南開大學
Nankai University

计算机学院

计算机网络实验 **lab3-3**

基于滑动窗口的流量控制并编程实现

姓名：杨冰雪

学号：2110508

专业：计算机科学与技术

2023 年 12 月 14 日

目录

1 实验要求 **2**

2 滑动窗口 **2**

2.1 发送端 2

2.2 接收端 2

3 多线程编程 **2**

3.1 发送端 2

3.1.1 发送线程 2

3.1.2 接收线程 3

3.1.3 滑动线程 4

3.2 接收端 4

3.2.1 接收线程 4

3.2.2 滑动线程 6

4 选择确认 **6**

5 实验结果 **6**

5.1 传输结果 6

5.2 性能对比 8

6 实验总结 **9**

1 实验要求

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，发送窗口和接收窗口采用相同大小，支持选择确认，完成给定测试文件的传输。其主要实现工作如下：

- 增加滑动窗口功能
- 支持选择确认（SR）
- 采用多线程

2 滑动窗口

因为发送端和接收端的滑动窗口大小相同，所以宏定义了一个变量 WindowSize 来设置他们的窗口大小。

2.1 发送端

定义了一个变量 sendbase 来表示发送端的窗口下界，定义了一个变量 sendtop 来表示发送端的窗口上界，对于 0-sendbase 之间的数据表示已经传输完毕并且收到了对应的数据包 (设状态为 1)，而在滑动窗口内只包含两种状态——已发送数据包但等待确认包 (设状态为-1) 和未发送数据包 (设状态为 0)，对应 sendbase-messagenum-1 之间的数据表示未发送数据包 (设状态为 0)。只有滑动窗口内收到确认号为 sendbase 的确认号，才会向前滑动。

2.2 接收端

同样在接收端定义了一个变量 recvbase 表示接收端的窗口下界，定义了一个变量 recvtop 表示接收端的窗口上界，对于 0-recvbase 之间的数据表示已经接收完毕 (设状态为 1)，而在滑动窗口内的数据表示可以接收并且等待接收的数据 (设状态为 0)，而在 recvtop-messagenum-1 之间表示无法接收对应的数据。只有当滑动窗口内已经接收到 seq 为 recvbase 的数据包时，才会向前移动。

3 多线程编程

3.1 发送端

3.1.1 发送线程

发送线程主要用于发送端发送滑动窗口内未发生的数据包，在发送时进行模拟丢包，并为每个数据包设计超时重传计时器，当某个数据包在规定时间内没有收到确认数据包，那么就触发超时重传机制，如果在规定时间内接收到确认数据包，则在其执行完毕后被系统回收。

发送线程

```
int sendthread() {
    message msg;
    while (true) {
        for (int i = sendbase; i <= sendtop; i++) {
            if (state[i] == 0) {
```

```
                state[i] = -1;
                msg.seq = i;
                if (isLossPack(5)==false) {  //模拟丢包,设置的数字是丢包率
                    sendOneMsg(msg, i,false); //发送消息
                }
                thread resendThread(resendthread, i); //对每个帧设置重传计时器
                resendThread.detach();
                if (i == (messagenum - 1)) {
                    return 0; //当为最后一个包时, 退出子线程
                }
            }
        }
    }
}
```

超时重传线程

```
int resendthread(int seq) {
    clock_t begin = clock();
    while (true) {
        if (state[seq]==1) { //已接收到该确认包, 超时线程可以退出
            return 0;
        }
        if ((clock() - begin) > MAXTIME) { //超时重发超时的包
            message msg;
            sendOneMsg(msg, seq,true);
            begin = clock();
        }
    }
}
```

3.1.2 接收线程

接收线程用来接收接收端发来的确认包，让接收到的数据包是确认包时，就将对应确认号的状态设为 1，表示已发送数据包并且收到确认包，当确认号为 messagenun 时，表示时结束的确认包，则退出该线程。

接收线程

```
int recvthread() {
    while (true) {
        message msg;
        if (recvfrom(Client, (char*)&msg, BUFFER, 0, (SOCKADDR*)&serveraddr, &len) ==
            -1 || cksum((u_short*)&msg, sizeof(msg)) != 0 ) {
            msg=message(); //出错后丢失数据包
        }
        if (msg.ACK) {
            unique_lock<mutex> lock(coutMutex);
            cout << "【接收】收到ack为" << msg.ack << "的数据包" << endl;
```

```
            state[msg.ack] = 1;
            lock.unlock();
            if (msg.ack == messagenum) {
                cout << "【结束】接收到结束包" << endl;
                return 0; //当为最后一个包时退出子线程
            }
        }
    }
}
```

