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

---

lab3-1: **停等协议下** UDP **的可靠数据传输协议编程实现**

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 4 日

摘要

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、确认重传等。流量控制采用停等机制，完成给定测试文件的传输。

关键字：UDP，rdt2.0，**停等机制，可靠数据传输**

目录

一、 实验流程

(一) 协议设计

```
struct packet
{
    unsigned char tag;//连接建立、断开标识
    unsigned int seq;//序列号
    unsigned int ack;//确认号
    unsigned short len;//数据部分长度
    unsigned short checksum;//校验和

    char data[1024];//数据长度
```

图 1: 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。

由于路由程序最大传输 15000 字节，而测试文件都大于 15000B，因此需要对数据包进行分包操作:

(2) 序列号 seq: 用来标记分包是第几个。

(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

(二) 实验核心代码

(1) 客户端成功建立后，将状态标记为 ture, 将 tag=0 请求连接从客户端发出，当服务器端接收到 tag 为 0 的数据报后，调用计时器开始计时，并且将标记位设置为 ture，表示自己准备好建立连接了，此后将 tag 设置为 100, 向客户端发出已经准备好的消息；等到客户端准备好发来客户端准备好的 tag=200 的消息后，完成了建立连接，可以开始传输文件了。

根据停等协议的特性，只有上一个数据包发出成功并被成功接收后，下一个数据包才有可能发送出去，因此，只需要比较一下接收到的 ack 确认总数和传输前计算分包个数一致，就可以说明所有的数据包成功发送了。此时服务器将 tag 标记为 88，向客户端传送 tag 为 88 的数据包后打印传输时间和吞吐率后退出，客户端接收到 88 后退出。

建立连接与断开连接

```
服务器端:
        if (pkt->tag == 0)//客户端发来请求建立连接0
        {
                clock_t st = clock();//开始计时
```

```
                cout << "建立连接" << endl;
                int stage = 0;
                bool runFlag = true;
                while (runFlag)
                {
                switch (stage)
                {
                case 0:
                        pkt = connecthandler(100, totalpacket);//设置为100
                        sendto(sockServer, (char*)pkt, BUFFER, 0, (SOCKADDR*)
                            &addrClient, sizeof(SOCKADDR));

                        Sleep(100);
                        stage = 1;
                        break;
                        case 1:
                        ZeroMemory(pkt, sizeof(*pkt));
                        recvSize = recvfrom(sockServer, (char*)pkt, BUFFER,
                            0,((SOCKADDR*)&addrClient), &length);

                        if (pkt->tag == 200)
                        {
                        pkt->init_packet();
                        cout << "文件传输" << endl;
                        memcpy(pkt->data, filepath, strlen(filepath));
                    pkt->len = strlen(filepath);
                        sendto(sockServer, (char*)pkt, BUFFER, 0, (SOCKADDR*)
                            &addrClient, sizeof(SOCKADDR));
                        stage = 2;
                                                }
                        break;

        断开连接
        case 2:

        if (totalack == totalpacket)//数据包传输完毕
                {
                        pkt->init_packet();
                        pkt->tag = 88;
                    cout << "数据传输成功!" << endl;
                        cout << "传输用时: " << (clock() - st)  /
                            CLOCKS_PER_SEC << "s" << endl;
                        cout << "平均吞吐率" << (8 * length1 / ((clock() - st
                            ))) << "bps" << endl;
sendto(sockServer, (char*)pkt, sizeof(*pkt), 0, (SOCKADDR*)&addrClient,
    sizeof(SOCKADDR));
                        runFlag = false;
                        exit(0);
```

```
                        break;
                                        }

        客户端:
        while (true)
        {

                pkt->init_packet();
                pkt->tag = 0;
                sendto(socketClient, (char*)pkt, BUFFER, 0, (SOCKADDR*)&
                    addrServer, sizeof(SOCKADDR));
                while (true)
                {
                        //等待 server 回复
                        switch (stage)
                        {
                        case 0://等待握手阶段
                recvfrom(socketClient, (char*)pkt, sizeof(*pkt), 0, (SOCKADDR
                    *)&addrServer, &len);
                        totalpacket = pkt->len;
                        cout << "准备建立连接, 总共有" << totalpacket << "个
                            数据包" << endl;
                        pkt->init_packet();
                        pkt = connecthandler(200);
sendto(socketClient, (char*)pkt, sizeof(*pkt), 0, (SOCKADDR*)&addrServer,
    sizeof(SOCKADDR));
                        stage = 1;
                        break;
                        case 2:
                                pkt->init_packet();
                                recvfrom(socketClient, (char*)pkt, BUFFER, 0,
                                    (SOCKADDR*)&addrServer, &len);
                                if (pkt->tag == 88)
                                {
                                        cout << "文件传输成功";
                                        goto success;

                                }}}
```

