南開大學
Nankai University

南　　开　　大　　学

网络空间安全学院

计算机网络实验报告

编程作业 3 基于 UDP 服务设计可靠传输协议并编程实现3

于文明

年级：2020 级

专业：信息安全

指导教师：徐敬东，张建忠

2022 年 12 月 30 日

摘要

关键字：socket UDP rdt3.0

目录

一、 实验内容

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。

作业要求

- RENO 算法；
- 也可以自行设计协议或实现其他拥塞控制算法；
- 给出实现的拥塞控制算法的原理说明；
- 有必要日志输出（须显示窗口大小改变情况）。

二、 协议设计 [1]

本实验在 3_1 的基础上，采用 RENO 协议处理拥塞控制问题

(一) 报文格式

报文格式仍然延续了实验 3-1 的报文格式，具体如下:

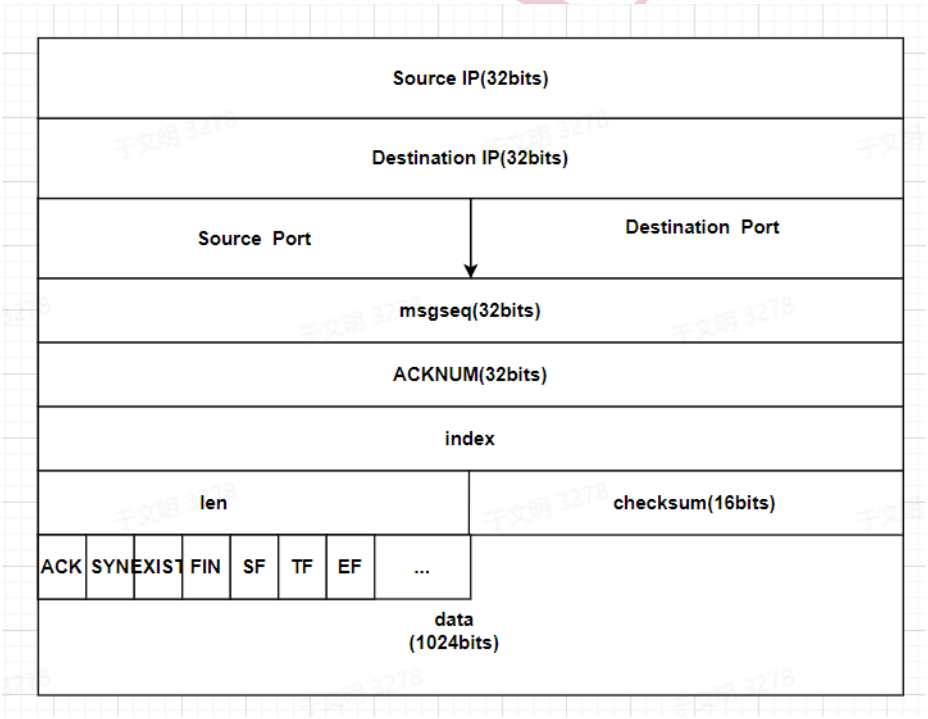

图 1: 报文格式

消息序列号字段 msgseq, 每次发送数据包，msgseq 增加 1

确认号字段 acknum, 它表示接收方期望收到发送方下一个报文段的第一个字节数据的编号。其值是接收计算机即将接收到的下一个序列号，也就是下一个接收到的字节的序列号加 1。在 GBN 的接收端，可以累计确认检查 acknum 确认消息发送成功

(二) 状态转换机 (FSM)