3.1.3 滑动线程

只有滑动窗口内收到确认号为 sendbase 的确认号，也即 sendbase 的状态变为了 1，滑动窗口才会向前滑动。

接收线程

```
int slidethread() {
    while (true) {
        if(state[sendbase] == 1) {
            if (sendbase == messagenum - 1) {
                return 0; //当为最后一个包时退出子线程
            }
            sendbase++;
            sendtop = ((messagenum - sendtop - 1) ? (sendtop + 1) : sendtop);
            unique_lock<mutex> lock(coutMutex);
            cout << "【滑动】滑动窗口前移一位，现在窗口底部是:" << sendbase <<
                ", 窗口顶部是:" << sendtop << endl;
            lock.unlock();
        }
    }
}
```

3.2 接收端

3.2.1 接收线程

接收端的接收线程用来接收发送端发来的数据包并发送确认包。如果接受的数据包 END 为 true，表示文件接收完毕，结束该线程；如果接收的数据包的序列号的状态为 1，或者序列号小于 recvbase，表示已经接收到该数据包，但可能因为丢包问题导致确认包丢失，则重新发送确认包；如果接收到的数据包序列号在滑动窗口内，则发送确认包，并将对应序列号的状态置为 1。

接收线程

```
int recvThread() {
    message recv;
    while (true) {
        if (recvfrom(Server, (char*)&recv, BUFFER, 0, (SOCKADDR*)&clientaddr, &len)
            == -1 || cksum((u_short*)&recv, sizeof(recv)) != 0) {
```

```
            recv = message();
        }
        else {
            int ack = recv.seq;
            message send;
            send.ACK = true;
            send.ack = ack;
            send.checksum = 0;
            send.checksum = cksum((u_short*)&send, sizeof(send));
            //发送结束
            if (recv.END) {
                unique_lock<mutex> lock(coutMutex);
                cout << "【结束】发送结束数据包" << endl;
                lock.unlock();
                if (sendto(Server, (char*)&send, BUFFER, 0, (SOCKADDR*)&clientaddr,
                    sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
                    cout << "发送错误了!!" << endl;
                }
                return 0;
            }
            //接收到数据包，但确认包可能丢失
            if (status[ack] == 1 || ack < recvbase) {
                unique_lock<mutex> lock(coutMutex);
                cout << "【接收】收到无效包裹" << endl;
                lock.unlock();
                if (sendto(Server, (char*)&send, BUFFER, 0, (SOCKADDR*)&clientaddr,
                    sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
                    cout << "发送错误了!!" << endl;
                }
            }
            //接收到数据包
            if (status[ack] == 0&&ack<=recvtop) {
                unique_lock<mutex> lock(coutMutex);
                cout << "【发送】发送了确认号为" << ack << "的数据包" << endl;
                lock.unlock();
                if (sendto(Server, (char*)&send, BUFFER, 0, (SOCKADDR*)&clientaddr,
                    sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
                    cout << "发送错误了!!" << endl;
                }
                if (ack == messagenum - 1) {
                    lastlen = recv.len;
                }
                memcpy(buffer[ack], recv.data, recv.len);
                status[ack] = 1;
            }
        }
    }
}
```

3.2.2 滑动线程

只有当滑动窗口内已经接收到 seq 为 recvbase 的数据包时，即 recvbase 的状态为 1 时，滑动窗口才会向前移动。

接收线程

```
int slideThread() {
    while (true) {
        if(status[recvbase] == 1) {
            if (recvbase == messagenum - 1) { //收到最后一个数据包
                return 0;
            }
            recvbase++;
            if (recvtop < messagenum - 1) {
                recvtop++;
            }
            unique_lock<mutex> lock(coutMutex);
            cout << "【滑动】滑动窗口移动,recvbase:" << recvbase << ",recvtop:" <<
                recvtop << endl;
            lock.unlock();
        }
    }
}
```

4 选择确认

当发送端接收到确认包时，只需要对收到的确认包对应的序号的状态设为 1，表示已发送数据包并且已接收到确认包，不需要像累积重传一样将确认号之前的状态都设为 1。

接收线程

```
if (msg.ACK) {
    unique_lock<mutex> lock(coutMutex);
    cout << "【接收】收到ack为" << msg.ack << "的数据包" << endl;
    state[msg.ack] = 1;
    lock.unlock();
    if (msg.ack == messagenum) {
        cout << "【结束】接收到结束包" << endl;
        return 0; //当为最后一个包时退出子线程
    }
}
```

