南开大学

Nankai University

南　开　大　学

网 络 空 间 安 全 学 院

计算机网络实验报告

---

lab3-3:UDP **的可靠数据传输协议拥塞控制实现**

姓名: 罗功成

学号: 1910487

年级 : 2019 级

专业 : 信息安全

指导教师 : 张建忠 徐敬东

2021 年 12 月 18 日

摘要

在实验 3-2 的基础上，选择实现 RENO 算法进行拥塞控制，完成给定测试文件的传输

关键字：RENO **算法，拥塞控制，**UDP

目录

一、 实验流程

(一) 协议设计

(1)操作标记位 tag: 相当于 TCP 协议中的 FIN 和 SYN, 根据 tag 值的不同, 确定进行不同的操作: 比如设置 tag 作为 0 时, 代表客户端发起建立连接请求, tag 为 88 时, 代表断开连接请求等。

(2) 序列号 seq: 用来标记数据包是第几个。

(3) 确认号 ack: 相当于 TCP 的 ack，确认数据包被正确接收

(4) 数据段长度 len: 标记可以容纳多少长度的数据。

(5) 数据段 data: 设计每个数据包大小为 1024 字节

(6) 窗口 window, 用来标记滑动窗。

参数补充:
int dupack = 0; 记录重复的 ack

float cwnd = 1; 记录拥塞窗口大小，初始设置为 1

int ssthresh = 32; 记录慢启动的阈值，初始设置为 32

(二) 实验核心代码

RENO 实现：慢启动，拥塞避免，快速重传

(1) 慢启动:
首先，发送数据的最小窗口大小取决于 cwnd 和接受缓冲区的窗口大小两者中的较小值，当比现在受到限制的窗口小时候，可以进行数据包的传输；此时需要判断 cwnd 和慢启动的阈值大小关系，若小于阈值，每次 cwnd 增 1，否则将需要进入到拥塞避免状态。

如果发送的数据包是成功接受的，那就把成功接受的 ack 总数 +1；若不成功，就接收到了重复的 ack，此时计数重复 ack 的 dupack 加 1。当接收到 3 个重复的 ack 时，启动快速重传，这时将 ssthresh 设为 cwnd 的减半，cwnd 设为当前 ssthresh+3。

慢启动

```
if ((index + seqnumber - curack) % seqnumber < minwindow(cwnd, WINDOWSIZE))
                {
                        cout << "收到第" << index << "号数据包的ack" << endl
                            << endl;
                        ack[index % WINDOWSIZE] = 3;
                        if (cwnd <= ssthresh)
                        {
                                cwnd++;
```

```
8
9                       }
10                      else
11                      {
12                              STATE = AVOID;
13                      }
14                      for (int j = curack; j != (index + 1) % seqnumber; j
                            = (++j) % seqnumber)
15                      {
16                              ack[j % WINDOWSIZE] = 1;
17                              ++totalack;
18                              curack = (curack + 1) % seqnumber;
19                      }
20              }
21              else if (index == curack - 1)
22              {
23                      dupack++;
24                      if (dupack == 3)//进入快速重传 状态跳转到拥塞避免
25                      {
26                              FASTRECOhandler();
27                              ssthresh = cwnd / 2;
28                              cwnd = ssthresh + 3;
29                              STATE = AVOID;
30                              dupack = 0;
31                      }
32              }
33              break;
```

（2）拥塞避免：进入拥塞避免阶段后，cwnd 每次只能递增 1/cwnd。该算法将一直保持到下次发生拥塞。

当再次看到接收到 3 个重复的 ack 时，启动快速重传，并将状态位设置为拥塞避免状态。

拥塞避免

```
1               cwnd = cwnd + 1 / cwnd;
2
3                       for (int j = curack; j != (index + 1) % seqnumber; j
                            = (++j) % seqnumber)
4                       {
5                               ack[j % WINDOWSIZE] = 1;
6                               ++totalack;
7                               curack = (curack + 1) % seqnumber;
8                       }
9               }
10              else if (index == curack - 1)
11              {
12                      dupack++;
```

```
                        if (dupack == 3)
                        {
                                FASTRECOhandler();
                                STATE = AVOID;
                                dupack = 0;
                        }
                }
                break;
```

