哈爾濱工業大學
Harbin Institute of Technology

计算机网络

课程实验报告

实验名称	可靠数据传输协议-GBN 协议的设计与实现				
姓名		院系	计算学部		
班级		学号			
任课教师	刘亚维	指导教师	刘亚维		
实验地点	格物 207	实验时间	2023.10.28		
实验课表现	出勤、表现得分(10)		实验报告得分(40)	实验总分	
	操作结果得分(50)				
教师评语					

哈尔滨工业大学计算学部
FACULTY OF COMPUTING, HIT

实验目的：

理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。

实验内容：

1) 基于UDP 设计一个简单的GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。
2) 模拟引入数据包的丢失，验证所设计协议的有效性。
3) 改进所设计的GBN 协议，支持双向数据传输；
4）将所设计的GBN 协议改进为SR 协议。

实验过程：

1、 GBN协议数据分组格式：

Seq	Data

Seq是序列号（1-20），Data是传输的数据。

2、 确认分组格式：

ACK

直接返回ACK编号。

3、 协议两端流程图：

传输文件时有以下两个端口——传输端和接收端（双向传输中客户端和服务端都同时有传输端和接收端的操作）

（1） 传输端流程图：

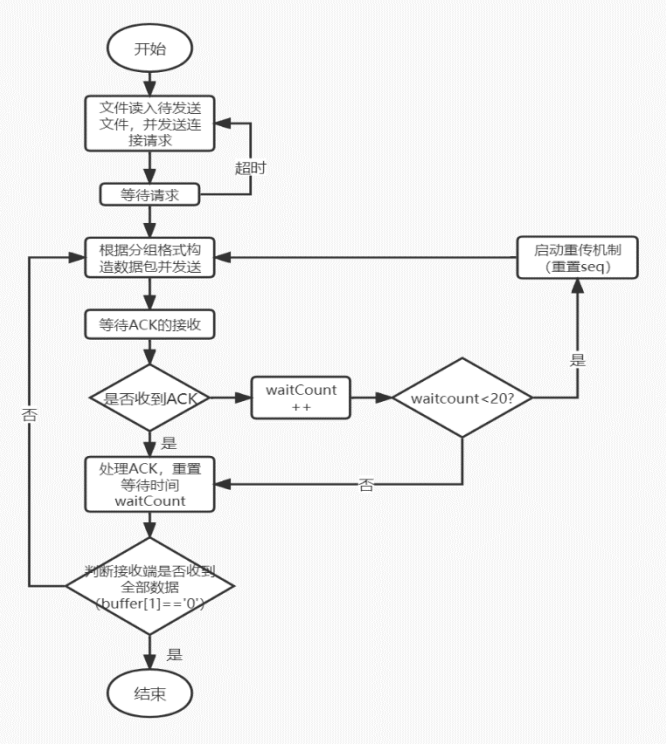

图 1 传输端流程图

（2） 接收端流程图：

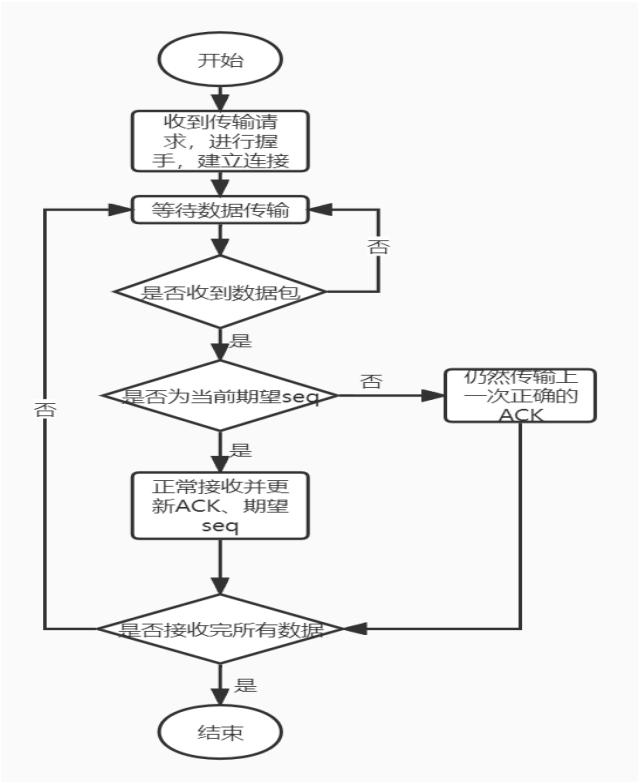

图 2 接收端流程图

4、 数据模拟丢失验证模拟方法：

采用函数lossInLossRatio进行模拟，先将概率按比例转换为一个整数P（0~100），然后使用rand()，若rand的数字%100小于P，则说明成功地被丢包，否则则不丢包正常进行。模拟丢包的时机主要在用户端发送/接收报文的时候，代码如下：

```
BOOL lossInLossRatio(float lossRatio) {
    int lossBound = (int)(lossRatio * 100);
    int r = rand() % 100;
    if (r < lossBound) {
        return TRUE;
    }
    return FALSE;
}
```