(2) 计算校验和: 将数据段 16 比特字的和进行反码计算, 累加后的 sum 右移 16 位, 完成高 16 位变成低 16 位, 并对其取反; 如果 checksum 是 0, 计算的 sum 取反后也是 0, 证明没有损坏, 否则将需要重传。

计算校验和

```
unsigned short makesum(int count, char* buf)
{
        unsigned long sum;
        for (sum = 0; count > 0; count--)
```

```
        {
                sum += *buf++;
                sum = (sum >> 16) + (sum & 0xffff);
        }
        return ~sum;
}


// 判断包是否损坏
bool corrupt(packet* pkt)

        if (pkt->checksum == ~sum)
                return true;
        return false;
}
```

(3) rdt2.0 中需要重传主要只有两种情况：超时和校验和不正确。因而这里设定当某个数据包传输时延时达到了 1 秒以上，或者是校验和不是全 0，就把当前数据包设计为一个新的 packet，重新向客户端单独发一遍。

差错重传

```
//差错重传
void repeat()
{
        if ((clock() > 1)||(pkt->checksum!=0x0000))
        {
                packet* pkt1 = new packet;
                sendto(sockServer, (char*)pkt1, BUFFER, 0, (SOCKADDR*)&
                    addrClient, sizeof(SOCKADDR));
        }
}
```

(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。

根据停等机制的特性，只需要收到最后一个数据包的 ack，且收到 ack 总数和原来计算的总的数据包个数可以对应上，就可以说明文件传输成功。成功后，在下面打印用时和计算的吞吐率。

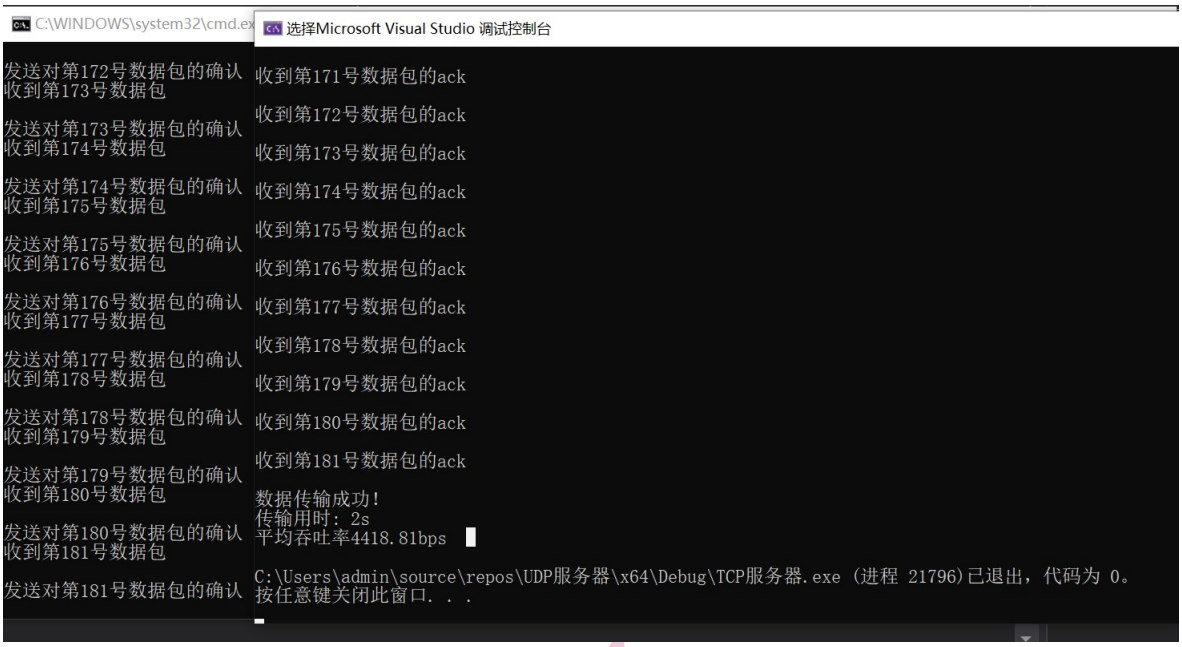

图 2: 延时和吞吐率

可以看到，向指定的文件夹输出了对应的文件。

图 3: 对应位置输出

二、 总结和收获

1. 熟悉和回顾了 TCP 和 UDP 之间的差异，通过对 TCP 机制的模拟，初步建立了 UDP 的可靠数据传输。

2. 对计算校验和，差错重传，建立连接的三次握手和断开连接四次挥手等相关内容进行复习和代码实现。