本实验在原有实验的基础上，在发送端增加了 RENO 算法的拥塞控制状态机

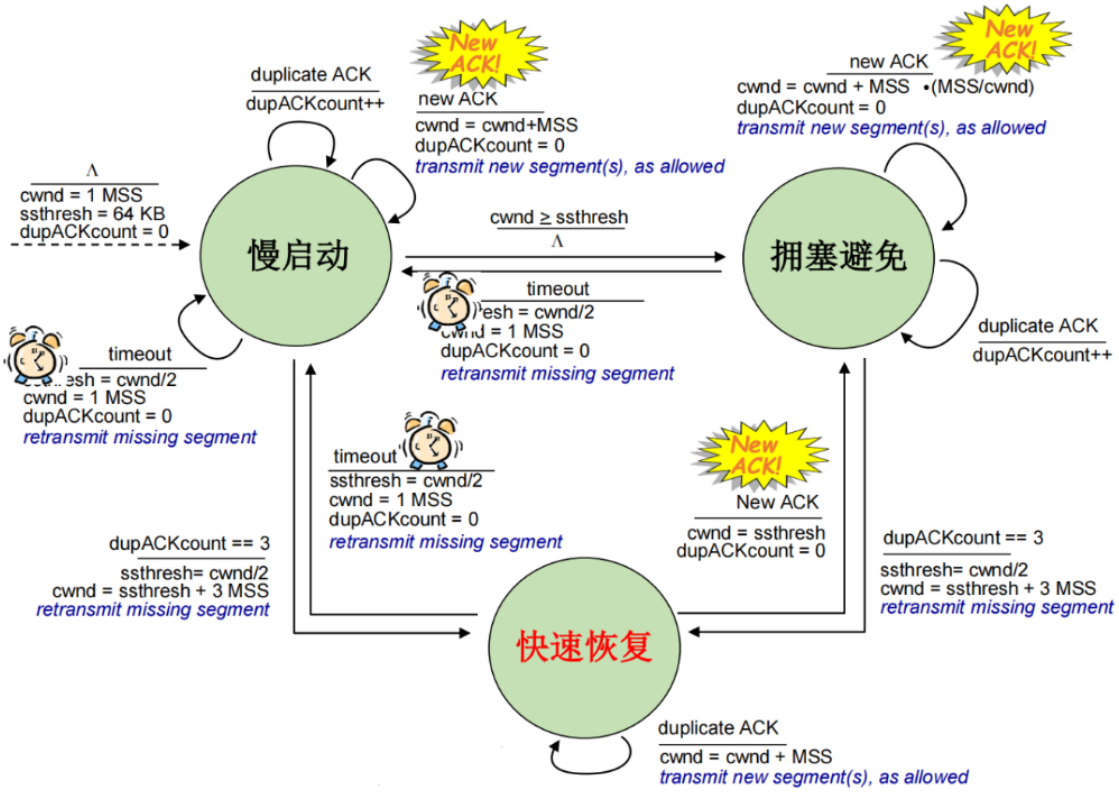

图 2: 拥塞控制状态机

发送端的拥塞控制状态机有三个状态：慢启动，拥塞避免，快速恢复

- 初始状态：慢启动
  - 窗口大小 cwnd 为 1，阈值 ssthresh 为 32
  - 如果收到新的 ACK, 窗口大小 +1, 效果呈现为单位时间内窗口呈指数增长
  - 如果收到冗余 ACK, 标记，如果连续收到 3 次冗余 ACK
    - 启动快速重传
    - 阈值设定为窗口大小的一半
    - 窗口大小设置为阈值 +3
    - 进入快速恢复阶段
  - 当窗口大小 >= 阈值时，进入拥塞避免阶段
  - 当超时没有收到新的 ACK 时
    - 阈值 = 窗口大小/2
    - 窗口大小设置为 1
    - 重新进入慢启动状态
- 拥塞避免

  - 每收到一个新的 ACK, 窗口大小增长为 1/窗口大小，单位时间内呈线性增长
  - 如果收到冗余 ACK, 标记，若连续收到 3 次冗余 ACK
    * 启动快速重传
    * 阈值设定为窗口大小的一半
    * 窗口大小设置为阈值 +3
    * 进入快速恢复阶段
  - 当窗口大小 >= 阈值时，进入拥塞避免阶段
  - 当超时没有收到新的 ACK 时
    * 阈值 = 窗口大小/2
    * 窗口大小设置为 1
    * 重新进入慢启动状态

- 快速恢复
  - 每收到一个冗余的 ACK, 窗口大小 +1, 直到窗口大小 >= 阈值，进入拥塞避免阶段

三、 核心代码

(一) 接收线程

recvhandler

```
DWORD WINAPI recvhandler(LPVOID lparam)
{
    printwindow();
    SOCKET* clientSock = (SOCKET*)lparam;
    //cout << "接收" << i << endl;
    clock_t finalovertime;
    int dupACKcount = 0;
    while (base < pktnum)
    {
        if (nextseqnum > base + cwnd || nextseqnum == base)//窗口已满（对方还
            没有发送ack）或者窗口为空
            Sleep(2);
        packet pak;
        recvpak(*clientSock,addrSer,pak);
        if (pak.get_flag(ACK)&&pak.checkchecksum(sizeof(pak)))//收到确认消息
            且序号正确
        {
            clockstart = clock();//重置计时器
            if (pak.get_ack() > base)//收到正确的消息序号
            {
                if (constatus == 0)
                {
                    cwnd += pak.acknum - base;//累积确认
                    if (cwnd >= ssthresh)
```

```
                        constatus = 1;//拥塞避免

                }
                else if (constatus == 1)
                {
                    cwnd += (pak.acknum- base) / cwnd;//线性增长
                }
                printwindow();
                base = pak.get_ack();
                overtime = 0;
                dupACKcount = 0;
            }
            else if (pak.get_ack() <= base)
            {
                dupACKcount++;
                printf("收到第%d个冗余包\n", dupACKcount);
                printwindow();
                if (constatus == 2) {//已经处在快速恢复阶段，增大窗口
                    cwnd += 1;
                    if (cwnd >= ssthresh) constatus = 1;//进入拥塞避免阶段
                }
                if (dupACKcount >= 3)//大于等于三个冗余ack
                {
                    constatus = 2; //快速重传
                    overtime = 1;
                    dupACKcount = 0;
                }
            }
        }
        clockend = clock();
    }
    printf("exitpoint2\n");
    return 1;
}
```

