重传”的算法来进行改进。

如图3为选择重传机制下文件传输可能出现的情况，可以看到此时第 6、7、8、9 号分组皆已被接收端收到并返回 ACK 分组，但是第 5 号分组数据包途中丢失。由于缺少 ACK=5 的数据包，滑动窗口不会向前移动，需要等到其对应的计时器超时触发重传并得到接收端相应的 ACK 数据包之后，滑动窗口才会再次移动。当网络中的丢包率较高时，这无疑会大大增加文件传输所需的时间。

但是，如果采用上述的改进思路，那么当 ACK=8 的数据包到达发送端时，发送端便会触发快速重传机制，重新发送第 5 号分组数据包，而不必等到其对应计时器到时触发重传，在一定程度上降低了文件传输的时间。

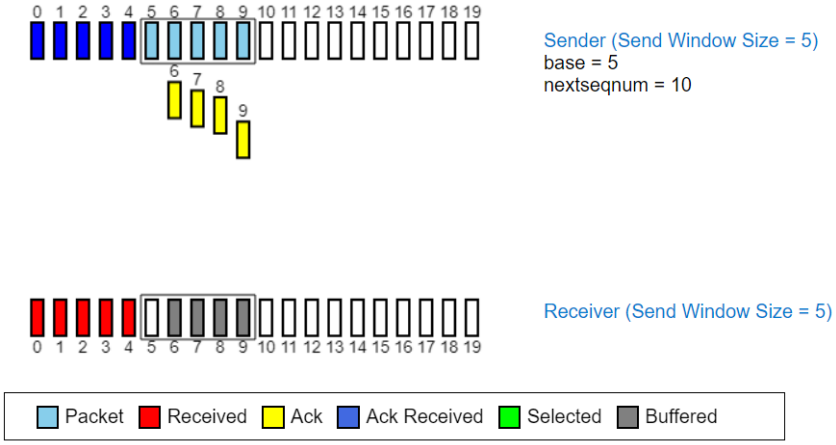

图 3: 选择重传机制下文件传输可能出现的情况

实现的 New Reno 算法代码如下:

New Reno 算法代码

```
DWORD WINAPI recv(LPVOID lparam) {
        Packet* recvPkt = new Packet;
        while (sendBase < packetNum) {
                err = recvfrom(socketClient, (char*)recvPkt, BUFFER_SIZE, 0,
                    (SOCKADDR*)&(socketAddr), &socketAddrLen);
                if (err > 0) {
                        int ack = recvPkt->ack;
                        printReceivePacketMessage(recvPkt);

                        mutexLock.lock();
                        switch (state) {
                        case SLOW_START:
                                cout << "======================== 慢启动阶段
                                    ========================" << endl;
                                cout << "Send Window Size: " << min(cwnd,
                                    WINDOW_SIZE);
                                cout << " sendbase: " << sendBase;
                                cout << " ack; " << ack;
                                cout << " cwnd: " << cwnd << " ssthresh: " <<
                                     ssthresh << endl;
                                if (ack >= sendBase && (ack - sendBase) < min
                                    (cwnd, WINDOW_SIZE)) {  // 在窗口内并且是
                                    一个还没收到的ACK
                                        if (cwnd <= ssthresh) {
                                                cwnd++;
                                                cout << "************ 慢启动
                                                    阶段接收到新的ACK, cwnd++
                                                     ************" << endl;
                                        }
                                        else {
                                                state = CONGESTION_AVOIDANCE;
                                                cout << "************ 慢启动
                                                    阶段结束, 进入拥塞避免阶
                                                    段 ************" << endl;
                                        }

                                        if (ack == sendBase)
                                            duplicateACKCount = 0;  // 收到
                                            sendBase的ACK, 置零
                                        else duplicateACKCount++;

                                        if (duplicateACKCount == 3) {  // 如
                                            果有三个ack都不是sendBase, 开始提
                                            前重传sendBase
                                                cout << "************ 连续三
                                                    个ack都不是sendBase, 开始
                                                    提前重传 ************" <<
```

```
                                        endl;
32                                  fastRetransmission(sendBase);
33                                  ssthresh = cwnd / 2 == 0 ? 1
                                        : cwnd / 2;
34                                  cwnd = ssthresh + 3;
35                                  state = FAST_RECOVERY;
36                                  duplicateACKCount = 0;
37                              }
38                          }
39                          break;
40
41                      case CONGESTION_AVOIDANCE:
42                          cout << "======================== 拥塞避免阶
                                段 ========================" << endl;
43                          cout << "Send Window Size: " << min(cwnd,
                                WINDOW_SIZE);
44                          cout << " sendbase: " << sendBase;
45                          cout << " ack; " << ack;
46                          cout << " cwnd: " << cwnd << " ssthresh: " <<
                                ssthresh << " cwndFlag: " << cwndFlag <<
                                endl;
47                          if (ack >= sendBase && (ack - sendBase) < min
                                (cwnd, WINDOW_SIZE)) {  // 在窗口内并且是
                                一个还没收到的ACK
48                              cwndFlag++;
49                              // 这里设置cwnd或者给定值应该都可以,
                                    但是cwnd快
50                              if (cwndFlag == cwnd) {
51                                  cwnd++;
52                                  cwndFlag = 0;
53                                  cout << "************ 拥塞避
                                        免阶段, cwnd线性递增1
                                        ************" << endl;
54                              }
55
56                              if (ack == sendBase)
                                    duplicateACKCount = 0;  // 收到
                                    sendBase的ACK, 置零
57                              else duplicateACKCount++;
58
59                              if (duplicateACKCount == 3) {  // 如
                                    果有三个ack都不是sendBase, 开始提
                                    前重传sendBase
60                                  cout << "************ 连续三
                                        个ack都不是sendBase, 开始
                                        提前重传 ************" <<
                                        endl;
61                                  fastRetransmission(sendBase);
```

```
62                                          ssthresh = cwnd / 2 == 0 ? 1
                                                : cwnd / 2;
63                                          cwnd = ssthresh + 3;
64                                          state = FAST_RECOVERY;