5 实验结果

5.1 传输结果

设置丢包率为 5% 时，进行传输，其结果如下：

1. 1.jpg

   第一张照片的传输时间为 0.86s，吞吐率为 2117.81kbps。

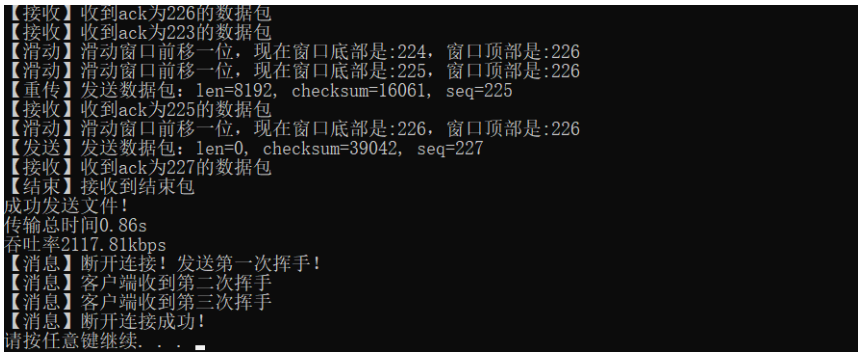


图 5.1: 1.jpg

2. 2.jpg

   第二张照片的传输时间为 2.946s，吞吐率为 1963.65kbps。

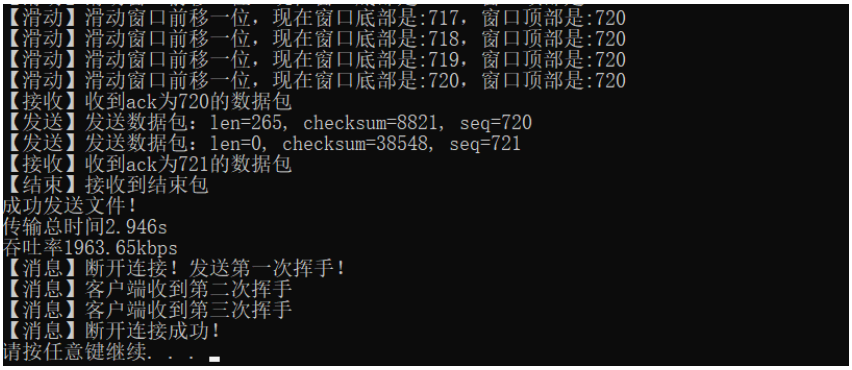


图 5.2: 2.jpg

3. 3.jpg

   第三张照片的传输时间为 5.679s，吞吐率为 2065.55kbps。

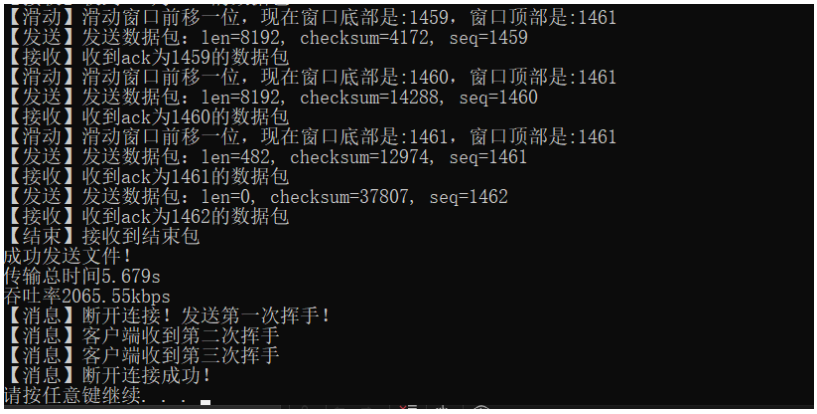


图 5.3: 3.jpg

