



南开大学  
Nankai University

南 开 大 学

计 算 机 学 院

计算机网络实验报告

---

### 实验三：基于 UDP 服务设计可靠传输协议

---

沙璇 1911562

年级：2019 级

专业：信息安全

指导教师：徐敬东

2022 年 9 月 19 日

## 摘要

基于 UDP 服务设计可靠传输协议并编程实现

关键字: UDP , C++

## 目录

一、 实验内容	1
(一) 内容概述 . . . . .	1
(二) 实验要求 . . . . .	1
二、 协议设计	1
(一) 累计确认 . . . . .	1
(二) 快速重传 . . . . .	1
(三) GBN 滑动窗口协议 . . . . .	2
(四) 日志打印 . . . . .	3
三、 编程赋现	3
(一) sever.cpp . . . . .	3
(二) client.cpp . . . . .	6
四、 实验结果	7
五、 实验收获	10

## 一、 实验内容

### (一) 内容概述

实验 3-2: 在实验 3-1 的基础上, 将停等机制改成基于滑动窗口的流量控制机制, 采用固定窗口大小, 支持累积确认, 完成给定测试文件的传输。

### (二) 实验要求

- (1) 实现单向传输。
- (2) 对于每一个任务要求给出详细的协议设计。
- (3) 给出实现的拥塞控制算法的原理说明。
- (4) 完成给定测试文件的传输, 显示传输时间和平均吞吐率
- (5) 性能测试指标: 吞吐率、时延, 给出图形结果并进行分析。
- (6) 完成详细的实验报告 (每个任务完成一份)。
- (7) 编写的程序应结构清晰, 具有较好的可读性。
- (8) 提交程序源码和实验报告。

## 二、 协议设计

由于停止等待协议效率太低, 为了提高基于 UDP 服务的传输效率, 本次实验在 3-1 的基础上实现以下机制:

- (1) 累积确认
- (2) 快速重传
- (2) GBN 滑动窗口协议

### (一) 累积确认

如果发送方发了包 1, 包 2, 包 3, 包 4; 接受方成功收到包 1, 包 2, 包 3。那么接受方可以发回一个确认包, 序号为 4, 那么发送方就知道包 1 到包 3 都发送接收成功, 必要时重发包 4。

一个确认包确认了累积到某一序号的所有包, 而不是对每个序号都发确认包。

累积确认无论丢失了多少接收方返回的确认, 对于发送方来说, 只要是在发送方超时时间之内成功接到了接收方返回的一串应答中的最后一个, 则包含了前面全部的确认信息。

累积确认的目的是为了避免一系列应答中的某个应答丢失造成的无必要的重传。

### (二) 快速重传

当发送方收到三个或以上的冗余 ACK (duplicate ACK), 就意识到之前发的包存在丢失, 于是快速重传。

如图1所示

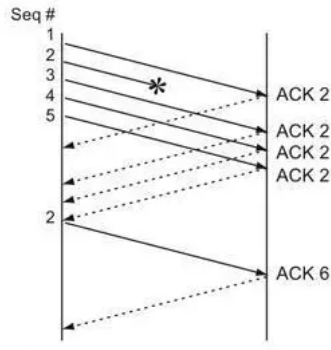


图 1: 快速重传

报文段 1 成功接收并被确认 ACK 2，接收端的期待序号为 2，当报文段 2 丢失，报文段 3 失序到来，与接收端的期望不匹配，接收端重复发送冗余 ACK 2。

### (三) GBN 滑动窗口协议

首先滑动窗口协议 (Sliding Window Protocol)，是一种用于网络数据传输时的流量控制协议。该协议允许发送方在停止并等待确认前发送多个数据分组。由于发送方不必每发一个分组就停下来等待确认。因此该协议可以加速数据的传输，提高网络吞吐量。