(二) 发送线程

sendhandler

```
DWORD WINAPI sendhandler(LPVOID lparam)//发线程
{
    int flag = 0;
    clock_t s = clock();
    SOCKET* clientSock = (SOCKET*)lparam;
    while (base < pktnum)
    {
        if (!overtime)
        {
```

```
10              int temp = base;
11              if (base + cwnd == nextseqnum)
12                  Sleep(40);
13              for (; nextseqnum < base + cwnd && nextseqnum < pktnum;
                    nextseqnum++)
14              {
15                  flag = 0;//正常发送
16
17                  if (!overtime)
18                  {
19                      sendpak(*clientSock, addrSer, paks[nextseqnum]);
20                      s = clock();
21                      //Sleep(20);
22                  }
23
24                  else {
25                      break;
26                  }
27                  sendpak(*clientSock, addrSer, paks[nextseqnum]);
28                  if (temp == base)
29                  {
30                      clock_t e = clock();
31                      if ((e - s) / CLOCKS_PER_SEC >= MAX_WAIT_TIME)//超时重发
32                      {
33                          overtime = 1;
34                          nextseqnum++;
35                          break;
36                      }
37                  }
38              }
39          }
40          if (overtime == 1)//重新发送
41          {
42              if (constatus == 1 || constatus == 0)
43              {
44                  ssthresh = cwnd / 2;
45                  cwnd = 1;
46                  constatus = 0;//重新进入慢启动阶段
47                  printwindow();
48                  printf("重新进入慢启动阶段\n");
49              }
50              else if (constatus == 2)
51              {//快速重传
52                  printf("快速重传\n");
53                  ssthresh = cwnd / 2;
54                  cwnd = ssthresh + 3;
55              }
56              if (flag)
```

```
57                  Sleep(10);//减少重传次数
58              for (int i = base; i <nextseqnum; i++)
59              {
60                  //重新发送
61                  if (overtime)
62                      sendpak(*clientSock, addrSer, paks[i]);
63                  else break;
64                  flag++;
65              }
66              overtime = 0;//重传标识归0
67              s = clock();//重置计时器
68          }
69          clock_t e = clock();
70          if ((e - s) / CLOCKS_PER_SEC >=MAX_WAIT_TIME)//超时重发
71              overtime = 1;
72      }
73      printf("exitpoint1\n");
74      return 1;
75  }
```

四、 实验结果

1. 慢启动阶段

刚开始处于慢启动状态，可以看到发送端每收到一个 ACK,cwnd+1, 窗口呈指数增长。

```
[2022-12-30 11:25:31 SEND] flag:36 SYN:0 ACK:0 msgseq:0 acknum:0
[stage:慢启动,cwnd:1,ssthresh:32]
[2022-12-30 11:25:31 SEND] flag:36 SYN:0 ACK:0 msgseq:0 acknum:0
[2022-12-30 11:25:31 RECV] flag:5 SYN:0 ACK:1 msgseq:2 acknum:1
[stage:慢启动,cwnd:2,ssthresh:32]
[2022-12-30 11:25:31 SEND] flag:36 SYN:0 ACK:0 msgseq:1 acknum:0
[2022-12-30 11:25:31 SEND] flag:36 SYN:0 ACK:0 msgseq:1 acknum:0
[2022-12-30 11:25:31 SEND] flag:36 SYN:0 ACK:0 msgseq:2 acknum:0
[2022-12-30 11:25:31 SEND] flag:36 SYN:0 ACK:0 msgseq:2 acknum:0
[2022-12-30 11:25:31 RECV] flag:5 SYN:0 ACK:1 msgseq:3 acknum:2
[stage:慢启动,cwnd:3,ssthresh:32]
[2022-12-30 11:25:31 RECV] flag:5 SYN:0 ACK:1 msgseq:4 acknum:3
[stage:慢启动,cwnd:4,ssthresh:32]
```

