南開大學
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并编程实现 3-3

学号：1911437

姓名：刘浩通

年级：2019 级

专业：计算机科学与技术

2021 年 12 月 16 日

目录

一、 实验要求

在实验 3-2 的基础上，选择实现一种拥塞控制算法，也可以是改进的算法，完成给定测试文件的传输。

1. RENO 算法
2. 也可以自行设计简单协议进行实现
3. 给出实现的拥塞控制算法的原理说明
4. 有必要日志输出（窗口改变情况等）

本次实验是基于拥塞控制算法完成的。

二、 协议设计

(一) 协议特点

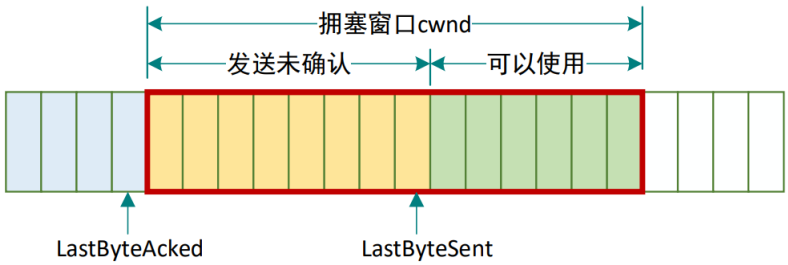
1. 发送方和接收方有握手过程，确认连接
2. 数据可以切包传输，保存序列号
3. 数据报有差错检测功能，正确接收返回确认
4. 数据报使用拥塞控制窗口传输机制
5. 数据报使用 ACK 累计确认机制，传输速率增长
6. 发送端有超时重传，解决数据包丢失的情况
7. 发送方和接收方有挥手过程，断开连接

(二) UDP 报文设计

1. 校验和 (checksum)：16 位，用于判断报文是否出错
2. 标志位 (Flag):8 位，ACK 标志字 (接收消息用于回复确认)、SYN 标志字段 (握手阶段使用)、SF 标志字段 (信息，标明未结束)、EF 标志字段 (传输的最后一条信息)、FIN 标志字段 (挥手阶段使用)。
3. 报文长度 (Length):16 位，整个报文的长度，数据字段长度就是减去前面定长的头部信息。
4. 序号 (seq)：16 位，本次实验使用的是 GBN 滑动窗口协议，给与足够长度的序号
5. data: 应用层的数据

(三) 拥塞控制控制窗口

采用基于窗口的方法，通过拥塞窗口的增大或减小控制发送速率

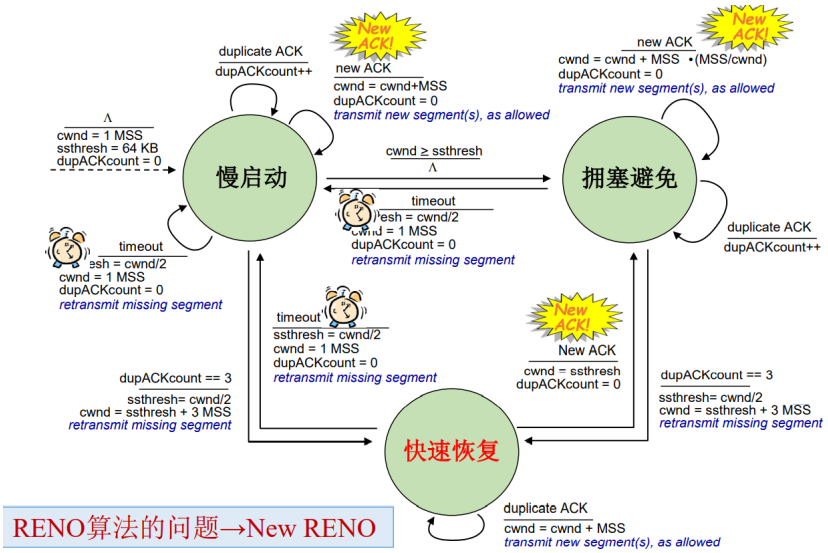


满足下面条件:

$LastByteSent - LastByteAcked\ CongestionWindow(cwnd)$

(四) RENO 拥塞控制算法

RENO 算法状态机如下:



1. 慢启动阶段:
   (a) 初始拥塞窗口: cwnd=1
   (b) 每收到一个 ACK, cwnd 增 1
   (c) 当收到三次重复 ACK 时, 进入快速恢复阶段
   (d) 当窗口大小大于阈值时, 进入拥塞避免状态
   (e) 当发生超时, 阈值 =cwnd/2, cwnd=1, 数据重传
2. 拥塞避免阶段:
   (a) 每收到一个 ACK, cwnd+=1/cwnd
   (b) 发生超时情况, 阈值减半, cwnd=1, 进入慢启动阶段
   (c) 连续收到三次重复 ACK, 阈值 =cwnd/2, cwnd= 阈值 +3, 进入快速恢复阶段, 数据重传
3. 快速恢复阶段:
   (a) 收到重复 ACK, 窗口加一
   (b) 发生超时, 阈值 =cwnd/2, cwmd=1, 进入慢启动状态, 数据重传
   (c) 收到新的 ACK, cwnd= 阈值, 进入拥塞避免状态