本次实验采取固定窗口大小。

在本协议中，将发送端的数据分为以下四类：

1. Sent and Acknowledged: 这些数据表示已经发送成功并已经被确认的数据，比如图中的前 31 个 bytes，这些数据其实的位置是在窗口之外了，因为窗口内顺序最低的被确认之后，要移除窗口，实际上是窗口进行合拢，同时打开接收新的带发送的数据。
2. Send But Not Yet Acknowledged: 这部分数据称为发送但没有被确认，数据被发送出去，没有收到接收端的 ACK，认为并没有完成发送，这个属于窗口内的数据。
3. Not Sent, Recipient Ready to Receive: 这部分是尽快发送的数据，这部分数据已经被加载到缓存中，也就是窗口中了，等待发送，其实这个窗口是完全有接收方告知的，接收方告知还是能够接受这些包，所以发送方需要尽快的发送这些包。
4. Not Sent, Recipient Not Ready to Receive: 这些数据属于未发送，同时接收端也不允许发送的，因为这些数据已经超出了发送端所接收的范围。

如图2所示

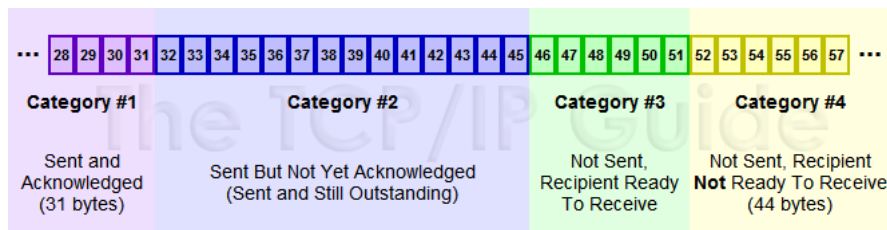


图 2: 数据分类

对于发送方来讲，窗口内的包括两部分，就是发送窗口（已经发送了，但是没有收到 ACK），可用窗口，接收端允许发送但是没有发送的那部分称为可用窗口。

1. Send Window : 20 个 bytes 这部分值是有接收方在三次握手的时候进行通告的，同时在接收过程中也不断的通告可以发送的窗口大小，来进行适应

2. Window Already Sent: 已经发送的数据, 但是并没有收到 ACK。

如图3所示

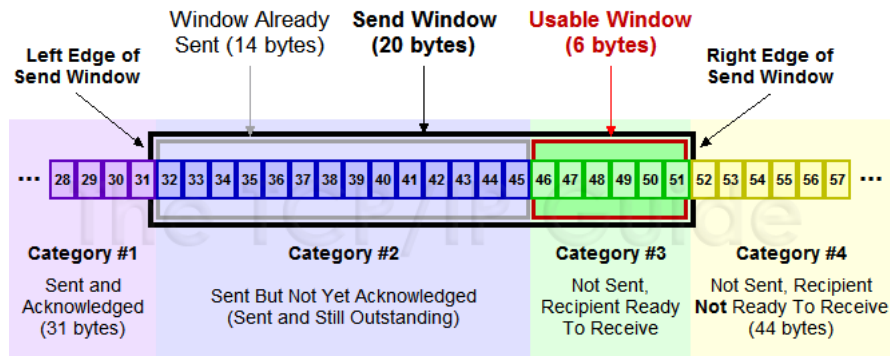


图 3: 滑动窗口

本次实验采取基于滑动窗口机制的 GBN 协议。

GBN 发送方要做的事:

1. 发送数据:

发送数据的时候, 发送方先检查窗口有没有满, 如果没有满, 则就产生一个帧并将其发送。

2. 接受 ACK:

GBN 协议中, 对  $n$  号帧的确认采用上文提到的累计确认的方式, 标明接收方已经收到  $n$  号帧和它之前的全部帧。

3. 处理超时事件:

协议的名字为后退  $N$  帧协议, 来源于出现丢失和时延过长帧时发送方的行为, 就像在停等协议中一样, 定时器将再次用于恢复数据帧或确认帧的丢失。如果出现超时, 则进行超时重传, 发送方重传所有已发送但未被确认的帧; 如果收到三次重复的 ack, 则进行快速重传, 同样重传所有已发送但未被确认的帧。