图 3 模拟丢包核心代码

5、 SR协议的改进思路：

SR协议与GBN协议不同，它不仅拥有发送窗口，同时也拥有接收窗口（缓存）；在超时之后不像GBN一样从最后一个成功接收（成功收到的ACK）的下一个位置进行传输，而是将发送窗口内把所有并未接收成功的数据进行传输。

对于接收端则是成功来什么数据（且未接受过）就接收，然后返回对应的ACK，而不是单纯表示<=ACK的数据全部成功接收，以此提高效率。

而在SR协议中，最重要的就是窗口内成功接收数据的记录（缓存），我在代码中使用ack[]数组来实现，ack[i]为FALSE表示i号数据成功接收并收到了ACK，否则说明还需要继续重传/接收。

接收端收完这一数据后，将窗口内全部的缓存数据按序写入文本中即可。

6、 主要函数的功能及实现：

（1） timeoutHandler函数

其主要作用是在GBN协议超时之后进行处理，其中需要处理当前成功接收的序列号（seq），其由上次成功收到的ack决定，以此来重新划定窗口，从ACK下一个数据开始传输；

```
void timeoutHandler() {
    printf("Timer out error.\n");
    int index;
    for (int i = 0; i < (curSeq - curAck + SEQ_SIZE) % SEQ_SIZE; ++i) {
        index = (i + curAck) % SEQ_SIZE;
        ack[index] = TRUE;
    }
    totalSeq -= ((curSeq - curAck + SEQ_SIZE) % SEQ_SIZE);
    curSeq = curAck;
}
```

图 4 timeoutHandler 函数

（2） ackHandler函数

其主要作用是在收到ack后对ack进行修改。若此时收到的ACK为x，则将当前成功收到的ACK更改为x，由于ACK前面所有数据都被成功接收，此时应该将占用状态重置（清零）。代码如下：

```
void ackHandler(char c) {
    unsigned char index = (unsigned char)c - 1; //序列号减一
    printf("Recv a ack of %d\n", index);
    if (curAck <= index) {
        for (int i = curAck; i <= index; ++i) {
            ack[i] = TRUE;
        }
        curAck = (index + 1) % SEQ_SIZE;
    }
    else {
        //ack 超过了最大值，回到了 curAck 的左边
        for (int i = curAck; i < SEQ_SIZE; ++i) {
            ack[i] = TRUE;
        }
        for (int i = 0; i <= index; ++i) {
            ack[i] = TRUE;
        }
        curAck = index + 1;
    }
}
```

图 5 ackHandler函数

实验结果：

1、 GBN协议的传输：

```
send a ack of 0
recv a packet with a seq of 1
The ack of 1 loss
recv a packet with a seq of 2
send a ack of 2
recv a packet with a seq of 3
send a ack of 3
recv a packet with a seq of 4
The ack of 4 loss
recv a packet with a seq of 5
send a ack of 5
recv a packet with a seq of 6
The ack of 6 loss
recv a packet with a seq of 7
send a ack of 7
The packet with a seq of 8 loss
recv a packet with a seq of 9
send a ack of 7
recv a packet with a seq of 10
The ack of 7 loss
recv a packet with a seq of 11
send a ack of 7
recv a packet with a seq of 8
send a ack of 8
recv a packet with a seq of 9
send a ack of 9
recv a packet with a seq of 10
send a ack of 10
```

图 6 GBN 协议的接收

上图可知，由于 seq=8 的数据丢包，导致返回 ack=7 不变，而 9-11 号的数据并不能被接收，等到超时的时候（重复收到 ACK），再次从 8 号 seq 的数据进行传输并接收。

2、 GBN协议的接收：

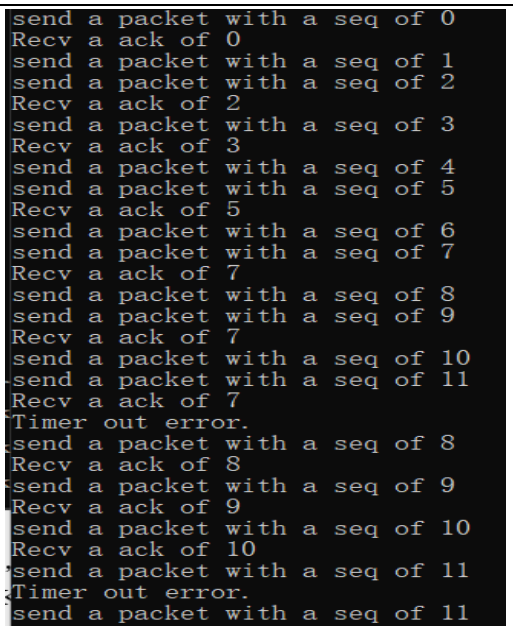

图 7 GBN 协议的传输

由图 7 可知，虽然之前 ack=6 返回的时候丢包了（图 6），然而在接到 ack=7 的时候，它仍是能顺利接收，因为 GBN 采用累计确认的方式，ack 代表的就是<=ack 的所有数据全部被成功接收，因此 6 虽然没有返回但实际上系统已判定成功接收。