三、 中心代码实现

两次握手和两次挥手同 3-2 一致, 不再赘述

出现的重要变量:

```
int state = 0;          //  0:慢启动、1:拥塞避免、2:快速恢复。在初始情况下是慢启动阶段。
int dupACKcount = 0;    // 发生了重复ACK的次数。
double cwnd = 1;        // 窗口大小,初始为1
int ssthresh = 5;       // 阈值大小
bool* ACKs;             // 数组存储。来判定重复ACK情况的，顺便解决以下累计确认的问题。
```

1. cwnd 为拥塞窗口大小
2. state 为状态机所处的状态，1 为慢启动状态，2 为拥塞避免阶段，3 为快速恢复阶段
3. dupACKcount 重复 ACK 计数
4. ACKs 数组，用于判定重复 ACK

状态转移，窗口和阈值变化，以及窗口移动和超时重传体现在发送方的接收线程：

```
DWORD WINAPI RecvThread(LPVOID lparam) {
  cout << "线程启动!!!!!!" << endl;
  int lastAckSeq = 1;
  while (true) {
    while (!timesaver.empty()) //时间队列有东西
        {
            SetPriorityClass(recver, HIGH_PRIORITY_CLASS);
            double start = timesaver.front(); //获取第一个时间也是最早的时间
            char recv[8];                     // 接收数组
            int lentmp = sizeof(clientAddr);
            bool flag = false;
            while (recvfrom(client, recv, 8, 0, (sockaddr*)&serverAddr, &lentmp) == SOCKET_ERROR) {
              if (clock() - start > timeout) {
                  //现在出现了时间超时
                    timesaver = queue<double>(); //队列清空
                    ssthresh = cwnd / 2;         // 阈值减半
                    cwnd = 1;                    // 窗口变成1
                    dupACKcount = 0;             // 收到的重复ACK计数为0
                    state = 0;                   // 状态变成慢启动状态(0)
                    curSeq = SendBase;           // 重传
                    flag = true;
                    cout << "超时啦" << curSeq << " " << SendBase << endl;
                    cout << "慢启动阶段 " << "窗口大小" << cwnd << endl;
                    break;
                    }
```

```
26                  }
27                  //若超时，时间队列被清空，跳出这层循环，直至等待时间队列有东
                        西
28                  if (flag)
29                    break;
30
31                  double sampleRTT = clock() - start;
32                  // 没超时，收到了一个数据报
33                  SHORT seq = (USHORT)((unsigned char)recv[6] << 8) + (USHORT)(
                        unsigned char)recv[7];
34                  // seq是收到报文的序号
35                  if (checksum(recv, 8) == 0 && recv[2] == ACK) {  // 校验和通
                        过，并且是一个ACK回复
36                    if (seq == SendBase + 1) {
37                    // 收到了正确的报文，窗口向后面移动一下
38                    addSampleRTT(sampleRTT);
39                    timesaver.pop();
40                    lastAckSeq = seq;
41                    SendBase++;
42                    ACKs[seq - 2] = true;    // 来一波确认
43                    if (state == 0) {  // 慢启动状态
44                            cwnd++;
45                            dupACKcount = 0;
46                            cout << "窗口大小:" << cwnd << endl;
47                    }
48                    else if (state == 1) {  // 拥塞控制状态
49                           cwnd += 1 / cwnd;
50                           dupACKcount = 0;
51                           cout << "窗口大小:" << cwnd << endl;
52                    }
53                    else if (state == 2) { // 快速恢复状态
54                           cwnd = ssthresh;
55                           dupACKcount = 0;
56                           state = 1;
57                           cout << "拥塞避免阶段" << endl;
58                           cout << "窗口大小:" << cwnd << endl;
59                    }
60                    if (state == 0 && cwnd >= ssthresh) {
61                           // 如果cwnd大于等于阈值，状态转移，到拥塞控制状态
62                           state = 1;
63                           cout << "拥塞避免阶段" << endl;
64                    }
65                    cout << "收到了 " << seq  << "号报文的ACK" << endl;
66                    if (lastSeq == seq) {
67                           closeall = true;
68                           cout << "都收到了" << endl;
69                           cout << curSeq << endl;
70                           return 0;
```