GBN 接收方要做的事:

1. 如果正确收号  $n$  号帧, 并且按序, 那么接收方为  $n$  帧发送一个 ACK, 并将该帧中的数据部分交付给上层

2. 其余情况都丢弃帧, 并为最近按序接收的帧重新发送 ACK。接收方无需缓存任何失序帧, 只要维护一个信息 `expectedseqnum` (下一个按序接收的帧序号)。

#### (四) 日志打印

测试文件传输时间、接收和发送窗口大小、平均吞吐率、建立连接、序列号、ack、校验和。

### 三、编程赋现

代码分为两部分, 包括 `client.cpp` 和 `sever.cpp`。具体代码见附件 `sever.cpp`、`client.cpp`。

#### (一) `sever.cpp`

由于基本功能在实验 3-1 已经赋现, 因此在本次实验中只对新增内容进行讲解。

滑动窗口

```

1 //滑动窗口
2 //curseq: 当前发送的序列号
3 //curack: 滑动窗口的最左端, 即当前等待被确认的数据包的最小序列号
4 //seqnumber: 序列号个数
5 while ((curseq + seqnumber - curack) % seqnumber < WINDOWSIZE && sendwindow > 0)
6 //只要窗口还没被用完, 就持续发送数据包
7 {
8     ZeroMemory(buffer[curseq % WINDOWSIZE], BUFFER); //内存清空
9     pkt->init_packet(); //初始化pkt包
10    if (length1 >= 1024) //大于1024字节则为每1024字节构建一个数据包
11    {
12        is.read(pkt->data, 1024); //读取1024字节
13        make_pkt(pkt, curseq, 1024); //创建pkt
14        length1 -= 1024; //文件长度—
15    }
16    else
17    { //不足1024部分也作为一个pkt发送
18        is.read(pkt->data, length1);
19        make_pkt(pkt, curseq, length1);
20    }
21    memcpy(buffer[curseq % WINDOWSIZE], pkt, BUFFER);
22    //从pkt开始拷贝BUFFER大小到buffer缓冲区中
23    sendto(sockServer, (char*)pkt, BUFFER, 0, (SOCKADDR*)&addrClient,
24           sizeof(SOCKADDR)); //socket发送
25    cout << "发送了序列号为 " << curseq << " 的数据包" << endl << endl;
26    //输出校验和部分
27    char buf1[128]; //string->char数组
28    itoa(pkt->checksum, buf1, 2); //char数组->2进制
29    string cks(buf1); //数组->字符串
30    cout << "校验和: " << cks << endl; //以二进制的形式输出校验和
31    ack[curseq % WINDOWSIZE] = 2; //已发送待确认
32    ++curseq; //发送序列号++
33    curseq %= seqnumber;
34    }
35    //等待接收确认ack
36    pkt->init_packet(); //pkt初始化
37    recvSize = recvfrom(sockServer, (char*)pkt, BUFFER, 0, ((SOCKADDR*)&
38    addrClient), &length);
39    if (recvSize < 0)
40    {
41        //recvSize < 0 代表没接收到发送的信息, 按照超时重传机制应该在到时间的时
42        候重新发送
43        waitcount++;
44        Sleep(200);
45        if (waitcount > 20) //往返时延
46        {
47            timeouthandler(); //超时重传
48            waitcount = 0;
49        }
50    }
51 }

```

```

46         }
47     }
48     else //接收到了ack之后 调用ackhandler来对ack数组中的元素进行累积确认
49     {
50         //ackhandler:滑动窗口机制
51
52         ackhandler(pkt->ack);
53         sendwindow = pkt->window;
54         cout<<"当前发送端窗口大小: "<<sendwindow<<endl;
55     }
56     break;
57 }