65                                          duplicateACKCount = 0;
66                                      }
67                                  }
68                                  break;
69
70                              case FAST_RECOVERY:
71                                  cout << "======================== 快速恢复阶
                                        段 ========================" << endl;
72                                  cout << "Send Window Size: " << min(cwnd,
                                        WINDOW_SIZE);
73                                  cout << " sendbase: " << sendBase;
74                                  cout << " ack; " << ack;
75                                  cout << " cwnd: " << cwnd << " ssthresh: " <<
                                         ssthresh << endl;
76                                  if (ack >= sendBase && (ack - sendBase) < min
                                        (cwnd, WINDOW_SIZE)) {  // 在窗口内并且是
                                        一个还没收到的ACK
77                                      if (ack == sendBase) {
78                                          cwnd = ssthresh;
79                                          duplicateACKCount = 0;
80                                          state = CONGESTION_AVOIDANCE;
81                                      }
82                                      else {
83                                          cwnd++;
84                                      }
85                                  }
86                                  break;
87
88                              default:
89                                  cout << "ERROR STATE!" << endl;
90                                  break;
91                              }
92
93                              ackHandler(ack);  //  处理ACK
94                              mutexLock.unlock();
95                      }
96              }
97              return 0;
98      }
99
100     void sendPacket(Packet* sendPkt) {  // 封装了一下发送数据包的过程
101             cout << "发送第 " << sendPkt->seq << " 号数据包";
102             printWindow();
103             printSendPacketMessage(sendPkt);  // 检查一下文件的各个内容
```

```
104         cout << endl;
105         err = sendto(socketClient, (char*)sendPkt, BUFFER_SIZE, 0, (SOCKADDR
                *)&socketAddr, sizeof(SOCKADDR));
106         if (err == SOCKET_ERROR) {
107             cout << "Send Packet failed with ERROR: " << WSAGetLastError
                    () << endl;
108         }
109 }
110
111 void resendPacket(HWND hwnd, UINT uMsg, UINT idEvent, DWORD dwTime) {  // 重
        传函数
112         // cout << endl << "resend" << " Timer ID " << idEvent << endl <<
                endl;
113         // mutexLock.lock();  这里还不能加锁，会有死锁问题
114         unsigned int seq = 0;
115         for (int i = 0; i < min(cwnd, WINDOW_SIZE); i++) {  // 找到是哪个
                Timer超时了
116             if (timerID[i] == idEvent && timerID[i] != 0) {
117                 seq = i + sendBase;
118                 break;
119             }
120         }
121         cout << "第 " << seq << " 号数据包对应的计时器超时，重新发送" << endl
                ;
122
123         Packet* resendPkt = new Packet;
124         memcpy(resendPkt, selectiveRepeatBuffer[seq - sendBase], sizeof(
                Packet));  // 从缓冲区直接取出来
125         sendPacket(resendPkt);
126         cout << endl;
127
128         ssthresh = cwnd / 2 == 0 ? 1 : cwnd / 2;  // 保证最小为1
129         cwnd = 1;
130         state = SLOW_START;  // 有超时就回到慢启动状态
131         cout << "======================== 回到慢启动阶段
                ========================" << endl;
132         // mutexLock.unlock();
133 }
134
135 void fastRetransmission(unsigned int seq) {  // 快速重传
136         cout << "[快速重传]";
137         Packet* sendPkt = new Packet;
138         memcpy(sendPkt, selectiveRepeatBuffer[seq - sendBase], sizeof(Packet)
                );  // 从缓冲区直接取出来
139         sendPacket(sendPkt);
140         cout << endl;
141 }
142
```

```
143 void ackHandler(unsigned int ack) {  // 处理来自接收端的ACK
144         if (ack >= sendBase && (ack - sendBase) < WINDOW_SIZE) {  // 如果ack
                分组序号在窗口内
145             cout << "KillTimer " << timerID[ack - sendBase] << " ack=" <<
                    ack << " sendbase=" << sendBase << endl << endl;
146             KillTimer(NULL, timerID[ack - sendBase]);
147             timerID[ack - sendBase] = 0;  // timerID置零
148 
149             if (ack == sendBase) {  // 如果恰好=sendBase，那么sendBase移
                    动到具有最小序号的未确认分组处
150                 for (int i = 0; i < WINDOW_SIZE; i++) {
151                     if (timerID[i]) break;  // 遇到一个有计时器的
                            停下来(如果当前timerID数组元素不为0，说明
                            有计时器在使用)
152                     sendBase++;  // sendBase后移
153                 }
154                 int offset = sendBase - ack;
155                 for (int i = 0; i < WINDOW_SIZE - offset; i++) {
156                     timerID[i] = timerID[i + offset];  // timerID
                            也得平移，不然对不上了
157                     timerID[i + offset] = -1;
158                     memcpy(selectiveRepeatBuffer[i],
                            selectiveRepeatBuffer[i + offset], sizeof
                            (Packet));  // 缓冲区也要平移
159                 }
160                 for (int i = WINDOW_SIZE - offset; i < WINDOW_SIZE; i
                    ++) {
161                     timerID[i] = -1;  // 状态-1表示目前对应位置没
                            有数据包被发出，状态0表示对应位置数据包已
                            发送并被ACK
162                 }
163             }
164         }
165 }
```