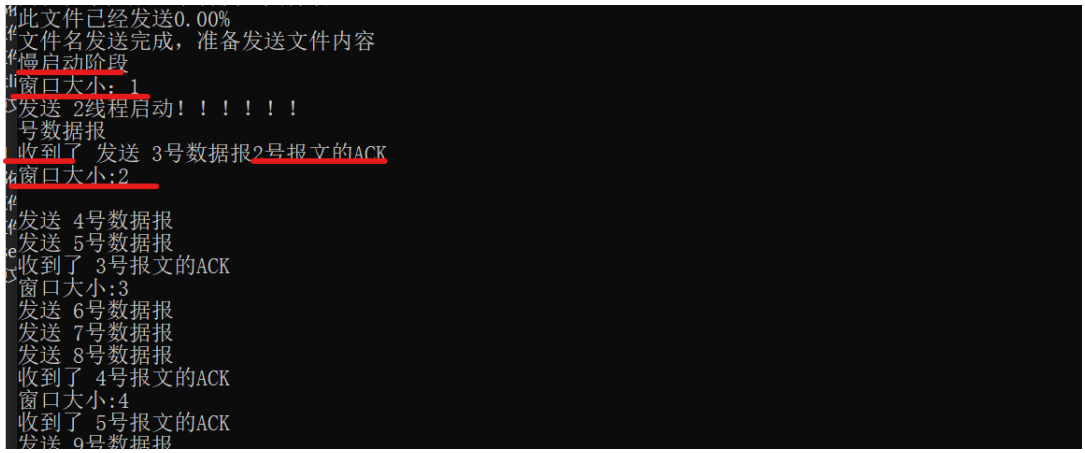
```
71                  }
72                }
73                else if (seq > SendBase + 1 && seq <= curSeq) {
74                  // 收到的编号，在累计确认情况下，收到的序号在窗口范围内
75                  // 累计确认的情况下
76                        addSampleRTT(sampleRTT);
77
78                        int sub = seq - SendBase;
79                        for (int i = 0; i < sub; i++) {
80                          timesaver.pop();              // 弹出时间
81                          ACKs[i + SendBase] = true;  // 写入记录ACK
82                        }
83                        SendBase = seq;
84                        cout << "收到了 " << seq  << "号报文的ACK" << endl;
85                        if (lastSeq == seq) {
86                          closeall = true;
87                          cout << "都收到了" << endl;
88                          cout << curSeq << endl;
89                          return 0;
90                        }
91
92
93                }
94                else {
95                  cout << "非正确回复，编号:  " << seq << endl;
96                  if (ACKs[seq - 2]) { // 考虑重复确认的情况
97                    if (state == 2) { //在快速恢复的情况下
98                          cwnd = cwnd + 1;
99                          cout << "窗口大小:" << cwnd << endl;
100                       }
101                       else {            // 其他两种情况下
102                         dupACKcount++;    // dupAckcount加一
103                       }
104                 }
105                 if (dupACKcount == 3 && state != 2) {    //出现三个重复ACK

106                       ssthresh = cwnd / 2;
107                       cwnd = ssthresh + 3.0;
108                       state = 2;              // 状态转移到快速恢复
109                       cout << "快速恢复阶段" << endl;
110                       cout << "窗口大小:" << cwnd << endl;
111                       curSeq = SendBase;     // 窗口重发
112                 }
113           }
114     }
115         SetPriorityClass(recver, NORMAL_PRIORITY_CLASS);
116       }
117 }
```

```
118 }
```

四、 实验结果

在路由器设置丢包率为 1% 的情况下进行测试

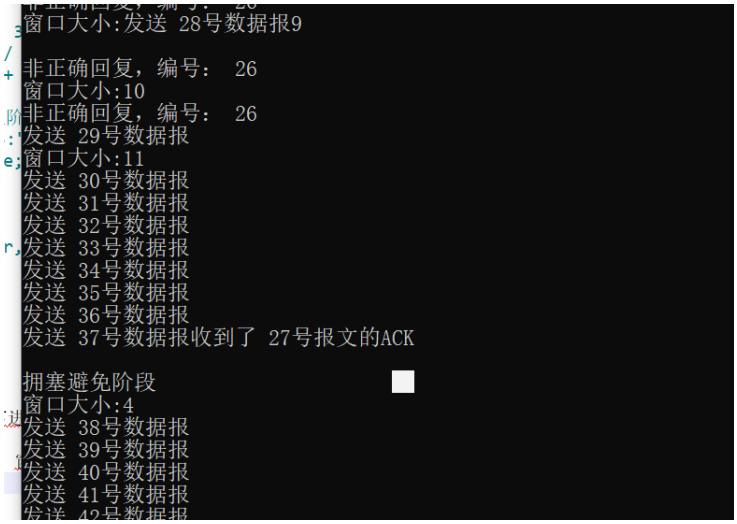
在刚启动的时候，处于慢启动状态，窗口大小为 1。等到收到一个 ACK 后，窗口大小加一。



继续发送和接收，最初设置的阈值为 5，所以窗口大小大于等于阈值时，进入拥塞避免状态。

等到出现重复 ACK 时，进入快速恢复阶段，并且在快速回复阶段收到重复 ACK，cwnd++:

在快速重传阶段收到正确 ACK，则会进入拥塞避免阶段：



若出现超时现象，则会进入慢启动状态