```

ackhandler 滑动窗口函数如下所示

ackhandler

```

1 void ackhandler(unsigned int a)
2 {
3     long long index = a; //序列号a
4     if ((index + seqnumber - curack) % seqnumber < WINDOWSIZE)
5     {
6         cout << "收到" << index << "号数据包的ack" << endl;
7         ack[index % WINDOWSIZE] = 3; //ack标记为已确认状态
8
9         //累积确认，一次把窗口左端直到当前序列号所有的包都确认
10        for (int j = curack; j != (index + 1) % seqnumber; j = (++j) %
11            seqnumber)
12        {
13            ack[j % WINDOWSIZE] = 1; //将滑动窗口里的包置为初态
14            ++totalack; //正确确认的数据包个数++
15            curack = (curack + 1) % seqnumber; //窗口左端右移，一直到确认
16            的序号处
17        }
18    }
19    else if (index == curack - 1)
20    {
21        //收到的ack是当前等待被确认的数据包的最小序列号-1的话，就证明收到了冗
22        余ack
23        {
24            dupack++; //冗余ack++
25            //快速重传是指如果连续收到3个重复的确认报文(DUPACK)则认为该报文很可能
26            丢失了
27            //此时即使重传定时器没有超时，也重传
28            if (dupack == 3) //进入快速重传
29            {
30                fasthandler();
31                dupack = 0;
32            }
33        }
34    }
35 }

```

快速重传函数如下所示

#### fasthandler

```

1 void fasthandler()
2 {
3     packet* pkt1 = new packet;
4     pkt1->init_packet(); //pkt初始化
5     //把当前滑动窗口最左端的包（当前等待被确认的数据包的最小序列号）一直到当前
    //发送的序列号的包都重传一遍
6     //即把所有窗口里已发送，未确认的包重传一遍
7     for (int i = curack; i != curseq; i = (i++) % seqnumber)
8     {
9         memcpy(pkt1, &buffer[i % WINDOWSIZE], BUFFER); //buffer传入
10        sendto(sockServer, (char*)pkt1, BUFFER, 0, (SOCKADDR*)&
            addrClient, sizeof(SOCKADDR)); //socket发送
11        cout << "***重传第 " << i << " 号数据包***" << endl;
12    }
13 }

```

## (二) client.cpp

在本次实验中只对 client.cpp 的新增内容进行讲解。

#### client 初始化

```

1 //GBN
2 if (pkt->seq == waitseq && totalrecv < totalpacket && !corrupt(pkt))
3 {
4     cout << "***收到第" << pkt->seq << "号数据包***" << endl << endl;
5     recvwindow -= BUFFER; //接收后窗口大小-BUFFER
6     cout << "当前接收端窗口大小: " << recvwindow << endl;
7     out_result.write(pkt->data, pkt->len); //把pkt所指的内存写入len个字节
        到所指的文件内
8     out_result.flush(); //清空缓存区
9     recvwindow += BUFFER; //应用进程读取后，窗口大小+BUFFER
10    cout << "写入后，当前接收端窗口大小: " << recvwindow << endl;
11    make_mypkt(pkt, waitseq, recvwindow); //ack=current waitseq-1 ,
        recvwindow+=BUFFER
12    cout << "-----发送对第" << waitseq << "号数据包的确认-----"
        << endl;
13    sendto(socketClient, (char*)pkt, BUFFER, 0, (SOCKADDR*)&addrServer,
        sizeof(SOCKADDR));
14    waitseq++;
15    waitseq %= seqnumber;
16    totalrecv++;
17 }
18 else

```



```
19 {  
20     make_mypkt(pkt, waitseq - 1, recvwindow);  
21     cout << "-----不是期待的数据包，发送了一个重复ack-----" <<  
        waitseq - 1 << endl;  
22     cout<<"当前接收端窗口大小："<<recvwindow<<endl;  
23     sendto(socketClient, (char*)pkt, BUFFER, 0, (SOCKADDR*)&addrServer,  
        sizeof(SOCKADDR));  
24 }
```