图 3: 慢启动阶段

2. 慢启动到拥塞避免

当 cwnd>=ssthresh 时，由慢启动进入拥塞避免阶段

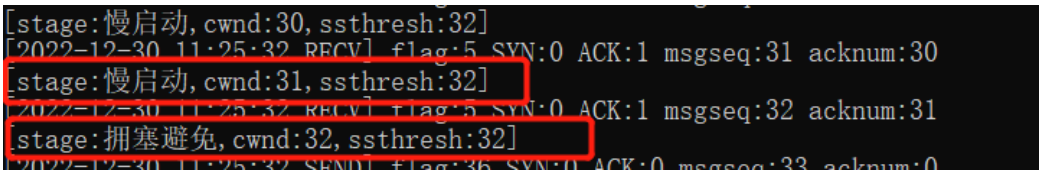

图 4: 慢启动到拥塞避免

3. 拥塞避免

在拥塞避免阶段，每收到一个 ACK,cwnd+=1/cwnd，窗口呈线性增长

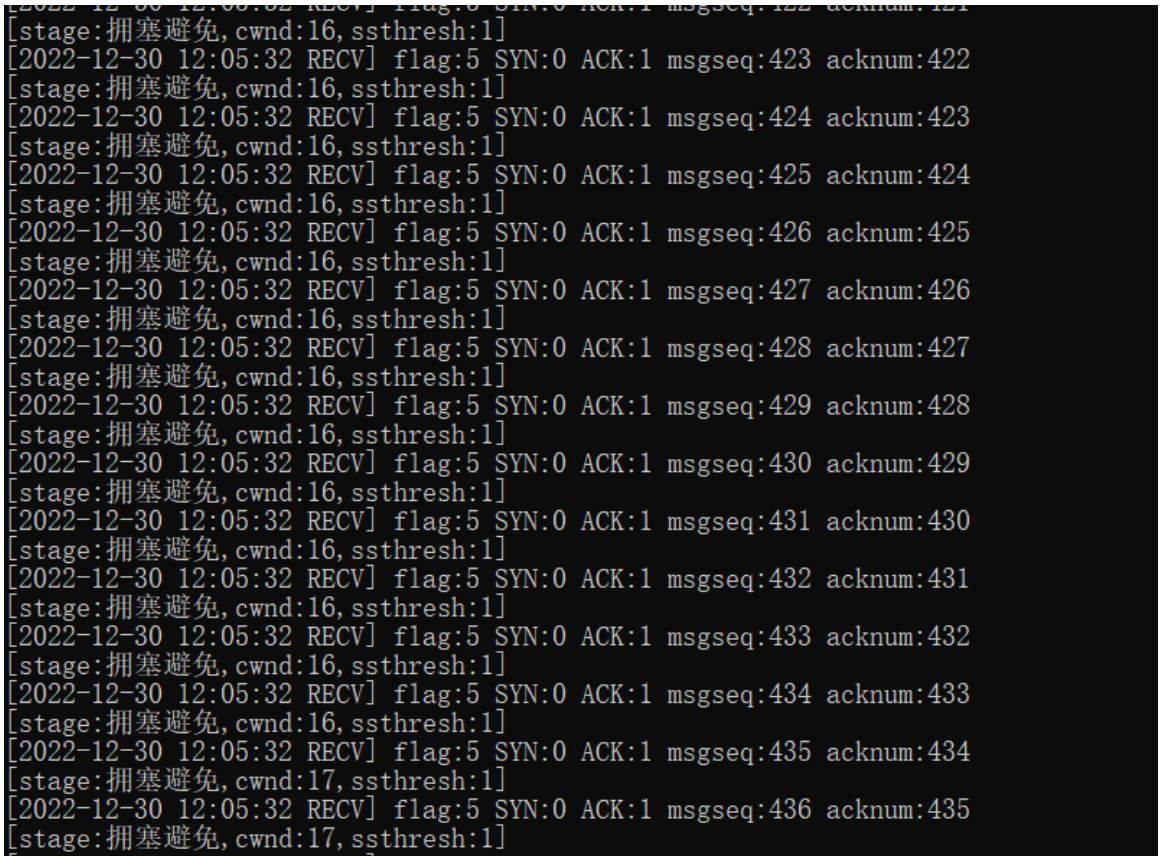

图 5: 拥塞避免阶段

4. 收到 3 个冗余 ACK

收到 3 个冗余 ACK, 将 ssthresh 设置为 cwnd 的一半，cwnd 设置为 ssthresh+3MSS, 并且可以看到立即重传了缺失的报文

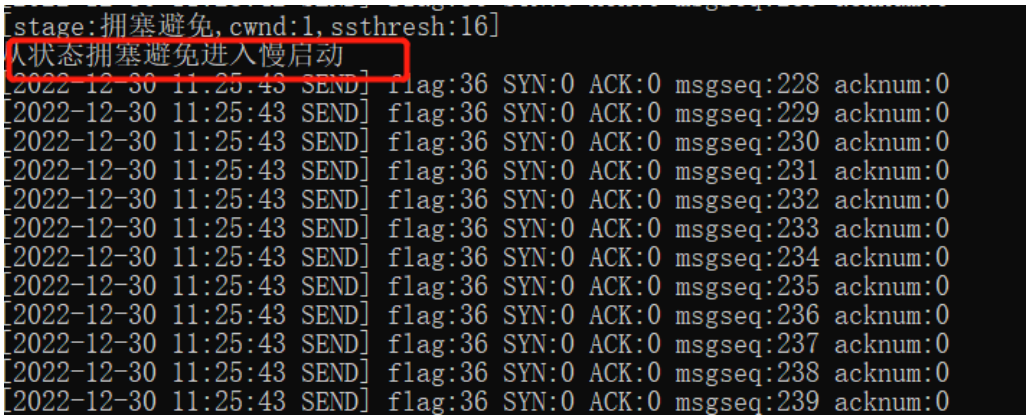

图 6: 收到 3 个冗余 ACK