3、传输文件比对（download/upload）：

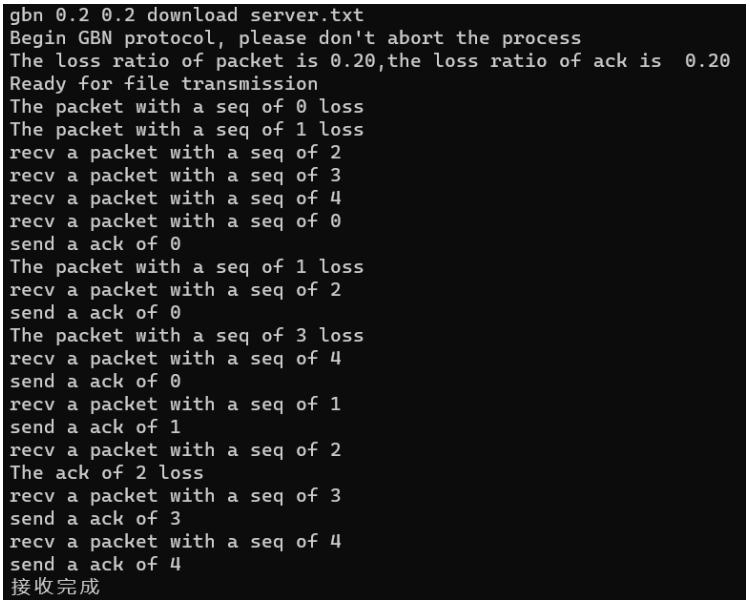

图 8 传输文件对比（download）

由图 8 可知，它在 0 丢包后，在超时之后只需重传 0 之后所有的数据。

```
gbn 0.2 0.2 upload client.txt
Begin GBN protocol, please don't abort the process
The loss ratio of packet is 0.20,the loss ratio of ack is  0.20
Begain to test GBN protocol,please don't abort the process
Shake hands stage
Begin a file transfer
File size is 5120B, each packet is 1024B and packet total num is  5
send a packet with a seq of 0
send a packet with a seq of 1
send a packet with a seq of 2
send a packet with a seq of 3
send a packet with a seq of 4
Timer out error.
send a packet with a seq of 0
send a packet with a seq of 1
Recv a ack of 1
send a packet with a seq of 2
send a packet with a seq of 3
send a packet with a seq of 4
Recv a ack of 2
Recv a ack of 3
传输完成
```

图 9 传输结果对比(upload)

4、SR协议的实现：

```
sr 0.2 0.2 upload client.txt
Begin SR protocol, please don't abort the process
The loss ratio of packet is 0.20,the loss ratio of ack is  0.20
Begain to test SR protocol,please don't abort the process
Shake hands stage
Begin a file transfer
File size is 5120B, each packet is 1024B and packet total num is  5
send a packet with a seq of 0
send a packet with a seq of 1
send a packet with a seq of 2
send a packet with a seq of 3
send a packet with a seq of 4
Recv a ack of 0
send a packet with a seq of 1
send a packet with a seq of 2
send a packet with a seq of 3
send a packet with a seq of 4
Recv a ack of 1
Recv a ack of 2
send a packet with a seq of 3
send a packet with a seq of 4
Recv a ack of 4
send a packet with a seq of 3
Recv a ack of 1
Recv a ack of 2
send a packet with a seq of 3
Recv a ack of 3
传输完成
```

图 10 客户端输出情况

由图 10 可知，它在 3 丢包后，对于后面的数据仍然正常接收，在超时之后只需重传单独的 3 号数据即可。

问题讨论：

1、 SR协议中窗口大小的设置可能会出现一些错误，原因在于它在$N_s+N_R > 2^k$时会导致无法区分S的某些数据是上一轮的数据还是这一轮的数据（seq号循环重复造成的错误），因此要确保$N_s+N_R \leqslant 2^k$。

2、 要注意等待超时的时间：过长会使文件传输过慢；过短会导致文件明明并未丢包但是由于超时等待时间过短而导致其被判定为了丢包。还有一种解决方法：将超时的判定改为接受相同ACK的次数（仅GBN协议）。

心得体会：
在本次实验中，我更深刻地体会了GBN协议和SR协议的内涵，对每一步骤以及它的核心都有了更好的理解（如GBN中的累计确认、SR中的发送和接收窗口、窗口长度等）； 同时我也进一步熟悉了socket编程中的更多细节，也很高兴能够成功实现基于UDP的可靠数据传输。