## 四、 实验结果

测试文件 1.png，传输文件如图4所示


 1.jpg	2020/4/2 21:24	JPG 图片文件	1,814 KB
---	----------------	----------	----------

图 4: 传输文件

注意此时日期为 2020/4/2。传输开始前 sever 端显示如图5所示

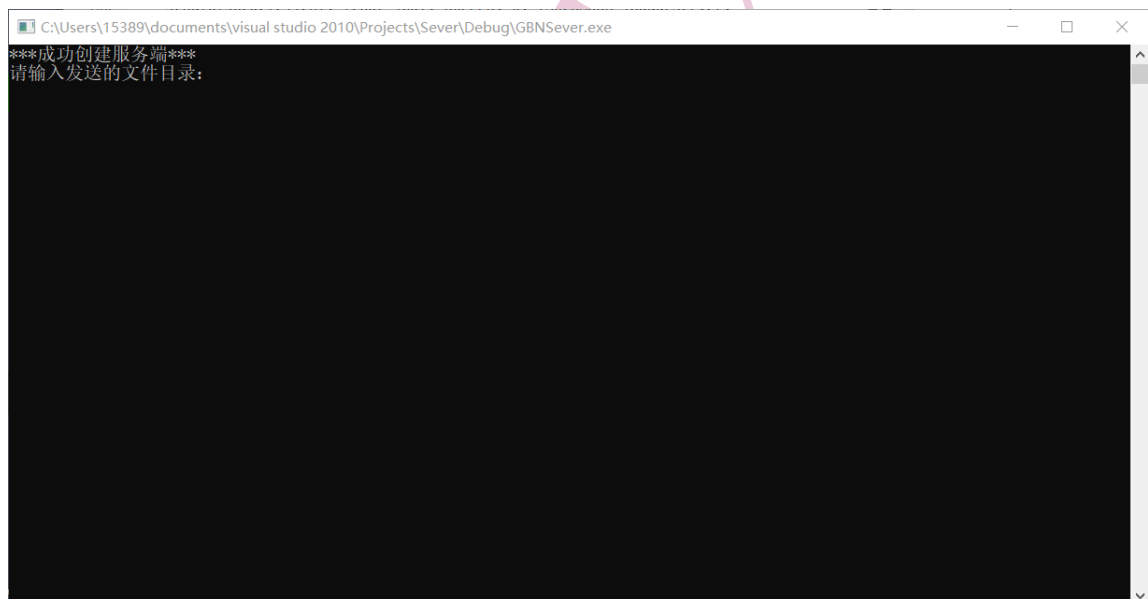


图 5: sever 开始前

传输开始前 client 端显示如图6所示

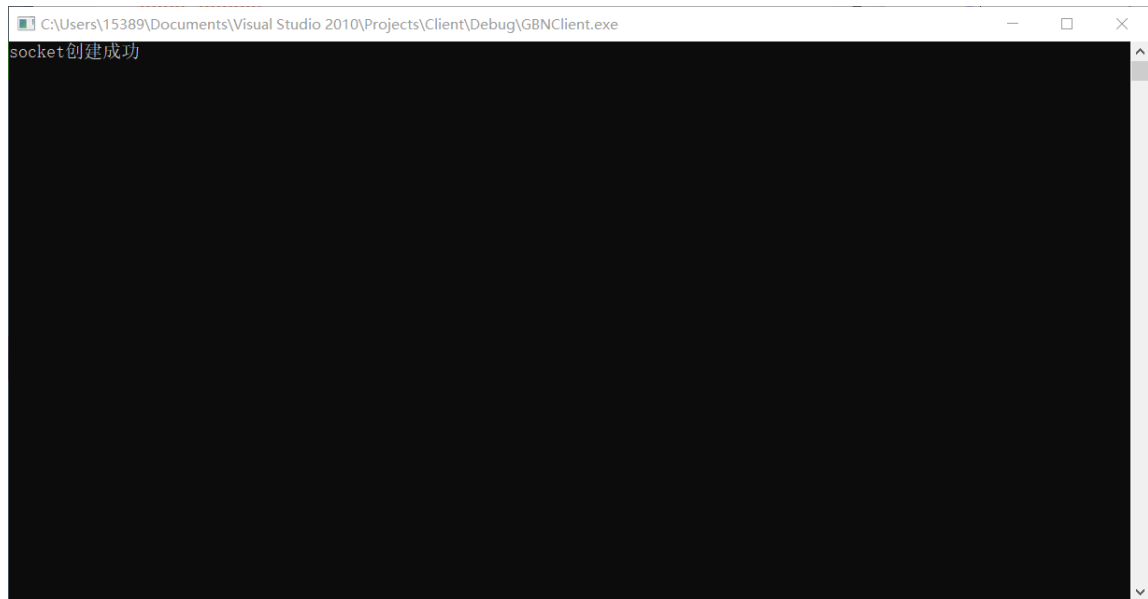


图 6: sever 开始前

传输过程中 sever 端显示如图7所示



图 7: sever 传输中

传输过程中 client 端显示如图8所示

```
C:\WINDOWS\system32\cmd.exe
socket创建成功
~~~准备建立连接~~~数据包个数: 1814
文件路径: C:\Users\15389\Desktop\任务1测试文件\11.jpg
校验和:10011001111011111
***收到第0号数据包***

当前接收窗口大小: 20368
写入后,当前接收窗口大小: 21440
-----发送对第0号数据包的确认-----
校验和:1111001111000001
***收到第1号数据包***

当前接收窗口大小: 20368
写入后,当前接收窗口大小: 21440
-----发送对第1号数据包的确认-----
校验和:10001111000101
***收到第2号数据包***

当前接收窗口大小: 20368
写入后,当前接收窗口大小: 21440
-----发送对第2号数据包的确认-----
校验和:1011000001000001
***收到第3号数据包***

当前接收窗口大小: 20368
写入后,当前接收窗口大小: 21440
-----发送对第3号数据包的确认-----
校验和:1111011101100010