4. helloworld.txt

文本文档的传输时间为 0.695s，吞吐率为 2343.54kbps。

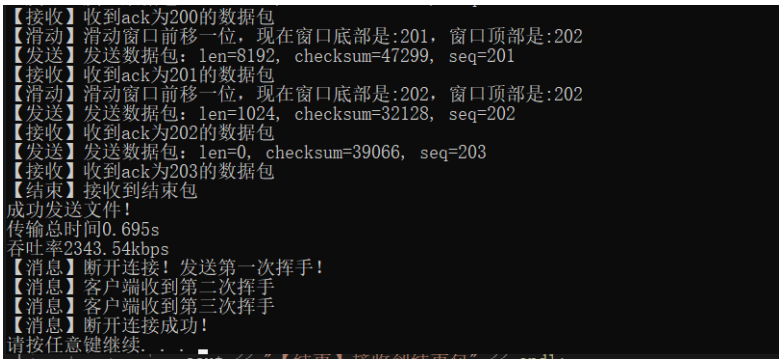
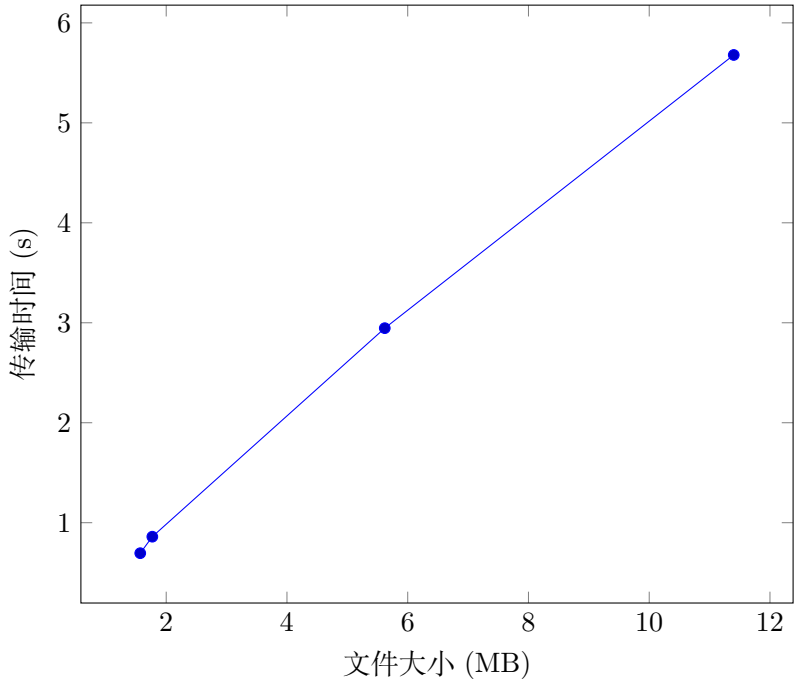


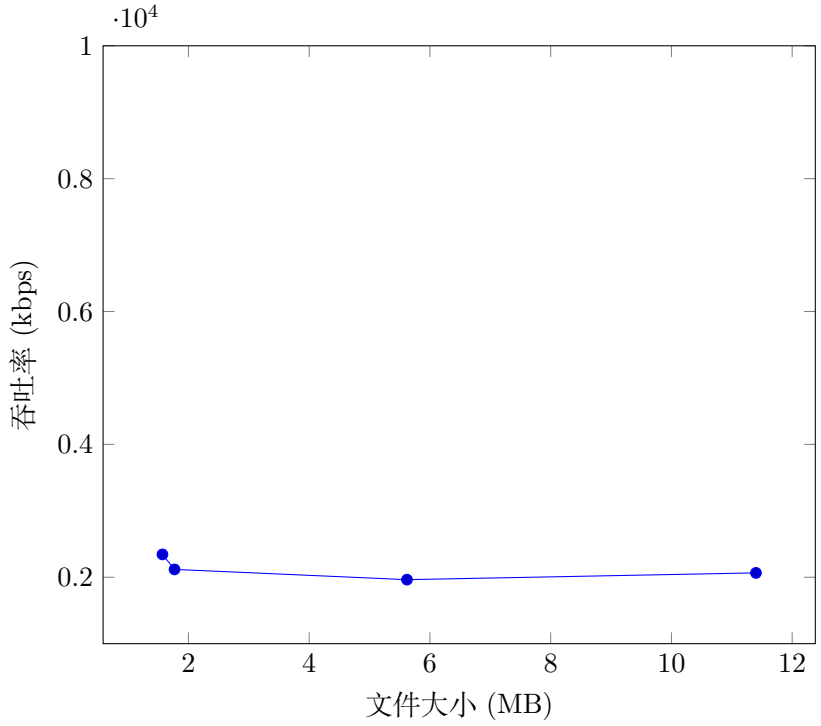
图 5.4: helloworld.txt

5. 接收结果

图 5.5: 接收结果

5.2 性能对比





6 实验总结

通过本次实验，让我对滑动窗口、选择确认等协议有了更深的理解。同时本次采用的协议相较于上一次的 GBN 在性能和传输时间上有了提升和优化，让我意识协议设计对性能上的影响是及其大的，所以我们要合理设计协议，充分考虑网络可能存在的各自问题和情况。