5. 快速恢复

快速恢复阶段，收到冗余 ACK,cwnd+1

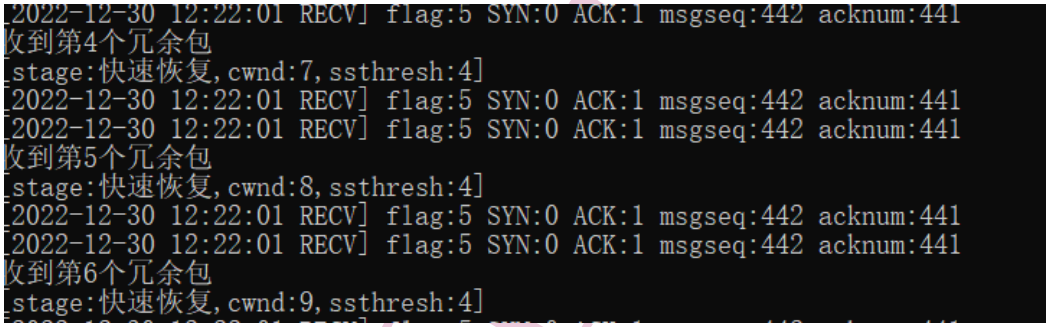

图 7: 收到 3 个冗余 ACK

6. 超时重传

无论处于什么状态，遇到超时，ssthresh 设置为 cwnd 的一半，cwnd 设置为 1，进入慢启动状态.

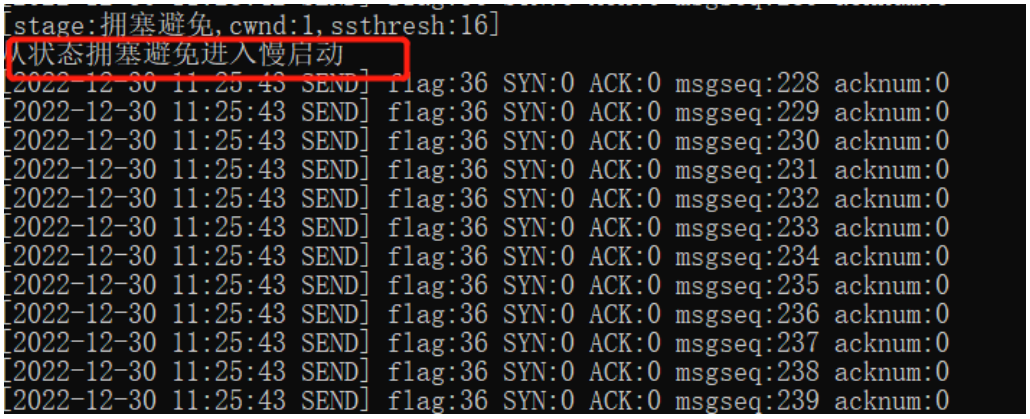

图 8: 慢启动阶段

五、 附录

本实验相关的资源 (源文件,测试文件,实验结果截图等) 都已上传至 github(https://github.com/Metetor/network)

参考文献

[1] 基思.W. 罗斯詹姆斯.E. 库罗斯. **计算机网络*:* 自顶向下方法**. 机械工业出版社, 2009.