***收到第4号数据包***
```

图 8: client 传输中

传输结束后 sever 端显示如图9所示

```
C:\WINDOWS\system32\cmd.exe
***重传第 1821 号数据包***
***重传第 1822 号数据包***
***重传第 1823 号数据包***
***重传第 1824 号数据包***
***重传第 1825 号数据包***
***重传第 1826 号数据包***
***重传第 1827 号数据包***
***重传第 1828 号数据包***
***重传第 1829 号数据包***
***重传第 1830 号数据包***
当前发送窗口大小: 21440
当前发送窗口大小: 21440
当前发送窗口大小: 21440
<<<<<<<<收到了1812号数据包的ack<<<<<<<<

当前发送窗口大小: 21440
发送了序列号为 1831 的数据包

校验和:1111111111111111
发送了序列号为 1832 的数据包

校验和:1111111111111111
<<<<<<<<收到了1813号数据包的ack<<<<<<<<

当前发送窗口大小: 21440
~~~~~
****文件传输完毕****
文件传输时间: 48411ms
吞吐率: 0.306931Mbps
请按任意键继续. . .
```

图 9: sever 传输后

传输结束后 client 端显示如图10所示

[illegible]

图 10: client 传输后

测试文件 1.png 传输后更名为 11.png，接收文件如图11所示



图 11: 接收文件

注意此时日期更新为 2021/12/1。

## 五、实验收获

本次实验深化了我对滑动窗口和 GBN 协议的理解，进一步熟悉了 socket 套接字的使用，对如何在用户空间实现面向连接的可靠数据传输、进行流量控制有了更深的理解，对于建立连接的三次握手、udp 校验和的差错检测、基于停等协议的超时重传机制、GBN 的实现都有了具体代码的实现。