三、 可靠 UDP 传输流程展示

本节进行实现了选择重传协议的基于 UDP 的可靠传输协议实际收发流程的展示。

首先是三次握手的 log 信息:

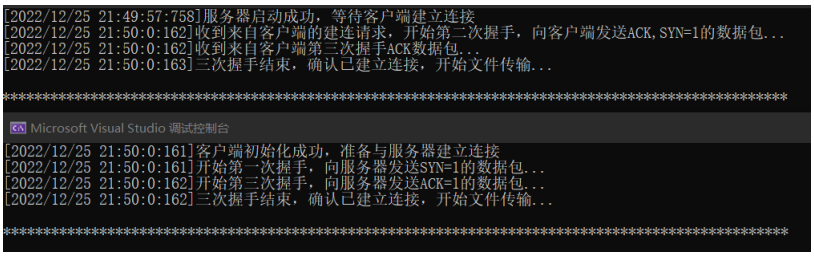

图 4: 服务器接收数据包部分过程 log 信息打印

接下来是文件传输的过程：
刚开始，发送端处于慢启动状态，cwnd 每次 +1

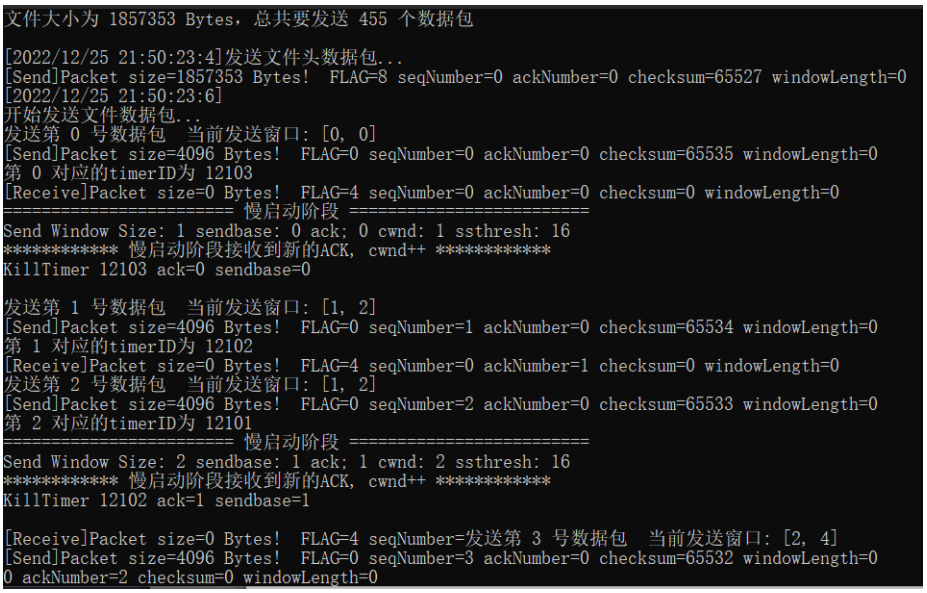

图 5: 发送端慢启动状态 log 信息打印

如果某个数据包出现了丢失，则两方相应的动作如下：

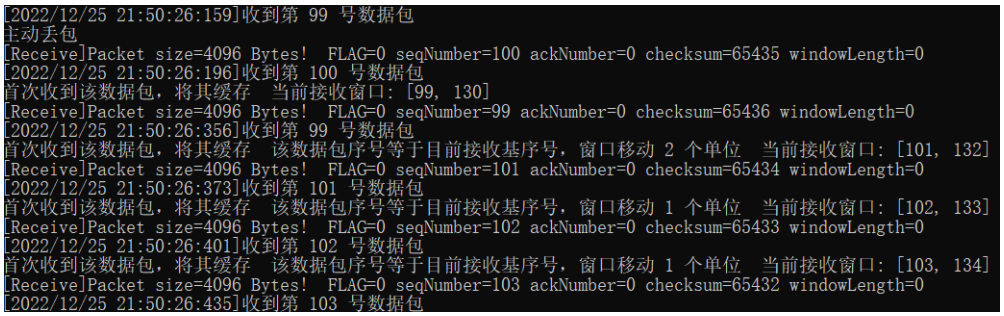

图 6: 某个数据包丢失后接收端动作

可以看到是第 99 号数据包出现了丢失。

```
发送第 100 号数据包  当前发送窗口: [99, 100]
[Send]Packet size=4096 Bytes!  FLAG=0 seqNumber=100 ackNumber=0 checksum=65435 windowLength=0
第 100 对应的timerID为 11991
[Receive]Packet size=0 Bytes!  FLAG=4 seqNumber=0 ackNumber=100 checksum=0 windowLength=0
====================== 拥塞避免阶段 ======================
Send Window Size: 2 sendbase: 99 ack: 100 cwnd: 2 ssthresh: 1 cwndFlag: 50
KillTimer 11991 ack=100 sendbase=99

第 99 号数据包对应的计时器超时，重新发送
发送第 99 号数据包  当前发送窗口: [99, 100]
[Send]Packet size=4096 Bytes!  FLAG=0 seqNumber=99 ackNumber=0 checksum=65436 windowLength=0

====================== 回到慢启动阶段 ======================
[Receive]Packet size=0 Bytes!  FLAG=4 seqNumber=0 ackNumber=99 checksum=0 windowLength=0
====================== 慢启动阶段 ======================
Send Window Size: 1 sendbase: 99 ack: 99 cwnd: 1 ssthresh: 1
********** 慢启动阶段接收到新的ACK, cwnd++ **********
KillTimer 11992 ack=99 sendbase=99
```

图 7: 某个数据包丢失后发送端动作

可以看到，发送端在对应的计时器超时之后进行重发，这里没有进行快速重传的原因是当前窗口大小只有 2，不可能收到 3 个重复的 ACK 来激活快速重传机制。并且，由于有数据包超时，发送端回到了慢启动状态。