（3）快速重传：

需要快速重传时，将需要发送的数据包作为一个新的数据包，对其进行初始化。此时数据包内容直接从缓存区里将上一个没能成功传送的数据包内容进行复制，然后再发出即可。

快速重传

```
void FASTRECOhandler()
{
        packet* pkt1 = new packet;
        pkt1->init_packet();
        for (int i = curack; i != curseq; i = (i++) % seqnumber)
        {
                memcpy(pkt1, &buffer[i % WINDOWSIZE], BUFFER);
                sendto(sockServer, (char*)pkt1, BUFFER, 0, (SOCKADDR*)&
                    addrClient, sizeof(SOCKADDR));
                cout << "重传第" << i << "号数据包" << endl;
        }
}
```

(三) 实验效果

先打开服务器端，再打开客户端，双方建立连接完成后，开始传输对应文件。

滑动窗实现效果，在对应文件传输成功过后，将会打印本次传输的延时和吞吐率信息。

图 1: 延时和吞吐率

可以看到，向指定的文件夹输出了对应的文件。

图 2: 对应位置输出

在路由程序中的输出效果:

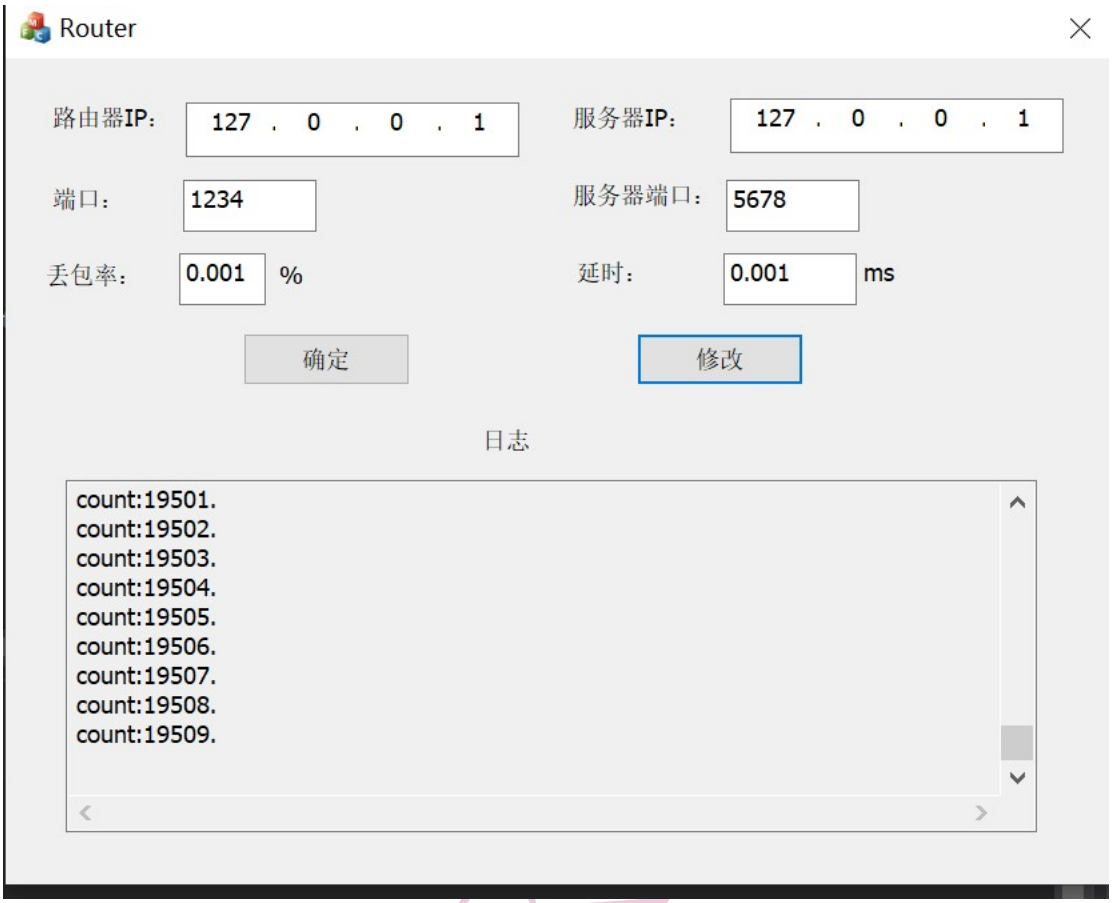

图 3: 路由程序

二、 总结和收获

1. 在 3-2 的基础上实现了拥塞控制算法。
2. 对慢启动和拥塞避免的执行过程和代码实现有了更深入的理解。