在另外一次文件传输的过程中，出现了丢包进行快速重传：

```
[Receive]Packet size=4096 Bytes!  FLAG=0 seqNumber=98 ackNumber=0 checksum=65437 windowLength=0
[2022/12/25 22:25:38:552]收到第 98 号数据包
主动丢包
[Receive]Packet size=4096 Bytes!  FLAG=0 seqNumber=99 ackNumber=0 checksum=65436 windowLength=0
[2022/12/25 22:25:38:577]收到第 99 号数据包
首次收到该数据包，将其缓存  当前接收窗口: [98, 129]
[Receive]Packet size=4096 Bytes!  FLAG=0 seqNumber=100 ackNumber=0 checksum=65435 windowLength=0
[2022/12/25 22:25:38:607]收到第 100 号数据包
首次收到该数据包，将其缓存  当前接收窗口: [98, 129]
[Receive]Packet size=4096 Bytes!  FLAG=0 seqNumber=101 ackNumber=0 checksum=65434 windowLength=0
[2022/12/25 22:25:38:640]收到第 101 号数据包
首次收到该数据包，将其缓存  当前接收窗口: [98, 129]
[Receive]Packet size=4096 Bytes!  FLAG=0 seqNumber=102 ackNumber=0 checksum=65433 windowLength=0
[2022/12/25 22:25:38:665]收到第 102 号数据包
首次收到该数据包，将其缓存  当前接收窗口: [98, 129]
[Receive]Packet size=4096 Bytes!  FLAG=0 seqNumber=98 ackNumber=0 checksum=65437 windowLength=0
[2022/12/25 22:25:38:718]收到第 98 号数据包
首次收到该数据包，将其缓存  该数据包序号等于目前接收基序号，窗口移动 5 个单位  当前接收窗口: [103, 134]
[Receive]Packet size=4096 Bytes!  FLAG=0 seqNumber=103 ackNumber=0 checksum=65432 windowLength=0
[2022/12/25 22:25:38:739]收到第 103 号数据包
首次收到该数据包，将其缓存  该数据包序号等于目前接收基序号，窗口移动 1 个单位  当前接收窗口: [104, 135]
[Receive]Packet size=4096 Bytes!  FLAG=0 seqNumber=98 ackNumber=0 checksum=65437 windowLength=0
[2022/12/25 22:25:38:778]收到第 98 号数据包
当前接收窗口: [104, 135]  该数据包分组序列号在[recv_base - N, recv_base - 1]，发送ACK，但不进行其他操作
[Receive]Packet size=4096 Bytes!  FLAG=0 seqNumber=104 ackNumber=0 checksum=65431 windowLength=0
[2022/12/25 22:25:38:837]收到第 104 号数据包
```

图 8: 某个数据包丢失后接收端动作

```
====================== 拥塞避免阶段 ======================
Send Window Size: 5 sendbase: 98 ack: 91 cwnd: 5 ssthresh: 3 cwndFlag: 0
[Receive]Packet size=0 Bytes!  FLAG=4 seqNumber=0 ackNumber=99 checksum=0 windowLength=0
====================== 拥塞避免阶段 ======================
Send Window Size: 5 sendbase: 98 ack: 99 cwnd: 5 ssthresh: 3 cwndFlag: 0
KillTimer 9117 ack=99 sendbase=98

[Receive]Packet size=0 Bytes!  FLAG=4 seqNumber=0 ackNumber=100 checksum=0 windowLength=0
====================== 拥塞避免阶段 ======================
Send Window Size: 5 sendbase: 98 ack: 100 cwnd: 5 ssthresh: 3 cwndFlag: 1
KillTimer 9116 ack=100 sendbase=98

[Receive]Packet size=0 Bytes!  FLAG=4 seqNumber=0 ackNumber=101 checksum=0 windowLength=0
====================== 拥塞避免阶段 ======================
Send Window Size: 5 sendbase: 98 ack: 101 cwnd: 5 ssthresh: 3 cwndFlag: 2
********** 连续三个ack都不是sendBase, 开始提前重传 **********
[快速重传]发送第 98 号数据包  当前发送窗口: [98, 102]
[Send]Packet size=4096 Bytes!  FLAG=0 seqNumber=98 ackNumber=0 checksum=65437 windowLength=0

KillTimer 9115 ack=101 sendbase=98

[Receive]Packet size=0 Bytes!  FLAG=4 seqNumber=0 ackNumber=102 checksum=0 windowLength=0
====================== 快速恢复阶段 ======================
Send Window Size: 5 sendbase: 98 ack: 102 cwnd: 5 ssthresh: 2
KillTimer 9113 ack=102 sendbase=98
```

图 9: 某个数据包丢失后发送端进行快速重传

可以看到，在第 98 号包丢失之后，接收端连续 3 个 ACK 都不是 98，快速重传第 98 号数据包，并进入快速恢复阶段。

文件收发完成后，四次握手断开连接

图 10: 某个数据包丢失后发送端动作

至此，整个过程结束。

四、 总结

在本次实验之中，深入地了解了 TCP 的拥塞控制算法包括 Tahoe 算法、Reno 算法以及 New Reno 算法。同时对于多线程编程以及消息队列的相关知识进行了深入了解。

个人 GitHub 仓库链接：https://github.com/Skyyyy0920/Computer-Network

参考文献

1、03-计算机网络-第三章-2022.pdf

2、上机作业 3 讲解-2022.pdf